嵌入式系统结构与设计课程实验

实验报告 8

Timer with millisecond precision

邱迪聪 / 11331262

2013/12/4

Content

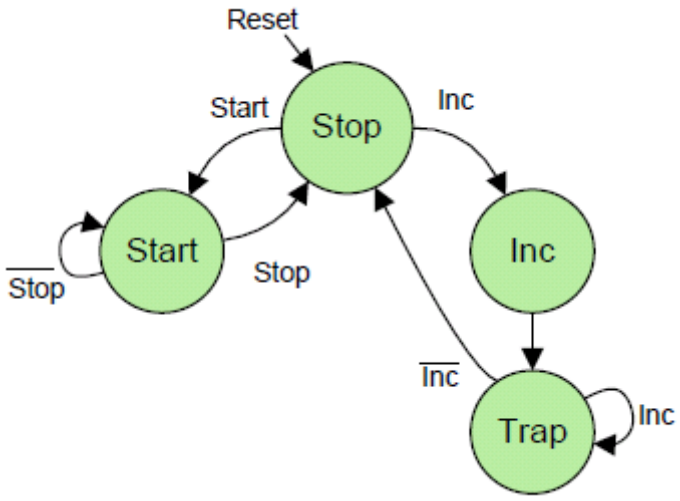
1. Target

- Review the learnt before;
- Further understand the High level design for FSM, and learn how to read ISE report to optimize your design.

2. Timer with millisecond precision

2.1 Designing

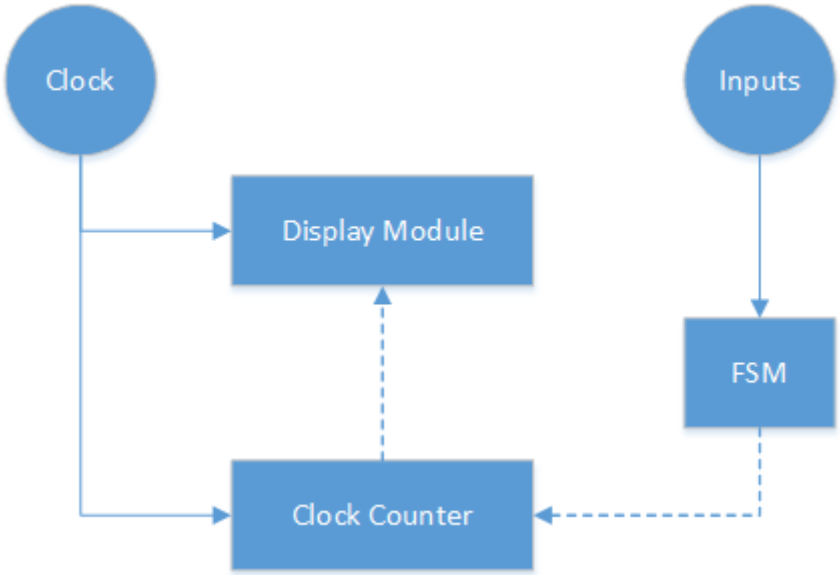
The FSM of the design is based on the following one.



This finite state machine provides a good overview of the design, but the states of it is too many, which can be optimized. The optimization accords to the running state of the timer, and only two states are necessary: RUNNING and NOT RUNNING. Only one bit is necessary to remember the two states. And the transition rules between of the two states are:

- The initial state is NOT RUNNING;
- Clicking Start button → RUNNING;
- Clicking Stop button → NOT RUNNING;
- Inc button down → RUNNING;
- Inc button up → NOT RUNNING.

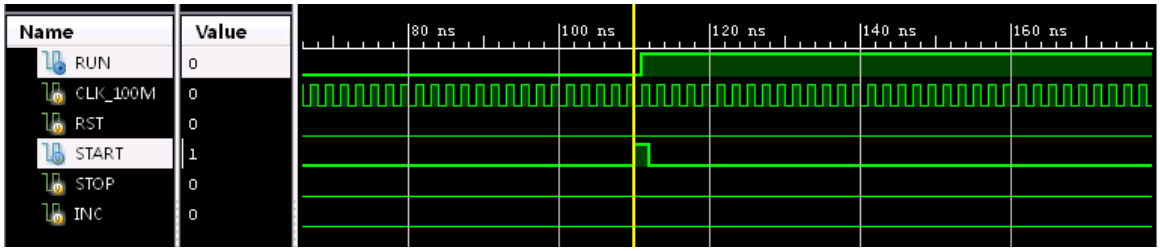
The timer consists of FSM, 7-segment digital tube display module, and clock counter module. The last two components are driven by the same clock signal source. The following diagram shows their relationship.



