Assignment 3: Stopwatch

Student Name: Paul Doherty

Student Number: 10387129

Assignment Co-worker: Scott Condron

# Objective:

The objective of this assignment is to design a stopwatch that measures time intervals and displays the result on the display of the provided circuit board. The stopwatch needs to of the following specification.

* The stopwatch shall have at least one pushbutton control. This button shall be used to start and stop the time measurement. After reset, the stopwatch should display a time value of zero. The first press of the button should start the time measurement, and the display should show the time value as it advances. When the button is pressed during a time measurement, it should stop the measurement, and the display should show the measured time interval. If the button is pressed again, the time measurement should resume from the displayed value, and so on. The reset button may be used to set the time to zero.
* The display shall use the four 7-segment displays on the circuit board to show the time

value. It shall be updated at a rate high enough to avoid any visible flicker of the digits, but

not higher than 5000 times per second. The minimum requirement is to display time in

seconds, in hexadecimal, with a resolution of one sixteenth of a second.

* If the time interval exceeds the maximum value that can be displayed, the time may roll

over to zero without warning.

* The hardware shall be designed to work on the Digilent Spartan-3 circuit board provided.
* All registers in the design shall operate on a 5 MHz clock, to be provided by a digital clock

manager (DCM) on the FPGA. All registers shall use asynchronous reset, derived from

btn3 on the circuit board, but also asserted when the DCM is not ready. The hardware to

generate these signals will be provided. A constraints file will also be provided, defining

the pin connections to the FPGA.

# Introduction:

The following report gives a detailed account of how our team designed a stopwatch to meet the given specification of the assignment. The report outlines our design methods from top level module description of the blocks needed to successfully implement a working stopwatch that operates from a clock with frequency of. The report then shows how these individual blocks were designed as separate modules that all perform a unique task. These modules were further simplified into blocks that could be realised by RTL diagrams and thus be described in Verilog to configure the design onto the provided FPGA chip.

## NOTE:

Before continuing reading it is important to note that sketched design solutions will not be available to view on this version of the report although they will be referred to. These sketched design solutions which include the layout of the different modules, their connecting signals, the inner blocks that make up the module and the RTL diagrams will be available to view on the hard copy of the report.

# Top Level Design

This highest level of the design is where only the input and output signals are considered. What design modules are needed to make the output signals behave as the specification requires and how is this done from the given input? It does not consider how the individual modules are designed; only what they need to do to enhance the input signal to give the desired output signal.

From the given specification the stopwatch will have two inputs: the pushbutton and the reset button. The pushbutton will start the timer on the first press and stop on the timer on the second press. The timer will resume counting on the third press and stop on the fourth etc. the reset is the only input that will force the count back to zero. The only output is the four digit seven segment display which shows the counted time interval. From the ‘The Digilent Spartan-3 Board manual’ the display needs two input signals: one 4-bit signal to tell the display which digit to light up and one 8-bit signal to tell it which segments to light up so that it display shows the correct time interval. Therefore our design requires two outputs: one for the digits and one for the segments.

## Required Modules

To decide what modules are needed to give the outputs the desired behaviour this required behaviour needs to be analysed; but also the undesired physical effects of the input pushbutton and the randomness of the human controlling it. The First issue to be handled is contact ‘bounce’ from the input button. Contact bounce a phenomenon that occurs in almost all switches. It occurs when a switch is closed but instead of the switch contacts establishing continuity in a single crisp moment the switch contact bounces for a short period of time before coming to rest at full desired contact[1]. Figure 1[1] shows an example of contact bounce in a circuit.

