Design Report
Stopwatch

Name: Jan Szczekot

Date: 15/05/2023

Contents

Chapter 1 – Introduction

FPGA(Field-Programmable Gate Array) is a type of integrated circuit that allows the user to be configured after manufacture. It is build from various programmable logic blocks that allows creating custom digital circuits which makes them very flexible and suitable for wide range of applications.

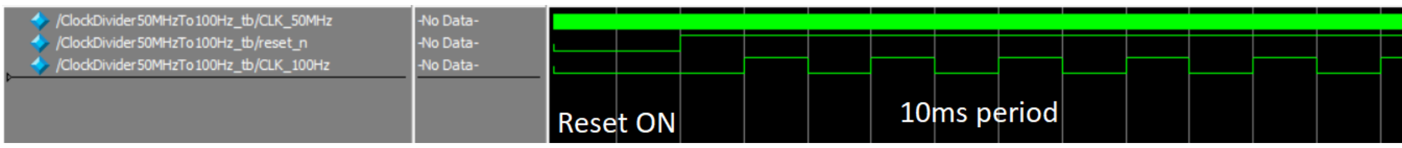
The aim of this project is to create full operating stopwatch module that counts the time and display it on inbuilt seven segment display and allows user to interact with it using buttons and switch based on MAX10 FPGA architecture. Whole module consist of some sub-modules like clock divider, stopwatch logic, binary to seven segment encoder organised in top-level design. User interface is capable of performing start/stop, hold and reset function.

The learning outcome is to gain deeper understanding of FPGA architecture, sequential and structural logic as well as some practical experience of coding in Verilog language. Project focuses on importance of testing and checking correct behaviour of each designed module and combining them all together to create working top-level design structure. Another thing that this project provide is the skill of implementing designed code onto the physical board and manual interaction with the inputs and outputs of the design.

Chapter 2 – Design and Development

Section 2-1: Clock Divider Module

The idea of creating 100Hz signal from 50MHz input is to count the number of cycles of the input and toggle the output every 250000 cycles which will result in desired frequency square wave. With the reset set to 0 output should not be generated and assigned a value of 0.
In the Verilog implementation input is assigned to built in the board 50MHz clock – "CLK_50MHz" and to the active-low reset signal – "reset_n" and output is assigned to – "CLK_100Hz". Then 25 bit wide register – "counter" – is created and used to count the number of the cycles of input clock signal. Always block determines the behaviour of the code, every positive edge of the input clock cycle or negative edge of reset button code is executed. Inside the code there is an if-else statement with another if inside the first else statement:
First statement checks if reset is active, it is active-low so it negates the value to make the condition true. When the condition is fulfilled "counter" and output "CLK_100Hz" are set to 0. Otherwise the counter is incremented by 1then the code checks if counter reached value of 249999 when this happens "counter" is set to 0 and the CLK_100Hz is toggled. The value 249999 corresponds to 250000 cycles of 50Mhz clock which toggles the output which result in 250000 cycles with output on low-state and another 250000 cycles on high-state which in summary gives the square wave of frequency 100Hz – our desired clock signal.


/ClockDivider50MHzTo100Hz_tb/CLK_50MHz
/ClockDivider50MHzTo100Hz_tb/reset_n
/ClockDivider50MHzTo100Hz_tb/CLK_100Hz
-No Data-
-No Data-
-No Data-
Reset ON
10ms period

When an active-low reset is 0 then the clock is not generated as visible on the above graph when the reset is set back to 1 then the clock divider generates 10ms peridot square wave which corresponds to 100Hz.

Section 2-2: Stopwatch Logic Module

The Stopwatch Logic module consists of 3 user controllable inputs and 100Hz clock signal generated in previous module. The aim of the module is to create 3 numbers that would correspond to the number of minutes, seconds and decaseconds passed with the limits for them to 99mins, 59secs and 99decs. The limits are introduced to assure continuous incrementation of the numbers with the conversion of units. When limit of first number is reached it should start counting from the beginning and increment next number by 1. When the limit of the last number is reached it should set the overflow flag to 1 meaning that module reached its maximum number that we want it to count to.

