Artem Kulakevich
Professors Brian Cruikshank & Ataur Patwary
TA Deepen Parmar
ECE-540 System on Chip
5 Oct 2020

HW_LAB_1: SoC Design with Programmable Logic

Introduction:

This lab will involve exploring and becoming familiar with Vivado, Synthesis, the RVFpga files, PlatformIO, and RISC-V assembly.

1. **Synthesize the RVfpga and show a screenshot of RVfpga synthesis done.**

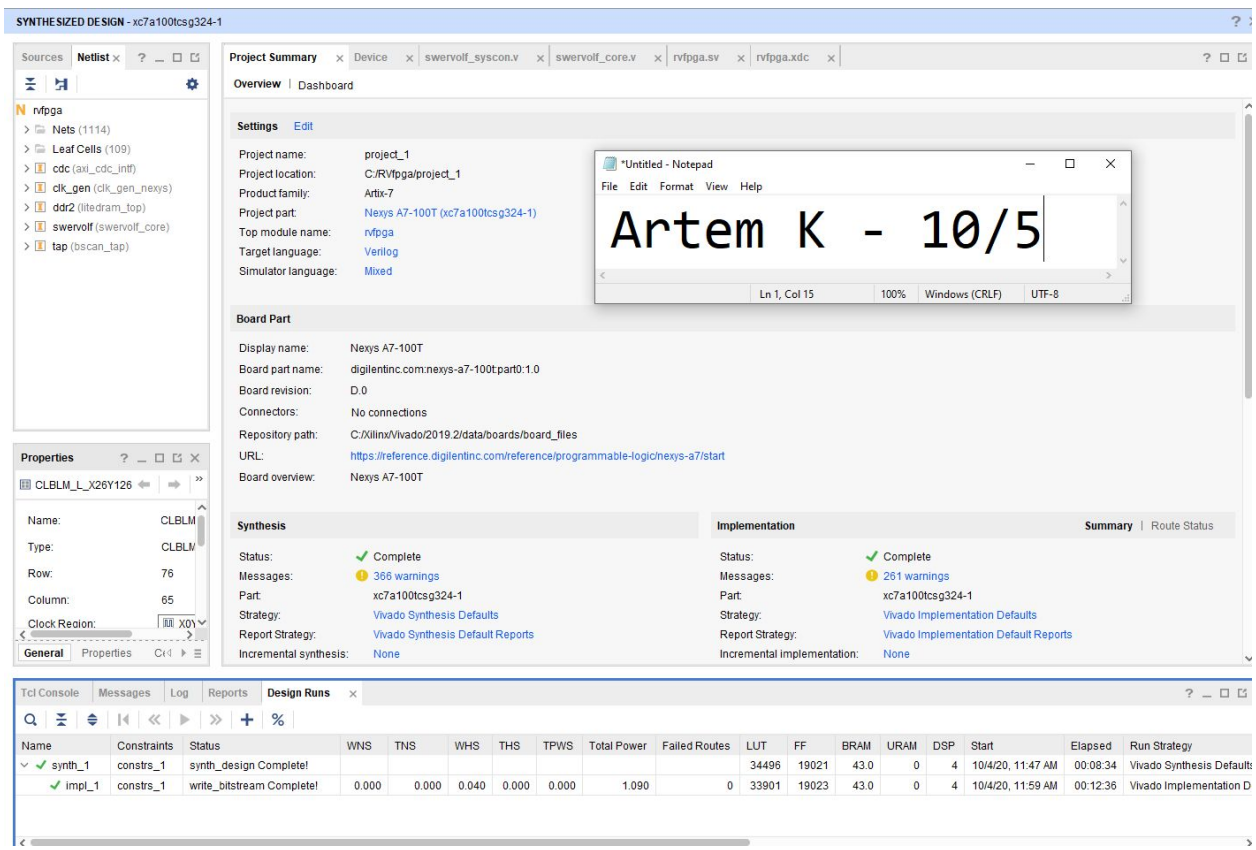


Figure 1: Synthesis of RVfpga project.

