Windowed Watchdog Timer For Real Time Application

Ashwini. V.B.N.[1], Divya. S[2], Dr. K.K. Senthil Kumar[3]
[1, 2, 3] Prince Shri Venkateshwara Padmavathy Engineering College, Ponmar

Abstract- *Embedded system that are employed in safety critical application requires 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 time out periods and limited features in terms of their functionality. This project describes the architecture and design of an improved configurable watchdog timer that can be employed in real time applications. The functionality and operations are general and it is mainly used to monitor any process based on real time systems. This is allows the design to be easily adaptable to different applications while reducing the overall system cost.*

I. INTRODUCTION

Very Large Scale Integration is used in real time applications which increases the performance of any system. It is the process of creating an integrated circuits by combining hundreds of transistors or device into a single chip.

The Simple logic gates might be considered as SSI devices and multiplexers and parity encoders as MSI, the world of VLSI is much more diverse. Generally, the entire design procedure follows a step by step approach in which each design step is followed by simulation before actually being put onto the hardware or moving on to the next step. The major design steps are different levels of abstractions of the device as a whole:

It is more of a high level representation of the system. The major parameters considered at this level are performance, functionality, physical dimensions, fabrication technology and design techniques. It has to be a tradeoff between market requirements, the available technology and the economical viability of the design. The end specifications include the size, speed, power and functionality of the VLSI system Basic specifications like Floating point units, which system to use, like RISC (Reduced Instruction Set Computer) or CISC (Complex Instruction Set Computer), number of ALU's cache size etc. Defines the major functional units of the system and hence facilitates the identification of interconnect requirements between units, the physical and electrical specifications of each unit. A sort of block diagram is decided upon with the number of inputs, outputs and timing decided upon without any details of the internal structure.

Boolean expressions, control flow, word width, register allocation etc. are developed and the outcome is called as Register Transfer Level (RTL) . This part is implemented either with Hardware Descriptive Languages like VHDL (VHSIC Hardware Description Language) and/or Verilog. Gate minimization techniques are employed to find the simplest, or rather the smallest most effective implementation of the logic.

Choosing the best layout for each block from partitioning step and the overall chip, considering the interconnect area between the blocks, the exact positioning on the chip in order to minimize the area arrangement while meeting the performance constraints through iterative approach are the major design steps taken care of in this step. The quality of placement becomes evident only after this step is completed. Routing involves the completion of the interconnections between modules. This is completed in two steps. First connections are completed between blocks without taking into consideration the exact geometric details of each wire and pin. Then, a detailed routing step completes point to point connections between pins on the blocks. The quality of placement becomes evident only after this step is completed. Routing involves the completion of the interconnections between modules. This is completed in two steps. First connections are completed between blocks without taking into consideration the exact geometric details of each wire and pin. Then, a detailed routing step completes point to point connections between pins on the blocks. The chips are put together on a Printed Circuit Board or a Multi Chip Module to obtain the final finished product.

Initially, design can be done with three different methodologies which provide different levels of freedom of customization to the programmers. The design methods, in increasing order of customization support, which also means increased amount of overhead on the part of the programmer, are FPGA and PLDs, Standard Cell (Semi Custom) and Full Custom Design.

While FPGAs have inbuilt libraries and a board already built with interconnections and blocks already in place; Semi Custom design can allow the placement of blocks in user defined custom fashion with some independence, while most libraries are still available for program development. Full Custom Design adopts a start from scratch approach where the programmer is required to write the whole set of libraries and also has full control over the block development, placement and routing. This also is the same sequence from entry level designing to professional designing.

Xilinx ISE (Integrated Synthesis Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

Xilinx ISE is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors. The Xilinx ISE is primarily used for circuit synthesis and design, while ISIM or the ModelSim logic simulator is used for system-level testing. Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and Chip scope.

Xilinx released the last version of ISE in October 2013 (version 14.7), and states that "ISE has moved into the sustaining phase of its product life cycle, and there are no more planned ISE releases." Hardware description languages such as Verilog are similar to software programming languages because they include ways of describing the propagation time and signal strengths (sensitivity). There are two types of assignment operators; a blocking assignment (=), and a non-blocking (<=) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables.

