FPGA Implementation of Auto Switching in a Multicore Hybrid Processor

M.S. Harisha, D. Jayadevappa

Abstract--- This paper proposes indigenously designed SMART processor core auto switching with two splitting strategies. First one is, Separating heterogeneous instructions based on functionality like hardware and software. Second one is, Checking out for FREE/BUSY status of individual core in a multi-core processor and allocate instructions or tasks to FREE status cores, thereby efficiently manage traffic and perform load balancing amongst various processor cores. I have also enclosed snapshots popular hardware interfacing & corresponding display results.

I. INTRODUCTION

The good processing speed of the multicore processors is due to the multiple cores which operate simultaneously on instructions, at lower frequency than the single core. At the same clock frequency, the multicore processor will process more data than the single core processor.

In addition to this, multicore processors deliver high performance and handle complex tasks at a comparatively lower energy or lower power as compared with a single core, which is crucial factor in appliances such as mobile phones, laptop etc. which operate on batteries.

Also, since the cores are fabricated close to each other on the same chip, the signals travel shorter distance between them due to which there is less attenuation of signals.

Since the signals don't attenuate much, more data is transferred in a given time and there is no need of repeating the signals.

Multicore processor advantages:

- Simultaneous execution with high performance
- Multithreaded application and instruction level parallelism
- Moore's law supportive
- Lesser heat generation and starvation free[1]
- Power efficient
- Handles all processes without priority
- Each processes get equal chance to execute
- The signals between different CPUs travel shorter distances, therefore they degrade less.
- Cache coherency circuitry can operate at a much higher clock rate than is possible if the signals have to travel off-chip [2].

Table I: Single core and multi-core comparison

Parameter	Single-Core Processor	Multi-Core Processor
Number of cores on a die	Single	Multiple
Instruction Execution	Can execute Single instruction at a time	Can execute multiple instructions by using multiple cores
Gain	Speed up every program or software being executed	Speed up the programs which are designed for multi-core processors
Performance	Dependent on the clock frequency of the core	Dependent on the frequency, number of cores and program to be executed
Examples	Processor launched before 2005 like 80386,486, AMD 29000, AMD K6, Pentium I,II,III etc.	Processor launched after 2005 like Core-2-Duo,Athlon 64 X2, I3,I5 and I7 etc.

1.1 Multi-core Processor Evolution

Computers and other technology originally began with single-core processors; in the early 2000s, Intel, AMD and several other manufacturers altered the history of computing forever by pushing multi core processors on the market. These multi core processors include general-purpose models, embedded multi core processors; CPUs used networking, DSPs, and graphics processors. The most notable theories and concepts enabling the possibility of multi cores include parallel computing and single integrated circuit dies (also known as chip multiprocessors or CMPs).

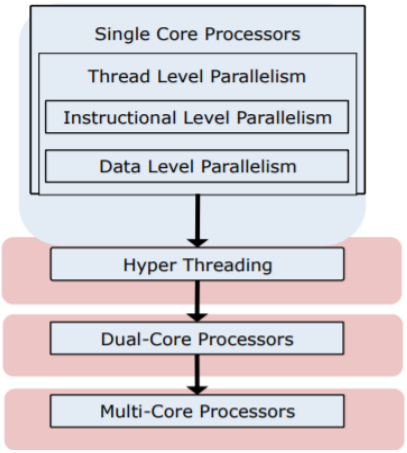


Fig. 1: Evolution of Multi-core technology

At some point after the beginning of the new millennium, computer giants Intel and AMD were forced to up the ante on the PC CPU market. Their single core processors were reaching a peak, and they could not physically improve these current designs without rewriting the entire production process. The manufacturers created the idea of designing their processors on a multi core layout. Up until that point, the market consisted of single core processors. Dual core designs (such as the AMD 64 Athlon X2 and the Intel Core Duo) changed the market forever. They were followed by four-core processors (or a quad core processor design such

Manuscript received February 01, 2019

M.S. Harisha, Research Scholar, Dept. of Electronics Engineering, Jain University, Bengaluru, Karnataka, India. (e-mail: harisha.maganahalli@gmail.com)

D. Jayadevappa, Professor, Dept. of E&IE, JSS Academy of Technical Education, Bengaluru, Karnataka, India. (e-mail: devappa22gmail.com)

as the AMD Phenom II X4 and the Intel i5 and i7), six-core devices (AMD Phenom II X6 and Intel Core i7 Extreme Edition 980X hexa-cores), octo core processors (Intel Xeon E7-2820 and AMD FX-8350) and ten core (Intel Xeon E7-2850).

Over time, the multi-core processor evolved from dual core to tri, quad, hex and octa core designs (or processor chips with 2, 3, 4, 6, or 8 processors). Some processors now hold dozens or hundreds of cores. The evolution also continues in terms of technological design sophistication. The current processors with numerous cores are now designed with multi threading features, breakthrough developments such as memory-on-chip, and heterogeneous cores designed for special purposes. We can attribute these evolutions to several emerging need patterns: on one hand, contemporary technologies must become more and more efficient in networking, multimedia processing, and device recognition. On the other hand, energy efficiency must increase as well.

