Embedded System Design
Project 2

Max Thrun — Ian Cathey — Mark Labbato

October 8, 2012

Abstract

The purpose of this project was to implement a system which kept a running average of randomly generated numbers using fixed-point arithmetic. The challenge was put forth to make the final implementation as small and as fast as possible.

Implementation/Design

We decided to take a completely module approach and break down each function into its own module. This allowed cleaner and easier testing as well as simple integration at the top level. Each module was designed to be a stage in the system pipeline. By using this synchronous pipelined approach we are able to ensure short propagation delays between registered stages which helps to increase the maximum frequency our design can run at. The LFSR outputs a random 8-bit value every clock cycle. The divider module assumes this value to be a signed fixed point number with 6 fractional bits. Every clock cycle it multiplies this value by the closest binary fractional equivalent to .. The sum module is the next stage in the pipeline and on each clock cycle shift in the result of the divider assuming a signed value. It keeps the last three divided numbers and continuously outputs their sum. At the top level the sum is assigned to the first 8 red LEDs. Simultaneous to the calculation pipeline is the counter. When the system is enabled (indicated by a top level flag .running.) the counter increments in scientific notation using 2 digits for the base and 1 digit for the exponent. This gives us a maximum count of 99E9 which, given a 50MHz clock, takes about 30 minutes before it will overflow. When the running flag is toggled off (by pressing KEY3) the system is paused and the current count value is held on the seven segment displays and the current sum is displayed on the red LEDs. When system state is toggled back into .running. mode the count is reset.

Testing strategy

For this lab we decided to test everything exhaustively. Each person wrote a test bench for the module that they wrote. Other team members then went through each test bench and validated the code, then tested the module themselves. Python scripts were developed to verify the full output of each test bench. For the LFSR a gnuplot script plots a histogram of a few thousand values. This allows us to visually verify the distribution of numbers and ensure it is flat. After each test was both spot checked by hand and automatically checked by a Python script we wrote a master top level test bench. Again, a Python script was written to fully verify the top level output. We found that this strategy of using Python to fully verify each module allowed us to quickly find corner case errors that would have been hard to find otherwise.

Implementation Resource Usage

Using an 8-bit LFSR our resource summary is as follows:

```
Fitter Status : Successful - Mon Oct  8 15:39:01 2012
Quartus II 32-bit Version : 12.0 Build 178 05/31/2012 SJ Web Edition
Revision Name : top
```

```
Top-level Entity Name : top
Family : Cyclone II
Device : EP2C20F484C7
Timing Models : Final
Total logic elements : 399 / 18,752 ( 2 % )
    Total combinational functions : 383 / 18,752 ( 2 % )
    Dedicated logic registers : 110 / 18,752 ( < 1 % )
Total registers : 110
Total pins : 61 / 315 ( 19 % )
Total virtual pins : 0
Total memory bits : 0 / 239,616 ( 0 % )
Embedded Multiplier 9-bit elements : 0 / 52 ( 0 % )
Total PLLs : 0 / 4 ( 0 % )
```

An attempt was made to maximize the size of the LFSR, registers, multipliers, and adders. Unfortunately we were unable to actually find the upper limit of our device due to a few issues. Firstly we were unable to find taps for an LFSR greater than 168 bits. Interestingly when we did implement the 168 bit LFSR the general logic usage actually went down because the optimizer started using on chip RAM. This optimization prevented us from extrapolating the maximum size as the resource usage was unpredictable. We also attempted to generate large random numbers (on the order of a 1000 bits) by using multiple LFSRs each generating 1 bit of the random number. After simulating this and plotting the number frequency histogram we found that our distribution was completely non uniform and was missing values. This makes sense as all of our LFSRs were the same width thus having the same period. Even though they were each seeded differently the fact that they have the same period prevents the output from completely covering all possible numbers. One solution to this problem would be to use a different size LFSRs for each bit but this bring us back to our original problem of not being able to make an LFSR greater than 168 bits.

Implementation Speed and Power

The max clock speed as reported by the timing analyzer is 102.51MHz and our Total Thermal Power Dissipation is 81.98mW.

```
+-----------------------------------------------+
; Slow Model Fmax Summary                       ;
+------------+----------------+------------+------+
; Fmax       ; Restricted Fmax ; Clock Name ; Note ;
+------------+----------------+------------+------+
; 102.51 MHz ; 102.51 MHz      ; CLOCK_50   ;      ;
+------------+----------------+------------+------+
```

Our critical path is through the huge 30-bit counter that we use to count in scientific notation.

```
+-------------------------------------------------------------------------------------------
; Fast Model Setup: 'CLOCK_50'
+--------+-------------------------------------+-------------------------------------+--------
; Slack  ; From Node                           ; To Node                             ; Launch
+--------+-------------------------------------+-------------------------------------+--------
; 16.161 ; counter:counter|Add0~4_OTERM117     ; counter:counter|Add0~58_OTERM63     ; CLOCK_5
; 16.196 ; counter:counter|Add0~4_OTERM117     ; counter:counter|Add0~56_OTERM65     ; CLOCK_5
; 16.224 ; counter:counter|Add0~20_OTERM101    ; counter:counter|Add0~58_OTERM63     ; CLOCK_5
...
```

Team Member Contributions

Ian Cathey - LFSR module, Seven Segment Display driver, LFSR testbench, attempted to write a divider but Max.s was better Mark Labbato - Top level design module, summation module, binary-to-BCD converter, top level testbench, module testbenches. Max Thrun - Divider, and counter circuits, testbenches for these modules

All members participated in verifying each others module in a group setting.