SESSION

EMBEDDED SYSTEMS AND NOVEL APPLICATION + EMBEDDED MICROCONTOLLERS + OPTIMIZATIONS

Chair(s)

TBA

FPGA Design and Implementation of Demodulator/Decoder Module for the EU eCall In-Vehicle System

Majeed Nader and John Liu
Electrical and Computer Engineering Dept., Wayne State University, Detroit, Michigan, USA

Abstract— *This paper presents the hardware design of a demodulator and decoder chip module for the In-Vehicle System (IVS) in the EU emergency call (eCall) system. FPGA technology is used to design and implement the demodulator/decoder module on a single chip. Xilinx ISE tools and Verilog HDL are used to develop the Register Transfer Level (RTL) of the module. The developed module is compiled, synthesized, and simulated. A Virtex-5 FPGA device is utilized to implement the developed system. The employed algorithm and hardware interfaces of the module is analyzed and discussed. A complete set of the input signals are used to simulate and verify the functionality of the module. The test and verification of the developed chip is done for different frequencies.*

Keywords: FPGA, EU eCall, in-vehicle system, demodulator, decoder.

1. Introduction

More than one million people and billions of dollars are lost in car accidents each year [1]. The automotive safety systems are always critical in the automotive industry and the automotive electronics industry. In the next few years, the electronic components in car manufacturing, including vehicular communication systems, will comprise of up to 40% of the total vehicle costs [2]. Utilizing telematics in vehicle safety system can provide significant benefits to automotive industry and road safety [2][3]. To reduce the fatality of car incidents, the European Commission has agreed to develop a telematics built-in emergency call (eCall) system that is to be operable in all EU countries by October 1, 2017 [4].

Responding to car accidents in the first moments known as the "golden hour" from the emergency centers to rescue the involved people can reduce the death rate by 11% and disability probability by 12% [1]. The EU eCall system provides vital role in shortening the arrival time of the emergency personnel in traffic accidents [5]. The system activates a voice call and provides an in-band data channel between an emergency center and the car involved in an accident [5]. The activation can be done manually through a specified button or automatically through the installed sensors in the vehicle [6].

The three crucial components of the system are the In-Vehicle System (IVS) that is to be designed and installed in all new cars in EU countries, Public Safety Answering Point (PSAP), and mobile telecommunication equipment. The IVS collects data about the accident and the vehicle to build the Minimum Set of Data (MSD)[5]. It also uses a cellular module to activate a voice call with the PSAP. There will be multiple stations of the PASP around EU countries. The mobile carriers should provide a dedicated emergency channel for the eCall system. Therefore, the system can use the channel anytime [5].

As soon as the IVS activates the eCall uplink channel, it monitors the downlink channel to receive the feedback messages for the PSAP. The feedback messages are Acknowledgment (ACK), Not Acknowledgment (NACK), and START messages. The feedback messages control the status of the IVS transmissions. The IVS uses a demodulation technique and a BCH decoder to demodulate and decode the downlink messages from the PSAP.

Developing the modem chip for the IVS is challenging because each module processes sophisticated algorithms. The Field Programmable Gate Array (FPGA) is a modern approach to design the digital modules of the IVS modem. In this paper, we present the design procedures of the demodulator and decoder modules to be used in the IVS and implement the developed module on an FPGA device. We employ Verilog HDL to describe the Register Transfer Level (RTL) of the modules. Utilizing a Xilinx synthesis tool, the developed modules are designed, simulated, and synthesized. We consider different clock frequencies to test and verify the modules.

2. The EU eCall System

The Third Generation Partnership Project (3GPP) emergency call system is a pan-European telematics system. The goal of the project is to reduce the fatalities in car accidents. It is expected that 3GPP eCall will greatly reduce the losses of lives in vehicle accidents. The system activates an emergency call and data transmission channel automatically via the installed sensors in a vehicle or manually by an occupant in the vehicle. The main parts of the eCall system are IVS, PSAP, and public mobile network carriers. While it is activated, the IVS collects the MSD containing information about the vehicle such as the VIN number and GPS coordinates. The information is sent to the most appropriate PSAP through a public mobile communication carrier [4][7].

In the idle state, the IVS receiver can monitor the downlink to receive a START message from the PSAP transmitter. As soon as the START message is detected, the IVS starts sending the MSD as uplink messages to the PSAP. After sending the MSD, the IVS expects feedback messages from the PSAP. If the IVS receives a NACK message from the PSAP, it transmits the MSD continuously until it receives an ACK message. The IVS also can reuse the downlink message to activate the connection between the IVS and the PSAP. In this case, the IVS uses the downlink message to push the PSAP to send the START message and monitors the received signals to detect the START message from the PSAP [5].
The IVS uses an emergency call (E112) through a cellular communication network to contact the nearest and most appropriate PSAP and provides the PSAP with the MSD. As soon as the PSAP receives the MSD from the IVS, the required emergency services will be sent to the location of the accident to aid the people who are involved in the accident and reorganize the traffic roads [6].

IVS is a built-in system in vehicles for emergency call purposes. The IVS collects the MSD about the vehicle such as the current geographical position of the car through a GPS or a similar system, the severity of the accident, and any other necessary information required for emergency assistance [6]. Figure 1 shows the different modules of the IVS.

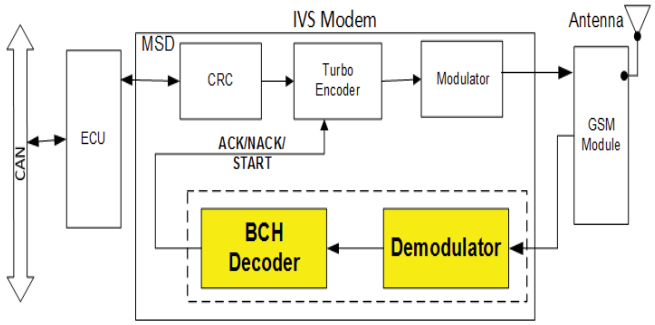


Fig. 1: The IVS modem block diagram.

The Cyclic Redundancy Check (CRC) module performs the CRC coding on the incoming data which is the MSD. The MSD data consists of 1,120 bits. The CRC module appends the input data with 28 bits of the CRC parity check for error detection during the data transmission. Another module is employed for encoding the MSD data which is the turbo encoder module. The turbo encoder performs a Forward Error Correction (FEC) coding technique. The code rate is 1/3. The input to the turbo encoder is the MSD data that is appended with the parity check bits. As the result of the turbo encoding code rate and trellis bits, the output of the turbo encoder is the encoded MSD data which consists of 3,456 bits [5].
The modulator groups the encoded bit streams into symbols and generates the corresponding waveform for each symbol. Each symbol is represented by three bits, so there are eight possible symbols. There are eight different uplink waveforms for the eight different possible symbols. In both IVS and PSAP modulators, the Bipolar Pulse Position Modulation (BPPM) is used as the modulation scheme [6]. There are three cyclic right shift versions of the positive or negative basic waveform to represent the different symbols [5].

3. The Demodulator/Decoder Module

The feedback messages control the IVS transmission status. the demodulator demodulates the downlink messages that are transmitted by the PSAP. There are only four feedback messages from the PSAP, which are: the START message to trigger the IVS to start transmitting the MSD to the PSAP, the NACK to signal IVS to continue to send the MSD, the ACK to acknowledge that the MSD is received by the PSAP, and one messages is reserved for future extra functionality which is the Reserved message. [6].
The modulator of the PSAP can modulate 16 BCH encoded messages. However, only four of them are utilized by the EU eCall system. Denote BCH encoded data bits as $\underline{b} = \{b_k\}$, $k = 0, 1, ..., 59$, and the demodulated symbols as $\underline{S} = \{s_l\}$, $l = 0, 1, ..., 14$. Table 1 shows the feedback messages in different formats. The feedback messages are represented by 60 bits [5][6]. The 60 bits of feedback messages are grouped

Table 1: The feedback messages.

Feedback	Binary	BCH (hexadecimal)
START	0000	A72 F298 41FA B376
NACK	0001	4C4 1FD6 6ED2 7179
ACK	0010	97A 8C41 FAB3 7693
Reserved	0011	DBE 9397 9461 07EA

into symbols by the PASP modulator, and each symbol represents four bits. So each message is represented by 15 symbols as they are represented by 15 hexadecimal digits in Table 1. The PSAP modulator modulates the symbols into downlink waveforms. Denote the downlink waveform bits as $\underline{e} = \{e_l\}$, $l = 0, 1, ..., 511$, the downlink waveform as $S_D(k), k = 0, 1, ..., 31$. The waveform is shown in Figure 2.

The downlink waveforms are represented by 32 samples and each sample is represented by 16 signed bits. The sampling rate is 8KHz, the modulation frame is 4 ms, and the modulation rate is 1000 bits/s [5].

The downlink waveforms (S_D) are mapped based on a basic waveform, which is [5]:
$S_D(k) = (40, -200, 560, -991, -1400, 7636, 15000,$
$7636, -1400, -991, 560, -200, 40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0, 0)$.
Since each symbol represents 4 bits, there are 16 possible symbols. Cyclic shift technique is employed to modulate all the possible symbols in the PSAP modulator. The mapping

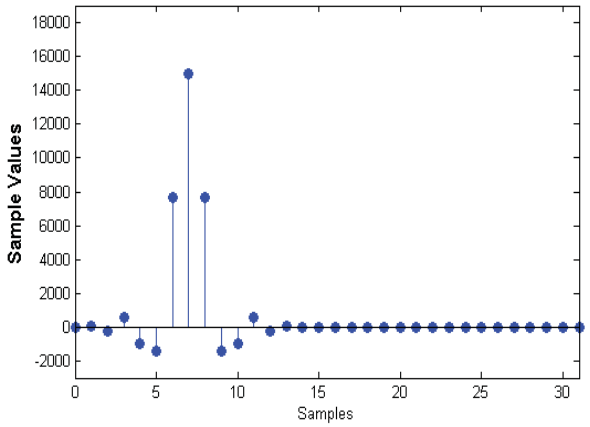


Fig. 2: The basic downlink waveform.

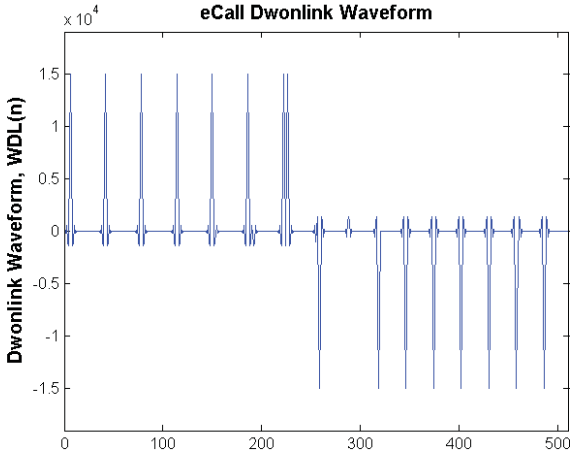


Fig. 3: The simulated 16 possible downlink waveforms.

technique between the possible symbols and downlink waveforms are illustrated in Table 2. The 16 downlink waveforms

Table 2: The symbols and the corresponding downlink waveforms.

S_i	$S_D(k), k = 0, 1, , 31$
0000	$S_D(k)$
0001	$S_D(k-4)$
0010	$S_D(k-8)$
0011	$S_D(k-12)$
0100	$S_D(k-16)$
0101	$S_D(k-20)$
0110	$S_D(k-24)$
0111	$S_D(k-28)$
1000	$-S_D(k-28)$
1001	$-S_D(k-24)$
1010	$-S_D(k-20)$
1011	$-S_D(k-16)$
1100	$-S_D(k-12)$
1101	$-S_D(k-8)$
1110	$-S_D(k-4)$
1111	$-S_D(k)$

are simulated in Figure 3.

3.1 The Design Algorithm

The flowchart of the employed algorithm is illustrated in Figure 4. The demodulator expects 16 different downlink waveforms, and after receiving and demodulating them, it generates the corresponding four bits of the symbols of the received waveforms.

The BCH decoder receives the demodulated symbols and stores multiple symbols. After storing 15 symbols, it builds the 60 bits of the feedback message. As there are only four feedback messages, they are stored in a ROM on the Demod/Decod module. The decoder correlates the received messages with the stored ones and generate the feedback messages in binary form, see Table 1. As soon as the feedback message is generated, the encoder module receives the message and performs a corresponding process in the IVS. The system ends the process after decoding a feedback message and restarts another round of message decoding by demodulation another 15 downlink waveforms.

3.2 Hardware Interfaces

The IVS needs a proper input and output interfaces. To receive the MSD data bits, the existing Controller Area Network (CAN) in vehicles can be a good source. CAN is a bus protocol in cars that can be used as an access to construct information for the MSD [8].

As the modules send and receive the data stream bits through a speech encoder/decoder, the interface between the GSM module and IVS modem can be done through an inter-IC sound (I^2S) bus which is a vital solution for IC manufactures to interface their ICs with other digital audio processors [10].

The designed modules can interface with a GSM module that supports I^2S bus. The u-blox LEON-G200 GSM module supports I^2S bus for digital audio data transmission [11][12]. It has four I^2S wires, the clock source (SCK), word select (WS), TX, and RX. Sampling frequency is 8 KHz, the word length is 16 bits, and the clock frequency is 256 KHz [12]. These parameters match our design modules. The feedback messages consist of 15 symbols, and each symbol is represented by a waveform that consists of 512 bit, $\underline{e} = \{e_l\}$, $l = 0, 1, ..., 1151$; therefore, there are $15 \times 512 = 7,680$ bits to be received through the I^2S bus for each feedback message demodulation. Considering the GSM module clock frequency, a feedback message can be received in $7680/256 = 30$ ms regardless the chip delay of few nanoseconds in our designs. Therefore, it can be verified that the design follows the 3GPP eCall standards.

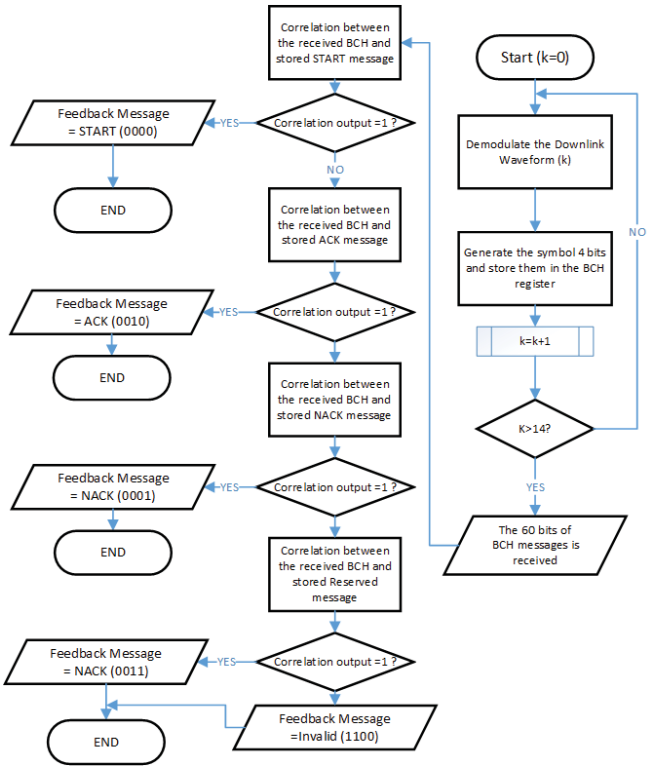


Fig. 4: The designed system flowchart.

4. FPGA Implementation

The designed system is simulated and implemented on a FPGA hardware chip by using Xilinx ISE tool and a Xilinx FPGA device [9]. Verilog HDL is employed to develop the RTL of the module. The module is compiled, synthesized, and simulated. Figure 5 shows the RTL architecture of the developed module.

The module has three input ports and three output ports. The

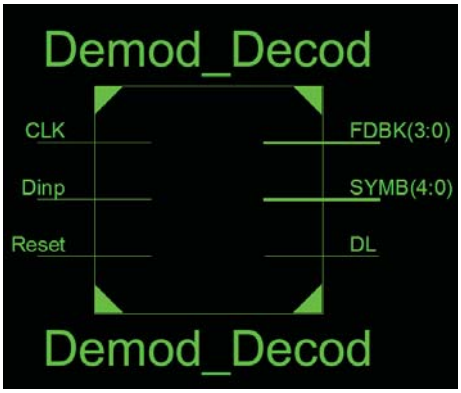


Fig. 5: The developed RTL of the Demodulator/Decoder module

"CLK" is the clock source input of the module, the "Dinp" is the input port for the MSD data, and the "Reset" is used to reset the module. The output ports are the FDBK[3:0], SYMB[4:0], and DL. The FDBK [3:0] generates the four bits of the feedback messages and they are the controlling information of the IVS transmission status. The SYMB[4:0] represents the four bits of the symbols, SYMB[3:0], and one bit to indicate if the symbol is invalid, the Most Significant Bit (MSB) of the SYMB. The DL port is designed to generate the downlink waveforms for testing and verification purposes. Both SYMB[4:0] and DL output ports are not necessary for controlling the IVS transmission status, but we designed the ports for research and test purposes. The FDBK[3:0] is the results of decoding the demodulated symbols.

4.1 Simulation

Verilog HDL is employed to design a test bench to simulate the designed module. All the 16 possible downlink waveforms are generated to build the 15 waveforms that are necessary to modulate the feedback messages. Using the Xilinx ISE simulator tool, the three feedback messages and the reserved feedback message in Table 1 are generated in the test bench and applied to the input of the module. The simulation checks all the downlink waveforms of feedback messages and the module demodulated the symbols and generated the corresponding feedback messages accordingly. The simulation of the four feedback messages and the effect of the Reset input is shown in Figure 6.

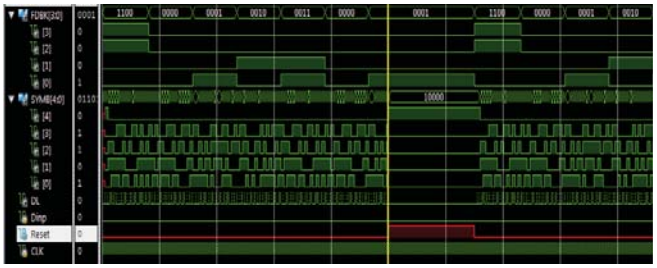


Fig. 6: The simulated feedback messages and the Reset port.

The START feedback message is simulated and shown in Figure 7. Note that the FDBK[3:0] represents the invalid message, "1100", until all the 15 waveforms of the START messages are demodulated and decoded. As soon as the 15 waveforms are demodulated (compare the simulated symbols in Figure 7 and START hexadecimal digits), the FDBK[3:0] generates the START message, "0000".

The ACK message is simulated and the result is shown in Figure 8. In both cases, it can be seen the downlink waveforms in trains of binary digits. The symbols are demodulated based on the received downlink waveforms.

4.2 Hardware Implementation

The developed module is implemented on a Virtex-5 FPGA device. The OpenSPARC evaluation platform with an

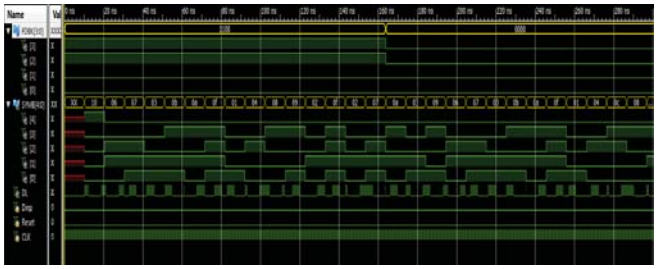


Fig. 7: The simulated START message.

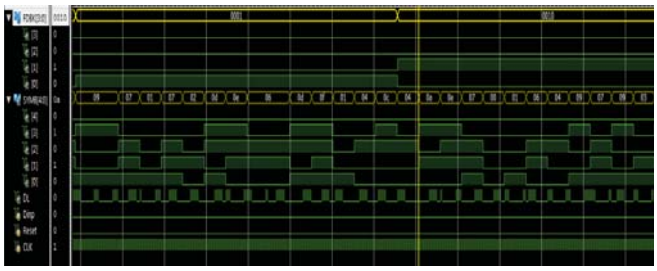


Fig. 8: The simulated ACK message.

installed Xilinx XUPV5-LX110T FPGA chip is employed [13][14]. The utilized FPGA kit is shown in Figure 9. The FPGA device have 69,120 slice registers and Look-Up Table (LUTs).

The utilized logic cells to implement the developed module

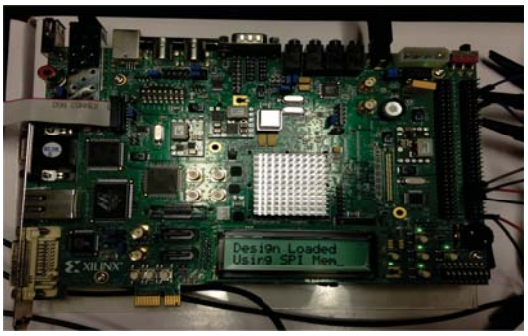


Fig. 9: The utilized FPGA evaluation kit.

are optimized by using Xilinx ISE tool. Using the tool, functions are realized for floor planing, placement, and routing of the design implementation. The utilized logic cells, LUTs, and flip flops are illustrated in Table 3.

Table 3: The logic cells utilization.

Logic Utilization	Used	Available	Utilization
Slice Register	8680	69120	12 %
Used as Logic	21025	69120	30 %
Slice LUTs	21070	69120	30 %
Bonded IOBs	13	640	2 %

5. Test and Verification

The designed module is tested and verified. A complete set of downlink waveforms are applied to the input of the Demodulator/Decoder module. Using Verilog HDL and FPGA technologies, a sub-module is designed to generate the structure waveforms to transmit the feedback messages in Table 1. The output of the sub-module, which is the "DL" port, is connected to the input pin of the Demodulator/Decoder module. Both generating downlink waveforms sub-module and the Demodulator/Decoder module work under the same clock frequency.

Two different frequencies are employed to test and verify the functionality of the designed system. The clock sources can be an external clock generator or an internal oscillator on the utilized FPGA evaluation kit. Clock frequencies 27 MHz and 33 MHz are implemented. The evaluation kit is of a built-in frequency generator for the two frequencies.

The extension ports of the FPGA evaluation board are assigned as the input and output pins of the designed module. A four channel oscilloscope is employed to study the generated symbols and feedback messages. As the I^2S bus is employed to interface the designed module with a GSM module, the downlink waveforms are applied to the module in the binary form. The output of the module is also in the binary form, which is the input data of the turbo encoder module in the IVS.

The 27 MHz is used as the clock frequency to run the implemented module on the FPGA device. Figure 10 shows a sample of the demodulated symbols that are represented by the received downlink waveforms. The four bits of the SYMB[3:0] are shown in different colors (Yellow, Blue, Purple, and Green respectively) and numbered as 4, 3, 2, and 1 respectively.

Decoding each set of 15 demodulated symbols, the

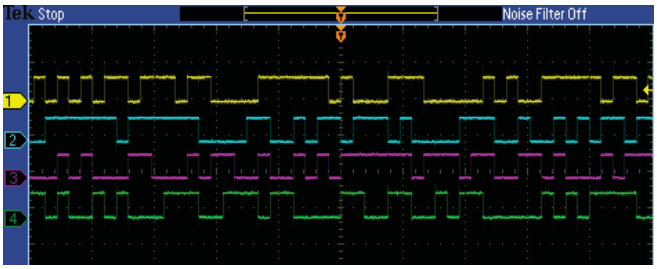


Fig. 10: The demodulated symbols, 27 MHz.

module generates the feedback messages. The waveforms of the feedback messages are applied to the input of the designed module. The feedback messages are decoded and represented by four bits. The FDBK [3:2] are zeros for all the feedback messages and the FDBK [1:0] represents the feedback messages. It can be seen that the START message (0000), the ACK message (0010), the NACK (0001), and the Reserved message (0011) are decoded and shown in

Figure 11. The two bits of the FDBK [1:0] are shown and numbered as 2 and 1 respectively.

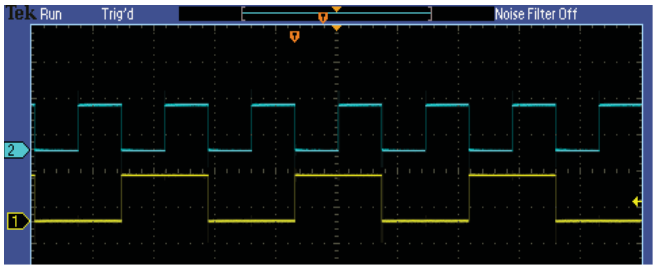


Fig. 11: The decoded feedback messages, the clock frequency is 27 MHz.

In Figure 12, it can be seen that the feedback messages are generated after decoding each 15 demodulated. The figure, compares two bits of the modulated symbols, SYBL[3:2], with the two bits of the feedback message, FDBK[1:0]. Note that the feedback messages have a duration time of a set of 15 symbols.

The clock frequency is increased to 33 MHz to see the response of the module to a higher frequency. The 33 MHz is also a dedicated output clock frequency on the utilized FPGA evaluational platform. Figure 13 shows the demodulated symbols with the higher frequency that is 33 MHz. The feedback messages, the START message (0000), the ACK message (0010), the NACK (0001), and the Reserved message (0011), are also successfully decoded with 33 MHz of the clock source frequency, as they are shown in Figure 14.

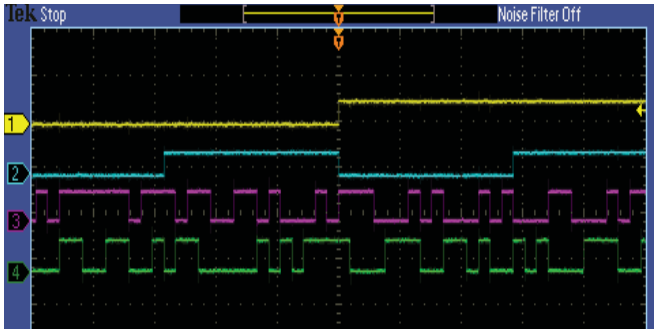


Fig. 12: The decoded feedback messages and modulated symbols, 27 MHz.

6. Conclusions

The demodulator and decoder modules of the in-vehicle system (IVS) are developed on a single module by using FPGA technologies. The hardware chip is designed, implemented, tested, and verified. Two on board frequency sources, 27 MHz and 33 MHz, on the FPGA evaluation platform have been tested. In order to modulate all feedback messages, a complete set of downlink waveforms are generated and applied to the developed module. The output feedback messages and modulated symbols are analyzed and verified.

The hardware architecture and interfaces are proposed and analyzed. The CAN bus can be employed as the interface protocol for the IVS interface with the MSD sources. The I^2S protocol is proposed to interface the designed module with a GSM module. It can be noted that in all cases for the simulation and implementation of the developed module, the feedback messages are decoded and generated accordingly.

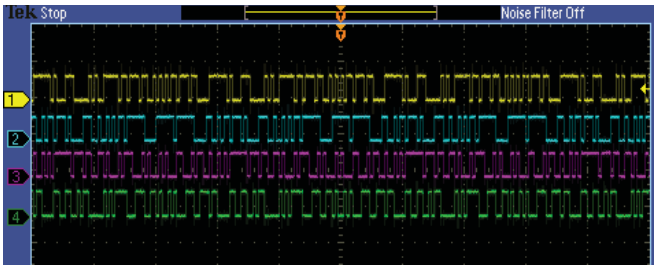


Fig. 13: The demodulated symbols, 33 MHz.

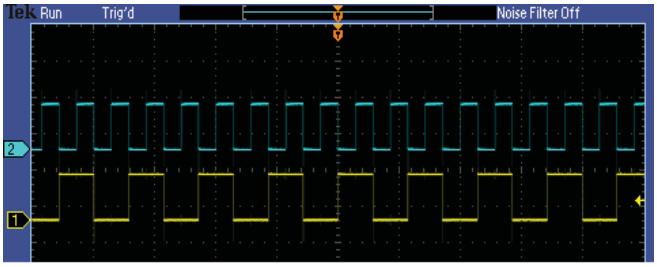


Fig. 14: The decoded feedback messages, 33 MHz.

References

[1] G. Biox, E., "Definition of a protocol of automatic identification and notification of road accidents and development of an advanced eCall system," *SAE Technical Paper 2014-01-2029,* 2014, doi:10.4271/2014-01-2029.

[2] G. Kiokes, et al. "Design and implementation of an OFDM system for vehicular communication with FPGA technologies," in *IEEE 6th International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, Athens, Greece, April 6-8, 2011, pp. 1-6.

[3] C. Yao, et al. "VLSI implementation of a real-time vision based lane departure warning systems," in *IEEE 12th International Conference on ITS Telecommunication*, Taipei, Taiwan, Nov. 5-8, 2012, pp. 170-174.

[4] Eroupean Commission, "Commission delegated regulation (EU) 305/2013," Brussels, 26.11.2012.

[5] "eCall data transfer; in-band modem solution; general description," 3GPP, Tech. Rep. TS26.267.

[6] M. Werner, et al. "Cellular in-band modem solution for eCall emergency data transmission" in *IEEE, Vehicular Technology Conference,* Barcelona, Spain, April 26-29, 2009, pp. 1-6.

[7] Eroupean Commission, "Vice-President Kallas Welcomes Parliament's Vote on 112 eCall," Press Release, Brussels, April 15, 2014. website: $http : //europa.eu/rapid/press - release\_IP - 14 - 438\_en.htm$.

[8] Xilinx, “Introduction to the controller area network (CAN),” Application Report, SLOA101A, August 2002.
[9] Xilinx,“http://xilinx.com/tools/designtools.htm.”
[10] Texas Instruments, “Using the I2S audio interface of DS90Ux92x FPD-Link III devices,” Application Report (SNLA221), June, 2013.
[11] u-blox AG, “LEON-G100/G200 - Data Sheet” GSM.G1-HW-09001-B, 2009.
[12] u-blox AG, “Wireless modules, data and voice modules, AT commands manual” WLS-SW-11000-2, 2011.
[13] Xilinx, “ML505/ML506/ML507 evaluation platform user guide” UG347 (v3.1.2), May 16, 2011.
[14] Xilinx, “Virtex-5 family overview,” Product Specification DS100 (v5.0), February 6, 2009.

Memory Management and Optimization in Lumousoft Visual Programming Language

Xianliang Lu
Lumousoft Inc.
Waterloo Ontario Canada
lu.x@lumousoft.com

Abstract— this paper presents memory management and optimization in Lumousoft Visual Programming Language (VPL) for microprocessor-based embedded system. Memory management has impacts on cost and power consumption as well as program performance, especially for the limited hardware resource of microprocessors. Real time, Safety and reliability of current complex and sophisticated electronic device are the most important factors in the design. Based on Control Flow Graphs (CFG) and Data Flow Graphs (DFG), variable liveness and a living list of variable is discussed in consideration of flow and context sensitive path as well as pointers. Following the rule that different variables can be mapped to the same physical memory as long as they are not alive at the same program path point, Lumousoft VPL can fully recycle memory and eliminate reductant assignment code in the static, safe and reliable approaches, resulting in lowering product cost, reducing the latency of memory transformation and decreasing power consumption and improving the real time performance.

Keywords—VPL; memory ; optimization; pointer; CFG; DFG

1 Introduction

Embedded based electronic products play more and more pivotal role in our daily life. Memory is one of the most important factors to be considered on both sides of embedded system designer and customer. The more memory means more functions and flexibility. On the other hand, it also means more power consumption [3,4] that becomes current portable electronic design critical issue. Making full use of memory and reduce reductant code are current compiler design trend [1].

Memory allocation approaches can be categorized into static and dynamic memory allocation[1,8,9,10]. Static memory allocation approach is to allocate memory during compilation. On the other hand, dynamic memory allocation approach is to allocate memory during running time. The static method is very fast access, safe and reliable, while dynamic allocation requires less memory but has latency, fragment and sometimes may cause memory leakage and unstable. Most of compilers adapt stack method to perform static memory allocation due to relatively simple and safe with good memory hierarchy. The stack method actually allows a variable first in and last out, it requires more memory and still has latency to perform the action of push and pop.

Data flow graphs (DFG) and control flow graphs (CFG) technologies give a very efficient way to manage memory, check the correctness, detect dead code and optimize process schedule during the compiling period [6]. Since this technology is complicated, it is hard to be found large-scale implementation for memory allocation. Data overflow or overlap should be avoided in embedded system design. Microprocessors have limited hardware sources, especially for data memory. If memory pool is not enough for heap or stack, it will come up with memory overflow or overlap resulting in unknown bugs.

The Lumousoft visual Programming language (VPL) [11] allows the user to write code in the block environments. The Lumousoft VPL program consists of sequential blocks. Each variable in a module should have a unique name, in other words, a variable cannot be declared more than twice unlike a variable with the same name can be declared in the different curly bracket in a textual language like c language. As we know, most textual language compilers use the stack method for the local memory arrangement when context switches like call function, this method is simple and safe, however, it requires more physical memory, and may cause stack memory overflow or overlap. Furthermore, it requires time to perform stack action like push and pop resulting in latency and increasing code size.

Variable liveness analysis is time-consuming and expensive activities, especially when the pointer [13] involved, it becomes more complicated. Usually this technology is applied to partial application in current compiler design. Data flow graph and control flow graph offer an efficient approach to define variable liveness. Therefore, we can efficiently allocate physical memory to variable, and map the same memory to the different variables if they are not alive at the same program point. Sometimes assignment can be eliminated if the variable on the right hand side of the assignment will be no longer used in the future, and the variable on the left, which data type requires no more than physical memories than the dead variable does, can be directly assigned the same physical memory of the dead variable. As a result, the assignment statement can be eliminated and codes can be reduced. This is

very helpful for compilation optimization since the compiler can reduce tons of intermediate assignments.

Lumousoft VPL is pointer based language in the block environment. By flow-sensitive and context-sensitive path analysis, the whole variable liveness and relationship at the program path point can be identified, based on these liveness and relationship, memory optimization can be reached.

2 CFG Equations

CFG analysis has two methods, backward and forward analysis. Practically, backward method can be used to determine liveness; while forward analysis is manipulated to determine propagation, or reach definition. The equations at the entry and the exit edge of node n are listed as below.

2.1 Backward equations

$$\begin{cases} in[n] = use[n] \cup (out[n] - def[n]) & (1) \\ out[n] = \cup\ in[s] & (2) \end{cases}$$

Where :

s - successor,
in[n] –a set of variables at entry edge of node n
out[n]- a set of variables at exit edge of node n
def[n]- a set of variable is generated

2.2 Forward equations

$$\begin{cases} in[n] = out[p_1] \cap \ldots \cap out[p_k] & (3) \\ out[n] = (in[n] - kill[n]) U\ def[n] & (4) \end{cases}$$

Where :

in[n],out[n] - the same as above Backward Equations define.
def[n] - new definition
$P_1,\ldots, P_k$ -- predecessors of n in CFG
Kill[n] – definition will no longer be used.

3 Pointer Analysis

A pointer is a variable whose value is the address of another variable. Therefore, a variable cannot be simply killed, we have to consider whether the address of this variable has been used by a pointer, and so for the variable on the left hand side of statement which cannot be simply considered to be generated or defined. We can break variable V into two groups Vp (pointer variable) and Vnp (non pointer variable).

$$Vp \subseteq V, Vnp \subseteq V \quad (5)$$

$$Vp \cap Vnp = \emptyset \quad (7)$$

A pointer variable may point to many variables, for example, in Figure 1 the pointer px points to x at node n3 and z at node n2. When px flows to successive join node n4, px may point to x or z. The below expression stands for that the pointer variable px may point to one of the variables in the list in the application:

$$P(px)=\{x_1,x_2,\ldots,x_k\} \quad k=1,2,3,\ldots \quad (8)$$

At the entry edge of the node n, if $\forall px \in Vp \wedge px \in out[p_1] \cap \ldots \cap out[p_k]$ then

$$Pin[n](px) = Pout[p_1](px) \cup \ldots \cup Pout[p_k](px) \quad (9)$$

Where

Vp is a set of pointer variable

P(px) is point- to list of the pointer px

Pin[n](px) is a set of the variable that the pointer px may point to at the entry edge of node n

Pout[n](px) is a set of the variable that the pointer px may point to at the exit edge of node n

At the exit edge of node n, we need to consider three conditions. First of all, when px is defined in a reference definition like px=&x, the variable x is in the point-to list of px. Secondly, when px is equal to other pointer py like px=py, the content of point-to list of px is the same as py. Finally, if px is not defined, px keeps the same point-to list as that at the entry edge. The point-to list for each pointer variable px can be given as below:

$$\begin{cases} Pout[n](px) = ref[n] & px \in def[n],\ ref[n]! = \emptyset \quad (10) \\ Pout[n](px) = Pout[n](py) & px \in def[n], py \in Vp \quad (11) \\ Pout[n](px) = in[n](px) & px \notin def[n] \quad (12) \end{cases}$$

Where

ref[n] -- a set of variable that present it as an address other than its value in reference definition, like &x.

The following is an example of computation algorithm for the point-to variable list by utilizing above equations.

```
foreach n
  foreach px
      Pin[n](px) ← Ø;
      Pout[n](px) ←Ø;
First = true
Repeat
  Change = false
  foreach n
    foreach px
      If FirstRepeat ∨ (∃ px ∈ Pout[p] ∩ Pin[n])
          Ptemp(px) = Pin[n](px) ∪Pout[p](px)
       Else
          Ptemp(px) = Ø
      If Ptemp(px) ≠ Pin[n](px)
          Change = true
      Pin[n](px) ← Ptemp(px)
      If px ∈ def[n] ∧ ref[n] != Ø
          Pout[n](px) ← ref[n]
      Else if px∈def[n] ∧ py∈Vp
          Pout[n](px) ← Pout[n](py)
      Else
```

$Pout[n](px) \leftarrow Pin[n](px) \cup Pout[p](px)$
First = false
until !Change

4 Liveness Analysis

Practically, the CFG backward approach can be implemented to solve the problem variable livesness. Lumousoft VPL is pointer based language, the variable liveness should involve all variable including pointer variable. In Lumousoft VPL it takes three steps to obtain variable liveness.

1. Computes point-to variable list of each pointer variable at both entry and exit edge of the node as previous mention
2. Using normal CFG backward to compute variable literal liveness at both entry and exit edge of the node without considering pointer after iteration. In other words, all variables are assumed to be independent of each other without any relationship to other variable. In this way we can easily use normal backward method to compute variable liveness through an iteration computation. If a pointer variable is used to get the content of the address that pointer variable point to, namely *p, this pointer variable p ∈ use[n].
3. We can combine point-to variable list computed by step1 with the variable liveness by step2 to compute real variable liveness as following expressions indicate at each node:

$$Realin[n] = in[n] \cup \left(\cup_{px \in Vp \cap in[n]} P\ in[n](px)\right) \quad (13)$$

$$Realout[n] = out[n] \cup \left(\cup_{px \in Vp \cap out[n]} P\ in[n](px)\right) \quad (14)$$

Where

Realin[n],Realout[n] are the set of the variable at the entry and exit edge of the node n respectively

in[n], out[n] are the set of the variable at the entry and exit edge of the node n respectively derived in step 2.

5 Context Sensitive Analysis

5.1 Living List

Lumousoft visual programming language is specially designed for microprocessors with limited hardware resource. For the safety and reliability this language does not allow recursion since the recursion may run the risk of using up stack memory. Lumousoft VPL does not use stack approach to call graph for context switch, instead, all local variable in the same thread are put on the same list, and the living relationship of variables can be clearly identified. The same physical memory can be allocated to different variables as long as they are not used at the same program path point. In this way, memory recycles can be realized.

Call graph or function can be broken into inline and regular call module function. As for inline module, we just duplicate the inline module and insert where they are called. We can use the normal flow sensitive approach to analyze. As for regular module functions like f(x,*y), the parameter of module f can be passed by a value like x and passed by a pointer like y. When call graph or module function f in the form of f(a,&b), the variable a pass its value to x, the liveness of the variable a depends on whether it is used in the successor nodes, if the successive nodes does not use the variable a anymore, it is dead at the front edge of call module f and the memory released by a can be used in the later process including intraprocedural program. On the other hand, when the parameter is passed by an address like the address of b that passesto y, if the variable b will not be found usage in the successive nodes, the variable b is still alive at the entry edge of this call graph or module, it will die somewhere in the call module process. However, in this case Lumousoft VPL supposes that the variable which address is passed to the module will live through the module process and die at the end edge of the call module if it is not used in the successors.

In order to allocate memory efficiently and make memory fully recycle, we need to identify all the living variables at each program path node, these living variables at the same time cannot share the same physical memory while those variable that are not alive simultaneously can share the same physical memory.

A special list of variable needs to be introduced to facilitate analysis. This list only concerns about which variable is alive simultaneously.

Define a living list L consists of sets of variables that are alive at the same time.

$L \succcurlyeq \{ L1, L2, \ldots \}$

The symbol $\succcurlyeq$ represents the list of living variable group set. L1,L2, are the sets of variable that are alive simultaneously. But a variable in L1 may not live with another variable in L2 at the same time.

The list has following features:

- Reduction: For a list $L \succcurlyeq \{ L1, L2 \}$, if $L1 \subseteq L2$ indicating that the variables in the L2 including L1 are alive simultaneously, so L1 can be eliminated in the list L.

$$L \succcurlyeq \{L1, L2\} \succcurlyeq \{L2\} \quad (15)$$

- Position exchange: for a list L ≽{ L1, L2 }, then

$$L \succcurlyeq \{ L1, L2 \} \Leftrightarrow L \succcurlyeq \{ L2, L1 \} \quad (16)$$

- Operation: For a list L ≽ { L1, L2 } and LL, then

$$LL \cap L \succcurlyeq LL \cap \{ L1, L2 \} \succcurlyeq \{LL \cap L1, LL \cap L2\} \quad (17)$$

$$LL \cup L \succcurlyeq LL \cup \{ L1, L2 \} \succcurlyeq \{LL \cup L1, LL \cup L2\} \quad (18)$$

A module or function can collect all the living list of the nodes to get a module living list as below:

$$ML(f) \succcurlyeq \{L[1], L[2], \ldots, L[i]\} \qquad i \in \{1,2,3,..\} \quad (19)$$

Where f ∈ MF , MF is a set of module or function

ML(f) is a living list for the module function f

L[i] is a iving list at the node i.

5.2 Living List at a Node

So far, we know there are two sets of variable (in [n] and out [n]) at a node, they can be part of living list. However, if a statement at the node contains call graph or module which in turn needs some other variables to perform a module function and these variables are generated and killed in the module function, these variables cannot be indicated at both the entry and exit edges of the node Therefore, we introduce a context list to represent the relationship of current node variable and variable used in the call module function.

If a node statement contains a call graph then

$$context[n] = \big((out[n] \cup vm[n] \cup cl(f)\text{-}cl(f)\big) \cup mp(f) \quad (20)$$

Where

vm[n] – a set of variables that pass its address to the module parameter and not longer used in the successive nodes, in the other words, the lives of these variables end at the exit edge of call module f.

mp(f) –the living list of excutivable call graph or module f

cl(f) – the return variable of call graph. In lumousoft VPL the call graph or module name is a variable. For example, in Figure 1, f is a module name and a variable as well. Since the value of module variable f is assigned during execution of the module program, we render $cl(f) \subseteq def[n]$ and $cl(f) \subseteq mp(f)$.

The expression like module function (20) indicates that all variables in the in[n], except those variables die at the front edge of call graph or module, will live together with all variables that are utilized in the call graph or module.

If the node statement does not contain call graph then

$$context[n] = \emptyset \quad (21)$$

At the node n, the living list can be described as:

$$L[n] \geqslant \{ in[n], def[n] \cup out[n], context[n] \} \quad (22)$$

L[n] is the living variable list at the node n, it consists of three parts: in[n] ,context[n] and def[n] ∪ out[n]. Here in[n] is the variables at the entry edge of node; context[n] is the variable living list in the call graph and it could be null if the statement at this point does not contain a call graph; def[n] ∪ out[n] means the collection of the variables that are defined and flow out to successive nodes. Adding def[n] can prevent from eliminating the variable that is defined but not used from the living list.

5.3 Non-recursion of call graph

In general, most of the compiler does not allow recursion for microprocessor application since microprocessor has very limited memory resource to satisfy the huge stack memory requirements for recursion. For this reason Lumousoft VPL does not support recursion. If recursion happens, Lumousoft VPL will give an error message.

For each module we can use the iterative computation method to get variable liveness without considering call graph. After we obtain each node entry and exit edge variable status, we can start from the most inside module in which there is no any call module to compute context[n], therefore, we get a whole variable living list mp(f). Secondly, we move to the module that involves that call graph with the derived variable living list mp(f), and compute context[n]. Therefore, we get the whole variable living list of this module. In this way we move from the inside module to the outside, step by step, we can compute the whole variable living list for every module. From this list we can define which variable can share same physical memory, which not. Using this static method we can maximally use memory, and even a variable in one module can share their physical memory with another variable in the other module. The whole physical memory can be predicted and allocated without memory confliction or overlap that may cause embedded system failure. Furthermore, it can eliminate the stack action resulting increase process speed without the latency of stack action of push and pop up.

5.4 Recursion of Call module

In general, stack technology is implemented in recursion method. When call graph takes place, it is unnecessary to push all the variables into the stack, we can choose the variables that live throgh the process of call graph or module to be pushed into the statck. According to (20)we define

$$Vs = out[n] \cup vm[n] \cup cl(f)\text{-}cl(f) \quad (23)$$

$$Vs \subseteq Vstack \quad (24)$$

Where

Vs – the set of variable that are pushed into the stack when call graph.

Vstack – stack memory.

Now we treat the stack memory as a special variable and put it into the living variable list. Rewriting (20), we got:

$$context[n] = Vstatck \cup mp(f) \geqslant$$

$$\{Vstack \cup L[1], Vstack \cup L[2], \ldots, Vstack \cup L[n]\} \quad (25)$$

Supposed at node k there exits a call graph ff in the module f

$$L[k] \geqslant \{in[k], def[k] \cup out[k], context[k]\} \quad (26)$$

$$context[k] = \big((out[k] \cup vm[k] \cup cl(ff)\text{-}cl(ff)\big) \cup mp(ff) \geqslant$$

$$Vstack \cup mp(ff) \quad (27)$$

From(26) and (27) rewrite (25) considering (15) we obtain:

$$Vstatck \cup mp(f) \geqslant \{Vstack \cup L[1], Vstack \cup L[2], \ldots,$$

$$Vstack \cup mp(ff)\} \quad (28)$$

We can keep replacing *mp() by its* living list until the module is the one we have call before, namely f==ff. Therefore

$$Vstatck \cup mp(f) \geqslant \{Vstack \cup L[1], Vstack \cup L[2], \ldots,$$

$$Vstack \cup mp(f)\} \quad (29)$$

Recursion cannot be performed forever, at some point it must return, therefore we got:

$$Vstatck \cup mp(f_{given}) \geqslant Vstatck \cup \{ \, ,_{\forall f,\, k \notin mf(f)} L[k](f) \, ,_{\forall f,\, k \in mf(f)} in[k](f) \, ,_{\forall f,\, k \in mf(f)} out[k](f) \cup def[k](f) \; \} \quad (30)$$

Where, mf(f) is a node that contains a call graph or module in a module f. While, ∀ f ∈ all inside module in the given module f_{given}, f also includes f_{given} . The symbol ,stands for collectting list that satisfy the condition in the subscript position. [k](f) means the set of the variable is at the node k in the module function f.

The variable living list of a module can be obtained, the stack and variable can be put together to allocate memory. If the stack is not used, its memory can be used for other variable that will not cause confliction.

6 Memory allocation

During compiling, each variable must be assigned a physical memory. According to variable living list we can map physical memory to different variables as long as these variables do not occur in the same item of variable living list. A variable might have more than one life, during different life span a different physical memory might map to it, this allows the compiler easily to allocate memory seamlessly and smoothly without gap and make usage of memory more efficiently.

7 Assignment Elimination

When the variable on the right hand side of an assignment statement is going to be dead, in the other words, it will be no longer used in the successive nodes, its memory should release for other variable usage. The variable on the left side is a new born variable and needs an available physical memory to be mapped to it. If the released memory is mapped to the new generated variable, the assignment can be eliminated, as a result, program code size can be reduced and speed up process. However, we cannot always allocate a physical memory to both side variables in the following case.

if the data type of the variable on the left hand side of assignment requires more memory than the number of memories that just release. For example the data type of the variable on the left hand side is 4 byte variable, while the data type of the variable on the right hand side is 2 byte. In this case, if the released memory of 2 bytes from the variable on the right hand side map to the variable on the left hand side of assignment statement, the last 2 byte memory that following the released memory might be occupied by other living variable, the memory overlaps will occur and can lead to program failure.

8 Example

Figure 1 is an executable graphic program for Lumousoft VPL. The program consists of a main module from START to END and a module function f. Here, The block in Figure 1is considered to be the same as the node. Table 1 shows variable set,point –to list of pinter and the living list at each node.

The column "Pointer to Variable List" in Table 1 lists the pointer and the variables that the pointer may point to. They are separated by a colon in the curly bracket. Because the module f has a parameter "b" to be passed by a pointer, which variable will be pointed to depends on where the module is called. However, we know that when we call this module, the variable the pointer point to is alive through the executing this module, hence we can leave empty on the right hand side of the colon as shown at node n13. The variable living list that a pointer point to can be computed by the approach discussed in section 3.

The column "literal Livness after iteration" shows liveness of a variable that is simply computed by the backward method of CFG after iteration without considering pointer.

The column "liveness with considering pointer" shows liveness of the variable with consideration of the pointer. This column can be easily filled. Copy the content of the column "literal Livness after iteration", pick up a pointer, and then add the corresponding variables that the pointer may point to. For instance, at node n9 there is pointer px, the pointer may point to x or z, we add x, z to the content of "liveness with considering pointer" and row n9.