At the time of Verilog introduction (1984), Verilog represented a tremendous productivity improvement for circuit designers who were already using graphical schematic capture software and specially written software programs to document and simulate electronic circuits. Since these concepts are part of Verilog language semantics, designers could quickly write descriptions of large circuits in a relatively compact and concise form.

The designers of Verilog wanted a language with syntax similar to the C programming language, which was already widely used in engineering software development. Like C programming, Verilog is case-sensitive and has a basic preprocessor (though less sophisticated than that of ANSI C/C++). Its control flow keyword (if/else, for, while, case, etc.) are equivalent, and its operator precedence is compatible with C.

Syntactic differences include: required bit-widths for variable declarations, demarcation of procedural blocks (Virology uses begin/end instead of curly braces {}), and many other minor differences. Virology requires that variables be given a definite size. In C these sizes are assumed from the 'type' of the variable (for instance an integer type may be 8 bits).

A Virology design consists of a hierarchy of modules. Modules encapsulate and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, register, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies). Sequential statements are placed inside a begin/end block and executed in sequential order within the block.

The blocks themselves are executed concurrently, making Verilog a dataflow language. Verilog concept of 'wire' consists of both signal values (4-state: "1, 0, floating, undefined") and signal strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths. A subset of statements in the Verilog language is synthesizable.

Verilog modules that conform to a synthesizable coding style, known as RTL (register-transfer level), can be physically realized by synthesis software. Synthesis software algorithmically transforms the (abstract) Verilog source into a net list, a logically equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific FPGA or VLSI technology. Manipulations to the net list ultimately lead to a circuit fabrication blueprint (such as a photo mask set for an ASIC or a bit stream file for an FPGA).

II. RELATED WORK

A watchdog timer (sometimes called a computer operating properly or COP timer, or simply a watchdog) is an electronic timer that is used to detect and recover from computer malfunctions. During normal operation, the

computer regularly resets the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to reset the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective action or actions. The corrective actions typically include placing the computer system in a safe state and restoring normal system operation. Watchdog timers are commonly found in embedded systems and other computer-controlled equipment where humans cannot easily access the equipment or would be unable to react to faults in a timely manner. In such systems, the computer cannot depend on a human to invoke a reboot if it hangs; it must be self-reliant. For example, remote embedded systems such as space probes are not physically accessible to human operators; these could become permanently disabled if they were unable to autonomously recover from faults. A watchdog timer is usually employed in cases like these. Watchdog timers may also be used when running un-trusted code in a sandbox, to limit the CPU time available to the code and thus prevent some types of denial-of-service attacks.

Real-time computer systems are defined as systems that are in any conditions able to guarantee their response time. Such systems are used mostly in various embedded devices to guarantee their usability, for example to ensure smooth video playback, and in various industrial control applications. Their utilization in industrial application is often connected with the mission-critical tasks that need to be accomplished in time to prevent system malfunction or damage.

The real-time computer system is usually implemented on specific hardware aimed for such purposes. It can run a simple application that takes care of the whole controlled system or an operating system with several applications of which each one has its own task and response deadline defined. One of the methods to recover such systems from error states and ensure their further functionality and responsiveness is utilization of watchdog timers. Watchdog timer is a hardware device usually realized by a counter with match register and specific system connections. The timer can be set to measure any amount of time within some reasonable boundaries. This initialization value is, with a small reserve, equal to the maximum execution time defined for the process or the whole system. When the timer is left to overflow, it automatically signalizes that an error has occurred in the program flow or that the program didn't meet the response time deadline. Thus, when the timer expires, it takes certain action to recover the system from such error state. The action could be as simple as restarting the system or as complex as running a system diagnostic test. A watchdog timer is said to have fired if it has not been reset within a programmable period. It is the role of the particular watched task or process to configure the watchdog timer and to periodically reset the timer before it expires.