Manufacturers continue developing multi core processors with improvements in the actual manufacturing process. As individual CPU gates grow smaller through manufacturer breakthroughs, semiconductor microelectronics also become more and more optimized in terms of physical properties. Manufacturers now create CPUs with increasing design efficiency[3].

II. RELATED WORK

Till date, several separate or individual attempts has been made to implement various processor related hardware interfaces (wired & wireless) on FPGA , but none of them have attempted to put several & popular hardware interfaces together as a part of processor core implementation on FPGA. Hence this work is very unique in itself owing to the totality of all the interfaces working together simultaneously (parallel) with other processor popular instructions using multi-core.

Arvind Kumar and Valarmathi[4]have explained the difference between basic working mechanism of stepper and DC motor. They have used VHDL to develop a wireless interface (Bluetooth) to drive a stepper motor using FPGA board. They have also developed a Graphic User Interface to control the direction & the angle of rotation of the motor. Vaidehiand Gopalakrishnan Nair[5] have proposed the anatomy of a typical multi-core processor with shared & distributed cache (memory). They have also proposed scaling partition techniques for multi processor in a RTOS environment. They have proposed automatic load balancing approach for managing processor resources in multi-core environment in related industrial applications. Yi-Jung Chen et al. [6] have developed three algorithms for processor allocation in multi-core architecture for MPSoCs. The three algorithms are based on execution path, initial allocation & path merging to optimize synthesis time for various workloads. Muhammad Faisal et al. [7] have implemented some strategies for Dynamic Core Allocation and Packet Scheduling in Multi-core Network Processors. They have experimented on packet scheduling in a network processor environment.

They have also designed packet level& flow level load balancing, hash schedulers, packet scheduler for multiservice routers. They have also done traffic analysis, load distribution on core release, throughout comparison of different schedulers & dynamic resource allocation. Neelappa and Kurahatti [8]have done very in depth study on RFID technology. They have surveyed about passive, active & semi passive RFID tags. They have studied the end to end working of RFID system, tag architecture, RFID coupling methods like back scattering, near field & far field coupling, applications of RFID with future trends in RFID technology. They have also proposed FPGA based RFID system with a GSM interface using up converter & down converter, ADCs & DACs etc. Shruti Hathwalia and Sansar Chand Sankhyan [9] have explained a method to interface 2x16 LCD display with FPGA board. They have also given the ASCII codes of LCD & Verilog display program. They have proposed to transfer text data from PC to FPGA board which in turn will be displayed on LCD.NSK product data sheets [10] gives complete information on RFID reader & GSM modem used in our research experimental setup. These documents also gave in-depth knowledge of working principle of those two products with theoretical background & practical applications. In this AT commands reference guide [11], all standard AT (Attention) commands are explained in detail along with examples. Two types of extended GSM commands are explained- parameter type & action type for modem control interface for SMS & call control.

This Xilinx application note [12] describes highly optimized Universal Asynchronous transmitter Receiver (UART) transmitter and receiver macros fully compatible with the standard UART communication protocols used for connecting to devices such as PCs or microcontrollers with specific signal description. They have also explained UART operation with break condition, buffer operation & use of MAX220 (RS232) to interface FPGA via UART to a PC or microcontroller.

III. SMART CORE AUTO SWITCHING IN MULTICORE PROCESSOR

A lot of intelligence or smartness is built-in to all the latest & contemporary multi-core processors to perform core allocation amongst the multi-cores. The said core allocation is performed in 2 ways:

Core allocation based on functionality- specific cores are dedicated for the following functionalities like hardware, software, communication, codec, wireless, ports etc. Core Specific instructions will be smartly separated & routed to respective cores for execution.

Core allocation based on traffic/instruction queue in a multi-core environment, individual cores are checked for BUSY (core is executing some instruction or interrupt) /FREE (core is free to take up & execute next assigned instruction) status of operation & accordingly instructions from the instruction queue are allocated to the free core. This is a process of SMART ARBITRATION of allocating incoming instructions from the queue to the freely available cores. The following figure depicts the Multicore hybrid processor.

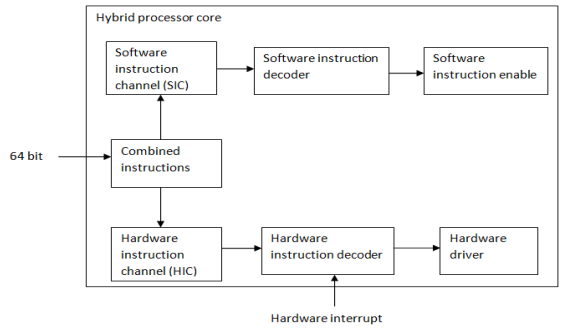


Fig.2: Multi-core Hybrid processor

The above figure indicates a strategy to segregate 64 bit combined hybrid processor instructions into soft instructions /interrupts & hardware interfacing instructions /interrupts.

