FAULT-TOLERANT EMBEDDED SYSTEMS WITH MULTIPLE FPGA IMPLEMENTED WATCHDOGS

Mária Pohronská and Tibor Krajčovič

Slovak University of Technology, Faculty of Informatics and Information Technologies
Ilkovičova 3, 812 19 Bratislava, Slovak Republic
Tel.: +421 2 60291111 Fax: +421 2 654 20 587
e-mail: {pohronska, tkraj}@fiit.stuba.sk

Abstract: The paper deals with the problem of fault-tolerant embedded systems. To ensure the system's fault tolerance, we use the multiple watchdog system that assigns a dedicated hardware watchdog timer to each of the concurrent processes. For sake of simplicity, effectiveness and adaptability, the watchdogs are implemented in the programmable hardware devices. This approach has several advantages and provides a method for implementing the reliable fault-tolerant embedded systems.

Keywords: embedded systems, concurrent processes, fault-tolerant systems, watchdog, reconfigurable circuits, FPGA

1 INTRODUCTION

Embedded computer systems are defined as systems that are an inseparable part of the devices, which they control. Generally, they work in real time and they use special input/output devices for an external communication. Another typical feature of these systems is that they are often designed on a microcomputer basis, with their software located in ROM. Another significant characteristic of some embedded systems applications is the requirement for increased level of reliability, especially of the fault tolerance (Berger, 2002). In general, an embedded system realizes a complex control algorithm in real time. The particular requests are serviced by concurrent processes. To ensure the tolerance of the faults in the program flow of a concurrent process, we often use a watchdog timer. A watchdog timer is a hardware device that takes certain actions when a timer expires. 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 (Ganssle, 2008).

For such application, we necessarily need to know the time needed for the accomplishment of the process. When starting new process, the watchdog timer that has been assigned to the process is initialized. The initialization value is, with a small reserve, equal to the maximum execution time of the process. If a fault occurs in the process and changes the program flow in such way that it lets the watchdog timer overflow, it automatically signalizes that an error has occurred in the program flow or that the program didn't meet the response time deadline.

Primary disadvantage of this approach is that the system is by standard equipped only with one or several few hardware timers. Each hardware timer is then used for the implementation of orders of tens to hundreds of software watchdog timers for the concurrent processes. An individual fault in a process 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.

This paper deals with the option of implementing multiple dedicated hardware watchdog timers, utilizing the programmable hardware. This solution has several advantages - particularly it contributes to the system's reliability improvement.

This document is divided into six sections. The next section deals with the related work in this research area. The third section briefly introduces the proposed solutions, summarizes its advantages and drawbacks and proposes various approaches to its implementation. The fourth section describes the developed architecture of a multiple watchdog timer circuit with serial communication interface. The results of the experimental synthesis are presented in the fifth section. The final section concludes the paper and discusses the future work.

2 RELATED WORK

The usual watchdog timer implementations in embedded and other self-sustaining systems utilize the hardware watchdog timers which are embedded in the microprocessor or are part of the microprocessor's chipset. Several commercial implementations of hardware watchdog expansion cards or devices exist, however, the discovered implementations are of very simple nature, outdated and they mostly offer only one watchdog timer per device. (Kochan, 2002) implemented and described the watchdog timer for personal and embedded computers that utilizes the keyboard communication port. The device generates the special scan-codes periodically and awaits corresponding answer from the system (scan codes for toggling the LED indicators). When it doesn't receive the answer, it resets the whole system. The work (Giaconia, 2003) deals with the FPGA implementation of customized embedded system's watchdog processor. In this case, the circuit does not provide timer for the several processes but has more complex fully customized functionality, it provides a reasonableness check on some variables and a basic program flow check.

3 FPGA IMPLENTATION OF MULTIPLE WATCHDOGS

Nowadays, the programmable hardware devices give to embedded designers much more flexibility and customizability of their products (Becker, 2003). These devices are used widely in the embedded systems and in their development process. More details on such utilization of the programmable devices can be found in (Pohronská, 2009). For our problem, the programmable hardware provides us with the possibility of implementing a hardware watchdog timer for each of the concurrent processes in the system. The creation and assignment of the own hardware watchdog timer for each of the system's processes eliminates the possibility of disabling the watchdog timer for another process.

