This blog covers another self-study FPGA project. Project #2 used the Digilent Basys 3 FPGA development board along with Vivado. That board has an Artix 7 Xilinx chip, specifically the xc7a35tcpg236-1 part. The tools used in that project were Vivado 2022.2 with the WebPACK license. *Project #3 branches out in two main areas described in the Introduction.*

Introduction

Project #3 adds a UART module to display the pseudo-random numbers numbers on a Putty terminal. To do this I wanted to also learn another development environment so I choose the Altera Cyclone IV FPGA and developed the RTL using Quartus Prime Lite 20.1. Both sets of tools are free downloads from Xilinx and Intel.

Project Approach

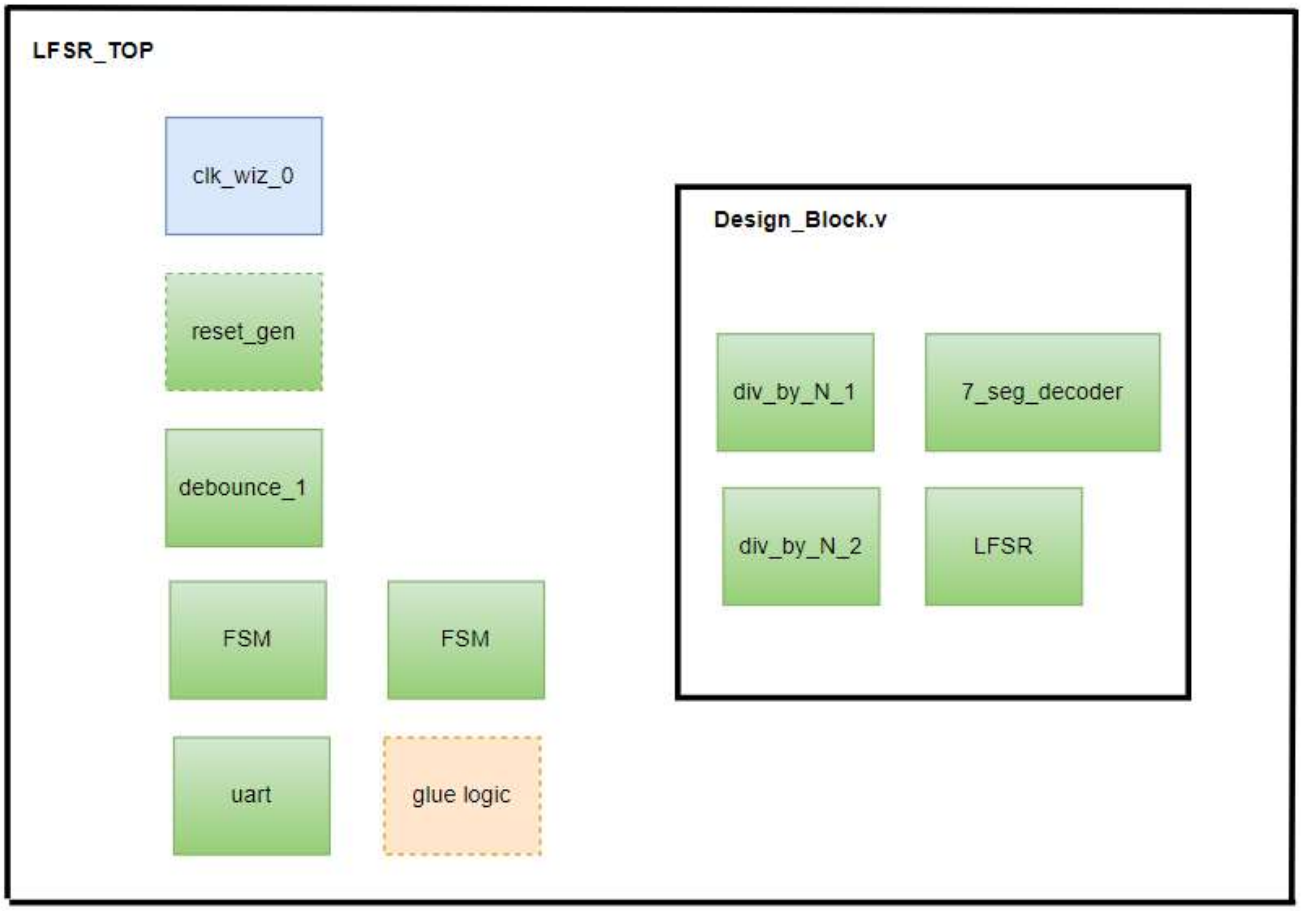
The project approach was to take the RTL modules from *Vivado* project #2 and create a project in *Quartus*. Since *Quartus* does not have a Block Design feature I created a new module, '*design_block.v*', and put the modules that were in the Vivado Block Design into that module. The project diagram is shown in figure 3 below. The only IP used was the clock generator. This is unique between Vivado and Quartus. Also, for Quartus I assumed a Cyclone IV chip on a Deo Nano board. This means the incoming clock was 100MHz for Vivado and 50MHz for Quartus. Other than these changes, and the pin-outs, the RTL code are the same.