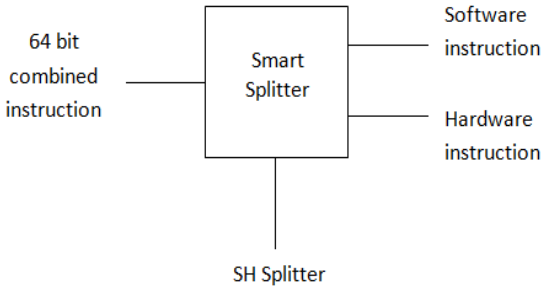


Fig. 3: Combined Instruction Smart splitter in a hybrid processor.

Table II: Combined Instructions

MSB of CI=select	Output	Action
0	SIC=CI	Incoming instruction is considered as soft instruction/interrupt & routed through SIC
1	HIC=CI	Incoming instruction is considered as hardware interfacing/interrupts & routed through HIC

The combined instructions (CI) are split into two ways depending on the most significant bit (MSB) of the Opcode, using the digital circuit shown in the figure. The input line of the splitter is connected to the select line also. The MSB of any incoming instruction to the hybrid processor is applied to the input of the splitter & also to the select line of the same splitter, thereby if MSB=0 then such instructions are routed through software instruction channel (SIC) & if MSB=1, then those instructions are routed through hardware instructions channel (HIC). Hence, CI gets separated into SIC & HIC, as shown in the table. II.

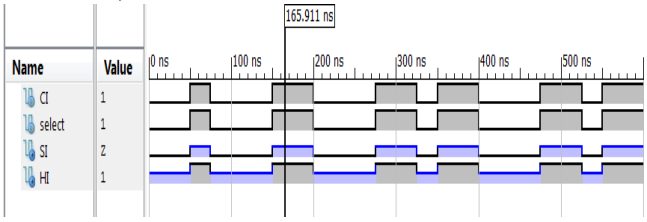


Fig. 4: Simulation waveform of Combined Instruction Smart splitter in a hybrid processor.

3.1 Smart Instruction switching in a dual core processor

In a multi-core processor environment, Instructions and tasks management is a very complex job that needs INTELLIGENCE to take a SMART DECISION about CORE ALLOCATION for various instructions applied to a multicore processor for execution. The below shows a typical instruction queue (I1,I2,I3 & I4) in a multi-core processor, wherein SMART DECISION about core allocation among dual Cores (C1 & C2) is performed by the Decision Circuit. Depending on the instruction mapping among the Dual core & the instruction pipeline /queue/traffic, each core is tested for free/BUSY state (0/1) respectively & the decision will be assigned.

Table III: Combined Instructions

State	C1	C2	Core allocation Decision	Core Status
S1	0	0	Any type	Both free
S2	0	1	Send next instruction to C1	C1 free
S3	1	0	Send next instruction to C2	C2 free
S4	1	1	Wait for C1 or C2 to become free	Both busy

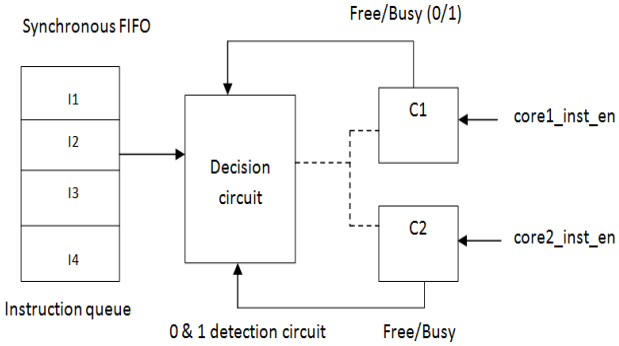


Fig. 5: Dual Core allocation in Smart Decision circuit

The table.III indicates the 4 possible core states (S1, S2, S3,S4) in a dual core processor with decision circuit output and respective core assignment status. For the above design module of smart decision circuit, Verilog RTL code was written to implement the design in Xilinx Spartan6 & behavioral Simulation was done using Xilinx ISE simulator. The RTL schematic indicating all the possible inputs & outputs are shown in the below Fig.6.

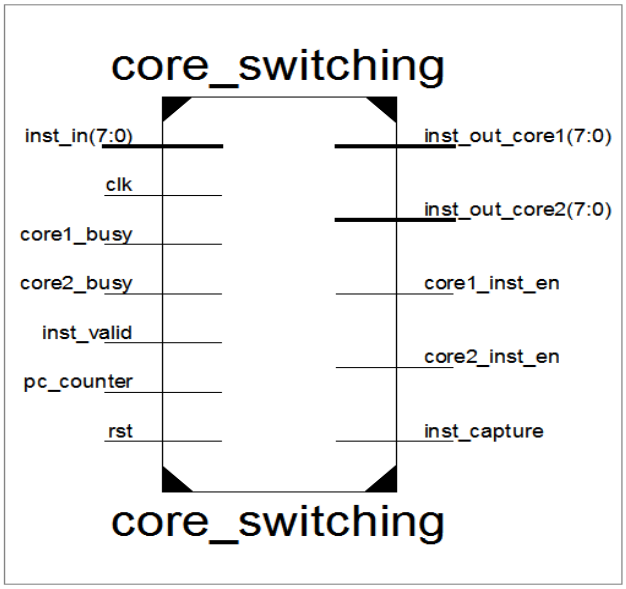


Fig. 6: RTL Schematic of core switching module.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