**DELAY!!!**

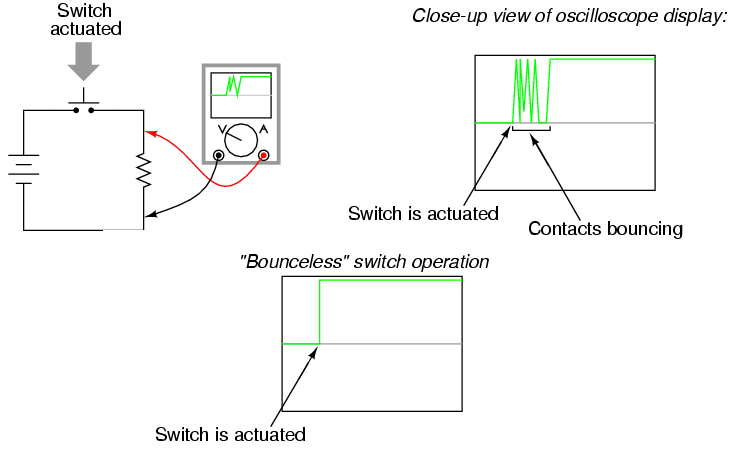


Figure 1: Contact bounce in circuit[1] and also an ideal “bounceless” switch

It is clear that any bounce seen from the input pushbutton will greatly affect the performance of the stopwatch. Next is the issue that deals with the random period of time for which the human pressing the input pushbutton will physically hold down the button for. This block needs to ensure that should the human go as far as holding the button down for a number of seconds, then the stopwatch will be able to continue counting and not stop counting until the button is released and pressed again. Therefore a ‘debounce’ block which we have named ‘The Bounce Stopper’ will be the first block seen by the input in our top level design module. Its input will be a 1-bit signal from the pushbutton and its output will be a 1-bit signal pulse; which we have named ’Pulse’.

Once the problem of contact bounce is overcome, the next problem that the input experiences that affects the desired output is concerning the actual state of the stopwatch, i.e. is the stopwatch counting or is it simply displaying a steady value on the display? To solve this issue a block in the top level design is needed to decide if the stopwatch should be counting or not counting. This will be the second block seen by the input. Its input will be the 1-bit output from ‘The Bounce Stopper’ module and its output will be a 1-bit signal pulse which we have named ‘Run’. This block is named “Control Block”. These two blocks handle all unpredictable behaviour that may be seen from the input pushbutton and unpredictable hold times from the human.

Once these issues from the input are overcome we can focus the real practical function of the stopwatch; counting time. This is the most important block of the design as it is what defines the design and also what type of stopwatch we will be outputting (type is referring to what does the counter display? hexadecimal or decimal, does it display minutes or just seconds? Etc.). The input to this block will be the 1-bit output from the Control Block “Run” and its output will be a 16-bit signal containing the binary code for the time that is to be displayed. The block is named “SLOW\_PULSE\_GENERATOR” its output signal is named “Time”.

The last block that is needed in the top level design is an interface for the display of the FPGA. This interface has the role of processing the incoming bits from the output of the “SLOW\_PULSE\_GENERATOR” block and outputting them to the display. This block needs two outputs: one to tell the display on the FPGA which digit to light up and one tell it which of the seven segments to light up to correspond to the counted time interval for that digit.

Thus concludes the top level design of the stopwatch. A drawing of this can be found on the hard copy of the report. Below is a straightforward list of the individual modules needed to implement the stopwatch:

1. The Bounce Stopper
2. Delay
3. Control Block
4. Counter Block (Slow\_Pulse\_Generator)
5. Display Interface

# The Bounce Stopper

In order to overcome the problem of contact bounce that was described above, it was agreed that the solution used to solve the problem would also need to have no effect on the accuracy of the output counter i.e. there would be no delay from the input button pressed to the output display. The solution that was used for this problem did contain a delay but since the counter would see the same delay each time the pushbutton was pressed then the delays would cancel out and so no delay would affect the overall result of the system.

The following behavioural outline of the module was agreed:

* When the input from the pushbutton goes high i.e. someone pressed the button to begin the stopwatch, “The Bounce Stopper” detects this and immediately starts a timer that is to count for a period of time that is sufficiently long enough for the contact bounce to have stopped.
* When the timer is up “The Bounce Stopper” then checks the input to see if it is still high or turned low. If it detects a high (1) input after the timer then it assumes that the button has indeed been pressed and outputs a “1”, if it detects a low input (0) after the timer it assumes that it was contact bounce that started the timer and so outputs a “0”. This design ensures that if bounce is detected when the button is released then the system will not mistake for an input

This design requires a separate module in our top level design which has the sole role of acting as the delay for the bounce stopper. This delay block will have will have an input coming from the “The Bounce Stopper” block its output will also be an input to the “The Bounce Stopper” block. This delay block which act as a timer for the “The Bounce Stopper” block is be named “Delay” and it will be analysed later.

It was agreed that the most efficient solution for “The Bounce Stopper” block was to design it as a Mealy machine with three states. The Mealy machine needs two inputs and two outputs. The two inputs are the 1-bit pushbutton named “button” and the 1-bit output from the “Delay” block named “timerOF” and the two outputs are the 1-bit signal “pulse” which feeds the “Control Block” and the 1-bit signal “timerstart” which feeds the input to the “Delay” block. The block also has an input clock named “clk” which triggers all the registers synchronously on its rising edge and an asynchronous reset input named “rst” which forces the state machine back to a known initial state. The three states are:

1. The known initial state “INIT”.
2. The state which represents a detected input from “button” called “waitingp” which waits on the timer to then decide its next move,
3. Finally a state which shows that a true input has indeed been detected called “pout”.

The state diagram for “The Bounce Stopper” which describes the exact behaviour of this block can be found hand drawn attached to this report. **Test bench simulations for this module which ensure that it is working as required are attached to this report**

## Delay

While the “Delay” block is implemented as a separate block in the top level design in this report I am going to treat it as a sub-part of “The Bounce Stopper” block because its only input comes from this block and its only output is used as an input to “The Bounce Stopper” block. This “Delay” block was entirely designed and implemented by Scott Condron.

**RTL diagrams and test bench simulations are attached to report.**

# Control Block

The “Control Block” was designed as a simple two state mealy machine. One state for when the stopwatch is counting named “started” and a state for when the stopwatch is not counting named “stopped”. This block has two synchronous inputs: the clock signal declared as “clk” and the output from the “The Bounce Stopper” block declared “pulse” and one asynchronous reset input named “rst” which forces the “Control Block” to the known state “stopped”. The “Control Block” has one output declared as “Run”, this feeds the input to the block named “SLOW\_PULSE\_GENERATOR”. The most important aspect of this block is that it changes state on the next clock edge of the when it sees a high input from “pulse” and then it remains in that state until it sees another input. The state diagram which completely describes the behaviour of this block is attached to this report. The test bench simulations of this block are also attached.

# Counter Block

The Counter block in the top level design is the most important block as it is the block that handles the operation of the stopwatch i.e. counting time intervals. This block has one input, the 1-bit signal “Run” and one output: a 16-bit signal declared as “Time” which holds the binary code for the time in seconds and minutes. The 16-bits of the signal “Time” holds four different values in each 4-bit section of the signal. They are divided as follows:

* bits - hold the value displaying tenths of seconds
* bits - hold the value displaying seconds
* bits - hold the value displaying tens of seconds
* bits - hold the value displaying minutes

Before we can implement the decimal counter there is some processing to be done. It needs to be ensured that the counter is in fact counting accurately in tenths of seconds. Since the clock has a frequency of a slow pulse can be generated from this that will output a high pulse every tenth of a second. Because of this, it easier to think of the counter block at two stages before implementing them together.

The first stage is the above mentioned slow pulse generation stage. This is implemented with an adder, a multiplexer, a register and a comparator. The input this section will be the 1-bit signal “Run”. The comparator value is the most important thing here as this determines the rate of the slow pulse. Doing the math so that there is a slow pulse every tenth of a second:

From this value it is clear that the signals used in this stage need to be 19-bits wide. The multiplexer is enabled by the 1-bit output of the comparator which we have named “spulse”. This is needed to choose which value, namely 19-bit “nextQ” is to be loaded into the register and on the clock edge the output of register named “Q” is to be assigned this value. “Q” is the input to the comparator. If the value is from the comparator then the multiplexer passes the next count value from the adder which is a 19-bit signal named “added”, if the value is from the comparator then the multiplexer passes a 19-bit . The adder is to add the signals “Run” and “Q” together. This description can be seen clearly in the RTL diagram attached to the report.

The second stage of the counter block is the more complicated of the two. This stage need to take slow pulses and count them and allocate the time intervals into the 16-bit signal described at the beginning. The easiest way to do this is to use four different counters all similar to the counter described above. Each counter in this section will correspond to 4-bits of the signal “Time”. Each of the multiplexers in these counters will use the signal “spulse” as one of its enablers. The other enablers of the multiplexers will be the outputs of the comparators of each of the individual counters depending on what section of the signal “Time” is being processed, i.e. the multiplexer of the first counter is enabled by “spulse” and the output of its comparator declared as “t2”, the multiplexer of the second counter is enabled by “spulse” and the output of its comparator declared as “t3” and the previous comparator “t2”, the same principle is carried out for the multiplexers of the next two counters declaring the outputs “t4” and “t5“. This is much clearer in the RTL diagram attached. A similar algorithm is employed for the comparators, i.e. the first comparator only depends on its own input from the register, the second comparator depends on its own input from the register but also on the conditions of the first comparator. The same idea is followed for the third and fourth comparators with fourth comparator depending on the values from all four counters. The reason for this is to control the overflow of the counted bits so that the entire correct time can be displayed in minutes, tens of seconds, seconds and tenths of seconds. The 16-bit output signal gets its value by truncating the outputs from each of the registers in the four counter blocks; “time1” corresponds to the tenth of seconds counter and is the least significant set of bits in the “Time” signal, “time2” corresponds to the second counter and represents the seconds of the output and is the second least significant set of bits in the output signal “Time”, “time3” corresponds to the third counter and represents the tens seconds of the output and is the third least significant set of bits in the output signal “Time”, “time4” corresponds to the fourth counter and represents the minutes of the output and is the most significant set of bits in the output signal “Time”. The entire behaviour of the design can be clearly seen in the RTL diagrams attached to the report. The decimal counter was simulated on a test bench; the results of the test bench are attached to the report.

# Display Interface

The role of this block is to take its input “Time”, a 16-bit signal, and transform the signal into something that can be displayed on the FPGA. The input signal “Time” was designed to carry the values of four different outputs each of which are to be allocated a digit on the display. However the first problem that needs to be tackled in this block is that of the screen resolution since the is much too fast and this will cause the seven segment display to appear blurry as though all the lights are being lit up. To solve this problem the same method that was used to generate the slow pulse in the counter block needs to be employed here. The exact same digital circuit was used except the value of the comparator is . This value meets the required specification as the display is being updated less than times per second, it is being updated at a rate of times per second. It also now has a time resolution better than , it has a s.

The output of the comparator is a 2-bit signal named”. This 2-bit signal will now constantly repeat the binary values . These binary values will be used as the enabler to multiplexer which will select which section of bits of the signal “Time” will be sent to a look-up-table (LUT) and then sent to the seven segment display of the FPGA. The values of “z” will also be sent to another LUT which will put them into a form that can be sent to the display to select which digit to light up. The values for the LUT’s were found in the manual of the FPGA. It important to note that the displays of the FPGA is active low, i.e. a corresponds to a high signal and a corresponds to a low signal. The least significant bit of the segment inputs controls a separate light which is a decimal point. In our design we have decimal point lighting up twice; once for separating the minutes from the seconds and again separating the seconds from tenths of seconds. The RTL diagrams and the test bench for the simulation of this can be seen attached to the report.

I will note that the Verilog code for the LUT also contains values of all the hexadecimal value of the binary input. This is because our original design was implemented entirely in hexadecimal before upgrading the stopwatch to decimal.

# References

[1] <http://www.allaboutcircuits.com/vol_4/chpt_4/4.html>