2. **Write assembly code to show a slowly incrementing counter to the 7-segment display**
   - **Include the Assembly code in your report and a picture of the Assembly success.**

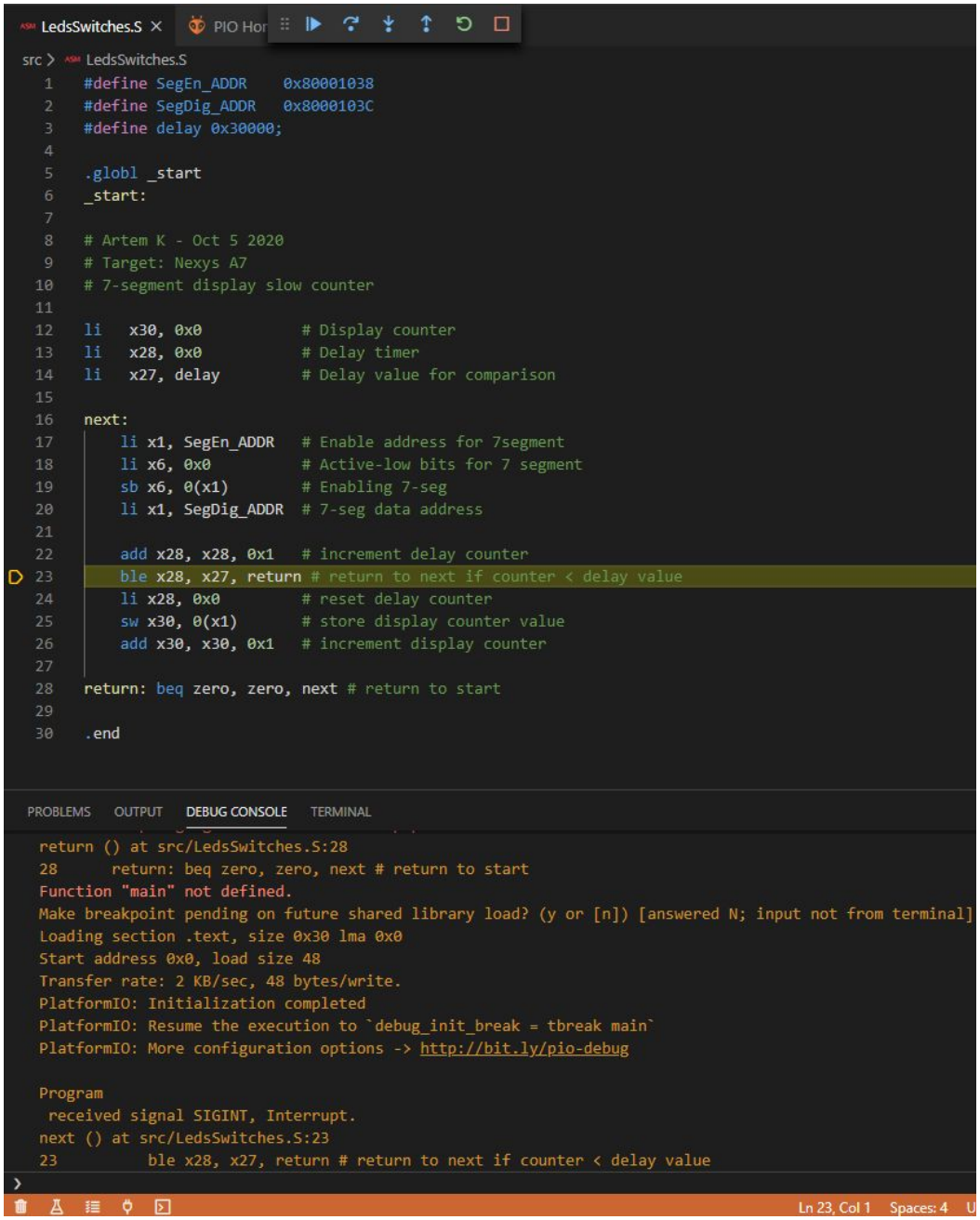


Figure 2: PlatformIO slow counter program.

```
#define SegEn_ADDR    0x80001038
#define SegDig_ADDR   0x8000103C
#define delay 0x30000;
.globl _start
_start:
```

```
# Artem K - Oct 5 2020
# Target: Nexys A7
# 7-segment display slow counter

li   x30, 0x0          # Display counter
li   x28, 0x0          # Delay timer
li   x27, delay        # Delay value for comparison
next:
   li x1, SegEn_ADDR   # Enable address for 7segment
   li x6, 0x0          # Active-low bits for 7 segment
   sb x6, 0(x1)        # Enabling 7-seg
   li x1, SegDig_ADDR  # 7-seg data address

   add x28, x28, 0x1   # increment delay counter
   ble x28, x27, return # return to next if counter < delay value
   li x28, 0x0         # reset delay counter
   sw x30, 0(x1)       # store display counter value
   add x30, x30, 0x1   # increment display counter
return: beq zero, zero, next # return to start
.end
```