The commonly used systems are usually equipped with one hardware watchdog timer that is capable of resetting the whole system and few hardware counters which can provide the system with timing information. In real-time operating systems, when there are a number of independent processes to be secured, we usually utilize one of the hardware timers to provide the time base for creating virtual watchdog timers. These virtual timers are then assigned to each system process that has to be monitored. This way, each hardware timer is utilized for the implementation of orders of tens to hundreds of software watchdog timers for the concurrent processes. As the virtual watchdog timers share one driver for hardware timer, an individual fault in a process or the driver can possibly manipulate the hardware timer in such way that it is no longer usable and thus blocks the control for all the processes assigned to this hardware timer.

The act of restarting a watchdog timer is commonly referred to as "kicking the dog" or other similar term, this is typically done by writing to a watchdog control port. Alternatively, in microcontrollers that have an integrated watchdog timer. The watchdog is sometimes kicked by executing a special machine language instruction. An example of this is the CLRWDT (clear watchdog timer) instruction found in the instruction set of some PIC microcontrollers.

In computers that are running operating systems, watchdog resets are usually invoked through a device driver. For example, in the Linux operating system, a user space program will kick the watchdog by interacting with the watchdog device driver, typically by writing a zero character to /dev/watchdog. The device driver, which serves to abstract the watchdog hardware from user space programs, is also used to configure the time-out period and start and stop the timer. Watchdog timers come in many configurations, and many allow their configurations to be altered. Microcontrollers often include an integrated, on-chip watchdog. In other computers the watchdog may reside in a nearby chip that connects directly to the CPU, or it may be located on an external expansion card in the computer's chassis. The watchdog and CPU may share a common clock signal, as shown in the figure 1.1 below, or they may have independent clock signals.

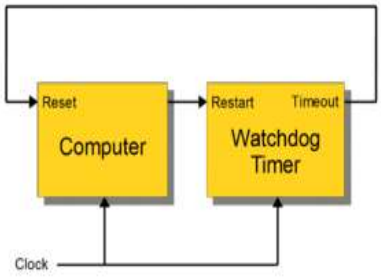

Figure 1.1

Two or more timers are sometimes cascaded to form a multistage watchdog timer, where each timer is referred to as a timer stage, or simply a stage. Figure 1.2 below shows a three-stage watchdog timer. In a multistage watchdog, only the first stage is kicked by the processor. Upon first stage timeout, a corrective action is initiated and the next stage in the cascade is started. As each subsequent stage times out, it triggers a corrective action and starts the next stage.

Upon final stage timeout, a corrective action is initiated, but no other stage is started because the end of the cascade has been reached. Typically, single-stage watchdog timers are used to simply restart the computer, whereas multistage watchdog timers will sequentially trigger a series of corrective actions, with the final stage triggering a computer restart.

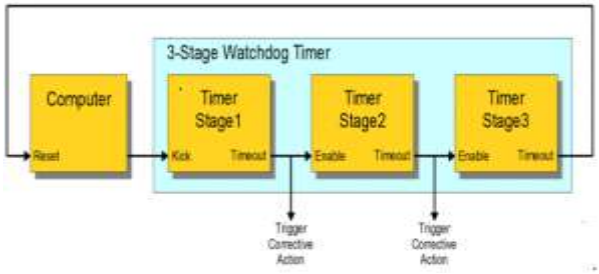

Figure 1.2

Watchdog timers may have either fixed or programmable time intervals. Some watchdog timers allow the time interval to be programmed by selecting from among a few selectable, discrete values. In others, the interval can be programmed to arbitrary values. Typically, watchdog time intervals range from ten milliseconds to a minute or more. In a multistage watchdog, each timer may have its own, unique time interval. A watchdog timer may initiate any of several types of corrective action, including processor reset, non-maskable interrupt, maskable interrupt, power cycling, fail-safe state activation, or combinations of these. Depending on its architecture, the type of corrective action or actions that a watchdog can trigger may be fixed or programmable. Some computers require a pulsed signal to invoke a processor reset. In such cases, the watchdog typically triggers a processor reset by activating an internal or external pulse generator, which in turn creates the required reset pulses.

In embedded systems and control systems, watchdog timers are often used to activate fail-safe circuitry. When activated, the fail-safe circuitry forces all control outputs to safe states (e.g., turns off motors, heaters, and high-voltages) to prevent injuries and equipment damage while the fault persists. In a two-stage watchdog, the first timer is often used to activate fail-safe outputs and start the second timer stage; the second stage will reset the computer if the fault cannot be corrected before the timer elapses.