In the code 4 inputs are set “CLK_100Hz”, “reset_n”, “start_stop” and “hold” which name explains their function, then there are 4 outputs “stopwatch_unit_mins”, ”stopwatch_unit_secs”, “stopwatch_unit_mins” which purpose is to store value representing counted time in corresponding units and “timer_overflow” which works as indicator when maximum value is surpassed. Some registers are initialised such as “count” which can work as an indicator of number of clock cycles passed, “state” that stores information about current state, and “mins_count”, “secs_count”, “decs_count” that allows to perform operations to count time. Code use always block that triggers on positive edge of the clock or negative edge of the reset same as previous module. First thing is to check if the reset value is low that would reset the counter and assign 0 to the values of time and flags.

Then 2 case statements are made one of them implement the logic of switching states:
0 – initial default state
1 – running state
2 – hold state
3 – stop state
So initial “0” state is switched to “1” – running state when “start_stop” is low. At running state “hold” and “start_stop” are checked again and states are changed to “2” and “3” correspondly at low “hold” and low “start_stop” entering stop state. At state “2” if “hold” is released code returns to running state, at state “3” when “start_stop” is pressed again code comes back to running state.

Second case statement - using the same states - implement logic of operations on number representing the time passed. At state “0” counters and flag are set to 0, then the most important “running state” increment by 1 number of deciseconds counter, then check if the value reached limit if it does, deciseconds counter is set to 0 and seconds counter is incremented by 1, then the same process happens for seconds counter, when the limit is reached it is reset to 0 and minutes counter is incremented. When minutes counter reaches its limit it is also reset but in addition “count” is reset as well and overflow flag is set to high. States “2” and “3” only set count value to 0. Outside the always block computed numbers of decs, secs and minutes are assigned to the output of the whole module.

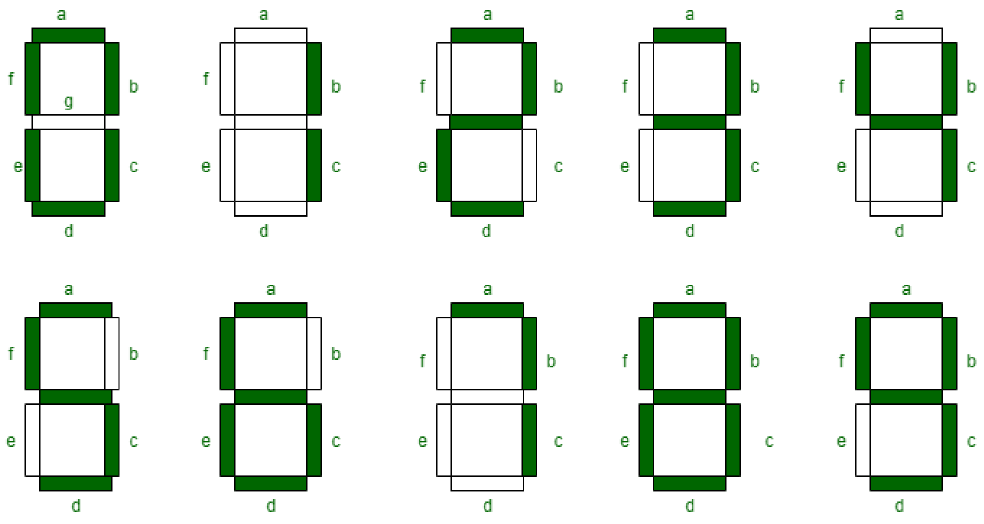
Section 2-3: Binary to Seven Segment Encoder

The function of the Binary to Seven Segment Encoder is to convert the binary values of our counted times into the binary values that represent decimal numbers on Seven Segment Display. The goal of displaying the values on Seven Segment Display is to allow the user to see the counted time in human friendly environment of decimal numbers.

