Journal of Advanced Research in Technology and Management Sciences
Volume: 01 Issue: 03 ISSN: 2582-3078 November 2019
Available online at: http://www.jartms.org

Performance of FPGA in an Enhanced Level of Watchdog Timer

R.Haripriya[1] , R.Sentamilselvam[2]

[1]II^{yr} M.Tech, Department of Electronics and Communication Engineering, Chadalawada Ramanamma Engineering College, Tirupati, Andhra Pradesh, India.
[2]Assistant Professor, Department of Electronics and Communication Engineering, Chadalawada Ramanamma Engineering College, Tirupati, Andhra Pradesh, India.
JNTU-Anantapur, Andhra Pradesh, India.
[1] ravvaharipriya@gmail.com.
[2]senthamilselvan87@gmail.com.

Abstract: Embedded systems that are employed in safety-critical applications require highest reliability. External watchdog timers are used in such systems to automatically handle and recover from operation time related failures. Most of the available external watchdog timers use additional circuitry to adjust their timeout periods and provide only limited features in terms of their functionality. This paper describes the architecture and design of an improved configurable watchdog timer that can be employed in safety-critical applications. Several fault detection mechanisms are built into the watchdog, which adds to its robustness. The functionality and operations are rather general and it can be used to monitor the operations of any processor based real-time system. This paper also discusses the implementation of the proposed watchdog timer in a Field Programmable Gate Array (FPGA). This allows the design to be easily adaptable to different applications, while reducing the overall system cost. The effectiveness of the proposed watchdog timer to detect and respond to faults is first studied by analyzing the simulation results.

Keywords: FPGA, Watch Dog Timer (WDT), Hardware Description Language (HDL), Input and Output Interface, Verilog and VHDL.

I. INTRODUCTION

For applications where a system crash could lead to human injury, highest reliability is required. Such systems should have fault tolerance mechanisms that account for the unexpected to ensure proper safety of operation. These systems should also be able to recover from a crash without any human assistance. These fault tolerance mechanisms detect when a fault occurs in order to handle the fault and to limit the system downtime [1]. One way to achieve fault tolerance is by implementing system redundancy. By using multiple copies of the critical components of the system, the overall system reliability is enhanced [2]. However, this improved system reliability is achieved through increased hardware and software complexity, depending on the type of architecture used. When developing a fault-tolerant system, one of the most cost effective ways of detecting and handling operation time related failures is the watchdog [3]. A watchdog timer (WDT) is a hardware subsystem that monitors the operations of the system and takes certain actions in the event of detecting a fault [4]. It typically consists of a timer circuit and the processor is required to periodically reset the timer. If the WDT expires, it is a secondary indication of some problem with the system under observation [5]. When the processor fails to reset the watchdog, a decision is made to restart the system or put the system into a known state from which it can recover, thus preventing further damages. A watchdog can be internal (on-chip) or external to the processor. Internal watchdog reduces the hardware complexity and cost, however, is not a robust solution. The software has control over it during runtime and a runaway code can disable the watchdog timer [3]. Moreover, since it is connected to the processor clock, a crystal failure will make the watchdog incapable of monitoring the hardware for faults [6]. When the reliability of an embedded system is crucial, external watchdogs become unavoidable. An external watchdog runs independent of the processor and does not share its clock with the processor. This overcomes the limitations of internal watchdogs and leads to much more robust fault-tolerant system architectures [7].A class of standalone watchdog timer microchips offer only fixed timeout periods, which make them less generic. Other set of devices allow adjusting the timeout periods by using additional external circuitry. Though useful, this method adds to the complexity of the hardware and increases the overall system cost. The increased cost and complexity of external watchdogs can be managed to a certain extend by realizing the watchdog functionality within a Field Programmable Gate Array (FPGA). Many of the modern embedded systems in corporate one or more FPGA devices to accomplish the desired system functionality [8]. Accommodating the watchdog timer within a FPGA can yield an efficient and robust solution. The work done by Giaconia et al. [9] considered the implementation of a custom concurrent watchdog processor in FPGA for real-time control systems. The design did not provide a timer for the processor; rather, it performed a reasonableness check on some variables and a basic program flow check. El-Attar et al. [10] proposed a sequenced watchdog timer that used time registers to determine whether or not fault has occurred. However, it did not offer much configuration options and the fault detection features implemented were limited. In [11] the authors addressed the basic concepts of a multiple hardware watchdog timer system in FPGA, but kept the design of the watchdog simple. In this paper, we propose the design of an improved windowed watchdog timer and its implementation in FPGA. Realizing the design in FPGA means that the same watch-dog hardware can be interfaced to different processors and systems, with only minor modifications of the associated hardware description language (HDL) code [8]. It also allows for accommodating multiple watchdog timers for multicore architectures. The proposed watchdog timer is well suited for safety-critical embedded systems, where redundant channels are employed to enhance the system reliability. Designing the WDT as a reusable IP core also addresses the component obsolescence issues faced by many embedded systems, especially those in the aerospace and military applications [12].The paper describes the architecture of the proposed watchdog timer, the fault detection features, and its implementation in FPGA.The remainder of this paper is organized as follows. The following section introduces the architecture of the proposed watchdog timer. The fault detection mechanisms built into the watchdog is discussed in III. Section IV describes the implementation of the watchdog timer in FPGA. Simulation results and evaluation of the design in hardware are detailed in section V. Finally, section VI concludes this paper.

II.LITERATURE SURVEY

The Design of a Fault-Tolerant COTS-Based Bus Architecture

In this paper, we report our experiences and findings on the design of a fault-tolerant avionics bus architecture comprised of two COTS buses, the IEEE 1394 and the I2C, for the X2000 program at the Jet Propulsion Laboratory. The X2000 design team is judicious about ensuring the fault tolerance provisions will not cause the bus design to deviate from commercial standard specifications, so that the economic attractiveness of using COTS will not be diminished. The hardware and software designs of the X2000 fault-tolerant bus are being implemented and flight hardware will be delivered to the Europa Orbiter missions. This effort provides an example of how to construct a highly reliable system with low cost COTS buses.