Before filling the column "Living List", the living list of any inside module needs to be computed out. The module f living list can be filled first at each node, we get mp(f) ⩾{{a, b},{f}}. With this module living list we can fill the main module living list as shown in the table 1. The main module living list can be derived from table1.

mp(main)⩾{ {y,z,s,x,py },{ z,s,x,f},{ z,px,x,s,k},{px,py,x,z},

{z,x,s,a,b},{px,x,z,a,b},{px,x,z,f}}

Based on this living list, each variable can be allocated by physical memory efficiently.

Notice that the variable s has two living periods, if we use different variable to replace variable s in each life span, this will not affect the program performance, say we use s1 to repace s in the first life span and s2 in the second life span, and py1 and py2 to repace py for different life span. We got :

mp(main)⩾{ {y,z,s1,x,py1 },{y,z,s1,x,f},{ z,px,x,s1,k},

{px,py2,x,z},{z,x,s1,a,b},{px,x,z,a,b},{px,x,z,f},{ px,s2,x,z }}

Hence, the variable might have different address in the different life span, this can allocate address more efficiently and smoothly without gaps.

If a pointer point to only one address and this address is known during compiling, the pointer variable can be considered to be as a constant and does not need to be allocated a memory to hold another variable address. For example py1 is the constant and equals to the address of y which address is known after allocation of memory.

Table 2 is a table of memory allocation generated by Lumousoft VPL. There are some intermediate variables during compiling. We have 13 variables in total, and only use 7

physical memory, this indicates that we reduce memory by 46% by this memory management method.

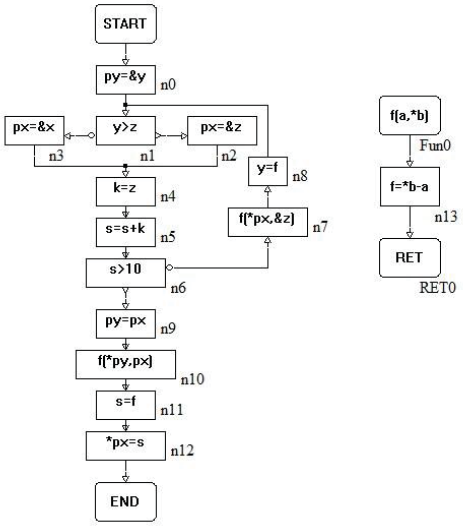


Figure 1 Graphic program in Lumousoft VPL

Table 1 variable livenss, pointer to list,living list at each node

Node		Pointer to Variable List	literal Livness after iteration	liveness with considering pointer	Living List
START	in	Ø			Ø
	context				Ø
	out	Ø	y,z,s,x	y,z,s,x	y,z,s,x
n0	in	Ø	y,z,s,x	y,z,s,x	y,z,s,x
	context				Ø
	out	{py: y}	y,z,s,x	y,z,s,x	y,z,s,x,py
n1	in	{py: y}	y,z,s,x	y,z,s,x	y,z,s,x
	context				Ø
	out	{py: y}	y,z,s,x	y,z,s,x	y,z,s,x
n2	in	{py: y}	z,s,x	z,s,x	z,s,x
	context				Ø
	out	{py: y}, {px:z}	z,s,px,x	z,s,px,x	z,s,px,x
n3	in	{py: y}	z,s,x	z,s,x	z,s,x
	context				Ø
	out	{py: y}, {px:x}	z,s,x,px	z,s,px,x	z,s,px,x
n4	in	{py: y}, {px:x,z}	z,s,x,px	z,s,px,x	z,s,px,x
	context				Ø
	out	{py: y}, {px:x,z}	z,px,x,s,k	z,px,x,s,k	z,px,x,s,k
n5	in	{py: y}, {px:x,z}	z,px,x,s,k	z,px,x,s,k	z,px,x,s,k
	context				Ø
	out	{py: y}, {px:x,z}	z,px,x,s	z,px,x,s	z,px,x,s
n6	in	{py: y}, {px:x,z}	z,px,x,s	z,px,x,s	z,px,x,s
	context				Ø
	out	{py: y}, {px:x,z}	z,px,x,s	z,px,x,s	z,px,x,s
n7	in	{py: y}, {px:x,z}	z,px,x,s	z,px,x,s	z,px,x,s
	context				{z,x,s,a,b}, {z,x,s,f}
	out	{py: y}, {px:x,z}	z,x,s,f	z,x,s,f	z,x,s,f
n8	in	{py: y}, {px:x,z}	z,x,s,f	z,x,s,f	z,x,s,f
	context				Ø
	out	{py: y}, {px:x,z}	z,s,x,y	z,s,x,y	z,s,x,y
n9	in	{py: y}, {px:x,z}	px	px,x,z	px,x,z
	context				Ø
	out	{py: x,z}, {px:x,z}	px,py	px,py,x,z	px,py,x,z
n10	in	{py: x,z}, {px:x,z}	px,py	px,py,x,z	px,py,x,z
	context				{px,x,z,a,b}, {px,x,z,f}
	out	{py: x,y}, {px:x,z}	px,f	px,f,x,z	px,f,x,z
n11	in	{py: x,y}, {px:x,z}	px,f	px,f,x,z	px,f,x,z
	context				Ø
	out	{py: x,y}, {px:x,z}	px,s	px,s,x,z	px,s,x,z
n12	in	{py: x,y}, {px:x,z}	px,s	px,s,x,z	px,s,x,z
	context				Ø
	out	{py: x,y}, {px:x,z}	Ø	Ø	Ø
END	in				
	context				
	out				
Fun0	in	Ø	Ø	Ø	Ø
	context				Ø
	out	{b: }	a,b	a,b	a,b
n13	in	{b: }	a,b	a,b	a,b
	context				Ø
	out	{b: }	f	f	f
RET0	in		f	f	f
	context				Ø
	out		f	f	f

In the table 2, the variable in a row separated by the semicolon share the same address. While the variable in a brackets indicates that the assignment is eliminated since the left and right variable share the same address. For example, in the row where the address is 0x9, the variables f, s, y, xx share the same address 0x9, the assignment in node n11 and n8 can be reduced, as a result, the process speed can be increased.

As we mentioned before the variable s has two life span. During the first period, the variable s has an address of 0x0b sharing with py2, while in the second life period, the address of the variable is 0xa.

Table 2 memory allocation by lumousoft VPL.

Address	Variable
0x7	:px(local, pointer(char)); temp(local, char)
0x8	:x(local, char)
0x9	:z(local, un char)[zz(local, un char)];
0xa	:f(local, char)[s(local, char); y(local, un char); xx(local, un char)]; k(local, char)[zz(local, char)]; temp(local, char)
0xb	:py(local, pointer(char)); s(local, char)[xx(local, char); xx(local, char)];
0xc	:a(local, char)[zz(local, char)];
0xd	:b(local, pointer(char))

9 Conclusion

Good memory management not only can reduce costs, but also speed up the process, increase reliability of the system and reduce power consumption as well. Lumousoft VPL adapts an advanced static memory management technology to fully recycle memory and eliminate reductant assignment codes, avoid memory, overlapping resulting in increasing the safety and reliability of the system.

Based on CFG and DFG technology, the variable liveness is discussed in detail with consideration of pointer and context switch situation. The variable living list of the recursion and non-recursion is also presented here. With the variable living list, the highly efficient memory allocation can be computed out according to the rule that memory can be mapped those variable who are not alive simultaneously. The copy assignment code can be eliminated if the variables on the other side of assignment statement share the same memory.

This memory management approach enables Lumousoft VPL to tremendously reduce the impact on memory requirements, increase safety and reliability of embedded system and decrease latency of memory transformation.

10 References

[1] Dick Grune, Herri E.Bal, Ceriel J.H. Jacobs and Koen G. Langendoen,"Modern Compiler Design", John Wiley & Sons, Inc., New York, NY, USA, 2000.

[2] L. Qiang, T. Todman, W. Luk, "Combining Optimizations in Automated Low Power Design," in Proc. of Design, Automation and Test Europe (DATE), 2010, pp. 1791-1796

[3] J. Cong, P. Zhang and Y. Zou, "Combined Loop Transformation and Hierarchy Allocation in Data Reuse Optimization," in Proc. of the 2011 Int. Conf. on Computer-Aided Design (ICCAD), 2011, pp. 185-192

[4] R. Banakar, S. Steinke, B. Lee, "Scratchpad memory design alternative for cache on-chip memory in embedded systems," in Proc. of the 10th Int. Symp. on Hardware/Software Codesign (CODES), 2002, pp. 73 - 78.

[5] M. Kandemir, J. Ramanujam, M.J. Irwin, et al, "A Compiler-Based Approach for Dynamically Managing Scratch-Pad Memories in Embedded Systems," in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2004, pp. 243 - 260.

[6] J. M. Barth, "An interprocedural data flow analysis algorithm" In Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, pages 119–131, January 1977.

[7] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck, Efficiently computing static single assignment form and the control dependence graph, ACM Trans. Program. Lang. Syst. 13 (1991), 451–490.

[8] P. Briggs, K. D. Cooper, T. J. Harvey, and L. T. Simpson. Practical improvements to the construction and destruction of static single assignment form. Software—Practice and Experience, 28(8):859–881, July 1998.

[9] I. Puaut. Real-Time Performance of Dynamic Memory Allocation Algorithms. 14 th Euromicro Conference on Real-Time Systems (ECRTS'02), page 41, 2002

[10] Dave Dice and Alex Garthwaite. Mostly lock-free malloc. In Proceedings of the 2002 International Symposium on Memory Management, pages 269–280, June 2002.

[11] Xianliang Lu, "Lumousoft Visual Programming Language and its IDE", The 2014 International Conference on Embedded Systems & Applications, pages 3-9, July, 2014.

[12] Khedker, U.P., Karkare, B.: Efficiency, precision, simplicity, and generality in interprocedural data flow analysis: Resurrecting the classical call strings method. In: Proc. of CC'08. (2008) 213–228

[13] Uday P. Khedker1 , Alan Mycroft2 , and Prashant Singh Rawat1, "Liveness-Based pointer analysis", SAS'12 Proceedings of the 19th international conference on Static Analysis, Pages 265-282,2012

Control of a Plotter with LabVIEW and Embedded Microcontollers

Bassam Shaer, Teresa Frigon, Adam Ferguson, Darius Bethel
Electrical & Computer Engineering Department
University of West Florida
Fort Walton Beach, FL
bshaer@uwf.edu, tlf22@students.uwf.edu, alf25@student.uwf.edu, dtb5@students.uwf.edu

Abstract— The objective of this work is to present the design of the control apparatus for a printed circuit board mill. A device with the ability to make circuit boards in just a few minutes is the electronic hobbyist's dream. It will eliminate the need to build a circuit on perforated boards, eradicate the need for the etching methods that use harsh chemicals. It will also remove the need to send Gerber files to a circuit board fabrication company. This paper will explain our research of designing a controller using LabVIEW and the myRIO, both made by National Instruments. This system utilizes computer-aided design software to layout complex images that can be printed using two brushless motors and a servo. The system also employs graphical user interfaces that can be used for system monitoring.

Keywords— LabVIEW; myRIO; FPGA; GUI; PCB; State Machine; Plotter

1. INTRODUCTION

There are several different ways to fabricate custom circuit designs. The use of a breadboard is a very effective method; it is designed for quick prototyping of an electronic circuit; however, the breadboard does not provide a permanent solution for custom project building. It can be a source of unwanted noise and can be difficult to maintain as the complexity to the design increases. There are also strip and perforated boards, but they can also be confusing to wire and very bulky to embed in an enclosure. Chemically etching printed circuit boards (PCB) is another quick fix to designing custom PCB's, but it requires using harsh chemicals. Lastly, the hobbyist can send the PCB layout files to a PCB fabrication company that can make the board for them, but this route is expensive for one-off designs and can take a while to get the board in hand. The hobbyist needs a new solution to this problem; hence the need for a PCB mill, something that can use the files created on computer-aided design (CAD) software and automatically produce a PCB.

Due to monetary constraints and time, it was decided that a plotter could serve as the perfect prototype [1]. The plotter is built on a larger scale, but still has the same design premise as the mill. The plotter and the PCB mill both need a platform; however the plotter designed for this research, uses an 8.5 X 11 sheet of plain white paper on which to draw an image, instead of milling a PCB. The system also uses motors to move a pen. Computer-aided manufacturing (CAM) software is needed to produce a machine compatible programming language called, g-code [2]. This is similar to the Gerber file, sent to PCB fabrication companies. The g-code contains the coordinates for each point that is plotted. Encoders are used to retrieve motor position information [3]. Lastly a controller was needed to control the operation of the plotter. LabVIEW is used as the programming environment for this project. For hardware control, a myRIO field programmable gate array (FPGA) is provided. Both LabVIEW and the myRIO are by National Instruments (NI). The purpose for this project is to explain the control system behind such a machine.

The control system is modeled after a mealy finite state machine. A custom interface communication protocol was designed to communicate between the State Machine (SM) of the myRIO and the microcontroller. The SM is responsible for several things which include, parsing the g-code file generated by the CAM software, turning the system on and off, system calibration analytics, starting the plot, running the plot, system shut down, sending and receiving messages, and fault condition monitoring. The system will also have two graphical user interfaces (GUI) that will be an integral part of the control system, by way of user interaction for system monitoring, emergency stopping, and system resets. There are other parts of the system that will not be covered by this paper but are included in the overall system, like a microcontroller used to communicate to a motor controller [4]. The rest of the paper is organized as follows. Section 2 discusses a brief review of LabVIEW. Section 3 presents an overview of the myRIO. Section 4, presents the formal description of the proposed system and its operations. Section 5 presents the results of the work. Section 6 ends the paper with concluding remarks.

2. LABVIEW OVERVIEW

LabVIEW provides the infrastructure to implement a controller. Due to LabVIEW's ability to operate in a parallel state when executing its code, the plotter will be able to perform multiple tasks at once i.e. track X and Y positions simultaneously.

LabVIEW is a functional programming language that uses a set of icons called virtual instruments (VI), similar to functions in C. The system has several modules that give the user access to different VIs. An operator of LabVIEW also has the ability to create VI's, and in effect every program written in LabVIEW can be a VI [5].

3. MYRIO OVERVIEW

The National Instruments (NI) myRIO is an embedded hardware device that is used much in the same way as any standard FPGA. It can be programmed in several different ways, C, C++, and LabVIEW. One can also import and reuse Hardware Description Language (HDL) code to design real, complex engineering systems. Wireless networking capabilities are also built into the myRIO, which are used for communications to both of the system's GUIs. There are unique benefits of FPGA-based hardware and LabVIEW programming due to the dataflow paradigm and inherent parallelism that they both utilize.

4. SYSTEM OVERVIEW

This overview covers all aspects of the controller, from system power on, plotting an image, and eventually to system shut down. The system is fully automated, with the exceptions that the user needs to turn on the system, select a file to plot, reset the system if a fault is detected, and to shut the system off. When the system is turned on, it will automatically calibrate both motors to the position origin. The origin is preprogrammed in each g-code file. Once the system has been calibrated, the file that will be plotted is parsed by a LabVIEW VI. Figure 1 shows a sample g-code file. Once the file has been parsed, information is sent to a microcontroller to begin plotting the image. There are constant communications between the myRIO and the microcontroller which includes the system status and the desired plotting coordinates. Upon completion of the plot, the user is given an option to plot another image or turn the machine off. The user can repower the system at any time and proceed with a new plot. In the event that a fault condition is detected, system operations are paused until the cause of the fault is removed and the fault flags are cleared. Once the fault is removed and the fault flags are cleared, the system resumes the plot. Figure 2 shows an operational flow chart of the system.

```
N1 G70
N2 SPINDLE ON SPEED 1200 FORWARD
N3 G90 G00Z0.4409
N4 G00X0.4959 Y1.4716
N5 G00
N6 G01 X0.4959 Y1.4716 Z0. F3
N7 G01 X10.4959 Y1.4716 Z0.F4
N8 G00 Z0.4409
N9 X10.4959 Y1.4716
N10 G00 Z0.4409
N11 X0.4959 Y1.6998
N12 G01 X0.4959 Y1.6998 Z0.F3
N13 G01 X0.6714 Y1.5839 Z0.F4
N14 G01 X0.8197 Y1.4331 Z0.
N15 G01 X0.9361 Y1.3145 Z0.
```

Fig. 1. Sample G-code Format.

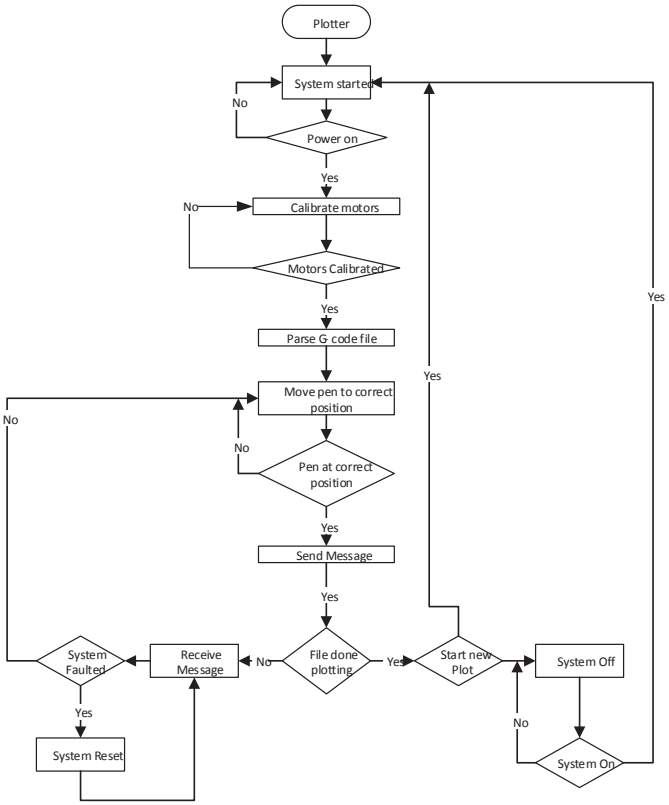


Fig. 2. Control System Flowchart.

4.1 Off State

The "Off" state of the system is a continuous loop of the SM that checks the status of the power button. While in this loop, a message is printed to the screen letting the user know the system is off. The g-code is parsed and redisplayed to the PC's GUI on every iteration of the off loop. Once the power has been turned on, the system transitions from the "Off" state to the "On" state.

4.2 On state

The "On" state of the system reads a g-code file that is parsed and displays an "ON" message to the PC's GUI for 5 seconds. When the 5 seconds are done, the motors are turned on, and the system goes to the "Calibration" state.

4.3 Calibrate State

In the "Calibration" state the motors are calibrated one at a time, first the X motor and then the Y motor. The system calibrates the motors to a position known as origin. Once the motors have been calibrated, the system is set to idle and transitions to the "Start" state.

4.4 Start State

Once in the "Start" state, the file to be plotted is read and parsed. The image that will be plotted is displayed on the PC's GUI for conformation. The image is displayed throughout the duration of the plotting process. The initial motor parameters, i.e. speed and direction, are saved, and then the system moves to the "Run" state.

4.5 Run State

The "Run" state is the heart of the systems operation. It is in the "Run" state that the system tracks the location of all three motors. The "Run" state is initialized from the "Start" state. This means that the motors are ready to plot data points; however, in order to do this the system needs to calculate how far it needs to travel to the next point. The system decides if the next point is greater than, less than, or equal to its current position. It also determines if the Z axis needs to be engaged or not.

Figure 3 shows the test case for Y axis motor. This section of the "Run" state, controls both the motors speed and direction. This information is decided based on the motors current position, and its desired position. If the motor's current position is less than the motor's desired position a message is sent to the motor controller to move the motor forward at the desired speed. Moreover, if the motor's current position is greater that the desired position, then a message is sent to the motor controller to move the motor in the reverse direction at the desired speed. Once the motor is in its desired position, a message is sent to the motor controller to stop or hold the motor at its current position.

A closed loop system is utilized to accomplish this process. The system does not try to execute the next line of g-code until both motors have reached their chosen destination. This procedure is executed in a loop until all of the g-code has been processed.

Once the system has processed its last point, the system's user is given the option to select whether to continue with a new plot or shut the system down. If the user decides to plot a new image, the system will recalibrate before plotting the new image. However if the user decides to end the system's operation, the system will return to the "OFF" state and remain there until the system is powered on again. See Figure 4 for a flowchart description. Figure 5 gives a broader picture of the overall functions of the "Run" state.

4.6 Send Message State

While the system is executing the "Run" state, it is periodically transitioning in and out of the "Send Message" state. This is the place in the system operation where messages pertaining to the system's current operations are packaged and sent to the motor controller for processing. Figure 6 show an example of the system motor position state packet sent to the microcontroller. Figure 7 show a flowchart of the "Send Message" state.

Once a packet is made and ready to be sent, a set of checksums are included at the end of the packet. This step is need to ensure the proper message is processed by the microcontroller before going to a new state. The "Send Message" state is responsible for computing which state it should go to next after a message has been successfully sent. If the power has been shut off, the system will switch to the "Shutdown" state and shuts the system off. If the system has not been calibrated properly, it will switch to the "Calibrate" state where the system is then recalibrated. It goes without saying, if the user has requested to plot a new image, then the system returns to the start state and begins a new plot. However, if the system is in full operation and is calibrated, it will proceed to the "Receive Message" state. The "Send Message" switch is turned off after each message is sent.

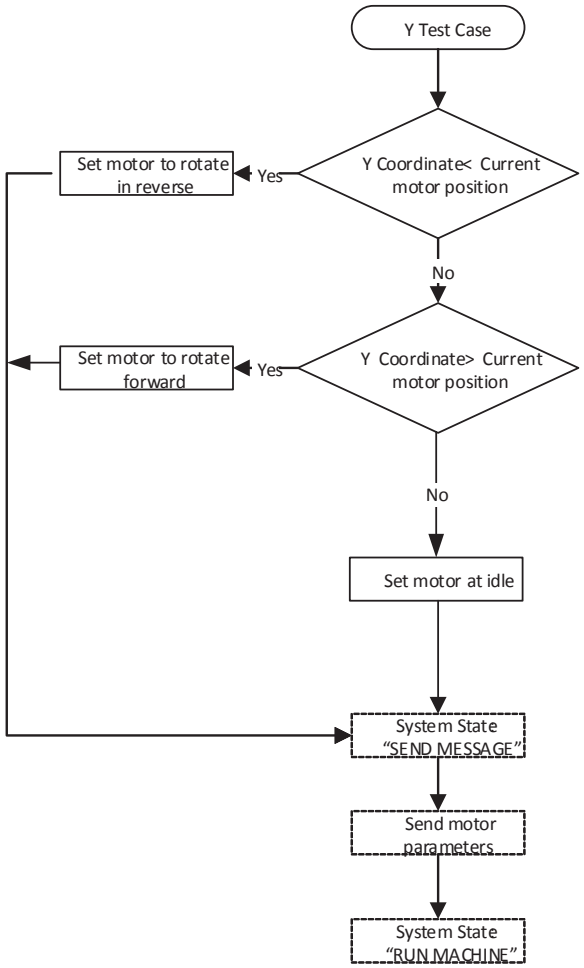


Fig. 3. Run State, Y Motor Test Position Flowchart.

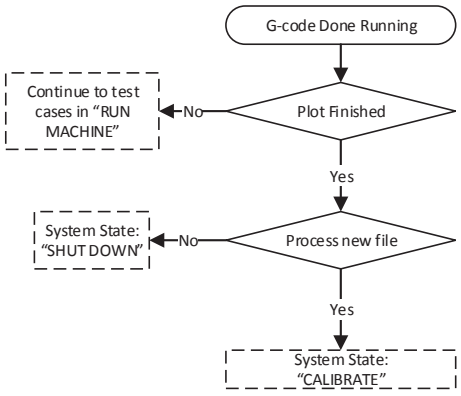


Fig. 4. Run State, G-code Done Running Flowchart.

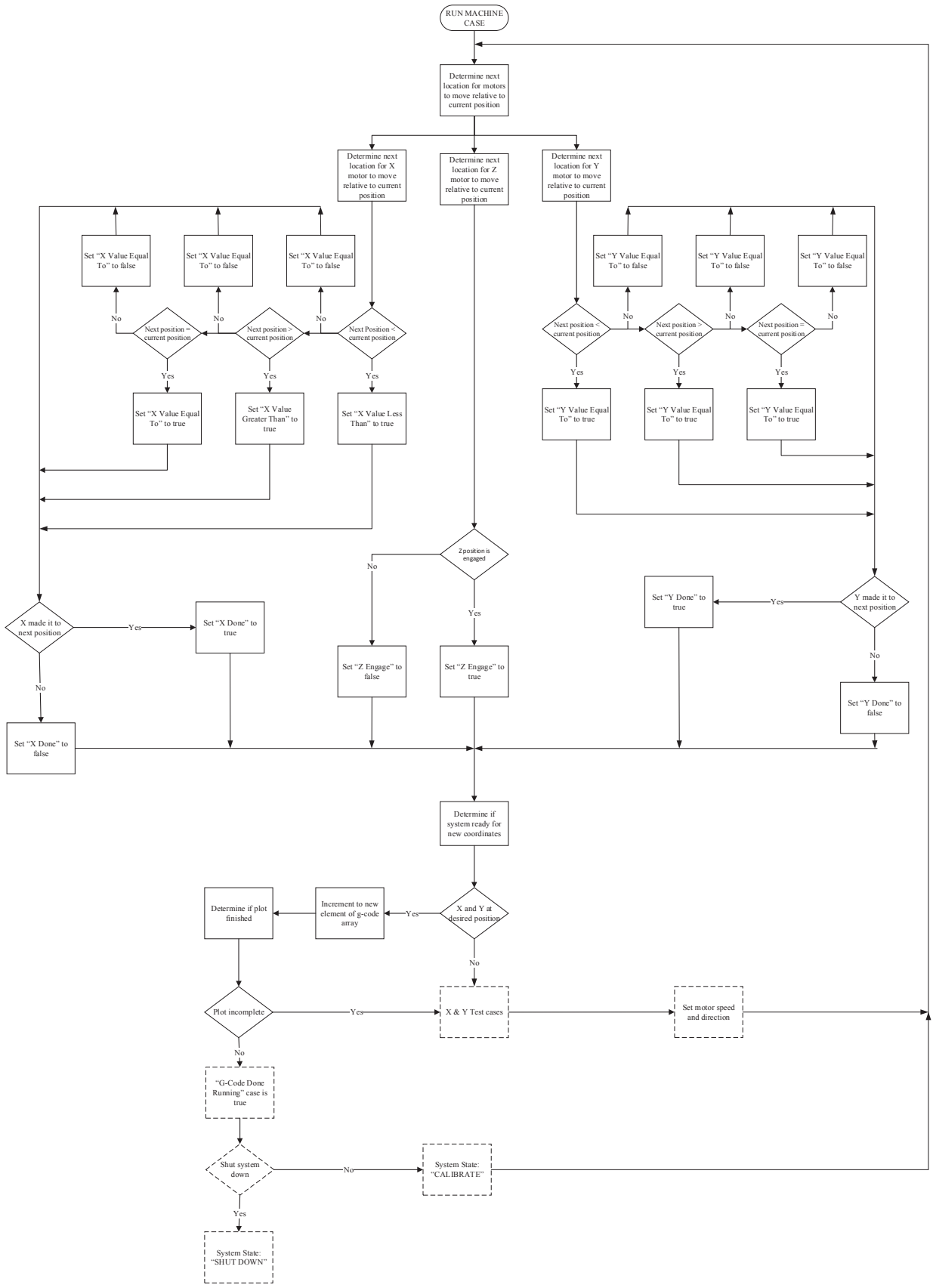


Fig. 5. Overall functions of the "Run" state.

4.7 Receive Message State

After messages are sent out from the myRIO via the "Send Message" state, a message is then retrieved from the internal message buffer of the myRIO, this is called the "Receive Message" state.

A special VI was written to retrieve messages from the message buffer, the flowchart of that VI can be seen in Figure 8. This algorithm is a pre-validation method used to ensure that garbled messages are discarded as soon as possible to make way for relevant messages. All messages begin with the hexadecimal number "55" and therefore a "55" must be received first, to indicate the beginning of a message. This lets the systems special VI know when to start recording a message to the buffer. The second section of a message should be the packet type. The third section of the message tells the size of the remaining message minus the checksums; the checksums will be verified in another process. Once the entire message is read, then the VI returns the message to the "Receive Message" state.

# of Bytes	Index	Type	Data	Description	Scalar
			3	State	
1	2	byte	C	Data Length (Bytes)	
2	3	int		X Position	10,000
2	5	int		Y Position	10,000
1	7	byte		Z Position	
1	8	byte		Status Msg 1	
1	9	byte		Status Msg 2	
1	10	byte		Status Msg 3	
2	11	int		Temp	100
2	13	int		Voltage	100
1	14	byte		Line Execution	

Fig. 6. FPGA State Packet.

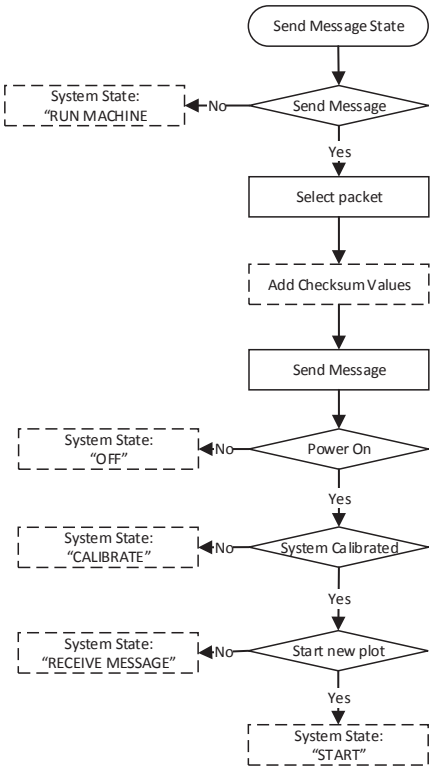


Fig. 7. Send Message State Flowchart.

The message is then parsed into an unsigned bit array. The second element of this array is read to determine the packet type. Once the packet type is determined, the checksum values are verified. If the checksum values derived are not the same as the ones receive in the message, a non-acknowledgment (NAK) message is sent to the controller, and the myRIO returns to the "Receive Message" state for the resubmission. Upon getting a valid message, the contents of the message are analyzed and the appropriate actions are taken.

There is prioritized data inside each message. If it is determined that the system has an error, the system is immediately moved to the "System Faulted" state, to be discussed later. If all is ok with the system, the SM returns to the "Run" state. See Figure 9 for a full view of the "Receive Message" state

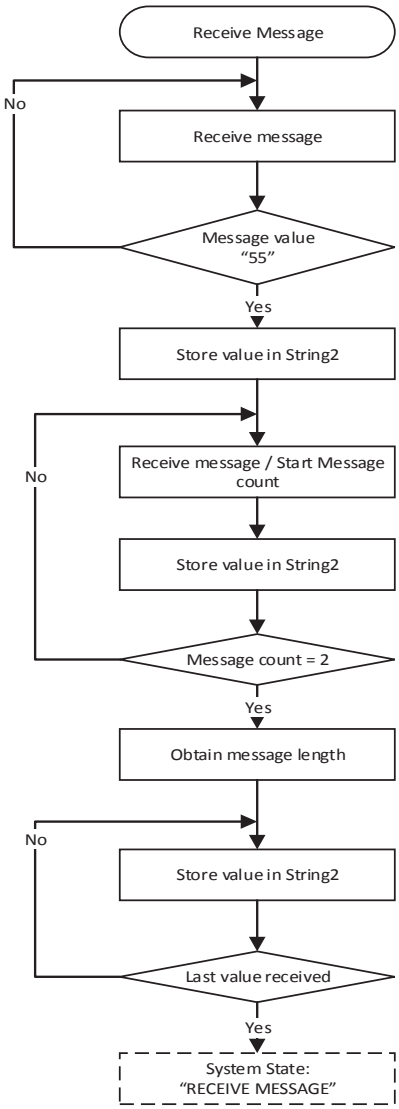


Fig. 8. Receive Message VI Flowchart.

4.8 System Faulted State

When a fault is detected, the system automatically moves to the "System Faulted" state. The system stays in this state until the condition that caused the fault is removed. When in this state, user input is required to exit. The user has to click, or press if using the tablet GUI, the "System Reset" button, and all faults are cleared. The system does not return to the "Run" state however, until it receives a new message stating that no new fault conditions exist. If no new faults exist, the system returns to "Run" state. If a new fault condition is detected, the system returns to the "System Faulted" state. Figure 10 shows the flowchart for the "System Faulted" state.

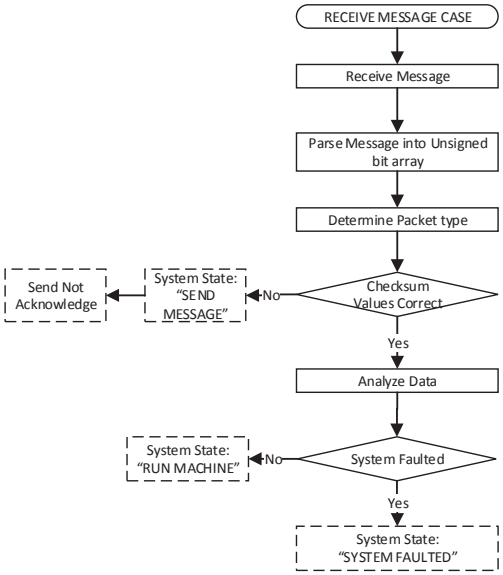


Fig. 9. Receive Message state Flowchart.

4.9 Shutdown State

The "Shutdown" state is a soft off selection used to halt system operations without damaging any equipment. Once it has been verified that the user has requested that the system needs to be shutdown, the system goes to the "Off" state where the system is turned off.

4.10 GUI

The LabVIEW program is also incorporated into the research to provide the software tools needed for system control from a GUI. There are two GUIs with which the system can be used. One is controlled from a PC, Figure 11 and the other from a tablet, Figure 12. Both of the GUIs control the system remotely. The GUI's are mentioned as part of the control system because they do give the user the ability to monitor and react to the system in real time. The GUI on the PC provides graphics of the image being plotted. Both GUI's provide fault condition monitoring, battery voltage monitoring, motor rpm, temperature of the motor controller, limit switch monitoring, motor fault monitoring, and the ability to restart the system.

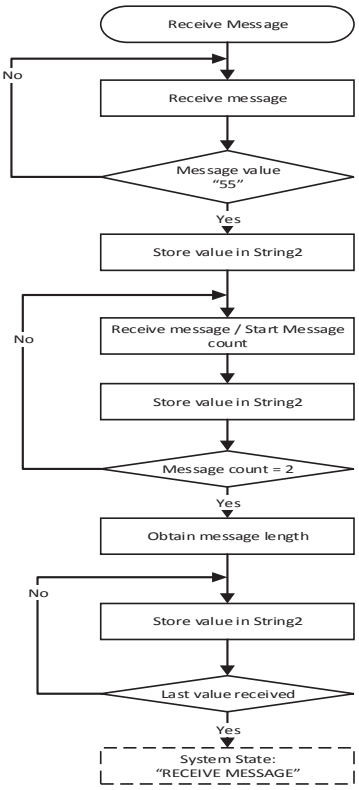


Fig. 10. System Faulted state Flowchart.

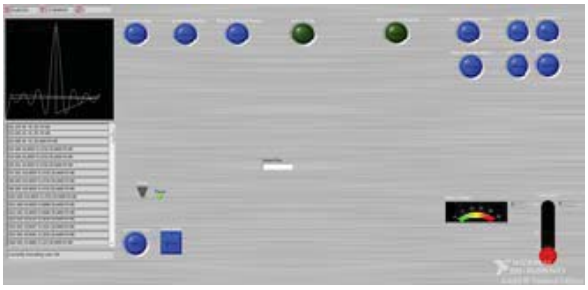


Fig. 11. PC GUI for the Plotter.

Data Dashboard is a tool developed by NI that allows custom creation of portable LabVIEW applications. In using this software the team was able to create dashboards, or a small GUI program, to display the values of network-published shared variables and deployed LabVIEW Web services on indicators, such as the fault conditions mentioned above.

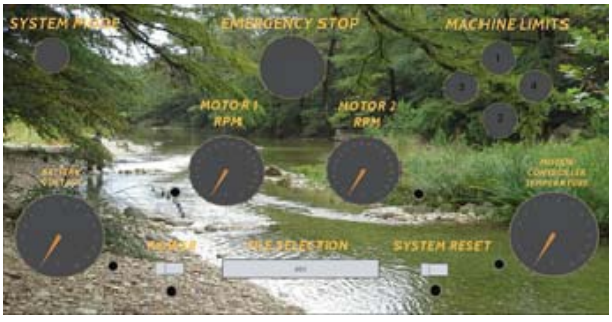


Fig. 12. Tablet GUI for the Plotter.

5. Results

Initial system testing revealed several inaccuracies. The problems however, were not due to the controller, but were partly due to the encoders that were chosen. Moreover, the platform bed had extra movements that were not accounted for during the running of the system. These problems caused overshoot and miscalculations. Figure 13 shows what the image looked like originally as designed in the CAM software, Figure 14 shows our initial plot, and Figure 15 shows the final plot of the system after adjustments that streamlined the communications processes and various tuning variables.

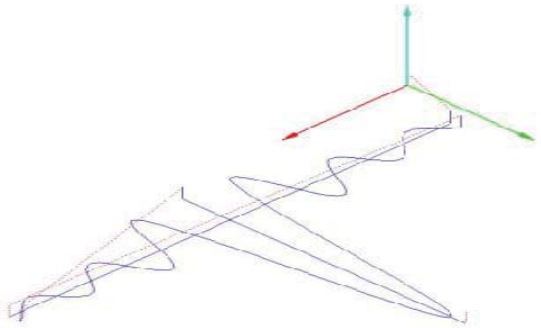


Fig. 13. Sync Function G-code Output.

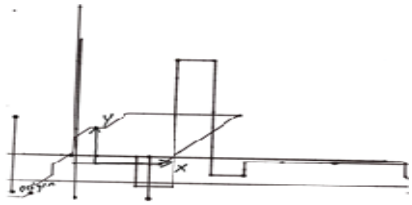


Fig. 14. Initial Plot

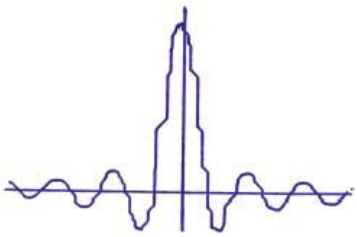


Fig. 15. Final Plot.

6. Conclusion

The need for custom PCB's will never wane and it is time that a PCB mill is introduced to hobbyists around the world. While this system is just a prototype of what is to come, it provided the opportunity to see more clearly the steps that need to be taken in order to achieve the goal of having a PCB mill in every electrical engineer, student of engineering, and hobbyist's hands. The goal for the next phase of this research is to build a smaller more stable gantry and include smaller motors so that the system can be built on a more realistic scale. There is also a need to get encoders that provide more precise position measuring, not enough attention was given to this beforehand; however, this is a crucial piece of the project.

References

[1] What is a plotter? HP Plotter explains, hp plotter HP Printer & Plotter Specialists [Online] 2015 http://www.hpplotter.co.uk/what-is-a-plotter (Accessed: January 20, 2015).

[2] Numerical control. wikipedia.org. [Online] Creative Commons Attribution-ShareAlike License, November 5, 2014. [Cited: November 25, 2014.] http://en.wikipedia.org/wiki/Numerical_control.

[3] Encoders, Anaheim Automation Inc.: ENC-LKE51_Magnetic Linear. Anaheim Automation.com. [Online] [Cited: March 4, 2014.] http://www.anaheimautomation.com/products/encoder/linear-encoders-item.php?sID=610&serID=1&pt=i&tID=1175&cID=546&dsID=611&tsID=666.

[4] Dorf, Richard C. and Bishop, Robert H. Modern Control Systems. 12th. s.l.: Pearson Education, Inc, 2010.

[5] LabVIEW. wikipedia.org. [Online] Creative Commons Attribution-ShareAlike License, December 5, 2014. [Cited: January 23, 2015.] http://en.wikipedia.org/wiki/LabVIEW.

GUI Implementation of a Vegetable Dryer using a Cortex-M4 Processor

Wongi Kim[1], Byoungchul Ahn[1], and Yeonbo Kim[2]
[1] Department of computer Engineering, Yeungnam University, Gyungsan, South Korea
[2] School of Electronic and Electrical Engineering, Daegu University, Gyungsan, South Korea

Abstract - *As semiconductor technology advances, mechanical control systems have been replaced with embedded systems to control machines easily. Most vegetable dryers are controlled by simple electronic circuits and mechanical switches, and their data are displayed very limited information of dryers such as temperature, operation time and etc. To extend functions of dryers, it is required to change user interfaces to input many data and to monitor operations of dryers. To improve these disadvantages, it is necessary to design a system using a graphical user interface and touch screen which monitor and control dryers and collect data and process them statistically. This paper describes a software design to control a dryer, which has IO interfaces, temperature setting, dry-time adjustment and humanity measurement by using a STM32F429 Cortex-M4 processor with limited embedded graphics.*

Keywords: Graphics, Embedded, User Interface, Touch Interface, Control

1 Introduction

As compared with the rapid growth of smart phones, control units of many home appliances are controlled by simple electronic circuits and mechanical switches to control and monitor their status. They have mechanical switches, simple LED indicators and seven-segment LEDs. They also show very limited information compared with many functions because of their internal space, cost, complex design and so on. Some appliances have large touch screens and displays but their retail prices are very expensive. Therefore it is necessary to design a graphical user interface with an low cost embedded processor.

To improve these disadvantages, it is necessary to design an embedded graphics system with low cost embedded processors. The information to display and control embedded systems is temperature, humidity, fan controls, camera images, video and accumulated power usage, peak power and so on. The information collected during operation can be transmitted to a main server through the internet, stored and analyzed data from remote distance. The main controls or settings are saved from touch interfaces instead of mechanical switches. The graphical user interface(GUI) provides users with easy control as well as easy upgrade or modification. Therefore, GUI can add values of a product and be updated or corrected quickly from customer requests. In order to implement the GUI interface and intelligent control functions, it is required a lot of external memory space, fast processing power for graphics and low cost.

This paper proposes a low-cost graphics implementation using a Cortex M4 processor, which is a ST32F429 chip from ST Micronics. The proposed system uses the embedded graphics hardware and a 4 GB NOR flash memory for image data. The graphics resolution is 800x480 16-bit TFT WVGA(Wide Video Graphics Array) display with a touch interface. This paper presents the software implementation techniques for embedded graphics to improve the GUI interface.

The rest of the paper is organized as follows. In section II, related works for agricultural dryers are discussed. Features and disadvantages of present dryer systems are described. In section III and IV, limitations and implementations are described to design the GUI software. Section V summarizes the implementation.

2 Related work

Before agricultural dryers have been introduced, sun drying methods are used at many places. One advantage of sun drying does not use fossil fuel during the drying process. But it needs a lot of manpower and can affect quality deterioration of agricultural product because of polluted dust, bugs and birds[1][2]. Conventional agricultural drying method takes a lot of time and drying time is affected by weather condition and surrounding environment. For this reason, dryer systems are developed by using a fossil fuel[3].

Now, most dryers of mechanical and electronic dryers use 8-bit microprocessors to control temperature and drying time. Their purpose is not to interface user friendly and to collect data but to control dryers. The existing mechanical and electronic dryers provide users with a seven-segment for display and three control switches, which are temperature, fan control, automatic dry function. Such a system based on mechanical methods is too difficult to control a dryer precisely, but one advantage of existing dryer control systems is easy to use because of simple design. Display interface using seven-segment simply displays temperature, humidity and drying time. Users can select automatic drying for control but only one or two vegetable products can be set for the automatic dry function. Therefore, most dryers have to

redesign their control panel to dry specific vegetable and they cannot use for general dryers of agricultural products. For such reasons, existing dryers have a limitation to modify and change its settings. They cannot display all information of dryers or operational errors, which are collected during operations.

It is worth to develop a low cost system with a graphical user interface and an intelligent system to control, diagnose, service dryers. As rapid growth of smartphones, GUI control systems will get attention from farmers and be controlled by smartphones. This paper presents a design and implementation of GUI software by solving several issues with low cost embedded graphics.

3 Software design limitations

To design a low cost embedded system, we have selected an ARM Cortex M4, which is a ST32F429 processor from ST Micronics. It has an internal graphics processor to process 16-bit graphic data for embedded applications. It provides 2 MB dual-bank flash memory, 256 KB SRAM including 64 KB of data RAM. It is possible to connect a display to the parallel interfaces and take full advantage of ST's Chrom-ART Accelerator graphics accelerator performing content creation twice as fast as the core alone[8]. This graphics accelerator supports efficient 2-D raw data copy, as well as image blending or image format conversion such as mixing and transparency. As a result, the Chrom-ART Accelerator boosts graphics content creation and saves processing bandwidth of the MCU core for the rest of the application.

But it has been designed to use the small screen size, 640x480 pixels. When this processor is applied to 800x480 pixels, several issues are solved to overcome graphics processing performance. They are internal memory space for graphics images, screen display time and external memory access time.

3.1 Limited internal memory

To display images fast on the screen, 2 MB NAND flash memory in the processor must be used. However, the memory capacity can store up to six 800x480 images with RGB 24-bit natural color. This internal memory must store both executable codes and images. Since the size of executable codes is 1 MB, the space is left about 1MB space for images. Memory capacity for image is very small because only 3 800x480 bitmap images can be stored in memory space. There are more than 10 screen transitions and many small sized images should be stored in the memory. Their estimated size is 5 MB. Therefore, internal 2 MB flash memory should be used efficiently for codes and images. The internal memory is designated for small-size and frequently-used images.

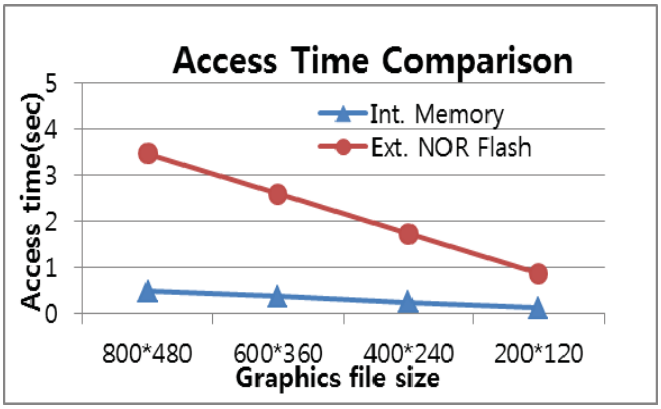


Figure 1. Access Time Comparison

Figure 1 shows the access times of the internal memory and the external NOR flash memory. The access time of the NOR flash memory is 9.5 times slower than that of the internal memory. To solve this problem, the following three options are considered.

① Internal memory utilization

Typically the memory access time of the external NOR flash memory is twice slower than that of the internal memory. Most small and frequent used images must be stored in the internal memory and other images are stored in the external NOR memory. Therefore the display time of graphics can be matched to the speed of external memory access.

② JPEG image compression

The file size of JPEG files can be reduced to one of tenth of file size of bitmap images but they are required to decode images to display on the screen. This decoding operation requires additional time and memory for the program. For fast display operation, image compression method is not efficient for small embedded system.

③ Scaling method

Because the size of total bitmap images is about 4.8MB, all images for the dryers cannot be stored in the internal memory. To solve this problem, a scaling-down method, which is reduced by half, is used. When images are displayed to the screen, images are scaled-up by duplicating method without losing processing speed and image blurring. Figure 2 shows the scaling method. Depend upon applications, images are scaled down to 1/4 or 1/16 of the original images. With this method, most frequently-used images are stored in the internal memory.

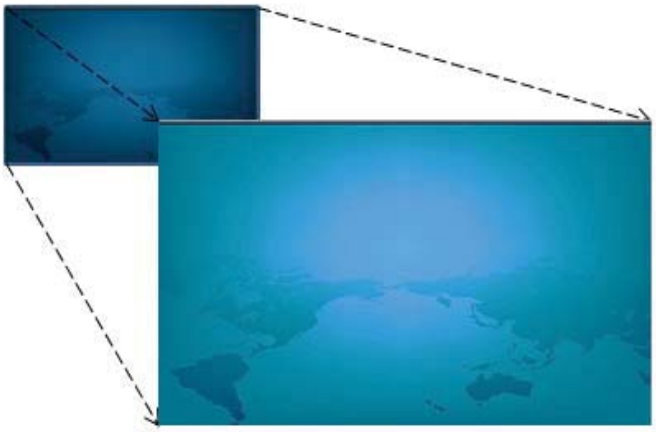


Figure 2. Image Scaling Method

3.2 External NOR flash issue

Unlike typical hard disks, NOR flash memory has several things to be considered when implementing the FAT file system. Since there are unit differences among writing and reading, erasing blocks of the flash memory, it does not support overwriting the same memory blocks after they are erased.

Flash memory devices of high-capacity and high-speed such as SSD(Solid State Device) can immediately apply a file system used in the existing hard disk using the FTL(Flash Translation Layer). It would pay a lot of costs if FTL is implemented in low-capacity-embedded devices without operating systems[7]. To write data in small memory block with low-performance, FAT file system is used for NOR flash memory using block buffers.

To implement the FAT file system in the flash memory, FatFs module, which is a generic FAT file system for embedded systems, is used. FatFs works on lower level I/O layer, which reads a data of the physical device and store them to block buffers. The hierarchical structure of FAT file system module is shown in Figure 3.

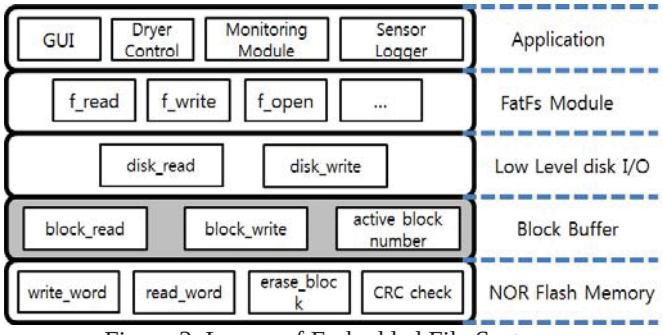


Figure 3. Layers of Embedded File System

In order to solve erase-before-write problem and difference of read/write and erase unit on the NOR flash memory, the block buffer is used. If a command of read/write from FAT is requested, the block buffer loads data from a corresponding block address on the flash memory. Then, the block buffer sends data to FAT as the size of data requested from the FAT. If the block buffer is dirty and a block address requested from FAT and a block address of block buffer is different, the block buffer writes to flash memory for write-back. This algorithm is shown in Figure 4. Algorithm 1 explains read-data from the flash memory and passes data to FAT. Algorithm 2 explains write-data from FAT and passes data to the flash memory through the block buffer.