Table IV: Signal description

Signal name	Input/output	Description
Inst_in [7:0]	input	Combined instructions to the Dual core processor
Clk	Input (synchronous)	Common clock
Core1_busy	Output	Core 1 free or busy status signal
Core2_busy	Output	Core 2 free or busy status signal
Inst_valid	Input	Validation of current instruction with CI
Pc_counter	Input	Program counter (presence of instruction & queue)
Rst	Input (control)	reset
Inst_out_core1[7:0]	output	Incoming CI moved to Core1 after allocation
Inst_out_core2[7:0]	Output	Incoming CI moved to Core2 after allocation
Core1_inst_en	Input	manual enable for instruction allocation to core 1
Core2_inst_en	Input	manual enable for instruction allocation to core 2
Inst_capture	output	Incoming valid CI considered for core allocation

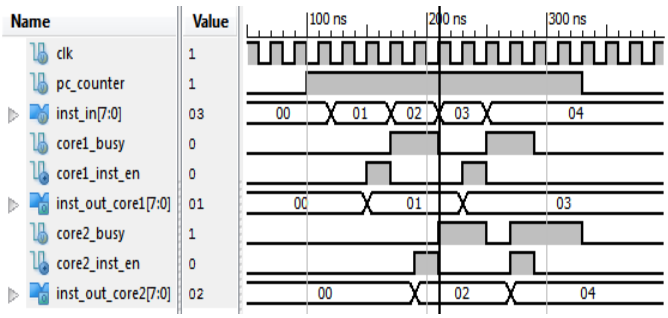


Fig.7: Simulation results of core switching between two cores as per the availability

Steps for the simulation results

- Clock is free running from 1MHz up to 1GHz
- Program counter (PC) would be enabled as soon as instruction is validated (instruction valid=1) & this leads to instruction CAPTURE go high. This PC will remain high till no more valid instructions are available in the queue for core allocation & execution (CAE).
- After global reset is toggled rst is initially 0 & it is made 1 for considerable amount of time (depending on setup & hold time of the target device & again made 0) (rst=1) &rst=0, the old instruction queue is flushed to make way for new instructions& also all the flip-flops, registers & memory contents are cleared. This instruction queue is stored in a synchronous FIFO.
- Instruction [7:0] is an 8 bit Opcode of all valid instructions & this will have an instruction pipeline of all incoming instructions for CAE. Instruction pipeline (IP) can be seen in the waveform (01,02, 03 & 04).
- It is assumed that after reset, all the cores are free & all of them are available for new instruction CAE or Core Arbitration (CA).
- To demonstrate the complex CA process, we are manually enabling individual cores, to take up incoming instructions from the IP. Assuming both Core1 (C1) & Core2 (C2) are free after reset, Core1_inst_en is made high & this leads to C1 taking up first instruction (01) from IP.
- As long as instruction 01 is running or is getting executing in C1, core1_busy will be held high, as can be seen in the simulation waveforms.
- Simultaneously, since C2 is free, we enable Core2_inst_en & thereby load second instruction (02) to C2, for execution. Likewise, core2_busy will be held high as long as instruction 02 is executed in C2.
- The instruction pipeline is lined up with 2 other instructions (03 & 04), as soon as C1 completes instruction 1 & core1_busy goes low indicating C1 is free to take up next instruction.
- Core1_busy going low will enable core1_inst_en to go high & instruction 3 is taken for execution by C1.
- As long as instruction 3 is executed by C1, core1_busy will be held high for the entire duration of instruction 3 execution & core1_busy will go low after the instruction 3 executions is completed.
- Simultaneously when instruction 03 is completed by C1, after C2 has completed executing 02 & core2_busy goes low, core2_inst_en is made high & hence C2 takes the execution of instruction 04.
- Again, core2_busy goes high till the execution of instruction 4 and later goes low.
- The cycle repeats for all the next incoming instructions in the IP (further instruction not shown)

Load Balancing Smart Instruction Arbiter for an Octo-core processor: The Fig.8 shows the architecture of a traffic based load balancing smart instruction Arbiter for an Octo-core processor (processor with 8 cores).

This arbiter is lot more complex in its architecture and functionality, but works in similar way as explained in Dual core smart instruction switching.

This Arbiter is extended version of Dual core smart instruction switch.