- **Show picture/video of it working on a board if you have one.**

Figure 3: Slow counter program in real life

3. Look at the disassembly of your new program by finding the firmware disassembly as mentioned in class.

- **Translate two lines between Assembly and Machine language.**
- **One line that is 32 bit instruction and one that is 16-bit**

```
.pio > build > swervolf_nexys > firmware.dis

.pio\build\swervolf_nexys\firmware.elf:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <_start>:
   0: 4f01                 li t5,0
   2: 4e01                 li t3,0
   4: 00060db7             lui   s11,0x60

00000008 <next>:
   8: 800010b7             lui   ra,0x80001
   c: 03808093             addi  ra,ra,56 # 80001038 <return+0x8000100c>
  10: 4301                 li t1,0
  12: 00608023             sb t1,0(ra)
  16: 800010b7             lui   ra,0x80001
  1a: 03c08093             addi  ra,ra,60 # 8000103c <return+0x80001010>
  1e: 0e05                 addi  t3,t3,1
  20: 01cdd663             bge   s11,t3,2c <return>
  24: 4e01                 li t3,0
  26: 01e0a023             sw t5,0(ra)
  2a: 0f05                 addi  t5,t5,1

0000002c <return>:
  2c: fc000ee3             beqz  zero,8 <next>
```

Figure 4: Disassembly of slow counter program.

Disassembly	**Instruction**
800010b7	lui ra, 0x80001
4301	li t1, 0

lui ra, 0x80001

1000 0000 0000 0000 0001	00001	0110111

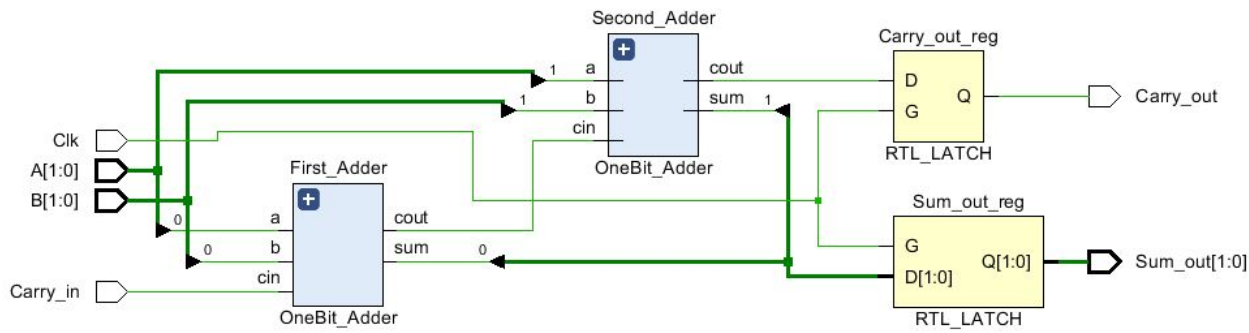
1000	0000	0000	0000	0001	0000	1011	0111
8	0	0	0	1	0	B	7