Algorithm 1: read from NOR flash memory for FAT
Before reading data from FAT: *FAT reads data from buffer* B_r
B: block buffer, N: address of B, R_a: block address of B_r
1 $R_a \leftarrow$ get block address from FAT
2 if $N \neq R_a$ **and Dirty** B **is set then**
// check for buffer change before overwrite block buffer
3 write B to flash memory
4 $B \leftarrow$ read data block of R_a from the flash memory
5 end if
6 $B_r \leftarrow$ copy data from B as size of B_r

Algorithm 2: write to NOR flash memory for FAT
After writing data to FAT: *FAT writes data to buffer* B_w
W_a : block address of B_w
1 $W_a \leftarrow$ get block address to write data from FAT
2 if $N \neq W_a$ **and Dirty** B **is set then**
// check for buffer change before overwrite block buffer
3 write B to flash memory
4 $B \leftarrow$ read data block of W_a from the flash memory
5 end if
6 $B \leftarrow$ copy data from B_w
7 Set a dirty bit of B

Figure 4. Algorithms to Read and Write Data

3.3 Screen update method

The screen resolution is 800x480 pixels. When images are stored on the internal memory, their display speed is fast. But display speed is slow when images are stored on the external flash memory. When an 800x480 image in the external NOR flash is displayed on the screen, it takes about 1.5 seconds on 7" screen.

There are two kinds of screen updates. One is updating full screen, which is 800x480 pixels. The other is partial updates such as date, temperatures, popup windows, and so on. To make update the screen fast and unperceived, it must update the small area only. Instead of updating the full screen,

the section update of the frame buffer is implemented. The section buffer can be divided into subsection areas again and subsection areas are updated specific data only. For example, when the system clock is updated, seconds can be updated in the frame buffer instead of updating hours, minutes and seconds. Figure 5 shows a process displaying time information in seconds.

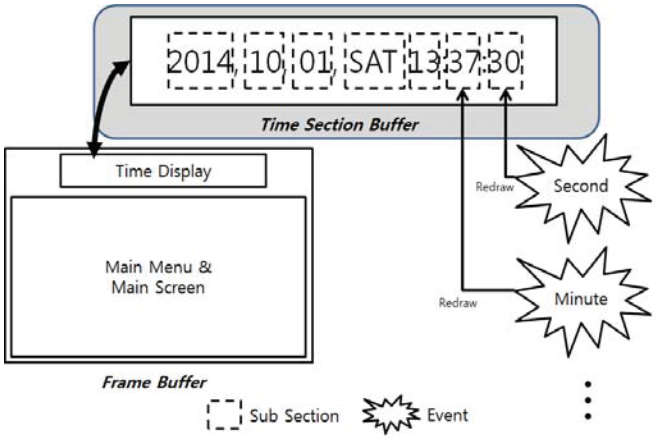


Figure 5. Partial Screen Update

4 Software implementation

The graphics software is implemented on a STM32F429 board with 7" Innolux TFT LCD and touch-screen. An External 64 Mbit NOR flash(IS45S16400J) memory is connected to store images and data storage.

4.1 GUI main screen

Issues in Section 3 are solved and the implemented GUI is shown in Figure 6. Since the LCD controller supports only 16-bit color, 24-bit color has been converted 16-bit color. The background bitmap images are stored in the internal memory after they are scaled down to 1/4 to 1/16. The displayed image is shown in Figure 6.

Figure 6. Main Screen

4.2 Automatic operation

Dryer can be operated by two modes. One is manual operation mode and the other is automatic dryer. Since each vegetable has its own drying time and control, automatic operation is very useful to dry various vegetables. 40 vegetables are programmed for automatic dry cycles. They are programmed for drying temperature, drying time, humidity control and so on.

4.3 Information display

While the dryer is operated, it collects drying data from sensors, which are temperature, humidity, fan speeds, electricity consumption, accumulated solar heat usage, and its operation time. All data are saved in the external NOR flash memory and are displayed information daily, weekly and monthly. Figure 7 shows on statistics screen, which are shown as a bar graph format for seven weeks.

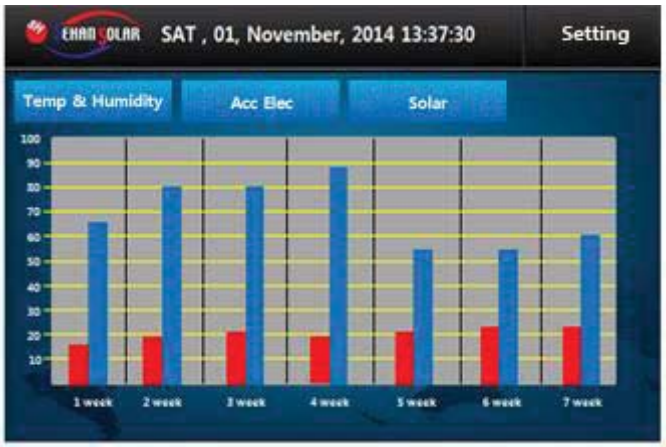


Figure 7. Statistics Display

5 Conclusion

This paper presents an implementation of low-cost embedded graphics system for a vegetable dryer. For graphics implementation, three issues are solved without losing graphics speed with maintaining low cost. Especially, to improve the speed of graphical representation, the scaling method are used and small images are stored in the NOR flash memory. To make fast screen update, two layers of the frame buffer are implemented. In near future, we plan to interface to 10" TFT screen for easy touch and large character display for old famers.

Please address any questions related to this paper to Yeonbo Kim by Email (ybkim@daegu.ac.kr).

6 References

[1] Munir, Anjum, Umair Sultan, and Muhammad Iqbal. "Development and performance evaluation of a locally fabricated portable solar tunnel dryer for drying of fruits, vegetables and medicinal plants." Pak J Agric Sci, Vol. 50, pp. 493-498, 2013.

[2] Chaudhari, Ashish D., and Sanjay P. Salve. "A Review of Solar Dryer Technologies." IJRAT, Vol 2, No. 2, pp. 218-232, Feb., 2014.

[3] Misha, S., et al. "Review on the application of a tray dryer system for agricultural products."World Applied Sciences Journal, Vol. 22, No. 3, pp. 424-433, 2013.

[4] Parai, Manas Kumar, Banasree Das, and Gautam Das. "An Overview of Microcontroller Unit: From Proper Selection to Specific Application." International Journal of Soft Computing, Vol. 2, pp. 228-231, 2013.

[5] Lin, Yuanxin, et al. "Design and Implementation of Remote/Short-range Smart Home Monitoring System Based on ZigBee and STM32." Journal of Applied Sciences, Engineering and Technology, Vol. 5, pp. 2792-2798, 2013.

[6] Xiao, Haihong, and Jiming Luo. "Design of Electrical Parameter Measurement System for Three Phase AC Motor Based on STM32." Sensors & Transducers, Vol. 174, No. 7, pp. 205-210, 2014.

[7] Chung, Tae-Sun, et al. "A survey of flash translation layer." Journal of Systems Architecture, Vol. 55, No. 5, pp. 332-343, 2009.

[8] STM32F437 datasheet, ST Micronics, Jan 2014.

[9] http://elm-chan.org/fsw/ff/00index_e.html

SESSION

HPC + VIRTUAL MACHINES + FAULT TOLERANT DEVICES

Chair(s)

TBA

A Standard Cell Based Power-Delay-Area Efficient 3-of-5 Majority Voter Design

P. Balasubramanian
School of Computer Engineering
Nanyang Technological University
50 Nanyang Avenue
Singapore 639798

H.R. Arabnia
Department of Computer Science
University of Georgia
415 Boyd Building
Georgia 30602-7404, USA

***Abstract*—This paper proposes a new standard cell based design for the 3-of-5 majority voter meant for use in quintuple modular redundant hardware. The voter can tolerate up to two faulty or erroneous inputs, and when a majority of the inputs are correct guarantees the production of the correct output. In comparison with the existing design of the 3-of-5 majority voter, the proposed design reports an increase in the figure-of-merit by 33.7%, where the figure-of-merit is defined as the inverse of the product of power, delay, and area. The results are based upon simulations, performed by targeting a 32/28nm CMOS process.**

Keywords—Majority voter; Standard cells; Digital design; Quintuple modular redundancy; Fault tolerance

I. INTRODUCTION

N-modular redundancy (NMR), where a majority M out of N function modules are expected to operate correctly is a scheme widely employed in the fault-tolerant hardware and software designs of safety-critical circuits and systems [1]. In this paper, we focus our attention on NMR as applied to fault-tolerant hardware design. Among the generic NMR family, triple modular redundancy (TMR), which is a 3-tuple version of the NMR is well-known and widely used for numerous safety-intensive systems applications [2]. However, in mission-critical space and aerospace electronic circuits and systems, besides the TMR [3], quintuple modular redundancy (QMR), which is a 5-tuple version of the NMR, is also preferred [4] [5]. In TMR, 3 copies of the hardware are used and at least 2 out of the 3 hardware units[1] should operate correctly, while in the case of QMR, 5 copies of the hardware are used and at least 3 out of the 5 hardware units should operate correctly. The TMR scheme can cope with any arbitrary faulty hardware unit or hardware unit failure whilst guaranteeing the correct operation and the QMR scheme can tolerate any two faulty hardware units or hardware unit failures which may occur at random whilst providing the correct operation.

The block diagram of a typical NMR system is portrayed by Figure 1. A hardware unit is duplicated (N – 1) times, and the N identical hardware units are combined using a voting element (voter) which produces a majority vote of the correctly functioning hardware as shown in Figure 1. The hardware units' outputs viz. H_1 to H_N are given as inputs to the majority voter whose output V reflects the majority of the input values. Since the TMR system employs 3 hardware units, the majority voter corresponding to the TMR can be called as '2-of-3 majority voter' [6]. Likewise, since the QMR system deploys 5 hardware units, the majority voter corresponding to the QMR can be called as '3-of-5 majority voter'.

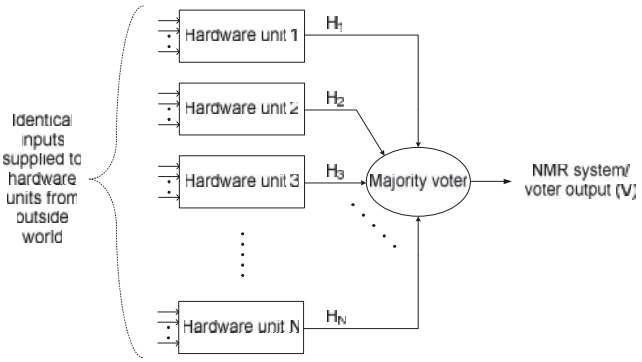


Figure 1: Block schematic of the NMR system

The reliability of the typical NMR system (R_{NMR}) is given by the following binomial expression (1), where K varies from 0 to (N – M), and $\binom{N}{K} = \frac{N!}{(N-K)!K!}$. It is implied in the equation that R = R(*t*), i.e. reliability is a function of time *t*. R_H signifies the reliability of the hardware unit used. Equation (1) specifies that a majority M out of N available hardware units are expected to maintain the correct operation to guarantee the reliable operation of the NMR system, and that the faulty or failure states of up to a maximum of K hardware units can be accommodated by the system.

$$R_{NMR} = \sum_{K=0}^{N-M} \binom{N}{K} R_H^{(N-K)} (1 - R_H)^K \quad (1)$$

It is implicitly assumed in the above expression that the majority voter is perfect since the voter is usually a small piece of hardware compared to the hardware unit used in an NMR system. Moreover, since identical hardware units are used to compose the TMR and QMR systems, the reliabilities of the hardware units are also assumed to be equivalent. Under these

[1] The term 'hardware unit' is generically used in this paper to refer to any circuit or system, which is duplicated as per need to form the NMR system.

assumptions, equation (1) is deduced, and the reliabilities of simplex (non-redundant), TMR and QMR systems are plotted in Figure 2 as a function of their hardware unit reliabilities.

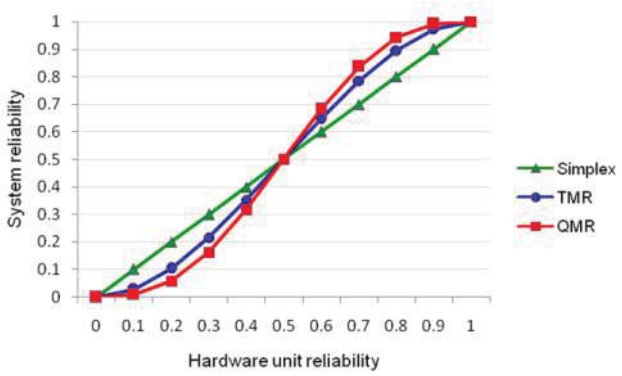


Figure 2: Reliabilities of simplex, TMR and QMR systems versus their hardware reliabilities

As seen in Figure 2, the simplex system reliability varies linearly as a function of its hardware unit reliability since it comprises just a single hardware. Up to $R_H < 0.5$, the simplex system features better reliability than the TMR and QMR systems. However, in reality, R_H tends to be equal to 0.9 or greater. Hence for R_H values ranging from 0.6 to 0.9, the TMR system exhibits improvement in reliability than the simplex system by 10% on average, and the QMR system exhibits an increase in reliability by a further 4.6%. In other words, the QMR system reports a mean enhancement in reliability by 15.1% compared to the simplex system for R_H values in the range of 0.6 to 0.9. Further, the TMR and QMR systems incorporate fault/failure tolerance, which is usually absent in the simplex system and the simplex system might constitute a single point-of-failure [1] during critical fault occurrences. In contrast, the TMR system is able to withstand up to 1 hardware unit fault/failure, and the QMR system is able to cope with double the number of hardware unit faults/failures compared to the TMR.

II. Majority Voters of QMR System

The existing 3-of-5 majority voter design corresponding to the QMR system is first presented, followed by a description of the proposed 3-of-5 majority voter design. The voters are assumed to be perfect in the following discussions.

A. Existing majority voter design for QMR

The existing 3-of-5 majority voter design corresponding to the QMR [7] is shown in Figure 3. Here, A, B, C, D and E represent the five equivalent outputs of five identical hardware units, which serve as the inputs to the majority voter, whose output is indicated as Y. A full adder, a half adder, a 2-input OR gate which combines the sum outputs of the full adder and the half adder (viz. SUM_{FA} and SUM_{HA}), and an AO222 cell, which determines the majority amongst incoming carry outputs of the full adder, the half adder (viz. $COUT_{FA}$ and $COUT_{HA}$), and the OR-ed output of SUM_{FA} and SUM_{HA} are used in this design. It should be noted that the full adder and the half adder are indeed available as elements of a standard cell library [8], and hence they may be treated on par with complex logic gates.

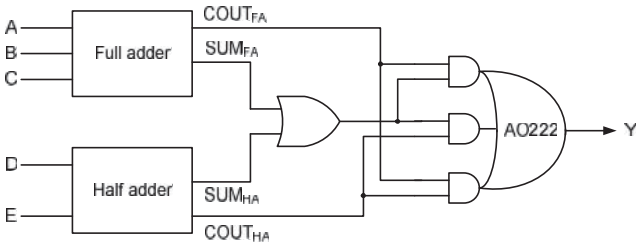


Figure 3: Existing design for the 3-of-5 majority voter

The logical expression of the 3-of-5 majority voter is given below, which highlights all possible majority conditions.

$$Y = ABC + ABD + ABE + ACD + ACE + BCD + BCE + ADE + BDE + CDE + ABCD + ABCE + ABDE + ACDE + BCDE + ABCDE \quad (2)$$

To explain the operation of the 3-of-5 majority voter shown above, let us consider a majority of 1s output by the hardware units for an illustration. Notice that the following illustration would be equally applicable for a consideration of majority of 0s output by the hardware units as well.

When the first majority condition, i.e. ABC becomes true in (2), SUM_{FA} and $COUT_{FA}$ become 1, as a result the OR gate outputs 1, and hence the AO222 gate produces 1 on Y since the majority of its inputs are 1.

When any of the majority conditions from ABD up to BCE in (2) becomes valid, SUM_{FA} would evaluate to 0 but $COUT_{FA}$ would evaluate to 1. Since the other voter input D or E is also 1 for any of these majority conditions at the same time, SUM_{HA} evaluates to 1 and $COUT_{HA}$ becomes 0. Therefore the OR gate outputs 1 since one of its inputs (SUM_{HA}) is 1. This again takes us back to the previous situation where 2 out of 3 inputs to the AO222 gate are 1, and hence the voter output Y equals 1.

When any of the majority conditions specified by ADE up to CDE in (2) becomes valid, $SUM_{HA} = 0$ and $COUT_{HA} = 1$. But since either A or B or C is also 1 simultaneously, SUM_{FA} equates to 1 and $COUT_{FA}$ equates to 0, which causes the OR gate to output 1. Since $COUT_{HA}$ is also 1, the two inputs to the AO222 gate are 1 and therefore it outputs 1 on Y.

If the majority conditions ABCD or ABCE in (2) become valid, SUM_{FA}, $COUT_{FA}$ and SUM_{HA} would evaluate to 1, while $COUT_{HA}$ alone would evaluate to 0. Again, the two inputs to the AO222 gate are 1, and hence it produces the output Y = 1.

For any of the majority conditions ABDE or ACDE or BCDE of (2) becoming valid, $COUT_{FA}$ and $COUT_{HA}$ become equal to 1, while SUM_{FA} and SUM_{HA} would evaluate to 0 and so the OR gate outputs 0. Since $COUT_{FA}$ and $COUT_{HA}$ serve as the inputs to the AO222 gate, it subsequently produces the output of Y = 1.

When the best-case condition i.e. ABCDE becomes valid in (2), $SUM_{FA} = COUT_{FA} = COUT_{HA} = 1$ and $SUM_{HA} = 0$ results.

For this condition, all the inputs to the AO222 gate are 1, and hence Y = 1 results.

B. Proposed majority voter design for QMR

The proposed design of the 3-of-5 majority voter depicted by Figure 4(a) is based on the direct synthesis of (3). Equation (3) is derived from (2) on the basis of the set theory based factoring technique [9] [10]. Equation (3) is synthesized using 5 simple logic gates and 1 complex logic gate: three 3-input OR gates (G1, G2 and G5), two 3-input AND gates (G3 and G4), and one complex gate viz. the OA221 cell, shown in Figure 4(a). The OA221 cell synthesizes (A + B) (D + E) C as a single entity.

$$Z = (C + D + E)\,AB + (A + B)\,(D + E)\,C + (A + B + C)\,DE \quad (3)$$

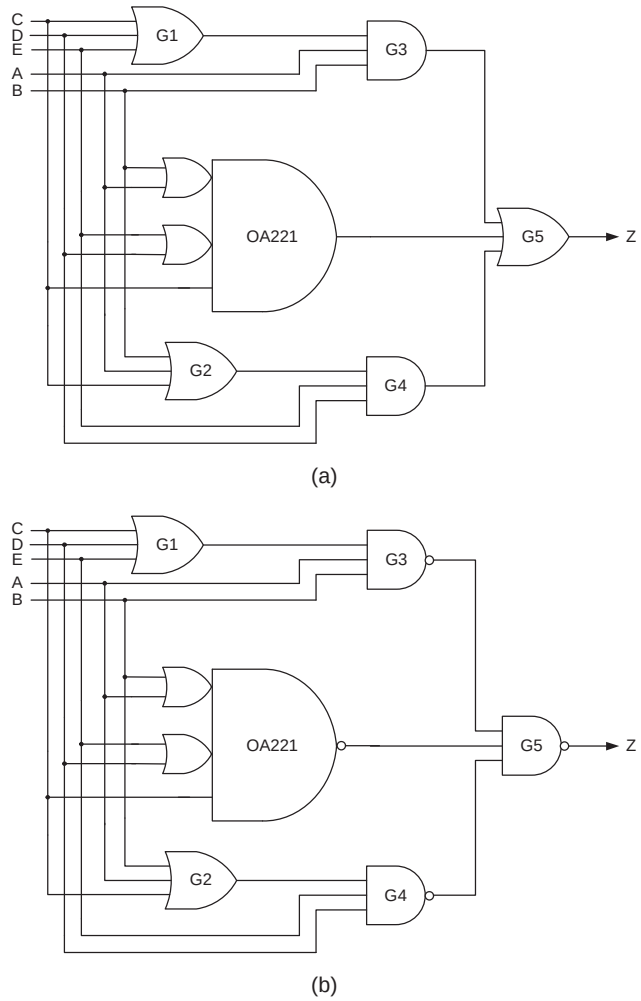


Figure 4: Proposed design of the 3-of-5 majority voter
(a) Basic implementation, (b) Optimized implementation

To understand the operation of the proposed voter design (Figure 4a); consider a majority of 1s output by the hardware units for an illustration similar to that of the previous sub-section. The following deliberations would equally apply for a consideration of majority of 0s output by the hardware units.

When the majority condition (C + D + E) AB given in (3) becomes valid, inputs A and B are 1 and any of the inputs C or D or E is also 1 (at least). Given this, gates G1 and G3 output 1, and because one of the inputs to gate G5 is 1, a value of 1 is produced on the voter output Z.

Alternatively, if the majority condition (A + B) (D + E) C specified in (3) is upheld, the OA221 cell would output 1 and the gate G5 would subsequently produce the output of Z = 1.

Lastly, provided the majority condition (A + B + C) DE of (3) is true, since A or B or C is 1 (at least), gate G2 outputs 1. As the remaining inputs D and E are also 1s simultaneously, gate G4 outputs 1, and gate G5 also produces 1 at its output.

Notice that the above deliberations have in fact considered majority clauses where as a minimum at least 3 out of 5 inputs to the majority voter are 1s, in which case only one input of G5 is 1, which leads to Z = 1. However, if four or all of the voter inputs are 1s, then more than one input to gate G5 becomes 1, which eventually results in the output, Z = 1.

Figure 4(b) represents an optimized version of Figure 4(a), where the AND-OR logic implemented by gates G3, G4, the complex gate OA221, and the OR gate G5 are realized using NAND-NAND logic. Figure 4(b) signifies the proposed 3-of-5 majority voter design which when physically realized leads to optimization of design metrics, as discussed in the following section. As per De Morgan's theorem of Boolean algebra, Figures 4(a) and 4(b) are logically equivalent.

III. Simulation Results and Discussion

Two sample implementations of the QMR system has been considered by treating the 4×4 array multiplier shown below in Figure 5 as the example hardware unit. The multiplier module consists of 8 primary inputs viz. A_3 to A_0 and B_3 to B_0, with A_3 and B_3 being the most significant bits, and A_0 and B_0 are the least significant bits. The 8 primary outputs of the multiplier are represented by the product bits P_7 to P_0, with P_7 being the most significant and P_0 being the least significant bit. The 4×4 array multiplier is realized using a total of 8 full adders and 4 half adders, and the full adder and half adder logic are directly synthesized using the corresponding elements of the digital cell library [8]. For implementing the QMR system, 5 copies of the 4×4 array multiplier are used besides the 8 majority voters.

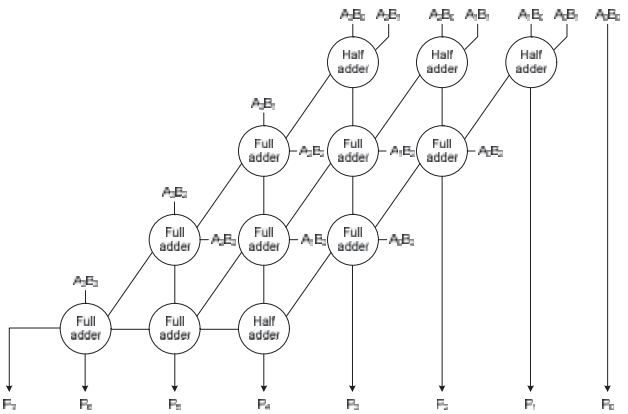


Figure 5: 4×4 Braun array multiplier used as the hardware unit

The simulation results (viz. power, delay, area) obtained for the two QMR implementations incorporating different majority voter designs are shown in Table 1. The structural integrity of the majority voters and the multiplier shown in Figures 3, 4(b) and 5 are preserved while performing the simulations. Given that the 4×4 array multiplier is uniformly employed for the hardware units, the two QMR implementations differ only in terms of the majority voters used with the one embedding the existing 3-of-5 majority voter and the other incorporating the proposed 3-of-5 majority voter. The 4×4 array multiplier consumes 84.38μm^2 of Silicon when realized using a 32/28nm CMOS process [8]. The area occupancies of the existing and proposed 3-of-5 majority voter designs are found to be similar, and are estimated to be 13.47μm^2.

For power estimation, the 4×4 array multiplier used in the QMR systems were supplied with all distinct input patterns viz. 256 inputs, which reflects the unique multiplication scenarios. The input vectors were supplied at time intervals of 2.5ns (400MHz) through test benches which represent the inputs coming in from the outside world. The .vcd files generated for the QMR system implementations, on the basis of the applied input vectors, were subsequently used for power estimation using Synopsys tool. The area and critical path delay metrics were also estimated for the QMR system implementations and are given in Table 1. The primary outputs of the QMR system implementations have fanout-of-4 drive strength.

To holistically comment on the design parameters of the two QMR system implementations, a figure-of-merit (FOM) is defined as the inverse of the product of power, delay, and area. Since minimum values of power, delay, and area metrics are desirable, a lower power-delay-area product and thus a higher FOM are preferable, and either of these could be considered to be an indicator of optimized design.

Table 1: Power, delay, area, and FOM of the two QMR system implementations incorporating different 3-of-5 majority voters

Design metric legend	**QMR system 1 (Existing voter)**	**QMR system 2 (Proposed voter)**
Power (in μW)	133.8	118.5
Delay (in ns)	1.09	0.92
Area (in μm^2)	529.64	529.64
FOM ($\times 10^6$)	12.95	17.32

Notice that since similar hardware units have been used in QMR system 1 and QMR system 2, they are different only in terms of their majority voters. The QMR system 1 employs the existing voter design, while the QMR system 2 incorporates the proposed voter design. Hence the differences between the power, delay, and area results obtained for the two QMR systems can be duly attributed to the differences between the existing and proposed majority voter designs. From Table 1 it is clear that the QMR system 2 employing the proposed 3-of-5 majority voter reports greater FOM by 33.7% compared to the QMR system 1 which embeds the existing voter design.

The propagation delay of the proposed voter is less than the existing voter since the former has one 3-input OR gate and two 3-input NAND gates in its critical path, while the latter features a full adder, a 2-input OR gate and an AO222 gate in its critical path. This explains the reason behind achieving a considerable delay reduction of 15.6% in the case of QMR system 2 compared to the QMR system 1. Since the proposed majority voter occupies the same amount of Silicon as that of the existing voter, when considering a QMR system implementation, this translates into similar area occupancy for both QMR systems 1 and 2. As a consequence, QMR system 2 may be expected to dissipate the same average power as that of QMR system 1, but QMR system 2 is indeed found to dissipate less power than QMR system 1 by 11.4%. Although this may be surprising, the reason for the low power dissipation of QMR system 2 vis-à-vis its counterpart (i.e. QMR system 1) is rather intricate and is explained as follows.

Referring to (2), it can be observed that the percentage of majority conditions which are specified by only 3 inputs out of the 5 voter inputs is found to be 62.5% among the possible majority conditions listed. As a consequence, if only 3 out of 5 inputs applied to the existing majority voter, portrayed by Figure 3 are 1, it will result in the activation of the full adder and/or half adder, the 2-input OR gate, as well as the activation of the final AO222 gate with 2 of its 3 inputs being driven to 1. On the other hand, for a similar consideration of only 3 out of 5 inputs applied to the proposed majority voter (depicted by Figure 4b) are 1, it will cause the activation of either three simple logic gates viz. G1, G3 and G5, or G2, G4 and G5, or the activation of just the complex logic gate (OA221) and G5. Hence, it may be understood that a less switching activity is anticipated for the proposed voter compared to the existing voter. This explains the reason why QMR system 2 is able to achieve less total power dissipation than QMR system 1. Further, this is construed to be the likely reason behind the less peak power dissipation of QMR system 2 by 6.1% in relative comparison with QMR system 1. The peak power dissipation of QMR systems 1 and 2 are estimated to be 16.3mW and 15.3mW respectively through a time-based power analysis.

IV. Conclusion

With multiple faults and failures becoming more prominent in the nanoscale electronics regime [11] – [13], the importance of and the need for QMR as opposed to TMR is expected to increase. In this backdrop, the design of an efficient 3-of-5 majority voter that forms an important constituent of the QMR hardware is very relevant. In this context, a novel power-delay-area efficient 3-of-5 majority voter design has been presented in this paper. In comparison with the existing 3-of-5 voter, the proposed 3-of-5 voter has led to optimization of design metrics (measured in terms of FOM) by 33.7% for a sample QMR system implementation that utilized a 4×4 array multiplier for the hardware units.

References

[1] B.W. Johnson, *Design and Analysis of Fault-tolerant Digital Systems*, Addison-Wesley Publishing, 1989.

[2] I. Koren, C. Mani Krishna, *Fault-Tolerant Systems*, Morgan Kaufmann Publishers, 2007.

[3] R.E. Lyons, W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200-209, April 1962.

[4] J.R. Sklaroff, "Redundancy management technique for space shuttle computers," *IBM Journal of Research and Development*, vol. 20, no. 1, pp. 20-28, January 1976.

[5] M.J. Azbug, E.E. Larrabee, *Airplane Stability and Control: A History of the Technologies That Made Aviation Possible*, 2nd edition, Cambridge University Press, 2002.

[6] P. Balasubramanian, N.E. Mastorakis, "A standard cell based voter for use in TMR implementation," *Proc. 5th European Conference of Circuits Technology and Devices*, pp. 115-124, 2014.

[7] T.J. Dysart, "It's all about the signal routing: Understanding the reliability of QCA circuits and systems," *PhD thesis*, University of Notre Dame, Indiana, USA, pp. 23, July 2009.

[8] Synopsys Digital Standard Cell Library, *SAED_EDK32/28_CORE* Databook, Revision 1.0.0, 2012.

[9] P. Balasubramanian, R. Arisaka, "A set theory based factoring technique and its use for low power logic design," *International Journal of Computer, Control, Quantum and Information Engineering*, vol. 1, no. 3, pp. 702-712, 2007.

[10] P. Balasubramanian, B. Raghavendra, "Analysis of effect of pre-logic factoring on cell based combinatorial logic synthesis," *Proc. World Academy of Science, Engineering and Technology*, vol. 2, pp. 577-582, May 2008.

[11] D. Rossi *et al.*, "Multiple transient faults in logic: An issue for next generation ICs?," *Proc. 20th IEEE DFT Symp*., pp. 352-360, 2005.

[12] H. Quinn *et al.*, "Domain crossing errors: Limitations on single device triple-modular redundancy circuits in Xilinx FPGAs," *IEEE Trans. on Nuclear Science*, vol. 54, no. 6, pp. 2037-2043, December 2007.

[13] N.Miskov-Zivanov, D. Marculescu, "Multiple transient faults in combinational and sequential circuits: A systematic approach," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1614-1627, October 2010.

Prototype of Light-weight Hypervisor for ARM Server Virtualization

Young-Woo Jung, Song-Woo Sok, Gains Zulfa Santoso, Jung-Sub Shin, and Hag-Young Kim
Cloud Computing Research Department, ETRI, Daejeon, Republic of Korea

Abstract - *As ARM CPUs become increasingly common in the server world, virtualization technologies for mobile systems need to be extended for ARM server systems. However, because this system has the limited resources compared to the traditional x86 server system, new virtualization technologies should be considered to allow as many virtual machines as possible to run efficiently and simultaneously on single ARM server system. In this paper, we present the prototype of light-weight hypervisor for ARM server system which can minimize the performance degradation of the guest operating system running on the hypervisor and provide full virtualization. We explore how to achieve the light-weight ARM hypervisor by describing and analyzing its detailed implementation. Through a performance comparison between the native operating system and the guest operating system running on the proposed hypervisor, we show that the proposed ARM hypervisor guarantees minimal virtualization overhead.*

Keywords: ARM server, virtualization, ViMo-S, hypervisor, virtual machine, light-weight

1 Introduction

The number of ARM-based devices has grown tremendously across smart phones, tablets, laptops, and embedded devices. It is because ARM CPUs are more power-efficient than any other CPUs in the market. Nowadays, ARM CPUs also continue to increase performance and some of them is now within the range of x86 CPU performance. This drives the development of ARM-based microservers and pushes ARM CPUs into the traditional server world.

A microserver (also written as micro server or MicroServer) is a small server appliance that Intel introduced the concept around 2010. This inexpensive and energy-efficient server can be squeezed onto a small system board to obtain a blade system which may be smaller than the conventional blade but still powerful enough for data processing. [1] Although Intel has launched microserver products based on Xeon or Atom processors on the market, ARM CPUs have been also considered as another excellent choice, because ARM based SoCs have a better performance to build servers and clusters than x86 and Atom processors, especially considering their performance per Watt relation. [2]

On the other hand, virtualization has been adopted as an important key technology in the x86 server systems for many years and is now spreading to microservers. With ARM beginning to enter the server world, virtualization support is very critical and ARM CPUs of the ARMv7-A [3] and ARMv8-A [4] architectures now include hardware support for virtualization, ARM virtualization extensions, that lets multiple virtualized OSes run efficiently and simultaneously.

The current major hypervisor (also known as virtual machine monitor) technologies using the hardware virtualization extensions of ARM seem to be KVM/ARM [5] and Xen on ARM [6]. However, KVM and Xen was the original purpose of virtualizing x86 server systems, so both basic structures have been optimized in x86 architecture, not in ARM architecture. In the KVM/ARM approach which supports a full virtualization, the host operating system (OS) runs directly on top of the hardware, in which the hypervisor is implemented as a kernel module, and then the guest OSes run as processes on top of the host kernel. Although this kernel component of KVM is included in mainline Linux, KVM/ARM must leverage QEMU [7] in user space to virtualize I/O devices and QEMU is a rather heavy program to be installed in ARM server system which has restricted resources. [8] Xen on ARM has been used as one of leading para virtualization technologies for ARM-based devices. With ARM providing virtualization extensions, Xen on ARM has supported hardware virtual machine (HVM), rather than paravirtual machine (PV). Although Xen is a very mature virtualization technology, it has a very complex configuration which is not easy for common user and it needs to modify the guest OS which means its compatibility and portability is poor.

In this paper, we present the prototype of light-weight hypervisor for ARM server virtualization with ARM virtualization extensions, which support full virtualization and minimize the performance degradation of the guest OSes. On the beginning stage of the design and implementation, we focused only on the ARM architecture and have optimized it.

This paper is organized as follows. Firstly, we give the brief explanation about ARM virtualization extensions, which is main technology to make the hypervisor more efficient and light-weight. The detailed architecture of the proposed hypervisor is introduced in Section 3, where we describe how to virtualize each resource such as CPU, memory, interrupt, and I/O devices. Section 4 briefly shows the experimental results including the performance comparison between the

native OS and the OS running on the proposed hypervisor and the porting to the prototype of ARM server system. Finally, we conclude this paper and suggest some future works.

2 ARM Virtualization Extensions

Similar to x86 architecture, ARM virtualization extensions enable the efficient implementation of the hypervisor for ARM compliant processors to the latest ARMv7-A and ARMv8-A architectures. For example, the ARM Cortex-A15 [9] is a core of ARMv7-A architecture and ARM Cortex-A53/A57 [10, 11] are cores of ARMv8-A architecture, respectively. In this section, we describe a brief overview of ARM virtualization extensions

2.1 New privilege level for hypervisor

As shown in Figure 1, ARMv7-A architecture includes a new CPU mode called Hyp mode as well as TrustZone [12] as Security Extensions. TrustZone splits the modes into two worlds, secure and non-secure. A special mode, Monitor mode, is provided to switch between the secure and non-secure worlds. According to the typical booting sequences in ARMv7-A, ARM CPUs power up by reset starting in ARM secure SVC mode, execute boot and startup codes, and then transition to ARM secure Monitor mode, by which ARM non-secure Hyp mode for the hypervisor can be activated on.

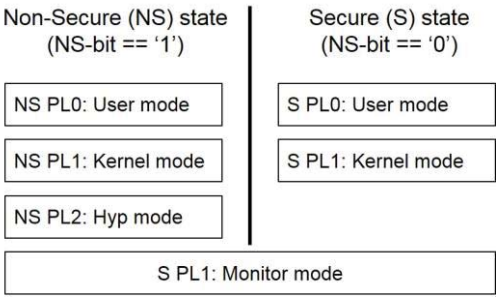


Figure 1. ARMv7-A processor modes

Hyp mode was introduced as trap-and-emulation mechanism to support virtualization in the non-secure world. It is more privileged than the existing non-secure kernel modes, the kernel and user modes, and leaves the guest OSes and applications unmodified. It has its own banked registers, as well as additional registers, such as SP, SPSR, and ELR, in which most of critical feature of hardware-assistant CPU virtualization is executed. Using this register set, the hypervisor software running in Hyp mode can configure hardware to trap into Hyp mode on several sensitive instructions and hardware interrupts.

2.2 Stage-2 translation

Virtualization requires that the guest OS cannot access to the hypervisor's memory space. Without virtualization extensions, a technique, for example, shadow page table which is maintained by the hypervisor, are enforced. In this technique, the guest OS kernel maintains its own page tables but the hypervisor should keep this OS kernel from setting the Memory Management Unit (MMU) registers. This approach makes the hypervisor complicated and causes performance overhead.

In ARM virtualization extensions, ARM provides hardware support to virtualize physical memory, two-stage memory address translation. When a virtual machine (VM) runs, the physical addresses managed by the VM are actually Intermediate Physical Addresses (IPAs) (also known as guest physical addresses) which are translated into physical addresses (PAs) (also known as host physical addresses). For the memory address translation in the guest OS, the stage-1 page tables using the translation table base register (TTBR) translate the virtual addresses (VAs) into IPAs, then stage-2 page tables using the virtual translation table base register (VTTBR) translates IPAs into PAs. This stage-2 translation can be enabled and disabled in Hyp mode.

2.3 Virtual interrupts

ARM defines the Generic Interrupt Controller (GIC) architecture. The GIC routes interrupts from devices to CPUs and CPUs discover the source of an interrupt through the GIC interfaces. The GIC architecture consists of two parts, the distributor and the CPU interfaces. There is only one distributer in a system, and each CPU core has a GIC CPU interface. The distributer is used to configure the GIC, for example, to configure the mapping of an interrupt to the CPU core, and the CPU interfaces are used to signal acknowledgment (ACK) and End-Of-Interrupt (EOI) to the corresponded interrupts.

If all interrupts are configured to be handled by the hypervisor, the hypervisor should generate virtual interrupts in software to signal them to VMs. This causes the interrupt processing in the hypervisor to be expensive, because even ACKs and EOIs for all virtual interrupts must be processed in the hypervisor. The next version of GIC introduced the concept of virtual interrupts which is supported by new hardware virtualization feature, virtual GIC (VGIC), which includes the virtual distributer and the virtual CPU interface. The virtual CPU interface can be mapped into the guest OS as the CPU interface, and can be used by the guest OS to signal ACKs and EOIs without trapping into the hypervisor, reducing overhead for manipulating interrupts on a CPU. The hypervisor generates virtual interrupts by writing to special registers in the virtual distributor, the list registers, and the virtual CPU interface signals these virtual interrupts directly to the guest OS's kernel mode. Nevertheless, the hypervisor must still emulate the distributor and all accesses by a guest OS will be trapped into the hypervisor.

2.4 Generic timer

ARM generic timer architecture provides virtualization support for physical timer resources by introducing virtual timers and virtual counters that measure the passing of virtual time, that is, the passing of time on a particular VM. While the hypervisor is configured to use the physical timer, the VM can be configured to use the virtual timer VMs can control their own virtual timers without any trap to the hypervisor. However, any access to the physical timer and counter by a VM arises trap to Hyp mode, in which the hypervisor only can control them.

3 Light-weight Hypervisor Architecture

The proposed hypervisor, ViMo-S, targets the virtualization of the ARM server system, which means it should be light-weight enough to provide the reasonable performance in the restricted resource environment. Originally, ViMo [13] was implemented by ETRI for mobile ARM processor, which doesn't support hardware virtualization extensions, so we expanded it for ARM server system with virtualization extensions. ViMo-S supports VM lifecycle management such as dynamic creation and destruction of VMs, which may be mandatory in the server virtualization. Another important feature of ViMo-S is to support the full virtualization for which ViMo-S completely virtualizes the physical hardware without any modification of the guest OS codes.

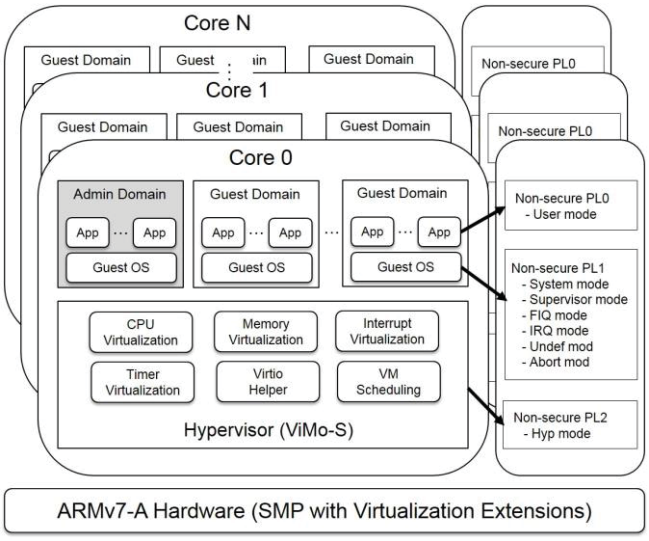


Figure 2. ViMo-S system architecture

As shown in Figure 2, ViMo-S runs in Hyp mode, with supporting the virtualization of CPU, memory, interrupt, and timer resources. It also supports Virtio-based I/O virtualization for full virtualization and can schedule multiple VMs at the same time. There are two kinds of domains running in user mode and kernel mode. While an admin domain knows the existence of the hypervisor and has interfaces to ViMo-S through the hyper calls for VM management and Virtio-based I/O operations, there is no interface to ViMo-S in the guest domains, so that they run in the same manner as the execution on the physical hardware.

In the following sections, we will describe in detail each core virtualization technology of ViMo-S.

3.1 CPU virtualization

To virtualize the CPU, ViMo-S must ensure that the guest OS running in the VM has the same access to the registers as the OS running on the physical CPU, while the hardware state controlled by the hypervisor is persistent across running VMs. With ARM virtualization extensions, a VM running in the kernel and user mode has same register state as register state without the hypervisor, and ViMo-S running in Hyp mode saves/restores the current VM context in/from the Hyp stack when a VM switches to ViMo-S and vice versa. ViMo-S configures all accesses to the other sensitive states such as WFI/WFE instructions, stage-2 page faults, and hyper calls for VM management and I/O virtualization, to be trapped and emulated in ViMo-S. Because trap-and-emulation may be expensive enough to affect VM performance, ViMo-S reduces the frequency of traps by leveraging ARM hardware virtualization support.

ARM boot loaders typically transition to the non-secure world at an early stage, which means there is no ways to switch on Hyp mode in which ViMo-S will execute, because Hyp mode can be activated only in secure Monitor mode. What we need to do is to trap into Hyp mode before uboot boots the guest kernel. In order to turn Hyp mode on, we used secure software, e.g. boot loader, running on secure state in which Monitor mode can activate Hyp mod. When uboot jumps to the entry point of ViMo-S in Monitor mode, ViMo-S performs the following actions in CPU core 0 to enable the hypervisor: (1) enable hyper call and disable secure monitor call, FIQ, IRQ and Abort of Monitor mode, (2) turn Hyp mode on and transition to Hyp mode, (3) activate other cores for SMP, (4) configure the exception vector table in Hyp mode, (5) set up the page table for the hypervisor and enable MMU by setting the page table base register of Hyp mode (HTTBR), (6) activate the hypervisor, (7) configure the distributor register (GICD), the CPU interface register (GICC), and the virtual interface control register (GICH), which are accessible only by the hypervisor, (8) configure the hypervisor timer (also known as Hyp timer) of the generic timer, then the hypervisor can receive the hypervisor timer interrupt, and (9) wait for creating a VM.

For the other CPU cores other than CPU core 0, after transitioning Hyp mode and configuring the exception vector table in Hyp mode, they wait for the event indicating that the page table setup for the hypervisor is complete in the CPU core 0, because all CPU cores share the page table for the hypervisor which is created by CPU core 0. And then, they

perform the same actions as in the CPU core 0, only except for configuration of GICD, because there is only one distributor in a system.

When ViMo-S creates a new VM by the request of the admin domain through the hyper call, it performs the following actions: (1) create and initialize the structure of the VM, which contains virtual CPU (VCPU) context, Hyp stack, MMU context including VTTBR, VGIC, virtual timer, vector floating point (VFP), and other necessary values for the VM, (2) allocate memory to the VM by unit of 2MB and configure the page table including I/O memory mapping for stage-2 translation, (4) activate the VM which is now schedulable. When the VM is scheduled, ViMo-S configures VTTBR and VTCR to enable stage-2 translation for the VM, and returns to the VM through ERET instruction, which performs the mode change to SVC mode and the program counter (PC) change at the same time.

After the VM is created, it runs in PL0 and PL1, and is trapped into ViMo-S only for the timer interrupts, the I/O interrupts, and the specified hyper calls by a VM of the admin domain.

3.2 Memory virtualization

ViMo-S supports memory virtualization of stage-2 translation in order that a VM cannot access physical memory belonging to ViMo-S or other VMs. ViMo-S controls all physical memory accesses and allows a VM only to access the memory regions allocated to it. If a VM tries to access the other memory regions, it causes stage-2 page fault and traps into ViMo-S. Actually we use this kind of stage-2 page fault mechanism for Virtio-based I/O virtualization, which will be explained in section 3.4. Since stage-2 page tables can be configured only in Hyp mode, they are completely transparent to each VM. When ViMo-S performs context-switching to the VM, it enables stage-2 translation and configures the stage-2 page table base register, VTTBR, of the VM. On the other hand, when switching back to ViMo-S, ViMo-S disables stage-2 translation and translates VA directly into PA by using HTTBR. After configuring stage-2 translation and the page table base registers, all memory translations are performed by the hardware without any intervention by ViMo-S, which gives better performance to VMs.

3.3 Interrupt virtualization

ViMo-S configures the CPU to trap all hardware interrupts to Hyp mode by setting GICD register, which enables the hypervisor to control hardware resources. While Hyp timer interrupt is processed only in ViMo-S for VM scheduling and maintenance, VMs must receive notification for other interrupts in the form of virtual interrupts for emulating devices. ViMo-S uses the VGIC to inject these virtual interrupts to VMs and reduce the number of traps to Hyp mode. Virtual interrupts are raised to VCPUs by configuring the list registers of the virtual distributors through GICH and VMs can access to the virtual CPU interfaces without being trapped to Hyp mode.

ViMo-S minimizes the execution of the interrupt hander in Hyp mode, because long execution in the hypervisor can affect the performance of the VM. For Hyp timer interrupt, it saves only the banked registers into the Hyp stack, executes its own jobs, and then transition to the VM. In the case of VM context switching, it additionally save the other contexts in the structure of the VM, such as MMU context, VGIC, virtual timer, and so on, which was explained in section 3.1. For the other interrupts, ViMo-S just injects the corresponded virtual interrupts to the VM using VGIC.

3.4 Virtio-based I/O virtualization

For the support of full virtualization, ViMo-S utilizes Virtio [14], the de-facto standard for I/O virtualization, to provide virtual devices in the guest domain. As described in Figure 3, we provide three key components for Virtio-based I/O virtualization; Virtio front-end, Virtio back-end, and Virtio helper.

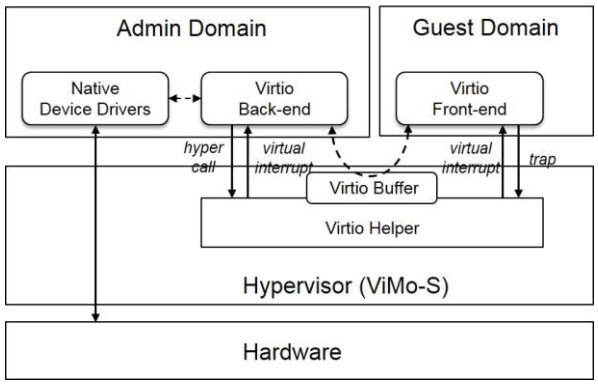


Figure 3. Virtio-based I/O virtualization

The Virtio front-end is located in the guest domain and considered as a normal device driver. There is no distinguishable difference between the Virtio front-end device drivers and other physical device drivers. Although the implementation of the Virtio front-end driver depends on the running OS of the guest domain, these drivers have been already included in Windows or Linux, and we just used it in the guest domain without any changes. That means ViMo-S can support the full virtualization without changing the codes of the guest OS. Through the configuration of the Virtio front-end driver, the access to the memory area for the transport via the memory-mapped I/O arises trap into ViMo-S, which can emulate I/O operations.

To serve I/O requests of the Virtio front-end, the Virtio back-end driver is necessary in the admin domain. The Virtio back-end driver leverages the already implemented virtualization programs in the OS running on the admin

domain. As an example, in Linux, the Virtio back-end driver uses TUN/TAP device to virtualize network devices for the guest domains. The Virtio back-end doesn't depend on the guest OS. A Virtio back-end driver can virtualize devices for multiple guest domains. The actions that can be done by the Virtio back-end may be limited due to the reasons of security and stability. For instance, the Virtio back-end cannot directly access to the memory area of the guest domain for reading and writing data. It means a module running on the hypervisor should support such operations. ViMo-S provides the Virtio helper which takes requests from the Virtio back-end as well as the Virtio front-end and processes them as necessary.