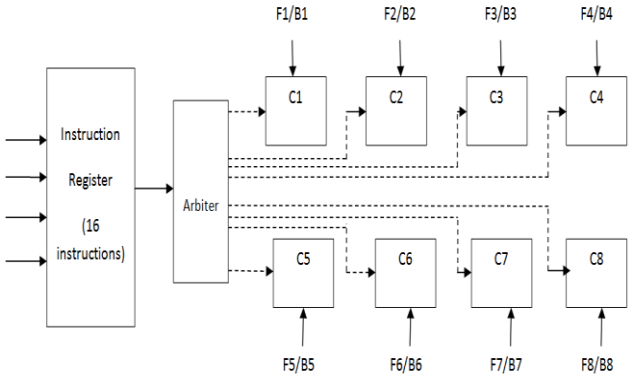


Fig. 8: Traffic based load balancing smart instruction Arbiter for an Octo-core processor

Top Level hardware core Architecture: The below figure illustrates the method for interfacing all the popular & standard hardware interfaces to be driven by the hardware instructions/interrupts derived out of combined instruction set.

All the below hardware interface's respective drivers are designed, coded in Verilog, simulated using Xilinx ISE tool & implemented on Xilinx Spartan3 board & results are displayed in LCD or 7 segment display.

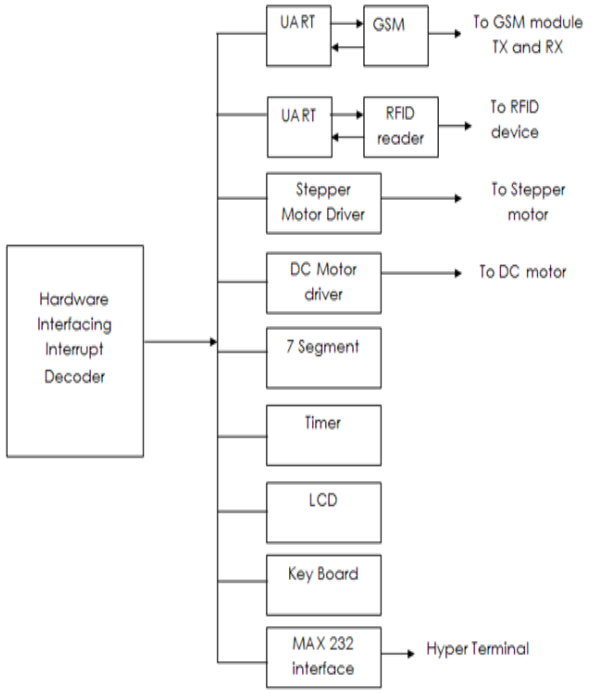


Fig. 9: Top level Hardware core Architecture

The table-V indicates mapping of various processor control codes for all the above mentioned hardware interfaces like-

1. GSM modem via UART
2. Stepper motor control
3. DC motor control
4. RFID reader via UART
5. 7 segment display – Real time clock (hours, minutes, seconds)
6. LCD interface
7. 4x4 matrix keyboard interface
8. Timer
9. Hyper-terminal – serial port read & write via UART

Table V: Mapping of various processor control codes for the hardware

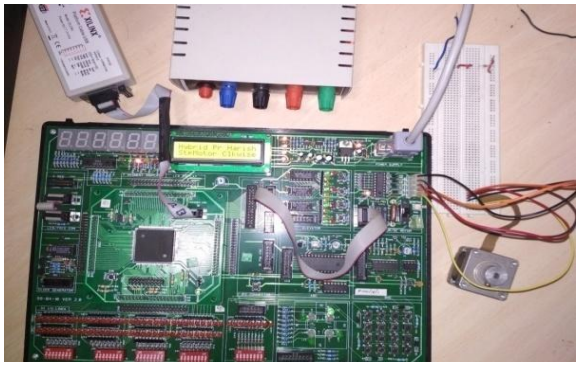
#	Control codes	Module enable	Operation
0	0000	No module	No operation
1	0001	GSM module	Enable GSM, and check the connectivity with Module
2	0010		Send SMS
3	0011		Make a call
4	0100		Read a SMS
5	0101	Stepper motor	Drive stepper in forward direction
6	0110		Drive stepper in reverse direction
7	0111	DC Motor	Drive DC motor in Forward direction
8	1000		Drive DC motor in reverse direction
9	1001	RFID	Read card and status show using a LED glow if card is valid
10	1010	7 Segment display	Display data Output to 6x7 segment display on FPGA kit
11	1011	LCD Output	Display data output in 16x2 LCD Display on FPGA kit
12	1100	Key board input	Read data from 4x4 matrix keyboard of FPGA kit
13	1101	Timer	Switch on timer to be run on FPGA kit
14	1110	HT Read	Receive data from FPGA UART to Hyper Terminal of Computer
15	1111	HT Write	Send data to FPGA UART from Hyper Terminal of Computer

IV. IMPLEMENTATION RESULTS

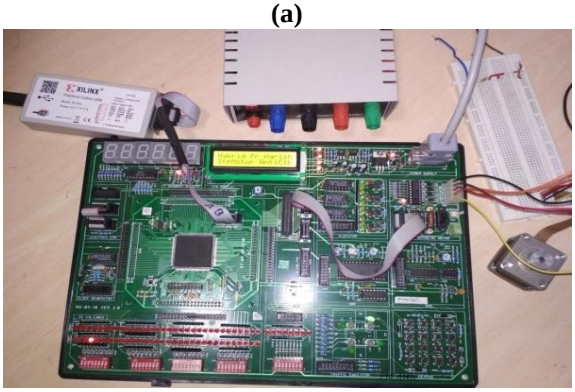
A. FPGA based Hybrid Processor Interfacing with Stepper motor

The Fig.10 shows a Hybrid Processor core implemented in FPGA driving a stepper motor connected through a bread board.