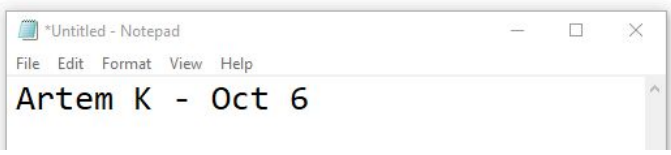
li t1, 0

010	0	00110	00000	01

0100	0011	0000	0001
4	3	0	1

3. Write SystemVerilog code for the following 2 schematic views:





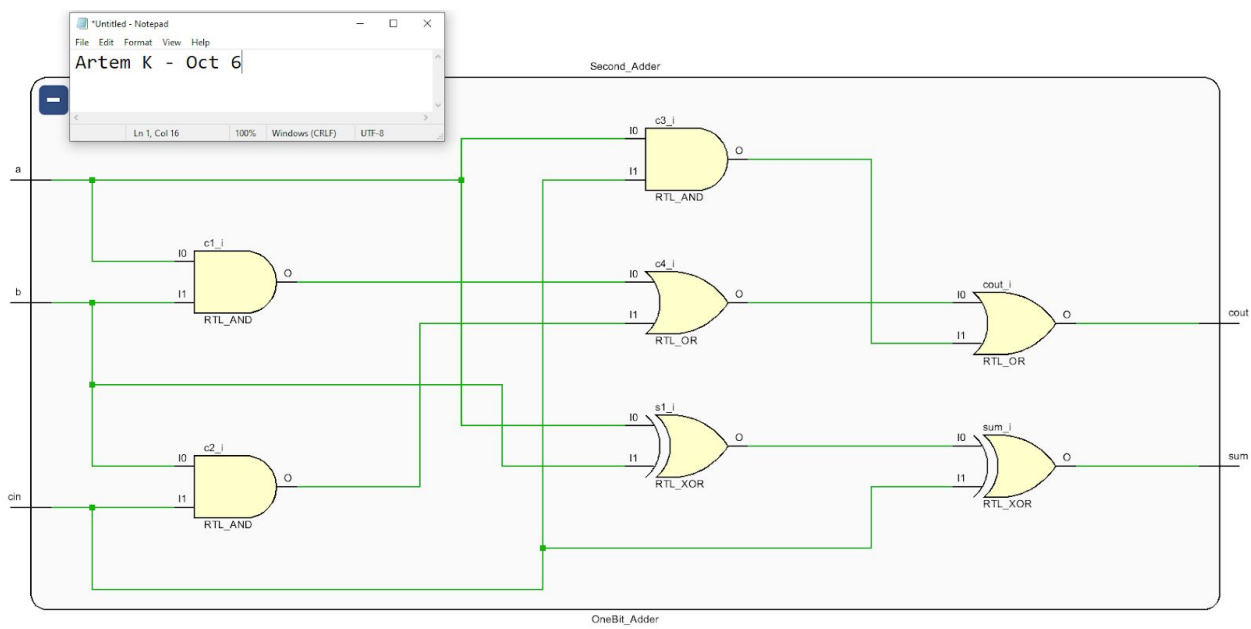


Figure 5: Vivado generated circuits.

```
module OneBit_Adder(
      input a,
      input b,
      input cin,
      output cout,
      output sum
      );
      wire c1,c2,c3,c4, s1;

      and(c1, a, b);
      and(c2, b, cin);
      and(c3, a, cin);
      or(c4, c1,c2);
      or(cout, c4, c3);

      xor(s1, a, b);
      xor(sum, s1, cin);

endmodule

module AdderCircuit(
      input Clk,
      input [1:0] A,
      input [1:0] B,
      input Carry_in,
      output logic Carry_out,
      output logic [1:0] Sum_out
      );

      wire cout_one, sum_one, cout_two, sum_two;

      OneBit_Adder First_Adder (.a(A[0]), .b(B[0]), .cin(Carry_in),
.cout(cout_one), .sum(sum_one));
      OneBit_Adder Second_Adder(.a(A[1]), .b(B[1]), .cin(cout_one),
.cout(cout_two), .sum(sum_two));

      always @(Clk or cout_two)
      if (Clk)
      begin
            Carry_out <= cout_two;
      end

      always @(Clk or {sum_two, sum_one})
      if (Clk)
      begin
            Sum_out <= {sum_two, sum_one};
      end
```

```
endmodule
```

4. **Trace in verilog RTL code between the following points and include in your report.**
   - **Axi2wb and SevSegDisplays_Controller Enables_reg/Digits_reg inputs**
     i. **Find the related ports of Axi2wb.**

swervolf.syscon.v	. In the file swervolf_syscon.sv there is a module called "SevSegDisplay_Controller", with an 8-bit input bus **Enables_Reg** and a 32-bit input bus **Digits_Reg**.
swervolf_syscon.v	. SevSegDisplay_Controller is instantiated in module "swervolf_syscon", with an 8-bit reg bus **Enables_Reg** and a 32-bit reg bus **Digits_Reg**. . In the module swervolf_syscon the bus **Enables_Reg** gets values from first 8 bits of the 32-bit bus **i_wb_dat** when the 6-bit bus bits **i_wb_adr[5:2]** are equal to 14 and **i_wb_sel[0]** = true. . In the module swervolf_syscon the 32-bit bus **Digits_Reg** gets various bytes from the 32-bit bus **i_wb_dat** when 6-bit bus bits **i_wb_adr[5:2]** are equal to 15 and the bytes received depend onwhich **i_wb_sel[0...3]** bits are enabled.
swervolf_core.v / wb_intercon.vh	. swervolf_syscon is initiated in the module "swervolf_core" as "syscon", **i_wb_dat** is connected to **wb_m2s_sys_dat**. . **wb_m2s_sys_dat** a 32-bit wire and is described in the file "wb_intercon.vh" and included in swervolf_core.v file. . **wb_m2s_sys_dat** is connected to **wb_sys_dat_o** which the output of the module "wb_intercon"
wb_intercon.v	. In wb_intercon the module "wb_mux" in instantiated and 32-bit bus **wbs_sys_dat_o** is connected to part of the wb_mux output called **wbs_dat_o.**
wb_mux.v	. In the module wb_mux, the wbs_dat_o value is assigned through the function "num_slaves{**wbm_dat_i**}};
wb_intercon.v	. **wbm_dat_i** is connected to a 32-bit bus **wb_io_dat_i** inside the module wb_intercon

