嵌入式系统结构与设计课程实验

实验报告 7

Traffic Controller with Parade Mode and Sola Machine Dispenser

邱迪聪 / 11331262

2013/11/20

Content

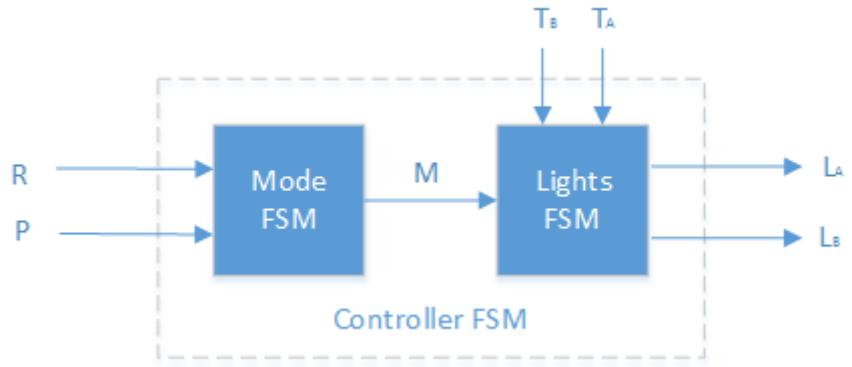
1. Target

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

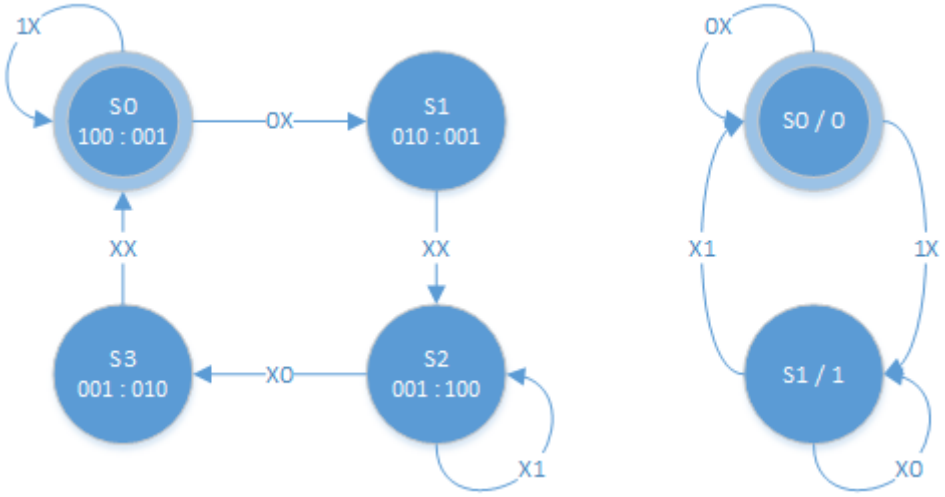
2. Traffic Controller with Parade Mode

2.1 Designing

The traffic controller with parade mode is designed to consist of two parts, the Mode FSM and the Lights FSM, as shown followed.



The detailed FSM designs are shown below, with the left diagram being the Lights FSM, and the right one being the Mode FSM.

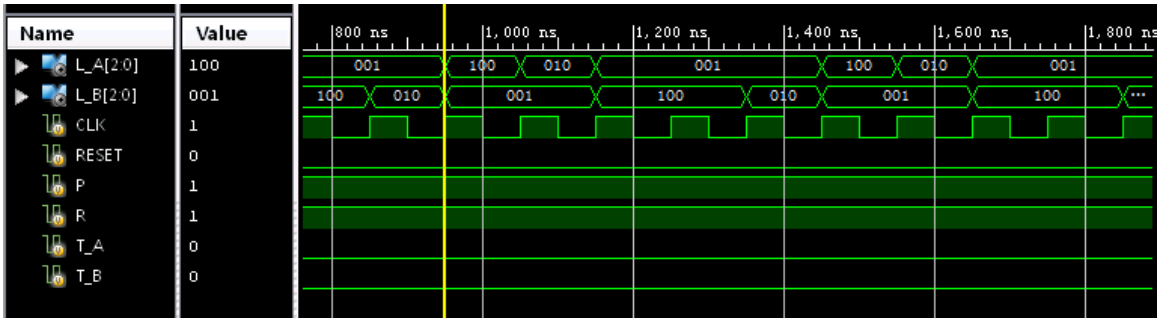


The Mode FSM is designed for remembering the parade state, which will send a signal M to the Lights FSM indicating the current parade state. The Mode FSM will be triggered for every full round of the Lights FSM. The Lights FSM is designed for traffic lights' control. The internal control of the FSM's uses Moore FSM design method.