Fault-Tolerant Platforms for Automotive Safety-Critical

Fault-tolerant electronic sub-systems are becoming a standard requirement in the automotive industrial sector as electronics becomes pervasive in present cars. We address the issue of fault tolerant chip architectures for automotive applications. We begin by reviewing fault-tolerant architectures commonly used in other industrial domains where fault tolerant electronics has been a must for a number of years, e.g., the aircraft manufacturing industrial sector. We then proceed to investigate how this architecture could be implemented on a single chip and we compare them with a metric that combines traditional terms such as cost, performance and fault coverage with flexibility, i.e. the ability of adapting to changing requirements and capturing a wide range of applications, an emerging criterion for platform design. Finally, we describe in some details a cost effective dual lockstep platform that can be used as a single fail-operational unit or as two fail-silent channels trading fault-tolerance for performance.

A Review of Watchdog Architectures and their Application to Cubesats

The push towards”faster, better, cheaper” missions with an emphasis on design reuse has led to the popularity of COTS products that have traditionally relied upon custom software and hardware. The motivation for moving towards COTS hardware is faster, cheaper and more powerful processors with lower power consumption compared to those available in hard versions. While moving towards COTS components has many advantages, it also requires developing mechanisms to protect the components from radiation harsh environments which high altitude and space systems are subject to

Concurrent Error Detection Using Watchdog Processors-A Survey

This is a survey of concurrent system-level error detection techniques using a watchdog processor. A watchdog processor is a small and simple coprocessor that detects errors by monitoring the behavior of a system. Like replication it does not depend on any fault model for error detection. However, it requires less hardware as compared to replication. It is shown that a large number of errors can be detected by monitoring the control flow and memory access behavior. Two techniques of control flow checking are discussed and compared to the current error detection techniques. A scheme for memory access checking based on capability-based addressing is described. The design of a watchdog for performing reasonableness checks on the output of a main processor, by executing assertions, is also discussed.

III.FPGAS

This paper will describe the use of digital Field Programmable Gate Arrays (FPGA) to contribute to advancing the state-of-the-art in software defined radio (SDR) transponder design for the emerging Small Sat and Cube Sat industry and to provide advances for NASA as described in the TAO5 Communication and Navigation Roadmap (Ref 4). The use of software defined radios (SDR) has been around for a long time. A typical implementation of the SDR is to use a processor and write software to implement all the functions of filtering, carrier recovery, error correction, framing etc. Even with modern high speed and low power digital signal processors, high speed memories, and efficient coding, the compute intensive nature of digital filters, error correcting and other algorithms is too much for modern processors to get efficient use of the available bandwidth to the ground. By using FPGAs, these compute intensive tasks can be done in parallel, pipelined fashion and more efficiently use every clock cycle to significantly increase throughput while maintaining low power. These methods will implement digital radios with significant data rates in the X and Ka bands. Using these state-of-the-art technologies, unprecedented uplink and downlink capabilities can be achieved in a 1/2 U sized telemetry system[15]. Additionally, modern FPGAs have embedded processing systems, such as ARM cores, integrated inside the FPGA allowing mundane tasks such as parameter commanding to occur easily and flexibly. Potential partners include other NASA centers, industry and the DOD. These assets are associated with small satellite demonstration flights, LEO and deep space applications. MSFC currently has an SDR transponder test-bed using Hardware-in-the-Loop techniques to evaluate and improve SDR technologies.

FPGA Implementation of an Improved Watchdog Timer for Safety-Critical Applications

Embedded systems that are employed in safety critical applications require highest reliability. External watchdog timers are used in such systems to automatically handle and recover from operation time related failures. Most of the available external watchdog timers use additional circuitry to adjust their timeout periods and provide only limited features in terms of their functionality [13]. This paper describes the architecture and design of an improved configurable watchdog timer that can be employed in safety-critical applications. Several fault detection mechanisms are built into the watchdog, which adds to its robustness. The functionality and operations are rather general and it can be used to monitor the operations of any processor based real-time system. This paper also discusses the implementation of the proposed watchdog timer in a Field Programmable Gate Array (FPGA). This allows the design to be easily adaptable to different applications, while reducing the overall system cost. The effectiveness of the proposed watchdog timer to detect and respond to faults is first studied by analyzing the simulation results [16]. Thus after designing the watchdog it is implemented in ATM and verified. The design is validated in a real-time hardware by injecting faults through the software while the processor is executing.

“Comparison of internal and external watchdog timer’s application note”

This article compares the benefits and drawbacks of internal (integrated with the microprocessor) and external (hardware-based) watchdog timers (WDTs). It explains that internal watchdog timers are easy to implement, but subject to failure. The MAXQ2000 microcontroller's WDT serves as an example of an internal watchdog. Hardware-based watchdog timers require additional board space, but are indispensable when reliability is paramount. This article concludes with a comparison matrix that highlights the main advantages and disadvantages of each WDT solution.

III. PROPOSED SYSTEM

An effective watchdog should be able to detect all abnormal software modes and bring the system back to a known state. It should have its own clock and should be capable of providing a hardware reset on timeout to all the peripherals [3]. The watch-dog timer proposed in this paper operates independently of the processor and uses a dedicated clock for its functions. The architecture follows a windowed watchdog implementation, where the window periods can be configured by the software during initialization. A fail flag is raised when the watchdog timer expires and after a fixed amount of time from raising the flag, a reset is triggered. The time in-between can be used by the software to store valuable debugging information to anon-volatile medium.A standard watchdog timer can catch problems in the system such as hanging because of endless loops in code execution. However, the main disadvantage of this watchdog is that if the system enters a fault state in which it continually resets the timer, the error state will never be detected. In other words, a standard watchdog timer can detect slow faults, but cannot detect fast faults which occur within the watchdog timer period [13]. However, a windowed architecture can handle this properly. Here the watchdog defines a small time window within which the watchdog must be reset in order to avoid a timeout. This provides protection against systems from running too fast and too slow [14], thus increasing the error recognition coverage. A. I/O Interface and Configuration Fig. 1 shows the input-output (I/O) interface of the pro-posed watchdog timer. The watchdog has two outputs, namely the watchdog fail output (WDFAIL) and the reset output (RSTOUT). When the SYSRESET input is low, the WDFAIL output remains asserted and the RSTOUT output stays de-asserted. The design also consists of a configuration register with bit fields defined as in the figure. The register enables adjustments to the watchdog parameters and also provides status information. The WDRST and WDSRVC fields are used respectively for resetting and servicing the watchdog. The state of the INIT input and the WDFAIL output are automatically updated in the configuration register. The SWSTAT field holds the state of the service window and the FLSTAT field logs the watchdog failure mode, if any. The control inputs to the watchdog timer, ENABLE and RD/WR, permit thread and write to the configuration register. The ABUS and DBUS signals in the figure indicate address bus and data bus, respectively.