swervolf_core.v / wb_intercon.vh	. **wb_io_dat_i** is connected to a 32-bit bus **wb_m2s_io_dat** in the module swervolf_core. . In the module swervolf_core.v, the module “axi2wb” is instantiated and the 32-bit **wb_m2s_io_dat** is connected to the 32-bit output reg **o_wb_dat**
axi2wb.v	. In the axi2wb module the **o_wb_dat** output receives it’s data from the function o_wb_dat <= hi_32b_w ? i_wdata[63:32] : i_wdata[31:0];. In the states AWACK and IDLE if i_wvalid is true. **i_wdata** is a 64-bit input wire bus to axi2wb.
swervolf_core.v / wb_intercon.vh	. **i_wdata** is connected to **io_wdata.**

- **SevSegDisplays_Controller AN, Digits_bits outputs and the top level CA..CG and AN[7:0] output ports inside rvfpga.sv.**

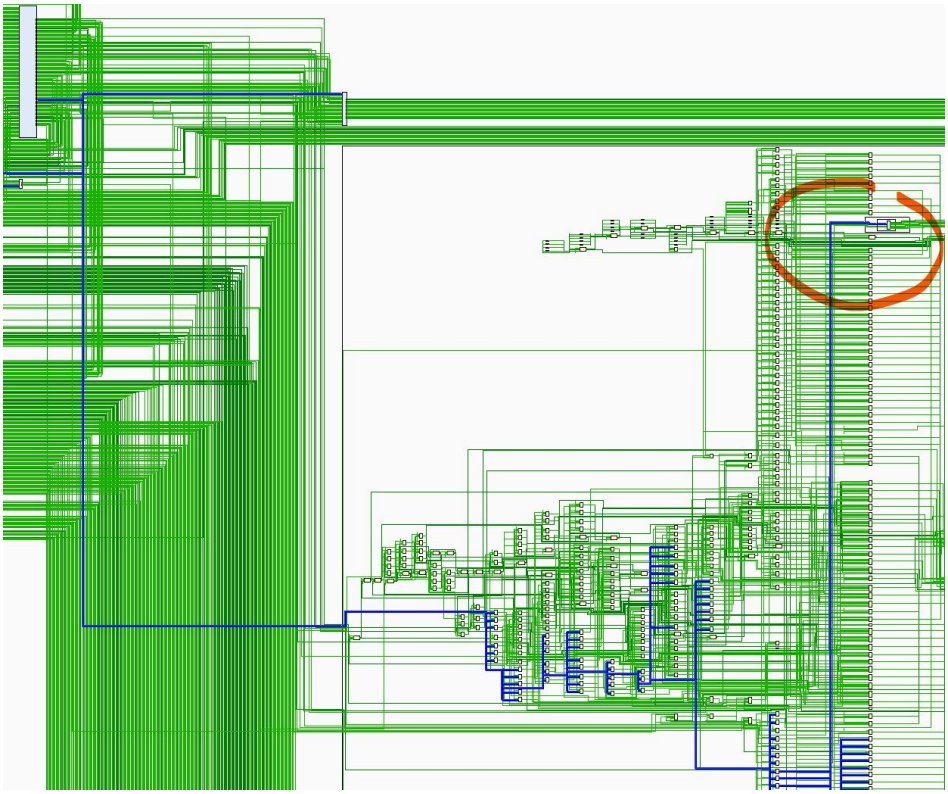
swervolf_syscon .v	. In the file swervolf_syscon.sv there is a module called “SevSegDisplay_Controller”, with an 8-bit output bus **AN** and a 7-bit output bus **Digits_Bits**.
swervolf_core.v	. The module swervolf_syscon is instantiated in the module “swervolf_core”, and the outputs **AN** and **Digits_Bits** of swervolf_syscon are connected to the same size outputs **AN** and **Digits_Bits** of swervolf_core.
rvfpga.sv	. Swervolf_core is instantiated in the module “rvfpga”, and the output **AN** is connected to the same size output reg **AN,** while the output **Digits_Bits** is connected to the concatination of the output regs **CA, CB, CC, CD, CE, CF, CG.**

