International Journal of Multidisciplinary Research and Development

Volume: 2, Issue: 7, 91-96
July 2015
www.allsubjectjournal.com
e-ISSN: 2349-4182
p-ISSN: 2349-5979
Impact Factor: 3.762

Nilesh U. Kagde
Department of Electronics & Telecommunication, RMDSSOE, Savitribai Phule Pune University, Warje, Pune, 411058, India.

Chittaranjan P. Mahajan
Director, Dolphin labs Pune.

Correspondence:
Nilesh U. Kagde
Department of Electronics & Telecommunication, RMDSSOE, Savitribai Phule Pune University, Warje, Pune, 411058, India.

Designing and implementation of Port Dog using Xilinx Tool

Nilesh U. Kagde, Chittaranjan P. Mahajan

Abstract
Embedded system is having very important role in today's fastly developing in automation and industrial area. We can say embedded system is "Computer in Disguise" system, carrying along a computer or processor based system that which cannot be programmed by users. Hardware, software and firmware are some major components of embedded system and "Time to Market" is the key for successful launching of the product in today's rapid development world for capturing the market. The current trend for handheld device is to provide the users various embedded multimedia applications. These new applications constrain architecture developers to embed dedicated hardware accelerators in order to meet the application timing requirements. Hard real-time operating systems have been mostly designed for uniprocessors. In hard real-time systems not meeting timing constraints leads to system failure. These systems are used when it is imperative that an event is processed to within a strict deadline. Such strong guarantees are required in systems for which not reacting in a certain interval of time would affect the system and cause great loss. Examples of hard real-time systems include traffic control, industrial control, robotics, avionics, etc. [1] This paper presents one of the new concept of portdog for the hardware acceleration. To make the processor faster and to accelerate its processing speed we have to do multitasking. So for certain processes we can use this portdog for checking the outputs of real time system. For that particular time period we can use our main processor for some another task or we can us sleep mode for main processor. For this portdog can be used. Portdog can work as like co processor of the main processor. Using Xilinx the coading of this portdog can be completed and it can be implemented using FPGA Spartan 3E kit.

Keywords: Portdog, Real time system, microprocessor, embedded system, FPGA, Xilinx tool.

Introduction
Key for today's rapidly increasing development technology in various automation and other sectors is Automation and Information transfer. Embedded system is playing a very important and vital role in this rapidly developing system. We can define embedded system as "Computer in Disguise" system, carrying along a computer or processor based system that is not programmed by users. Major components of embedded system are hardware, software and firmware. "Time to Market" is the key for successful launching of the product in today's rapid development world. The current trend for handheld device is to provide the users various embedded multimedia applications. These new applications constrain architecture developers to embed dedicated hardware accelerators in order to meet the application timing requirements. However the use of dedicated monolithic hardware accelerators is onerous to achieve due to physical and economical constraints. When the multimedia applications share common functionalities, monolithic hardware accelerators could be split into smaller accelerators in order to cut down redundancy in hardware implemented functionalities and save silicon area. Nevertheless lowering the granularity of accelerator will increase synchronization calls between the main processor, and the accelerators.
Hard real-time operating systems have been mostly designed for uniprocessors. In hard real-time systems not meeting timing constraints leads to system failure. These systems are used when it is imperative that an event is processed to within a strict deadline. Such strong guarantees are required in systems for which not reacting in a certain interval of time would affect the system and cause great loss. Examples of hard real-time systems include traffic control, industrial control, robotics, avionics, etc. [1]
1.2 Acceleration of Hardware in Embedded System
The current trend for handheld device is to provide the users various embedded multimedia applications. These new applications constrain architecture developers to embed dedicated hardware accelerators in order to meet the application timing requirements. However the use

of dedicated monolithic hardware accelerators is onerous to achieve due to physical and economical constraints. When the multimedia applications share common functionalities, monolithic hardware accelerators could be split into smaller accelerators in order to cut down redundancy in hardware implemented functionalities and save silicon area. Nevertheless lowering the granularity of accelerator will increase synchronization calls between the main processor, and the accelerators.