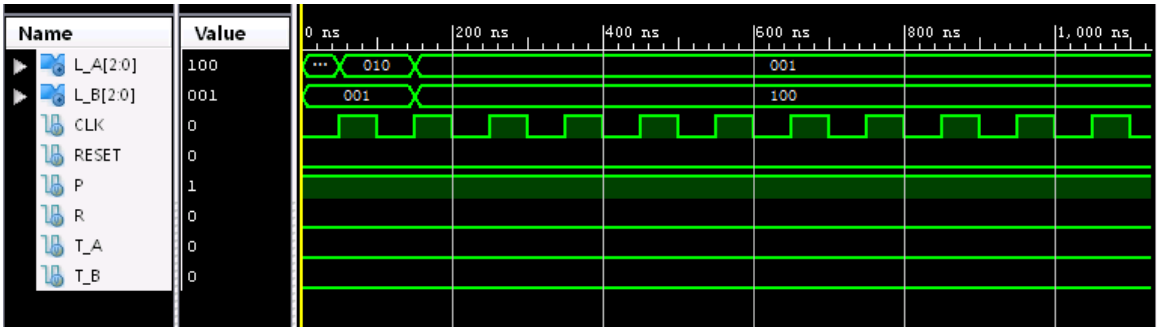
2.2 Simulation

Three simulations are designed for different cases, and the results are shown as below.

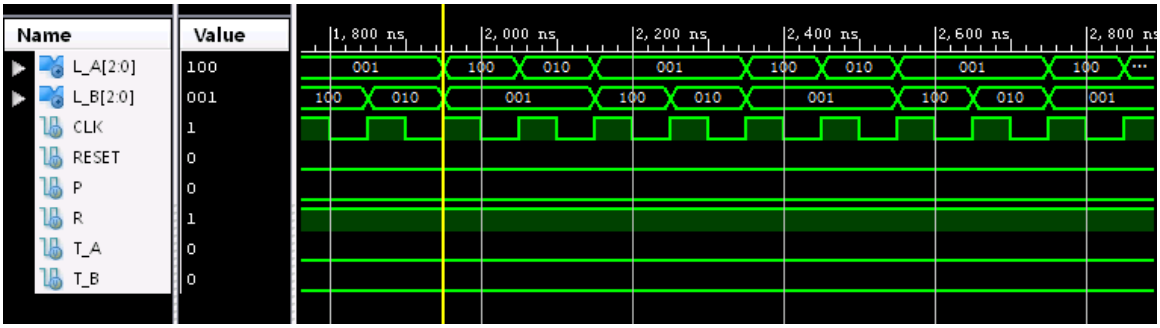
P=1, R=1:



P=1, R=0:



P=0, R=1:



2.3 Synthesization, Implementation and Program Generation

The implementation constraints file is as followed:

```
// Input: T_A, T_B, P, R
NET "T_A" LOC = "T5"; // left most button
NET "T_B" LOC = "V8";
NET "P" LOC = "U8";
NET "R" LOC = "N8";

// Input: RESET
NET "RESET" LOC = "B8"; // central button

// Clock: CLK_100M
NET "CLK_100M" LOC = "V10"; // 100MHz clock

// Output: L_A
NET "L_A[2]" LOC = "T11"; // left most LED
NET "L_A[1]" LOC = "R11";
NET "L_A[0]" LOC = "N11";

// Output: L_B
NET "L_B[2]" LOC = "M11";
NET "L_B[1]" LOC = "V15";
NET "L_B[0]" LOC = "U15";
```

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 `run` module generated from Synthesization is as below:

(See Appendix 2: RTL Diagrams)

Below is the RTL diagram of the core control module, the `TC_Parade` module.

(See Appendix 2: RTL Diagrams)

2.5 Design Summary

Device utilization summary:

```
Slice Logic Utilization:
 Number of Slice Registers:                 4  out of  18224     0%
 Number of Slice LUTs:                     49  out of   9112     0%
    Number used as Logic:                  49  out of   9112     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:       49
   Number with an unused Flip Flop:        45  out of     49    91%
   Number with an unused LUT:               0  out of     49     0%
   Number of fully used LUT-FF pairs:       4  out of     49     8%
   Number of unique control sets:           2

IO Utilization:
 Number of IOs:                            12
 Number of bonded IOBs:                    10  out of    232     4%
```

Cross Clock Domains Report:

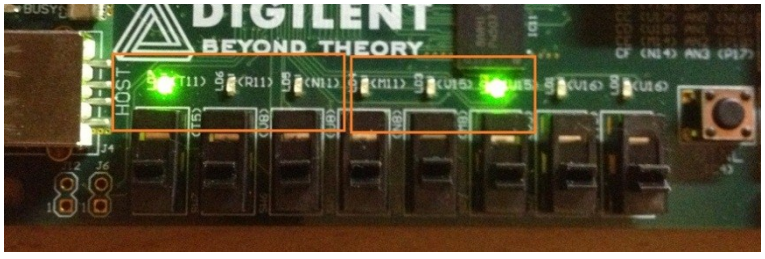
```
Clock to Setup on destination clock CLK
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
CLK            |    1.731|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock count[31]_GND_1_o_equal_2_o<31>7_f7
----------------------------------+---------+---------+---------+---------+
                                  | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock                      |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
----------------------------------+---------+---------+---------+---------+
count[31]_GND_1_o_equal_2_o<31>7_f7|         |         |    2.133|         |
----------------------------------+---------+---------+---------+---------+
```

Power Report:

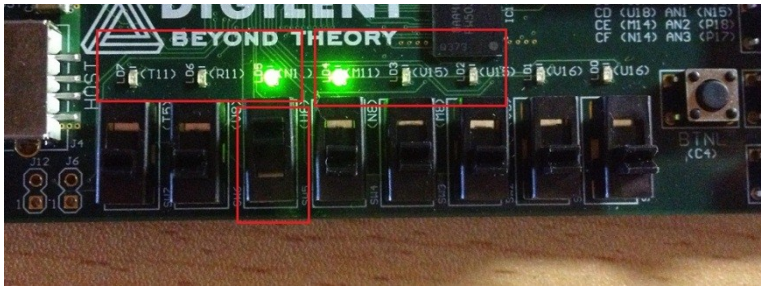
```
------------------------------------------------------------------------------
|                           On-Chip Power Summary                            |
------------------------------------------------------------------------------
|        On-Chip        | Power (mW) |  Used  | Available | Utilization (%) |
------------------------------------------------------------------------------
| Clocks                |       0.00 |      2 |    ---    |       ---       |
| Logic                 |       0.00 |     47 |      9112 |               1 |
| Signals               |       0.00 |     58 |    ---    |       ---       |
| IOs                   |       0.00 |     12 |       232 |               5 |
| Quiescent             |      14.84 |        |           |                 |
| Total                 |      14.84 |        |           |                 |
------------------------------------------------------------------------------
```

2.6 Online Testing

No traffic jam or parade:

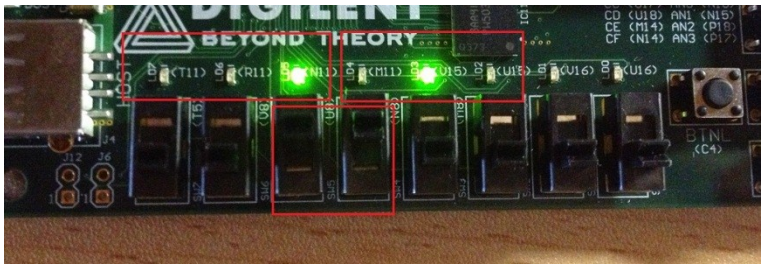


P=1, R=0:



The traffic lights always keep this state, and have no transition.

P=1, R=1:



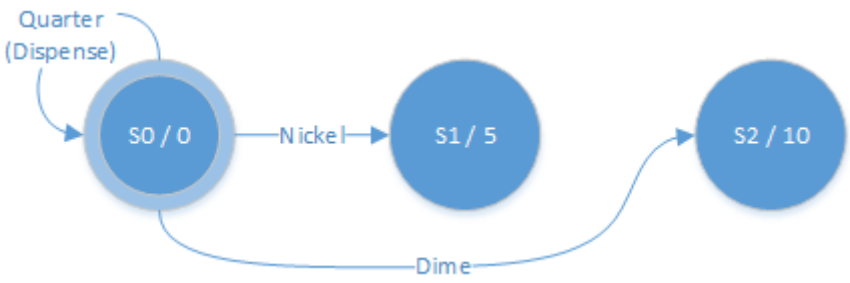
After the previous state, the R button is switched to 1, and after one more round, the traffic lights start to change their state. The whole loop is the same as that of no traffic jam and no parade, but the state **[L_B=Green; L_A=Red]** stays one more round (of totally 10 seconds) per loop.

3. Soda Machine Dispenser

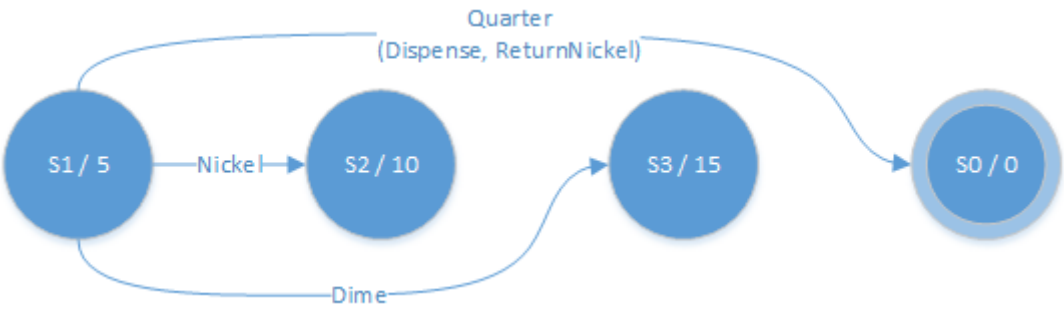
3.1 Designing

The state transition diagrams are as followed. Note that each of the diagrams is related to one current state, and the transitions of that are the actions related the specific input.