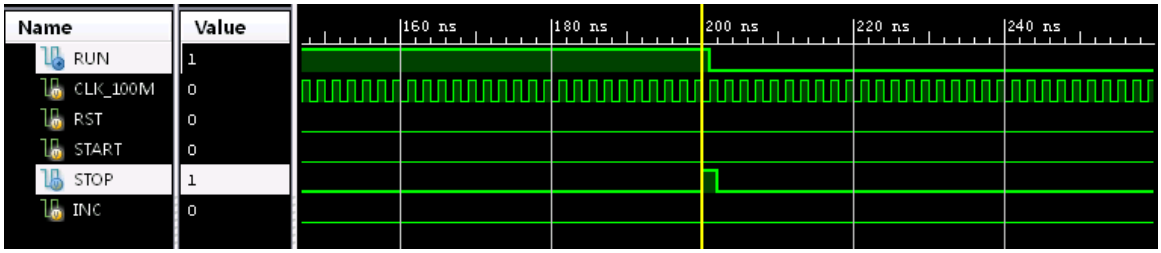
2.2 Simulation

The simulation focuses on state transition of the timer FSM, and the result is shown as followed. Note that the result displays in order respecting to time.

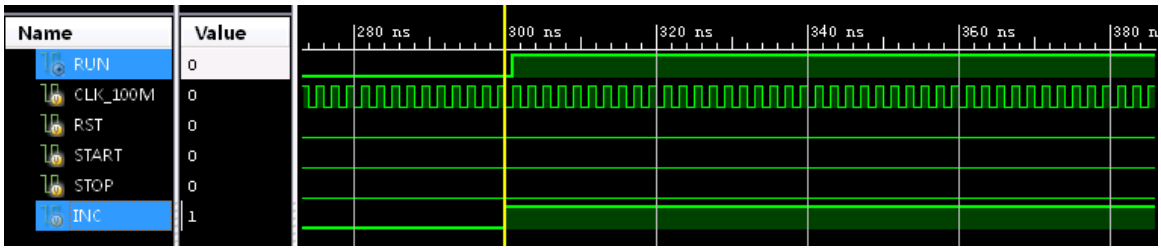
Click START:



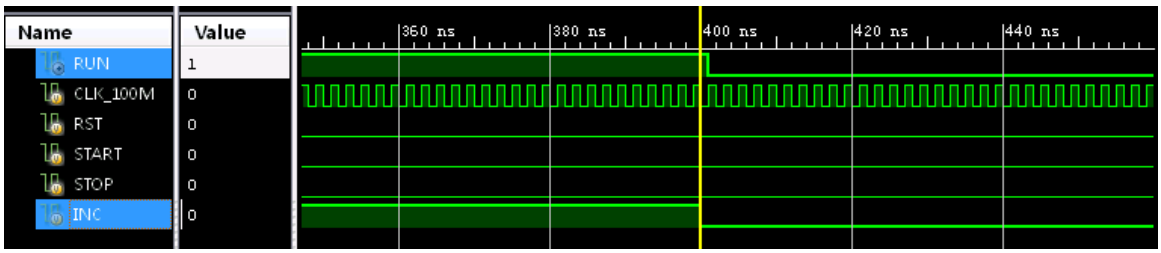
Click STOP:



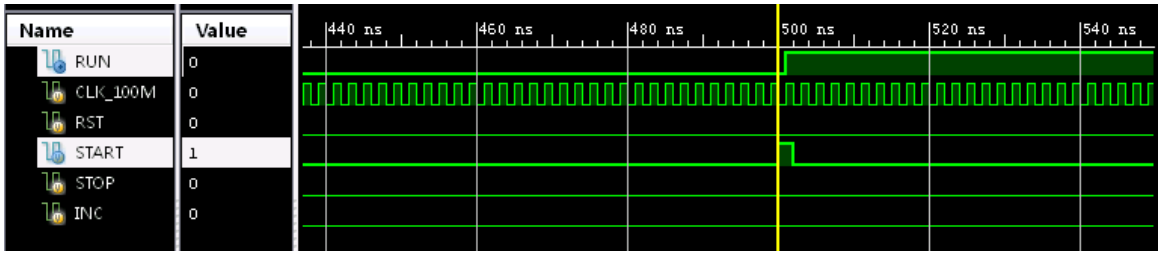
Press down INC:



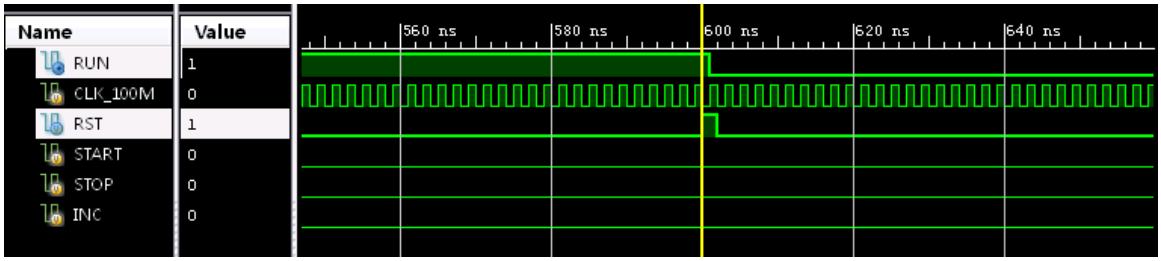
Released INC:



Click START:



Click RST:



2.3 Synthesization, Implementation and Program Generation

The implementation constraints file is as followed:

```
// Clock
NET "CLK_100M" CLOCK_DEDICATED_ROUTE = FALSE;
NET "CLK_100M" LOC = "V10"; // 100MHz Clock

// 7-seg display
NET "SEG[1]" LOC = "T17"; // CA
NET "SEG[2]" LOC = "T18"; // CB
NET "SEG[3]" LOC = "U17"; // CC
NET "SEG[4]" LOC = "U18"; // CD
NET "SEG[5]" LOC = "M14"; // CE
NET "SEG[6]" LOC = "N14"; // CF
NET "SEG[7]" LOC = "L14"; // CG
NET "SEG[0]" LOC = "M13"; // DP
NET "AN[0]" LOC = "N16"; // AN0
NET "AN[1]" LOC = "N15"; // AN1
NET "AN[2]" LOC = "P18"; // AN2
NET "AN[3]" LOC = "P17"; // AN3

// Control buttons
NET "START" LOC = "B8"; // BTN Central
NET "RST" LOC = "C9"; // BTND
NET "STOP" LOC = "C4"; // BTNL
NET "INC" LOC = "D9"; // BTNR

// State
NET "RUN" LOC = "T11"; // Left most LED
```

The `V10` pin is a 100MHz user clock of the Nexys 3 board. The clock pulse will be counted in the `run` module, and only after 5 seconds will the FSM be refreshed.

The RTL diagram of the core module, the `Timer` module, generated from Synthesization is as below:

(See Appendix 2: RTL Diagrams)

2.5 Design Summary

Device utilization summary:

```
Slice Logic Utilization:
 Number of Slice Registers:              39  out of  18224     0%
 Number of Slice LUTs:                   96  out of   9112     1%
    Number used as Logic:                96  out of   9112     1%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     96
   Number with an unused Flip Flop:      57  out of     96    59%
   Number with an unused LUT:             0  out of     96     0%
   Number of fully used LUT-FF pairs:    39  out of     96    40%
   Number of unique control sets:         3

IO Utilization:
 Number of IOs:                          18
 Number of bonded IOBs:                  18  out of    232     7%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               1  out of     16     6%
```

Cross Clock Domains Report:

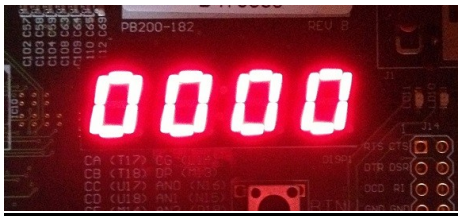
```
Clock to Setup on destination clock CLK_100M
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
CLK_100M       |    4.503|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock cclk_16
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
CLK_100M       |    3.418|         |         |         |
cclk_16        |    4.660|         |         |         |
---------------+---------+---------+---------+---------+
```