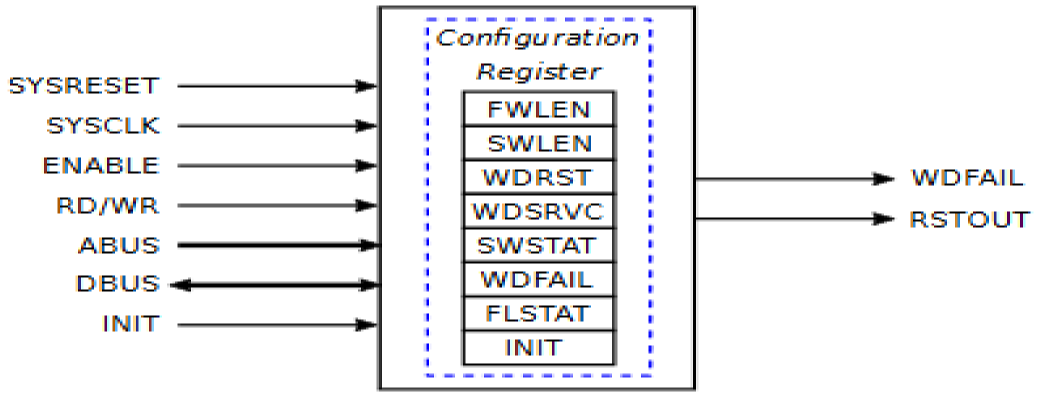


Figure 1: windowed watchdog design

Watchdog Timer Initialization

On power-up or reset the watchdog wakes up in a failed state,i.e., the WDFAIL output will be asserted high. It is the responsibility of the software to initialize the watchdog and keep it running. Fig. 2 illustrates the waveform for watchdog reset initialization and general operation. In order to bring the watchdog to a working state, first the watchdog reset (WDRST) field in the configuration register must be toggled from low-to-high [19]. This, followed by servicing the watchdog inside the service window, will de-assert the WDFAIL flag and make it operational. Since the frame window is kept larger than the system frame time, another service window will start before the current frame window expires. When the watchdog is again properly serviced, the frame window will be reinitialized. As long as the frame window counters keep running, no failures will be flagged by the watchdog particularly suitable for embedded systems that schedule their tasks in frames.

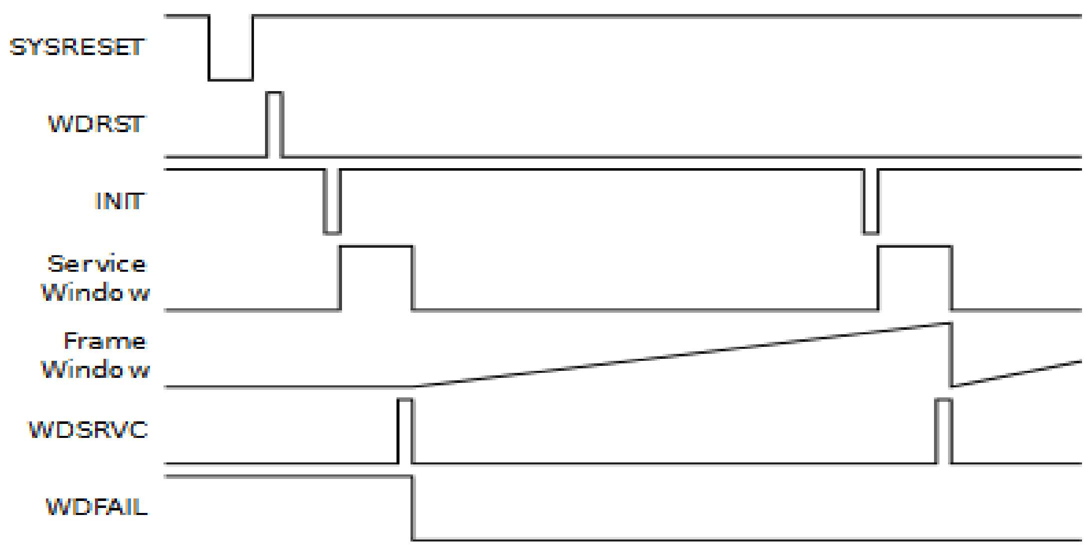


Fig. 2. Watchdog timer initialization and service operation

IV. FAULT DETECTION FEATURES

Several fault detection mechanisms are built into the pro-posed watchdog timer in order to improve its effectiveness in capturing erratic software modes. When the software fails to service the watchdog inside the service window, the window expires and sets a fail flag internally [12]. In this case, the frame window does not reinitialize and expires upon reaching its terminal value. On the expiry of the frame window the watchdog asserts its WDFAIL signal, indicating a failure. This failure mode is depicted in Fig. 3