First step to achieve this is to convert binary numbers into BCD (Binary Coded Decimal) which is 4 bit long binary representation of a number form 0-9. Then BCD needs to be converted into 8 bit long binary value that would display correct decimal number on Seven Segment Display.

BCDEncoder sub-module provided in this project uses double dabble algorithm which take the 8 bit binary input and shift it “left” bit by bit into 12 bit BCD output. This output can be divided into 3 separate 4bits long BCDs as the shift process happens the algorithm checks if any of this BCDs is greater than 4, if it is then next step is to add 3 to this BCD. Process continues as long as last binary value is shifted. The resulting output is an actual BCD representation of a binary input.

The next sub-module used is BCDToSevenSegmentEncoder that uses BCDs generated by previous module and convert them into binary representation on decimal number to be displayed on the 7Seg. It takes all 4 bits from BCD and perform logic operations on them so that the each segment of the display is turned on or off correctly. This approach was used and shown during lab 4.2 of this module. Each segment of the display has its own logic operations based on this 4 bits of the input which was derived manually using Boolean algebra. Important thing to notice is that overall logic had to be inverted due to that the Seven Segment Display on the board flashes on when the state of the pin is low.



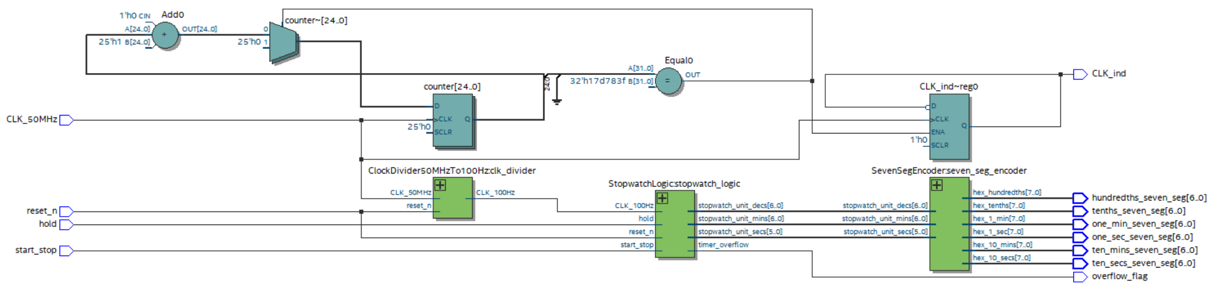
The whole module uses three of BCDEncoder modules to create 3 sets of BCDs for minutes, seconds and decaseconds. Only thing important in this code is that BCDEncoder works on 8 bit numbers and our inputs have 7 or 6 bits so it was necessary to add one or two zeros in the front of the number which does not change its value but assures that code would work. Then these sets of BCDs are passed to BCDToSevenSegmentEncoder modules - there are 6 of them in total – to convert them into the value that should be displayed on 7Seg. First set of BCDs corresponding to minutes is split and first 4 bits are passed to the first BCDToSevenSegmentEncoder and another 4 are passed to the second, last 4 bits of BCD set are not used as the numbers would never be greater than 100 and therefore require 3 BCDs. The same process happens to other sets to create decimal display for seconds and decaseconds.

Section 2-4: Stopwatch (Top-level entity)

The top level design stopwatch combines modules designed and explained earlier, ClockDivider divide the input 50MHz clock into 100Hz clock signal, StopwatchLogic handles the logic of the stopwatch based on input states and finally SevSegmentEncoder displays the values on the SevenSegment.

Input CLK_50MHz is passed to the clock divider to create slower clocks signal that runs StopwatchLogic module along with the start_stop reset_n and hold inputs. StopwatchLogic creates stopwatch units in mins, secs and decs and passes them to SevenSegEncoder which converts them into binary value of the decimal that should be displayed. In addition clock indicator is toggled every second to inform user if the main clock signal is running. This indicator is independent of any inputs and should work as long as the device is turned on.

