|  |
| --- |
| AGSTU |
| WATCHDOG TIMER |
| Designing both hard – and software for a watchdog timer with VHDL and low level C programing |
|  |
| **Saif Saadaldin** |
| **2021-02-13** |

|  |
| --- |
| The constructed system is a watchdog timer. The project was done in both hardware description language VHDL and C programing. The timer reset a CPU system if the timer don’t get a “heartbeat” signal from the CPU at least once a second. The system is verified on DE10-Lite FPGA board with Altera MAX10 chip (10M50DAF484C7G). The delivered system contains all source files and documentation. |

[1 INTRODUCTION 2](#_Toc64216004)

[2 REQUIREMENT SPECIFICATION 3](#_Toc64216005)

[3 TIMETABLE 4](#_Toc64216006)

[4 SYSTEM ARCHITECTURE 5](#_Toc64216007)

[4.1 Hardware part (VHDL) 6](#_Toc64216008)

[4.1.1 component\_watchDog 6](#_Toc64216009)

[4.1.2 polling\_interrupt\_cpu\_system 7](#_Toc64216010)

[4.1.3 binary\_to\_bcd 8](#_Toc64216011)

[4.2 Software part (C programming) 9](#_Toc64216012)

[5 VALIDATION 10](#_Toc64216013)

[6 CONCLUSIONS 11](#_Toc64216014)

[7 APPENDIX. 12](#_Toc64216015)

# INTRODUCTION

The aim of this project is designing a watchdog timer in VHDL. The timer will reset the system that is connected to the timer whenever the system get stuck. In this project, the system the timer is connected to a CPU system. The project has a software part to simulate the “CPU stuck” situation, and also to control registers, LEDs etc.

# REQUIREMENT SPECIFICATION

In table 1 below the customer requirements for the constructed system are listed.

Table 1: System requirements

|  |  |  |
| --- | --- | --- |
| **Requirement ID** | **Description** | **Status** |
| 1 | Build a watchdog timer system outside of a CPU system. The timer will get trig signal once in a second. If no trig signal detected the watchdog will reset the system. | Done |
| 2 | Using C programing design a program that “feeds” the watchdog timer with a trig signal in the needed frequency. During normal functionality the software till count up a number shown on 7-segment display. | Done |
| 3 | When pressing push button KEY[0] the system will get stuck (infinite loop) | Done |
| 4 | Switch SW[0] will activate and deactivate the watchdog timer | Done |
| 5 | Project rapport and video for verification of functionality | Done |

# TIMETABLE

Planed schedule:

* Week 1 – implementation, simulation and validation of the construction.

It was planned that the total time for this project will be 20 hours. After the project was done, the total time was 20 hours. That means the cost of the projects total cost is 20\*600kr = 12k Swedish krona

# SYSTEM ARCHITECTURE

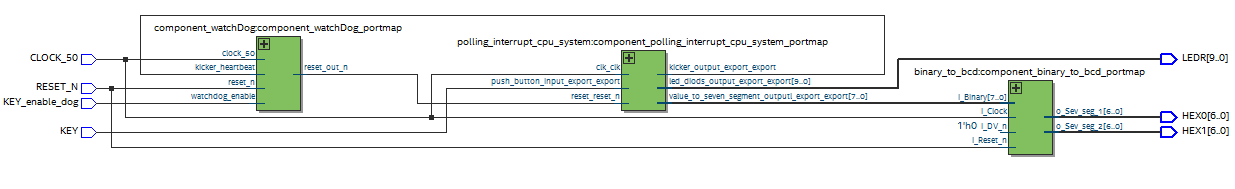
The constructed system contains three blocks, see figure 1. First one from left is the watchdog timer block, the one in the middle is the CPU system and the third one from left is binary coded decimal component.

Figure 1: RTL view of the system

## Hardware part (VHDL)

This chapter will describe the VHDL blocks and its functions.

### component\_watchDog

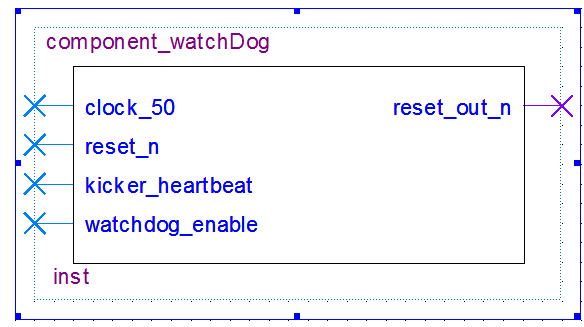
This component has the watchdog functionality, se figure 2 and table 2 for the in – and outputs for this block. When watchdog\_enable pin is activated the block will continually check kicker\_hearbeat signal. If there’s no high kicker\_hearbeat within 1 second then the reset\_out\_n will goes active low to reset the CPU system. VHDL code for this block via GitHub link in chapter 7 appendix.