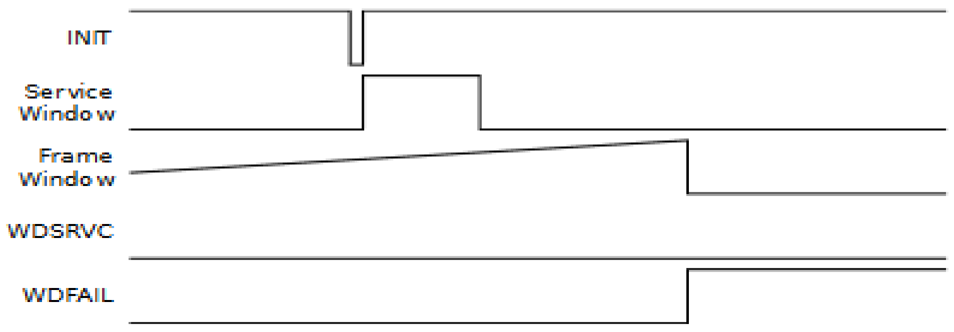


Fig. 3. Watchdog fail due to frame window expiry

A watchdog fail will occur when the software services the watchdog outside the service window, as shown in Fig.4. It can be seen that the invalid service operation instantly terminates the frame window and asserts the WDFAIL signal.A favorable consequence of this feature is that two successive service operations will also lead to a watchdog fail. Here, the first service operation will immediately close the service window and the next one will invariably occur outside the window [15]. This becomes equivalent to servicing the watchdog outside the service window and leads to a watchdog failure

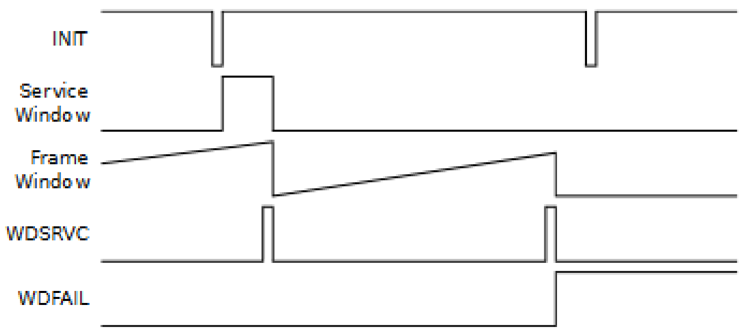


Fig. 4. Watchdog fail due to service outside the service window

Fig. 2 illustrates a scenario where the WDSRVC falling edge is occurring inside the service window. This is also considered as an illegal service operation and the watchdog fails signal is asserted. This implies that, after servicing the watchdog, the software is required to de-assert the WDSRVC signal before the start of the next service window [17]. All of these fault detection mechanisms ensure that a software running haywire will not go undetected by the proposed watchdog timer.

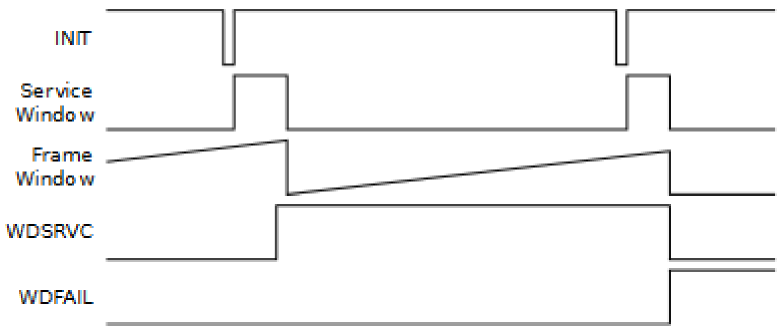


Fig. 5. Watchdog fail due to WDSRVC falling edge inside service window

The WDFAIL output from the watchdog timer can be used to activate a fail-safe state, or warn the processor of the fault by issuing a non-maskable interrupt (NMI) signal. After a predefined amount of time from asserting the WDFAIL output the watchdog will assert its RSTOUT output. This signal canbe tied to the reset pin of the processor, causing it to reset the embedded system automatically [16]. The intervening time will give the software an opportunity to save information that can be valuable for debugging. In the event of a failure, the corresponding failure mode will be logged in the FLSTAT field in the watchdog configuration register. The software can attempt to save this information also to a non-volatile memory for debugging purposes.

V. WATCHDOG TIMER IMPLEMENTATION IN FPGA

This section details the realization of the proposed watchdog timer in FPGA. The high-level diagram of the watchdog hardware is shown in Fig. 6. The design is clocked by its SYSCLK input, which is independent of the processor clock.The possible sets of window lengths are arrived based on the application and hard-coded in the design. These values can be selected by writing to the appropriate bits in the configuration register - SWLEN for the service window and FWLEN for the frame window - after power-on. Once the values are selected, the window length configuration fields get locked automatically; i.e., writes to these bits are disabled. For the cases where the window lengths have to be modified again, a 16-bit unlock register is provided in the design [10]. In order to change the window lengths, the software will have to perform two successive writes to this register with data 0xAAAA and 0x5555. Subsequent to writing the first pattern the second one must be written within 10µs, after which the software gets a 10µs period to modify the length configuration fields. If these timings are not strictly met, writes to these bits will remain disabled. The service window is started when a high-to-low transition is detected on the INIT signal. The service window uses a derived clock (SWCLK) that is much slower than the SYSCLK. The slower clock helps in reducing the number of comparators required, thus minimizing the resource utilization in FPGA. The service windows have an offset up/down counterthat are clocked by the SYSCLK, and a main counter that runs at SWCLK. The offset up counter finds the offset (Toffset) between the INIT input and the next rising edge of the SWCLK. This is necessary as the INIT signal may be asynchronously driven and can come at any time within the SWCLK period, Tswclk. The offset value is saved and the main counter is started, which then runs for (SWLEN - 1) times. Once the main counter expires, the offset down counter runs for a durationTswclk−Toffset [18]. This counting procedure allows for a precise control over the window length. The running status of the service window is also updated in the watchdog configuration register periodically. When the watchdog is correctly serviced, the counters in the service window stop immediately and the frame window starts. The frame window also uses a derived slower clock (FWCLK) for its operations. It has an offset up/down counter and a main counter with functionalities similar to that of the service window. The offset up counter here finds the offset between the termination of the service window and the next rising edge of the FWCLK. The main counter counts for (FWLEN - 1) times which is then followed by the offset down counter. The frame window counters reset when a watchdog service operation occurs within the next service window duration, before the frame window expires. A. Reset Initialization and Fault Detection the diagram in Fig. 7 shows the finite state machine (FSM) implementation of watchdog reset initialization and fault detection logics. On power-up the WDFAIL output is asserted, indicating a watchdog failure [19]. A rising edge on the