Power Report:

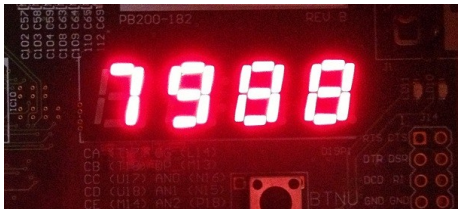
```
--------------------------------------------------------------------------------
|                            On-Chip Power Summary                             |
--------------------------------------------------------------------------------
|        On-Chip        | Power (mW) |  Used  | Available | Utilization (%) |
--------------------------------------------------------------------------------
| Clocks                |       0.01 |      2 |    ---    |       ---       |
| Logic                 |       0.00 |     88 |      9112 |               1 |
| Signals               |       0.00 |    118 |    ---    |       ---       |
| IOs                   |       0.00 |     18 |       232 |               8 |
| Quiescent             |      14.84 |        |           |                 |
| Total                 |      14.86 |        |           |                 |
--------------------------------------------------------------------------------
```

2.6 Online Testing

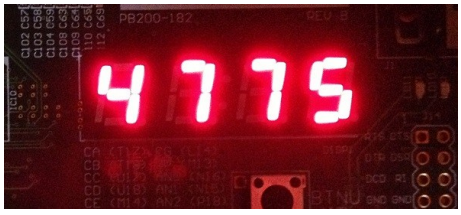
Initial state:



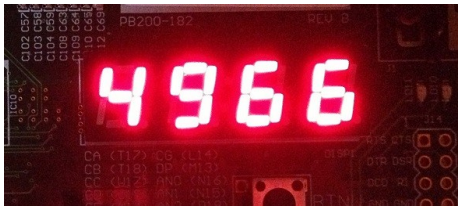
Click START:



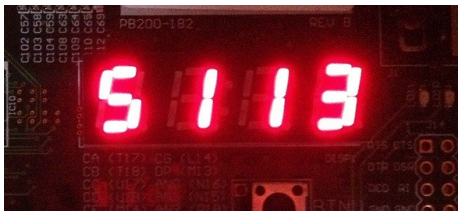
Click STOP:



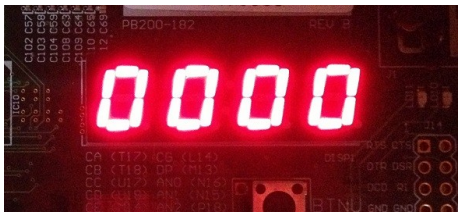
INC:



INC:



Click RST:



Afterthought

A lot of difficulties have been met at this experiment. Actually this experiment and the last one cost more than three weeks of my time to do, and yet there are still many questions with no solutions of mine.

One of the strangest things I met is the register. When I used the a register called `running` to remember running state of the timer without connecting it to the outside as a LED signal, the bit it stored turned to be zero immediately as I released my finger. And if I make it an output signal to the first LED, the bit could be held on the register after I released my finger from the START button.

What's more, I wrote the codes as followed two segments:

```
always@(posedge START, posedge INC) begin …
```

or

```
always@(posedge START, posedge STOP) begin …
```

with the `INC` and `STOP` doing exactly the same thing (turning the running state to zero) at the time, the program run well. But as I put them together as followed:

```
always@(posedge START, posedge STOP, posedge INC) begin …
```

the program just run strangely, and the function of it was quite wrong.

And I do not find any reason yet, so instead, I now use the clock signal as the driver, and continuously checking the value of `START`, `STOP` and `INC` signals.

As the experiments get into timing part, lots of things have become too strange to understand, and many of those strange things are like being bugs of the ISE software, as far as I consider. I think so because I cannot tell any reason so far to explain those things, and I found lots of bug reports to the Xilinx website. However, I think most of these things happened due to my poor understanding of the Verilog language, and there are a lot of questions to ask the TA or the teacher.

David Qiu (邱迪聪)

2013.12.4

Appendix 1: Attachment List

1. Timer

    \ HEX_7seg.v

    \ Timer.v

    \ Timer_test.v

    \ Timer_ctr.ucf

X.v files are primary code files; X_ctr.ucf files are implementation constraints files; X_test.v files are test files.

Appendix 2: RTL Diagrams