Hard real-time operating systems have been mostly designed for uniprocessors. In hard real-time systems not meeting timing constraints leads to system failure. These systems are used when it is imperative that an event is processed to within a strict deadline. Such strong guarantees are required in systems for which not reacting in a certain interval of time would affect the system and cause great loss. Examples of hard real-time systems include traffic control, industrial control, robotics, avionics, etc. [1]

1.2 Acceleration of Hardware in Embedded System

The current trend for handheld device is to provide the users various embedded multimedia applications. These new applications constrain architecture developers to embed dedicated hardware accelerators in order to meet the application timing requirements. However the use of dedicated monolithic hardware accelerators is onerous to achieve due to physical and economical constraints. When the multimedia applications share common functionalities, monolithic hardware accelerators could be split into smaller accelerators in order to cut down redundancy in hardware implemented functionalities and save silicon area. Nevertheless lowering the granularity of accelerator will increase synchronization calls between the main processor and the accelerators.

Handheld devices integrate more and more functionality, and providing more multimedia applications is becoming a de facto requirement. Solutions are therefore needed for accelerating these computationally intensive applications in order to fulfil the requirements. The main acceleration approaches can be classified into two categories

A. A short portion of code is accelerated by extending the processor instruction set with a corresponding instruction. In this case the new instruction has typical execution latency from 1 to 4 cycles, thus limiting the size of the accelerated software. Developing longer instruction would make the pipeline execution flow inefficient.
B. Full application functionality is accelerated with a monolithic hardware accelerator used as peripheral device. The hardware accelerator is then synchronized with the application by the means of interrupts. In this case the hardware accelerator has typical execution latency from several thousand cycles up to several hundreds of thousands of cycles. However dedicated monolithic hardware accelerators are onerous to achieve due to physical and economical constraints [3].

The use of fine grained hardware accelerators has the advantage of saving silicon area by allowing collaborative use of common accelerated functionalities among several applications, thus cutting down implementation redundancy over several accelerators. For example, applications using reconfigurable media coding (RMC), where arbitrary combinations of algorithms may be assembled without pre-defined standardization, could easily take advantage of collaborative use of common accelerated functionalities. In such a case an access management system or dedicated scheduler is needed in order to avoid blocking state when two tasks would request the use of an accelerator at the same time. The study of such access management system or dedicated scheduler is however beyond the scope of this paper. This would restrict our study to a specific set of applications while we are here exclusively interested in analyzing the impact of short latency hardware accelerators on a typical embedded system. Splitting a monolithic hardware accelerator into several fine grained hardware accelerators can also in some cases permit a pipelined execution of the accelerators. Nevertheless, it transfers control complexity to the software running on the processor and as a consequence increases the synchronization frequency between the accelerators and the processor. Synchronization between an accelerator and the processor is needed to inform the processor about execution termination of the accelerator. As a result, the use of short latency hardware accelerators tends to amplify the interface cost used for synchronization between the processor and the hardware accelerators [3].

1. Literature Survey

The fundamental study reveals that the operations of embedded systems are partitioned in to hardware and software. This aspects suffers with various trade-offs in design metrics like to make system flexible more tasks must be performed in the software. The size of software thus increases. As size increase memory requirement increases which in turn increase the hardware size, power consumption and cost. The system throughput is affected by various inherent latencies generated due to processor architecture. To avoid this various steps are already taken like pipeline concept, Brach perdition etc. but still some challenges are required to pay attention such as size and weight solution is reduction in memory requirement, reduced OS footprints etc. another is speed or response time solution for this is custom instruction set, Hardware acceleration, multi core technique etc.

Limitations of traditional general processor architectures have resulted in the emergence of hybrid core computers. These computer systems integrate specialized hardware, called accelerators; to augment the generic CPU and can provide fast computations for certain classes of compute operations. The CPU offloads operations to the hardware accelerators such as GPUs or the FPGAs to perform certain operations that may run faster on these, and this in turn improves application performance. Accelerators have gained popularity recently and indeed many of the world's fastest supercomputers listed in the TOP500 list have hardware accelerators. For example, Tianhe-1A supercomputer has 7,168 NVIDIA Tesla M2050 general purpose GPUs. One of the challenges of hybrid core architecture is the effort required to port existing applications to these new system architectures. Large HPC applications typically have code sizes that exceed 100,000 lines, and porting them can take years of effort. Moreover, while in general an accelerator may benefit an application, the choice of the particular best accelerator device may not be readily apparent. Hence, before undertaking procurement, or time consuming reprogramming effort, one should explore the projected benefits from a given set of accelerator devices on the workload of interest. We show that performance models have a useful part to play in this evaluation and can help answer these questions accurately and speedily. To predict speedup and evaluate choice of accelerator device, one must first identify sections of existing source code that are potential candidates to run on accelerators. For identification of code sections we can leverage the fact that many HPC applications' computational and data access patterns can be expressed by a small set of commonly occurring idioms examples of idioms

include reduction, transpose, stencil and the like. Hence by identifying idioms one can identify instances of idioms that may run faster on the accelerator. Porting can then simply replace the selected idiom instances in source code to run on the selected accelerator [5].

Nowadays, with the amount of on-chip resources that can be exploited at any particular time limited by the so-called frequency, or voltage or power wall, the online specialization offered by partially reconfigurable FPGAs emerges as a promising way to combine computation in space and time to obtain the best performance per transistor count and unit of consumed energy. Indeed, the same on chip resources can be reused to efficiently implement different functionalities over time. Furthermore, the same flexibility that makes online specialization possible permits to solve or at least mitigate the reliability concerns that appear when pushing semiconductor manufacturing to its physical limits. Reconfigurable hardware allow for building systems capable of keeping their architecture fault free at all times by reconfiguring around damaged portions of the chip. However, despite the numerous advantages FPGAs are to bring, their success is highly conditioned by the way they reach application developers. After decades of prevalence, software programming style has spread through all application domains, including even those which were traditionally hardware-cantered, and thus, it seems impossible to radically change this situation without a major collapse in productivity and increase in cost. Faced with this, the concept of a reconfigurable operating system (ROS) to give to reconfigurable hardware a "software look and feel" has been gaining momentum since the mid-1990s [7]. Field programmable gate array (FPGA) based hardware accelerators are a viable option available to designers looking to improve performance of large software systems. The high performance computing domain contains many applications that take days or weeks to run and can benefit from a hardware implementation, including molecular dynamics simulations, genetic string matching, and XML query matching. While a custom hardware implementation will provide the most benefit, this approach is very expensive and time-intensive and not cost effective for most designers. A wide variety of platforms incorporating both microprocessors and FPGAs are now available. The main difficulty of using FPGAs as hardware accelerators is in the programming of the configurable hardware. Hardware is commonly programmed with low level hardware description languages such as VHDL and Verilog. These languages require low-level knowledge such as complete timing information and can be tedious and error-prone to program in. Software application designers are typically unfamiliar with the requirements of a hardware design and have difficulty utilizing the hardware optimally [6].

2. Proposed Concept of PortDog

Hardware acceleration is important factor in design of embedded real time applications. In these devices time constraints for completion of tasks or events in the system speed of processor should be high. Many time processors have to check the inputs which are from real time physical system or environment. So that for that particular task processor get busy and it cannot Handel other task at that time. So that energy loss takes place in system. Total system will need more energy. To avoid this situation portdog. concept is proposed in this paper. It will work like dog. As dog give signal to human in his language if thief is around same way this portdog will give signal if some change occurs as per our desired change in the real time physical system. So that this portdog can be used as coprocessor of the main processor. This will continuously check for the input signal from the real time system which we want to control. By continuous checking of input signal if some change occurs then this portdog will give output signal which we can use as control signal for the main processor. This protdog is to be implemented on hardware so that its speed will be more than that of software because software takes more time to execute program that that of hardware. Hardware execution is much faster than software so that acceleration and speed of system get increase and that is most important for today's rapidly development in embedded system devices. Due to this efficiency of system increase and also less power is required. So this portdog concept will increase the efficiency of system, save the power of system so that energy consumption will be reduced. This concept is shown in below figure 1.

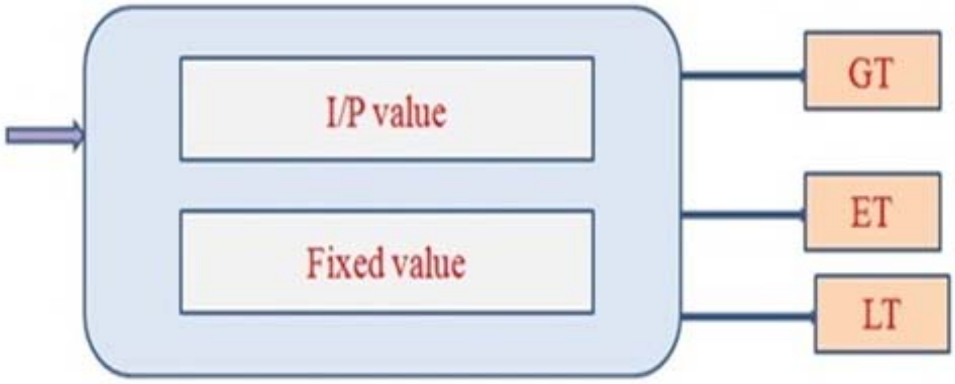
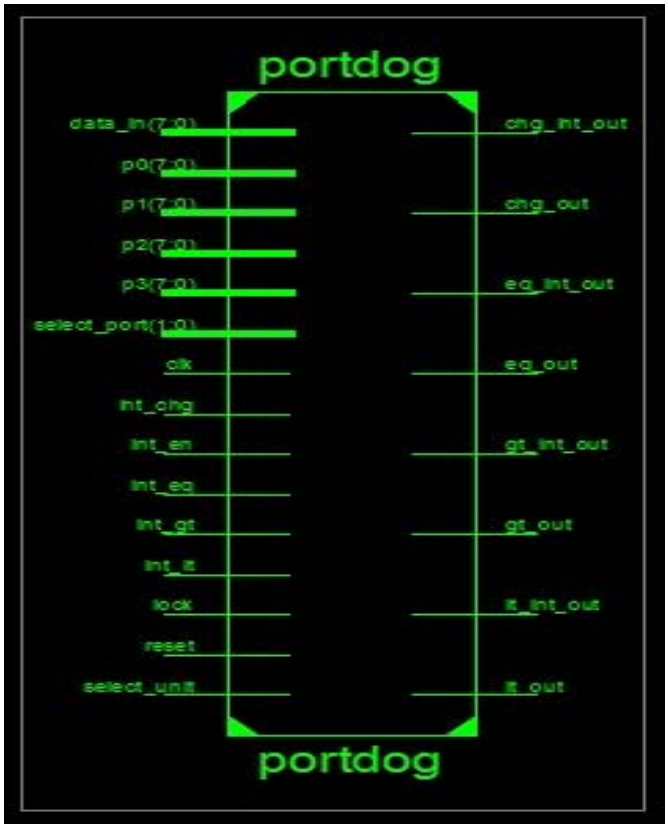


Fig 1: Conceptual block diagram of Portdog

[Courtesy Mr. Chittaranjan Mahajan, Dolphin labs, Pune]

Figure 1 shows the basic concept of portdog. Here we can measure the given input with our fixed input. Depend upon that we can give output as greater than, less than or equal to. So depend on our requirement we can use any one or all signals for our controlling system. In this port dog we can also check the status of ports. Due to this we can make our main system in sleep or ideal mode as per requirement and so that power consumption of system will be less. If we want we can assign another task to system during the period of this checking time. We our required signal if there from portdog that time we can bring our system in active mode. Due to this power consumption will be reduced and efficiency of system will be increase.

3. Implementation of System



This concept of PortDog is designed using Xilinx tool. As FPGA is more useful for full custom design we have used this. Designing is done in Xilinx 13.4 and it is tested using Spartan 3E FPGA. Figure 2 above shows the RTL schematic for the PortDog.

4.1 Synthesis Tool

a. Xilinx Ise 13.4

Xilinx, Inc. is the world's largest supplier of programmable logic devices, the inventor of the field programmable gate array (FPGA) and the first semiconductor company with a fables manufacturing model.

Xilinx designs, develops and markets programmable logic products including integrated circuits (ICs), software design tools, predefined system functions delivered as intellectual property (IP) cores, design services, customer training, field engineering and technical support. Xilinx sells both FPGAs and CPLDs programmable logic devices for electronic equipment manufacturers in end markets such as communications, industrial, consumer, automotive and data processing.

Xilinx's FPGAs have even been used for the ALICE (A Large Ion Collider Experiment) at the CERN European laboratory on the French-Swiss border to map and disentangle the trajectories of thousands of subatomic particles.

The Virtex-II Pro, Virtex-4, Virtex-5, and Virtex-6 FPGA families are particularly focused on system-on-chip (SoC) designers because they include up to two embedded IBM PowerPC cores.

The ISE Design Suite is the central electronic design automation (EDA) product family sold by Xilinx. The ISE Design Suite features include design entry and synthesis supporting Verilog or VHDL, place-and-route (PAR), completed verification and debug using Chip Scope Pro tools, and creation of the bit files that are used to configure the chip.

XST-Xilinx Synthesis Technology performs device specific synthesis for CoolRunner XPLA3/-II and XC9500/XL/XV families and generates an NGC file ready for the CPLD fitter.

The general flow of XST for CPLD synthesis is the following:

1. HDL synthesis of VHDL/Verilog designs
2. Macro inference
3. Module optimization
4. NGC file generation

3.2 Hardware Required

FPGA Sparten 3E

4.2.1 FPGA Design Flow

The designer facing a design problem must go through a series of steps between initial ideas and final hardware. This series of steps is commonly referred to as the ‘design flow’. First, after all the requirements have been spelled out, a proper digital design phase must be carried out. It should be stressed that the tools supplied by the different FPGA vendors to target their chips do not help the designer in this phase. They only enter the scene once the designer is ready to translate a given design into working hardware.

The most common flow nowadays used in the design of FPGAs involves the following subsequent phases:

1. Design entry:

This step consists in transforming the design ideas into some form of computerized representation. This is most commonly accomplished using Hardware Description Languages (HDLs). The two most popular HDLs are Verilog and the Very High Speed Integrated Circuit HDL (VHDL) [2]. It should be noted that an HDL, as its name implies, is only a tool to describe a design that pre-existed in the mind, notes, and sketches of a designer. It is not a tool to design electronic circuits. Another point to note is that HDLs differ from conventional software programming languages in the sense that they don’t support the concept of sequential execution of statements in the code. This is easy to understand if one considers the alternative schematic representation of an HDL file: what one sees in the upper part of the schematic cannot be said to happen before or after what one sees in the lower part.

2. Synthesis:

The synthesis tool receives HDL and a choice of FPGA vendor and model. From these two pieces of information, it generates a netlist which uses the primitives proposed by the vendor in order to satisfy the logic behaviour specified in the HDL files. Most synthesis tools go through additional steps such as logic optimization, register load balancing, and other techniques to enhance timing performance, so the resulting netlist can be regarded as a very efficient implementation of the HDL design.

3. Place and route:

The placer takes the synthesized netlist and chooses a place for each of the primitives inside the chip. The router’s task is then to interconnect all these primitives together satisfying the timing constraints. The most obvious constraint for a design is the frequency of the system clock, but there are more involved constraints one can impose on a design using the software packages supported by the vendors.

4. Bit stream generation:

FPGAs are typically configured at power-up time from some sort of external permanent storage device, typically a flash memory. Once the place and route process is finished, the resulting choices for the configuration of each programmable element in the FPGA chip, be it logic or interconnect, must be stored in a file to program the flash. Of these four phases, only

the first one is human-labour intensive. Somebody has to type in the HDL code, which can be tedious and error-prone for complicated designs involving, for example, lots of digital signal processing. This is the reason for the appearance, in recent years, of alternative flows which include a preliminary phase in which the user can draw blocks at a higher level of abstraction and rely on the software tool for the generation of the HDL. Some of these tools also include the capability of simulating blocks which will become HDLs with other blocks which provide stimuli and processing to make the simulation output easier to interpret. The concept of hardware co-simulation is also becoming widely used. In co-simulation, stimuli are sent to a running FPGA hosting the design to be tested and the outputs of the design are sent back to a computer for display (typically through a Joint Test Action Group (JTAG), or Ethernet connection). The advantage of co-simulation is that one is testing the real system, therefore suppressing all possible misinterpretations present in a pure simulator. In other cases, co-simulation may be the only way to simulate a complex design in a reasonable amount of time.

4.2.2 Advantages of Using Hardware Description Languages (HDLs) to Design FPGA Devices

Using Hardware Description Languages (HDLs) to design high-density FPGA devices has the following advantages:

a. Top-Down Approach for Large Projects Designers use HDLs to create complex designs. The top-down approach to system design works well for large HDL projects that require many designers working together. After the design team determines the overall design plan, individual designers can work independently on separate code sections.
b. Functional Simulation Early in the Design Flow You can verify design functionality early in the design flow by simulating the HDL description. Testing your design decisions before the design is implemented at the Register Transfer Level (RTL) or gate level allows you to make any necessary changes early on.
c. Synthesis of HDL Code to Gates Synthesizing your hardware description to target the FPGA implementation:
d. ♦ Decreases design time by allowing a higher-level design specification, rather than specifying the design from the FPGA base elements.
e. ♦ reduces the errors that can occur during a manual translation of a hardware description to a schematic design.
f. ♦ allows you to apply the automation techniques used by the synthesis tool (such as machine encoding styles and automatic I/O insertion) during optimization to the original HDL code. This results in greater optimization and efficiency.

4. Simulation Results

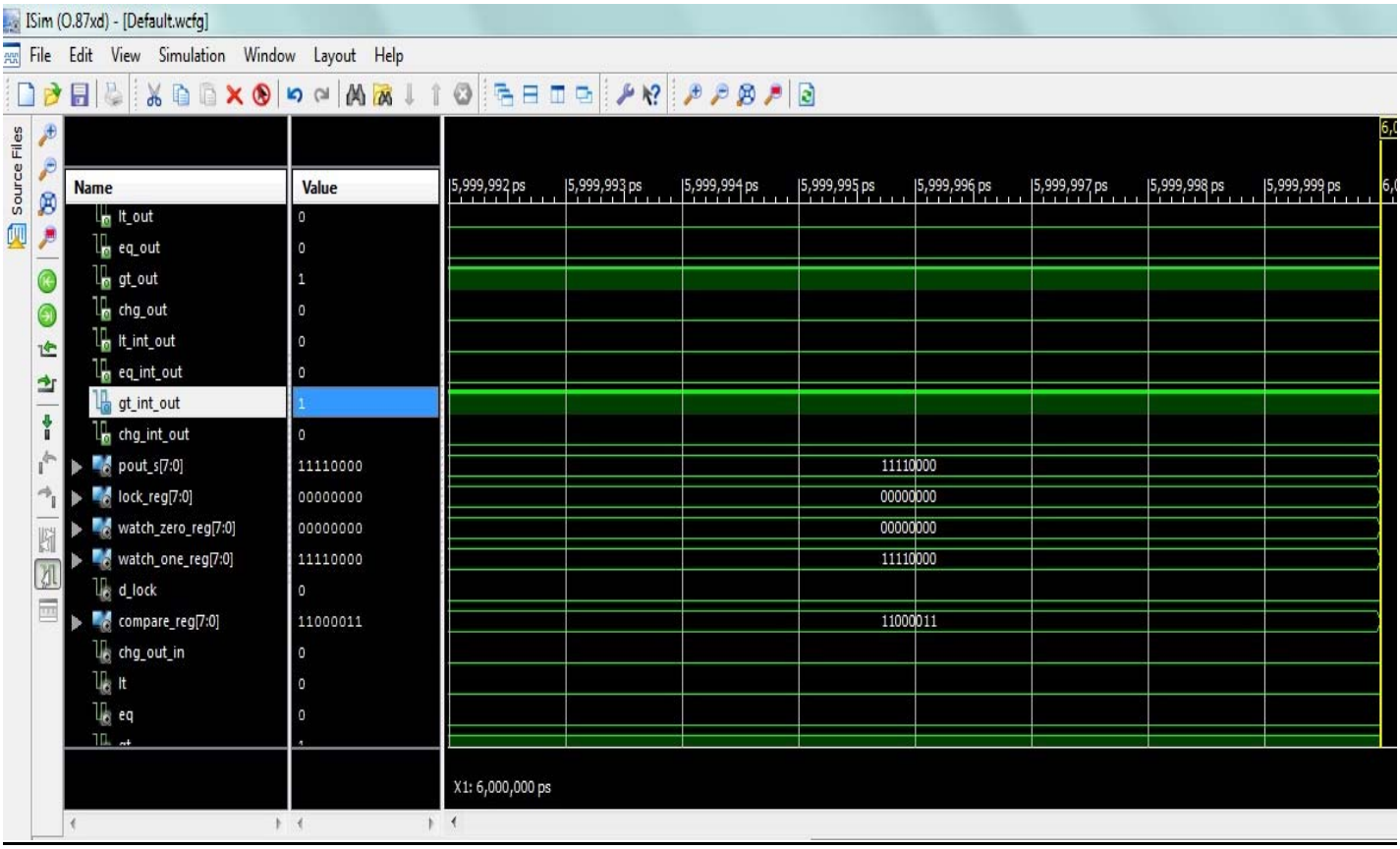


Fig 3: Simulation Output

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	25	4,896	1%
Number of 4 input LUTs	59	4,896	1%
Number of occupied Slices	38	2,448	1%
Number of Slices containing only related logic	38	38	100%
Number of Slices containing unrelated logic	0	38	0%
Total Number of 4 input LUTs	59	4,896	1%
Number of bonded IOBs	59	158	37%
IOB Flip Flops	8		
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	2.09		

Fig 4: Device Utilization

5. Conclusion

This paper has presented the importance of hardware acceleration for today's rapidly developing embedded system devices. Speed is important factor of the system. So that hardware should be work fast for better efficiency of the system. Hardware acceleration is most important for this. For increasing the efficiency of processor we can use portdog for some part of the input signal checking and its controlling. Protdog can be used as coprocessor of the main processor. Simulation results show that design implemented works properly.

6. Acknowledgement

This proposed concept of Portdog is part of patent numbered "4139/MUM/2013" dated "31/12/2013" by "Chittaranjan Pramod Mahajan, Pune"(Director, Dolphin labs Pune), Related to "Port dog/ Peripheral dog to monitor ports/ on chip peripherals present on microcontroller".

7. References

1 Andre Nogueira, Mario Calha, "Predictability and Efficiency in Contemporary Hard RTOS for Multiprocessor Systems", 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications.

2 Thomas A. Henzinger and Joseph Sifakis, "The Embedded System Design Challenge".

3 Sebastien Lafond, Johan Lilius, "Interrupt Costs in Embedded System with Short Latency Hardware Accelerators", 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems.

4 "Adding Hardware Accelerators to Reduce Power in Embedded Systems", White Paper by ALTERA, September 2009.

5 Mitesh R. Meswani, Laura Carrington, Didem Unat, "Modeling and Predicting Performance of High Performance Computing Applications on Hardware Accelerators", IEEE 26th International Parallel and Distributed Processing Symposium Workshops, 2012.

6 Jason Villarreal, Adrian Park, Walid Najjar, Robert Halstead, "Designing Modular Hardware Accelerators in C With ROCCC", 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, 2010.

7 Xabier Iturbe, Khaled Benkrid, Chuan Hong, Ali Ebrahim, Raul Torrego, Imanol Martinez, Tughrul Arslan, Jon Perez, "R3TOS: A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive, Efficient, and Dependable Computing on FPGAs", IEEE Transactions on computers, Vol. 62, NO. 8, AUGUST 2013.

8 Marcello Lajolo, Anand Raghunathan, Sujit Dey, and Luciano Lavagno, "Co simulation-Based Power Estimation for System-on-Chip Design", IEEE transactions on very large scale integration (VLSI) systems, vol. 10, no. 3, June 2002.

9 David Boland and George A, "A Scalable Precision Analysis Framework", IEEE transactions on multimedia, vol. 15, no. 2, February 2013.

10 Joao Bispo, Nuno Paulino, Joao M. P. Cardoso, and Joao C. Ferreira, "Transparent Trace-Based Binary Acceleration for Reconfigurable HW/SW Systems", IEEE transactions on industrial informatics, vol. 9, no. 3, August 2013.

11 Scott Mahlke, Rajiv Ravindran, Michael Schlansker, Robert Schreiber, and Timothy Sherwood, "Bitwidth Cognizant Architecture Synthesis of Custom Hardware Accelerators", IEEE transactions on computer-aided design of integrated circuits and systems, vol. 20, no. 11, November 2001.

12 Khai Lik, Khoo, Mohd, Fadzil Ain, Chee Hak, Teh , Weng Li, Leow, "Scalable Storage Architecture in Modular Hardware Accelerators", 2011 IEEE Symposium on Industrial Electronics and Applications (ISIEA2011), September 25-28, 2011, Langkawi, Malaysia.

13 WWW.Xillinx.com Xilinx Spartan-3 FPGA Family Data Sheet, DS099-2