Using, appropriate instructions [0101 & 0110] the motor can be driven clockwise (Fig. 10.a) and anti-clockwise (Fig. 10.b) in step respectively. The number of steps rotation of the stepper motor can be controlled using additional instructions & implementing related logic in the FPGA stepper motor control module.



(a)



(b)

Fig. 10: Hybrid Processor core implemented in FPGA, (a) Driving a stepper motor in clock wise, (b) Driving a stepper motor in anti clock wise

B. FPGA based Hybrid Processor Interfacing with DC Motor

The Fig.11 shows a Hybrid Processor core implemented in FPGA, driving a DC motor connected through a bread board. Using, appropriate instructions [0101 & 0110] the motor can be driven clockwise (Fig.11) and anti-clockwise respectively. Speed control of DC motor can be achieved by using additional instructions and adding associated logic into FPGA DC motor control module.

Fig. 11: FPGA based Hybrid processor driving DC motor.

C. FPGA based Hybrid Processor Interfacing with GSM modem

The GSM Modem [10] can accept any GSM network operator SIM card and act just like a mobile phone with its own unique phone number. Advantage of using this modem will be that we can use its RS232 port to communicate and develop embedded and FPGA applications. Applications like SMS Control, data transfer, remote control and logging can be developed easily.The modem can either be connected to PC serial port directly or to any microcontroller or FPGA. It can be used to send and receive SMS or make/receive voice calls. It can also be used in GPRS mode to connect to internet and do many applications for data logging and control. In GPRS mode we also connect to any remote FTP server and upload files for data logging.GSM modem is a highly flexible plug and play quad band GSM modem for direct and easy integration to RS232 applications. Supports features like Voice, SMS, Data/Fax, GPRS and integrated TCP/IP stack.

The instructions used for controlling the GSM modem are called AT commands. GSM modem supports a common set of standard AT commands. In addition to this, GSM modem also supports an extended set of AT commands. One use of the extended AT commands is to control the sending and receiving of SMS messages[10].The following table lists the AT commands that are related to the writing and sending of SMS messages:

Table VI: GSM commands

AT COMMAND	Modem Action
+CMGS	Send message
+CMSS	Send message from storage
+CMGW	Write message to memory
+CMGD	Delete message
+CMGC	Send command
+CMMS	More messages to send

One way to send AT commands to a GSM modem is to use a program. A program's function sends the characters typed to the GSM modem. It then displays the response it receives from the GSM modem on the screen.

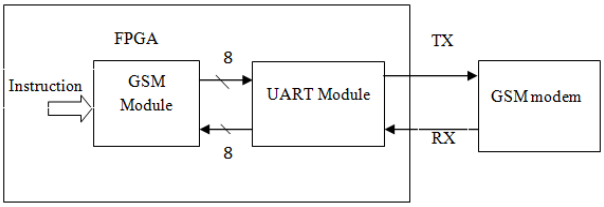


Fig.12: Block Diagram of Interface of GSM and UART

In Fig.13, hybrid processor core implemented in Xilinx Spartan FPGA interfaced to a GSM modem through UART and level converter MAX232 module is depicted. The hybrid processor core controls GSM modem to Enable GSM modem, send SMS, make call & read SMS using instructions 0001, 0010, 0011 & 0100 respectively. Using this setup, any GSM mobile can send & receive SMS globally to the GSM modem driven by FPGA based Hybrid processor. Voice calls also can be made globally using same setup.

Fig. 13: Hybrid processor core implemented in Xilinx Spartan FPGA interfaced to a GSM modem.

D. FPGA based Hybrid Processor Interfacing with RFID Reader

RFID Reader consists ofthree stages namely, RFID Reader Chip, PIC Microcontroller to decode the data into Serial o/p and wigend o/pto convert signal into TTL to RS232.

RFID Data Transmission in ASCII Standard

Data read from the tag is Manchester encoded. The Manchester encoded data is decoded to ASCII standard & the decoded data is sent to the UART serial interface for wired communication with the host systems. ASCII data format is shown below:

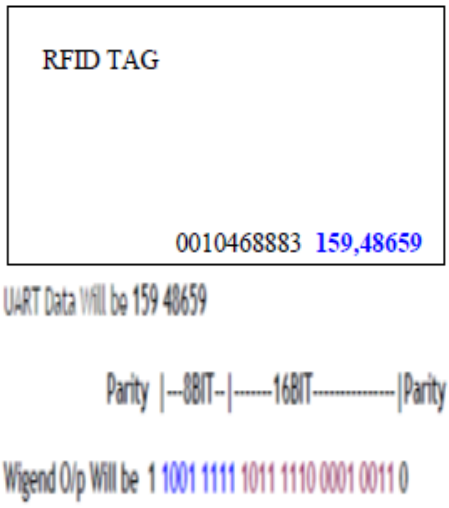


Fig. 14: RFID Tag &Wiegand Output

For example, if the cardisplaced on the reader, the output is as follows.