Watchdog timers are sometimes used to trigger the recording of system state information—which may be useful during fault recovery—or debug information (which may be useful for determining the cause of the fault) onto a persistent medium. In such cases, a second timer—which is started when the first timer elapses—is typically used to reset the computer later, after allowing sufficient time for data recording to complete. This allows time for the information to be saved, but ensures that the computer will be reset even if the recording process fails.

For example, in a two-stage watchdog timer, during normal operation the computer regularly kicks Stage1 to prevent a timeout. If the computer fails to kick Stage1 (e.g., due to a hardware fault or programming error), Stage1 will eventually timeout. This event will start the Stage2 timer and, simultaneously, notify the computer (by means of a non-maskable interrupt) that a reset is imminent. Until Stage2 times out, the computer may attempt to record state information, debug information, or both. The computer will be reset upon Stage2 timeout.

III. METHODS

In the 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 peripheral. The watchdog 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. 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, thus increasing the error recognition coverage.

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 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 the read and write to the configuration register. The ABUS and DBUS signals in the figure indicate address bus and data bus, respectively. The length of the two windows can be programmed by the software after power-up by writing to the bit fields, SWLEN and FWLEN, in the configuration register.

Once the window periods are configured after power-up, modifying the values is disabled by design. If needed, the software will have to go through a stringent unlock procedure in order to be able to once again write to the configuration register.

This prevents any accidental modification of the watchdog window parameters by a runaway code. The INIT input to the watchdog timer initializes the service window. A high-to-low transition on this input will start the service window, provided the fail flag (WDFAIL) is not active. The processor is required to service the watchdog within the service window, in order to prevent a timeout.

The watchdog timer is serviced using the watchdog service (WDSRVC) field in the configuration register. A rising edge on this bit inside the service window will immediately close the window and start the frame window. The frame window defines how periodically the watchdog should be serviced. Typically, the duration of this window is kept slightly more than the main loop of the embedded control system and the watchdog is serviced once in every cycle.

Initialization of watchdog timer

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. 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.

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.

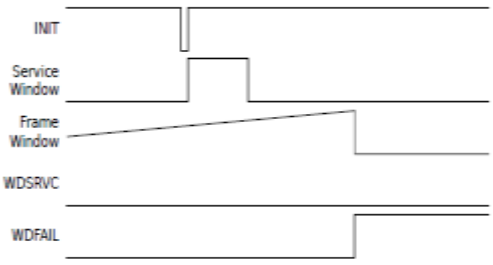

Figure 1.3

A watchdog fail will occur when the software services the watchdog outside the service window. It can be seen that the invalid service operation instantly terminates the frame window and asserts the WDFAIL signal.

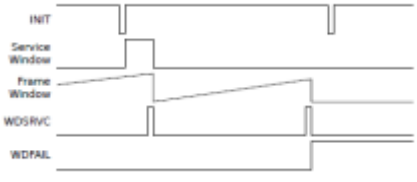

Figure 1.4

A favourable 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. This becomes equivalent to servicing the watchdog outside the service window and leads to a watchdog failure.

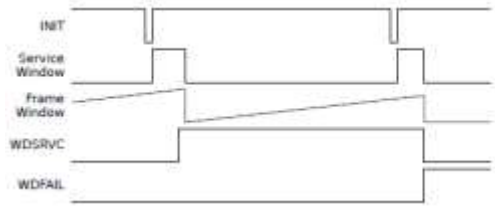

Figure 1.4

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 fail 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. All of these fault detection mechanisms ensure that a software running haywire will not go undetected by the proposed watchdog timer.

Implementation of Watchdog timer

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.

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 10 μs period to modify the length configuration fields. If these timings are not strictly met, writes to these bits will remain disabled.

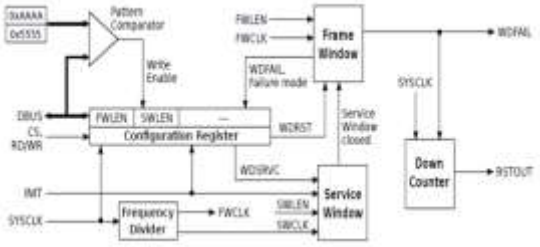

Figure 1.5

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 window has an offset up/down counter that are clocked by the SYSCLK, and a main counter that runs at SWCLK.

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 frame window counters reset when a watchdog service operation occurs within the next service window duration, before the frame window expires.

Reset initialization and fault detection

On power-up the WDFAIL output is asserted, indicating a watchdog failure. 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. Fault detection and initialization process is clearly explained.

The WDFAIL signal gets de-asserted only when the watchdog is properly initialized. Assertion of the watchdog fail also triggers a reset counter that runs for a predefined 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 nonfunctional 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.

IV. CONCLUSION

An efficient windowed watchdog timer is proposed. This windowed watchdog timer runs completely independent of the processor and permit adjusting the timer parameters according to the applications. It has the capability to identify the failure type which can become valuable while debugging. The same design can also be implemented in different

processors and applications with minor modifications which is one of the advantages of the proposed system.

REFERENCES

[1] Blem, E. Compton, K. Gracia, P. Schulte, M. and Fu, W. (2006) 'An overview of reconfigurable hardware in embedded system', EURASIP journal on embedded system.

[2] Capponi, G. Di Stefano, A. Giaconia, G.C. (2003) 'FPGA-bassed concurrent Watchdog for real-time control systems' Electronics Letters, Vol.39,no. 10,pp.769-770.

[3] Craig Lee, Ian Foster, Lazzewski, G.V. Kesselman, C. (1999) 'A Fault Detection Service for Wide Area Distributed Computations' Cluster Computing,v.2 n.2,p.117-128.

[4] Ganssle, G Jack. (2004) 'Great Watchdog', V-1.2, Gaanssel Group, Update.

[5] Hadad, E. Friedman, R. (2002) 'FTS: A High Performance CORBA Fault Tolerance Service' Proc.IEEE Workshop Object Oriented Real-Time Dependable Systems.

[6] Harrick, M.Vin. Lorenzo Alvisi, Sriram Rao, (1999) 'Egida: An Extensible ToolKit For Low over head Faut Tolerance, Fault Tolerant Computing', Digest of Papers. 29th Annual International Symposium,P.45-55.

[7] Ian Foster and Iamnitchi, A. (2000) 'A Problem Specific Fault-Tolerance Mechanism for Asynchorous, Disturbuted System', IEEE, p.4-13

[8] Jie Xu and Paul Toenend, (2003) 'Replication-based Fault-Tolerance in a Grid Environment',citeceer.

[9] Konchan, R. Kopylchak, A. and Korkishko, T. (2002) 'Improved Watchdog timer for Control the IBM Pc based Autonoums Computer systems, in Modern Problems of Radio Engineering, Telecommunications and computer science, pp. 181-182.

[10] Lennon, W.K. and Passino, K.M. (2009) 'Intelligent Control for Brake systems',IEEE Transcations on Control Systems Technology , vol.7,pp.188-202

[11] Mahmood, A. and McCluskey, E.J. (1998) 'Concurrent Error Detection Using Watchdog Processors - a survey', IEEE Transcations on Copmuters, vol. 37, no. 2, pp.160-174.

[12] Pascal Felber. Proya Narasimhan, Member (2004) 'Experiences, Strategies, and challenges in Building Fault-Tolerant CORBA Systems', IEEE Transcations on Computers, vol.53,no.5.

[13] Pohronska, M. and Krajcovic, T. (2010) 'Fault-Tolerant embedded Systems with multiple FPGA implemented Watchdogs', proceedings of the International Conference CYBERNETICS AND INFORMATICS, D.R Stefan kozak Alena Kozakova,ed., Vydavatelstvo STU,p.37.

[14] Pohronska, M. and Krajcovic, T. (2010) 'Embedded Systems with increased reliability using the Multiple Watchdog Timers Approach', International conference of Applied Electronics, proceedings, J. Pinker ed., University of West Bohemia in pilsen, pp.273-276.

[15] Stracka, B. (2013) 'Implementing a microcontroller Watchdog with a field programmable gate array (FPGA)'.