Figure 2: Component’s symbol

Table 2: in and outputs for component\_watchDog

|  |  |  |
| --- | --- | --- |
| **Signal** | **Type** | **Info** |
| reset\_n | In std\_logic | System reset |
| clock\_50 | In std\_logic | System clock 50MHz |
| kicker\_hearbeat | In std\_logic | signal to feed the watchdog timer |
| watchdog\_enable | In std\_logic | 1 = watchdog active, 0 = watchdog inactive |
| reset\_out\_n | out std\_logic | signal = 0 when no kicker\_hearbeat within 1 sec |

### polling\_interrupt\_cpu\_system

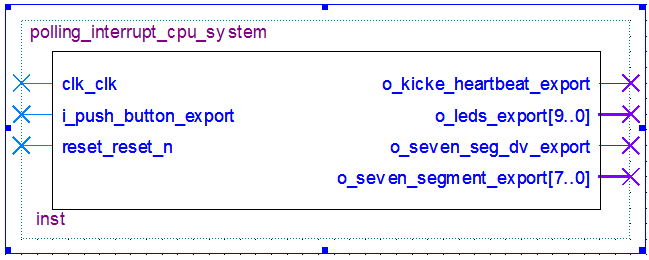
This CPU block was reused from previous project with some modification to suit this project. The component has the in – and outputs seen in figure 3 and table 3 below.

Figure 3: component’s symbol

Table 3: in and outputs for polling\_interrupt\_cpu\_system

|  |  |  |
| --- | --- | --- |
| **Signal** | **Type** | **Info** |
| clk\_clk | In std\_logic | System clock |
| i\_push\_button\_export | In std\_logic | push button KEY[0] |
| reset\_reset\_n | In std\_logic | System reset |
| o\_kicker\_heartbeat\_export | out std\_logic | kicker signal to “feed” the watchdog timer |
| o\_leds\_export | out std\_logic\_vector(9 downto 0) | control LEDs diods |
| o\_seven\_seg\_dv\_export | out std\_logic | data validate for next VHDL block |
| o\_seven\_segment\_export | out std\_logic\_vector(7 downto 0) | timer value to 7 segment LEDs |

### binary\_to\_bcd

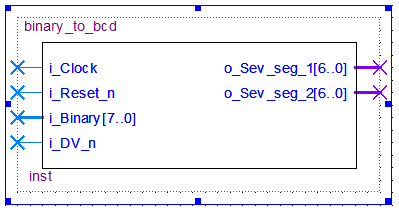
This block was reused from previous project with some modification to suit this project. This component is converts the std\_logic\_vector(7 downto 0) from CPU system to two std\_logic\_vector(6 downto 0) which can be displayed on 7 segment display. Its possible to add more output signals if more 7 segment display is needed.

Figure 5: component’s symbol

Table 4: in and outputs for binary\_to\_bcd

|  |  |  |
| --- | --- | --- |
| **Signal** | **Type** | **Info** |
| i\_Clock | In std\_logic | System clock |
| i\_Reset\_n | In std\_logic | System reset |
| i\_Binary | In std\_logic\_vector(7 downto 0) | The counter number from CPU |
| i\_DV\_n | In std\_logic | Data valid signal ensure data arrival to this block |
| o\_sev\_seg\_1 | Out std\_logic\_vector(6 downto 0) | The output data to set the 7 segment LEDs |
| o\_sev\_seg\_2 | Out std\_logic\_vector(6 downto 0) | The output data to set the 7 segment LEDs |

## Software part (C programming)

When the VHDL block in section 4.1 were done and it was time to build the C software to communicate with the hardware and drive the CPU system.

The communication with the hardware is done by HAL commands (Hardware abstraction layer) to write and read data from hardware registers. Se pseudocode below for the C software.

***main****(){*

***while****(1){*

***while****(TIMER\_READ <= 25000000){*

***if****(push\_button\_is\_pressed){*

*set LEDs;*

*write\_low\_signal\_to\_kicker\_heartbeat;*

***while****(1);//Infinite loop*

*}*

*}*

***if****(counter == 99){*

*counter = 0;*

*}*

***else****{*

*counter += 1;*

*}*

*set seven\_segment\_LEDs;*

*write\_high\_signal\_to\_kicker\_heartbeat;*

*}*

***return*** *0;*

*}*

C code is available via GitHub link in chapter 7 appendix.

# VALIDATION

The validation is done by loading the hardware design to the DE-10 Lite FPGA board and running the C-code. Table 6 shows the testes to validate. The result from the validation can be seen in a short YouTube video <https://www.youtube.com/watch?v=2QJb1YyQV8g>

Table 6: validation test cases

|  |  |  |  |
| --- | --- | --- | --- |
| **Case ID** | **Description** | **Expected behavior** | **Result** |
| A | Watchdog disabled while pressing bush button | CPU stuck. All LEDs are set. Timer value on 7 segment stops | Ok |
| B | Watchdog enabled while pressing push button | CPU stuck then the watchdog reset the system | OK |

# CONCLUSIONS

It’s obvious that a watchdog timer increase the availability time of a system. If there’s now “heartbeat” signal from a system its means that the system needs to restart and this is automatically done by the watchdog. If a CPU system get stuck without a watchdog timer the system will be unavailable until a manually reset is done.

# APPENDIX.

The documentation for the watchdog timer can be found in GitHub via this link: <https://github.com/S81f/VGA_intellectual_property>