This top-level design is great example why structural logic and behavioural logic is useful, dividing the project into smaller modules achieved modularity and make sure that maintain and read the code is much simpler. These modules can be easily connected together and reused in many different design. Behavioural logic allows for the description of the functionality without implementing the hardware structure which make the design flexible and easier to modify.

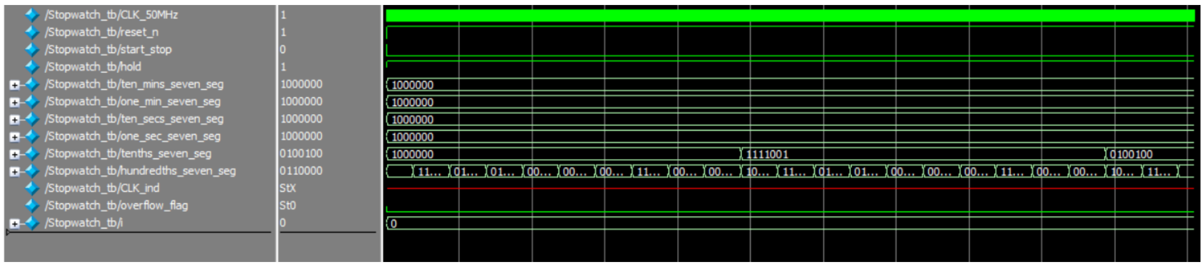


On the above diagram it is visible how each modules are connected together and how inputs flows through the system to create outputs.

Chapter 3 – Validation

Testing process of the whole Stopwatch module would require to change every input and try different combinations of them to test if the module behaves as it was meant to. Stopwatch module input is 50MHz clock which has a period of just 20ns but to be able to see anything in our simulation at least couple of cycles of the 100Hz clock should pass. Therefore testing the whole module using testbenches is not co convenient and results in huge numbers of waiting time, therefore it is better to just test each module individually.

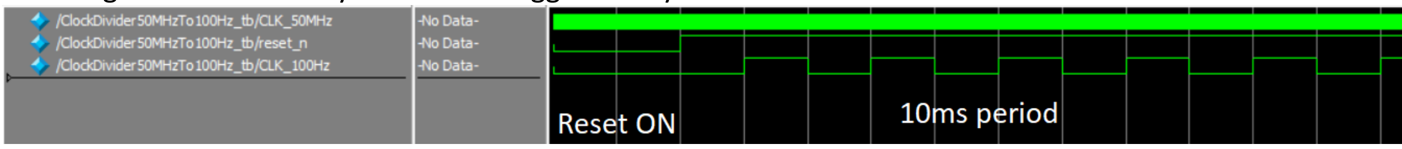
The testbench for the whole stopwatch module is designed in a standard way assigning inputs and creating device under a test and then values of inputs are set to 1 which corresponds to stop state. After 10ns start_stop is set low which puts the stopwatch in running state. Next step is to wait maximum amount of time to cover most cycles of the 100Hz clock and then put stopwatch back into stop state and finish the test. Input clock in this testbench uses always block to toggle every 5ns possible the lowest value.



```
/Stopwatch_tb/CLK_50MHz               1
/Stopwatch_tb/reset_n                 1
/Stopwatch_tb/start_stop              0
/Stopwatch_tb/hold                    1
/Stopwatch_tb/ten_mins_seven_seg      1000000   1000000
/Stopwatch_tb/one_min_seven_seg       1000000   1000000
/Stopwatch_tb/ten_secs_seven_seg      1000000   1000000
/Stopwatch_tb/one_sec_seven_seg       1000000   1000000
/Stopwatch_tb/tenths_seven_seg        0100100   1000000  1111001  0100100
/Stopwatch_tb/hundredths_seven_seg    0110000   11... 01... 01... 00... 00... 00... 11... 00... 00... 10... 11... 01... 01... 00... 00... 00... 11... 00... 00... 10... 11...
/Stopwatch_tb/CLK_ind                 StX
/Stopwatch_tb/overflow_flag           St0
/Stopwatch_tb/i                       0         0
```

Resulting simulation does not cover the whole range of values but it shows that in deed values start to increment by 1 starting from decaseconds.