Other advantages of such implementation are summarized below. Firstly, we can theoretically implement as many timers as we need for the system. The only limitation lies in the size of the programmable device. For larger architectures and systems, we can use multiple FPGA devices. Second advantage lies in the enhanced security of the system – when a fault occurs in the process, it can only modify its own timer or, in the worst case, the timer of another process. At any rate, it is far better than the alternative of disabling the common timer of many processes. Furthermore, the service routine for the single dedicated hardware timer is much more simple that the service routine for the timer which is shared by multiple processes. Thus, the system gains on speed and software simplicity. Another advantage of using programmable hardware is the possibility of the custom design that often leads to resource savings. Particularly, we can configure each timer with the precise bit width needed for the computation. Most of the today's FPGA devices also offer the ability of run-time reconfiguration. The part of the circuit can be reconfigured while the rest of the circuit is working continuously. Thus the process can have its timer provided and assigned anytime it is initialized. This gives the system great flexibility comparable with the standard watchdog implementations.

The main drawback of our proposed approach is the need for the usage of additional hardware element that brings additional demands for space, power consumption and cost of the system. Another disadvantage comparing to the commonly used systems is the time needed for the reconfiguration of the programmable hardware when initializing the timer. This process can take much longer than the software initialization of the shared watchdog timer.

For the effective utilization of the proposed solution we have to take particular care of the architecture of the embedded system. We decided to consider only the watchdog timers that generate interrupt request, so the system could serve as universal additional component for various embedded systems. There are several possibilities of realizing the connection between the programmable hardware and the control processor. In each case, we need the FPGA to generate an external interrupt. Then, we need to provide the processor with the particular details of the occurred situation. The first variant is the connection of the FPGA as the coprocessor. This architecture provides the fastest communication facilities but it also is quite expensive solution that is not always feasible. The second variant is the utilization of the data bus for the communication. In such case, we would also have to implement hardware connections and software routines needed for the bus handshake functions. The third variant is the communication through the shared memory. This unloads the processor of the task of serving the FPGA and gives the serving process more flexibility. The last proposed connection architecture is the connection via standardized serial communication interface, for example USB, I2C or SPI. This solution is the cheapest and most simple one but it has drawbacks of the need of first-hand service and slow communication speed that can lead to unwanted system's overload.

In our experimental work, we would like to implement and compare all of these proposed alternatives. Currently, we deal with the implementation of the watchdog timer circuit with serial communication interface.

4 THE ARCHITECTURE OF THE EXPERIMENTAL FPGA WATCHDOG SYSTEM

As the first step, we decided to implement the watchdog circuit with the serial communication interface, which is the simplest from the above mentioned alternatives. This architecture is sufficient for the initial proving of the proposed concept.

The basic element of the circuit is the single watchdog counter. The watchdog counter has been designed as simple as possible, in order to minimize space requirements. It offers four inputs – Reset, Serial Load, Counter Clock and Communication Clock. It provides only one output – the Overflow flag. This basic architecture is shown in the Figure 1.

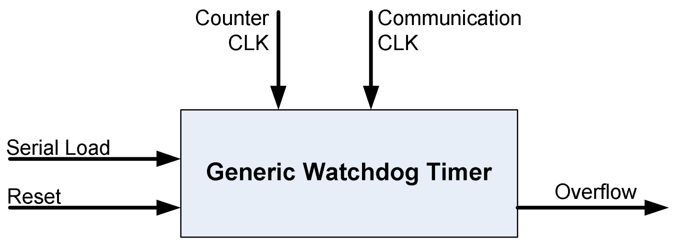

Figure 1: The watchdog timer block scheme

The watchdog timer can be loaded with new value via the serial input “Serial Load” which is synchronized by the “Communication Clock” input. After loading new value, the counter begins to count from the loaded value to zero. This count is synchronized by the “Counter Clock” input. As soon as it reaches the zero value, the counting stops and the “Overflow” output is activated. In order to reset the counter and begin the count from the initial value, the input “Reset” has to be activated. The watchdog counter is implemented as a generic component of arbitrary width. Thus, the width of each counter can be fully customized. The serial load process takes as long as needed for the particular counter. For this reason, the master processor needs to know the width of each counter when initializing it. The following paragraph shows the declaration of the watchdog entity with generic counter width.

```
entity WDT_generic is
   generic(
           WDT_width : natural := 16
           );
   port(
           Clk: in Bit;     -- Clock for the counting process
           ComClk: in Bit;  -- Communication clock (for the
loading process)
           Res: in Bit;
           Sload: in Bit;
           Overflow: buffer Bit
       );
end WDT_generic;
```

For integrating many timers into one circuit, we needed to develop an architecture that would be appropriate and would require minimum overhead costs. We decided to use address bus and address decoders for enabling individual counters. The data lines of „Reset“ and „Serial Load“ are lead to each counter. The usage of counters with serial load unloads us from the need of using the parallel data bus and routing it to each counter. It also enables us to fully exploit the potential of the counters with customized width.