UART Data Will be 159 48659

Parity |---8BIT--|-------16BIT----------|Parity

Wigend O/p Will be 1 1001 1111 1011 1110 0001 0011 0

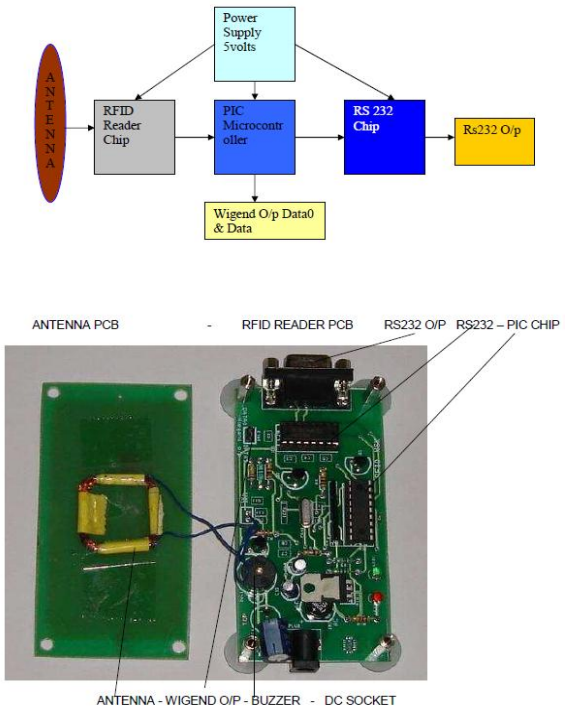


Fig.15: RFID Reader block diagram & Circuit layout

Data Transmission in Wiegand26 Standard: The reader module supports the Wiegand standard that gives the Wiegand encoded output. This output comprises of 3 bytes of data. It is indicated as low pulse on data line if it is a Data 1 signal and low pulse on the zero line if it is a Data 0 signal.

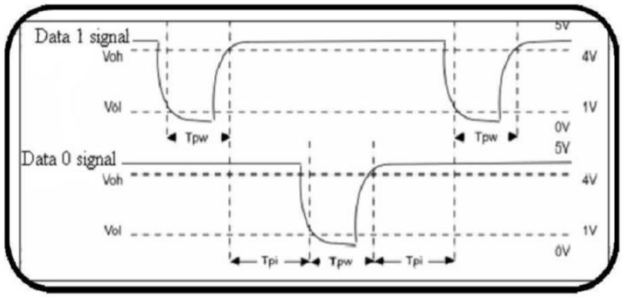


Fig.16: Wiegand Timing patterns

Figure 16 shows the pattern of data bits sent by the reader. This timing pattern falls within the Wiegand guidelines as prescribed by the SIA's Access Control Standard Protocol for the 26-bit Wiegand Reader Interface (a Pulse Width time between 20 µS and 100 µS, and a Pulse Interval time between 200 µS and 20 mS). The Data 1 and Data 0 signals are held at logic high level (above the Voh level) until the reader is ready to send a data stream. The reader places the data as asynchronous low-going (negative) pulses (below the Vol level) on the Data 1 or Data 0 lines to transmit the data stream to the access control panel. The Data 1 and Data 0 pulses do not overlap or occur simultaneously.

The composition of the open existing industry standard 26- bit Wiegand format contains 8 bits for the facility code field and 16 bits for the ID number field. Mathematically these 8 facility code bits allow a total of 256 (0 to 255) facility codes, while the 16 ID number bits allow a total of only 65,536 (0 to 65,535) individual ID's within each facility code [10].**RFID TAG 0010468883 159, 48659.**

Figure 17shows FPGA based Hybrid processor core (HPC) interfaced to a RFID card reader used for access control, security, attendance, inventory control etc, through MAX232 level converter. Using instruction 1001, the RFID reader is enabled or put to card reading mode. The valid card's unique codes are stored in HPC. Any RFID card swiped across the RFID reader will read the card unique ID code (printed on the card) & a comparison is made between the swiped card ID & the valid card database already stored in the HPC. If the match is found between the two, the swiped card ID is validated LCD on FPGA board will display a message-VALID CARD & swiped card number or ID (fig.18 (a)). If there is mismatch between the swiped card ID & the valid cad database IDs, the card is treated INVALD & LCD on FPGA board will display a message INVALID CARD & swiped card number or ID (fig.18 (b)). The (fig.18 (c)) indicates a RFID with card number 133, 39112 is validated by the FPGA based HPC & the same RFID card number is displayed in the LCD of the FPGA board.