- **Find the related XDC constraint lines to the top level rvfpga.sv ports.**

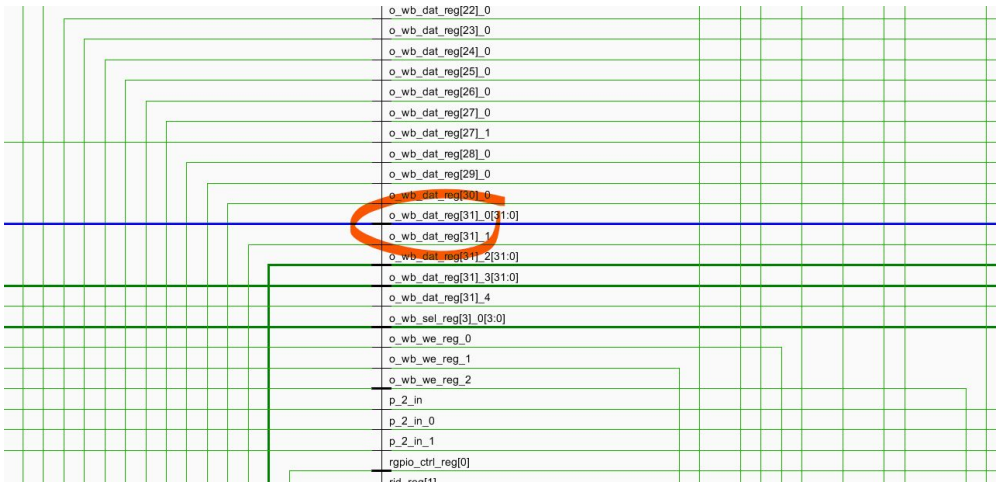
rvfpga.xdc	##7 segment display set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { CA }]; #IO_L24N_T3_A00_D16_14 Sch=ca

	set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { CB }]; #IO_25_14 Sch=cb set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { CC }]; #IO_25_15 Sch=cc set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { CD }]; #IO_L17P_T2_A26_15 Sch=cd set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { CE }]; #IO_L13P_T2_MRCC_14 Sch=ce set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { CF }]; #IO_L19P_T3_A10_D26_14 Sch=cf set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { CG }]; #IO_L4P_T0_D04_14 Sch=cg

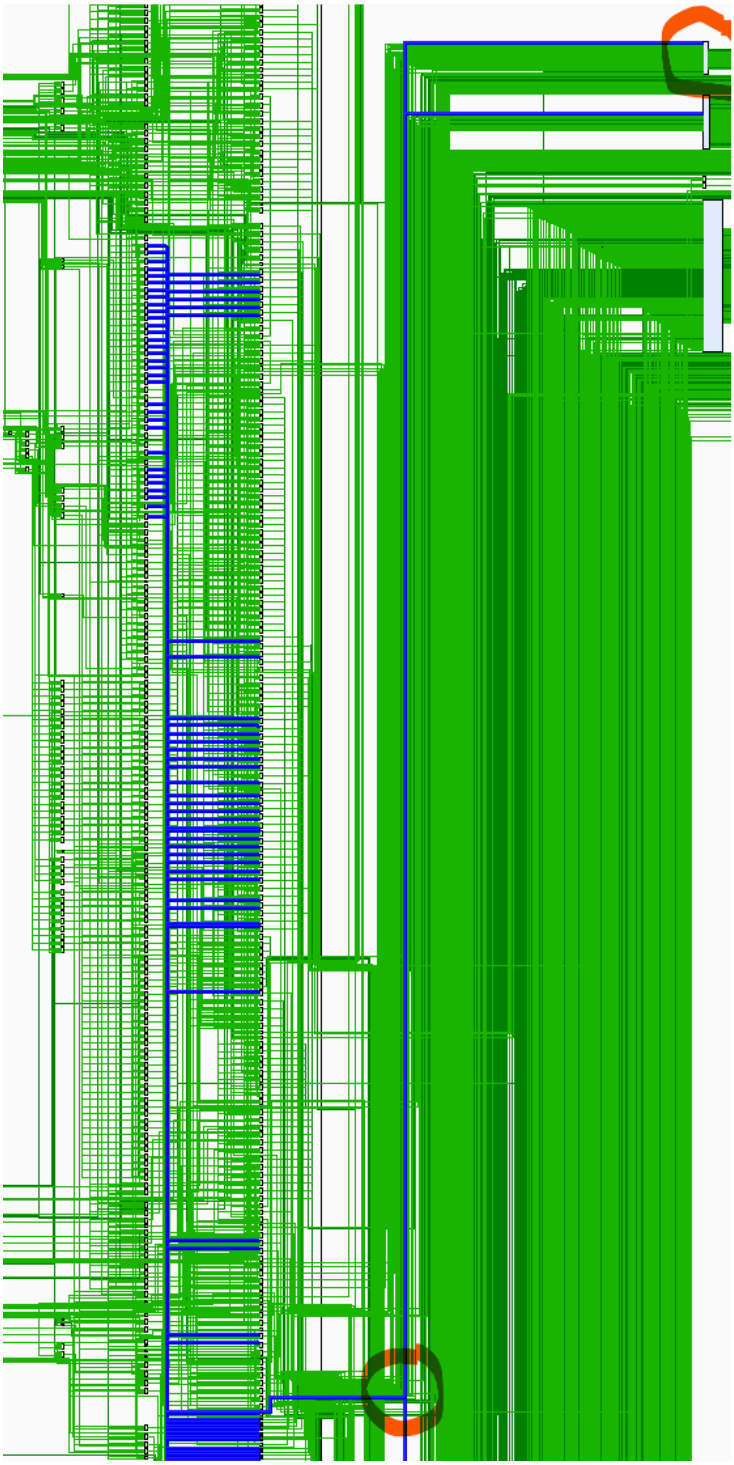
- **Trace in the schematic view of Vivado if you can.**