Areas of Learning

The areas of learning in this project were:

- A new development environment, *Quartus*.
- Designing a Uart in RTL.
- Designing a FIFO buffer in RTL.
- Designing multiple FSM in RTL.

Figure 1 – FPGA Project #3



What the Project Does

The project displays a pseudo-random sequence on the 4-digit 7-segment display that is updated every one second. The pseudo-random sequence is generated by a linear feedback shift register (LFSR). Information on this topic is available https://en.wikipedia.org/wiki/Linear-feedback_shift_register#Fibonacci_LFSRs . In particular the LFSR type used in this project is a Fibonacci configuration that cycles through through 2^m -1 patterns before repeating, and never outputs 0.

The details of *Project #1* are in the tutorial here: https://github.com/Btremaine/Basys_3_LFSR.

Project #3 Additions

Project #3 modified the design to instantiate the *uart_tx.v* RTL module at the top level and add the IO needed for tx.

The additional functions needed include that at every update of the LFSR value the 16-bit hex word needs to be converted to four ASCII bytes and those bytes need to have a '\n' appended. the five ASCII bytes which are then transmitted. The uart then remains inactive until the data valid is asserted at the next update of the LFSR. A state machine is used to count the four hex nibbles and do the translation to

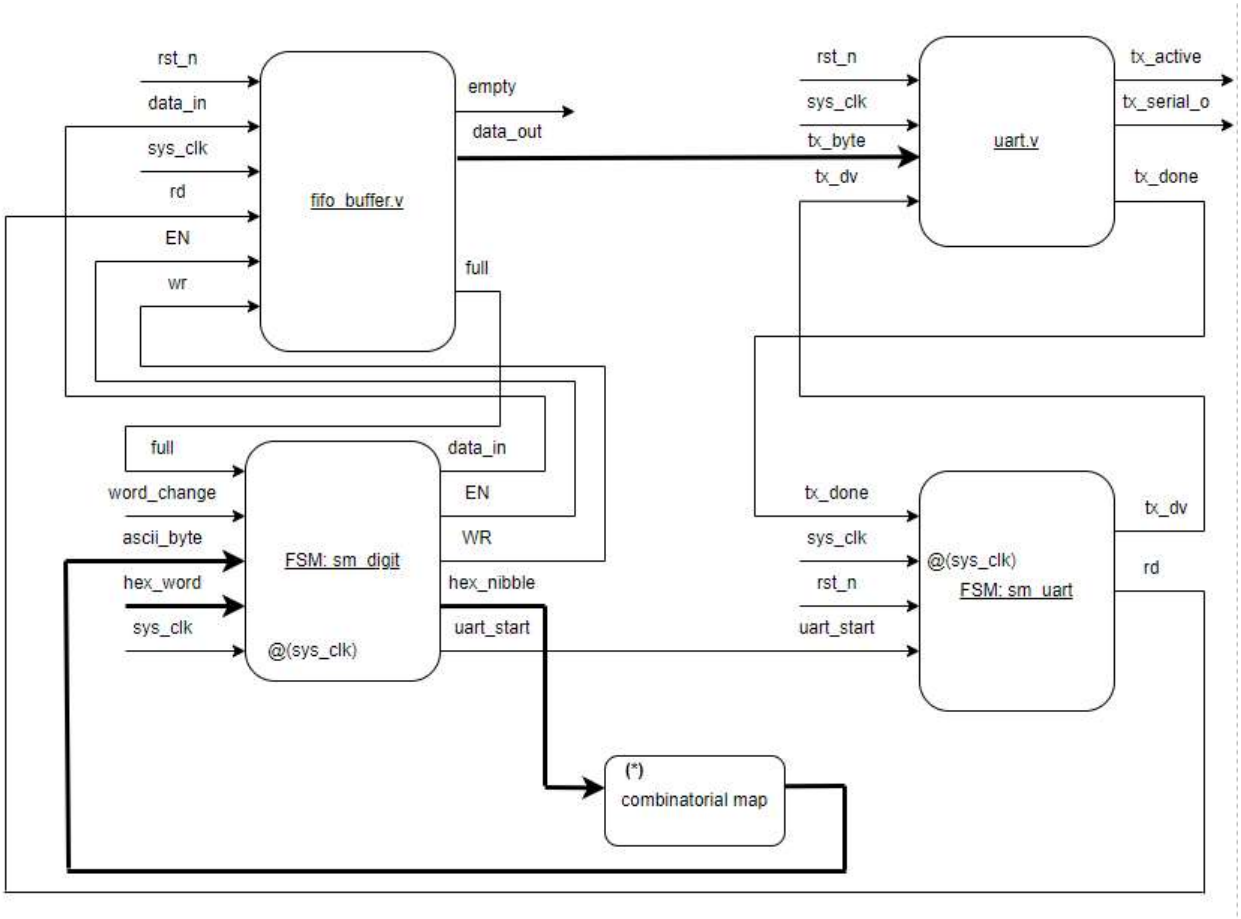
ASCII then add the '\n' character. The terminal display should show one 16-bit hex display per line with a line return. As example, when the 7-segment displays "**f76e**", the uart should transmit {'66', '37' '36', '65', '0a'}. Putty is set-up to operate with an implied CR.

The uart uses the existing 5MHz clock from project 2 and configures the uart timers for a 9600 baud transmission. The uart is configured as 8-bit, start & stop bit and no parity bit.