WDRST bit prepares the watchdog timer for initialization. When the service window opens, a rising edge on the WDSRVC bit de-asserts the WDFAIL output and the window counters start running. However, if the watchdog is serviced incorrectly, the whole initialization process is discarded and the software will have to repeat the entire procedure. The WDFAIL signal gets de-asserted only when the watchdog is properly initialized. While the watchdog is up and running if any of the failure modes described in section III occurs, the WDFAIL output is again asserted [11]. The configuration register is updated with the failed status and the nature of the failure. Assertion of the watchdog fail also triggers a reset counter that runs for redefined amount of time. The duration of the counter can be determined by considering the amount of debug information that needs to be stored. On the expiry of the counter, the WDT asserts its RSTOUT output high. The reset counter will be non-functional during power-up and the RSTOUT output will be set to low at this point. When the watchdog is initialized for the first time, the counter gets automatically enabled.

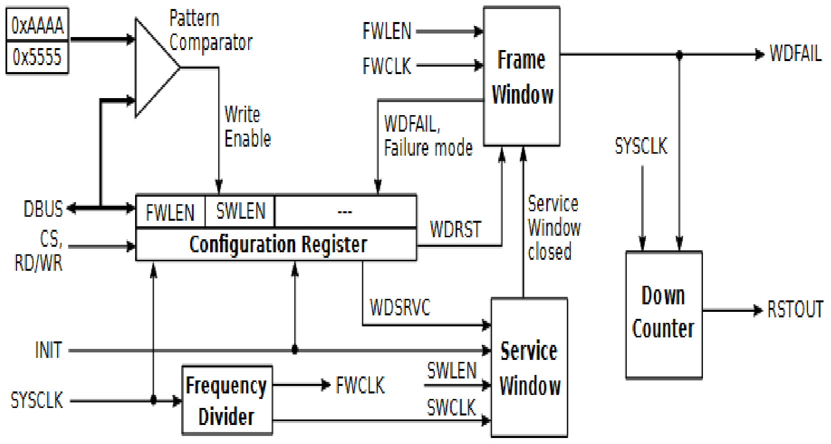


Fig. 6. Functional block diagram of the proposed watchdog timer

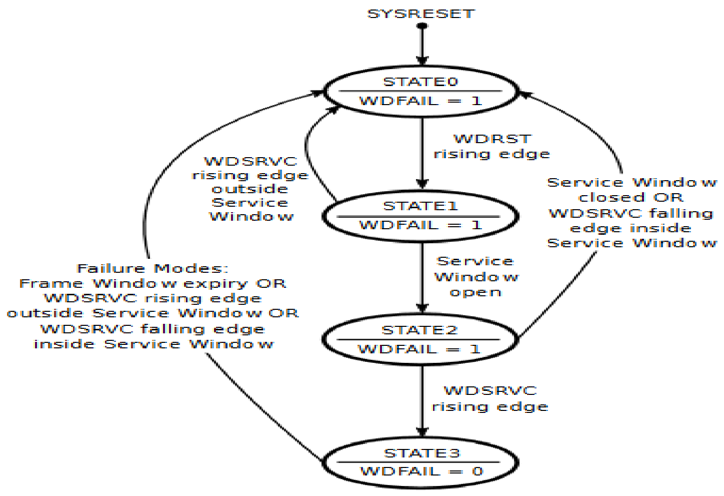
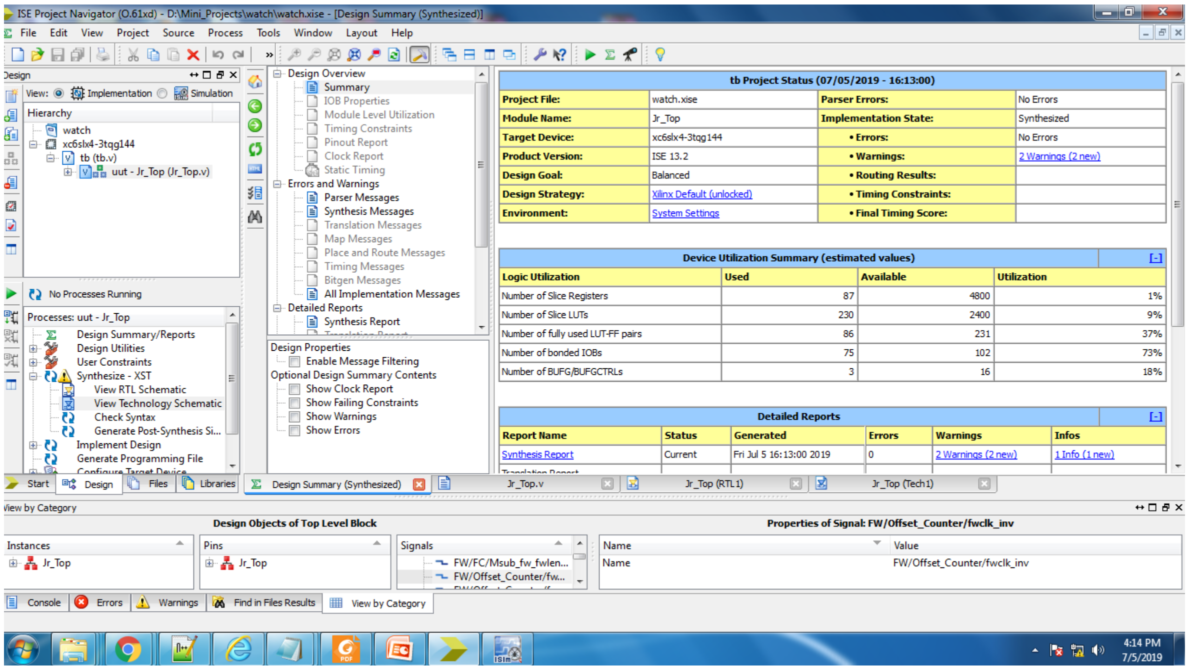

SYSRESET
STATE0
WDFAIL = 1
WDRST rising edge
WDSRVC rising edge outside Service Window
Service Window closed OR WDSRVC falling edge inside Service Window
STATE1
WDFAIL = 1
Service Window open
Failure Modes: Frame Window expiry OR WDSRVC rising edge outside Service Window OR WDSRVC falling edge inside Service Window
STATE2
WDFAIL = 1
WDSRVC rising edge
STATE3
WDFAIL = 0

Fig. 7. Finite state machine design of watchdog fault detection logic

VI. SIMULATION RESULTS

Project Navigator



tb Project Status (07/05/2019 - 16:13:00)			
Project File:	watch.xise	**Parser Errors:**	No Errors
Module Name:	Jr_Top	**Implementation State:**	Synthesized
Target Device:	xc6slx4-3tqg144	**• Errors:**	No Errors
Product Version:	ISE 13.2	**• Warnings:**	2 Warnings (2 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

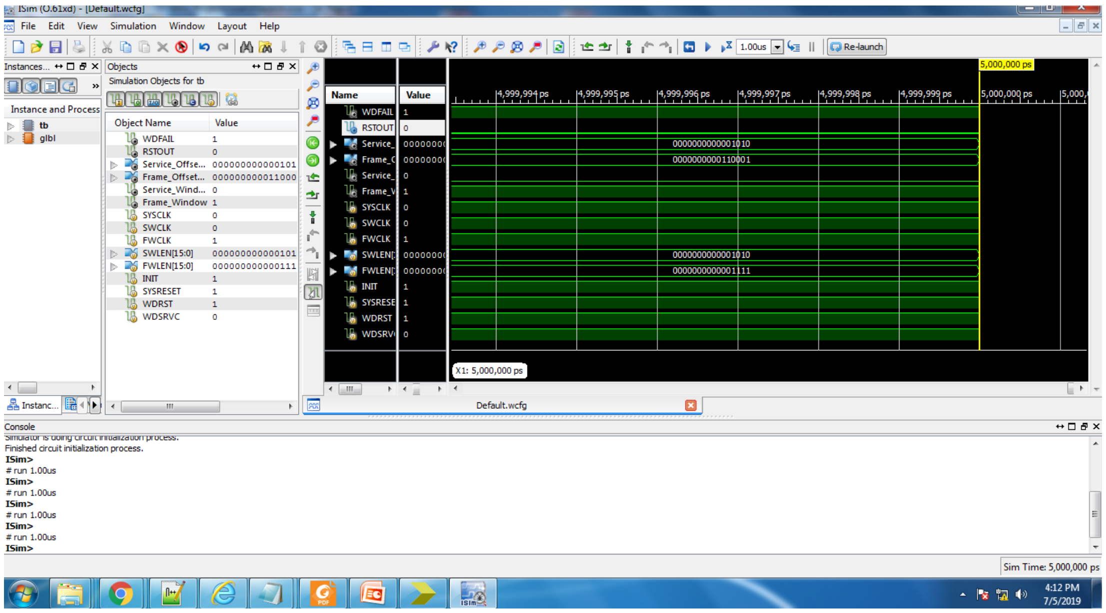
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	87	4800	1%
Number of Slice LUTs	230	2400	9%
Number of fully used LUT-FF pairs	86	231	37%
Number of bonded IOBs	75	102	73%
Number of BUFG/BUFGCTRLs	3	16	18%

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Fri Jul 5 16:13:00 2019	0	2 Warnings (2 new)	1 Info (1 new)

Properties of Signal: FW/Offset_Counter/fwclk_inv

Name	Value
Name	FW/Offset_Counter/fwclk_inv

Simulation Output

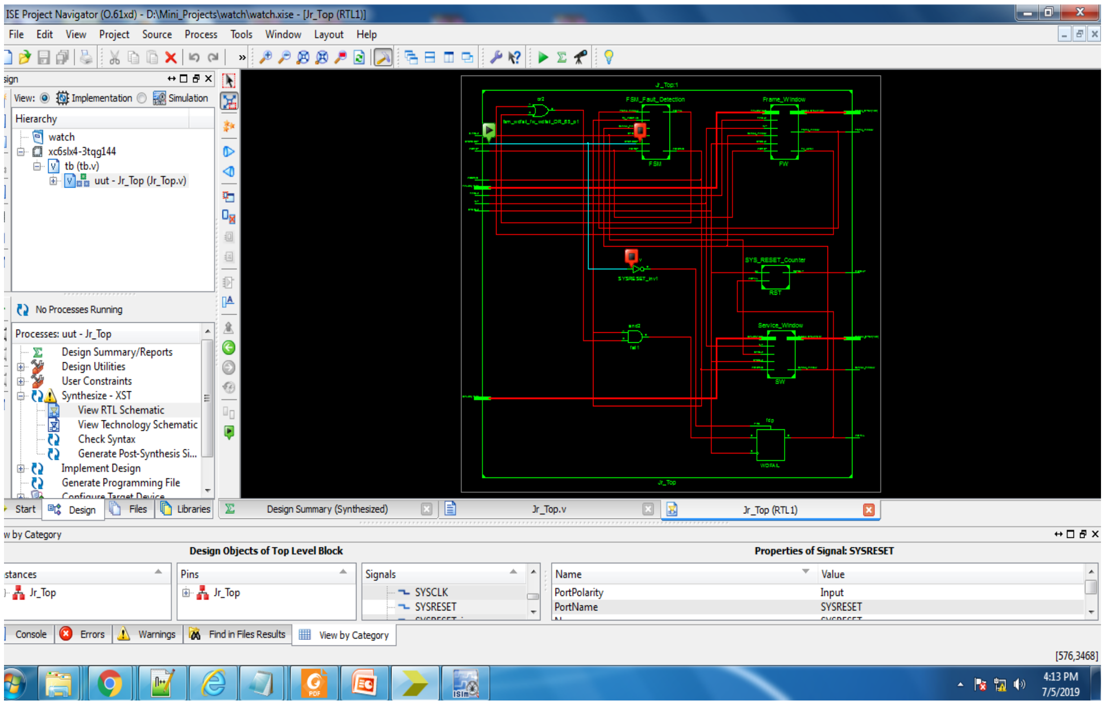


Object Name	Value
WDFAIL	1
RSTOUT	0
Service_Offse...	000000000000101
Frame_Offset...	000000000011000
Service_Wind...	0
Frame_Window	1
SYSCLK	0
SWCLK	0
FWCLK	1
SWLEN[15:0]	000000000000101
FWLEN[15:0]	000000000000111
INIT	1
SYSRESET	1
WDRST	1
WDSRVC	0

```
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# run 1.00us
ISim>
# run 1.00us
ISim>
# run 1.00us
ISim>
# run 1.00us
ISim>
```

Sim Time: 5,000,000 ps

Design Objects of Top Level Block Output


ISE Project Navigator (O.61xd) - D:\Mini_Projects\watch\watch.xise - [Jr_Top (RTL1)]
File Edit View Project Source Process Tools Window Layout Help
View: Implementation Simulation
Hierarchy
watch
xc6slx4-3tqg144
tb (tb.v)
uut - Jr_Top (Jr_Top.v)
No Processes Running
Processes: uut - Jr_Top
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Si...
Implement Design
Generate Programming File
Configure Target Device
Start Design Files Libraries
Design Summary (Synthesized)
Jr_Top.v
Jr_Top (RTL1)
Console Errors Warnings Find in Files Results View by Category
[576,3468]
4:13 PM
7/5/2019

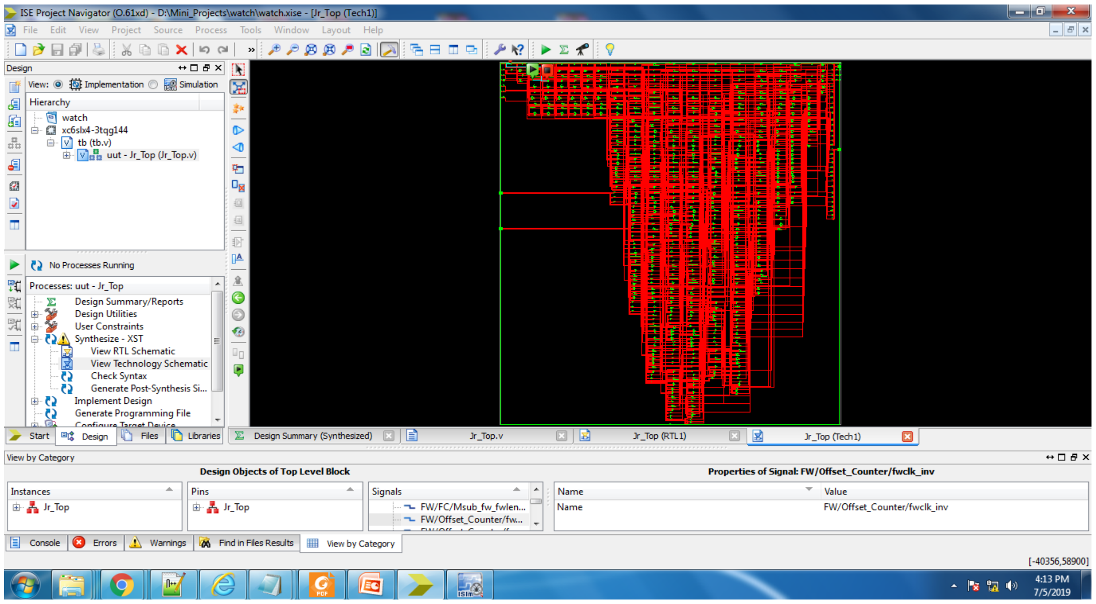
View by Category

Design Objects of Top Level Block

Instances	Pins	Signals
Jr_Top	Jr_Top	SYSCLK
		SYSRESET

Properties of Signal: SYSRESET

Name	Value
PortPolarity	Input
PortName	SYSRESET


ISE Project Navigator (O.61xd) - D:\Mini_Projects\watch\watch.xise - [Jr_Top (Tech1)]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
watch
xc6slx4-3tqg144
tb (tb.v)
uut - Jr_Top (Jr_Top.v)
No Processes Running
Processes: uut - Jr_Top
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Si...
Implement Design
Generate Programming File
Configure Target Device
Start Design Files Libraries
Design Summary (Synthesized)
Jr_Top.v
Jr_Top (RTL1)
Jr_Top (Tech1)
Console Errors Warnings Find in Files Results View by Category
[-40356,58900]
4:13 PM
7/5/2019

View by Category

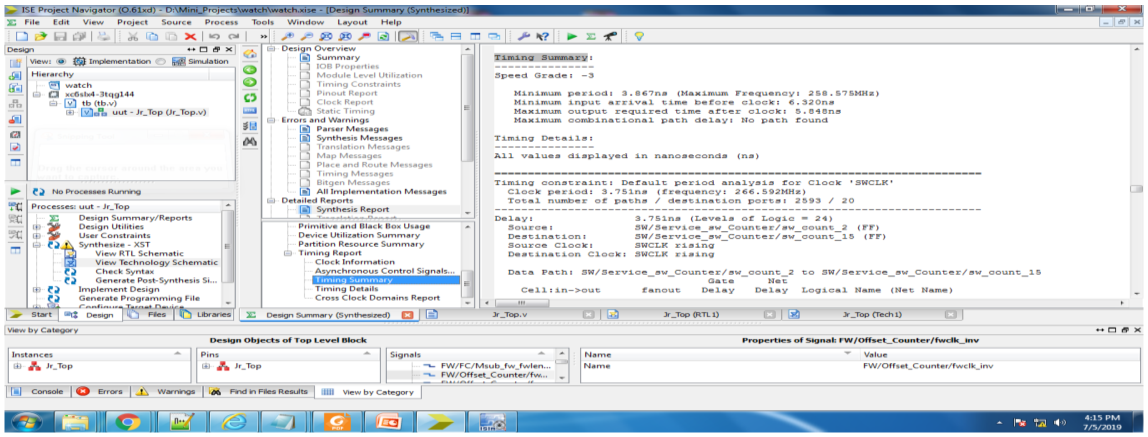
Design Objects of Top Level Block

Instances	Pins	Signals
Jr_Top	Jr_Top	FW/FC/Msub_fw_fwlen...
		FW/Offset_Counter/fw...

Properties of Signal: FW/Offset_Counter/fwclk_inv

Name	Value
Name	FW/Offset_Counter/fwclk_inv

Timing Summary



```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 3.867ns (Maximum Frequency: 258.575MHz)
   Minimum input arrival time before clock: 6.320ns
   Maximum output required time after clock: 5.848ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'SWCLK'
  Clock period: 3.751ns (frequency: 266.592MHz)
  Total number of paths / destination ports: 2593 / 20
-------------------------------------------------------------------------
Delay:               3.751ns (Levels of Logic = 24)
  Source:            SW/Service_sw_Counter/sw_count_2 (FF)
  Destination:       SW/Service_sw_Counter/sw_count_15 (FF)
  Source Clock:      SWCLK rising
  Destination Clock: SWCLK rising

  Data Path: SW/Service_sw_Counter/sw_count_2 to SW/Service_sw_Counter/sw_count_15
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
```

VII. CONCLUSION

This paper presented in detail the architecture and design of an improved windowed watchdog timer and its implementation in FPGA. The watchdog timer runs completely independent of the processor and permits adjusting the timer parameters according to the application. Several fault detection techniques are built into the watchdog for the early detection of erratic software modes. It has the capability to identify the failure type and log it, which can become valuable while debugging. Upon detecting a failure, the watchdog timer also allows the software sufficient time for saving the debug information, before initiating a reset.

VIII.REFERENCES

[1].S. N. Chau, L. Alkalai, A. T. Tai, and J. B. Burt, “Design of a fault-tolerant COTS-based bus architecture,”IEEE Transactions on Reliability, vol. 48, no. 4, pp. 351–359, Dec. 1999.

[2] A. M. El-Attar and G. Fahmy, “An improved watchdog timer to enhance imaging system reliability in the presence of soft errors,” in Signal Processing and Information Technology, 2007 IEEE International Symposium on. IEEE, Dec. 2007, pp. 1100–1104.

[3] M. Pohronsk ́a and T. Krajˇcoviˇc, “FPGA implementation of multiple hardware watchdog timers for enhancing real-time systems security,” in EUROCON-International Conference on Computer as a Tool (EURO-CON), 2011 IEEE. IEEE, Apr. 2011, pp. 1–4.

[4] H. Guzman-Miranda, L. Sterpone, M. Violante, M. A. Aguirre, and M. Gutierrez-Rizo, “Coping with the obsolescence of safety- or mission-critical embedded systems using fpgas,”IEEE Transactions on Industrial Electronics, vol. 58, no. 3, pp. 814–821, 2011.

[5] H. Amer and A. Sobeih, “Increasing the reliability of the Motorola MC68HC11 in the presence of temporary failures,” in Electro technical Conference, 2002. MELECON 2002. 11th Mediterranean. IEEE, May2002, pp. 231–234.

[6] A. M. El-Attar and G. Fahmy, “A study of fault coverage of standard and windowed watchdog timers,” in Signal Processing and Communications, 2007. ICSPC 2007. IEEE

International Conference on.IEEE, Nov.2007, pp. 325–328.
[7] M. Barr, “Introduction to watchdog timers,”Embedded Systems Design, 2001.
[8] N. Murphy, “Watchdogs timers,”Embedded Systems Programming, p.112, 2000.
[9] F. Afonso, C. A. Silva, A. Tavares, and S. Montenegro, “Application-level fault tolerance in real-time embedded systems,” in Industrial Embedded Systems, 2008. SIES 2008. International Symposium on.IEEE, Jun. 2008, pp. 126–133.
[10] M. Wirthlin, “High-reliability fpga-based systems: Space, high-energy physics, and beyond,”Proceedings of the IEEE, vol. 103, no. 3, pp.379–389, Mar. 2015.
[11] H. Ziade, R. A. Ayoubi, R. Velazcoet al., “A survey on fault injection techniques,”The International Arab Journal of Information Technology,vol. 1, no. 2, pp. 171–186, Jul. 2004.
[12] V. B. Prasad, “Fault tolerant digital systems,”IEEE Potentials, vol. 8,no. 1, pp. 17–21, Feb. 1989.
[13] J. Beningo, “A review of watchdog architectures and their application to Cubesats,” Apr. 2010.
[14] A. Mahmood and E. J. McCluskey, “Concurrent error detection using watchdog processors - a survey,”IEEE Transactions on Computers,vol. 37, no. 2, pp. 160–174, Feb. 1988.
[15] B. Straka, “Implementing a microcontroller watchdog with a field-programmable gate array (FPGA),” Apr. 2013.
[16] J. Ganssle, “Great watchdogs,”V-1.2, The Ganssle Group, updated January 2004, 2004.
[17] E. Schlaepfer, “Comparison of internal and external watchdog timers application note,”Maxim Integrated Products, 2008.
[18] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, “An overview of reconfigurable hardware in embedded systems,”EURASIP Journal on Embedded Systems, vol. 2006, no. 1, pp. 13–13, Jan. 2006.
[19] G. C. Giaconia, A. Di Stefano, and G. Capponi, “FPGA-based concur-rent watchdog for real-time control systems,” Electronics Letters, vol. 39, no. 10, pp. 769–770, Jun. 2003.