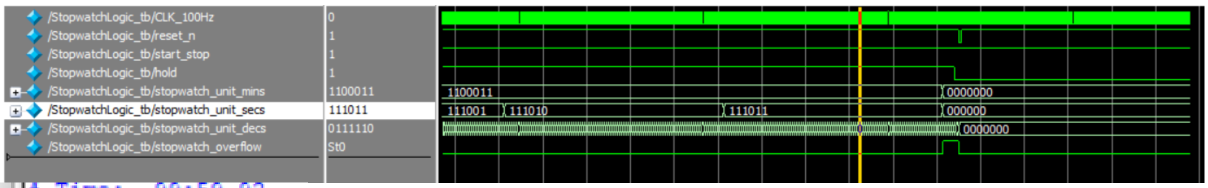
Testing ClockDivide50MHzTo100Hz:
It uses reset_n, and 50Mhz clock, device under test is instantized and it firstly set the reset to 0 and switches it to 1 after some time then continuous for ten 100Hz clock cycles. Input clock is generated in always block and toggles every 20ns.



```
/ClockDivider50MHzTo100Hz_tb/CLK_50MHz    -No Data-
/ClockDivider50MHzTo100Hz_tb/reset_n      -No Data-
/ClockDivider50MHzTo100Hz_tb/CLK_100Hz    -No Data-
```

Reset works as intended and the output generated is an actual 100Hz square wave.

Testing StopwatchLogic:
Testbench toggles the input clock every 5ns and then initial values of reset, start_stop and hold are assigned. The aim is to change values and let the stopwatch run with these values for some amount of time and simulate counting up to the overflow. The test displays current time of the timer and notifies when the overflow occurs.

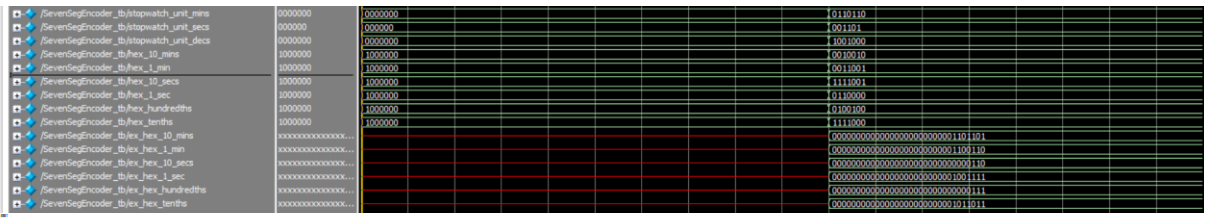


```
/StopwatchLogic_tb/CLK_100Hz                 0
/StopwatchLogic_tb/reset_n                   1
/StopwatchLogic_tb/start_stop                1
/StopwatchLogic_tb/hold                      1
/StopwatchLogic_tb/stopwatch_unit_mins       1100011
/StopwatchLogic_tb/stopwatch_unit_secs       111011
/StopwatchLogic_tb/stopwatch_unit_decs       0111110
/StopwatchLogic_tb/stopwatch_overflow        St0
```

```
# Time:  99:59.92
# Time:  99:59.93
# Time:  99:59.94
# Time:  99:59.95
# Time:  99:59.96
# Time:  99:59.97
# Time:  99:59.98
# Time:  99:59.99
# Time:    0: 0. 0
# Timer Overflow!
# Time:    0: 0. 1
# Timer Overflow!
# Time:    0: 0. 2
# Timer Overflow!
# Time:    0: 0. 3
# Timer Overflow!
# Time:    0: 0. 4
# Timer Overflow!
# Time:    0: 0. 5
# Timer Overflow!
# Time:    0: 0. 6
# Timer Overflow!
# Time:    0: 0. 0
```

On the graph it is visible when stopwatch reached its limits and raised the overflow flag as it happened it reset itself to 0. Then hold was pressed to maintain this state. This testbench confirmed working function of the reset, start_stop and hold inputs as well as logic for the whole stopwatch including overflow.