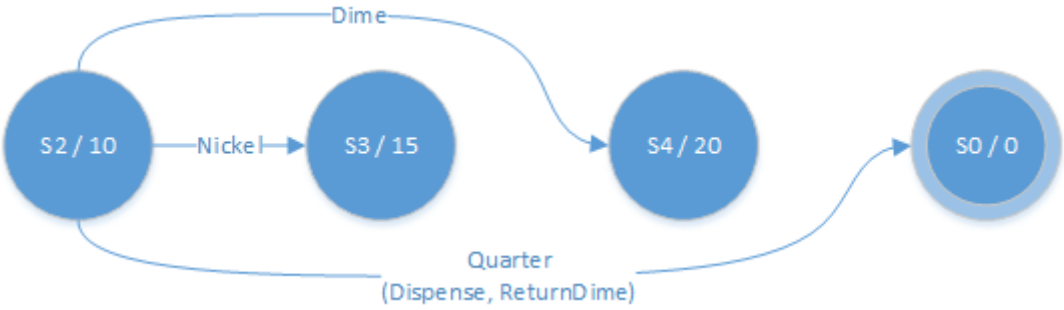
Current State: S0



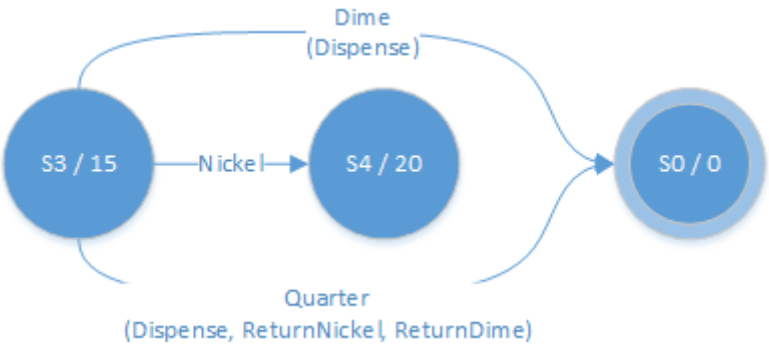
Current State: S1



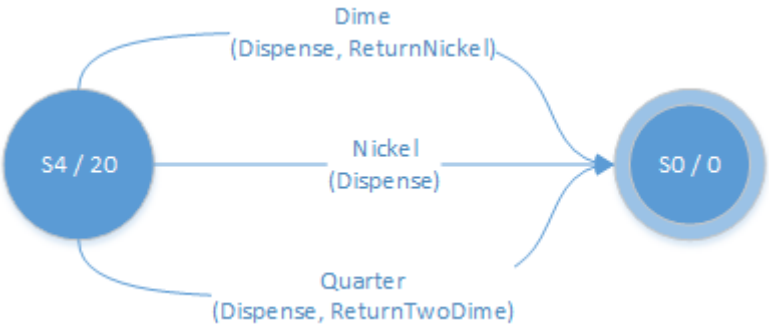
Current State: S2



Current State: S3



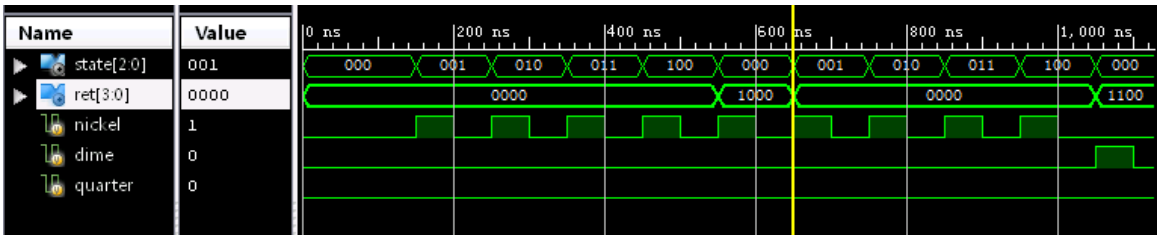
Current State: S4



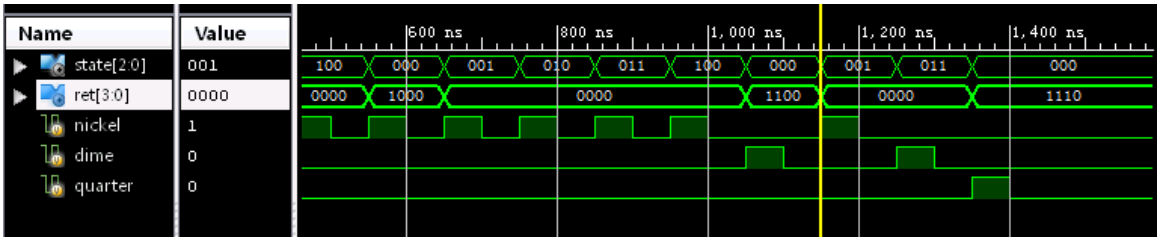
The design applies both Moore and Mealy FSM methods, so as to optimize itself. The Mealy output without specification is “Clear”, which means clearing the output of the dispensing output of the machine. And the Moore output corresponds to the money put into the machine.

3.2 Simulation

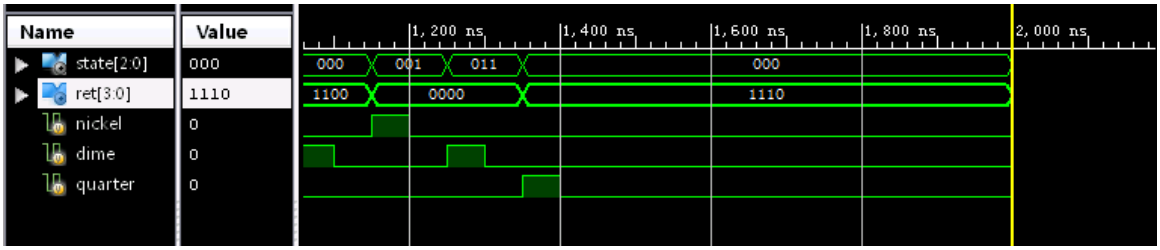
Put in 5 nickels:



Put in 4 nickels followed with one dime:



Put in 1 nickel and 1 dime, followed with 1 quarter:



3.3 Synthesization, Implementation and Program Generation

The implementation constraints file is as followed:

```
// Clock
NET "CLK" CLOCK_DEDICATED_ROUTE = FALSE;
NET "CLK" LOC = "V10";

// 7-Segment 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

// Inputs
NET "nickel" LOC = "C4"; // BTNL
NET "dime" LOC = "B8"; // BTN Central
NET "quarter" LOC = "D9"; // NTNR

// Return values
NET "ret[3]" LOC = "T11"; // Left most LED
NET "ret[2]" LOC = "R11";
NET "ret[1]" LOC = "N11";
NET "ret[0]" LOC = "M11";
```

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.

Below is the RTL diagram of the core control module, the `SDM` module.

(See Appendix 2: RTL Diagrams)

The following steps were successfully gone through and a .bit file was generated successfully.

3.4 Design Summary

Device utilization summary:

```
Slice Logic Utilization:
 Number of Slice Registers:                3  out of  18224     0%
 Number of Slice LUTs:                    12  out of   9112     0%
    Number used as Logic:                 12  out of   9112     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:      12
   Number with an unused Flip Flop:        9  out of     12    75%
   Number with an unused LUT:              0  out of     12     0%
   Number of fully used LUT-FF pairs:      3  out of     12    25%
   Number of unique control sets:          2

IO Utilization:
 Number of IOs:                           10
 Number of bonded IOBs:                   10  out of    232     4%
    IOB Flip Flops/Latches:                4
```

Cross Clock Domains Report:

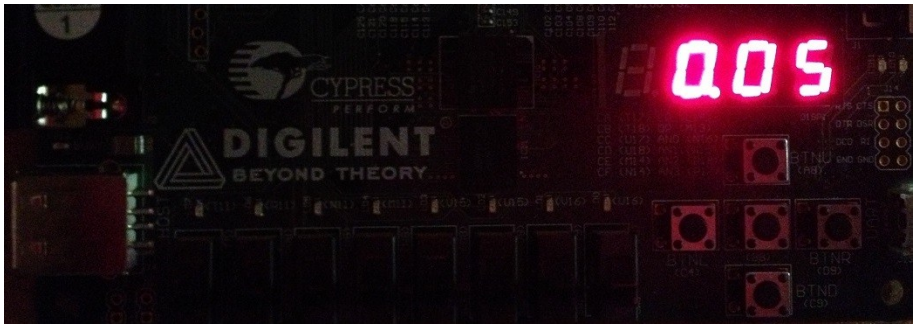
```
Clock to Setup on destination clock dime_nickel_OR_10_o
-------------------+---------+---------+---------+---------+
                   | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock       |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
-------------------+---------+---------+---------+---------+
dime_nickel_OR_10_o|         |         |    1.825|         |
-------------------+---------+---------+---------+---------+

Clock to Setup on destination clock quarter_dime_OR_14_o
-------------------+---------+---------+---------+---------+
                   | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock       |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
-------------------+---------+---------+---------+---------+
dime_nickel_OR_10_o|         |         |    3.317|         |
-------------------+---------+---------+---------+---------+
```