During the guest domain boots, the Virtio front-end asks the Virtio helper for the type, features and status of the Virtio device. After the Virtio helper gets the information from the Virtio back-end and sends it to the Virtio front-end, the Virtio front-end initializes the device accordingly. The initialization includes the buffer creation. The information about the size and location of the buffer is transferred to the Virtio helper which then map this buffer to the memory region in the admin domain, where the Virtio back-end driver can process the data directly in this buffer. By this way of conduct, we can be sure that all the virtual devices in the guest domain work under the capabilities of the admin domain's devices. As an example, the virtual block device in the guest domain can support SCSI device, as long as the admin domain supports it. Otherwise, the SCSI support in the guest domain is turned off.

For the Virtio operations, the Virtio front-end writes data into its buffer and the Virtio back-end reads the buffer and acts properly according to the request and the type of the device. As for the detailed explanation of the Virtio operations, please refer to [14]. There are several mechanisms to interact between each component for Virtio operations. The Virtio front-end interfaces with the Virtio helper through memory trapping. The access to the memory-mapped I/O region traps into ViMo-S where the Virtio helper handles it. As for the Virtio back-end, it interfaces with the Virtio helper through the hypervisor calls. The Virtio helper kicks both the Virtio front-end and the Virtio back-end by injecting virtual interrupts. The Virtio front-end and back-end drivers should register handlers to manage virtual interrupts sent by the Virtio helper. While the virtual interrupts injected to the Virtio front-end depend on which virtual devices are used, we use the virtual interrupt number 155 to be injected to the Virtio back-end, because 155 is not used by ARM v7-A architecture.

3.5 VM scheduling

Currently, ViMo-S provides the simple Round-Robin (RR) VM scheduler, which maintains its own VM queue on a CPU core. When a VM is created, it allocates a time quantum and the VM scheduler makes context-switching according to the specified time quantum. In order to schedule VMs in ViMo-S, ViMo-S always uses Hyp timer of ARM generic timer. In each core, the VM scheduler decreases the time quantum of a running VM whenever it receives the Hyp timer interrupt, and makes the context-switching of VMs when the remaining time quantum of the running VM is 0.

When the current VM traps into ViMo-S by the Hyp timer interrupt, the Hyp timer interrupt handler in ViMo-S saves the current VM contexts, such as all general purpose registers, R13 register of USR mode, and the banked registers of SVC, ABT, UND, IRQ, and FIQ mode, into the Hyp stack. In case of VM switching, ViMo-S additionally saves the MMU control registers including VTTBR, VGIC-related registers, virtual timer control registers, VFP registers, and fault state registers, into the data structure of the current VM. And, then it restores the saved registers and values from the Hyp stack and the saved data structure of the next VM, and traps into the next VM.

4 Experimental Results

In this section, we present some experimental results that can measure the performance of ViMo-S on ARM multicore hardware, and show how many VMs can be provided in an ARM server system. We evaluated the virtualization overhead of ViMo-S compared to native execution by running the AIM benchmark suite [15] within both a VM and directly on the hardware. The results provide the real measurements of the performance of ViMo-S with ARM hardware virtualization support. Moreover, we show the construction of a 32-bit ARM server system to run many VMs simultaneously on top of ViMo-S.

4.1 Methodology and measurement

For ViMo-S measurement, we used an Insignal Arndale board [16] with a dual core 1.7GHz Cortex-A15 CPU on a Samsung Exynos 5250 SoC, which has been the most widely used and commercially available development board supporting ARM virtualization extensions and multicore. It supports onboard 100Mb Ethernet, 2Gbyte memory, eMMC 4.5, SDIO 3.0, and SD 2.0.

We used the mainline Linux 3.8 kernel for our experiments, with several patches on top of the source tree. Although an OS running on ViMo-S was slightly modified to provide the hyper calls, we kept the software environments of both platforms as the same as possible to provide comparable measurements. Our focus was not on measuring absolute performance of ViMo-S, but rather the relative performance degradation between virtualized and native execution of OS. As the AIM benchmark suite, we used Re-AIM7 [17] open source software which is a rework of the AIM benchmark suite for the Linux community. Although it benchmarks several workloads such as CPU, disk, file server, database, and so on, we focused only on the measurement for CPU/disk-intensive workloads, because these workloads may be the important criteria to evaluate the virtualization environment.

As shown in Figure 4, two Arndale boards are connected to the desktop, using the serial ports. While one is for evaluating the performance of the native OS, the other one is for evaluating the performance of the guest OS running on ViMo-S. We continually executed the Re-AIM7 benchmark programs in both OSes by configuring the *-g* flag in order to increase the number of users up to 10. For each number of users, the Re-AIM7 runs until the maximum Job/Minute (JPM) is reached. For the CPU and disk performance, we repeated this benchmark in both OSes up to 10 times, and produced the average of the results.

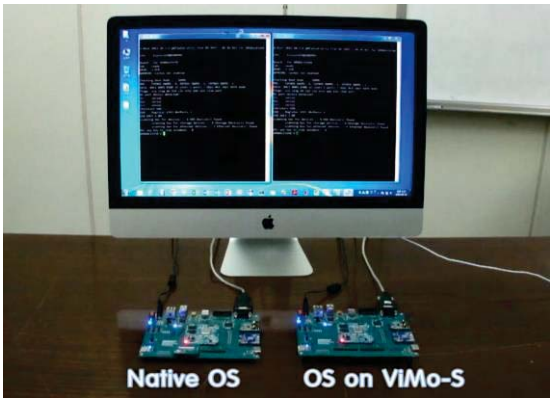


Figure 4. Test environment: native OS vs. OS on ViMo-S

For CPU-intensive workloads and disk-intensive workload, Figure 5 and 6 show respectively normalized performance for running application workloads in the VM versus running directly on multicore. The horizontal axis is essentially the number of simultaneous jobs (workloads), and vertical axis is the overall rate at which the jobs complete (throughput). Figure 5 shows that ViMo-S has minimum virtualization overhead across CPU-intensive workloads, despite the performance degradation of the maximum of 4.5% and the average of 2%. Although Figure 6 shows the substantial differences in virtualization overhead compared to CPU-intensive workload, the overhead by ViMo-S is less than 4%. In this evaluation, we found that ARM virtualization extensions significantly reduce complexity of ViMo-S and are also likely to reduce virtualization overhead.

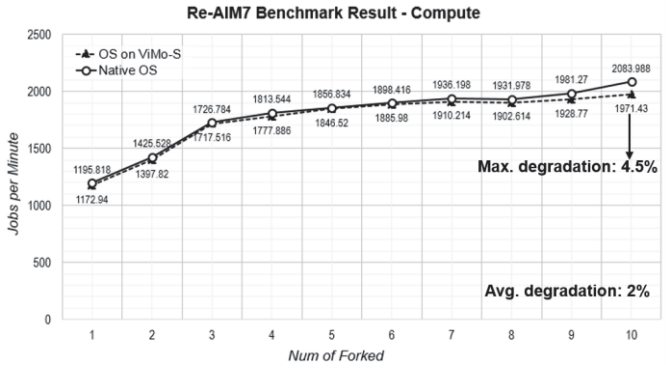


Figure 5. Re-AIM7 benchmark result - compute

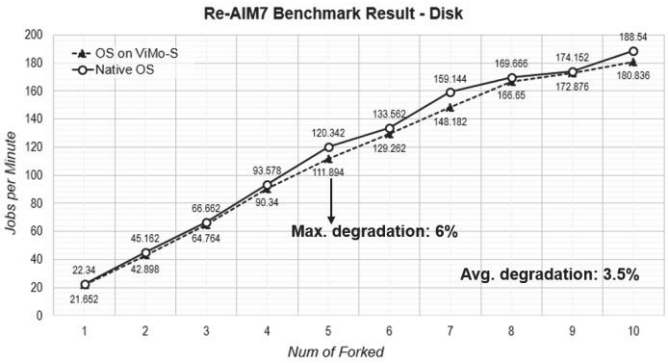


Figure 6. Re-AIM7 benchmark result - disk

4.2 Prototype of ARM server system

To evaluate ViMo-S as ARM server virtualization, we developed the reference platform of 32-bit ARM server system as shown in Figure 7, which consists of eight computing nodes, HDD pool, and power supplier. The system configuration may be very simple, but it is sufficient, because each ARM computing board provides CPU, memory, Gigabit Ethernet, USB 3.0, and so on. A computing board supports the two CPU sockets based on Samsung Exynos 5250 SoC, which is same as the CPU of the Insignal Arndale board.

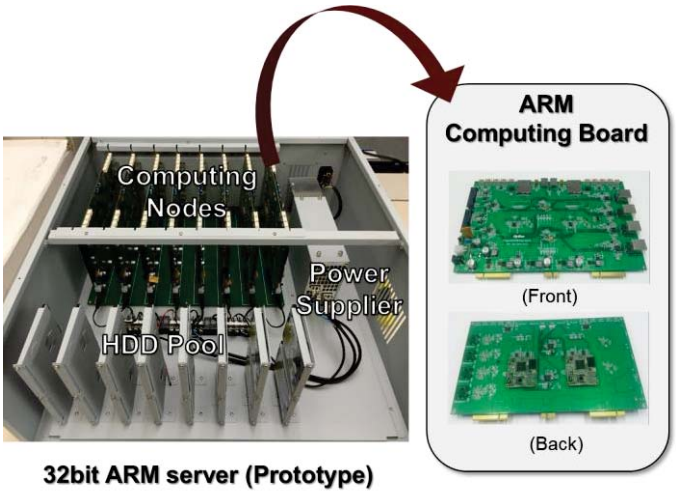


Figure 7. Reference platform of 32-bit ARM server system

In this single system, we generated and ran 64 VMs simultaneously, including the admin domains and the guest domains. The normal execution of each VM can be verified through the remote SSH connection to the SSH daemon running on each VM.

5 Conclusion and Future Works

In this paper, we presented the prototype of the light-weight hypervisor, ViMo-S, which targets the virtualization of the ARM server system. With the benefits from ARM hardware virtualization extensions, ViMo-S can minimize the

virtualization overhead on ARM multicore hardware. We also presented in detail how to use ARM hardware virtualization extensions to virtualize the hardware resources, such as CPU, memory, and interrupt. ViMo-S supports the full virtualization by using Virtio, the de-facto standard for I/O virtualization. Our experimental results show that ViMo-S incurs minimal performance impact and has modest virtualization overhead, within 4% of direct native execution on multicore hardware for CPU-intensive and disk-intensive workloads.

For the future works, we need to improve the performance of the Virtio-based I/O virtualization, because all I/O requests from the guest domains can be centralized into the admin domain. And then, we will expand ViMo-S to 64-bt ARM server system, for which we consider the X-Gene [18] development board with an octa core 2.4 GHz Cortex-A5x CPU on an Applied Micro APM883208 SoC, and the development of the reference platform for 64-bit ARM server.

6 Acknowledgements

This work was supported by the ICT R&D program of MSIP/IITP [R0101-15-237, Development of General-Purpose OS and Virtualization Technology to Reduce 30% of Energy for High-density Servers based on Low-power Processors].

7 References

[1] Hsieh, Wen-Hsu, et al. "Energy-saving cloud computing platform based on micro-embedded system." *2014 16th International Conference on Advanced Communication Technology (ICACT)*, 2014.

[2] Aroca, Rafael Vidal, and Luiz Marcos Garcia Gonçalves. "Towards green data centers: A comparison of x86 and ARM architectures power efficiency." *Journal of Parallel and Distributed Computing* 72.12 (2012): 1770-1780.

[3] ARM Ltd. ARM Architecture Reference Manual ARMv7-A DDI0406C.b, July 2012.

[4] ARM Ltd. ARM Architecture Reference Manual ARMv8-A DDI0487A.a, Sept. 2013.

[5] Dall, Christoffer, and Jason Nieh. "KVM/ARM: The design and implementation of the linux arm hypervisor." *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 2014.

[6] Xenproject.org. ARM Hypervisor - Xen. http://www.xenproject.org/developers/teams/arm-hypervisor.html.

[7] Bellard, Fabrice. "QEMU open source processor emulator." *URL: http://www. qemu. org* (2007).

[8] Minnich, Ronald G., and Don W. Rudish. "Ten Million and One Penguins, or, Lessons Learned from booting millions of virtual machines on HPC systems." *Workshop on System-level Virtualization for High Performance Computing in conjunction with EuroSys*. Vol. 10. 2009.

[9] ARM Ltd. ARM Cortex-A15 Technical Reference Manual ARM DDI 0438C, Sept. 2011.

[10] ARM Ltd. ARM Cortex-A53 MPCore Processor Technical Reference Manual ARM DDI 0500D, Feb. 2014.

[11] ARM Ltd. ARM Cortex-A57 MPCore Processor Technical Reference Manual ARM DDI 0488C, Dec. 2013.

[12] Alves, Tiago, and Don Felton. "TrustZone: Integrated hardware and software security." *ARM white paper* 3.4 (2004): 18-24.

[13] Oh, Soo-Cheol, et al. "ViMo (virtualization for mobile): a virtual machine monitor supporting full virtualization for ARM mobile systems." *CLOUD COMPUTING 2010, The First International Conference on Cloud Computing, GRIDs, and Virtualization*. 2010.

[14] Russell, Rusty. "virtio: towards a de-facto standard for virtual I/O devices." *ACM SIGOPS Operating Systems Review* 42.5 (2008): 95-103.

[15] Wikipedia website. http://en.wikipedia.org/wiki/AIM_Multiuser_Benchmark.

[16] InSignal Co. ArndaleBoard.org. http://www.arndaleboard.org.

[17] Sourceforge.net. Re-AIM_7. http://re-aim-7.sourceforge.net.

[18] Applied Micro Circuits Co. X-Gene. https://www.apm.com/products/data-center/x-gene-family/x-gene/

Mapping the Conjugate Gradient Algorithm onto High Performance Heterogeneous Computers

Anas M. Alfarra
Electrical & Computer Engineering Dept.
Jackson State University
Jackson, MS
alfarra.anas@gmail.com

Jamory D. Hawkins
Electrical & Computer Engineering Dept.
Jackson State University
Jackson, MS
jamoryhawkins@gmail.com

Gerald R. Morris
Information Technology Lab.
US Army Engineer
Research and Development Center
Vicksburg, MS
gerald.r.morris@us.army.mil

Khalid H. Abed
Electrical & Computer Engineering Dept.
Jackson State University
Jackson, MS
khalid.h.abed@jsums.edu

Abstract

Mapping scientific kernels onto high performance heterogeneous computers (HPHCs) must comply with certain rules of thumb or heuristics. Previous research by Jackson State University's (JSU) HPHC research group has provided anecdotal evidence illustrating some of these rules/heuristics. The research highlighted by this paper corroborates the credibility of these rules. In particular, four versions (two pairs) of a floating-point sparse matrix conjugate gradient (CG) iterative solver are presented. JSU's state-of-the-art HPHC utilizes general purpose processors (GPPs) and heterogeneous computational hardware, in particular, a field programmable gate array (FPGA), to develop the CG kernels. The first version of the pair executes strictly on the GPP and the second uses both the GPP and FPGA to map the entire CG algorithm onto hardware. For the second pair, a refactored version of CG is used, which is statically analyzed to determine where the most computationally expensive operation occurs. This operation is the sparse matrix vector multiply (MVM) kernel. Based on this analysis, the software version of CG is refactored to call MVM as a subroutine. An FPGA version of the MVM algorithm is also developed and a static analysis of that algorithm suggests a speedup of the MVM kernel. All four version of CG are executed using a specially designed set of sparse matrices and the results demonstrate that adherence to the rules of thumb and heuristics when mapping scientific kernels onto HPHC can lead to significant speedups.

1. Introduction

HPHCs, such as field programmable gate array (FPGA)-augmented reconfigurable computers (RC), can sometimes outperform their general purpose processor (GPP)-based counterparts. In the past, lack of support for floating-point arithmetic within FPGA tool suites often forced designers to use fixed-point or integer arithmetic [1]. Now, semiconductor technology scaling allows companies such as Altera to fit floating-point intellectual property (IP) cores onto contemporary FPGAs. As a result, there have been some successes at mapping iterative solvers onto FPGAs [2, 3, 4]. For floating-point applications, FPGA-based processors must satisfy several heuristics and rules of thumb to achieve a speedup compared with their GPP counterparts. This paper highlights the challenges in the computational mapping process while simultaneously showing that such mappings can result in significant speedups. The focus is to show the importance of "the three P's," which expresses the crucial relationship among performance, pipelining, and parallelism as well as several of the other heuristics[5]. This paper is organized as follows: Section 2 provides background on HPHC design considerations. Section 3 introduces the conjugate gradient method and a high-level design of the solver. Section 4 details the FPGA-based solver design. Section 5 describes the HPHC hardware and details the implementation. Section 6 describes the experiments, compares the runtime performance of the two conjugate gradient versions, and provides an analysis of the results. Section 7 presents the conclusions.

2. HPHC Design Considerations

The JSU HPHC research group has developed a set of heuristics to determine if a given algorithm is suitable for mapping onto HPHCs [6, 7]. These include a) the three p's, b) resource utilization, c) control and memory intensive vs. compute intensive, d) monolithicity of the algorithm, e) available bandwidth, f) ability to reuse data, g) design stability of the algorithm, h) efficiency of the algorithm, and i) memory access patterns. As an example, a common approach to estimate sp eedup is via Amdahl's Law shown in Equation 1,

$$s_o = \frac{1}{1 - f_e + f_e/s_e}, \tag{1}$$

Based on earlier work, a conservative value is $s_e \approx 10$ for the CG algorithm. A profile of the software version of CG showed matrix vector multiply (MVM) consumed nearly 67 percent of the runtime. By Amdahl's Law, an overall speedup $s_o = 1/(0.33 + 0.67/10) = 2.5$ is anticipated.

3. Conjugate Gradient

3.1. CG Algorithm

Discussions of the CG iterative method can be found in many introductory numerical analysis textbooks including [8, 9]. CG is typically used when A is a sparse matrix. Sparse matrices are usually represented in a compressed format, which only stores the non-zero elements and provides some bookkeeping mechanism for determining the row and column number of each matrix entry. A representation of the conjugate gradient algorithm (as implemented in software) is shown in Figure 1 [10].

3.2. CG Operation

An arbitrary starting point $\mathbf{x}_0$ is selected, from which the descent proceeds. The first A-orthogonal vector, $\mathbf{p}_1$, is a search direction opposite to the gradient as depicted in line 2, therefore, the initial search direction is also the same as the initial residual as illustrated on line 3. The next line is a criteria for forcing the algorithm to enter the loop. Line 5 is part of the equation for calculating the residual. Rather than continually computing this value, it is simply calculated one time. At line 6, the body of the loop requires the dot product of the current residual, $\mathbf{r}_k$, and the previous residual, $\mathbf{r}_{k-1}$. Instead of calculating two dot products within the loop, the dot product from the previous iteration was reused, i.e., $\mathbf{r}_0^T\mathbf{r}_0$. Line 7 creates the iteration index, k, which is also part of the criteria for the while loop to prevent infinite looping. It can be shown that the matrix vector product is needed twice within CG, however, it is computed once on line 9 because it is an expensive $\mathbf{O}(n^2)$ algorithm. The next line computes the step size, α_k, along the direction of the

```
1:  algorithm CGSW(A, x, b)
2:      p_1 ← b − A x_0
3:      r_0 ← p_1
4:      Δ ← ε + 1
5:      over b_norm ← 1/‖b‖
6:      rTr_old ← r_0^T r_0
7:      k ← 1
8:      while (Δ > ε) .AND. (k < k_max) do
9:          v_ap ← A p_k
10:         α_k ← rTr_old / p_k^T v_ap
11:         x_k ← x_{k−1} + α_k p_k
12:         r_k ← r_{k−1} − α_k v_ap
13:         rTr_new ← r_k^T r_k
14:         β_k ← rTr_new / rTr_old
15:         rTr_old ← rTr_new
16:         p_{k+1} ← r_k + β_k p_k
17:         Δ ← ‖r_k‖ · over b_norm
18:         k++
19:     end while
20:     return (x_{k−1})
21: end algorithm
```

Figure 1. Software CG algorithm

CG. For line 11, the new approximation for $\mathbf{x}$ is calculated by descending in the conjugate search direction a distance of step size. Line 12 computes the new residual. Then the dot product of the new residual, rTr_{new} is calculated, which will be used in subsequent computations. Line 14 computes the projection operator β which removes from the residual all previous search directions. Since the dot product of the previous residual is no longer needed and to prevent calculating the dot product at the next iteration, line 15 retains the current residual for the next iteration. The new search direction can finally be calculated, by using the projection operator to remove the residual from all components along the previous conjugate search directions, i.e., $\mathbf{r}_k + \beta_k\mathbf{p}_k$. Line 17 computes the residual norm to check the algorithm for convergence. Line 18 increments the iteration index. If the algorithm has converged, the solution, $\mathbf{x}_{k-1}$, is returned.

4. CG Processor Detailed Designs

4.1. Two CG Versions

As mentioned above, two versions of CG were implemented in hardware. The first (monolithic) version offloads the entire CSR-based CG algorithm onto the FPGA without regard to the heuristics that previous research has shown to significantly impact performance. The second version offloads only the MVM kernel (which comprised some 67% of the software run time). In the algorithms below, compressed sparse row (CSR) format is used.

4.2. CG Hardware Design

A monolithic hardware version of CG, which was mapped onto the FPGA hardware, is shown below. The monolithic CG algorithm that was executed in software is essentially the algorithm depicted in Figure 1.

```
1:  algorithm CGHW(kval, kcol, kptr, b, b2, n, knz, x)
2:      parBegin  // only two GCM banks so
3:          BUF_DMA_GCM1:OBM (kval, stripe-8)
4:          BUF_DMA_GCM2:OBM (kcol, stripe-8)
5:      parEnd
6:      parBegin  // parallel DMA limited to 2
7:          STREAM_DMA_GCM1:BRAM (kptr)
8:          STREAM_DMA_GCM2:BRAM (b)
9:      parEnd
10:     for i in [1,n] do   // x^(0) = 0 .·. p^(1) ← r^(0) ← b
11:         x_i ← 0
12:         p_i ← r_i ← b_i
13:         MAC(b_i, b_i, rTr_old)   // calculate r^(0)T r^(0)
14:     end for
15:     δ ← 0
16:     repeat
17:         p1 ← ... ← p11 ← p
18:         parBegin  // compute v ← A p^(δ)
              p1:  // dot8's into V_FIFO stream
19:           for i in [1,knz] do
20:               a8 ← (a_1 ··· a_8) stripe-8 from kval_i
21:               j8 ← (j_1 ··· j_8) stripe-8 from kcol_i
22:               p8 ← (p1_j1 ··· p8_j8)
23:               V_FIFO ← a8^T p8
24:           end for
              p2:  // # dot8's per row into C_FIFO stream
25:           for i in [1,n] do
26:               C_FIFO ← kptr_i+1 − kptr_i
27:           end for
              p3:  // n dot products into S_FIFO
28:           S_FIFO ← Σ_STREAM(V_FIFO, C_FIFO)
              p4:  // dot_i = a_i^T p^(δ)
29:           for i in [1,n] do
30:               v_i ← S_FIFO
31:               MAC(v_i, p9_i, pTv)   // calculate p^(δ)T v
32:           end for
33:     parEnd
34:     α ← rTr_old / pTv   // step size
35:     for i in [1,n] do
36:         x_i ← x_i + α p10_i   // next point: x^(δ)
37:         r_i ← r_i − α v_i   // residual: r^(δ)
38:         MAC(r_i, r_i, rTr_new)   // calculate r^(δ)T r^(δ)
39:     end for
40:     β ← rTr_new / rTr_old   // projection operator
41:     rTr_old ← rTr_new
42:     for i in [1,n] do   // new search direction: p^(k+1)
43:         p_i ← r_i + β p11_i
44:     end for
45:     r2b2 ← √rTr_old · b2   // ||r^(δ)|| = √rTr_old
46:     δ ← δ + 1
47:   until (r2b2 ≤ ε .OR. δ > δ_max)
48:   BUF_DMA_BRAM:GCM2 (x)
49: end algorithm
```

Figure 2. Monolithic hardware CG algorithm

4.3. MVM Hardware Design

Because of the significant speed degradation of using multiple serialized multiply-accumulators (MACs) within the hardware version of CG after the parallel sections finish, a CSR-based MVM kernel was mapped onto the HPHC. All other modules of CG are executed via the GPP. The principle speedup is obtained via the four parallel sections $\mathbf{p}_1$ through $\mathbf{p}_4$, which communicate via a set of FIFO streams. This hardware MVM algorithm is idealized in Figure 3 and operates in three phases: *input*, *compute*, and *output*. During

```
1:  algorithm MVMHW(kval, kcol, kptr, v, p, *first)
2:      if (*first) then
3:          parBegin  // only two GCM banks so
4:              BUF_DMA_GCM1:OBM (kval, stripe-8)
5:              BUF_DMA_GCM2:OBM (kcol, stripe-8)
6:          parEnd
7:          parBegin  // parallel DMA limited to 2
8:              STREAM_DMA_GCM1:BRAM (kptr)
9:              STREAM_DMA_GCM2:BRAM (p)
10:         parEnd
11:     else
12:         STREAM_DMA_GCM2:BRAM (p)
13:     end if
14:     p1 ← ... ← p8 ← p
15:     parBegin
            p1:  // feed values stream
16:         for i in [1,kn_z] do
17:             a ← (a1 ··· a8) stripe-8 from kval_i
18:             j ← (j1 ··· j8) stripe-8 from kcol_i
19:             p ← (p1_j1 ··· p8_j8)
20:             V_FIFO ← dot8Tree (a, p)
21:         end for
            p2:  // feed counts stream
22:         for i in [1,n] do
23:             C_FIFO ← kptr_i+1 − kptr_i
24:         end for
            p3:  // streaming accumulator
25:         S_FIFO ← Σ_STREAM(V_FIFO, C_FIFO)
            p4:  // the dotN's
26:         for i in [1,n] do
27:             v_i ← S_FIFO
28:         end for
29:     parEnd
30:     STREAM_DMA_BRAM:GCM2 (v)
31: end algorithm
```

Figure 3. Sparse MVM Hardware Algorithm

input, two parallel blocks use direct memory access (DMA) to input the problem data from global common memory (GCM). Lines 3–10 constitute the *compute* phase. Since the FPGAs do not have multiport memory to support eight address and data buses on a single memory bank, and since parallel sections $\mathbf{p}_1 \ldots \mathbf{p}_3$ operate simultaneously, independent banks are needed to avoid a multicycle pipeline. Therefore, line 14 creates eight copies of $\mathbf{p}$ for the dot product tree.

Parallel section $\mathbf{p}_1$ (lines 15–21) is a fully pipelined $8 \cdot 8$ dot product unit. Each clock cycle it consumes the next eight a_{ij} values from **kval** and the matching eight values from **p** and outputs the resulting partial dot products (dot8s) to the V_{FIFO} stream. Parallel section $\mathbf{p}_2$ (lines 22–24) calculates the number of dot8s for each row and sends them to the C_{FIFO} stream. Parallel section $\mathbf{p}_3$ (line 25) is the streaming accumulator that consumes the V_{FIFO} and C_{FIFO} streams, computes the n dot products, $\text{dotN}_i = \sum_j a_{ij} p_j$ for all i, and feeds the results into the S_{FIFO} stream. Parallel section $\mathbf{p}_4$ (lines 26–29) consumes the dotNs from S_{FIFO} and streams the resultant vector into v_i. During *output*, the resultant **v** is then DMAed to GCM as shown on line 30.

5. Implementation

5.1. High-level CG Design

The high-level design for the CG solver is shown in Figure 4. It consists of four major components: a main routine and matrix support libraries; several symmetric positive definite sparse matrices, $A_1 \ldots A_m$; the software or hardware (FPGA-based) CG solver; and the output result and statistics files, $\mathbf{x}_1 \ldots \mathbf{x}_m$ and $\theta_1 \ldots \theta_m$. The $\mathbf{b}_i$ vectors are shown as inputs, but for the experiments in this research they are generated from a known **x** vector at runtime. The

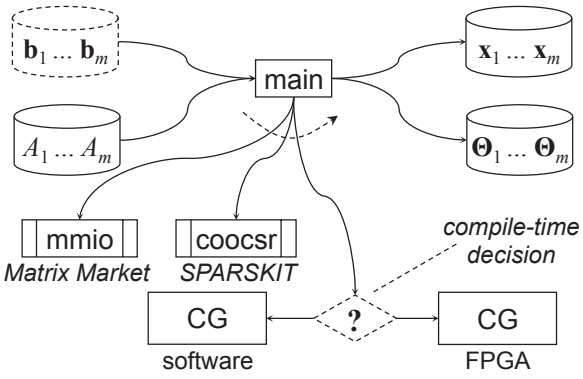


Figure 4. High-level CG Design

main routine is a driver program, which essentially measures how long it takes for CG to solve each set of equations. The coordinate-format matrices are read in using the Matrix Market I/O library [11] and converted to CSR format using Saad's SPARSKIT library [12]. The software CG kernel implementation is based on the algorithm shown in Figure 1, and the FPGA-based CG kernel is based on the algorithm shown in Figure 2.

A compile-time decision selects either the software or FPGA-based version of CG. At runtime, main reads in each coordinate-format matrix, converts it to CSR format, and uses a known **x** vector to generate **b**. It then invokes the selected CG kernel sending matrix, A (**val**, **col**, and **ptr**), starting point $\mathbf{x}^{(0)}$, and constant vector **b**. After convergence, CG stores the result and returns. The main routine writes the solution to the results file; it also writes the input matrix name, number of iterations, and wall clock execution time to the statistics file and then terminates.

5.2. High-level MVM Design

To recap, there were four CG algorithms that were mapped onto the HPHC. The first pair was a monolithic version which comprised of a software and hardware version. The second pair also had a software and hardware version, however, this kernel involved a subroutine call that implemented MVM in software or hardware as illustrated in Figure 5. Macroscopically, the operation is similar to the operation of the monolithic CG described in the previous subsection.

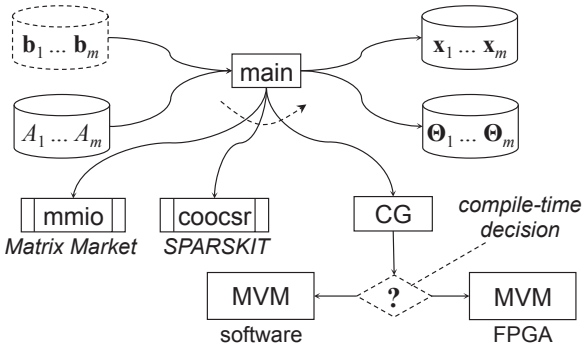


Figure 5. High-level MVM Design

5.3. Implementation Summary

The software and hardware CG designs described in Section 5 were coded for the SRC-7 platform. The same set of files were used in both implementations to ensure a valid side-by-side comparison. The only difference was the CG kernel. Most of the work involved implementing the hardware modules. The $k = 8$ dot product width was limited by the available configurable logic on the FPGA. Thus, only eight values per clock cycle are read from **kval** and **kcol**. Since the **kval** and **kcol** CSR arrays are placed in OBMs and each OBM holds about half a million entries, the largest number of nonzero values that can be processed is $nz_{max} \approx 4\text{M}$. Lastly, as most of the BRAM stores the multiple copies of **x** needed to fully pipeline the loop, the largest matrix was limited to $n_{max} = 8,192$.

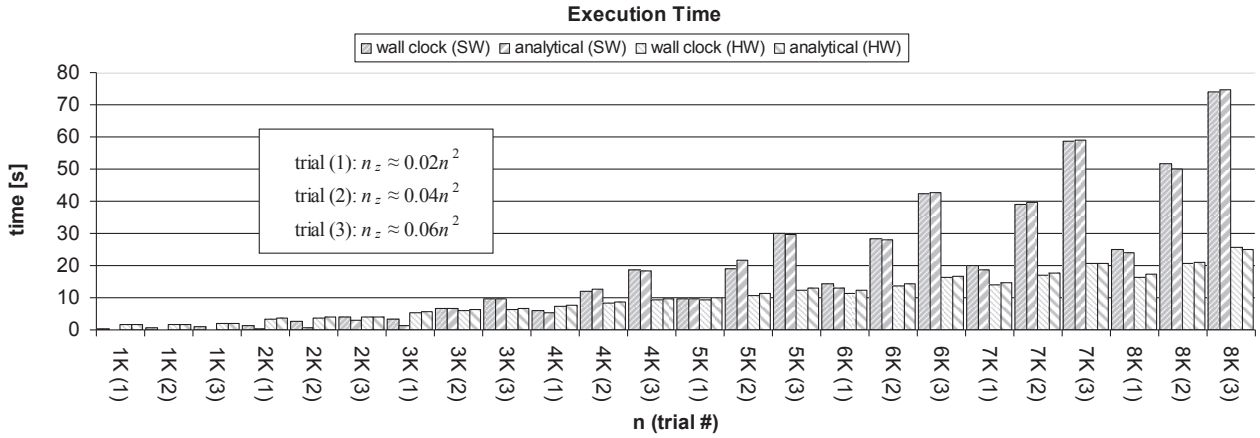


Figure 6. Representative Run Time Comparison

6. Results

6.1. UFL Sparse Matrix Collection

The matrices used to test CG came from the University of Florida Sparse Matrix Collection, managed by Dr. Tim Davis and Dr. Yifan Hu. This repository is a large and actively growing set of sparse matrices widely used by the numerical linear algebra community for the development and performance evaluation of sparse matrix algorithms [13] such as conjugate gradient. The collection covers a vast spectrum of domains ranging from mathematics and physics, to civil engineering and computer science. With over 2547 matrices, UFL's Sparse Matrix Collection boasts its largest matrix as having a dimension of 118 million with almost 2 billion nonzero entries.

6.2. Monolithic Results

As shown in table 1, the monolithic approach, which ignored the heuristics developed by the JSU HPHC research group, did not result in a speedup. In fact, it resulted in a slowdown as depicted by the average of six samples of matrices.

Table 1. Slowdown Using Monolithic Method

Matrix Name	Size (NZ)	t_{sw} (μs)	t_{hw} (μs)	Slowdown $\left(\frac{t_{hw}}{t_{sw}}\right)$
fass1.128.mtx	438	66381	258797	3.89
fass2.500.mtx	2298	55247	435985	7.89
fass1.1024.mtx	4918	396983	286116	0.72
fass1.2048.mtx	10038	74304	300623	4
fass4.4096.mtx	26270	398214	3759980	9.4
fass1.8192.mtx	40758	150452	490218	3.25
			Average	4.85

6.3. MVM-only Results

In contrast, following the heuristics discussed in Section 2, it is possible to achieve a speedup. At the time we were preparing this paper, our target platform was undergoing upgrades at the vendor's site so we do not yet have the results of the refactored CG hardware mapping. However, previous research mapping a Jacobi iterative solver via an analogous set of scenarios [5] resulted in a 3-fold increase in performance as illustrated in Figure 6. Given the slight increase in algorithmic complexity of CG versus Jacobi, the 3-fold speedup seen for Jacobi is in excellent agreement with the anticipated 2.5-fold speedup described in Section 2.

7. Conclusion

The research thus far has suggested strong evidence of an overall speed up of conjugate gradient (CG) when the rules and heuristics developed by the Jackson State University's HPHC research group are adhered to. The negative impact of performance has also been illustrated when these rules and heuristics are ignored [14]. Finding the "sweet spot" when mapping scientific kernels onto HPHCs has indeed helped in demonstrating the value of using the heuristics developed by the JSU HPHC group during their the extensive research within the HPHC domain [15, 16, 17, 18]. Because our target platform was being upgraded from ARO funds when this paper was drafted, we used the analogous set of results from earlier research [5] as shown in Figure 6. We plan to confirm our results at the earliest opportunity.

Acknowledgments

This work was supported in part by the Army Research Office grant number W911NF-07-1-0527, and in part by the U.S. Army Engineer Research and Development Center.

References

[1] Michael Parker. Taking advantage of advances in FPGA floating-point IP cores, October 2009.

[2] Antonio Roldao and George A. Constatinides. A high throughput FPGA-based floating point conjugate gradient implementation for dense matrices. *ACM Transactions on Reconfigurable Technology and Systems*, 3(1):1–19, 2010.

[3] Gerald R. Morris and Viktor K. Prasanna. A hybrid approach for accelerating a sparse matrix Jacobi solver using an FPGA-augmented reconfigurable computer. In *Proceedings of the 9th Military and Aerospace Programmable Logic Devices Conference*, Washington, DC, September 2006.

[4] Gerald R. Morris and Viktor K. Prasanna. Sparse matrix computations on reconfigurable hardware. *Computer*, 40(3):58–64, March 2007.

[5] Gerald R. Morris and Khalid H. Abed. Mapping a Jacobi iterative solver onto a high performance heterogeneous computer. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):85 – 91, January 2013.

[6] Justin L. Rice, Khalid H. Abed, and Gerald R. Morris. Design heuristics for mapping floating-point scientific computational kernels onto high performance reconfigurable computers. *Journal of Computers*, 4(6):542–553, June 2009.

[7] Khalid H. Abed and Gerald R. Morris. Improving performance of codes with large/irregular stride memory access patterns via high performance reconfigurable computers. In *Proceedings of the High Performance Computing Modernization Program Users Group Conference 2009*, pages 422–429, San Diego, CA, June 2009.

[8] David M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, 1971.

[9] Richard S. Varga. *Matrix Iterative Analysis, Second Edition*. Springer, 2009.

[10] Gerald R. Morris. Conjugate gradient. Class notes for CPE 505 "Analysis of Algorithms" course., August 2013.

[11] NIST. Matrix Market. `math.nist.gov/MatrixMarket`, June 2004.

[12] Yousef Saad. SPARSKIT: A basic tool-kit for sparse matrix computations (version 2). `www-users.cs.umn.edu/~saad/software/SPARSKIT`, 2009.

[13] Tim Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1 – 25, November 2011.

[14] Jamory D. Hawkins. *Mapping The Conjugate Gradient Algorithm Onto High Performance Heterogeneous Computers*. MSCE thesis, Jackson State University, Jackson, MS, May 2014.

[15] Justin L. Rice, Khalid H. Abed, and Gerald R. Morris. Design heuristics for mapping floating-point scientific computational kernels onto high performance reconfigurable computers. *JCP*, 4(6):542–553, 2009.

[16] Justin L. Rice, Kevin C. Pace, Miguel D. Gates, Gerald R. Morris, and Khalid H. Abed. High performance reconfigurable computer application design considerations. In *Proceedings of the IEEE Southeast Conference 2008*, pages 236–243, Huntsville, AL, April 2008.

[17] Gerald R. Morris, Antoinette R. Silas, and Khalid H. Abed. Analytical and measured bandwidth for an FPGA-based processor. In *Proceedings of the IEEE SoutheastCon 2012 (SoutheastCon'12)*, pages 1 – 7, Orlando, FL, March 2012.

[18] Gerald R. Morris. *Mapping Sparse Matrix Scientific Applications onto FPGA-Augmented Reconfigurable Supercomputers*. Ph.D.E.E. dissertation, University of Southern California, Los Angeles, CA, December 2006.

UDP/IP Protocol Stack with PCIe Interface on FPGA

Burak Batmaz and Atakan Doğan
Department of Electrical-Electronics Engineering, Anadolu University, Eskişehir, Turkey

Abstract – *Network packet processing in high data rates has become a problem especially for the processors. This work offers a solution to this problem by implementing a hardware-accelerated UDP/IP protocol stack on FPGA. Packets are processed by a UDP/IP hardware on FPGA and UDP/IP communicates over PCIe interface with the related application running on PC. Consequently, a processor core only deals with the data processing, while the proposed UDP/IP hardware on FPGA takes care of the packet processing. The design and implementation of UDP/IP stack are verified on Xilinx XUVP5-LX110T board. Test results and area utilization of our UDP/IP stack are presented as well.*

Keywords: FPGA, UDP/IP, PCIe, Network Protocols

1 Introduction

Nowadays many applications need high speed data transfers. Encapsulation and decapsulation (packet processing) processes for the high speed data transfers, on the other hand, require considerable computing power. Consequently, a processor (CPU) typically has to split its computing power between the packet and data processing tasks, which in turn adversely affects its useful data processing performance. Fortunately, the computing power of CPU spared for the packet processing can be saved provided that the packet processing tasks are offloaded to such a system that can perform these tasks purely on hardware. Motivated by this fact, in the present study, UDP/IP network protocol stack is implemented completely on a FPGA hardware and its error-free operation in terms of offloading UDP/IP functions for data communication is proven by means of Xilinx XUVP5-LX110T board.

In the literature, there are several examples of design and implementation of UDP/IP and TCP/IP protocol stacks on hardware. Löfgren et al. [1] presented three IP cores as minimum, medium, and advanced. The minimum IP core is similar to ours. However, this study offers better throughput and bigger maximum packet size than the minimum IP core. Alachiotis et al. [2] presented a UDP/IP core design in order to provide PC-FPGA communication over Ethernet. The main difference between this work and ours is that PCIe interface is used for PC-FPGA communication. Alachiotis et al. [3] proposed an extended version of their previous work. This extended version has a better performance, but takes 56% more area. Herrmann et al. [4] proposed a similar study with 1960 Mbps full duplex throughput and area usage near to that of other works. Dollas et al. [5] presented one of the most comprehensive work in the literature which implements TCP, UDP, ARP, ICMP, and IP protocols. However, our UDP/IP design offers better throughput values than theirs. Vishwanath et al. [6] proposed the most similar work to ours in terms of the design goals. They implemented a UDP/IP offload engine based on a commercial TCP/IP Offload Engine and this work has 35% better performance as compared to a host-based UDP/IP stack.

The remainder of this paper is organized as follows: In Section 2, background information about the network protocol stack is given. In Section 3, we present our system design. Section 4 shows our UDP and IP hardware components. The experimental results are in Section 5. Finally, we present a conclusion in Section 6.

2 Network protocol stack

Network protocol stack is composed of several layers, while each layer has different responsibilities and functionalities. These layers are elaborated by OSI (Open System Interconnection) and TCP/IP reference models in [7].

2.1 Physical layer

Physical layer is the bottom layer of OSI reference model and every network device has this layer. Messages reach this layer as electrical signals, and then they are converted to data bits and delivered to upper layers, or vice versa.

2.2 Data link layer

Data Link layer is the second layer of OSI reference model, and it is composed of two sublayers, namely Media Access Control and Logical Link Control.

2.2.1 Media access control

Media Access Control (MAC) sublayer is placed between Physical layer and Logical Link Control sublayer. This layer is primarily responsible for

providing a data communication channel among network nodes that share a common medium. In order to avoid collisions in the shared channel, MAC sublayer runs a medium access control algorithm such as CSMA/CD (Carrier Sense Multiple Access with Collision Detection). In addition, MAC sublayer deals with framing as well. Before sending a packet, MAC layer appends a preamble, MAC source and destination addresses, etc. at the start of packet and a CRC data (cyclic redundancy check) at the end of packet. While receiving packets, CRC is calculated for any received packet and packet errors will be determined.

2.2.2 Logical link control

Logical Link Control (LLC) sublayer is a bridge between the network layer and Media Access Control sublayer. LLC adds two bytes to the head of any packet received from network layer to specify the packet type (IP, ARP). These two bytes are known as LLC header. For the packets that are received from MAC sublayer, this part is controlled and they are delivered to the appropriate protocol.

2.3 Network layer

Network layer is the third layer of OSI reference model. Forwarding, routing and logical addressing are the main duties of this layer. The most commonly used network protocol is IPv4 [8], and also preferred in this study. Among its tasks are fragmentation of those packets bigger than the maximum transmission unit (MTU) and defragmentation of the received fragmented packets. It should be noted here that fragmentation/defragmentation is not supported by our hardware-based IP layer. Thus, if a fragmented packet is received, it will be dropped.

2.4 Transport layer

Transport Layer is the fourth layer of OSI reference model. Multiplexing/demultiplexing, end-to-end reliable packet transmission, end-to-end flow control, and end-to-end congestion control are the main functions of this layer. Mostly used transport layer protocols are Transmission Control Protocol (TCP) [9] and User Datagram Protocol (UDP) [10]. TCP is a reliable and connection oriented protocol. UDP is an unreliable protocol and does not guarantee that packets will be delivered to their destination network nodes. Applications that require low latency packet transmission such as Domain Name System (DNS), Voice over IP (VoIP) use UDP protocol. In our work, UDP protocol is preferred as the transport layer protocol.

3 System design

The system consist of three main parts: Xillybus core for PC-FPGA communication over PCIe interface, UDP/IP Sender and Receiver in order to account for the transport and network layers, and Xilinx Ethernet MAC (EMAC) [11] core for sending and receiving packets over Ethernet.

Xillybus [12] uses Xilinx PCIe interface core for the communication over a PCIe interface. Connected to Xillybus IP core are two FIFO buffers. Xillybus core writes any incoming data from PC to Application Send FIFO and reads data from Application Receive FIFO and sends them to PC.

UDP-IP Sender and UDP-IP Receiver components are the cores that have been designed and implemented in this study. Basically, UDP-IP Sender reads data from Application Send FIFO, encapsulates them, and delivers them to EMAC core. UDP-IP Receiver accepts incoming packets from EMAC core, drops or accepts packets, writes any accepted packet into Application Receive FIFO.

The third part of the system is Xilinx Ethernet MAC core. This core basically takes care of the functions briefly mentioned in Section 2.2.1.

Xillybus, Application Send FIFO, Application Receive FIFO, UDP-IP Sender and UDP-IP Receiver, and Ethernet MAC core components are all implemented on a FPGA. Physical layer for Ethernet communication, on the other hand, is realized by another chip on XUPV5 board.

3.1 Xillybus

Xillybus is a DMA based solution for the data transport between PC-FGPA over PCIe interface. On the PC side, Xillybus has a driver that works with device files. A user can write to or read from these device files with simple functions like write() and read(). On the FPGA side, there are two FIFOs connected to Xillybus core, where one for PC-to-FPGA data transfers (Application Send FIFO) and the other one for FPGA-to-PC data transfers (Application Receive FIFO). Data written to the sender device file are send to Application Send FIFO and data written to Application Receive FIFO are copied to the receiver device file. Thus, Xillybus provides an easy-to-use interface to the application logic over FIFO buffers.

Xillybus core realized on Virtex-5 FPGA works with a 100 MHz clock, so the maximum theoretical achievable throughput is 800 Mbit/s with an 8-bit sender device file. However, since Xillybus does not guarantee

a continuous stream of data, the maximum practical achievable throughput falls to nearly 600 Mbit/s. In order to achieve better throughput figures, in this study, a 32-bit writer device file and 32-bit reader device file are used.

As explained above, Xillybus IP core works with a 100 MHz clock signal, and its write and read interfaces are chosen to be 32-bit. However, UDP/IP Sender, UDP/IP Receiver, and EMAC cores require a 125 MHz clock signal, and they have 8-bit data I/O interfaces. Consequently, Xillybus IP core and UDP/IP components are interfaced over Application Send FIFO and Application Receive FIFO, where they are generated through Xilinx Core Generator. The write interface of Application Send FIFO is 32-bit and runs at 100 MHz, and its read interface is 8-bit and runs at 125 MHz. As far as Application Receive FIFO is concerned, it is the other way around as compared to Application Send FIFO.

3.2 Xilinx Tri-Mode Ethernet MAC

Xilinx EMAC core provides data communication over Ethernet and realizes some of data link layer operations. EMAC core runs at 125 MHz clock, and provides an 8-bit application logic data interface by means of its TxFIFO, which receives packets from IP Sender to send them over Ethernet, and RxFIFO, which provides the received Ethernet packets with IP Receiver.

EMAC communicates with PHY chip over a GMII (Gigabit Media Independent Interface) interface. When EMAC core receives a packet, it checks the CRC of packet. If the packet is not corrupted, the core will deliver it to UDP/IP Receiver. Otherwise, this packet is dropped. The core, however, does not check for MAC addresses.

3.3 Sender- and Receiver-Application

Sender- and Receiver-Application run on a PC. Sender-Application is basically a file transfer application, which is develop to test whole system. Specifically, Sender-Application first opens a text file to be sent for reading and 32-bit sender device file in binary mode for writing. Then, in each iteration, it performs a 1472 byte read from the text file and writes into the device file until it reaches the end of text file.

Receiver-Application opens a 32-bit receiver device file and allocates a related memory space. The data received by Receiver-Application are initially written to this memory space. When the receiver device file is closed or the received data reach a predetermined size, Receiver-Application opens a text file, and dumps the data in memory into this text file.

4 UDP/IP hardware

UDP and IPv4 protocols are implemented for the transport layer and network layer in two separate components, respectively.

4.1 UDP component

In order to provide the same functionality as a software-based UDP layer, UDP Sender and UDP Receiver hardware components, whose designs are given in detail below, are developed.

4.1.1 UDP Sender

UDP header structure consist of four fields: Source Port, Destination Port, Length, and Checksum. UDP header together with Application Data form a UDP segment. In our UDP Sender, Source Port is simply hardcoded in FPGA; Destination Port is supplied by Sender-Application; Length Field is calculated for each UDP segment on the fly; Checksum field is not supported and filled with all zeros.

UDP Sender component design is based on a FSMD (finite state machine with datapath) and two FIFOs. Incoming data from Application Send FIFO are written into the first FIFO, where the data consist of Destination Port, Data Length, and Application Data. FSMD controller repeats the following steps in an unending loop: (i) It reads Destination Port from FIFO and saves it to a datapath register. (ii) It fetches Data Length, calculates Length by adding UDP header size (8 bytes) to Data Length, and saves it to another register. (iii) Since UDP header now becomes ready, it starts writing UDP header into the second FIFO. (iv) After writing the header is finished, it lets the first FIFO to write Application Data into the second FIFO, which will complete the processing of a UDP segment by the transport layer.

In order to provide a pipelined system operation, the second FIFO immediately starts forwarding a UDP segment to the FIFO of IP Sender component as soon as it receives the first byte of UDP segment.