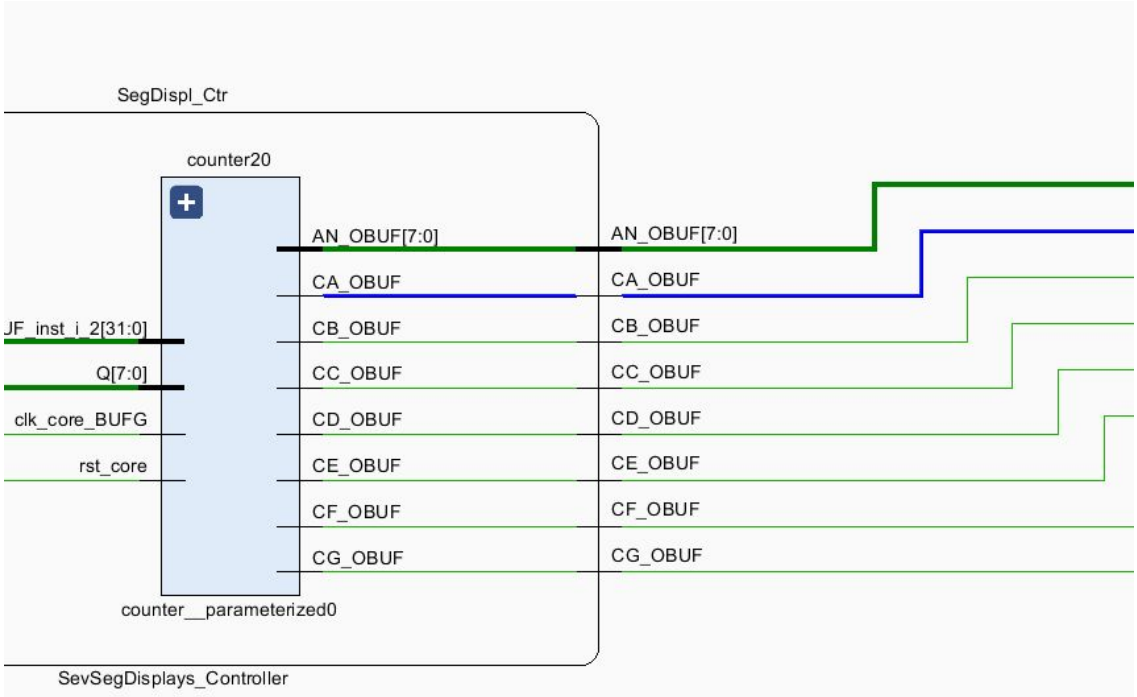
SegDisp_Ctrl to axi2wb



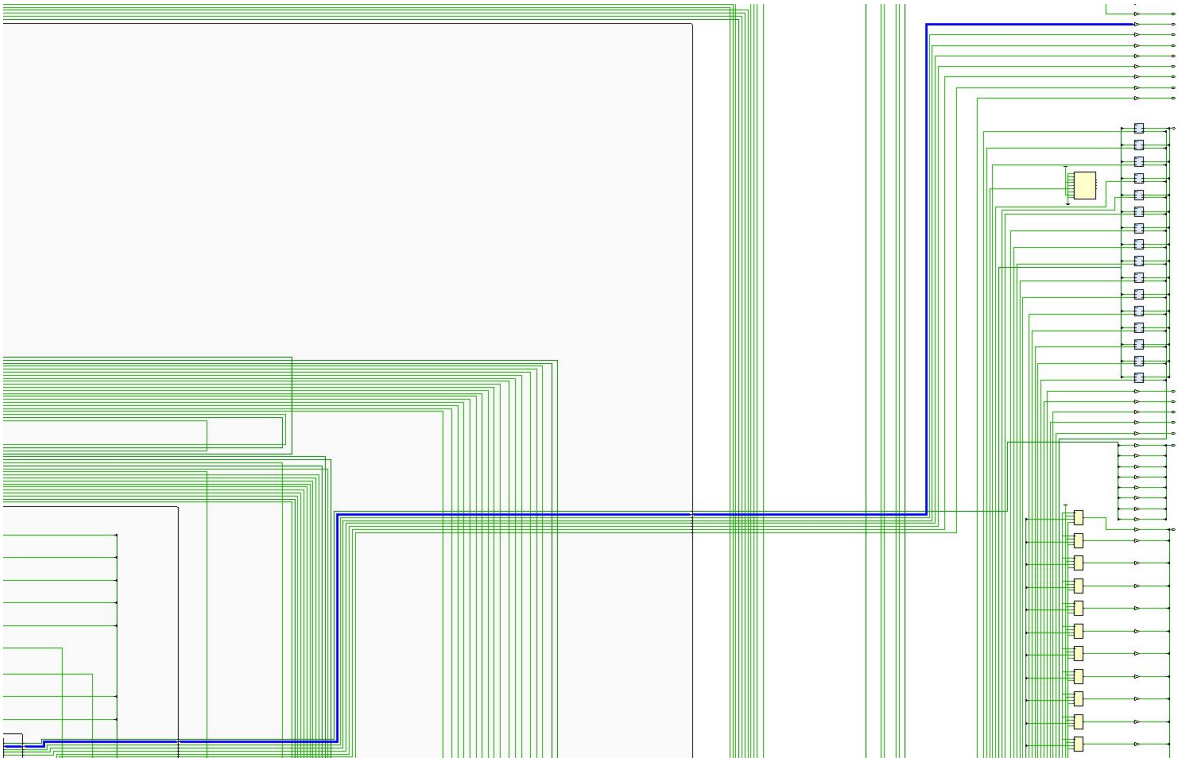
Connection to axi2wb



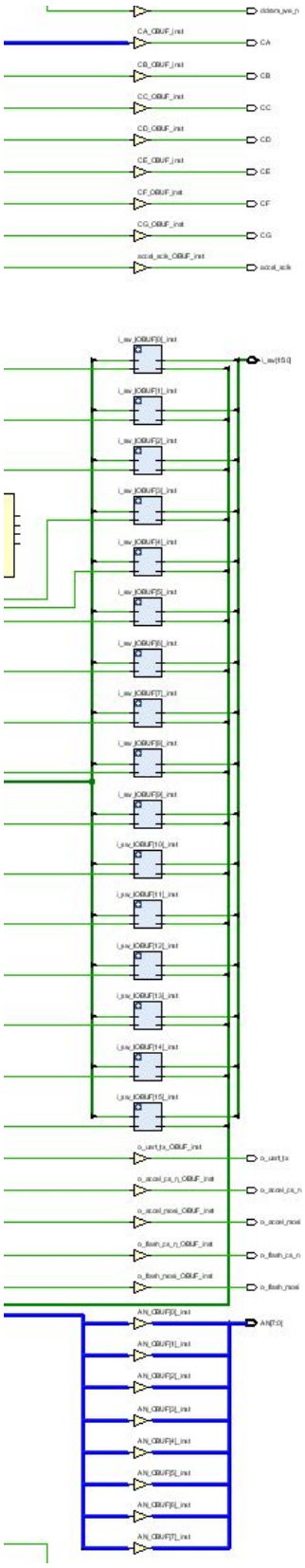
Connection from Swervolf_syscon to axi2wb.



SegDispl_Ctr CA output



SegDesplay_Ctr CA to RVFPGA IO out



RVfpga IO output

Conclusion:

This lab was a great way to introduce all the tools. Synthesizing our own verilog code did result in some issues, since it looks like the libraries and setting are not always all the same. Additionally searching for a path through the files was incredibly difficult especially for the Digits_Reg route, since there were so many additional files and hoops to jump through. It's a bit surprising that all of this works so well. And it's very interesting being able to control everything in the chip, even the structure of it. The biggest difficulty in this project comes from the fact that there are no solid datasheets for RVFPGA. This will potentially be an even bigger problem when programming the other projects in assembly.