Testbench for SevenSegmentEncoder is self checking testbench for one case, it assignes calculated values for defined inputs of numbers and compare them with the actual value of the module. If comparison is correct it returns the message and display the time. Otherwise it notifies that comparison was not successful.



```
/SevenSegEncoder_tb/stopwatch_unit_mins     0000000            0110110
/SevenSegEncoder_tb/stopwatch_unit_secs     000000             001101
/SevenSegEncoder_tb/stopwatch_unit_decs     0000000            1001000
/SevenSegEncoder_tb/hex_10_mins             1000000            0010010
/SevenSegEncoder_tb/hex_1_min               1000000            0011001
/SevenSegEncoder_tb/hex_10_secs             1000000            1111001
/SevenSegEncoder_tb/hex_1_sec               1000000            0110000
/SevenSegEncoder_tb/hex_hundredths          1000000            0100100
/SevenSegEncoder_tb/hex_tenths              1000000            1111000
/SevenSegEncoder_tb/ex_hex_10_mins          xxxxxxxxxxxxxx...  00000000000000000000000001101101
/SevenSegEncoder_tb/ex_hex_1_min            xxxxxxxxxxxxxx...  00000000000000000000000001100110
/SevenSegEncoder_tb/ex_hex_10_secs          xxxxxxxxxxxxxx...  00000000000000000000000000000110
/SevenSegEncoder_tb/ex_hex_1_sec            xxxxxxxxxxxxxx...  00000000000000000000000001001111
/SevenSegEncoder_tb/ex_hex_hundredths       xxxxxxxxxxxxxx...  00000000000000000000000000000111
/SevenSegEncoder_tb/ex_hex_tenths           xxxxxxxxxxxxxx...  00000000000000000000000001011011
```

```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# Case 2 Test failed

VSIM 2>
```

Chapter 4 – Conclusions and Critical Reflections

The part in my project that I am the most proud of is certainly whole StopwatchLogic module as well as creating a testbench for it. Creating a working logic for the timer was a difficult task, three inputs that's controls its behaviour did not make it easier at all. Making sure that each input is read as it should and the system responds correctly required a lot of debugging and changing the code many times. In addition it also required to design state changing logic with a lot of confusion mainly because of the negative edge triggered inputs which are not so initiative. So eventually after all this difficulties I am proud that the whole module works as intended.

Apart from the things mentioned above the challenging part was for sure SevenSegEncoder module and the reason why is that it consists of many submodules and different binary representations of the numbers. So the main difficulty here was assigning correct bits of the binary bus to the correct inputs and making sure which order should be used. A lot of confusion with most significant and least significant bits, whether first bit should be last etc. Also the testbench for the module was quite difficult because of the self checking part where it did not work as planned because I did not account for the delay in comparison part and so the code did the comparison between expected values and actual values as the values were changed and some register still could use previous ones and therefore it was not working.

The function I though about is to create different modes of operating for example instead of timer we could design an actual clock that would display hours, minutes and seconds. This function would not be hard to implement because it would only require a different clock frequency some different limits but the general logic would stay the same. Another function which that could be done but its more complex is to create a timer where the user could set the time and it would count down until zero when some indicators would be implemented. This function is much more complex and requires whole different logic for the timer as well as detecting user inputs to set the time. It would require much more time to implemented this probably a month would be reasonable time to do it.