Several finite state machines (FSM) were used in this design. The UART is implemented as a 4-state FSM. The RTL that writes the FIFO buffer was implemented as a 6-state FSM and a separate 6-state FSM was used to transfer the FIFO buffer contents to the UART. All three FSM use a *1-always* structure. The conversion of a 4-bit hex nibble to 8-bit ASCII byte is done with an asynchronous combinatorial *always* block.

A flow diagram of the logic comprising just the UART and finite state machines is shown figure 2 below. This is at the top level module and does not include the modules for the LFSR or Seven-segment hex display.

Figure 2 – Flow Diagram for modules added to this project.



I initially set up a *Quartus* project and used the RTL modules from project #2. The development, initial simulation and debug were done in *Quartus*. I don't have an Altera development board, so for demo the RTL was moved back to a *Vivado* project and built for the Basys 3 board.

Comparing Quartus and Vivado

For simulation, *Quartus* comes bundled with ModelSim (Intel FPGA Starter Edition 2020.1). There was a learning curve in using this package but in the end I like it better than than the simulation built into *Vivado*. When running simulation in *Quartus*, *ModelSim* is launched as a separate program. The full power of *ModelSim* is available.

Also, *Quartus* has a feature in which it will diagram all FSM in the project. This was especially helpful in debugging. Similar to *Vivado*, *Quartus* can also display a schematic of RTL modules.

Static Timing

The static timing was studied after the synthesis run in both *Vivado* and *Quartus*.

The timing summary after the modification is shown in Figure 3 for project #3.