We have designed a generic architecture that allows us to customize the width of the address bus and the thus the maximal number of counters in the circuit. The only restriction is that the address bus width should be divisible by four. The principal scheme of our proposed architecture is shown in Figure 2.

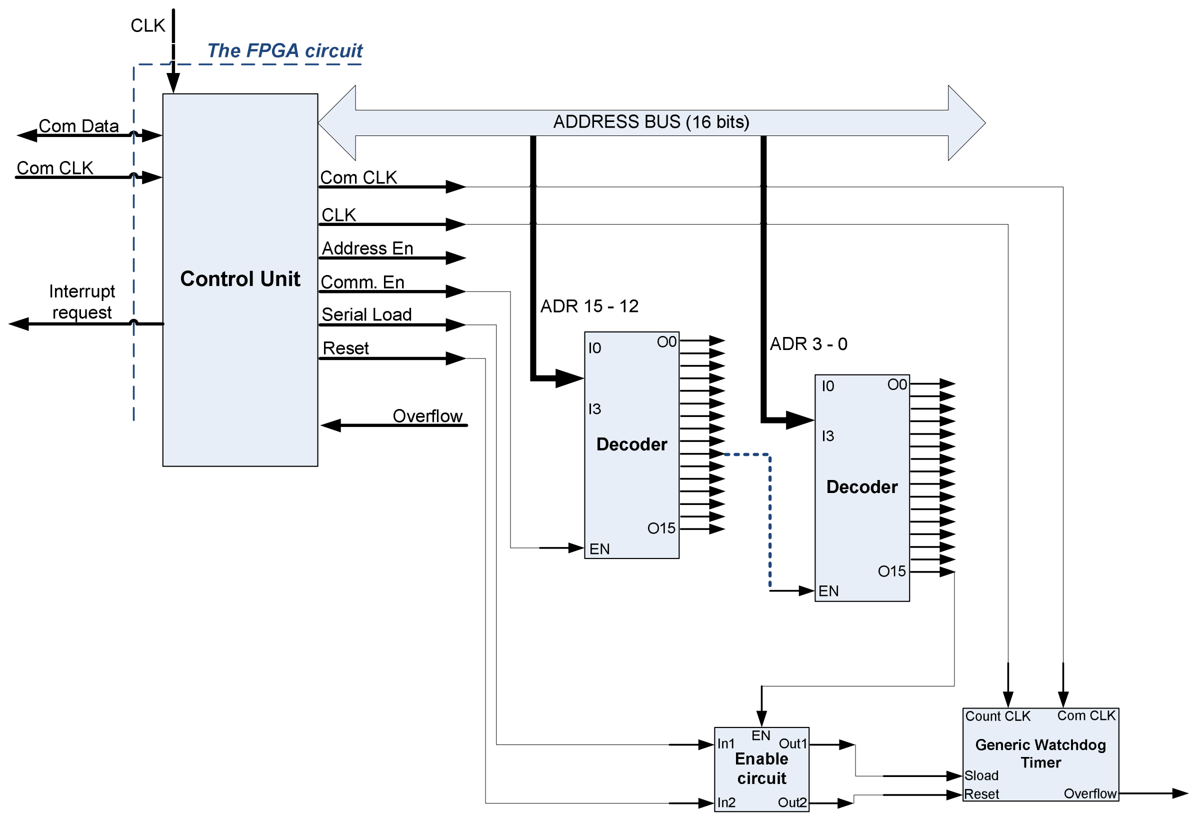

Figure 2: The basic architecture of the counter addressing mechanism

When the overflow occurs in one of the counters the circuit has to generate the interrupt request for the master processor. The processor then asks for the number (address) of the overflown counter. Thus, when the counter overflows, we need to obtain its address. This is realized by encoding the actual counter position and placing the encoded address on the address bus. The encoders are connected to the cascade, similarly as the decoders. We also need to deliberate that the overflow can occur in several timers at the same time. Thus, the circuit needs to decide of the priorities and also has to ensure that the address bus would be written only by one device at the same time. For the fulfilment of the first requirement we decided to use encoders with priority. The exclusive bus access could be however achieved only by providing feedback from the hierarchicaly higher “priority encoders” to the subordinate “priority encoders”. For such function, we needed to develop a non-standard architecture which provides both encoding with priority and feedback decoding function. We also need to propagate the overflow flag through this circuit. The following code shows the declaration of the “Priority encoder-decoder entity”.

```
entity pri_encoder_decoder is
     port (
             enable:in Bit;
             encoder_in :in std_logic_vector (15 downto 0);
             decoder_out:out Bit_vector (15 downto 0);
             binary_out :out Std_logic_vector (3 downto 0);
             overflow_active: out Bit
           );
end pri_encoder_decoder;
```

The scheme of the proposed architecture for the address encoding process is shown in Figure 3.