4.1.2 UDP Receiver

UDP Receiver, which is architecturally different from UDP Sender, is basically a FSM, and it does not have any FIFOs. When UDP Receiver receives a UDP segment, it checks its destination port field. If the received destination port field is equal to the built-in source port field, UDP segment is sent to Application Receive FIFO. Otherwise, it will drop the segment.

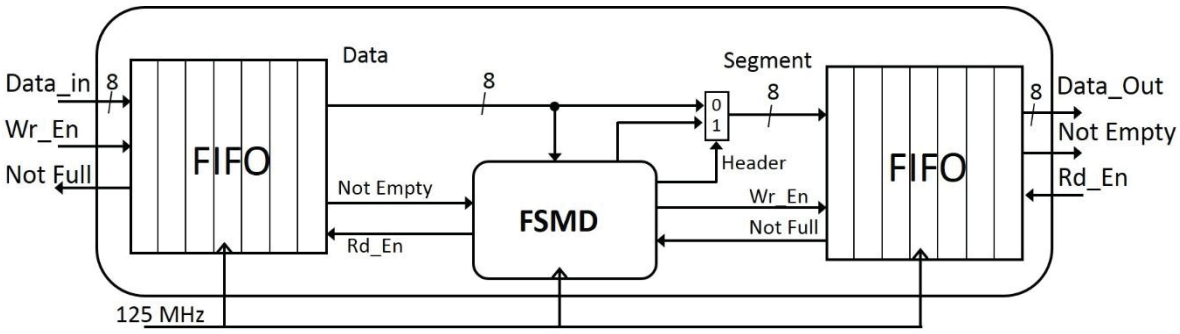


Figure 1. UDP Sender Component Structure

4.2 IP component

Similar to the design of UDP component, IP component is composed of two subcomponents, namely IP Sender and IP Receiver, whose design details are described below.

4.2.1 IP Sender

IPv4 is the de-facto standard network layer protocol. IPv4 Header Structure has many fields. Note that IP header together with UDP segment forms an IP packet. In IP Sender, Version, Header Length (IHL), Time to Live (TTL), Protocol, Source (IP) Address, and Destination (IP) Address fields are hardcoded in FPGA; Type of Service (TOS) and Identification fields are filled with zeros; Fragmentation is not supported, so Flags and Fragment Offset fields are also filled with zeros; Total Length and Header Checksum fields are calculated on the fly.

IP Sender is similar to UDP Sender in terms of its architecture, which is based on a FSMD and two FIFOs. Incoming UDP segment (UDP header and Application Data) from UDP Sender is written into its first FIFO. Then, its FSMD controller repeats the following steps in an unending loop. (i) It writes the built-in Destination MAC Address, Source MAC Address, and Packet Type (set to IP packet) into the second FIFO so that EMAC core can later use these information to form an Ethernet frame accordingly. Meanwhile, Total Length (UDP segment length plus twenty) and then Header Checksum for the IP header are computed. (ii) It writes IP header into the second FIFO. (iii) After writing the header is finished, it lets the first FIFO to write UDP segment into the second FIFO which will complete the processing of an IP packet by the network layer.

In order to provide a pipelined system operation, the second FIFO immediately starts forwarding any IP packet to TxFIFO of EMAC core as soon as it receives the first byte of IP packet. In addition to forwarding IP packets over an 8-bit data interface, the second FIFO provides start of frame signal with the first byte and an end of frame signal with the last byte of every IP packet with Tx FIFO for one clock cycle.

4.2.2 IP Receiver

IP Receiver does not include any FIFOs and consists of only an FSM. When IP Receiver receives an IP packet, it checks Total Length, Fragmentation Flags, Protocol, and Destination Address fields. If Total Length is bigger than 1500 bytes, or Fragmentation Flags indicate a fragmented packet, or Protocol is not UDP, or Destination Address is different from our IP address, IP Receiver drops such a packet. Otherwise, the UDP segment encapsulated by this packet is delivered to UDP Receiver. Note that IP Receiver does not check Header Checksum.

5 Experimental results

UDP/IP core is verified by sending and receiving files with different sizes as follows.

Verification of send functionality: XUPV5 board is installed into 1xPCIe slot of our source PC on which Sender-Application runs, and another PC is employed as the destination so as to run Receiver-Application in the same subnetwork. Then, Sender-Application is used to send files of different sizes up to 250 MBytes. On the destination, Receiver-Application writes the received packets into a file and compares against the original one in order to see if the send function of our UDP/IP system works correctly.

Verification of receive functionality: For these tests, Sender-Application running on a PC tries to transfer different files to another PC with Receiver-Application and XUPV5 board installed.

We repeat each of these send and receive verification tests as many as 100 times with different

files. We have observed that our UDP/IP architecture have successfully sent and received our test files. During these tests, the average throughput of 540 Mbit/s has been achieved.

Table 1. Resource utilization and maximum speed of the proposed UDP/IP system

Component	Occupied Slices	BRAMs	Fmax (MHz)
Xillybus	2742	12	159,9
UDP/IP Sender-Receiver	420	6	244,1
EMAC	200	2	266,8

Table 1 represents the resource utilization and maximum achievable clock frequency in Xilinx Virtex-5 LX110T-1. Overall design can work with a clock of 159,9 MHz; but, a 125 MHz clock is enough for the gigabit operation.

Since our UDP/IP design is based on FIFOs, our FSM controllers in the related send and receive components occupy a really small area. According to Table 1, Xillybus core with PCIe endpoint block plus [13] occupies the greatest area, followed by EMAC core.

6 Conclusions

In this study, we present a gigabit speed UDP/IP stack with PCIe interface implemented on FPGA. Future work will include implementing ARP, ICMP and DHCP protocols in order to complement our UDP/IP core. In addition, supporting multiple UDP streams on our UDP/IP core will be considered.

7 References

[1] A. Lofgren, L. Lodesten, S. Sjoholm, and H. Hansson, "An analysis of FPGA-based UDP/IP stack parallelism for embedded Ethernet connectivity," *Norchip 2005, Proceedings,* pp. 94-97, 2005.

[2] N. Alachiotis, S. A. Berger, and A. Stamatakis, "Efficient pc-fpga communication over gigabit ethernet," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, 2010, pp. 1727-1734.

[3] N. Alachiotis, S. A. Berger, and A. Stamatakis, "A Versatile UDP/IP based PC <-> FPGA Communication Platform," *2012 International Conference on Reconfigurable Computing and Fpgas (Reconfig),* 2012.

[4] F. L. Herrmann, G. Perin, J. P. J. de Freitas, R. Bertagnolli, and J. B. dos Santos Martins, "A gigabit udp/ip network stack in fpga," in *Electronics, Circuits, and Systems, 2009. 16th IEEE International Conference on*, 2009, pp. 836-839.

[5] A. Dollas, I. Ermis, I. Koidis, I. Zisis, and C. Kachris, "An open tcp/ip core for reconfigurable logic," in *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, 2005, pp. 297-298.

[6] V. Vishwanath, P. Balaji, W.-c. Feng, J. Leigh, and D. K. Panda, "A case for UDP offload engines in LambdaGrids," in *4th International Workshop on Protocols for Fast long-distance Networks (PFLDnet'06)*, 2006, p. 5.

[7] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks: Pearson New International Edition*: Pearson, 2013.

[8] J. Postel, "RFC 791: Internet protocol," 1981.

[9] J. Postel, "RFC 793: Transmission control protocol, September 1981," *Status: Standard,* vol. 88, 2003.

[10] J. Postel, "RFC 768: User Datagram Protocol (UDP)," *Request for Comments, IETF,* 1980.

[11] Virtex-5 Embedded Tri-mode Ethernet MAC Wrapper.http://www.xilinx.com/products/intellectualproperty/v5_embedded_temac_wrapper.html (last visited 16.III.2015)

[12] Xillybus, http://www.Xillybus.com (last visited: 03.02.2010)

[13] L. Xilinx and I. E. B. Plus, "v1. 9 for PCI Express User Guide," ed: September, 2008.

SESSION

RESOURCE CONSTRAINED DEVICES

Chair(s)

TBA

Neural Cryptography for Secure Voice Communication using Custom Instructions

C. H. Lin, B. C. Yang, C. B. Duanmu, B. W. Chen, De-Sheng Chen, Yiwen Wang

Department of Information Engineering and Computer Science
Feng Chia University,
No. 100 Wenhwa Rd., Seatwen, Taichung, Tawian

Abstract - *Cryptography of resource constrained devices represents a very active area of cryptographic research. Custom instructions have been widely used to achieve the conflicting demands between performance and flexibility. This paper proposes a neural cryptography implementation for secure voice communication using custom instructions to achieve real-time performance on very low resource devices. The experiments show that using only very limited hardware to implement the CIs, the 40 speed-up can be saved to speed-up the performance.*

Keywords: Neural Cryptography; Custom Instruction (CI)

1 Introduction

The importance of cryptography on resource constrained devices is related to the current trend of pervasive/ubiquitous computing, which means an ever increasing demand for computing capabilities in diverse, wireless and low-resource scenarios, in both civilian and military applications, including mobile phones, smart cards, toll collection, animal and cargo tracking and electronic passports, and etc. Due to the low-resource environments, cryptographic algorithms are typically hardware-oriented, and designed to be particularly compact and efficient. The balance between security, high performance (in hardware), and low overall cost (throughput, power consumption, area, price) in low-resource environments represents a major challenge in cryptographic acceleration.

The learning and classifying abilities of neural networks can be used for different aspects of cryptography such as to learn the inverse-function of any cryptographic algorithm in cryptanalysis or to solve the key distribution problem in public-key cryptography using neural network mutual synchronization. The classical key exchange problem in cryptography are mainly based on algebraic number theory, but the synchronization phenomenon of interacting two neural nets provides a new idea to solve this problem. Good reviews of neural cryptography can be found in [1, 2], for analysis of neural learning rule and protocol dynamics. Mislovaty et al. reported that ANN was secure against brute-force attacks [3]

The custom instruction set extensions have been highly successful such as Intel MMX and SSE, AMD 3DNow! and DSP instructions for digital signal processors. A partially customizable instruction-set which can be tuned towards the specific requirements of applications by extending the basic instruction set with dedicated custom instructions within custom functional units (CFUs). By using a base processor, the design process can focus on the CIs only, that significantly reduces verification efforts and hence shortens the design cycle by sharing development tools such as compilers, debuggers, simulators. Commercial examples are Tensilica Xtensa [4], Xilinx MicroBlaze [5], and Altera Nios II [6]. The security support has been included in commercial embedded processors such as ARM SecurCore [7], STMicroelectronics SmartJ [8], and Atmel XMEGA [9]. However, most commercial cryptographic instruction extensions did not release the detail implementation to the public.

Various approaches to optimize or accelerate feedforward neural nets for embedded systems [10, 11, 12]. Santos [13] proposed a custom instruction to approximate the value of ***tanh()*** through the use of a range addressable lookup table for the acceleration of a pre-trained feedforward artificial neural network executing on a NIOS II processor.

The proposed neural cryptography for secure voice communication is to derive a set of synaptic weights as a pair of keys through the neural learning mechanism to achieve the data encryption and decryption between two private neural nets. The suitable "*neural crypto instructions*" are designed and implemented as CIs of Altera Nios II/e, to achieve a secure voice communication in real-time. The neural crypto CIs may retain the flexibility of original embedded processors to shorten design cost and time but increase performance as well as lower energy consumption and hardware cost.

This research was supported by MOST grant 103-2221-E-035 -053.

2 Methodology

In this paper, we combine approaches in [14], [15], [16] and [17] to accelerate the neural cryptography computation for secure voice communication on very low resource devices.

2.1 Neural Cryptography for Secure Voice Communication

A typical neural cryptography application for voice communication is depicted in Fig.1. The voice is first pre-processed, followed by a 512-point short-time Fourier transform (STFT) of ***voice(t)*** with a 20-ms Hamming window to obtain ***P(n, d)***, where ***n*** is the frequency bin sample index, n =1,…,256, and ***d*** is the frame index, d=1,…D. The ***P(n, d)*** will be the plain text as the input of the neural cryptography encryption. The ***C(n, d)*** will be the cipher text of the output and can be sent to the unsecure public communication channels. Once the ***C(n, d)*** is received, it can be decrypted to the plain text, ***P'(n, d)***, by the neural cryptography decryption and converted to ***voice'(t)*** by inverse short-time Fourier transform (ISTFT).

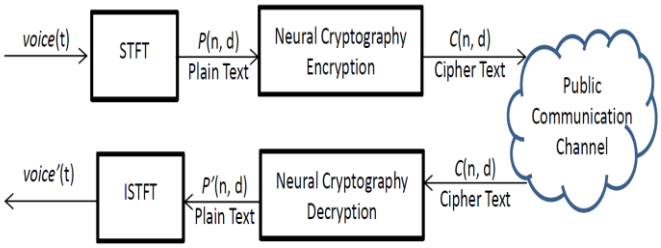


Fig. 1. A typical neural cryptography application for voice communication.

A four stage multilayer feedforward neural network will be used for neural cryptograph encryption and decryption as shown in Fig. 2.

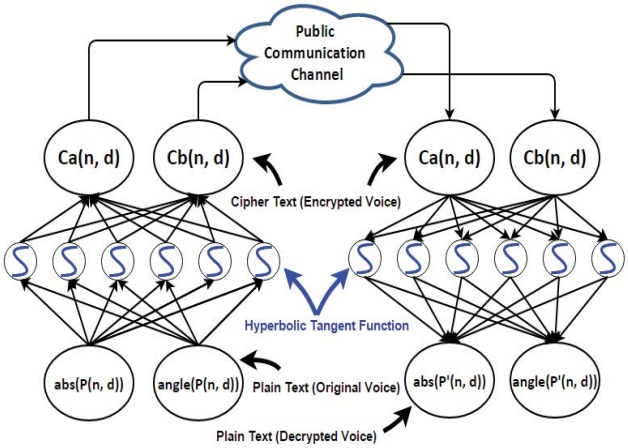


Fig. 2. A four stage multilayer feedforward neural network for neural cryptograph.

The first two stages are consisted of one non-linear function layer, ***tanh()***, and one linear function layer to perform the neural cryptography encryption. The last two stages are still consisted of the same neural net structure to perform the neural cryptography decryption. The random and unpredictable initial values are used by the back-propagation learning rule to train the four stage multilayer feedforward neural network using the training set that the desired target patterns are equivalent to the original input patterns, ***P(n, d)***. Once the required minimum mean-square error (MSE) between neural net outputs, ***P'(n, d)***, and the desired targets, ***P(n, d)***, is achieved, the synaptic weights of the first and last two stages will be the encryption key and the decryption key correspondingly.

2.2 Custom Instruction

Fig. 3 shows the block diagram of a typical custom instruction processor. The interface between the base processor and the custom functional unit (CFU) only includes the control signals for the CI encoding and the synchronization of multi-cycle custom instructions. The input and output bandwidths of the data transfer buses are limited by the number of read ports and write ports of the general purpose register file (GPRF) of the base processor. This simple interface keeps the base processor data-path unchanged to simplify the CI implementation and to reduce the design and verification cost.

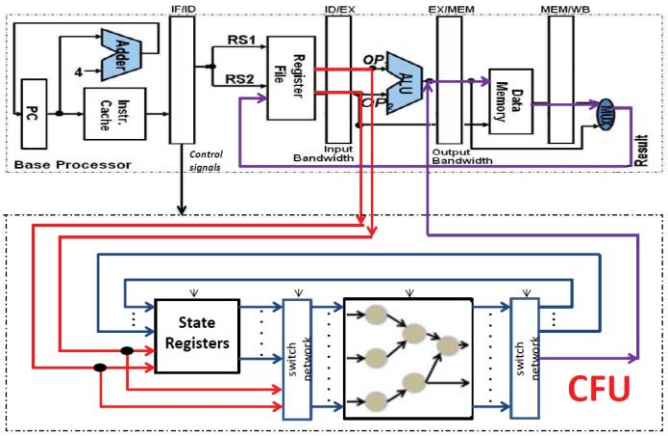


Fig. 3. A typical custom instruction processor.

Typical multiply-accumulators (MACs) are implemented as CIs to achieve parallel computation between the synaptic weights and neuron output signals. The property of non-linear hyperbolic tangent sigmoid function, ***tanh()***, is implemented as a single CI to approximate the value of ***tanh()*** through the use of a hybrid range addressable lookup table to store the mapping data from C code precision simulation according the [15].

Several data movement CIs are implemented to explicitly move additional input and output operands between the base processor and the state registers in CFU so that the

performance of CIs will not be limited by the available data bandwidth between the base processor and CFU. The CI scheduling and state register assignment can be optimized according to sequential ordering of data for better use of the CIs to achieve a faster execution time.

3 Results

Firstly, we use the MATLAB to determine the topology parameters of feedforward neural networks and conduct the cryptography feasibility analysis. Secondly, we use C++ programs to normalize the MATLAB double floating point results to the correct range of the fixed point values as shown in Fig. 4. The appropriate bit-precision of fixed point is then selected for hardware implementation. Finally, we implement the proposed neural cryptography on an embedded processor with CIs to speed up the execution of secure voice communication on very low resource embedded processors.

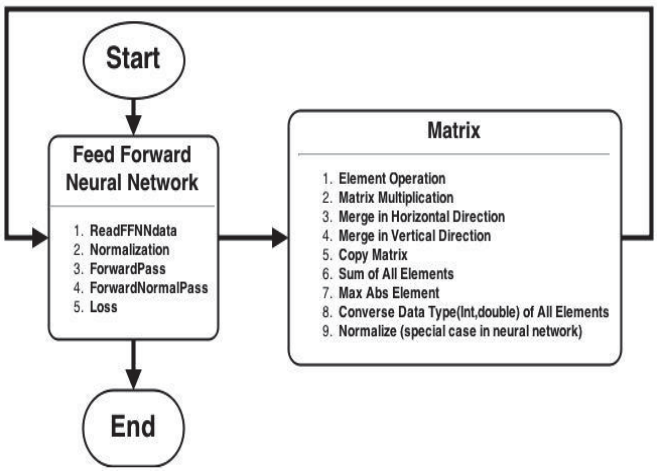


Fig. 4. Conversion from MATLAB results to C++ for the bit-precision analysis.

3.1 MATLAB Results

Fig. 5 depicts the original ***voice(t)*** as well as the results of STFT, abs(***P(n, d)***) and angle(***P(n, d)***), respectively.

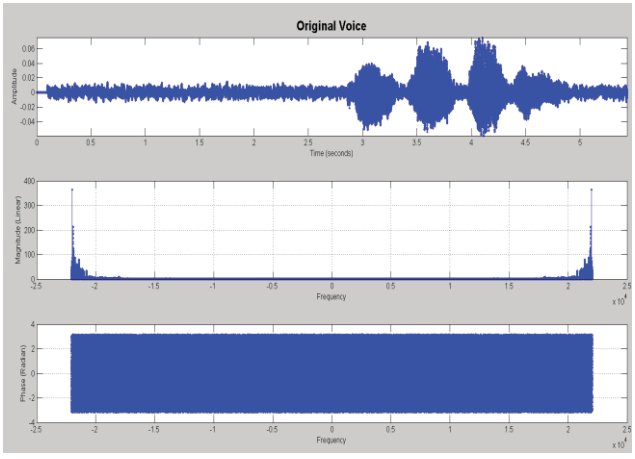


Fig. 5. The original ***voice(t)***, abs(***P(n, d)***), and angle(***P(n, d)***).

Fig. 6 depicts the encrypted signals of ***voice(t)*** and the encrypted cipher texts of ***Ca(n, d)*** and ***Cb(n, d)***, respectively. Fig. 7 shows the decrypted ***voice'(t)*** as well as the decrypted plain texts of abs(***P(n, d)***) and angle(***P(n, d)***), respectively.

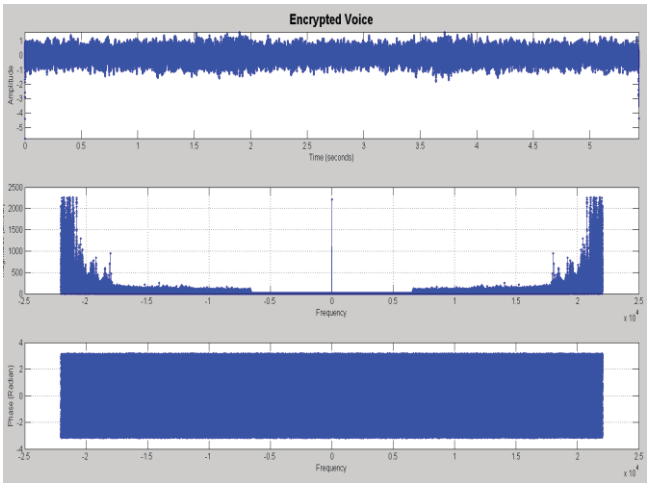


Fig. 6. The original ***voice(t)***, abs(***P(n, d)***), and angle(***P(n, d)***).

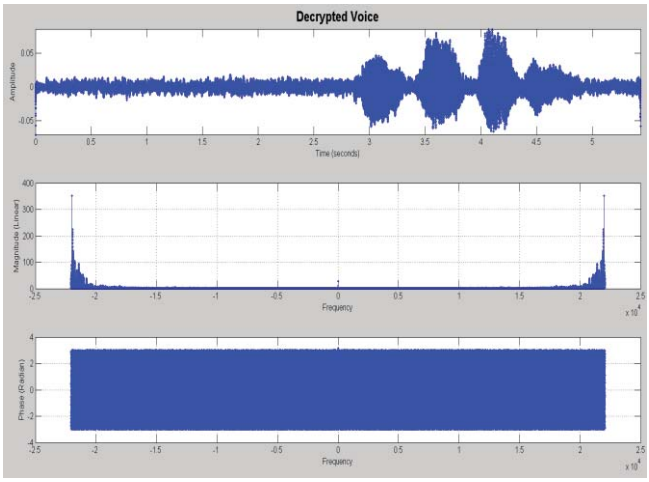


Fig. 7. The original ***voice(t)***, abs(***P(n, d)***), and angle(***P(n, d)***).

The Equ. (1) is defined as the mean-square error (MSE) of the four stage multilayer feedforward neural network to measure the quality of learning results after the 1000 iterations of the back-propagation weight update equations. Usually the smaller MSE obtains the better quality of neural network representations.

$$\text{MSE after leraning} \triangleq \frac{\sum_{n=1}^{N}\sum_{d=1}^{D}(P(n,d)-P'(n,d))^2}{N\times D} \quad \ldots\ldots(1)$$

The total entropy is defined in Equ. (2) to measure the discrepancy between the plain text and the cipher text. The larger total entropy is the better effectiveness of the encrypted voice.

$$\text{Total Entropy} \triangleq \frac{\sum_{n=1}^{N}\sum_{d=1}^{D}(P(n,d)-C(n,d))^2}{N\times D} \quad \ldots\ldots\ldots(2)$$

The Signal-to-Noise Ratio (SNR) of the received voice is defined in Equ. (3) to evaluate the quality between the original voice and the decrypted voice. The larger SNR is the better quality of the decrypted voice.

$$\mathrm{SNR}_{time} \triangleq 20\ log_{10}\left(\sqrt{\frac{\sum_{i=1}^{N\times D} voice(i)^2}{N\times D}} \Big/ \sqrt{\frac{\sum_{i=1}^{N\times D}(voice(i)-voice\prime(i))^2}{N\times D}}\right) \quad \ldots\ldots(3)$$

The MATLAB results are shown in Table 1~3 which are used to conduct the feasibility analysis of neural cryptography for secure voice communication and to determine the neural network topology parameters according to the number of hidden neurons and the number of bits for data type representations. Table 1 shows the MSE results of our proposed neural network after the 1000 iterations of the back-propagation learning rules to train 400 randomly generated data. Table 2 shows the total entropy and Table 3 shows the SNR for various numbers of hidden neurons with respected to different data types. The larger number of hidden neurons and the larger number of bits will result in the larger hardware cost and the longer execution time.

The MSE of neural net after learning						
Number Of Neurons / Data Type	8	16	32	64	96	128
Double-precision Floating-point	4.73E-04	8.03E-04	4.57E-04	1.49E-03	8.21E-03	2.05E-03
32 Bits Signed Integer	4.68E-04	7.98E-04	4.42E-04	1.51E-03	8.24E-03	1.91E-03
16 Bits Signed Integer	1.19E-03	1.03E-03	9.25E-03	4.50E-02	1.83E-01	6.20E-01
8 Bits Signed Integer	1.21E-01	9.48E-02	2.92E-01	1.64E-01	1.64E-01	1.64E-01

Table 1. The MSE of our proposed neural net after learning.

Total Entropy						
Number Of Neurons / Data Type	8	16	32	64	96	128
Double-precision Floating-point	8.41E-01	1.28E+00	1.17E+01	6.53E+01	4.14E+02	4.93E+02
32 Bits Signed Integer	8.40E-01	1.28E+00	1.17E+01	6.52E+01	4.13E+02	4.91E+02
16 Bits Signed Integer	7.27E-01	1.09E+00	8.82E+00	3.65E+01	2.07E+02	1.90E+02
8 Bits Signed Integer	3.03E-01	2.15E-01	5.95E-01	1.64E-01	1.64E-01	1.64E-01

Table 2. The total entropy between the plain and cipher texts.

SNR of received voice						
Number Of Neurons / Data Type	8	16	32	64	96	128
Double-precision Floating-point	1.44E+01	1.51E+01	1.41E+01	1.00E+01	1.01E+01	9.31E+00
32 Bits Signed Integer	1.45E+01	1.52E+01	1.42E+01	1.01E+01	1.04E+01	9.58E+00
16 Bits Signed Integer	9.62E+00	9.53E+00	4.15E+00	3.76E+00	-3.24E-01	-6.27E-01
8 Bits Signed Integer	-3.08E-01	8.85E-02	1.19E-01	0.00E+00	0.00E+00	0.00E+00

Table 3. The SNRs of received voice from decrypted data.

3.2 Custom Instruction Results

We choose the Altera Nios II/e as the very low resource target base processor for the DE2-70 board run at 100Mhz. Thus, the number of input and output ports for general purpose register file is 2 and 1. The software cycle count of a primitive instruction is estimated by the cycle count in the execution stage of the Altera Nios II/e. The hardware cycle count of a CI is estimated by synthesizing the corresponding template using Altera Quartus II. The cycle count of data transfers between the Nios II/e and the SR of CFU is single cycle latency.

The matrix multiplications and additions are performed by MAC CIs and data movement CIs. The CI block diagrams of the 32-bit implementation is depicted in Fig. 8. The 16-bit and 8-bit data types are shown in Fig. 9. Fig. 10 shows the simulation results of the 8-bit ***tanh()***.

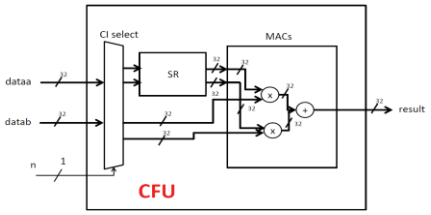


Fig. 8. The 32-bit data type implementation in CFU.

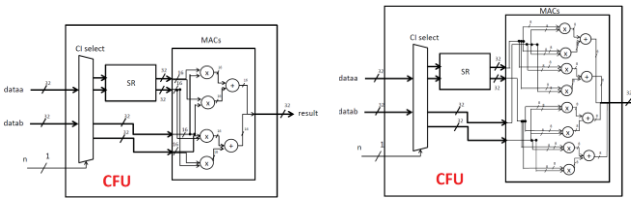


Fig. 9. The 16-bit and 8-bit data type implementations.

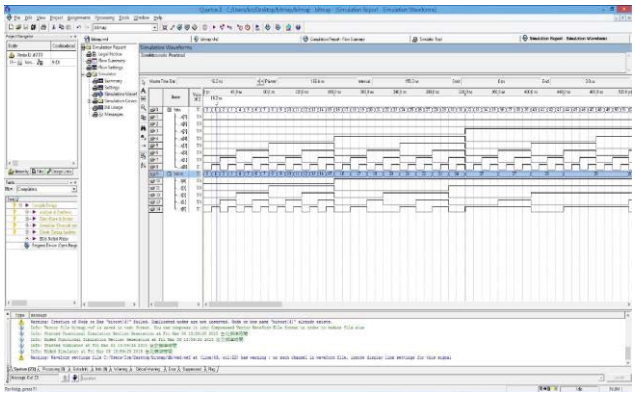


Fig. 10. The simulation results of the 8-bit ***tanh()***.

Table 4 shows the total number of execution cycles and the total number of logic elements (LEs) for various CI implementations of neural cryptography for secure voice communication using 32 hidden neurons.

3.3 Performance analysis and discussion

Basically for the better sound quality and the more security of the voice communication, we need the more number of hidden neurons for the neural net and the more number of bits for the arithmetic computations. The larger numbers of neurons or bits will result in the more hardware cost and the execution time. The desired sound quality can be chosen using the SNR of received voice from Table 2 as

well as the desired security can be chosen using the total entropy between the plain and cipher texts from Table 3. However, the numbers of neurons or bits is not a linear relationship with the quality and security.

From these experiments, we may suggest using the number of bits to determine the desired sound quality and using minimum number of neurons to decide the security. Because of a lot of simple and low cost perturbation algorithms for encryption and decryption can enhance the desired security. Once the numbers of neurons and bits are decided, the hardware cost of CI implementations will be determined by using the Table 4 according to the overall computation overheads of real-time applications, usually the number of STFT points within a fixed latency Hamming window.

Computational Custom Instructions	# of CIs	Total # of LEs	Total cycles	Total reduced cycles
None(16-bit)	0	700	53720416	0
Four 16-bit **MAC**s	2	**865** =700+ 66(SR)+ 99(MAC)	52393536	1326880
Four 16-bit **MAC**s Two 16-bit **tanh()**	3	**19331**=700+ 66(SR)+ 99(MAC)+ 18466(tanh)	1327216	52393200
None(8-bit)	0	700	53720416	0
Eight 8-bit **MAC**s	2	**865** =700+ 66(SR)+ 99(MAC)	52393488	1326928
Eight 8-bit **MAC**s Four 8-bit **tanh()**	3	**897** =700+ 66(SR)+ 99(MAC)+ 32(tanh)	1327016	52393400

Table 4. The total numbers of execution cycles and logic elements (LEs) for various CI implementations for neural cryptograph using a four stage multilayer feedforward neural network with 32 neurons.

4 Conclusions

This paper presented an approach to implement neural cryptography for secure voice communication in real-time. Our approach took benefits from both software and hardware to make flexible and scalable neural computation available on resource limited embedded processors with good performance for the neural cryptography. The experiments show that using only very limited hardware to implement the CIs, the 300 speed-up can be saved to speed-up the performance. The future work of this paper will be the continuous alternation between theoretical investigation and practical implementation of neural CIs to simutaneously achieve the neural crypto, learning, classifying abilities of various application programs running on the same embedded system.

5 References

[1] A. Adel, et al., "Survey Report on Cryptography Based on Neural Network," International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 12, December 2013.

[2] Andreas Ruttor. Neural synchronization and cryptography. Ph.D. Thesis, Bayerische Julius-Maximilians-Universitat, Wurzburg, 2006.

[3] R. Mislovaty, E. Klein, I. Kanter ve W. Kinzel, "Security of neural cryptography." sign (hi) 1 (2004): 1.

[4] R. E. Gonzalez, "Xtensa: a configurable and extensible processor," *IEEE Micro*, pp. 60-70, 2000.

[5] Xilinx MicroBlaze Soft Processor Core http://www.xilinx.com/tools/microblaze.htm

[6] A. N. I. Processor. Altera Nios II Processor. http://www.altera.com/products/ip/processors/Nios II/.

[7] http://www.arm.com/products/ CPUs/families/SecurCoreFamily.html.

[8] http:// www.st.com/stonline/products/ families/smartcard/sc_sol_ics_st22.htm.

[9] http://www.atmel.com/dyn/resources/ prod_documents/doc7925.pdf

[10]S. Al-Kazzaz, et al., "FPGA implementation of artificial neurons: Comparison study," in Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on, 2008, pp. 1 –6.

[11]D. Shapiro, et al., "ASIPs for Artificial Neural Networks," 6th IEEE International Symposium on Applied Computational Intelligence and Informatics • May 19–21, 2011, Timişoara, Romania

[12]S. Ameen, et al., "FPGA Implementation of Neural Networks Based Symmetric Cryptosystem," 6th International Conference: Sciences of Electronic, Technologies of Information and Telecommunications May 12-15, 2011 – TUNISIA

[13]A. Namin, et al., "Efficient hardware implementation of the hyperbolic tangent sigmoid function," in Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on, May 2009, pp. 2117 –2120.

[14] Santos, et al., "Artificial neural network acceleration on FPGA using custom instruction," Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on , vol., no., pp.450-455, 8-11 May 2011.

[15]M. A. Sartin and A. C. R. Silva, "Approximation of hyperbolic tangent activation function using hybrid meth-ods," in Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 8th International Workshop on, pp. 1–6, july. 2013.

[16]G. Li, C. Hung, D. Chen, and Y. Wang, "Application-Specific Instruction sets Processor with Implicit Registers to Improve Register Bandwidth," 2011 World Academy of Science Engineering and Technology (WASET 2011), Paris France, June, 2011

[17]C. Hung, H. Lin, D. Chen and Y. Wang, "ASIP Instruction Selection with the Encoding-Space Constraint for High Performance," 2012 International Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies, Okinawa, Japan, May 30 - June 1, 2012.

Raspberry Pi Webserver

Max Runia[1] **, Kanwalinderjit Gagneja**[1]
[1]Department of Computer Science, Southern Oregon University, Ashland, OR, USA

Abstract - *We used a Raspberry Pi to configure and set up a webserver with an IP address and port forwarding, which would allow access from another source connected to a network. The webserver will have minimal features; we were focusing more on the development and configuration process with a very fundamental format that makes it easy to understand and simple to teach to someone who has little to no experience setting up and configuring a webserver with a Raspberry Pi. We end up using some basic SQL and a few fundamental Unix/Linux commands and some SSH since the webserver was setup on Linux and involved port forwarding from another computer.*

Keywords: Raspberry Pi, apache, web server, Linux, code index;

1 Introduction

The raspberry pi was invented by Eben Upton. He invented it to help the kids learn the coding in a simpler way. Up to 5 million units of raspberry pi have been sold. This tiny computer is just $35 and is sparking a revolution. There is a large user community of Raspberry Pi, although, it was launched just recently on February, 29th 2012. The Pi enthusiasts are organizing community led events all over the world. Such events are full of learning for the kids to code.

We planned to use a Raspberry Pi to configure and set up a webserver with an IP address and port forwarding, which would allow access from another source connected to a network. The webserver will have minimal features; we were focusing more on the development and configuration process with a very fundamental format that makes it easy to understand and simple to teach to someone who has little to no experience setting up and configuring a webserver with a Raspberry Pi. We end up using some basic SQL and a few fundamental Unix/Linux commands and some SSH since the webserver was setup on Linux and involved port forwarding from another computer.

Now to elaborate on what a Raspberry Pi is, the Raspberry Pi is a credit-card-sized single-board computer developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools [6]. The Raspberry Pi sports a meager 256MB of RAM and a 700MHz ARM-11 processor. The Model B also contains two USB ports, an HDMI out and a 10/100 Ethernet port. For audio it possesses a 3.5mm audio jack, the HDMI output also supports audio transmission. The Raspberry Pi's GPU boasts 1 Gpixel/s, 1.5 Gtexel/s or 24 GFLOPs of general purpose compute power and is OpenGL 2.0 Compliant. Nothing too fancy or complex, just a very simple machine meant for learning and is shown in figure 1.

Figure 1: Raspberry Pi

There is one daunting question 'What could you do with such a small computer?' The people are using it for various purposes. For example, Picade Arcade Cabinet has put it into practice like a arcade machine, which is very small but fully functional [2]. Another inventor used a weather ballon to put his camera attached with Raspberry Pi to the upper enviornment to record what all is happening underneath. One of the invertors used Raspberry Pi for streaming music, where it is being used as a very low-cost wireless music streaming equipment [5]. Another inovator prepared a Raspberry Pi Keyboard Computer. This innovation presents a complete computer packaged with a keyboard. One of the inovators used Raspberry Pi as a Bitcoin Miner and named it as 'Pi-Powered Bitcoin Miner'. Therefore, this $35 tiny computer allows you to take part in mining of Bitcoin that can give you certain amount of monetary benefit [2].

The Raspberry Pi has a microcontroller. And this microcontroller's functions depend on specific timing. However, the microcontroller is largely designed to work for one purpose only. Therefore there are not operating system overheads or there are no drivers's to slow the system down. So, this microcontroller uses exceptionally detailed clock cycles to perform any task.

2 Process Followed

To begin the process of configuring a Raspberry Pi to work as webserver, an Operating System of a Linux Distribution, called Wheezy, specifically created for the Raspberry Pi, was downloaded onto an SD memory card with at least 4 GB of memory, via another computer, and inserted into the Raspberry PI [8]. Details of the completed installation can be viewed by opening the website [1] and logging in with the correct user credentials. A Windows Distribution or a Mac OS X can be installed instead, Linux was a personal preference.

The Raspberry PI is then connected to a power source through a micro USB power supply and an Ethernet network outlet and requires a monitor and keyboard to view and give the commands, since it is a Linux based OS we did not require a mouse. With the SD card inserted and the physical connections set up properly, a configuration window appears on first boot, called raspi-config, from there the Raspberry PI can then be setup through command prompts and an options menu, as shown on page 4 in figure 2.

You can alter the local time or time zone and enable ssh, as we did, but when you are finished you are going to select the second command prompt "expand_rootfs" and press enter. To put it simply, this will partition the OS of choice onto the entire space of the SD card. Afterwards you will reboot the Raspberry Pi by selecting “Finish” and pressing enter.

Now from here on there are many different methods to finish the configuration and installations of your Raspberry Pi to make it work as a webserver, we will stick to the methods we used to accomplish this, if you are interested in other methods they are easily found all over the internet. The rest can be accomplished through following simple instructions or tutorials online. Since we enabled ssh we were then able to use to finish the setup and installations from a laptop [7]. We used the cmd console to access the pi with ssh and used sudo and Linux bash commands [4] to finish the configurations and install apache webserver and php as shown in fig. 3.

These commands can be seen in the code appendix after the Conclusion section of the paper. Other OS’s on the Raspberry Pi may possess a GUI you can interact with directly from the Raspberry Pi, being a Linux OS we had to use the command console without any GUI.

Once the installation and configuration was finished we needed to obtain a domain name for the server. We went to noip.com [9] which allows us to use their domain name for free; otherwise we would have to purchase a domain name. With the noip domain name or server name is rasppiserver.no-ip.org, the noip domain name is highlighted for reference (fig 4.).

As mentioned earlier, we used port forwarding to access the Raspberry PI from another source; another computer, phone, etc. To do this, the source would have to attempt to gain access through the router where the Raspberry Pi is connected.

This can be accomplished by typing in the Public IP address 75.142.152.19 and either port 80, 21, 22, or 23 into the URL bar as so: 75.142.152.19:80. This will forward any traffic attempting to access the Public IP address towards the Local/Private IP address 192.168.1.6 of Raspberry PI, thus granting us remote access to the Raspberry Pi and its contents as shown in fig. 5.

2.1 Problems Faced

Over the course of the whole set up, we encountered a few problems. Each of which required a little research to solve. Following is a list of problems we encountered and a few screen shots of the solutions we used. Here we were having trouble getting phpmyadmin to show up at rasppiserver.noip.org/phpmyadmin [1]. To solve this problem we needed to make a short cut to phpmyadmin in the apache2.conf file, so that apache knew where to include it from. (Notice the last line added to the apache2.conf file as shown in fig 6.)

Another problem we came across was not being able to find a repository for the no-ip update program. First we tried to install it using *sudo apt-get install no-ip2* but it was not available for the wheezy distribution. To solve this we had to download a tar file directly from the no-ip server and compile and install it ourselves. We used the command:

wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz to download the tar file.

We were able to get the Raspberry Pi configured and set up with a Private IP address and port forwarding working. We can access the Raspberry Pi from any computer through port forwarding, using the local IP address of designated router and a specified port [3]. The results conclude that using a Raspberry Pi to configure a webserver is simple enough that mostly anyone could accomplish it in a number of different ways.

2.2 Lessons learnt

What we learnt from this work is how to configure a webserver through the use of a Raspberry Pi. We learned how port forwarding works with IP addresses. We gained some basic familiarization with sudo commands in Linux. It can also be acknowledged that webservers can host more than just html or php webpages. They can be used for file storage and organization among many more tasks with a number a uses. As mentioned before, this is not the only way to configure a Raspberry Pi, there are multiple procedures to accomplish the goal of creating a webserver with any design the owner would

prefer. Also, this is not the only way to set up a webserver in general, just a very fundamental method we used for this project; there exist many more complicated methods with varying methods and different results.

3 Conclusions

We were to setup a web server in very small budget. So we planned to use a Raspberry Pi to configure and set up a webserver with an IP address and port forwarding, which would allow access from another source connected to a network. In this case the given webserver has minimal features. The main focus is more on the development and configuration process with a very fundamental format that makes it easy to understand and simple to teach to someone who has little to no experience setting up and configuring a webserver with a Raspberry Pi. We used very basic SQL commands, some Linux fundamental commands, and some SSH.

Code Appendix

Here is a list of the commands that we used from the laptop through the command console after accessing the Raspberry Pi with ssh.

154 cd /etc/network

155 ls

156 sudo nano interfaces

157 more interfaces

(edit interfaces file to make local ip static ip: 192.168.1.6)

160 sudo apt-get install apache2 php5 libapache2-mod-php5

162 sudo service apache2 restart

(install apache and php)

164 sudo apt-get install mysql-server mysql-client php5-mysql

(install mysql)

171 sudo chown -R pi /var/www

(make user: pi the owner of /var/www -the root directory where all files go for the website hosted on our server)

172 sudo apt-get install vsftpd

173 sudo nano /etc/vsftpd.conf

174 sudo service vsftpd restart

(install and edit the ftp program)

177 sudo apt-get install libapache2-mod-auth-mysql php5-mysql phpmyadmin

(install phpMyAdmin)

181 sudo nano /etc/apache2/apache2.conf

182 sudo service apache2 restart

(edit the apache config file so that phpMyAdmin can be accessed)

193 mkdir /home/pi/noip

194 cd ~/noip

196 wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz

197 tar vzxf noip-duc-linux.tar.gz

198 cd noip-2.1.9-1

199 sudo make

200 sudo make install

(install the noip update program)

202 sudo /usr/local/bin/noip2

(start the no-ip update program)

4 References

[1] rasppiserver.no-ip.org/phpmyadmin Last visited Feb. 2015.

[2] http://www.scribd.com/doc/255957808/Raspberry-Pi-for-Beginners-Revised. Last visited March 26th 2015

[3] "Hacking Raspberry Pi" Timothy L. Warner, 2013.

[4] http://www.openvisionnetworks.com/dev/Learn%20Raspberry%20Pi%20with%20Linux.pdf last visited March 26th 2015.

[5] http://www.instructables.com/id/Raspberry-Pi-Projects/ Last visited April 3rd 2015.

[6] https://dev.windows.com/en-us/featured/raspberrypi2support Last visited March 3rd 2015.

[7] http://elinux.org/RPi_Hub Last visited March 3rd 2015.

[8] "Learn Raspberry Pi with Linux", Peter Membery and David Hows

[9] noip.org

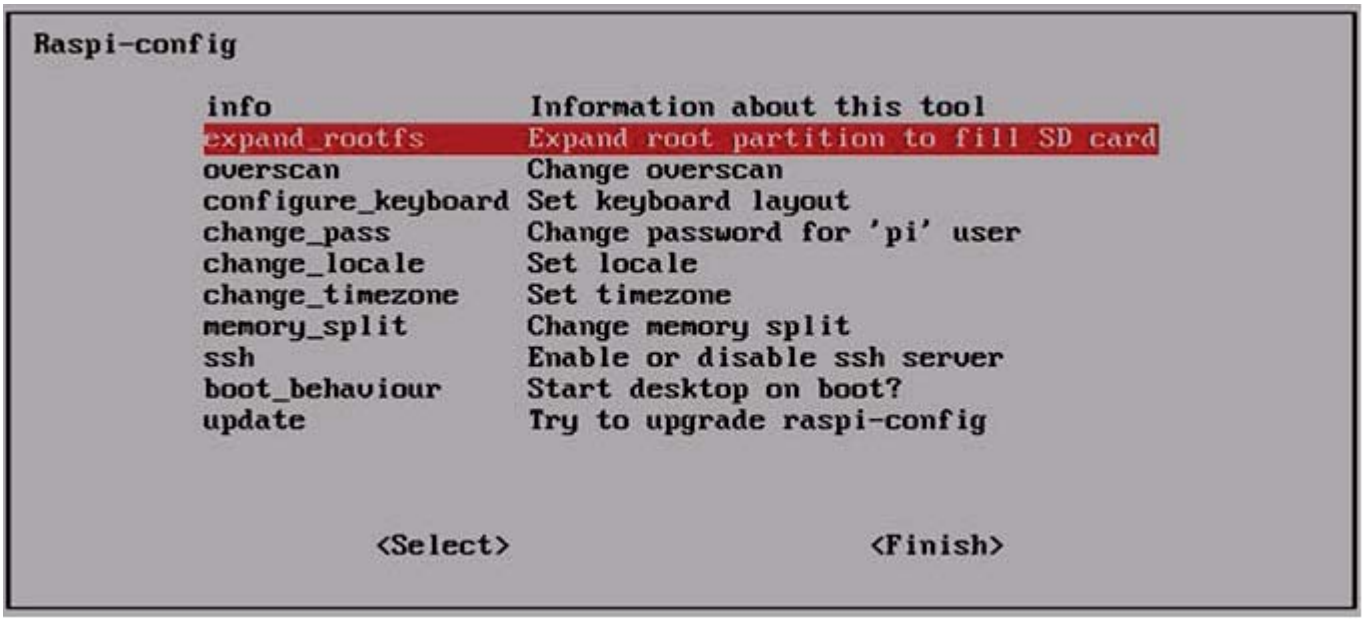


Figure2: Raspi-config window, information about the tool

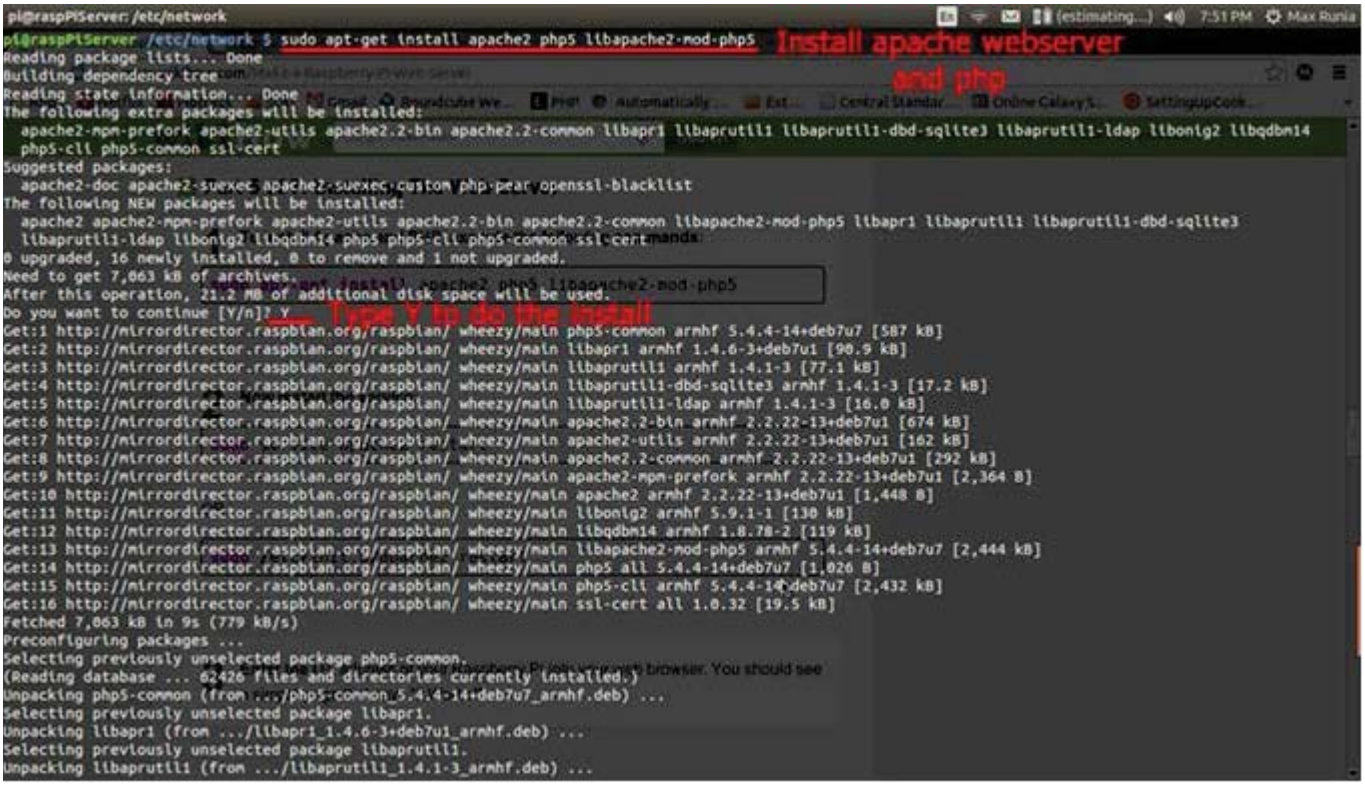


Figure 3: installing apache webserver and PHP

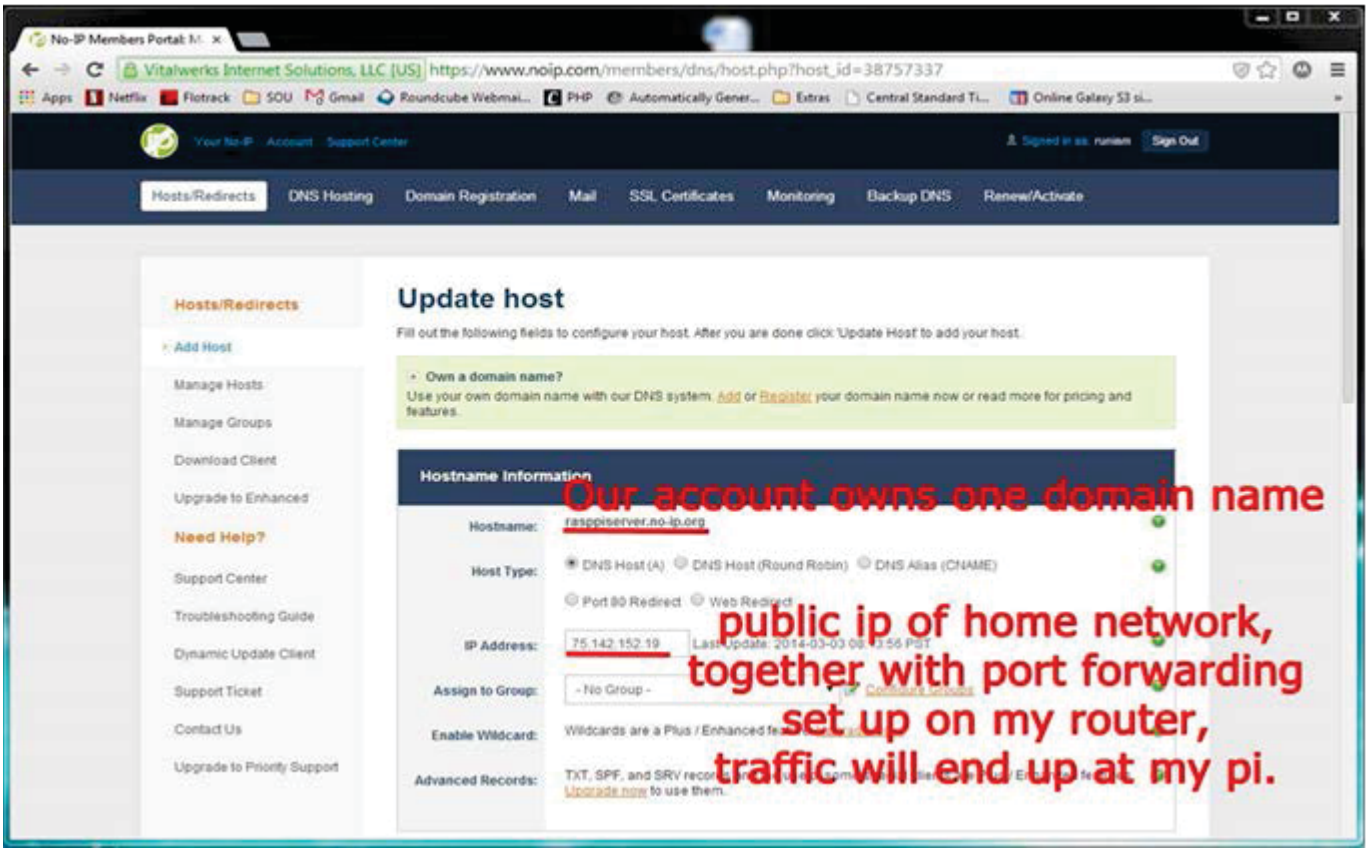


Figure 4:NO-IP free member portal

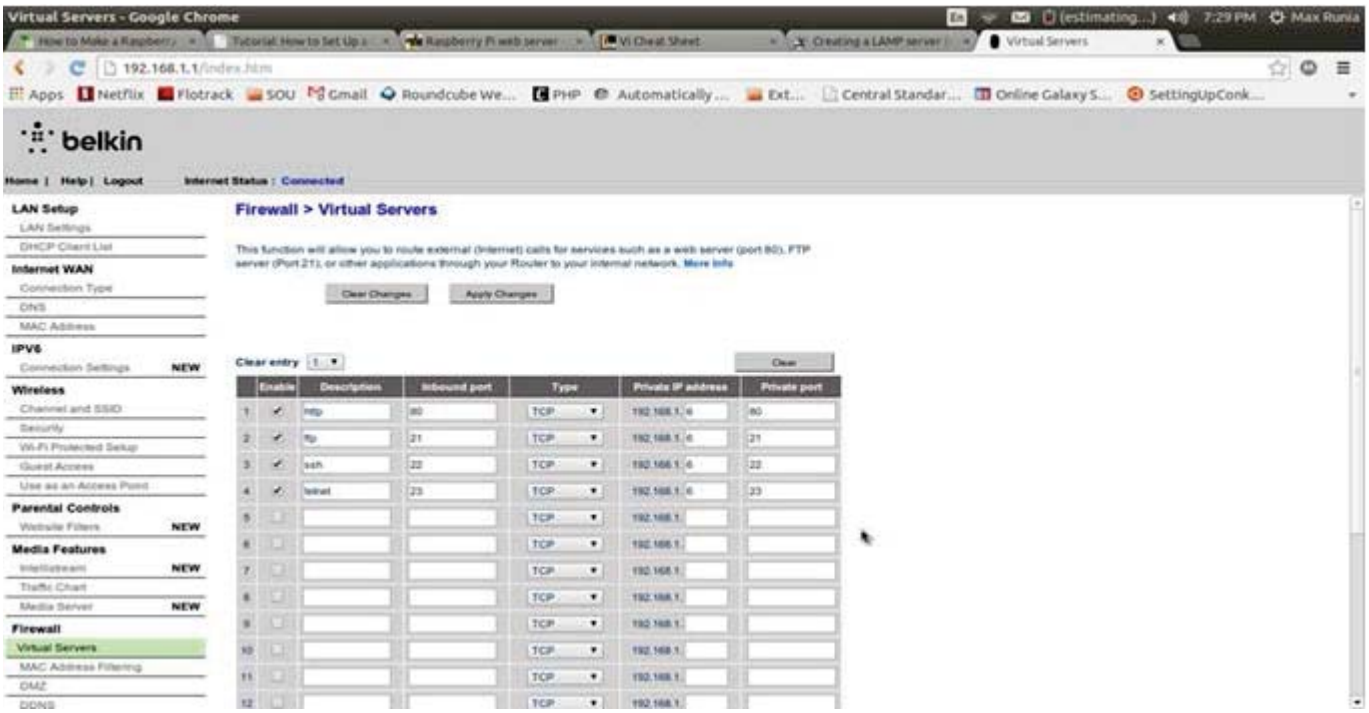


Figure 5: Virtual server working

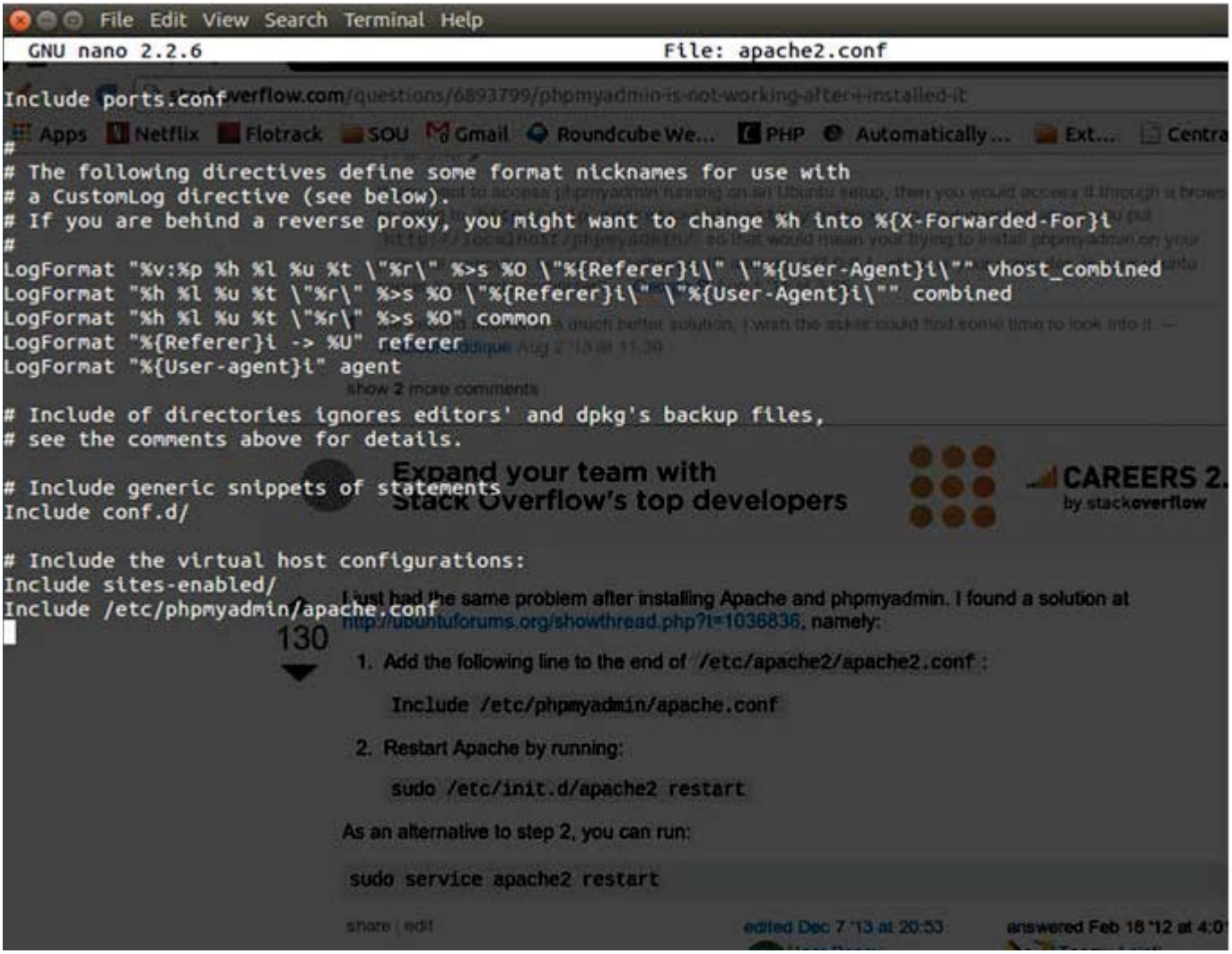


Figure 6: Modifications to apache.conf

SESSION

POWER AWARE COMPUTING AND ENERGY MINIMIZATION + SUPPORTING TOOLS

Chair(s)

TBA

A Visualization Method of Inter-module Communications for Profiling Energy Consumption of Android Applications

Hiroki Furusho[1], Kenji Hisazumi[2], Takeshi Kamiyama[3], Hiroshi Inamura[3], Shigemi Ishida[1] and Akira Fukuda[1]

[1]Graduate School/Faculty of Information Science and Electrical Engineering, Kyushu Univ., Fukuoka, Fukuoka, Japan

[2]System LSI Research Center, Kyushu Univ., Fukuoka, Fukuoka, Japan

[3]Research Laboratories, NTT DOCOMO INC., Yokosuka, Kanagawa, Japan

Abstract— *We propose a method for visualizing the relationship between software modules of applications running on the Android OS. Existing energy estimation methods can analyze energy consumption for each modules of an application. However, it is difficult for application developers to choose a module as tuning target by the above profiling result.*

Our proposed method observes data modules communicating each other, and visualizes the relationship between a large energy-consuming module and other modules. In this study, we analyzed a verification application with proposed method and showed the relationship between these application modules.

Keywords: Energy consumption, Profiling method, Mobile application, Android

1. Introduction

An important task for Android application developers is reducing the energy consumed by their Android app in order to prolong the battery life of the Android smartphones. Although this problem can be addressed at both hardware and software levels, it is important to reduce the energy consumption of individual applications that vary significantly in behavior. Considering the foregoing, it is not sufficient to identify modules to be tuned by individually visualizing energy consumption for each module.

The simplest method for reducing the energy consumption of smartphone applications is to eliminate problems such as excessive creation instances, loop statement errors, communication process errors, and bugs. Energy-profiling methods can identify the points at which the applications consume excessive energy and determine methods to reduce their overall consumption. Existing methods can estimate energy consumption of the entire smartphone by using data obtained from the OS (such as CPU time, amount of file system access, and traffic). The authors have proposed a profiling method that analyzes energy consumption of modules of an application running on the Android OS [1].

However, a module that must be tuned might be different from modules that consume energy excessively. For instance, when module A consumes a large amount of energy, there are two possible causes: module A itself consumes the energy, or other modules use module A excessively. In the latter case, it is not enough to visualize energy consumption for each module individually to identify modules to tune.

Hence, we propose a method to visualize relationships between modules to assist developers in determining modules that must be tuned. The proposed method is applied to a simple application for testing.

The remainder of the paper is organized as follows. Section **2** describes related work. Section **3** explains about the type of communication monitored by our proposed method and specifies the logging process. Experimental results are reported in Section **4**. Section **5** concludes with a summary and specifies the direction of our future works.

2. Related Work

Most smartphone energy analysis methods employ model-based estimation. The basic form of the energy-consumption model can be represented by the following linear equation

$$E_{estimate} = \sum_{m \in M} C_m \cdot V_m \tag{1}$$

$E_{estimate}$, M, C_m and V_m represent estimated energy, a set of the factors related to energy consumption (such as CPU time, data communication, access to storage and display), usage of a factor $m \in M$, and its coefficient, respectively. C_m is calculated by regression analysis of resource usage and measured energy consumption.

Several researchers present methods that use a energy model to estimate the energy consumption of the entire device, using operating times of each part of the device as parameters [2][3]. However, it is difficult to identify the contribution of an application to the total energy consumption because smartphones can run several processes simultaneously.

An estimation method using values obtained from a Linux process file system [4] overcomes the issue [5] because the process file system records the device usage (hereinafter referred to as "resource usage") for each process separately. Mittal et al. proposed an energy consumption profiling

method for the CPU, wireless communication (3G, Wi-Fi) and display[6]. The display energy is consumed by the application because the interface of the application itself usually occupies the smartphone display.

The authors have proposed profiling method that analyzes energy consumption of modules of an application running on the Android OS[1]. However, a module that must be tuned might be different from modules that consume energy excessively as mentioned above.

3. Profiling of relationships between modules

3.1 Overview

This section describes the monitoring process of inter-module communication. Our method provides application developers with profiling result based on the actual usage of users. Regardless of a developer's understanding of an application, the profiling results are helpful.

Our proposed method monitors behavior of an application and records the behavior in the log file. To hook various method in an application, we implemented the logging code using AspectJ[7]. Fig. 1 depicts code weaving using AspectJ. AspectJ includes an additional object named Aspect, which is not a part of Java. The Aspect object contains the conditions of the embedding point (pointcuts) in the source code and the embedding codes (advice). When an application is generated, Java bytecode, which generated from the advice,is embedded into the application.

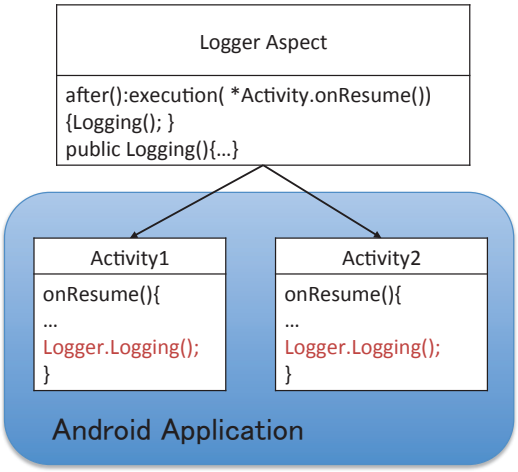


Fig. 1: Code embedding using AspectJ.

3.2 Inter-module communication of an Android application

We classify communication type of Android application modules. Table 1 shows monitoring target modules and calling patterns of modules. The target modules are `Activity` class, `Service` class, `BroadcastReceiver` class and `AlarmManager` classes. Activity provides the GUI for the functions of an application. Service runs longer than the Activity class and performs background processing. These application components call each other's method with data called `Intent`. `Broadcast Intent` is a kind of Intent. Broadcast Intent is sent to all of modules that have a particular attribute value. A BroadcastReceiver can receives a Broadcast intent. AlarmManager sends an Intent in a constant cycle.

Table 1: Calling pattern of Android module.

Activity class and Service class • Normal calling • Calling via AlarmManager
BroadcastReceiver class • Calling from Android application • Calling from Android system

3.2.1 Normal invocation

Fig. 2 shows the logging process of normal invocation of an Activity. This case occurs when switching screen of an application or using a function of Service. In Android system, we send an Intent to invoke the Activity from other Service or Activity. Our proposed method records communication log between sender module(Activity1) and receiver module(Activity2). When an Intent instance is created, our proposed method gives a hash value to the Intent, which enables us to trace a sender from a receiver. After that, when Intent is sent by `startActivity()` ,`startService()` and `bindService()`, the sender module's name and hash value are recorded in the log file. The receiver module receives the Intent and starts processing. Our proposed method obtains the receiver module's name and the hash value given by sender and records them.

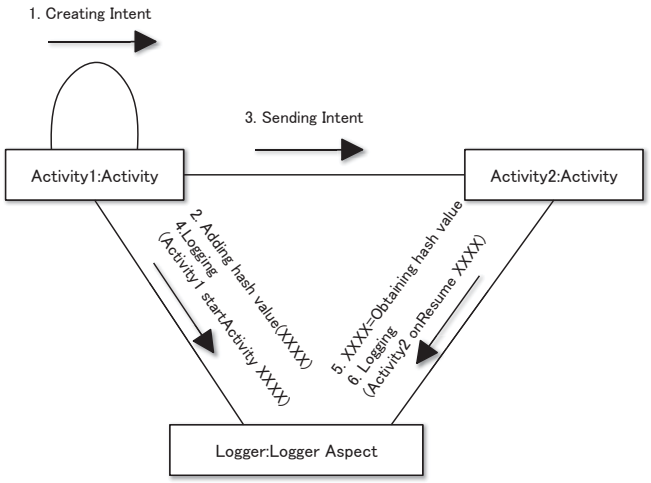


Fig. 2: Log collection of a normal invocation.

3.2.2 Periodic invocation

This section describes how to determine the periodic invocation of services. AlarmManager sends Intents to Services according to settings in advance. For energy saving, it is important to identify which service received the intents from AlarmManager and the number of times they were received. Fig. 3 depicts a method to log such Intents from the AlarmManager to Services. Our proposed method also gives a hash value to the Intent as in the previous section. Our method also gives a hash value to the Intent to trace the sender and receiver in the same way as described in the previous section. The Intent is used to create an instance of a Pending Intent. Unlike in the Activity invocation case, we cannot see the hash value of the sent Intent because we cannot obtain the Intent instance from the `Pending Intent`. To check the hash value of the sent Intent, our proposed method collects the ID numbers of the Pending Intents when creating and sending a Pending Intent. The ID number of Pending Intent is a return value of java.lang.Object.hashCode()[8]. The receiver can log the same way that we mentioned in Section 3.2.1.

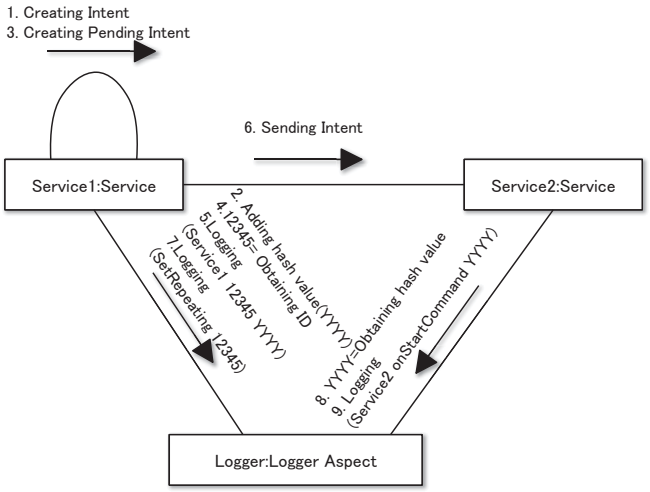


Fig. 3: Log collection of a periodic invocation.

3.2.3 Invocation by Broadcast Intents

The Android system or applications send Broadcast Intents to applications to notify certain events. The application can receive Broadcast Intent to implement BroadcastReceiver and specify a type of Intent that the application wants to receive. The analysis method of this type of communications determines whether frequency and type of Broadcast Intent are appropriate.

Fig. 4 shows how to log for inter-application communication. The logger gives a hash value to an Intent and logs it as a sender, as mentioned in Section 3.2.1. The Android system calls the onReceive() method implemented in BroadcastReceiver whenever the Broadcast Intent is sent. We can hook the onReceive() method to identify the Intent. Fig. 5 shows how to log for receiving a Broadcast Intent from the Android system. We can annotate an attribute called "action" to an Intent. We can distinguish whether the Broadcast Intent is sent from the Android system because a Broadcast Intent sent by the Android system has a value that begins with a particular string, such as "android.action".

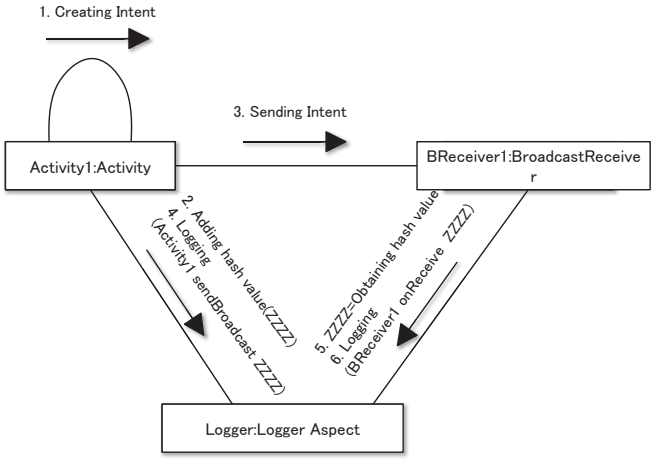


Fig. 4: Log collection of inter-application Broadcast Intent.

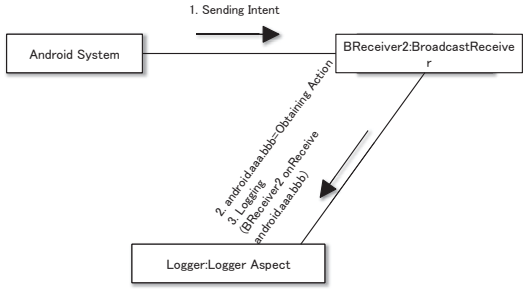


Fig. 5: Log collection of Broadcast Intent from the Android system.

3.3 Log analysis for visualization

This section shows a method, which analyzes and visualizes the collected logs, as mentioned previously. Our method generates a directed graph to visualize communication using Intent between Activities, Services, and the Android system, and annotates information such as energy consumption for each module and frequency of communication for each edge. The analyzer collects all necessary data from an Android terminal. It classifies these logs according to type of module and communication and totals them as shown as in Fig. 6. The type of communication is mentioned in Table 1.

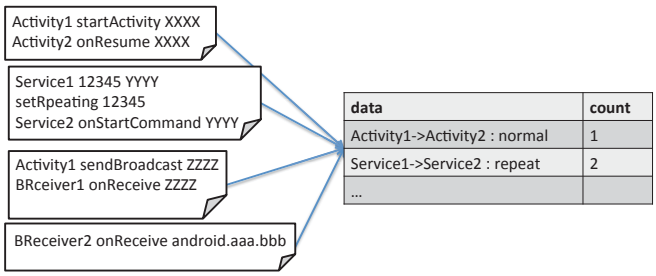


Fig. 6: Counting log files.

4. Evaluation

4.1 Environment

This section demonstrates the proposed method preliminarily to apply to a simple application for evaluation. The authors implemented a log collection function of our method and analyzed applications that are running on a smartphone with our method. Fig. 8 shows a screenshot of the application for verification. The application has three Activity classes, three Service classes, and one BroadcastReceiver class. Each Activity class can transition to another Activity except itself. There are three Service classes, and these classes are: MyService, MyService2, and MyService3, invoked from the Activity classes using startService(), setRepeating(), and bindService(), respectively. The BroadcastReceiver implemented in the application receives Intents from Service classes and the Android system. We implemented the log collection function of our method and analyzed applications that are running on a smartphone with our method.

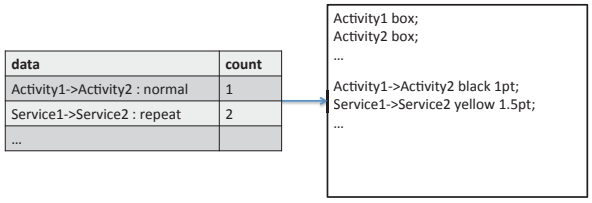


Fig. 7: Generating the DOT file.

4.2 Result

Fig. 9 shows relationships of the testing application modules. We generated this directed graph using the developed logging software and `dot`, which is a program in `Graphviz`[9]. We collected a log, as mentioned in Section 3, and converted the log to the DOT language that dot interprets. Black edges indicate normal invocation. Yellow edges show periodic invocation and Broadcast Intents. The numbers in the edge labels indicate the communication count between pairs of modules. And line thickness of these edges shows the percentage of communication count.

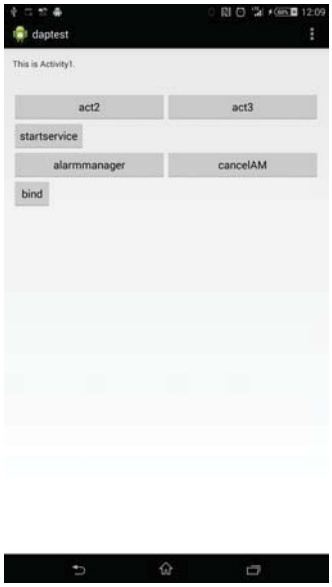


Fig. 8: Screenshot of the application for verification.

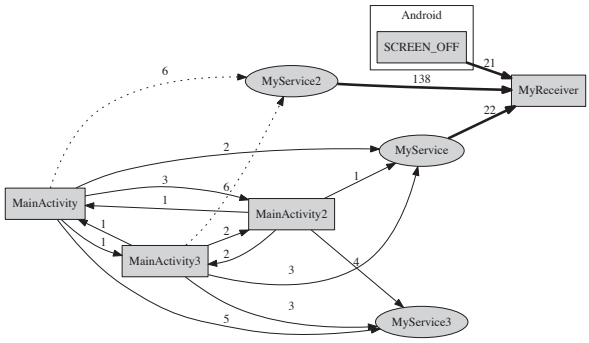


Fig. 9: Relationship graph of the testing application modules.

4.3 Discussion

$E_{processing}$, energy consumption of processing, can be estimated from information that integrates the call graph and module' s energy consumption. A subgraph of call graph indicates a processing of an application. Our proposed method[1] can estimate energy consumptions of modules themselves.

$E_{processing}$ is represented by the summation of module's energy of a processing.

$$E_{processing} = \sum_{m \in Ms} e_m \qquad (2)$$

Ms and e_m represent a set of module elements of a processing and energy of $m \in Ms$, respectively.

5. Conclusion

In this paper, we presented a profiling method identifying relationships between Android application modules. Our method monitors and records communication between modules in an Android application, and visualizes them. Our proposed method helps application developers in identifying hidden energy consumption problems that are caused by communication in the application. We identified types of communication that should be visualized in an Android application. We also preliminarily demonstrated the method using a simple application for verification and showed that it can depict communications in the application in the form of a directed graph.

Our future work will include planning to identify communication patterns in a more complicated module structure. At present, we can identify relatively simple communication patterns. We will also demonstrate our method to be applicable in real applications.

References

[1] H. Furusho, K. Hisazumi, T. Kamiyama, H. Inamura, T. Nakanishi and A. Fukuda: Power Consumption Profiling Method based on Android Application Usage,Lecture Notes in Electrical Engineering, Vol. 339, pp.891–898, Springer Berlin Heidelberg(2015)

[2] L.T. Cignetti, K. Komarov and C.S. Ellis: Energy estimation tools for the Palm, Proc. the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM '00), pp.96–103, ACM, New York, NY, USA(2000)

[3] L. Zhang, B. Tiwana, Z. Qian, et al: Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones, Proc. the eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS '10), pp.105–114, ACM, New York, NY, USA(2010)

[4] T. J. Killian: Processes as Files, USENIX Summer Conf. Salt Lake City(1984)

[5] Y. Kaneda, T. Okuhira, T. Ishihara, K. Hisazumi, T. Kamiyama and M. Katagiri: A Run-Time Power Analysis Method using OS-Observable Parameters for Mobile Terminals, 2010 International Conference on Embedded Systems and Intelligent Technology (ICESIT 2010), Vol.1, pp.39–44(2010)

[6] R. Mittal, A. Kansal and R. Chandra: Empowering Developers to Estimate App Energy Consumption, Proc. the 18th Annual International Conference on Mobile Computing and Networking (Mobicom '12), pp.317–328, ACM, New York, NY, USA(2012)

[7] The Eclipse Foundation, The AspectJ Project, available from(http://www.eclipse.org/aspectj/) (accessed 2015-05-04)

[8] PendingIntent | Android Developers, `http://developer.android.com/reference/android/app/PendingIntent.html#hashCode()` (accessed 2015-05-10)

[9] Graphviz | Graphviz - Graph Visualization Software, `http://www.graphviz.org/` (accessed 2015-05-10)

Task Mapping with Cache Reconfiguration and Partitioning for Energy Minimization on Real-Time Multicores

Zhihua Gan[1,2], Zhimin Gu[1]
[1]School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081 ,China
[2]Software School, Henan University, Kaifeng 475004 ,China

Abstract - *In this paper, we investigate the problem of task mapping with Dynamic Cache Reconfiguration (DCR) and Cache Partitioning (CP) which are promising techniques to alleviate cache energy consumption. Our goal is to obtain an optimal task mapping, L1 cache configuration and L2 cache partition factor on a target multi-core architecture such that cache energy consumption is minimized while timing constraints is satisfied. Two approaches are presented to solve this problem: the first optimal approach is based on integer linear programming (ILP), whereas the second approach is a genetic algorithm (GA) that is near-optimal, but scalable. Experimental results show that our ILP based approaches can find the optimal task mapping, leading to significant energy reduction, and the computation time is tolerable. Moreover, our GA can also find a near-optimal solution with little time overhead.*

Keywords: Mulitcore system, energy consumption,cache, task mapping.

1 Introduction

Multi-core architectures are becoming increasingly popular in real-time embedded systems. This is obvious from the variety of multi-core processors available, such as ARM Cortex-A15 [1] and MIPS32 74K [2]. Multi-core architectures provide the flexibility of simple design, high performance and low-cost implementations. One of the major challenges in multi-core systems is task mapping. Task mapping need determine which tasks should be allocated on which core. Meanwhile, the task mapping is subject to the required objectives, platform constraints and energy requirements.

Energy consumption is still a primary concern for real-time embedded system, especially for battery-driven embedded system. In multi-core platform, to alleviate the off-chip memory access latency, it is usually equipped with multi-level caches (e.g. private L1 cache and shared L2 cache). Although caches effectively improve the system performance, its energy consumption is a problem. Some researches [3] [20] show that the energy consumption of the caches account for up to 50% of the total system. Therefore, reducing the energy consumption of cache is critical for prolonging the lifespan of the system,

Dynamic cache reconfiguration (DCR) is a promising technique to reduce cache energy consumption, which can tune the cache configuration (e.g. cache size, line size and associativity) at run time according to the cache requirement of task, then significant energy saving can be achieved without violating timing constraints. Cache partitioning is also an active research field for reducing cache energy and improving performance, which divides the shared L2 cache into private region, each assigned to a different core or task.

There exist a lot of efforts [4, 5, 17, 27] on cache energy saving for multi-core architecture. Most of works only focus on shared cache partitioning technique, To the best of our knowledge, [5] is only one research study, which jointly takes into account cache partitioning and dynamic cache reconfiguration. However their task mapping is predefined, leading to the local-optimal energy saving. In fact, task-mapping significantly influences energy consumption of cache on multi-core platform. This is because different task-mapping could lead to different scheme of cache partitioning and dynamic cache reconfiguration. Changes of cache partitioning and dynamic cache reconfiguration, in turn, affect cache access behavior of the tasks and cache access energy. Eventually, task mapping impact the energy consumption of embedded systems. Therefore, task mapping is deeply related to cache partition and dynamic cache reconfiguration, and they should jointly be considered to optimize the cache energy consumption.

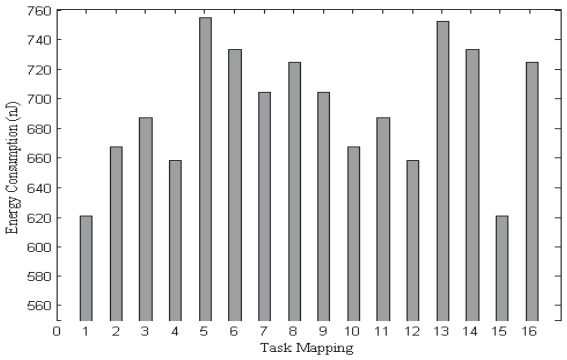


Figure1. Motivation example

Figure 1 shows the impact of task mapping on cache energy consumption for a given task sets consisting of 4 tasks (expint,qurt,prime,jfdcint) selected from MiBench [14], which run on 2 homogeneous core architecture with 2-level cache. Private L1 cache support DCR and shared L2 cache offer CP. For this simple example, we can exhaustively enumerate all the 16 possible task mappings. For each task mapping, we determine the optimal L1 cache configuration and L2 cache

partition factor, and then calculate the cache energy consumption. As expected, different task mapping leads to different energy consumption of caches. Therefore, if the tasks are not pre-assigned to core, we can have chance to reduce energy consumption by grouping such tasks and mapping them to core. Meanwhile, an exhaustive search to find optimal task mappings for reducing energy consumption is not suitable for a large number of task sets. For example, if it has 18 tasks to map 4 cores, the total number of task mappings will be 4^{18}, for each task mapping, finding optimal L1 cache configuration and L2 cache partition takes only α ms, the computation time for energy minimization takes $\alpha * 4^{18}$ ms. Obviously, it is not acceptable. Therefore it is useful to design an algorithm that can find the optimal task mapping with little time overhead.

In this paper, we study the task mapping problem with DCR and CP. We propose two approaches to solve it. This paper makes the following contribution:

1) We propose integer linear programming (ILP) formulation that can find the optimal task mapping, L1 cache configuration and L2 cache partition factor, leading to the best cache energy saving while guaranteeing all timing constraints, and the computation time for finding the optimal solution is tolerable.

2) We develop an effective genetic algorithm (GA) that can find the near-optimal solution without violating timing constraints, meanwhile, have very little time overhead.

3) We demonstrate that our ILP-based approach and GA-based approach is very effective in reducing energy consumption of caches.

The rest of this paper is organized as follows. Related works are presented in Section 2. Section3 describes the architecture model and problem formulation, Section 4 presents our ILP-based approach and GA-based approach. Experimental evaluation is presented in Section 5. Section 6 concludes the paper.

2 Related Work

Task mapping. Many research efforts have been developed for task mapping on multi-core. Here, we only present the task mapping considering cache behavior. In [10]. Li et al. illustrate a task mapping and cache partitioning algorithm to improve the performance of heterogeneous multi-core system. Their target is not to reduce energy consumption and algorithm is not suitable for energy saving. Calandrino et al. [11] aim to improve the performance of shared caches by co-scheduling of groups of tasks and avoiding co-scheduling groups that will thrash the cache. Fedorova [12] propose a new cache-fair scheduling algorithm that reduces co-runner-dependent performance variability and addresses non-uniform cache allocation.

Reconfigurable cache. Numbers of reconfigurable cache architectures are proposed in recent years. For example, zhang et al. [20] proposed an efficient and highly configurable cache architecture whose cache way could be tuned via hardware register at runtime. In [7], a cache architecture, which can be dynamically partitioned and resized at run-time, is designed to improve the performance of embedded systems. Yang et al. [13] propose a selective-sets cache architecture which varies the number of cache sets. Above work are devoted to the reconfigurable cache simulation and the analysis of theoretical proposals.

Cache partitioning. Cache partitioning techniques are studied for various targets in real-time embedded system. Bui et al. [28] aim to minimize the system utilization based on static cache partitioning. In [27], the author proposes a WCET-aware cache partitioning algorithm to decrease the system's WCET. Reddy et al. [24] focus on eliminating inter-task cache contention and reducing energy consumption of cache. Above works are designed to single-core platform. For multi-core platforms, Suh et al. [26] exploit cache partitioning to reduce the average cache miss rate of the concurrent thread. Kim et al. [16] care about fair cache sharing using both dynamic and static cache partitioning. Liu et al. [19] propose a joint task assignment and cache partitioning algorithms with cache locking to minimize the WCRT.

3 Model and Problem Formulation

3.1 Architecture Model

This paper considers an embedded multi-core architecture composed of m identical cores. As shown in Figure 2. Each core has private caches (e.g. private and separate L1 instruction and data caches). All the cores share an L2 cache which is connected to main memory. Here, L1 instruction and L1 data caches are highly reconfigurable in terms of cache size, cache line and associativity. In other words, L1 cache is DCR. This underlying reconfigurable caches we adopt are based on the architecture described in [5][20], which requires very simple hardware augmentation and minor overhead. We use a way- based cache partitioning (CP) in the shared L2 caches. As show in Figure 3, the share L2 cache (here with an 8-way associalitivity) is partitioned in the ways. Each core is allocated a portion of ways and will only access ways allocated in all cache sets. We refer to the number of ways allocated to each core as its partition factor. For example, the L2 partition factor of core 2 in Figure 3 is 2.

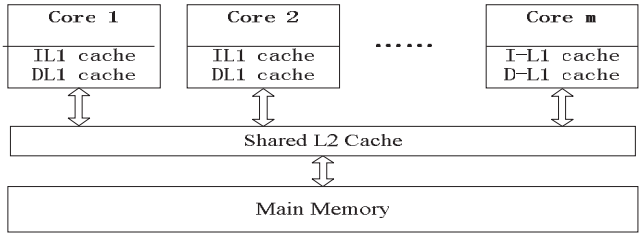


Figure 2 Mulitcore architecture with 2-level cache

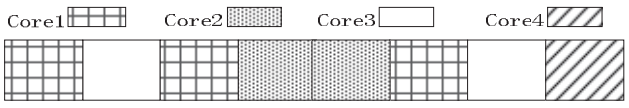


Figure 3 Way-basd partition example for 4cores with 8-way associative shared L2 cache

3.2 Energy Model

Cache energy consumption are composed of dynamic energy E_{dyn} and static energy E_{sta}[20]: $E = E_{dyn} + E_{sta}$. The dynamic energy dissipation E_{dyn} originates from cache accesses and cache misses:

$$E_{dyn}=N_{access}\cdot E_{access}+N_{miss}\cdot E_{miss} \quad (1)$$

Where N_{access} and N_{miss} are the number of cache accesses and misses, respectively. The cache access energy E_{access} is constant according to cache specification. E_{miss} denotes the energy dissipation of a cache miss and is computed as:

$$E_{miss}=E_{mem} + E_{\mu Pstall}+ E_{block\ fill} \quad (2)$$

Where E_{mem} is the energy dissipation of accessing the lower levels memory, $E_{\mu P\_stall}$ is the energy consumed when the core is waiting for instruction or data from the lower levels memory. They can be obtained from memory and processor specification. E_{block_fill} is the energy for filling a fetched data into the cache. E_{sta} is calculated as $E_{sta} = P_{sta}\cdot t$, where P_{sta} represents the static power consumption of cache and t is the total execution time of task. Note that the value of E_{access}, E_{block_fill}, and P_{sta} highly depend on cache configuration, which can be collected from CACTI [21].The access and miss numbers N_{access} and N_{miss} can be obtained using SimpleScalar [25].

3.3 Task Model

There are a set of independent n tasks $T = \{t_1, t_2, \dots t_n\}$ with common deadline D in the system. Each task has a different execution time (ET) and energy consumption, which depend on L2 partition factor allocated to the core it will be running on and L1 cache configuration selected. In addition, the set of tasks assigned to a core will execute sequentially on that core.

3.4 Problem Formulation

The problem of task mapping with DCR and CP on embedded multi-core processor to minimize the overall cache energy consumption can be defined as follows.

Input: Given a set of independent n tasks $T = \{t_1, t_2, \dots t_n\}$ with common deadline D, an embedded multi-core processor with m cores $P = \{p_1, p_2, \dots p_m\}$, every core has private L1 cache which support h different dynamic configurations, all core share an s-way associative L2 cache which support way-based cache partitioning.

Output: The goal of the problem is to design an assignment for n tasks, a partition scheme for shared L2 cache and to select a L1 cache configuration for each task so that the energy of the cache is minimized while the timing constraint D is satisfied.

4 Proposed Approach

4.1 ILP formulation for task mapping

The section presents our ILP approach for task mapping with DCR and CP. For our algorithm1, A given task set, number of L1 cache configuration, number of L2 cache way are required as input data. The algorithm iterates over all tasks (line1), partitions each task for all L2 cache way (line 2) and selects all possible L1 cache configurations (line 3). Subsequently, the execution and energy consumption of task for different cache configuration is determined by invoking simulator (line 4-5) and stored. An ILP formulation is generated (line 9-16) and solved in line 17. We will represent ILP formulation for task mapping problem, which is the one contribution of this paper. ILP formulation can be categorized in four groups: objective function, core constraint, cache constraint and task mapping constraint.

4.1.1 Core constraint formulations

We model a task mapping decision using binary decision $M_{i,j}$, which is equal to 1 if task t_i is mapped to core p_j and 0 otherwise. A task can only be mapped one core, therefore,

$$\forall \text{task i} \quad \sum_{j=1}^{m} M_{i,j} = 1 \quad (3)$$

We define a binary decision variable $Q_{i,j,r}$, which is equal to 1, if task t_i select j^{th} L1 cache configuration and is mapped the core with r ways of L2 cache. Otherwise, it is equal to 0. The execution time and energy consumption of task t_i are bounded by

$$\forall \text{task i} \quad ET_i = \sum_{r=1}^{s}\sum_{j=1}^{h} ET_{i,j,r}\cdot Q_{i,j,r} \quad (4)$$

$$\forall \text{task i} \quad Energy_i = \sum_{r=1}^{s}\sum_{j=1}^{h} Energy_{i,j,r}\cdot Q_{i,j,r} \quad (5)$$

Where $ET_{i,j,r}$ and $Energy_{i,j,r}$ denote the execution time and energy consumption of the task t_i with j^{th} L1 cache configuration and r ways of L2 cache, which have been recorded in the line 4 and 5.

Algorithm1: ILP based algorithm
Input: Set of core *P*, Set of task *T*, number of L1 cache configuration h, L2 cache way s, deadline D
Output: the optimal task mapping, the L1 cache configuration, the L2 cache partition factor.

```
1   for  t_i ∈ T do
2   for r=1 to s
3   for j=1 to h
4           ET_{i,j,r}=determine_ET(t_i,j,r)
5           Energy_{i,j,r}=determine_Energy(t_i,j,r)
6     end for
7   end for
8   end for
9   ILP_obj=setup_objective_function(T,Energy)
10  ILP = ILP ∪ ILP_obj
11  ILP_core=setup_core_constraint(T,P,C,WCET,Energy,D)
12  ILP = ILP ∪ ILP_core
13  ILP_cahe=Setup_cache_constraint(T,P,C)
14  ILP = ILP ∪ ILP_cahe
15  ILP_task_mapping=Setup_mapping_constraint(T,P,C)
16  ILP = ILP ∪ ILP_task_mapping
17  CPLEX_Solver(ILP)
```

Let $Start_i$ and End_i represent the starting time and the completion time of taskt_i, respectively. Obviously, Equation 6 and 7 must be hold. In addition, each task must be completed before its deadline. Equation 8 describes the detail of this constraint.

$$\forall task\ i\ End_i = Start_i + ET_i \quad (6)$$

$$Start_i \geq 0 \quad (7)$$

$$End_i \leq D \quad (8)$$

4.1.2 Task mapping constraint formulations

For task t_i and t_j , we need to indicate whether they are mapped to the same core p_k . A binary decision variable $Z_{i,j}$ is

defined to describe this relationship. Let variable $Z_{i,j} = 1$, if task t_i and t_j are mapped to same core p_k and 0 otherwise. In other words, only if variable $M_{i,k} = 1$ and $M_{j,k} = 1$, variable $Z_{i,j}$ is equal to 1. This constraint can be expressed as following.

$$\forall task\ i,j\ \forall core\ k\ \ M_{i,k} + M_{j,k} - Z_{i,j} \leq 1 \quad (9)$$

$$M_{i,k} + M_{j,k} - 2 \cdot Z_{i,j} \geq 0 \quad (10)$$

All tasks mapped to the same core do not overlap, In other words, they must not be executed on the same core at the same time. In order to guarantee the non-overlapping constraints, for each pair of task t_i and t_j, two binary variables $B_{i,j}$ and $B_{j,i}$ are defined. Let $B_{i,j}$ =0 if task t_i and task t_j are mapped to same core and task t_i execute before task t_j. Let $B_{j,i}$=0, if task t_i and task t_j are allocated to same core and task t_i execute after task t_j. Then, the following constraints must be hold. *C* is a larger constant.

$$\forall task\ i,j\ \ i \neq j, B_{i,j} + B_{j,i} - Z_{i,j} = 0 \quad (11)$$

$$Start_j \geq End_i - C \cdot (1 - B_{i,j}) \quad (12)$$

$$Start_i \geq End_j - C \cdot (1 - B_{j,i}) \quad (13)$$

4.1.3 Cache constraint formulations

We define a binary decision variable $C_{i,j}$, which is equal to 1 if task t_i is assigned j[th] L1 cache configuration and 0 otherwise. Each task can only be assigned one of h different L1 cache configurations. Therefore,

$$\forall task\ i\ \ \sum_{j=1}^{h} C_{i,j} = 1 \quad (14)$$

We define a binary decision variable $W_{j,r}$, which is equal to 1 if r ways of L2 cache (i.e. partition factor) is partitioned to core p_j and 0 otherwise. Each core can only use one partition factor.

$$\forall core\ j\ \ \sum_{r=0}^{s} W_{j,r} = 1 \quad (15)$$

The total sum of assigned ways of L2 cache for all cores cannot exceed available the number of way of L2 cache. Therefore,

$$\forall core\ j\ \ \sum_{p=0}^{s} p \cdot \sum_{r=0}^{s} W_{j,r} \leq s \quad (16)$$

Each task can only select one of h different L1 configurations and one of s different L2 cache factor.

$$\forall task\ i\ \ \sum_{r=1}^{s} \sum_{j=1}^{h} Q_{i,j,r} = 1 \quad (17)$$

For each task t_i, if it is mapped to core j and r ways of L2 cache is assigned to core j .the following constraints should be satisfied.

$$Q_{i,j,r} = \begin{cases} 1, & if\ C_{i,j} = 1\ and\ W_{j,r} = 1 \\ 0, & otherwise \end{cases} \quad (18)$$

The above euqation can easily be linearized, due to space limitions, this is omitted from this paper.

4.1.4 Objective function

Our objective is to minimize the cache energy consumption. Therefore, objective function is defined as:

$$Minimize\left(\sum_{i=1}^{n} Energy_i\right) \quad (19)$$

Solving the above ILP formulation, we can obtain the optimal task mapping, L1 configuration and L2 cache partition factors. The ILP-based approach can find an optimal solution, and it has little time overhead compared to exhaustive search method. But, as the number of tasks increases, ILP formulation may take long time to obtain an optimal solution. For example, when there are 8 tasks, it takes less than 14 seconds. When there are 13 tasks, it takes more than 5 minutes. To relieve this problem, we also develop an approach based GA which has high efficient and near-optimal solution for large number of task sets

4.2 Genetic algorithm for task mapping

Genetic algorithm is probabilistic search method, which simulates the Darwinian principles of natural selection and the survival of the fittest. They can solve a large number of complex optimization and design problems, and a lot of researches have proven that the GA is superior to many heuristic techniques available. Algorithm 2 illustrates the major steps of our genetic algorithm. In step 2, we first generate all feasible L2 partition schemes, and then GA will perform for each L2 partition scheme. In step 4, the initial population is filled with chromosomes that are generated at random. Step 6 evaluates the fitness value of each chromosome using the fitness measure in Equation 20. Step 8 to 19 generates the new population by selection, crossover, mutation, Step 21 tests whether the termination condition is reached. If so, the best chromosome in the current population is returned as our solution. If the termination condition is not satisfied, then a new generation is created by applying the genetic operators, this process is repeated until the termination condition is satisfied.Step 22 records the best solution for all L2 cache partitioning scheme. This is achieved by comparing the current solution with the latest solution and preserving the best one of the two solutions.

4.2.1 Generate initial random population

In this step, we create randomly an initial population. Each chromosome o_i in the population is constructed by two lists of genes Φ and Θ. The content in list Φ denotes the mapping of task to core, the content in list Θ represents the L1 cache configuration of each task (using the L1 configuration index instead of L1 configuration). Obviously, each chromosome is a solution for our problem. Figure 4 shows a chromosome. For each pair of genes of a task in figure 4, we can obtain its task mapping and L1 cache configuration. For example, task 2 is mapped to core 3 and use the 2th L1 cache configuration.

Task	1	2	3	4	5
(Φ)	C1	C3	C2	C1	C1
(Θ)	1	2	1	3	3

Figure 4. chromosome

4.2.2 Evaluation of each member of generation

In the population, all the chromosomes are evaluated and a fitness value is assigned to them. This fitness value reflects the ability of this chromosome to survive in current environment. The greater fitness, the higher is the probability

of survival of the chromosome during evolution. The fitness is calculated through the following fitness function.

$$fitness(o_i) = \begin{cases} \frac{1}{E(o_i)} & T(o_i) \le D \\ 0 & T(o_i) > D \end{cases} \quad (20)$$

Where o_i is a chromosome. $T(o_i)$ is schedule length, which are defined the maximum completion time of all cores in a chromosome , $E(o_i)$ is the sum of energy of all tasks in a chromosome, D is the deadline of task sets. In this fitness function, both the timing constraint $T(o_i)$ and the energy consumption $E(O_i)$ are considered. If the schedule length $T(o_i)$ is greater than the deadline D, the fitness value of o_i is equal to 0. This indicates chromosome o_i do not satisfy the timing constraints and is an infeasible scheme, otherwise, its fitness value is inversely proportional its total energy consumption $E(o_i)$.

Algorithm2: The Genes algorithm
Input: Set of core P, Set of task T, number of L1 cache configuration h, L2 cache way s, deadline D
Output: task mapping, L1 cache configuration, L2 cache partition factor.
1. Get the population size(**Ps**)
2. Generate all feasible L2 partition schemes.
3. **for** each schemes **do**
4. Generate initial random population.
5. **for** j =1 to Ps **do**
6. Evaluate the fitness value of chromosome based on Equation 20
7. **end** for
8. Sort all chromosomes in the ascending order of their fitness value.
9. Derive the average value of fitness Fit_{av} of all chromosomes.
10. Remove chromosomes whose fitness values are smaller than Fit_{avg} ,and
11. record R, which is the number of removed chromosomes
12. **for** j=1 to R **do**
13. Apply **algorithm3** on preserved chromosomes to generate the new chromosomes.
14. **end** for
15. Sort all chromosomes in the ascending order of their fitness value.
16. **for** j=2 to Ps **do**
17. Select a chromosome from current generation.
18. Perform **algorithm4** to generate new chromosomes.
19. **end** for
20. Let the current chromosomes be chromosomes in the next generation
21. **if** the termination criteria is satisfied
22. Record chromosomes with the biggest fitness value in current L2 partition scheme
23. **else**
24. Go to Step 7
25. **end** if
26. **end for**

4.2.3 Creation of new populations

This step is to form a new population by genetic operators from current population with the aim of finding better solutions, and the chromosome in new population is slightly different from the chromosome in current population. This is done by using three types of genetic operators as follows:

Selection: The selection operators are used to select chromosome through its fitness value from the current population so that further genetic manipulation. In the process of selection, all the chromosomes are given a probability of selection which is proportional to the fitness value assigned to the chromosome. The rationale is that the chromosome with higher fitness value should have a higher probability of surviving into the next generation. In our approach, a simple selection method, but quite effective is used from our experiment results. We sort all chromosomes in the ascending order of fitness value, then, remove chromosomes whose fitness value are smaller than the average of fitness of all chromosomes. In this way, we can guarantee that best chromosome is preserved

Crossover: The crossover operators are used to select genes from parent chromosomes and generate offspring. In selection stage, we have removed chromosomes whose fitness value is smaller than the average value of fitness values of all chromosomes. In this stage, we perform crossover on preserved chromosome to generate new chromosome. Note that the number of removed chromosomes is not same in each generation due to different average of fitness values of chromosomes. For each pair of chromosomes, the multi-dot crossover is applied. We generate random number q for all crossover points i from 1 to n, if q is equal to 0, then swap the genes corresponding to crossover points of two parent chromosomes to create new chromosomes. Algorithm3 shows

Algorithm3: Crossover Operator
Input : Parent $P_1[1..n]$, Parent $P_2[1..n]$ // n is the number of tasks
Output Children $C_1[1..n]$, Children $C_2[1..n]$
1. p=random[0,1] //generate random number between 0 and 1
2. **if** (p> p_c) // p_c is crossover probability
3. **for** i=1 to n **do**
4. q= random{0,1} //generate random number 0 or 1
5. **if** (q==0) // p_c is crossover probabilit
6. $C_1[i]= P_2[i]$ // copy mapping task and L1 cache configuration in parent chromosome
7. $C_2[i]= P_1[i]$
8. else
9. $C_1[i]= P_1[i]$
10. $C_2[i]= P_2[i]$
11. **end** if
12. **end** for
13. return
14. **end** if

the procedure of the crossover operator, which takes two parents as input and produces two children, and guarantees that if parent chromosomes are feasible then their children chromosomes are also feasible.

Mutation: The mutation operator is used to help to search beyond local optima by randomly changing allele values of some genes. It creates new chances for finding the optimal solution. In our approach, we perform mutation by randomly selecting one task for each chromosome and change its L1 cache configuration and mapping of it to core. Algorithm 4 shows the procedure of the mutation operator.

Algorithm4: Mutation Operator
Input : chromosomes P[1..n]
Output :chromosomes P[1..n]
1. i=random[1,n] //generate random mutation point between 1 and n
2. p= random[0,1] //generate random number between 0 and 1
3. **If** (p> m_c) // m_c is crossover probability
4. Assign P[i]'s the gene of task mapping at random in feasible mapping
5. Assign P[i]'s the gene of L1 cache configuration at random in feasible L1 configuration
6. **end** if

4.2.4 Termination conditions check

In order to terminate the algorithm, a finite number of iterations are defined in advance as termination condition. If the termination condition is met, the near-optimal chromosome is returned in population. It should be mentioned that algorithm may converge prior to defining the number of iterations. In this case, the near-optimal chromosome (lowest

energy consumption) in the current population is returned as the optimal solution.

5 Experimental Evaluation

5.1 Experimental setup

To evaluate our approaches, we use 18 benchmarks from MRTC [18] in our experiment. These benchmarks have different combinations listed in Table 1. We reduce input sets of several benchmarks. Since these benchmarks with larger input sets has an excessive execution time absolutely dominating the deadline of task sets. The deadline D is set by using the same way with [5] .i.e. it is a feasible L1 cache assignment for every partition factor in each core.

We use the SimpleScalar cycle-accurate architectural simulation platform [25].The cache model of which is modified to support way-based L2 cache partitioning. In this work, we use four core and two levels of caches architecture,

Table 1 Task sets consisting of real benchmarks

Groups	# tasks	Tasks
Set1	8	*expint,qurt,prime,jfdcint, insertsort,bs,fir,bssort100*
Set2	9	*expint,qurt,prime,jfdcint, insertsort,bs,fir,bssort100,fibcall*
Set3	10	*st,fdct, prime,jfdcint, insertsort,fir fibcall,crc,cnt,select*
Set4	11	*crc,cnt,expint,qurt,prime,jfdcint, insertsort,bs,fir, fiball,fdct*
Set5	12	*fdct, expint,qurt,prime,jfdcint, insertsort, bs,fir,bssort100,fibcall,crc,cnt*
Set6	13	*qurt,jfdcint,insertsort,bssort100,crc,cnt, select,ndes,ud,lms,matmult,st,fdct*

where every core runs at 1GHz. The private L1 cache has a base size of 256B. The shared L2 cache configuration is assumed to be 4KB, 8-way associative with 32-byte lines. The access latencies of L1 cache, L2 cache, and the main memory are set to 1, 6, and 20 cycles, respectively, and the number of cache accesses and misses, as well as the execution time are obtained under different L1 cache configurations and way based L2 partition. We collect the energy parameters of the cache from CACTI [21] with 65 nm technology. We perform the experiments on 3GHz processors with 4GB memory and use CPLEX12.2 as ILP solver for our ILP approach [22].

5.2 Experiment results

In this section, we compare four approaches as follows:

1. ECP: Task mapping is determined by the minimal schedule length. L1 Cache use base configuration and L2 cache is equally partitioned among the different core. We use it as the baseline.
2. ODCR+OCP: Task mapping is determined by the minimal schedule length. L1 DCR and L2 CP are optimal based on exhaustive search method.
3. Our approach: ILP based approach and GA based approach.

The simulation of genetic algorithm for all task sets use the following parameters, population size (Ps) = 16, crossover probability (p_c) = 0.6, mutation probability (p_m) = 0.32, maximum number of iterations = 1200. Our experiments use the following L1 base configuration: 256B with 2-way associativity and 16-byte line size (256B_2W_16).

Figure 5 shows the energy consumption for ODCR+OCP, ILP, GA normalized to ECP. Compared to the ODCR+OCP, our ILP can achieve 10.6% energy savings on average and GA can reach 9.4% energy savings.

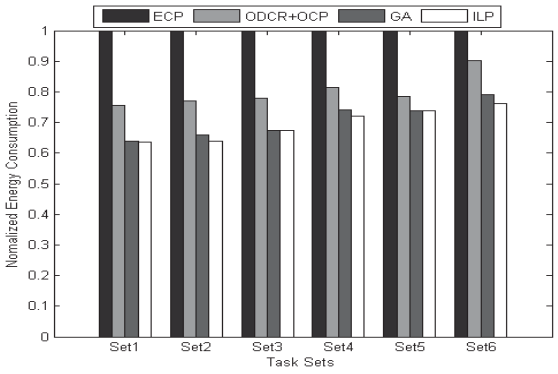


Figure 5. Cache energy saving for ODCR+OCP, ILP, GA normalized to ECP

5.3 Effect of deadline

In the section, we conduct experiments to show the effect of deadline. Using the same example above, we vary the deadline from 200 μs to 165 μs in step of 5μs. Figure 6 shows the result for ODCR+OCP, ILP and GA. It is observed that a decrease in deadline results in increase in energy consumption of all approaches. However, we can also observe that energy consumption of ODCR+OCP do not change from 200μs to 180μs, and then strongly increase with deadline decreasing due to fixed task mapping. On the contrary,energy consumption based on ILP and GA slightly increases from 200μs to 165μs by proper task mapping.

5.4 Time overhead

We cannot simply conclude that our approaches outperform ODCR+OCP since they only use the task mapping based on the minimal schedule length. However, when they explore the whole solution space of task-mapping for reducing energy consumption, it has significantly time overhead. In this subsection, we compare the time overhead of our approaches with ODCR+OCP considering all possible task mapping, we exhaustively test all possible task mappings and invoke the ODCR+OCP algorithm. To achieve a fair comparison, ODCR+OCP only need to find the task mapping solution which was obtained by our ILP. In other words, we first perform our approach to obtain the optimal task mapping. Time overhead for ODCR+OCP is to find this optimal task mapping from the whole solution space. Note that for the same task mapping, three approaches can find an optimal L1 cache configuration and L2 cache partition factor due to small solution space of L1 cache configuration and L2 cache partition. Table 2 shows the time overheads for three approaches. We can see that the time increases exponentially for ODCR+OCP with number of tasks. Whereas the runtime

of ILP based approach is within 347 seconds, and the runtime of GA based approach is within 5 seconds.

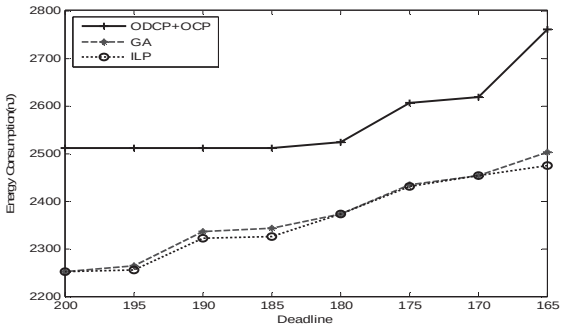


Figure 6. Deadline effect on total energy consumption

Table 2 Time overhead comparison

Task Sets	ILP based approach	GA based approach	ODCR+OCP approach
Set1	0.54s	1.69s	0.13s
Set2	2.96s	1.97s	53.68s
Set3	6.47s	2.01s	411.30s
Set4	258.68s	4.13s	1238.61s
Set5	283.16s	3.89s	4891.85s
Set6	346.27s	4.91s	12517.43s

6 .Conclusion

This paper focuses on problem of task mapping for energy optimization on multi-core system. The goal is to find the optimal task mapping, L1 cache configuration and L2 cache partition factor. We present two approaches to solve this problem. The ILP based approach can find the optimal solution. The GA based approach is near-optimal compared to ODCR+OCP. Meanwhile, two approaches are more efficient in runtime compared to an exhaustive task mapping, which invoke ODCR+OCP to produce optimal solution.

Acknowledgment

We are grateful to all the members of the Scalable Computing Lab research group of Beijing Institute of Technology for their contributions. We also thank the anonymous reviewers. This work is supported by the National Science Foundation of China (Grant No. 61370062).

References

[1] ARM Cortex-A15 serious. http://www.arm.com/products.
[2] MIPS, 2010. http://www.mips.com/
[3] A.Malik etal.,A low power unified cache architecture providing power and performance flexibility,ISLPED, 2000.
[4] Gang Chen, Kai Huang et.al. Cache Partitioning and Scheduling for Energy Optimization of Real-Time MPSoCs , 2013.
[5]W.Wang,P.Mishra,andS.Ranka.Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems,DAC, 2011
[6] P. Bentley, Evolutionary Design by Computers, Morgan Kaufmann, 1999.
[7]M.Modarressi,S.Hessabi,andM.Goudarzi. A reconfigurable cache architecture for object-oriented embedded systems. CCECE, 2006.
[8]W.Wang et al.,Dynamic cache reconfiguration for soft real-time systems, ACM TECS, 2011.
[9]W.Wang and P. Mishra, Dynamic reconfiguration of two-level cache hierarchy in real-time embedded systems, J of Low Power Electron, vol. 7, no 1, pages 28-38 , 2011.
[10]Yanbing Li and Wayne Wolf, A Task-Level Hierarchical Memory Model for System Synthesis of Multiprocessors, DAC, 1997.
[11]John M. Calandrino and James H. Anderson, Cache-Aware Real-Time Scheduling on Multicore Platforms:Heuristics and a Case Study. In Proceedings of 2008 Euromicro Conferenceon Real-Time Systems (ECRTS08), 2008.
[12]A.Fedorova, M. Seltzer, M.D. Smith, Cache-fair thread scheduling for multicore processors, Technical Report TR-17-06, Division of Engineering and Applied Sciences, Harvard University, 2006.
[13]S.-H.Yang, B. Falsafi, M. D. Powell, K. Roy, and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep submicron high-performance i-caches,HPCA, 2001.
[14]MiBenc Benchmark http://www.eecs.umich.edu/mibench/
[15] R. Reddy and P. Petrov. Cache partitioning for energy-efficient and interference-free embedded multitasking. ACM Transactions on Embedded Computing Systems, Mar. 2010
[16] S. Kim et al., Fair cache sharing and partitioning in a chip multiprocessor architecture, PACT, 2004.
[17] X. Fu, K. Kabir, and Xiaorui. Cache-aware utilization control for energy efficiency in multi-core real-time systems. ECRTS, 2011.
[18] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET benchmarks – past, present and future. In WCET, 2010.
[19] T. Liu, Y. Zhao, M. Li, C.J. Xue, Task assignment with cache partitioning and locking for wcet minimization on mpsoc, in: Proceedings of the 39th International Conference on Parallel Processing, ICPP 2010.
[20] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache for low energy embedded systems.ACM Transactions on Embedded Computing Systems, 2005.
[21] CACTI. http://www.hpl.hp.com/research/cacti
[22] IBM ILOG CPLEX. http://www.ibm.com/software/
[23] B. D. Bui, M. Caccamo, L. Sha, and J. Martinez. Impact of cache partitioning on multi-tasking real time embedded systems. RTCSA,2008.
[24] R. Reddy and P. Petrov. Cache partitioning for energy-efficient and interference-free embedded multitasking, ACM Transactions on Embedded Computing Systems, 2010.
[25] SimpleScalar LLC. http://www.simplescalar.com
[26] G.E. Suh, L. Rudolph, S. Devadas, Dynamic partitioning of shared cachememory, Journal of Supercomputing, vol 2, no 8, pages 7–26, 2004.
[27] Sascha Plazar, Paul Lokuciejewski Peter Marwedel, WCET-aware software based on cache partitioning for multi-task real-time systems, 2009.
[28] B. D. Bui, M. Caccamo, L. Sha, and J. Martinez. Impact of cache partitioning on multi-tasking real time embedded systems, RTCSA, 2008.

SESSION

LATE BREAKING PAPERS: EMBEDDED SYSTEMS AND ENERGY MANAGEMENT METHODS

Chair(s)

TBA

A Comparative Study on the Energy Efficiency of 4th Gen Intel ® Core ™ Processor vs 3rd Gen Intel ® Core ™ Processor

Siti Nur Diana Muhd Azmi
School of Computing, Creative Technologies, and Engineering, Leeds Beckett University, Leeds, UK
s.muhdazmi2847@student.leedsbeckett.ac.uk

Nazarudin Bujang
Intel Kulim, Perak
Malaysia

Ah-Lian Kor
School of Computing, Creative Technologies, and Engineering, Leeds Beckett University, Leeds, UK
A.Kor@leedsbeckett.ac.uk

Colin Pattinson
School of Computing, Creative Technologies, and Engineering, Leeds Beckett University, Leeds, UK
C.Pattinson@leedsbeckett.ac.uk

Abstract—This research aims to compare the energy efficiency in between two generations Intel processors; the 4th Gen Intel ® Core ™ Processor and 3rd Gen Intel ® Core ™ Processor. It also surveys the technologies that provide better energy performance for both of the processors. The methodology used for this research is a physical experiment conducted in an Intel production plant. The results obtained from the experiment show that the 4th Gen Intel ® Core ™ Processor is more energy efficient than the 3rd Gen Intel ® Core ™ Processor.

Keywords—energy consumption, energy efficiency, Intel Processor, 3rd Gen Core Processor, 4th Gen Core Processor

I. INTRODUCTION

Statistics has shown an increasing trend of ICT use and its growth rate could surpass that of the aviation industry. Consequently, the ICT-related energy use is comparable to that of the aviation industry (UK Parliamentary Office of Science and Technology (2008)[1]. ICT's substantial energy consumption has a significant impact on GHG emissions and climate change where 2% of global carbon emissions come from manufacturing and using of Information and Communication Technology (ICT)[2]. In Europe, ICT equipment and services account for 2.5%-4% for EU's carbon emissions[3]. According to the Smart2020[4] report by the Global e-Sustainability Initiative, GeSI (2008), the ICT sector's emissions are expected to increase, from 0.53 billion tonnes (Gt) carbon dioxide equivalent (CO_2e) in 2002 to 1.43 $GtCO_2e$ in 2020 (in Business As Usual, BAU, scenario). Figure 1 shows the estimated distribution of global CO_2emission for ICTs. The main contributors are PCs and monitors (40%), telecommunications (31%), followed by data centres (23%).

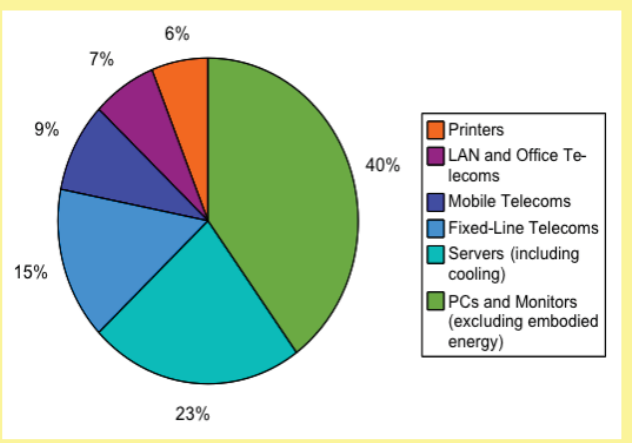
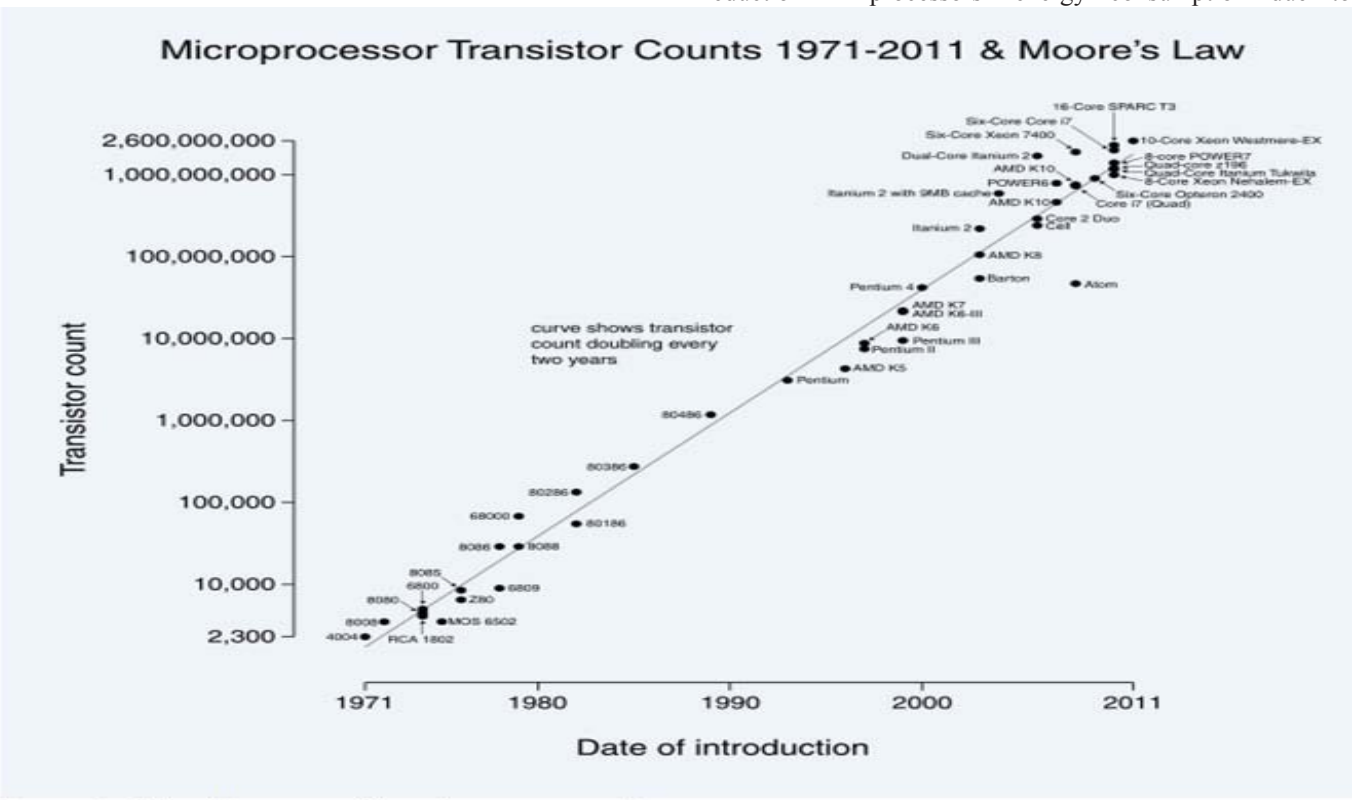


Figure 1: Estimated Distribution of GlobalCO_2 emission for ICTs (extracted from ITU,2009, p.4)[5]

ICTs play a significant role to limit and reduce GHG emissions. According to the SMART2020 Report[6] there is scope for reducing the carbon footprint of the ICT sector by approximately 36% by 2020 (equivalent 770 Mt CO_2eq) using existing technologies. There are two ways to mitigate ICT impact on climate change[7]. The first is a direct mitigation which reduces the ICT sector's own carbon emissions and energy requirements while the second concerns the exploitation of ICT for offering solutions to reduce the carbon footprint of other sectors and to facilitate efficient and low carbon development. Based on the SMART2020 and SMARTer2020[8] Reports, employing ICT-driven efficiency across the economy will deliver emission savings. The latter demonstrates how the increased use of ICT could reduce the projected 2020 global greenhouse gas (GHG) emissions by 16.5% (equivalent to 9.1 $GtCO_2e$) and this is more than seven times the ICT sector's emissions in the same period.

Murugesan's (2013) [9] definition of Green IT is environment sustainability-focused. It refers to environmentally friendly computer, information systems, applications, and practices which aim to improve energy efficiency, lower GHG emissions, use of less toxic materials, encouraging reuse and recycling. **Greening of IT** aims to mitigate the environmental impact of ICT itself. This encompasses energy efficient and environmental sustainable designs, operations, use and disposal ICT equipment, infrastructure and systems.



Figure·2:·°The·history·of·Intel·processors¶

Since 2012, Intel has set their 2020 environmental goals[10] which aim to reduce greenhouse gas emission and increase energy efficiency of their products and operations. Consequently, Intel has developed many energy-efficiency products through their product innovation. Additionally, they prioritise the minimisation of their products' environmental footprint throughout their entire life cycle. Intel (2015) is an innovation leader and has come up with innovative technology and products in every two years following Moore's Law[11] (see Figure 2) which states that computing would dramatically increase in power, and decrease in relative cost, at an exponential pace.

In this research, we shall examine and compare the energy efficiency in between 3rd and 4th Gen Intel ® Core ™ Processor. The energy consumption during standby and active mode of both processors are measured and analysed. The Ivy Bridge (Figure 3) 3rd generation processor is an enhanced version of Sandy Bridge[12] 2nd generation processor while Haswell (Figure 6) is the 4th generation processor.

Rationale

Even though Intel continually innovate their product architecture design, they ensure the innovation does not compromise with the performance and energy efficiency of new microprocessors. Hence, this study investigates the reduction in processors' energy consumption due to innovative architecture design.

Based on Intel's microprocessors history, their first microprocessor was introduced to market in 1971 (Intel 4004) and the latest recently launched microprocessor by Intel is the 5th generation processor, with the codename, skylake. Intel has continuously improved on the processors' clock speed and sizes where it ranges from 10 micron for Intel 4004 up to 22-nanometer for Intel 4th generation processor and finally, 14-nanometer for recently launched 5th generation processor [13].

Since energy efficiency is one of the greatest issues in ICT and computer application in dealing with environmental issues. Lower energy consumption could help to reduce carbon footprint. Producing environmental friendly products could give Intel a competitive advantage.

Aim and objectives

The aim of this research is to conduct an investigation on the energy efficiency of the 4th Gen Intel ® Core ™ and 3rd Gen Intel ® Core ™ Processors. The following is a set of objectives to help achieve this aim.

i. To critically review literature on Intel ® Core ™ Processors, their performances and energy efficiency;
ii. To conduct experiments to investigate the energy consumption for both 3rd and 4th Intel ® Core ™

Processors in the following states: active and on standby;

iii. To draw a comparison between 3rd and 4th Intel ® Core ™ Processors using the following parameters: performance (based on document review); and energy consumption (states: active and on standby).

II. LITERATURE REVIEW

A. 3rd Gen Intel ® Core ™ Processor (Ivy Bridge)

Figure 3: 3rd Generation Intel ® Core ™ Processor (Ivy Bridge)

According to Intel, the 3rd generation Intel® Core™ processor, is featured with smart technologies that allow users to exploit it to meet their needs. It also incorporates incredible visual built-in for visual enhancement where users do not need additional graphics card or software to experience a brilliant visual. Intel® HyperThreading Technology1 further improves the performance and multi-tasking capability to spped up workflow. In addition, the 3rd generation Intel® Core™ processor improves energy consumption and efficiency[14].

Officially launched in April 2012, Ivy Bridge is the codename for the 3rd Generation Intel ® Core ™ Processor and is the successor to the Sandy Bridge the 2nd Generation Intel ® Core ™ Processor. As a record, Ivy bridge or 3rd Generation Intel ® Core ™ Processor carry 1.4 billion transistors on the chip compared to Sandy Bridge the 2nd Generation Intel ® Core ™ Processor carry 1.16 billion. Ivy Bridge has a microarchitecture (see Figure 5) on a processing die which shrinks from 32nm to 22nm. Ivy Bridge is the first processor in the Intel family with a 22 nm logic technology microprocessor that uses first high-volume chip called Tri-Gate technology that provides significant processing performance [15]. Tri-Gate technology allows power consumption reduction and die size. Tri-Gate technology is the world's first 3-D transistor. According to National Instrument [16], it is a technology that could boost the performance by up to 37 percent compared to 32 nm planar transistors, the traditional two-dimensional. Subsequently, power reduction and energy usage by the chips on board are made possible due to lower voltage and lower leakage. However, according to Intel, 3-D planar could perform at the same same level as 2-D planar transistors with a reduction of 50 per cent power consumption. Intel's 3-D Tri-Gate transistor (see Figure 4) employs three gates where, a single gate crosses on top of the other two vertical gates which then form all three sides. This formation allows electrons to travel three times on the surface area that could give benefits to less power consumption and greater current flow due to leakage reduction effect from current control. This new transistor designed by Intel brings about ultra-low power benefits for handheld devices.

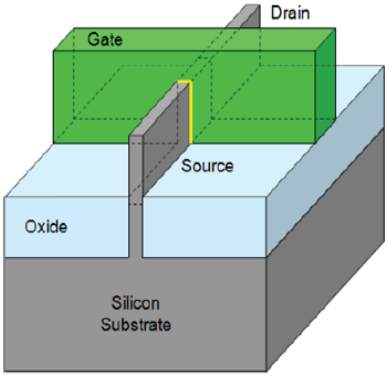


Figure 4: 3D Tri-Gate transistors (Intel, 2015)

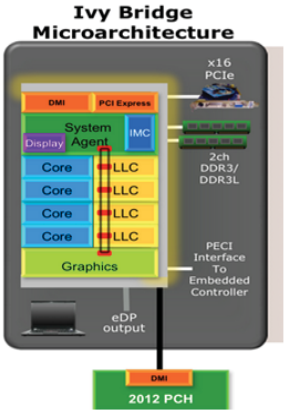


Figure 5: Ivy Bridge Graphics and Media Microarchitecture Overview

B. 4th Gen Intel ® Core ™ Processor (Haswell)

Haswell is the Codename for 4th Generation Intel® Core™ processor, a replacement to the Ivy Bridge. It is released in 2013 and Haswell is the extension of Intel advanced version of 22nm Tri-gate process technology [17-18]. Haswell is a combination of few building blocks; CPU, memory platform controller hubs (PCHs) and graphics and media processing engines that could create high-performance application. In addition, several integrated technologies: FIVR – 5 platform consolidated to 1; graphic performance improvement by on-die eDRAM cache; optimized IO interfaces; lower-power states; 256b SIMD integer and an Intel AVX2 instruction set. Haswell is optimized by the 22nm process to reduce leakage by 75% at V_{min} and also reduce power consumption[17]. Low power to enable smaller form factors and platform integration are the main objectives for Haswell.

The key comparisons between the 3rd and 4th gen processors are[19]: the latter is the first System on Chip (SoC) which integrates all major building blocks for a system onto a single chip; enhanced battery life for the latter (9.1 hours compared to 6 hours for HD video; 10-13 days on standby power compared to 4.5 days); graphiocs performance on the 4th gen processor doubles its predecessor's; enhancement of power-performance efficiency. There are few techniques that have been employed by Intel to improve power-performance efficiency[20]. Firstly, low-level implementation which involves the optimization of manufacturing, process technology and circuits, optimize microarchitecture and algorithms and finally optimization of

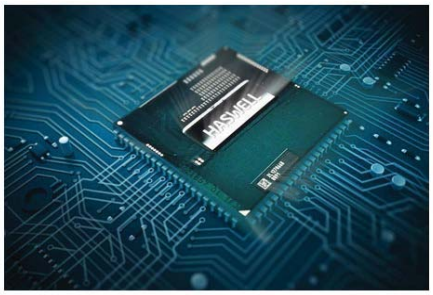


Figure 6: 4th Generation Intel ® Core ™ Processor (Haswell)

and Analysis of Experiment" – 'experimentation is a vital part of the scientific (or engineering) method' (p.2) and this

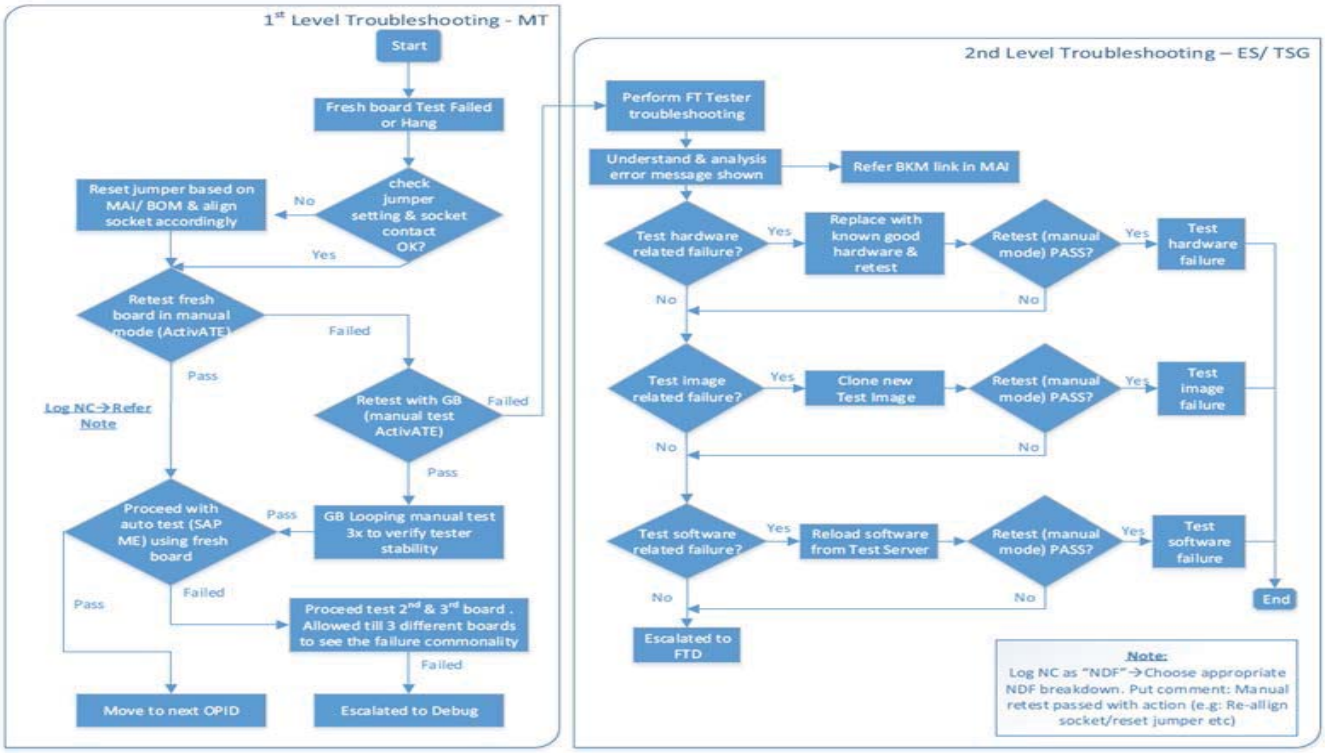


Figure 7: Functional Tester Troubleshooting Flow (Intel, 2015)

design and implementation. Secondly, high-level architecture improvement in Haswell encompasses the use of independent voltage-frequency domains (e.g. cores, caches, graphics, system agent, etc…) which run on dedicated, and individually controlled voltage-frequency points. In order to maximize performance, a power control unit (PCU) dynamically allocates power budget among these domains. Lastly for the platform power management operation, Intel has improved battery life by introducing new active idle-power state, S0ix that could deliver 20 times improvements in idle-power compared to 3rd gen processors supported by fully integrated voltage regulator (FIVR) [18, 20].

III. METHODOLOGY

"Experimentation and data collection are the tools of science for validating theories" [21]. This method is chosen as according to Montgomery [22] in the book title "Design

is the best method to evaluate a system performance.

A well-design experiment is crucial as the method selected could affect the end result and conclusion drawn. Therefore, these experiments were carried out at Intel Kulim, Malaysia production plant (KM1 and KM2) as they could provide appropriate tools and equipment for reliability and high level of control. The experiment was conducted from 26th January 2015 to 4th February 2015 by a certified and highly qualified Intel's operator due to safety purposes and high precision requirements of the experiments. The entire experiment was closely observed by the student. Experiments were conducted to examine and investigate the energy consumption for both 3rd and 4th Gen Intel ® Core ™ Processor during active and standby mode. However, in order to obtain the energy consumption for both processors, a series of tests had to be executed to verify and ensure all the components perform well.

Data Point	HSW (4th Gen CPU) Energy (KJ)	IVB (3rd Gen CPU) Energy (KJ)	Energy Reduction (KJ)	% Increased Energy Efficiency
1	1.968	3.168	1.200	37.88%
2	1.872	3.264	1.392	42.65%
3	1.776	3.024	1.248	41.27%
4	1.632	3.600	1.968	54.67%
5	1.584	3.216	1.632	50.75%
6	1.872	3.168	1.296	40.91%
7	1.824	3.168	1.344	42.42%
8	1.584	3.360	1.776	52.86%
9	1.824	3.456	1.632	47.22%
10	1.824	3.072	1.248	40.63%
11	1.920	3.744	1.824	48.72%
12	1.776	3.168	1.392	43.94%
13	1.872	3.504	1.632	46.58%
14	1.536	3.600	2.064	57.33%
15	1.776	3.648	1.872	51.32%
16	1.728	3.072	1.344	43.75%
17	1.632	3.216	1.584	49.25%
18	1.632	3.264	1.632	50.00%
19	1.776	3.120	1.344	43.08%
20	1.680	3.552	1.872	52.70%

Table 1: Energy Consumption for 3rd and 4th Gen Intel ® Core ™ Processor during standby mode

The first test undertaken was 'Functional Test Level 1 and 2'. These procedures were carried out to verify and ensure the motherboard and each of the components were functioning in accordance to product specification. These tests could be conducted using few methods which are; Auto Test Process with Activated Test Platform (APSE) (Host Control System (HSC) + Scan Terminal (ScanTer)) or Manual Test with APSE or Manual Test Process with PQIUHC. The entire process is depicted in Figure 7. The method conducted during test was Auto test Process with APSE (HSC) + ScanTer and the operating procedures are as follows:

i. Motherboard was placed properly on the Base Plate / Carrier Plate of the (Standardized Test Hook Interface (STHI);
ii. Motherboard was clamped with STHI Back Plane / Control System. CPU, Memory cards and other testing peripherals were also placed on the boards according to operating manual provided;
iii. All the cards installed and toggle clamper was in clamping down position. At this stage, it must be ensured that the card was properly fully clamped down which enabled proper contact;
iv. Next motherboard to be tested was removed from the pick location of the incoming trolley;
v. Unit Under test (UUT) was examined for defects according to the Quality manual requirement;
vi. UUT was then placed on the test station and tested following the Motherboard testing procedure;
vii. Control system powered on and waited until the control system was fully initialized (i.e. until LED became green);
viii. Location Barcode on the ASTF (At Speed Test Fixture) tester and motherboard serial number base were scanned into HSC system which then triggered the APSE/HSC software to execute the test automatically;
ix. Screen for the UUT observed and on-screen instruction was followed;
x. Test was completed by displaying a green screen with prominent "PASS" message on the station monitor. Power OFF the UUT and control system MBPS3 switches. At this stage, it was ensured that the control system was properly power OFF to avoid APSE BLT content going missing;
xi. Waited until 5V stand by totally off (normally until the green LED light disappeared);
xii. The motherboard moved from the station to the passed conveyor or indicated trolley and ready for the next test.

The following test to measure processors' power consumption could not be done right away after the functional test had completed. The temperature of the board needed to be released (until room temperature) as the heat dissipated during previous test could affect the result obtained.

Data Point	HSW (4th Gen CPU) Energy (KJ)	IVB (3rd Gen CPU) Energy (KJ)	Energy Reduction (KJ)	% Increased Energy Efficiency
1	**1.872**	**5.856**	3.984	68.03%
2	**1.872**	**5.52**	3.648	66.09%
3	**1.872**	**5.04**	3.168	62.86%
4	**1.584**	**6.528**	4.944	75.74%
5	**1.488**	**6.672**	5.184	77.70%
6	**1.92**	**7.584**	5.664	74.68%
7	**1.584**	**5.088**	3.504	68.87%
8	**1.968**	**7.824**	5.856	74.85%
9	**1.776**	**7.344**	5.568	75.82%
10	**1.68**	**5.568**	3.888	69.83%
11	**1.92**	**6.144**	4.224	68.75%
12	**1.824**	**7.872**	6.048	76.83%
13	**2.304**	**6.096**	3.792	62.20%
14	**1.632**	**4.848**	3.216	66.34%
15	**1.728**	**6.192**	4.464	72.09%
16	**1.92**	**5.088**	3.168	62.26%
17	**1.92**	**4.992**	3.072	61.54%
18	**1.824**	**6.384**	4.56	71.43%
19	**1.824**	**5.664**	3.84	67.80%
20	**1.632**	**6.144**	4.512	73.44%

Table 2: Energy Consumption for 3rd and 4th Gen Intel ® Core ™ Processor during active mode

Processor power consumption test for 3rd and 4th Gen was executed at Intel KM1 production plant. 20 units of processor from each generation were tested separately in 2 different modes (standby and active) in order to gain more precise and accurate result.

The processor power consumption test operating procedures are as follows:

i. 20 units of processor were placed properly on the Base Plate / Carrier Plate of the (Standardized Test Hook Interface (STHI);
ii. UUT then placed on the test station and tested following the processor power testing procedure;

iii. Control system powered on and waited until the control system fully initialized (normally until LED became green);
iv. Location Barcode on the ASTF (At Speed Test Fixture) tester and processors Serial Number base scanned into HSC system followed by an automatic test by the APSE/HSC software;
v. Screen for the UUT observed and on-screen instruction followed;
vi. Test completed by displaying a green screen with prominent "PASS" message on the station monitor. Power OFF the UUT and control system MBPS3 switches. At this stage, it was ensured that the control system was properly power OFF to avoid APSE BLT content going missing;
vii. Waited until 5V stand by totally off until the green LED light disappeared;
viii. The processors moved from the station to the passed indicated trolley and ready for next test;
ix. Data from the test was then generated and transferred to the station monitor.

IV. Results and Discussion

A. Standby Mode

Table 1 below shows the energy reduction from the 3rd generation to the 4th generation range between 1.2KJ and 2.064 KJ in the standby mode.

a. Standby Mode

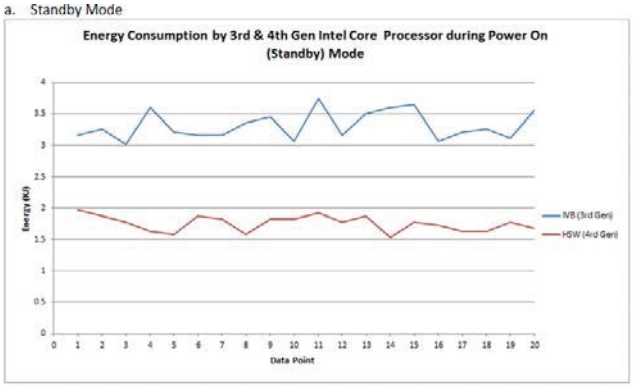


Figure 8: Energy consumption 3rd and 4th Gen Intel ® Core ™ Processor during standby mode

Figure 8 shows the energy consumption for both 3rd and 4th Gen Intel ® Core ™ Processor during standby mode. It shows that the 3rd Gen Intel ® Core ™ Processor consumes more energy compared to 4th Gen Intel ® Core ™ Processor during standby mode.

B. Active Mode

Table 2 shows the energy reduction for the 4th generation processor and it ranges from 3.168KJ to 4.944 KJ in the active mode.

Figure 9 shows the energy consumption for both 3rd and 4th Gen Intel ® Core ™ Processor during active mode. It shows that the 3rd Gen Intel ® Core ™ Processor consumes more energy compared to 4th Gen Intel ® Core ™ Processor during active mode.

Figure 10 shows the energy efficiency between 3rd and 4th Gen Intel ® Core ™ Processor during standby mode during processor standby mode. It shows that the efficiencies range from 40.63% to 57.33% during standby mode.

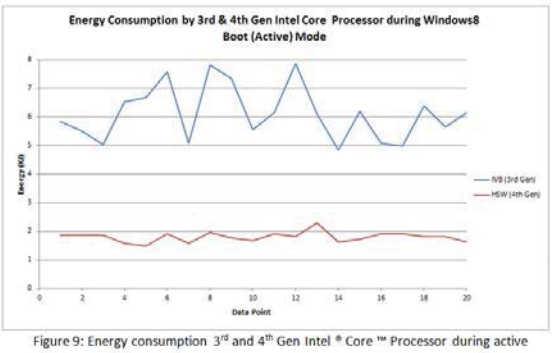


Figure 9: Energy consumption 3rd and 4th Gen Intel ® Core ™ Processor during active

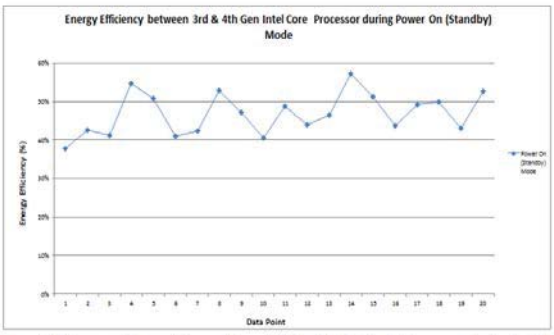


Figure 10: Energy efficiency between 3rd and 4th Gen Intel ® Core ™ Processor during standby mode

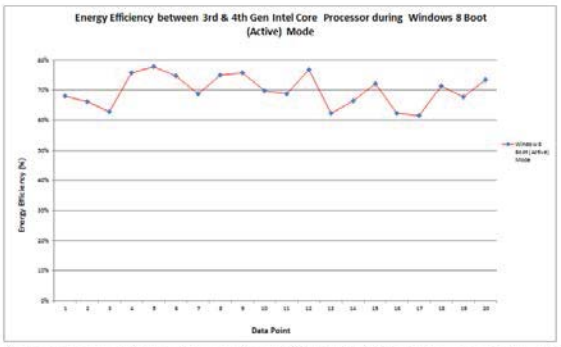


Figure 11: Energy efficiency between 3rd and 4th Gen Intel ® Core ™ Processor during active mode

Figure 11 shows the energy efficiency between 3rd and 4th Gen Intel ® Core ™ Processor during active mode. It shows that the efficiencies range from 61.54% to 77.7% during active mode.

C. T-Test

T-Test is the experimental design to compare mean or average values of two groups (are 4th Gen and 3rd Gen processor). T-Test version employed in this study is an independent-mean t-test because the experimental conditions for the two groups are independent of each other. Table 3 below shows the value of a two-tailed t-test with repeated measures (at confidence level, $\alpha = 0.05$) for the energy consumption of the 3rd and 4th Gen Intel ® Core ™ Processors (N=20) during the standby mode. The result of the analysis shows that the difference between the energy consumption of the two types of processors during standby mode is significantly different.

Repeated	N	df	Mean	SD	t-value	significance
HSW (4th Gen CPU)	20	19	1.754	0.123	27.768	Significant
IVB (3rd Gen CPU)			3.319	0.220		

Table 3: Two-tailed test (α = 0.05): Power On (Standby) Mode

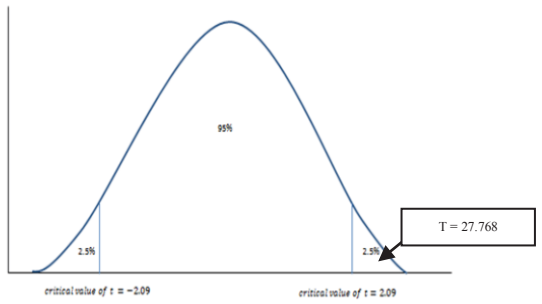


Figure 12: Two-tailed test (α = 0.05): Power On (Standby) Mode

Table 4 below shows the value of a two-tailed t-test with repeated measures (at confidence level, α = 0.05) for the energy consumption of the 3rd and 4th Gen Intel ® Core ™ Processors (N=20) during the active mode. The result of the analysis shows that the difference between the energy consumption of the two types of processors during active mode is significantly different.

Repeated	N	df	Mean	SD	t-value	Significance
HSW (4th Gen CPU)	20	19	1.807	0.181	19.873	Significant
IVB (3rd Gen CPU)			6.122	0.954		

Table 4: Two-tailed test (α = 0.05): Windows8 Boot (Active) Mode

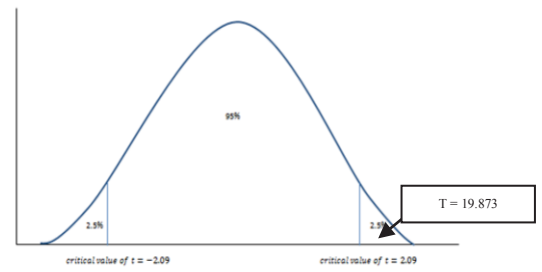


Figure 13: Two-tailed test: Windows8 Boot (Active) Mode

V. CONCLUSIONS

The study aims to make comparisons of energy efficiency for the 3rd Gen Intel ® Core ™ Processor and 4th Gen Intel ® Core ™ Processor. It reveals that the 4th Gen Intel ® Core ™ Processor consumes less energy than the 3rd Gen Intel ® Core ™ Processor. The feature that distinguishes both processors is the Fully Integrated Voltage Regulator (FIVR) which helps the 4th Gen Intel ® Core ™ Processor reduce energy consumption by the CPU and subsequently lower its temperature. Other advanced technology in the 4th Gen Intel ® Core ™ Processor helps to accelerate the performance which results in the reduction of energy consumption as it helps users to speed up their application and reduce execution time. Fully Integrated Voltage Regulator (FIVR), the innovative technology for the 4th Gen Intel ® Core ™ Processor obviously helps 4th Gen Intel ® Core ™ Processor to save energy.

The fact that the 4th Gen Intel ® Core ™ Processor is more energy efficient than the 3rd Gen Intel ® Core ™ Processor shows that product architecture design innovations by Intel aims to boost up performance as well as decrease energy consumption. However, Ivy Bridge, the 3rd Gen Intel ® Core ™ Processor may not completely phase out from the market in the near future as the 4th Gen Intel ® Core ™ Processor seems to be more suited for high end systems that require very powerful technologies.

Acknowledgment

A special thanks to Mr. Nazarudin Bujang and team from Intel for the knowledge sharing, guidance and assistance during Diana's data collection at their production plant in Intel Kulim, Perak, Malaysia.

References

[1]. http://www.parliament.uk/documents/post/postpn319.pdf
[2]. http://globalactionplan.org.uk/sites/gap/files/Green%20ICT%20Handbook.pdf and http://www.gartner.com/newsroom/id/503867
[3]. http://ec.europa.eu/digital-agenda/en/pillar-vii-ict-enabled-benefits-eu-society/action-69-assess-whether-ict-sector-has-complied-common
[4]. http://gesi.org/files/Reports/Smart%202020%20report%20in%20English.pdf
[5]. http://www.itu.int/dms_pub/itu-t/oth/06/0F/T060F00600C0004PDFE.pdf
[6]. http://www.smart2020.org/_assets/files/02_Smart2020Report.pdf
[7]. http://www.itu.int/dms_pub/itu-t/oth/06/0F/T060F00600C0004PDFE.pdf
[8]. http://gesi.org/SMARTer2020
[9]. Murugesan, S. (2013). Harnessing Green IT: Principles and Practices to be published by John Wiley.
[10]. http://newsroom.intel.com/community/intel_newsroom/blog/2012/05/17/intel-sets-2020-environmental-goals
[11]. http://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html
[12]. http://www.pcmag.com/article2/0,2817,2405317,00.asp
[13]. http://www.intel.com/content/www/us/en/processors/core/5th-gen-core-processor-family.html
[14]. http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/3rd-gen-core-family-mobile-brief.pdf
[15]. Bohr, M. & Mistry, K. (2011). Intel's Revolutionary 22 nm Transistor Technology. Tech. rep., Intel, pp.1–28.
[16]. National Instrument (2013). 3rd Generation Intel ® Core TM Processor Family Delivers Cutting-Edge Performance to the PXI Platform. , pp.2–7.
[17]. Kurd, N. et al. (2014). 5.9 Haswell: A family of IA 22nm processors. Digest of Technical Papers - IEEE International Solid-State Circuits Conference, 57, pp.112–113.
[18]. Aswell, H. et al.(2014). The Fourth Generation Intel Core Processor. , pp.6–20.
[19]. https://software.intel.com/sites/default/files/introduction-to-intel-4th-generation-core-processor.pdf
[20]. http://pages.cs.wisc.edu/~rajwar/papers/ieee_micro_haswell.pdf
[21]. Zelkowitz, M. V. & Wallace, D. (1997). Experimental validation in software engineering. Information and Software Technology, 39(11), pp.735–743.
[22]. Montgomery, D. (2001). Design and analysis of experiments. New York: John Wiley.

Hybridized Energy Management Scheme In Smart Homes

Fazal-e-Wahab, Azzam ul Asar, Wasim Habib, Sundas Anwar
Department of Electrical Engineering, CECOS University, Peshawar, Khyber Pukhtunkhwa, Pakistan

Abstract - *A small scale hybridized Power system is very important to overcome the energy crisis of the country without any environmental problems. The Solar PV-Grid hybrid system is the most suitable option to overcome country's energy crisis because of the easy implementation and accessibility on local level of solar energy. In this system the use of utility company supply is minimized by giving priority to PV source instead of AC grid supply. An algorithm has been developed to utilize the green energy more efficiently and as first option in any possible case. The Proteus simulator has been used to simulate the proposed system. The programming of central controller is done in C language using Keil Micro Vision.*

To implement the whole system in said simulator, one central controller has been used with other auxiliary components in control circuitry. A monitoring system has also been included to display the status of the system. The proposed smart hybrid power system (SHPS) has many advantages over conventional hybrid systems. The energy production of solar PV array according to the solar irradiance of the site has been analyzed using MATLAB software. Status of loads, sources during different timings of a day and Cost of the system has also been analyzed. The impact of SHPS on the peak demand and reduction of load on the Utility Company has been analyzed for the proposed site.

Keywords: Smart Home, PV Grid Hybrid System

1 Introduction

With the increasing demands and needs of life, different energy sources have been discovered, and with the passage of time many new techniques and method have been developed to utilize these energy sources easily and effectively [1-2]. The main emphasis of this research is to obtain a solution for power management of grid hybrid power system. An innovative predictive method is incorporated into the power management strategy to improve the hybrid power system operation. The power management strategy aims towards utilizing solar power efficiently for meeting load requirements in day time, designing a smart home keeping in view the life style of consumer at various levels, and finding the impact of electricity saving through smart energy management scheme at utility and consumer levels.

2 Background

Hybrid power systems is rapidly growing area of research, not only domestic, commercial and industrial power systems are studying to be switched to these systems but this technology is also under research to be implemented effectively in automobiles [3]. Renewable energy resources are solar power, wind power, tidal power, geothermal power, wave power and biomass. They are persistent, naturally renewing themselves and environment friendly that is why are called green energy resources. Conventional energy resources slowly decrease with time, like petrol, coal, gas etc [4]. A lot of research and development has been carried out in the area of renewable energy resources and hybrid systems. Off-grid alone hybrid systems have been designed and proposed for the places where either sun or wind are available or both. But the stand alone hybrid system is not reliable, therefore on-grid hybrid system is proposed. In an on grid hybrid system, the system is connected to the AC grid, so that renewable energy helps the national grid in energy provision to the user [5]. Research and development has been made on the control and management of on grid/off grid hybrid power system [6]. But limited work has been done to make this technology handy to the domestic user. Still further study and research are needed to design, test and analyze the hybrid systems for domestic use.

3 Design of the Proposed System

A hybrid solar PV system is designed to function along with the AC main grid. This hybrid power system operates in such a manner that for specific pre-determined load, the maximum available solar power is utilized and the remaining power is drawn from AC grid. The power generated by solar panels is preferred over grid power so that the required total load equals the sum of the two powers and thus the maximum solar power is utilized. This system is suitable for operating during daytime and provides the best possible economical use of solar PV power with a lot of environmental benefits [7].

A general diagram of the designed Smart Hybrid Power System is shown in Figure 1. The conversion and regulation unit is shown in central block which further consists of different components that includes the main circuitry and the controller. The required source is connected to the load through auxiliary and central control unit according to the

algorithm. The load has been distributed for ease of required implementation but in the figure, it is shown as one unit.

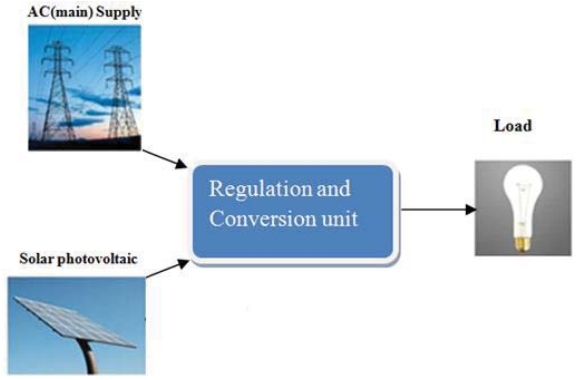


Figure 1: Domestic Hybrid Power System

3.1 Description of Different Parts

The block diagram of SHPS is shown in Figure 2. The PV cell provides DC output which is first passed through DC-DC Buck Converter which gives a constant output before feeding it to the inverter. AC supply from the main grid station is 220V (RMS) and 50Hz has been introduced in the system for the continuous supply to the load. Microcontroller is the main brain of this project. It will receive data, manipulate and then will make a decision accordingly. Unlike other systems, the main beauty of Smart Hybrid Power System is that it uses only one microcontroller with several conversions, regulation and control devices [4]. There are many inputs and outputs in it. The controller gives different outputs through C language coding fed in it on the basis of various inputs.

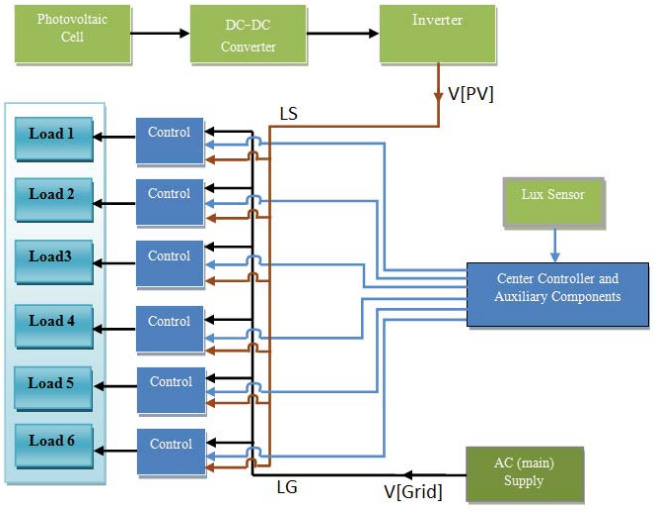


Figure 2: Block Diagram of Proposed Smart Hybrid Power System

In block diagram of system it can be seen that two power lines "LG" and "LS" are coming from conventional and solar sources respectively. The block diagram of the central controller with corresponding inputs and outputs is shown in Figure 3. The inputs which are given to the controller through conversion devices such as ADC, Comparator etc. are known as indirect inputs.

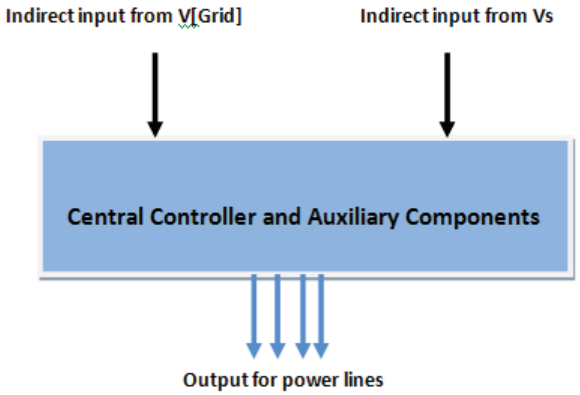


Figure 3: Inputs and outputs of a central controller

Since all majority of the modern appliances run on AC power, thus inverter is used to convert the DC power from photovoltaic source into AC power. The power lines "LS" are used to feed this AC power to the load through control switches. The inverter output is represented by V[PV] in whole work.

Different equipment and devices used with microcontroller or other parts of SHPS for conversion and control are called Auxiliary components. As microcontroller is a digital device it can only understand digital signals. In order to operate practical analog devices for load selection and management, certain auxiliary components such as ADC, control switches etc. are required. The overall load has been divided in six portions i.e. L1(50Watt), L2(100Watt), L3(150Watt), L4(150watt), L5(200Watt) and L6(200Watt). For efficient utilization of solar power, switching of selective loads is based on the availability of solar power.

3.2 Proteus Model of the SHPS

The Proteus ISIS-8 Professional is used to simulate the designed SHPS. The real time simulation of system is shown in Figure 4, Appendix I. The library of simulator consists of thousands of devices. To meet the design requirements of the system, several components have been used which are available and can be implemented at a practical level.

4 Interfacing Major Components

In this section major components of the system have been explained used for simulation purpose. The LCD display (659M4) is used to show the status of the system. As microcontroller is a digital device and can only understand digital signals. In order to perform the SHPS operations by

microcontroller, the continuous type input from the sensor is converted to digital form using an ADC0808.

Lux readings are directly proportional to the energy that is absorbed per square meter per second. For this purpose, TSL251D optical sensor is used that converts light into voltage. The microcontroller compares the instantaneous values of lux with its predefined values and switches the load between PV source and Grid according to available solar power.

Current sensor is a device which senses the AC or DC current in a conducting wire, and produces a proportional voltage to it. A current sensor ACS-712 is used which consists of linear Hall sensor circuit with a copper conduction path located near the surface of the die. The ACS-712 continuously senses the total load current and generates proportional voltage which is fed to controller via ADC. When a load is switched from grid to PV by controller, the current sensor will detect the status of the load by measuring the total load current. If the load is manually switched off, then there will be no change in total current measured by current sensor. Hence, controller will switch another load from grid to PV in order to fully utilize the available solar power.

4.1 Control Switches Interfacing

Six relays RL1, RL2, RL3, RL4, RL5 and RL6 have been used for the purpose of load management which has two possible connections; one has been used to connect the load to V[PV] and the other to V[Grid]. A microcontroller is not able to supply current required for the working of a relay. ULN2003 IC is used to operate the load relays. Seven relays can be connected using ULN2003.

4.2 Sources and Loads Interfacing

Six loads L-1, L-2, L-3, L-4, L-5 and L-6, are connected with sources V[PV] and V[Grid] by means of relays which are driven through relay driver by the microcontroller. The connections are made according to the block diagram and algorithm explained in the following sections. According to the given conditions the first priority is given to the V[PV] source.

5 Flow Chart and Algorithm

The microcontroller has been programmed according to the effective, well defined and efficient algorithm to get required goals. Priority has been given to the PV source over V[Grid] The whole operation principle of the microcontroller that is essential for the operation of the system is presented by flow chart in Figure 5.

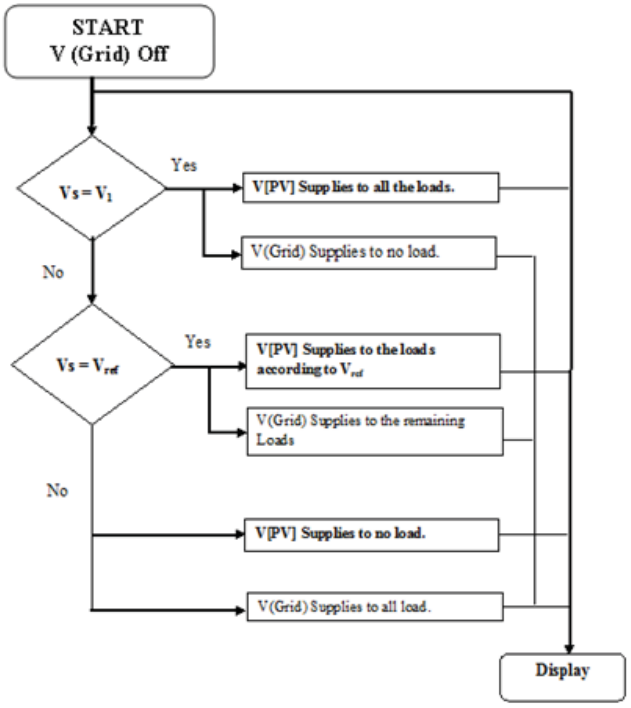


Figure 5: System representation via flow chart

V [Grid] is off at the start. The status of Vs (light sensor output) is continuously checked by central controller and compares it with the reference voltage V1 which has a fixed value of 1.8V. Similarly other reference voltages are V2= 1.69V, V3= 1.59V, V4=1.48V, V5=1.37V, V6=1.27V, V7=1.16V, V8=1.06V, V9=0.95V, V10=0.85V, V11=0.74V, V12=0.63V, V13=0.53V, V14=0.42V, V15=0.32V, V16=0.21V, V17=0.11V and V18=0V. When Vs is greater or equal to V1 (1.8V), this indicates that the power generated by PV is enough to operate all the loads L1, L2, L3, L4, L5 and L6. When Vs is greater than 1.69V and less than 1.8V, means that now PV is not able to run all the loads. Hence, the Loads L2, L3, L4, L5, and L6 are powered by V[PV] and L1 is switched to V[Grid].

When Vs becomes less than 0.11V, all the loads are powered by V[Grid] till the moment PV power become enough to supply power to at least one load. Vref represents the voltages from V2 to V17 i.e. V17≤ Vref<V1. The monitoring system is included in the system for constant display of the sources status.

6 Solar Potentials: A Case Study

6.1 Site Information

Hayatabad is a first planned and modern suburb of Peshawar, the capital of Khyber Pukhtunkhwa, Pakistan. The coordinates of the given site are 34°01′North and 71°35′East. A significant amount of sunshine is received in Peshawar throughout the year which provides a great opportunity to

overcome the shortage of power by using the SHPS for better utilization of solar energy.

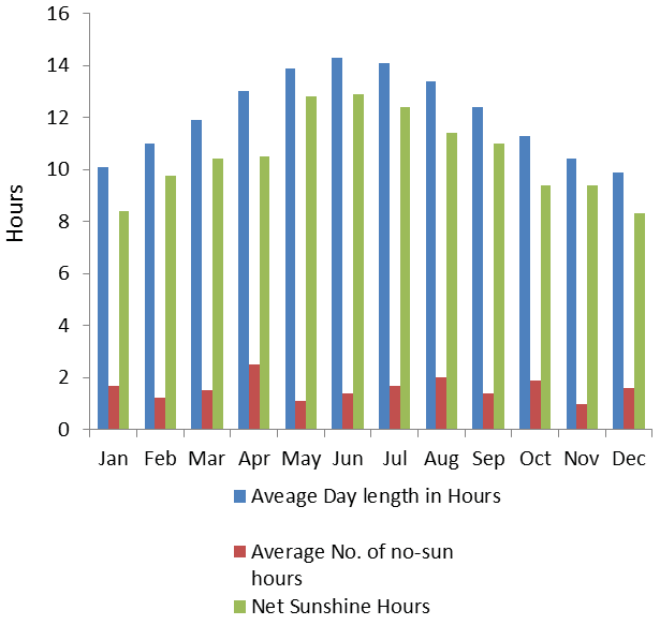


Figure 6: Monthly day length, no-sun and net sunshine hours of Peshawar

Figure 6 shows the net sunshine hours of the whole year, in which December and January has lowest averaged sunshine hours with 8.3 and 8.4hr/day due to short winter days. Average of annual net sunshine hours are 10.5 hour per day. According to NASA's data, the total cloudy days with no sunshine hours (black hours) are 1.6hr/day and 1.7hr/day consecutively.

6.2 Solar Irradiance of Peshawar

The average monthly irradiance of Peshawar is shown in Table1.

Table1: Monthly average irradiance of Peshawar

	Jan	Feb	Mar	Apr	May	Jun
Solar Irradiance (kWh/m^2/ day)	3.09	3.79	4.78	5.99	7.07	7.68
	Jul	Aug	Sep	Oct	Nov	Dec
	6.96	6.19	5.69	4.86	3.72	2.88

6.3 Hourly Solar Irradiance of the Site

The average hourly solar irradiance of an average day of Peshawar for different months is given in Table2. The information of the solar radiation intensity at a given location is of essential for the development of solar energy based projects and in the long term evaluation of the solar energy conversion system performances. This information can be used in the design, cost analysis and efficiency calculation of project.

7 Simulation and Results

In this Section, the Power Calculation, economic analysis and impact of the proposed system on peak demands are covered. The solar panel model is designed using Simulink for the real time simulations and calculation of output power at different irradiance for a day.

7.1 Modeling of Solar Panel in Matlab Simulink

The block diagram of the model of solar panel for Simulink's GUI environment is shown in Figure 7 (Appendix I) along with voltage and current sensors. The last stage of the model is the block that is composed of Solar panel model for GUI. This block consists of sub-models and by connecting these sub-models a final model is built. The solar panel comprises of 72 solar cells which are connected in series to get a required output voltage. The current sensor is shorted to measure the solar panel's short circuit current I_{sc}. To measure open circuit voltage (V_{oc}) at peak irradiance the voltage sensor is shorted to ground. Irradiance is taken as input to the panel in order to analyze the effects of the solar radiation at different levels. Different PV and VI graphs were taken by changing the value of solar irradiance.

7.2 Hourly Power Calculation Using Matlab Model

The panel output at different irradiance values for different months is given in Table 1. The average hourly irradiance values of each month of the site given in Table 2 (Appendix I) are used to calculate the output power of Solar Panel. The hourly irradiance value for an average day of June has been considered. The calculation of percent shares of Grid and Panel is shown in Table 3, Appendix I.

8 Economical Analysis of Smart Power

The economic analysis of photovoltaic solar system is investigated in this section. The 'Levelized cost of electricity' (LCOE) and energy calculation of panel are two major parts that are discussed.

8.1 Energy Calculation of solar Panel

As the net sunshine hours vary during a year, the energy produced by panel also varies. The sunshine hours in Peshawar increase from January to July then decreases in august till December. The month of June has highest sunshine

hours of 12.9hr/day due to long summer days and has highest energy production.

The total energy produced in KWh/year by panel is 2435.77KWh. Figure 8 shows the graph of energy in KWh/month produced by solar panel.

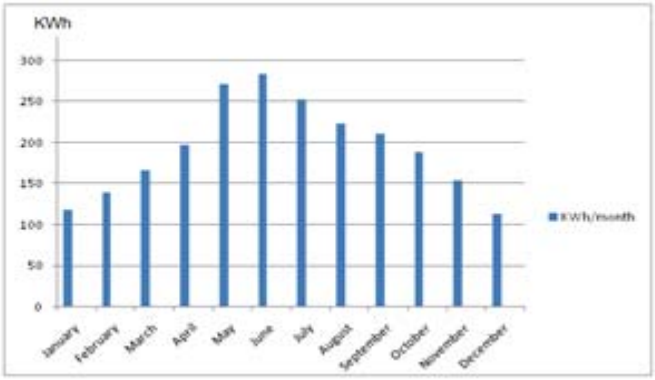


Figure 8: Overall energy production of the panel.

8.2 Levelized Cost of Electricity

For the economic assessment of Solar system, Levelized Cost of Energy (*LCOE*) is the most common tool for finding the cost of per unit generation from PV panel. Levelized Cost Of Energy (*LCOE*) depends on the performance, costs of solar power system, operation and maintenance over the life time of system and cost of other equipment's installation (i.e. Inverter ,converter, racking ,wiring & labour) [8]. Equation (3) represents the specific formula while equation (2) represents the generalize formula of Levelized Cost of Electricity (*LCOE*)

$$LCOE = \frac{Total\ Life\ Cycle\ Cost\ of\ Solar\ Power\ Plant}{Total\ Lifetime\ Energy\ Production} \quad \text{.....(2)}$$

$$LCOE_{Solar} = \frac{C_{panel} + C_{O\&M} + C_{CU} + C_{inst}}{E_t} \quad \text{.....(3)}$$

Where LCOE Solar is Levelized Cost of Electricity for Solar System (Rs/KWh), C_{panel} is Cost of solar panel, $C_{O\&M}$ is cost of operation and maintenance, CCU is the Cost of Control unit, C_{inst} is Cost of installation and Et is Annual energy produced by panel

The price of polycrystalline solar panel varies from 110 to 120 PKR/watt.

$C_{panel} = 1050 * 120 = 126{,}000$ *PKR.*

For less than 1.5 kilo watt the operation and maintenance cost of solar power system up to 1kW is 0.27$/watt for total life cycle of photovoltaic system [9]. In Pakistan it is almost *Rs. 27/watt* for total life cycle of solar system. So,

$C_{O\&M} = 1050 * 27 = 28{,}350$ *PKR.*

Cost of installation includes the wiring, rack (frame for solar module) cost, labour cost and other auxiliary appliances such as converter etc, required in the PV system.

$C_{isnt} = 5000$ *PKR*

The cost of control unit consists of Microcontroller, Load Relays, sensors, DC-DC converter, inverter etc

$C_{CU} = 40{,}230$ *PKR*

8.3 Total Energy Produced by Panel throughout its Life

The total units of energy produced by solar panel per annum are 2319KWh. According to the TATA BP Solar data sheet the valuable life of panel is 25years but after 20 years the output power of panel decreases

$E_t = 2319 * 20 + 0.8(2319 * 5)$

$= 48699 + 9276 = 57{,}975$ *KWh/25yrs*

Now eq. (3) becomes

$$LCOE_{Solar} = \frac{126000 + 28350 + 40230 + 12303}{57975}$$

$$LCOE_{Solar} = \frac{206883}{57975}$$

OR

$$LCOE_{Solar} = 3.56\ \frac{PKR}{KWh}$$

Where 3.56 PKR/KWh is the cost of solar energy unit produced by the panel during day time.

8.4 Cost of Energy using Utility Tariff

The PV system produce 2,319 kWh per year, and the total lifetime energy production of the panel is 57 975 units. Using the utility tariff COE_{ut} , cost of energy can be obtain by multiplying the average rate with the total units produced by PV system.

$COE_{ut} = 57{,}975 * 13 = 753{,}675$ *PKR*

The total average cost of 1050 watt Solar system is 167,883 PKR, while the total saving during the module life time is 753,675 PKR.

8.5 Payback Period

The Payback time provides the total recovery time required to overcome the lifetime cost of the panel. The payback period in years can be obtained by dividing the Total life cycle cost of PV system by saving per year.

$$Payback\ period\ in\ years = \frac{life\ time\ cost\ of\ solar\ PV\ system}{saving\ per\ year}$$

Saving per year= Average energy produced by panel per annum * utility tariff:

*Saving per year= 2319 * 13= 30,147 PKR/year*

Payback period in years= /30,147 = 6.86 years

Duration of 6.86 years is required for the solar PV system to recover the total investment made on it.

8.6 Impact of SHPS on Power Utility

Smart Hybrid Power System (SHPS) is the real solution to overcome energy crisis. It can shave the peak loads in day times and can help reduce load on WAPDA and as a result load shedding will be reduced. So the problem of electricity breakdown can be reduced or removed by using solar energy in such a way that it provides continuous services of electricity. The annual energy saving by installing the proposed system in the area of hayatabad is given in Figure 9.

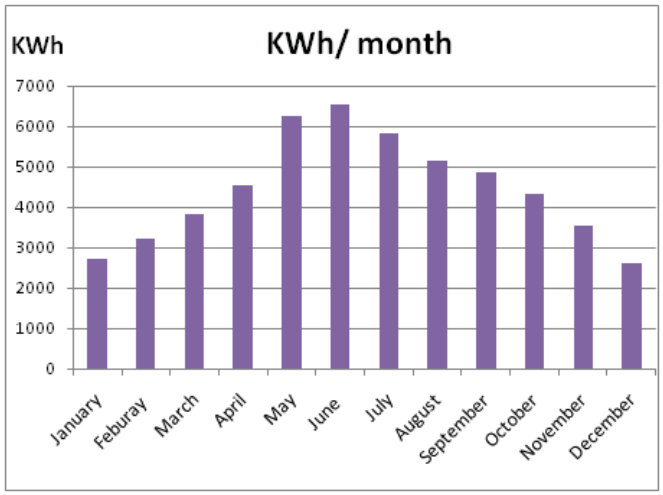


Figure 9: Energy saved by installing the proposed system

The total energy produced in Kilo-watt-hours per year in Hayatabad is 5.33 *1010 kWh. By the implementation of smart hybrid power system two benefits can be achieved. First, by installing such system the consumer will pay 3.56 PKR/KWh which is much lower than the utility company's tariff. Secondly, at day time the burden on utility companies will be greatly decreased and during the day time the utility power can be saved.

9 Conclusions

To make the proposed SHPS cost effective, efficient and to reduce electricity breakdown, different tasks have been are performed. The whole system is simulated in Proteus simulator ISIS 7 using a single central controller instead of several controllers, which makes it more efficient and cost effective. As the main idea of this project was to prioritize the solar energy source, hence the highest priority has been given to PV source. The system has been connected with the utility company supply to make the system reliable for continuous power supply. The utility supply has been kept on second priority to minimize the cost of energy per unit and reduce load on utility company.

There are plenty of reasons that equate to the advantages of using Smart Hybrid Power System and are a real solution to overcome energy crisis by reducing load on WAPDA. It has been analyzed that 5.33 *1010 kWh energy per year can be saved by installing the designed SHPS on the proposed site. The electricity crisis can be completely remedied by introducing this system in the whole country.

10 References

[1] Wicaksono, H; Rogalski, S; Kusnady, E. (2010). Knowledge-based intelligent energy management using building automation system. IPEC, 2010 Conference Proceedings, Singapore, 2729 ,1140–1145.

[2] Yang, J.M; Wu, J; Dong, P ; Wang, B. (2004). The study of the energy management system based-on fuzzy control for distributed hybrid wind-solar power system. First International Conference on Power Electronics Systems and Applications, Hong Kong, 113-117

[3] Zhang, X., Chau, K., & Chan, C. (2009). Design and implementation of a thermoelectric-photovoltaic hybrid energy source for hybrid electric vehicles. World Electric Vehicle Journal, 3, 1–11. European Association for Battery,Hybrid and Fuel Cell Electric Vehicles (AVERE).

[4] Fesli, U., & Bayir, R.2009). Design and implementation of a domestic solar-wind hybrid energy system. Electrical and Electronics, 29-33.

[5] Ambial, M. N., Islam, K., Shoeb, A., Maruf, N. I., & Mohsin, A. S. M. 2010. An Analysis & Design on Micro

[6] Generation of A Domestic Solar-Wind Hybrid Energy System for Rural & Remote Areas - Perspective

[7] Bangladesh Solar Power Wind Power. Engineering, 2(Icmee), 107-110.

[8] Prasad, G., & Srinivasan, S. 2010. Hybrid Solar and Wind Off-Grid System-Design and Control. International,

[9] Imtiaz Ashraf. 2004. " Techno economic viability of rooftop hybridized solar PV-AC grid assisted power system for peak load management" Power Electronics, Machines and Drives, 2004. (PEMD2004). Int Conf on Power Electronics, Machines and drives.1(2): 442 – 446

[10] Cambell M, 2008, The Drivers of the Levelized Cost of Electricity for Utility Scale Photovoltaic, Sunpower Corportation

[11] [9] Paul E, D Bray, 2012, Evolution of Solar Operating Practices: Advanced O&M Benefits from Module-Level Monitoring, Solution Deployment Brief, Altaterra Research Works.

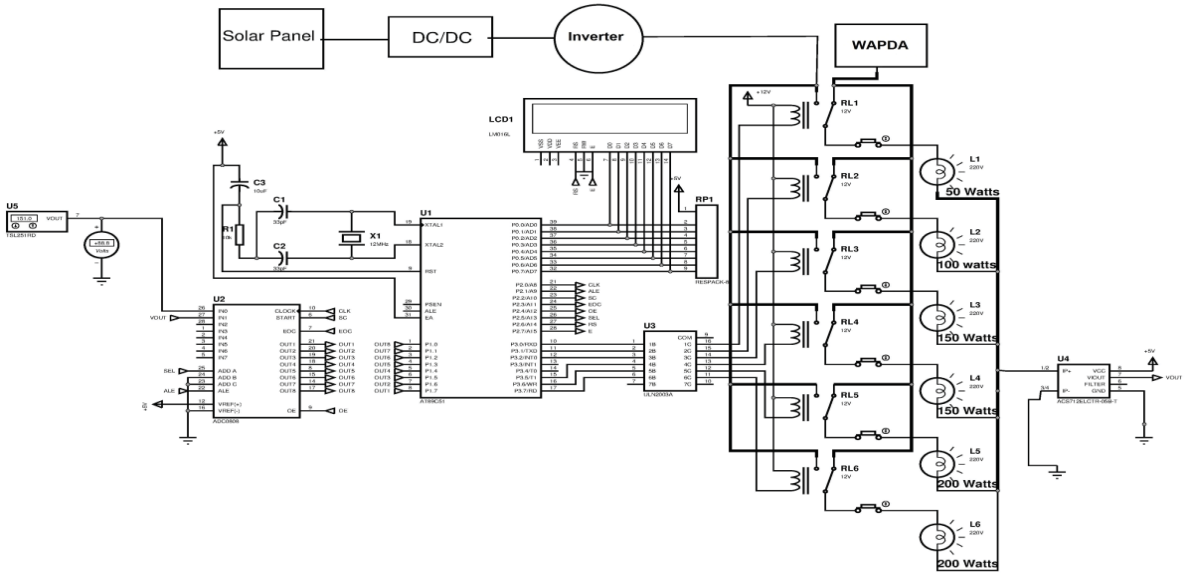


Figure 4: Real time simulation of the system

Table 1: Monthly Average hourly solar radiation data for Peshawar

Hour	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	July	Aug
5	0	0	0	0	0	0	0	0	0	39	0	0
6	32	0	0	0	0	0	0	45	131	196	109	68
7	130	49	31	0	0	0	37	176	251	301	217	193
8	258	192	97	42	42	81	183	311	400	422	289	331
9	453	372	264	183	178	256	372	494	564	586	551	528
10	578	517	406	322	311	400	525	650	703	722	694	681
11	653	594	514	439	433	531	633	758	806	822	802	794
12	706	658	569	489	489	606	683	808	856	858	837	826
13	706	639	578	486	492	617	683	783	794	842	791	780
14	648	601	388	359	399	517	601	660	681	701	670	661
15	482	470	258	231	282	401	521	553	576	581	558	554
16	350	318	174	72	129	243	390	422	451	460	449	431
17	190	171	40	0	31	98	211	279	331	362	309	278
18	42	30	0	0	0	0	41	103	178	214	167	105
19	0	0	0	0	0	0	0	0	32	56	38	0

Table 3: Calculation of percent shares of Grid and Panel.

Time	Irradiance (W/m^2)	Panel Power (W)	Grid Power (W)	Total Connected load (W)	Panel %Share	Grid % Share
5	39	42	858	850	5	95
6	196	196	704	850	23	77
7	301	308	592	850	36	64
8	422	420	480	850	59	41
9	586	609	291	850	71	29
10	722	742	158	850	87	13
11	822	840	60	850	99	1

Table 3: Calculation of percent shares of Grid and Panel (Continued …)						
12	858	896	4	850	100	0
13	842	875	25	850	100	0
14	701	714	186	850	84	16
15	581	602	298	850	71	29
16	460	441	459	850	52	48
17	362	350	550	850	41	59
18	214	210	690	850	25	75
19	56	56	844	850	7	93

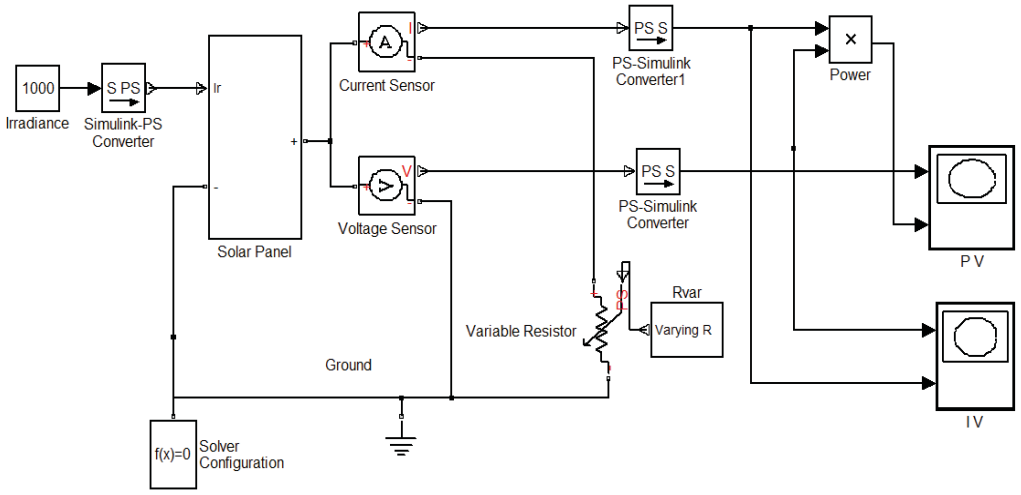


Figure 7: Block diagram of 150 watt solar panel designed in MATLAB Simulink

Table 3: Percentage shares of Grid and Panel.

Time	Irradiance (W/m^2)	Panel Power (W)	Grid Power (W)	Total Connected load (W)	Panel %Share	Grid % Share
5	39	42	858	850	5	95
6	196	196	704	850	23	77
7	301	308	592	850	36	64
8	422	420	480	850	59	41
9	586	609	291	850	71	29
10	722	742	158	850	87	13
11	822	840	60	850	99	1
12	858	896	4	850	100	0
13	842	875	25	850	100	0
14	701	714	186	850	84	16
15	581	602	298	850	71	29
16	460	441	459	850	52	48
17	362	350	550	850	41	59
18	214	210	690	850	25	75
19	56	56	844	850	7	93

Microcontroller Based Embedded System for Vehicle Plate Number Authentication and Verification Using Radio Frequency

Nweke Chisom B[1]., Chukwugozie Ihekweaba[2], Ogechi Linda Ihekweaba[3]

[123]Department of Computer Engineering Michael Okpara University of Agriculture, Umudike, Abia State Nigeria

ABSTRACT-The loss of vehicles as well as proliferation of traffic offenders has made it needful to develop a mobile intelligent compact system for the authentication and verification of the vehicle plate number. This paper aims at presenting an embedded system for quick identification and authenticity of vehicle plate number. The proposed system uses a Radio Frequency (RF) module interfaced to an 8051 microcontroller family. The plate number, Engine Number, and Chassis Number of the vehicle is programmed into the ROM of the 8051 microcontroller and transmitted through the RF. The received information will be compared with the documents of the vehicle to determine the authenticity of the car.

Keywords: Vehicle, RF, Plate, Engine, chase, number, microcontroller

1.0 Introduction

Car theft has become the order of the day in recent years, as no day passes without the news of missing vehicle. It is most worrisome that many of such vehicles pass police check points without being identified. The thieves are so smart that once they lay hand on these cars, they immediately change the vehicle plate number and in most cases, remold the engine number which makes it difficult for identification. Many technologies that have been developed towards tracking down these stolen vehicles have proven ineffective. These are as a result of high cost of the necessary devices. In Nigeria today, many people are using government official plate numbers to avert the law. A vehicle registration plate is a metal or plastic plate attached to a motor vehicle for official identification purposes. This plate is inscribed with a numeric or alphanumeric code that uniquely identifies this vehicle within the issuing region's database [1]. Vehicle registration numbers are a way of identifying vehicles. It also serves as a legal license that permits vehicle to ply the roads. The number plate remains the principal vehicle identifier, despite the fact that it can be deliberately altered or replaced in crime situations [2]. Plates are designed to conform to standards with regards to being read by the eye in the day or at night, or by electronic equipment. To this end, there is an urgent need to develop other alternative means of preventing indiscriminate removal of vehicle plate numbers by individuals without appropriate authorization. The system presented in this work is an embedded system for vehicle plate number (ESVPN) based on the RF. The system will contain the vehicle plate number, chassis number and engine number, which makes it difficult for anyone to easily change his or her plate number without following the appropriate

procedures. Radio-frequency identification (RF) is an automatic identification method, relaying stored information and remotely retrieving data using devices called the RF or transponders [3]. This system if adopted will deter those who specialize in removing and replacing plate numbers for what so ever reasons.

Embedded system is now an emerging technology in various fields, which is well known for its compact size and processing speed. It is also playing a significant role in security and process management [5]. In order to prevent thefts, there exist many methods: Automatic Identification and Data Capture (AIDC) such as biometric systems, image processing techniques like License Plate Recognition(LPR)systems, Optical Character Recognition (OCR), Virtual Barcodes, smart cards, authentication methods such as one time passwords (OTP) [6]. All these in no doubt have served the purpose. As level of crime increases on daily bases, it is imperative to adopt new measures to assure security for vehicle owners.

License plates have been around since the invention of automobiles. France was the first country to introduce the license plate with the passage of the Paris Police Ordinance on August 14, 1893, followed by Germany in 1896 [3]. The Netherlands was the first country to introduce a national license plate, called a "driving permit", in 1898. The first licenses were plates with a number, starting at 1. By August 8, 1899 the counter was at 168. When the Netherlands chose a different way to number the plates on January 15, 1906, the last issued plate was 2001. Plate's numbers are usually fixed directly to a vehicle or to a plate frame that is fixed to the vehicle [4]. Sometimes, the plate frames contain advertisements inserted by the vehicle service centre or the dealership from which the vehicle was purchased. Fig 1 below shows a typical Vehicle plate number.

Fig. 1: Vehicle Traditional Plate Number

2.0 Materials and Methods

On conceptualization, the following system block diagram was realized. This system consists of a Power supply, RF Module, Micro-controller, LCD, and Keypad. Figure 2 below shows the complete block diagram of the systems.

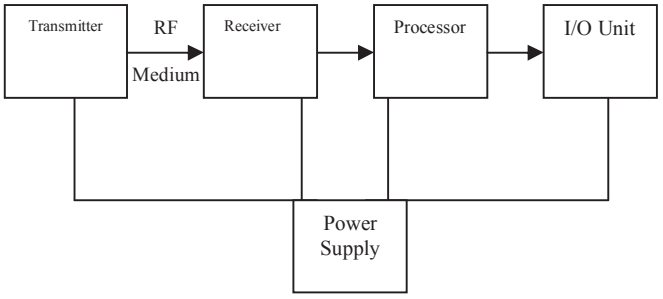


Fig. 2, System Block Diagram

The basic modules are then developed using the relevant discrete random logic and integrated circuit (IC) components. The ICs used involved include LSI, MSI, and VLSI. The hardware is separately tested for functionality. The system software is developed and subsequently programmed into the system, soldering guns, wires, breadboards; tweezers, etc were used to develop the hardware. The functionality tests were maintained by avometers, oscilloscopes and logic analyzers. The assembler was used to develop and test the system software.

On proof of operational readiness, the object code derived from the assembler was blasted into the system ROM using the Erasable Programmable Memory.

3.0 System Hardware Implementation

3.1 Power Supply

A Direct current (DC) power supply is used in this system, both for the transmitter, receiver systems, processors and Capacitors are connected in parallel to act as a filter from the battery source. Then 7805 power regulate is connected at the output of the filtered battery to produce a constant and stable 5volts DC output.

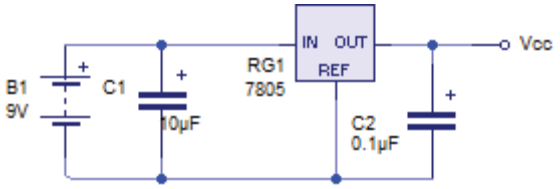


Fig 4. Power supply circuit

3.2 The (RFID) Module

The RFID technology is based on the principle of magnetic coupling. Here, electric current flowing in one circuit induces current flow in another circuit through a magnetic field generated in the space between the circuits. The two major classes of RFID transponders are active and passive. In passive RFID, there are two major components; the reader and the mobile tag. The reader has two main functions: the first is to transmit a carrier signal, and the second is to receive a response from any tags in close proximity to the reader. A tag needs to receive the carrier signal, modulate it with respect to the data on the card. It then retransmits the modified response back to the reader. In modern passive RFID devices, the tag consists of a small integrated circuit (that performs the modulation) and an antenna. The benefit of passive RFID is that it requires no internal power source; the circuit on the tag is actually powered by the carrier signal. Thus, the carrier signal transmitted from the reader must be considerably large so that the response can be read even from the card. As shown in the above block diagram RFID systems are classified according to the properties of the data carrier called a transponder or tag.

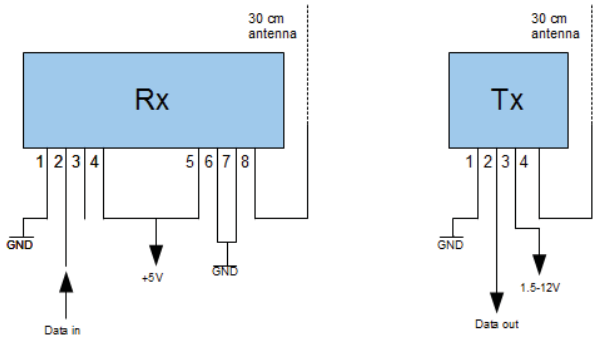


Fig. 5 Radio transmitter and Receiver

3.4 Microcontroller

Microcontrollers are computers that are designed to carry out specific functions. They are embedded in any other computer or machine. They carry out their functions by taking inputs from the devices they are incorporated into. They have the ability of turning the appliances ON and OFF based on the SMS sent to the phone connected to the microcontroller [4]. In this design, 8051 family of micro-controller is employed. It comes in a 40-pin dual in-line package (DIP) with internal peripherals. The 40 pins make it easier to use the peripherals as the functions are spread out over the pins. Fig 6 shows the 8051 pin configuration of the microcontroller. The microcontroller used in this work (AT89S52) It is a 40 pin DIP, 8 bit microcontroller with Complex Instruction Set Computer (CISC) architecture. It has four ports P0 through P3 (port0 – port3). It has 16-bit timers/counters, one serial port, 64K bytes of external program memory (ROM) and 64K bytes of Data memory (RAM), as well as on-chip oscillator. It supports a power supply of 2.0-5V dc, which makes it flexible, cost effective and most suitable for many embedded system applications.

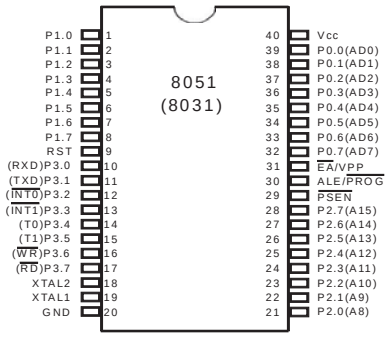


Fig 6. The 8051 Microcontroller

3.3 The Liquid Crystal Display (LCD)

A liquid crystal display (LCD) is a thin, flat panel used for electronically displaying information such as text, images, and moving pictures. Its uses include monitors for computers, televisions, instrument panels, and other devices ranging from aircraft cockpit displays, to every-day consumer systems such as video players, gaming devices, calculators, and telephones. Among its major features are its lightweight construction, its portability, and its ability to be produced in much larger screen sizes than are practical for the construction of cathode ray tube (CRT) display technology. Its low electrical power consumption enables it to be used in battery-powered electronic equipment. It is an electronically-modulated optical device made up of many number of pixels filled with liquid crystals and arrayed in front of a light source (backlight) or reflector to produce images in color or monochrome.

Fig. 7. Liquid Crystal Display (LCD)

3.4 Embedded Vehicle Plate Number Transmitter.

The system is expected to be embedded to a vechicle at the piont of registration. The system will be programmed with the car details like, the vehicle plate Number, chasiss Number, and Engine Number. This will be transmitted at a request to the embedded plate number receiver.

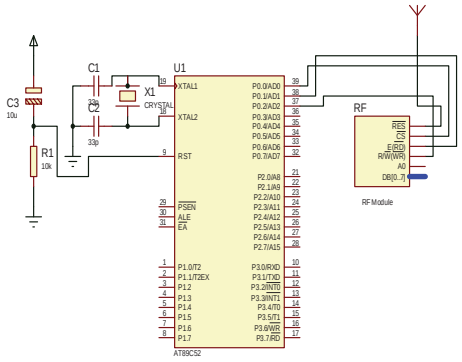


Fig 8. The embedded vehicle plate number circuit diagram.

3.4 Embedded Vehicle Plate Number Receiver

The receiver system has a transceiver which enables it to send a requist to the Vehicle and receive back the required information, which may be compared with the physical Number plate, placed on the vehicle and the ones on the vehicle particulars. This system in no doubt will improve security of cars on our roads and make it easier to be investigated in time of necessity.

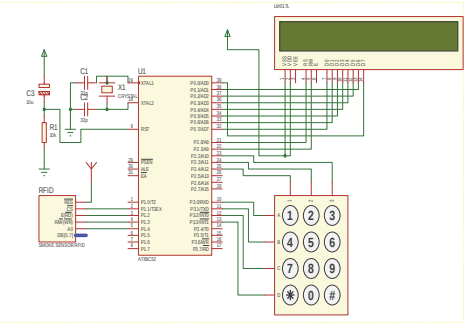


Fig.9 The hand held Receiver for the embedded Vehicle plate number circuit diagram

4.0 Tests and Results

After careful design and integration of different components of this system, the system was power and the embedded system that will contain the vehicle particulars programmed with these information (engine number, chassis number, and plate number). A query was sent from the receiver system and a feedback immediately sent displaying the vehicle plate number,

engine number, and chassis number. The last 4 (four) digits of the vehicle plate number was used for the query. The idea of using the last four digits was based on the fact that vehicle last four digits are unique.

5.0 Conclusion

Vehicle registration numbers must be correctly displayed on number plates as set by the road safety agency for easy identification. When this mark is not there or has been deliberately removed, it makes it almost difficult to identify vehicle. The proposed system in this paper provides alternative means of identifying vehicles even though the plate number has been tampered with.

REFERENCES

[1].http://en.wikipedia.org/wiki/Vehicle_registration_plate

[2]. Harpreet kaur, Naresh Kumar Garg "NUMBER PLATE RECOGNITION" International Journal of Computer Application and Technology (IJCAT) May - 2014, pp. 20-24

[3]. Robertson, Patrick (1974). The book of firsts. C. N. Potter: distributed by Crown Publishers. p. 51. Retrieved 07-09- 2014.

[4]. "License Plates of New Zealand". Worldlicenseplates.com. Retrieved, 07-09-2014.

[5] S. Dharanya and A. Umamakeswari, "Embedded Based Conveyance Authentication and Notification System" International Journal of Engineering and Technology (IJET). Vol 5 No 1 Feb-Mar 2013

[6]. Nor Azlina , Bt Abd Rahman , Mohsen Bafandehkar, Behzad Nazarbakhsh , Nurul Haniza Bt Mohtar , "Ubiquitous Computing For Security Enhancement Of Vehicles", IEEE International Conference on Vehicular Electronics and Safety (ICVES) ,Beijing, pp: 113-118, 2011.

[7]. Orukpe, P. E. and Adesemowo A, 2012, "Digital control of external devices through the parallel port of a computer using Visual basic, Nigerian Journal of Technology, Vol. 31, No. 3, pp. 261 – 267.

[9]. Kumar Parasuraman, P.Vasantha Kumar, "An Efficient Method for Indian Vehicle License Plate Extraction and Character Segmentation," 2010 IEEE International Conference on Computational Intelligence and Computing Research.

Authors:

Nweke Chisom B. received his B.Sc. degree in Computer Science from Michael Okpara University of Agriculture, (MOUAU) Umudike, Abia State Nigeria in 2012.His currently running PGD in electrical/electronics engineering from same institutions. His research interests are in the fields of Electronics Design and Embedded Systems, Computer Programming; Microcontroller based System, Communication, Computer hardware Maintenance, security System design, Expert Systems, etc. Email: Nweke_ncb@yahoo.com

Ihekweaba, Chukwugoziem is an Associate Professor and currently the Head, Department of Computer Engineering, Michael Okpara University of Agriculture, Umuahia, Abia State, Nigeria. His research interests include Computer Hardware design and maintenance, Security system design, Electronic and communication systems, etc. Email: ihekweaba.gozie@mouau.edu.ng, gozihekwaba@yahoo.com

Ihekweaba Ogechikanma Linda is a lecturer in Department of Computer Engineering, Michael Okpara University of Agriculture, Umuahia, Abia State, Nigeria. Her research interests include Computational Intelligence , Security system design, Expert systems, Design of Microcontroller and Microprocessor based system, Electronic and Communication Systems and other computer, etc. Email: ogechi_ihekweaba@yahoo.co.uk