Figure 3 – Post Implementation Static Timing Report (a) Vivado (b) Quartus

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	194.308 ns	Worst Hold Slack (WHS):	0.086 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	386	Total Number of Endpoints:	386	Total Number of Endpoints:	194

All user specified timing constraints are met.

```
        Found TIMING_ANALYZER_REPORT_SCRIPT_INCLUDE_DEFAULT_ANALYSIS = ON
        Analyzing Slow 1200mV 125C Model
332146 Worst-case setup slack is 194.431
332146 Worst-case hold slack is 0.427
332146 Worst-case recovery slack is 196.301
332146 Worst-case removal slack is 1.884
332146 Worst-case minimum pulse width slack is 9.865
```

The worst-case timing slack was positive in all cases with a 5MHz clock.
The system clock is 5Mhz (200ns period), so the WNS of +194.506ns represents a 5.494ns shift from ideal. There were no Methodology violations reported and no DRC errors reported.

The utilization is very low, as expected for this simple project.

Figure 4 – Utilization (a) Vivado (b) Quartus

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Bonded IOB (106)	BUFGCTRL (32)	MMCME2_ADV (5)
top_lfsr	167	175	1	67	160	7	17	2	1
clk_wiz_0_1 (clk_wiz_0)	0	0	0	0	0	0	0	2	1
debounce_1 (debounce)	20	15	1	6	20	0	0	0	0
design_block_1 (design_block)	82	63	0	30	82	0	0	0	0
fifo_buffer_1 (fifo_buffer)	21	17	0	7	15	6	0	0	0
reset_gen_1 (reset_gen)	16	17	0	13	16	0	0	0	0
uart_tx_1 (uart_tx)	20	18	0	11	20	0	0	0	0

Analysis & Synthesis Resource Utilization by Entity

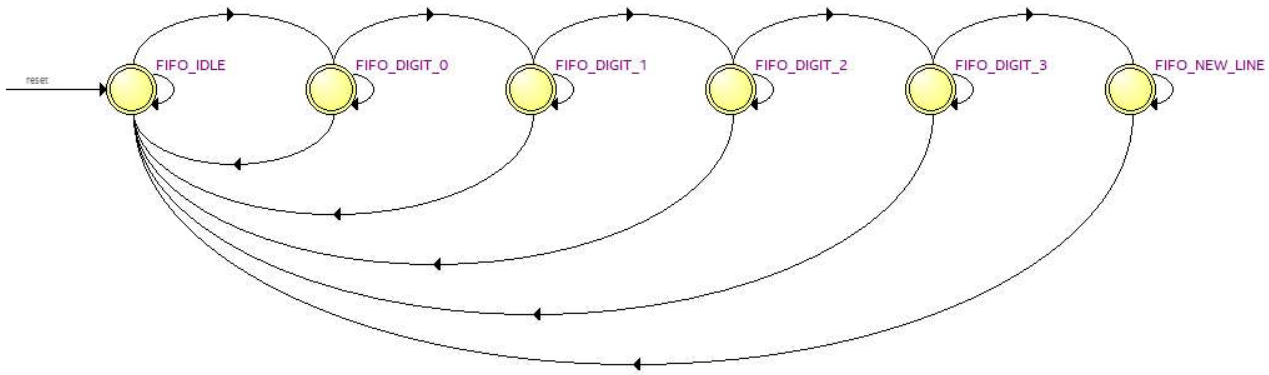
<<Filter>>

Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	F
|top_lfsr	313 (63)	233 (53)	0	0	0	0	17	0	|top_lfsr
|clk_wiz:clk_wiz_1|	0 (0)	0 (0)	0	0	0	0	0	0	|top_lfsr|clk_wiz:clk_wiz_1
|altpll:altpll_component|	0 (0)	0 (0)	0	0	0	0	0	0	|top_lfsr|clk_wiz:clk_wiz_1|altpll:altpll_
|clk_wiz_altpll:auto_generated|	0 (0)	0 (0)	0	0	0	0	0	0	|top_lfsr|clk_wiz:clk_wiz_1|altpll:altpll_
|debounce:debounce_1|	23 (23)	13 (13)	0	0	0	0	0	0	|top_lfsr|debounce:debounce_1
|design_block:design_block_1|	94 (0)	57 (0)	0	0	0	0	0	0	|top_lfsr|design_block:design_block_1
|Seven_Seg_Display_Control:Seven_Seg_Display_Control_1|	21 (21)	4 (4)	0	0	0	0	0	0	|top_lfsr|design_block:design_block_1
|div_by_N:div_by_N_1|	23 (23)	17 (17)	0	0	0	0	0	0	|top_lfsr|design_block:design_block_1
|div_by_N:div_by_N_2|	31 (31)	18 (18)	0	0	0	0	0	0	|top_lfsr|design_block:design_block_1
|lfsr:lfsr_1|	19 (19)	18 (18)	0	0	0	0	0	0	|top_lfsr|design_block:design_block_1
|fifo_buffer:fifo_buffer_1|	69 (69)	72 (72)	0	0	0	0	0	0	|top_lfsr|fifo_buffer:fifo_buffer_1
|reset_gen:reset_gen_1|	25 (25)	17 (17)	0	0	0	0	0	0	|top_lfsr|reset_gen:reset_gen_1
|uart_tx:uart_tx_1|	39 (39)	21 (21)	0	0	0	0	0	0	|top_lfsr|uart_tx:uart_tx_1

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.

This is an example of the FSM viewer in Quartus, accessed through Tools→ Net list Viewers→ FSM

Figure 5 – FSM Viewer from Quartus

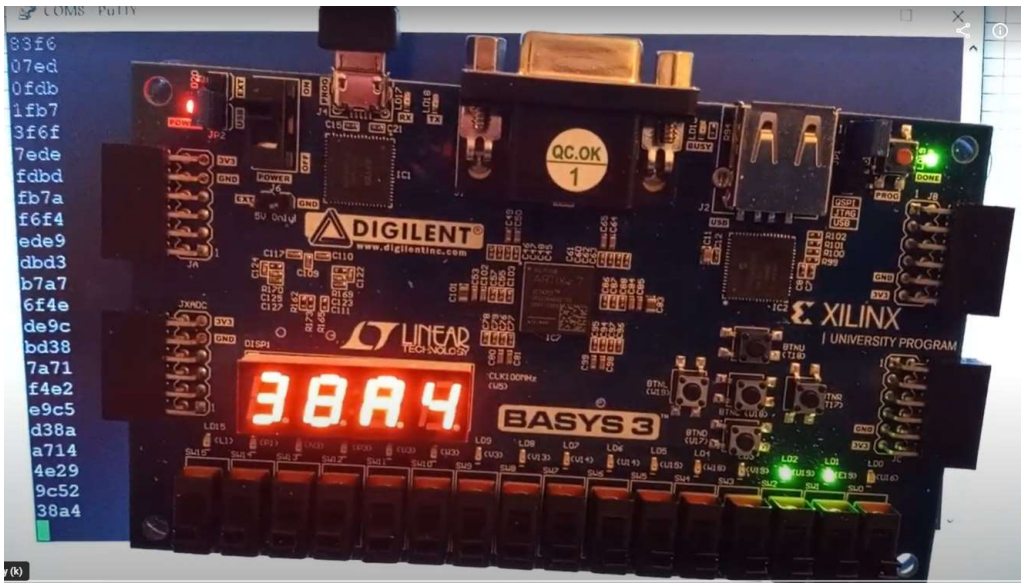


The FSM viewer was very helpful in identifying an error caused by a typo. It compiled okay but the viewer showed missing and unwanted links. Clicking on a link displays the conditions to take that path.

Source Code

The Verilog source (Vivado & Quartus) code for this project is available on github at: https://tinyurl.com/367mk52k Each project is individually zipped and the RTL *.v files are also in a separate sources folder.

A display of the board is shown here in **figure 6.** Note the Putty terminal in the background displaying the values.



The 7-segment display is showing a random hexadecimal number and the green LED in the lower right blinks with a 1 sec period. A video is shown at this link: https://photos.google.com/photo/AF1QipOCaGtcXtZpQuaBfom1GL8xVYiPAlb2abkTilhA

After using Vivado to download theough the USB port, Putty is configured to access COM8 as the USB Uart port.

Closing

It was a fun project learning Quartus. It took me longer than expected, but I was starting with zero experience on Quartus or ModelSim. My initial impression is ModelSim runs slower on Quartus, but it is the full blown ModelSim and has many functions I haven't explored.

This project allowed me to gain experience in writing RTL for a Uart and also gave me good experience in developing the FSM structure.