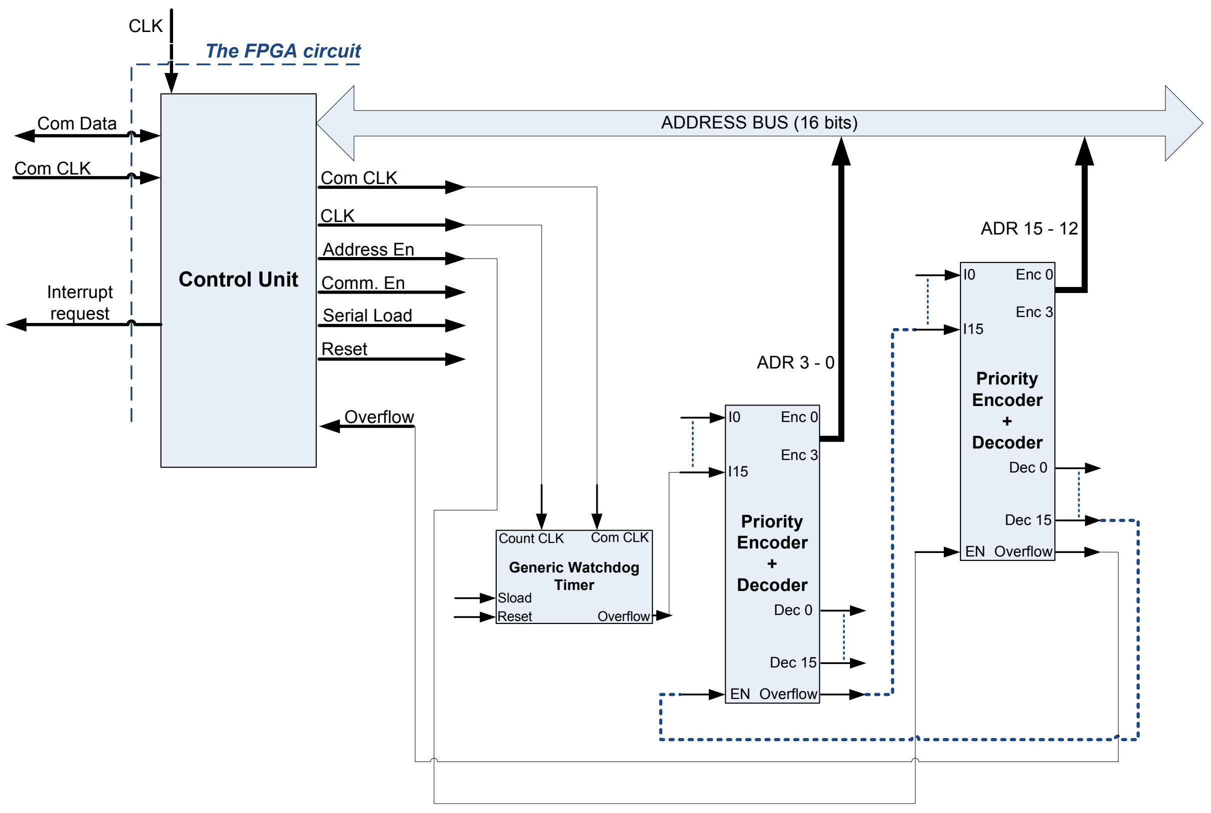

Figure 3: The proposed architecture of the encoding circuitry.

The control unit of the circuit that can be seen on both pictures provides the communication interface for the master processor. It also provides the connectivity and service for the inner counter circuits. The communication interface is realized by two wires – the "Communication Clock" signal that is used for synchronization can be generated only by the master device – the processor. The "Communication data" wire bears the transformed information. The control unit is able to recognize and serve three commands. First command is a reset request. After this command, a 16-bit address follows. As soon as the whole command is received, the control unit resets the requested counter and sends the acknowledgement message to the master processor. The second command is a request of loading an existing counter with a new value. As the control circuit doesn't store information about the width of each counter, the load process should be realized by connecting the counter with the data line directly. So, the connection is made instantly after obtaining the command and a complete address of the circuit. The load process then continues for as long, as needed. The control unit returns to the idle state when there has been no activity of the "Communication Clock" signal for several clock cycles. This ensures recovery from the communication errors and also enables the serial load process without need of explicit knowledge of the loaded counter's width. The following paragraph of code shows the "communication bus watchdog" process.

```
process (CLK)  -- watchdog for the communication bus clock
signal
begin
     if(CLK'Event and CLK='1') then
          if (communicating='1') then
               idle_time := 0;
               idle_id <= '0';
          elsif idle_time >= max_idle_time then
               idle_id <= '1';
          else
```

```
                idle_id <= '0';
            end if;
        idle_time := idle_time + 1;
        end if;
    end process;
```

The third recognized command is the request for the number of the overflown counter. As already mentioned, the control unit recognizes only one address of the overflown counter at the same time. So after receiving the command, it sends the address of the overflown counter that has maximum priority. This counter is then reset and when there are no remaining overflown counters, the interrupt request signal is deactivated.

5 EXPERIMENTAL RESULTS

For the initial experiments, we have implemented the watchdog circuit with the master control unit and a single watchdog counter. Implementation and simulation has been successful and approved the suitability of the proposed architecture. The experiments have also shown us the space required for the realization of the circuit with a single counter, in a FPGA device. Firstly, we have synthesized the same architecture while changing the counter width. As a reference circuit, we have chosen the Xilinx xc3s4000 circuit of the Spartan 3 family. The number of consumed LUTs can be seen in the second column of Table 1. As we can see, counters with various widths can be implemented into the FPGA technology more or less effectively.

Table 1: Implementations of a single watchdog counter architecture of various widths

Timer width (bits)	Number of LUTs	Utilization
8	912	1%
16	1767	3%
32	3425	6%
64	3428	6%
128	3431	6%

In the second experiment we have implemented the architecture with single 64 bit counter in several FPGA circuits, observing the consumed place. The results are summarized in the Table 2. The experiments have been realized using the Xilinx ISE Design Suite 10.1.

Table 2: Implementation of a single watchdog counter architecture in various FPGA devices

Device family	Device	Number of LUTs	Total number of LUTs	Utilization	Consumption (W)
Spartan 3	xc3s200	(4-input LUTs) 3428	3840	89%	0,15
Spartan 3	xc3s4000	(4-input LUTs) 3428	55296	6%	0,265
Virtex 4	xc4vfx15	(4-input LUTs) 3404	12288	27%	0,23
Virtex 5	xc5vfx330	(Slice LUTs) 2852	207360	1%	13,848

6 CONCLUSION

We have proposed a new method for the implementation of fault-tolerant embedded systems, utilizing the programmable hardware for implementation of multiple watchdog timers. The proposed solution promises significant improvements in the system's reliability and stability and can be utilized in any embedded computer system. A recursive generic architecture of a multiple-watchdog timer system with serial communication interface has been developed. Our experiments based on architecture with single timer show that the proposed concept is viable and that the implementation of more complex architectures based on this concept would be feasible.

We plan to continue our work by testing the implemented system in a real embedded application. Afterwards, we plan to focus on the more complex architectures with better communication capabilities.

This work was supported by the Grant No. 1/0822/08 of the Slovak VEGA Grant Agency.

REFERENCES

BECKER, J., H. R. (2003): Configware and morphware going mainstream, *In: Journal of Systems Architecture,* **49**(4-6), 127-142.

BERGER, A. S. (2002): Embedded Systems Design. CMP Books, Lawrence, 237 pages.

GANSSLE, J. G. (2008): The Art of Designing Embedded Systems. Second Edition. Elsevier, 298 pages.

GIACONIA, G.; DI STEFANO, A. & CAPPONI, G. (2003): 'FPGA-based concurrent watchdog for real-time control systems', *In: Electronics Letters,* **39**(10), 769-770.

KOCHAN, R.; KOPYLCHAK, A. & KORKISHKO, T. (2002): Improved watchdog timer for control the IBM PC based autonomous computer systems, *in 'Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2002. Proceedings of the International Conference'*, pp. 181-182.

POHRONSKÁ, M. (2009): Utilization of FPGAs in Real-Time and Embedded Systems, *in M. Bielikova, ed.,'Proceedings in Informatics and Inormation Technologies Student Research Conference',* Vydavateľstvo STU, pp. 300-307.