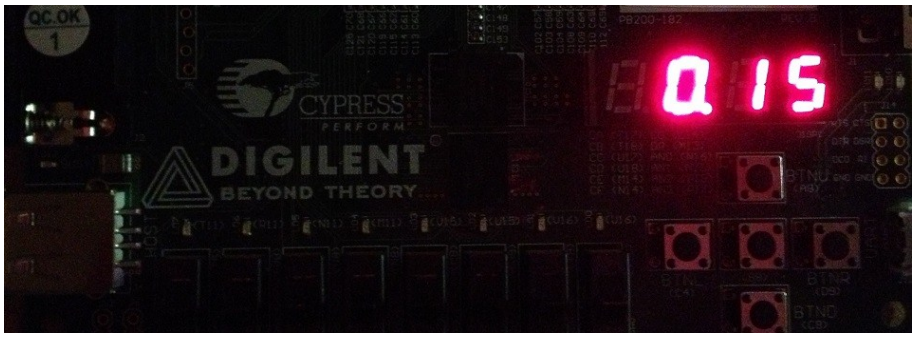
Power Report:

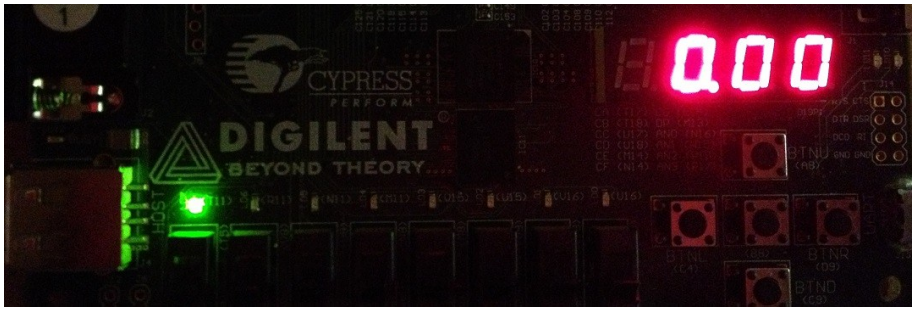
```
------------------------------------------------------------------------------
|                          On-Chip Power Summary                             |
------------------------------------------------------------------------------
|        On-Chip        | Power (mW) |  Used  | Available | Utilization (%) |
------------------------------------------------------------------------------
| Clocks                |       0.00 |      2 |    ---    |       ---       |
| Logic                 |       0.00 |      8 |      9112 |               0 |
| Signals               |       0.00 |     14 |    ---    |       ---       |
| IOs                   |       0.00 |      7 |       232 |               3 |
| Quiescent             |      14.84 |        |           |                 |
| Total                 |      14.84 |        |           |                 |
------------------------------------------------------------------------------
```

3.5 Online Testing

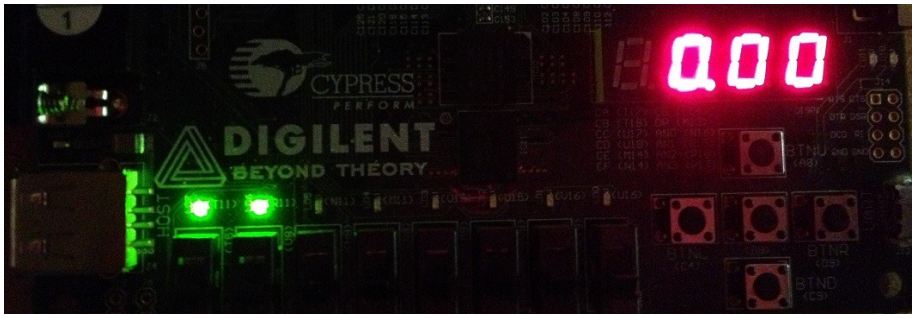
Nickel + Dime + Dime:



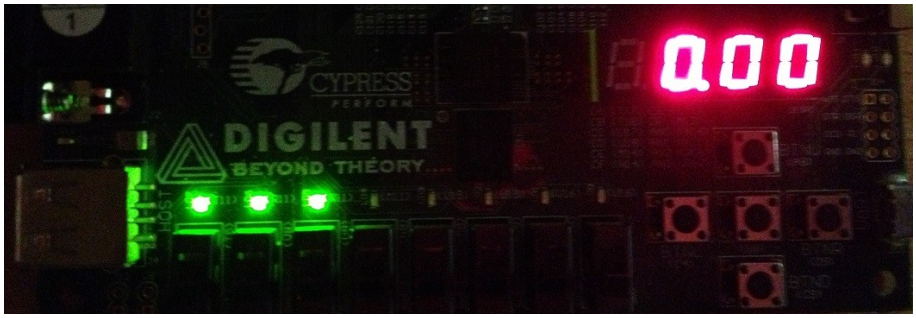




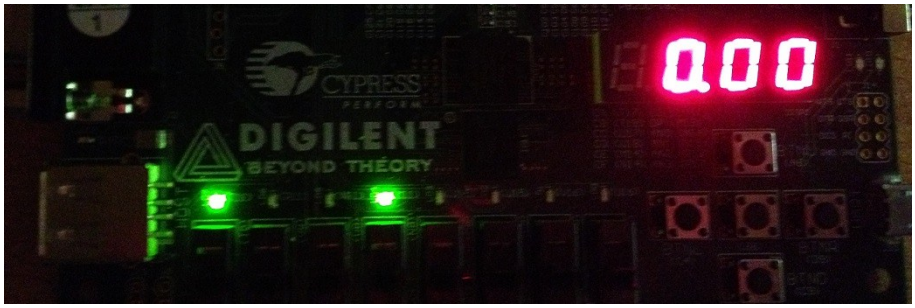
Dime + Dime + Dime:



Nickel + Dime + Quarter:



Dime + Dime + Quarter:



Afterthought

This experiment lasted three weeks before I came out with a proper solution.

The initial design used asynchronous input of the three kinds of coins. But it turned out to be failure, when three input signal put together in one always block. No matter how I changed the design, as more than two input signal gathered in the trigger condition of an always block, the on-board testing of putting coin into the Soda Machine Dispenser became continuous putting the same coin a lot of times, when the corresponding button was clicked. In other words, the positive edge trigger of coin input came out to be unknown clock trigger.

The situation is too wired to meet before this experiment, and the trick played in asynchronous-reset D flip-flop experiment seemed disabling.

Finally, the idea of using clock trigger came to my mind. A clock signal is used as the only triggered condition of an always block, and any coin input is detected inside the block. Therefore, extra registers are necessary for remembering the coin input state.

The clock signal is always scanning. If coin-input-state register is empty and any coin signal is HIGH, the register will record the input. If the register is not empty and all coin signals are LOW, we know that a coin in put into the machine successfully. At this moment, the FSM will go to the corresponding state about how much money has been put into the machine.

The last way works, but a lot of questions remain. So I am going to as the TA or teacher during the course.

David Qiu (邱迪聪)