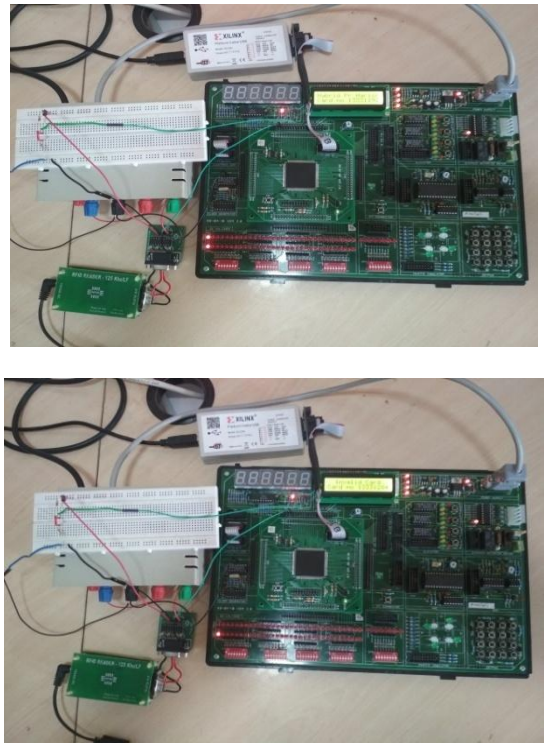
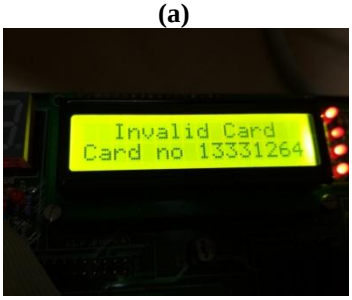
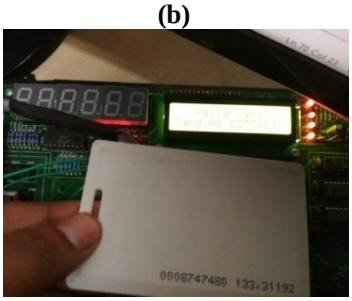


Fig. 17: FPGA based Hybrid processor core (HPC) interfaced to a RFID card reader.

(a)



(b)



(c)

Fig. 18: Evaluation of RFID card. (a) Valid RFID card, (b) Invalid RFID card and (c) RFID with card number 133, 39112 is validated by the FPGA

V. CONCLUSION

In continuation with previous work, implementation of popular instructions of the hybrid processor on FPGA is carried out. In this paper, the hybrid processor hardware core to interface popular processor hardware like UART, Hyper terminal, RFID Reader, GSM modem, 7 segment, LCD, Stepper motor, DC motor, etc., with single clock execution were used. The final solution will have multi-cores driving all possible soft instructions and hardware interfaces, simultaneously and working together in parallel. This putting it all together is a unique work. Also, design and implementation of two methods of smart auto switching of processor cores with load balancing for efficient traffic management, mentioned in detail in this paper is an innovative and novel research work in multi-core processor domain.

REFERENCES

1. Zeeshan Aslam, CA presentation of multicore processors, Dec 22, 2016.
2. https://slideplayer.com/slide/10678624/
3. A.S.V.Bala Krishna, Evolution of Multi-core Processors, International Journal of Latest Trends in Engineering and Technology, Vol. 3, Issue 1, September 2013.
4. Arvind Kumar and M. Valarmathi, "High Precision Stepper Motor Controller Implementation on FPGA with GUI on Lab VIEW", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Volume 2, no.4, April 2013.
5. Vaidehi M, T.R.Gopalakrishnan Nair "Multi-core Applications in Real Time Systems", Conference: Journal of Research and Industry, At Bangalore, Volume 1, December 2008.
6. Yi-Jung Chen, Wen-Wei Chang, Chia-Yin Liu, Cheng-En Wu, Bo-Yuan Chenand Ming-Ying Tsai, "Processors Allocation for MPSoCs with Single ISA Heterogeneous Multi-Core Architecture" IEEE Access, Volume 5, April 2017.
7. Muhammad Faisal Iqbal, Jim Holt, JeeHoRyoo, Gustavo de Veciana, and Lizy K. John, "Dynamic Core Allocation and Packet Scheduling in Multi-core Network Processors", IEEE Transactions on Computers, Volume 65, no. 12, Dec. 2016.
8. Neelappa and N.G.Kurahatti,"A survey of RFID reader leading to FPGA based RFID system", International Journal of Advanced Research in Electronics and Communication Engineering, Volume 4, no. 1, Jan 2015.
9. ShrutiHathwalia and Sansar Chand Sankhyan, " A Novel approach for displaying data On LCD using FPGA", International Journal of Technical Research and Applications, vol.1, no. 4, Oct 2013.
10. NSK Embedded KITS an Introduction Manual, Nsk Electronics, India, @Copyright 2007.
11. "AT Commands Reference Guide", Telit Wireless solutions, 80000ST10025a Rev. 24 – 2016-09-07.
12. Ken Chapman, Xilinx XAPP223, 200 MHz UART with Internal 16-Byte Buffer, Version 1.2, April 24, 2008.