2013.12.4

Appendix 1: Attachment List

1. TC_Parade

    \ run.v

    \ run.ucf

    \ TC_Parade.v

    \ TC_Parade_test.v

    \ TC_Parade_test2.v

    \ TC_Parade_test3.v

2. SMD

    \ SMD.v

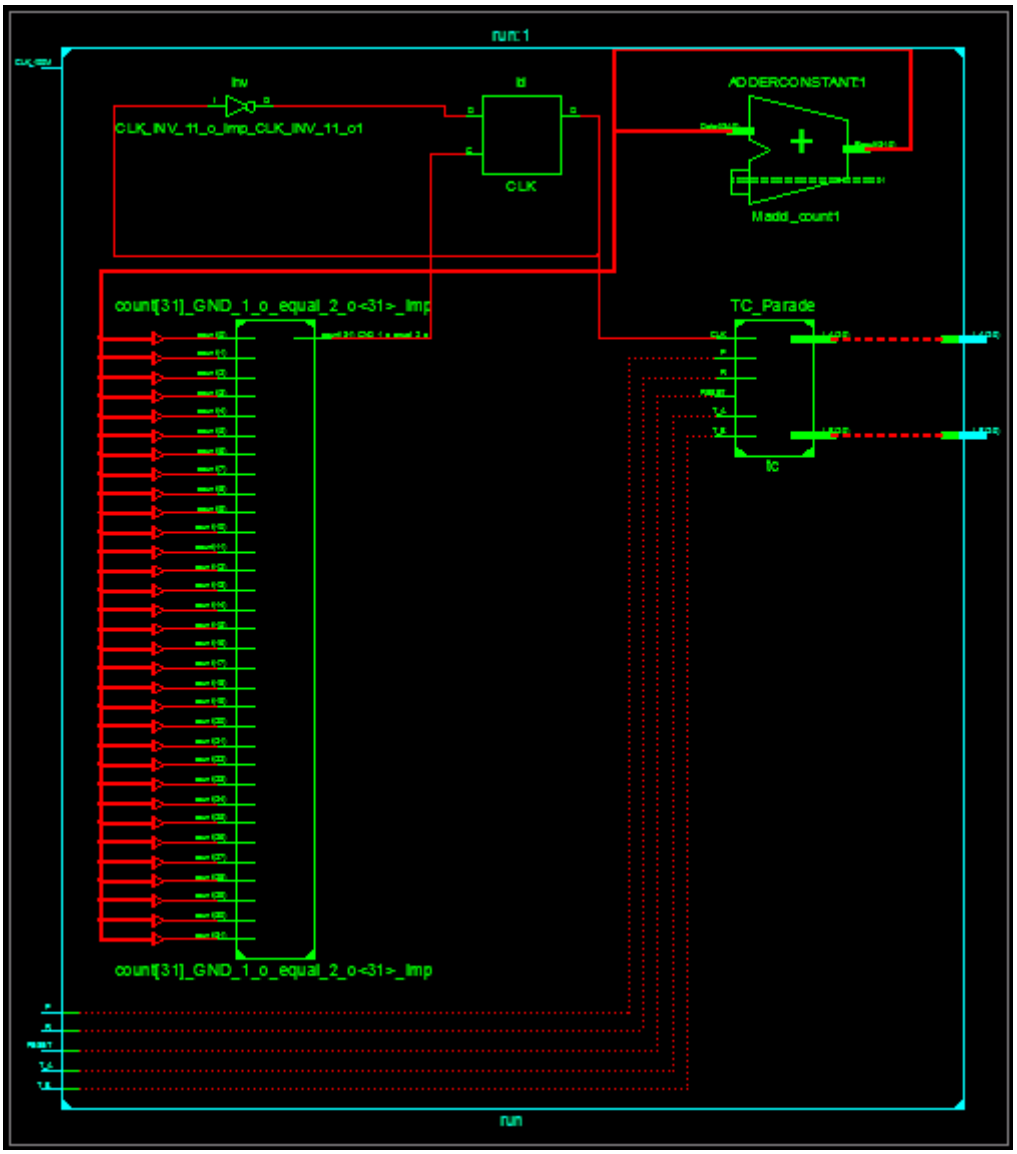
    \ SMD_ctr.ucf

    \ SMD_test.v

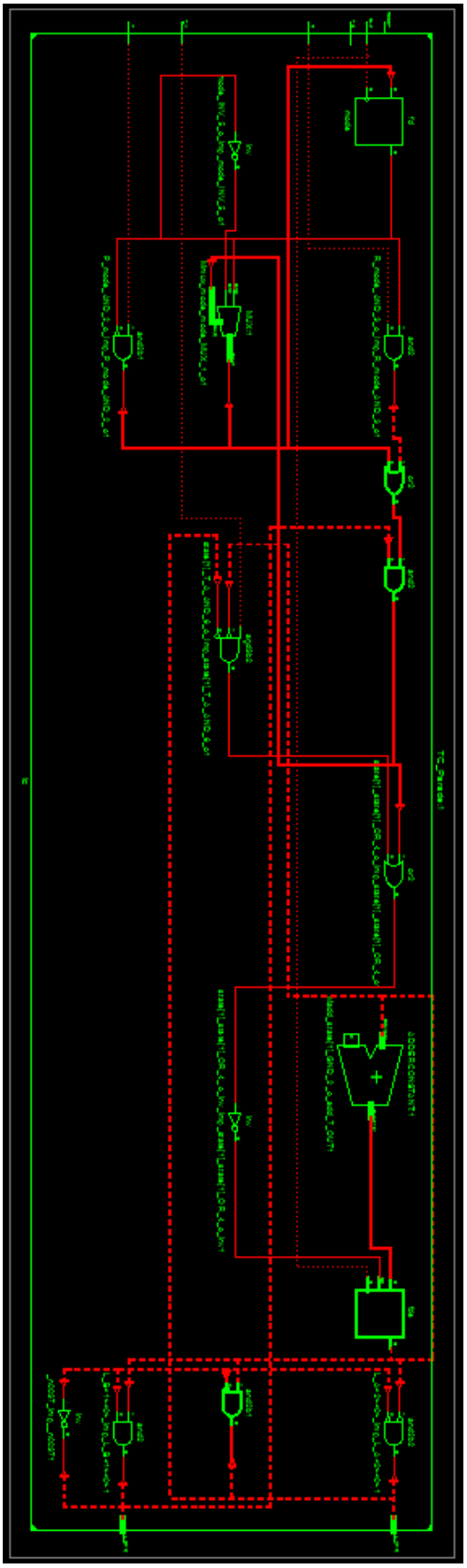
X.v files are primary code files; X_ctr.ucf files are implementation constraints files; X_test.v files are test files.

Appendix 2: RTL Diagrams

1. Traffic Controller with Parade Mode, `run` module:



2. Traffic Controller with Parade Mode, core controller module:



3. Soda Machine Dispense, core module: