

Digital Bubble Level - Cell Layout

Vladislav Pomogaev - 26951160

December 5, 2021

1 Introduction

This device forms a digital bubble level; also called a spirit level. It can be helpful in levelling things horizontally. The project began as a synthesised set of Verilog files for the Efinix XylonI FPGA development board, but I have since extracted the core FSM from that project, laid it out with the 45nm standard cell library, and verified its operation though simulations.

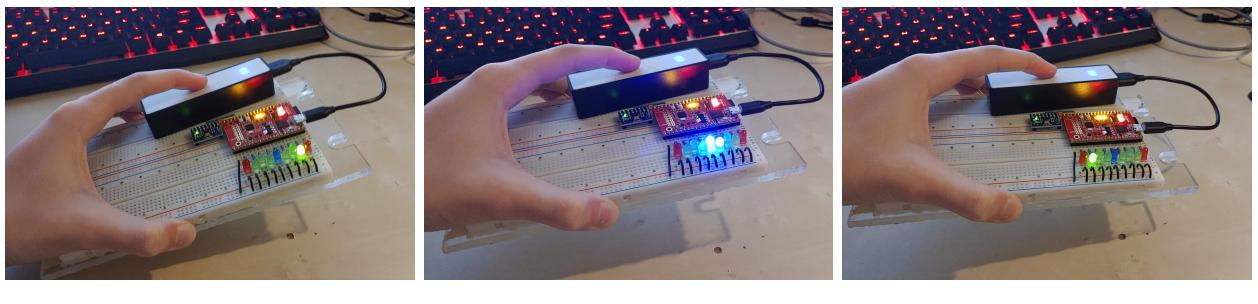


Figure 1: Photo stills of the original device in action. When you tilt the device to the left as in a), the LEDs to the right light up. As you tilt the device more to the right, the lit LED moves from right to left as in b) and c). When level, the blue center LED lights up.

2 Overview

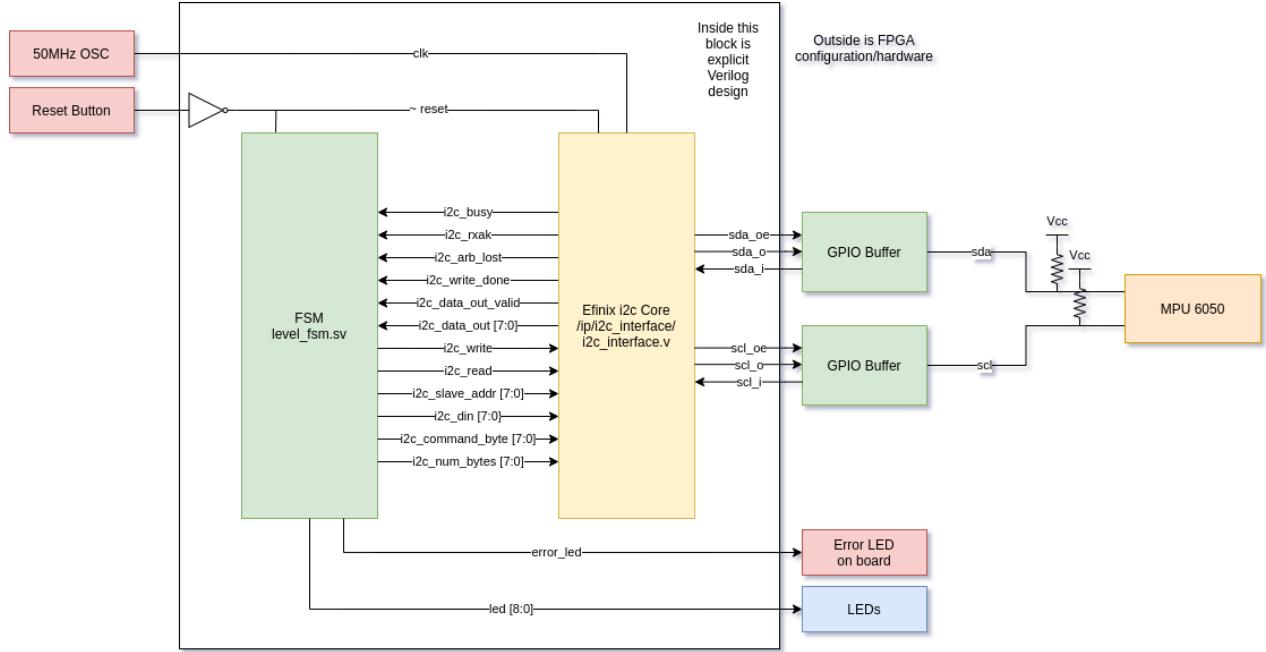


Figure 2: Block diagram of the FSM and how it forms the bubble-level system. The level FSM talks to the I2C core, which communicates to the MPU via a set of buffered GPIO blocks on the FPGA. A reset button resets all FSMs to their initial state. The design runs is compiled for a 50MHz clock constraint. The output to outside of the FPGA is buffered through the GPIO logic blocks, which was the reason I used IP specific to the FPGA. I only did the layout and verification for the level_fsm.sv file from the original project.

2.1 Ports

Port	Width (bits)	Interface	Direction	Description
clk_i	1	System	Input	Clock input. 50MHz max.
reset_i	1	System	Input	Active high reset input
i2c_busy_i	1	I2C	Input	Logic high indicates that the I2C bus is busy.
i2c_rxak_i	1	I2C	Input	Logic low indicates that the I2C slave device received and acknowledged the I2C transfer
i2c_arb_lost_i	1	I2C	Input	Logic high indicated there is arbitration lost in the I2C transfer
i2c_write_done_i	1	I2C	Input	Logic high indicates that I2C master write data is sent and ready to accept by I2C slave device.
i2c_data_out_valid_i	1	I2C	Input	Logic high indicates that I2C master read data is valid and ready to read by user.
i2c_data_out_i	8	I2C	Input	Data read input port
i2c_write_o	1	I2C	Output	Write to slave strobe high
i2c_read_o	1	I2C	Output	Read from slave strobe high
i2c_slave_addr_o	8	I2C	Output	Address of slave in 8bit format. Add trailing zero to use 7-bit addressing.
i2c_din_o	8	I2C	Output	Data read from slave
i2c_command_byte_o	8	I2C	Output	Command byte sent to slave. (Register to read from)
i2c_num_bytes_o	8	I2C	Output	Number of bytes of data to write or read. Includes the command byte (if sending one byte, i2c_num_bytes.o = 'd2)
error_led_o	1	System	Output	Active high if an error has occurred in state machine; lost bytes, error writing to slave or reading. Reset FSM to try again.
led_o	9	System	Output	LED array output of bubble level

3 Standard Library Layout

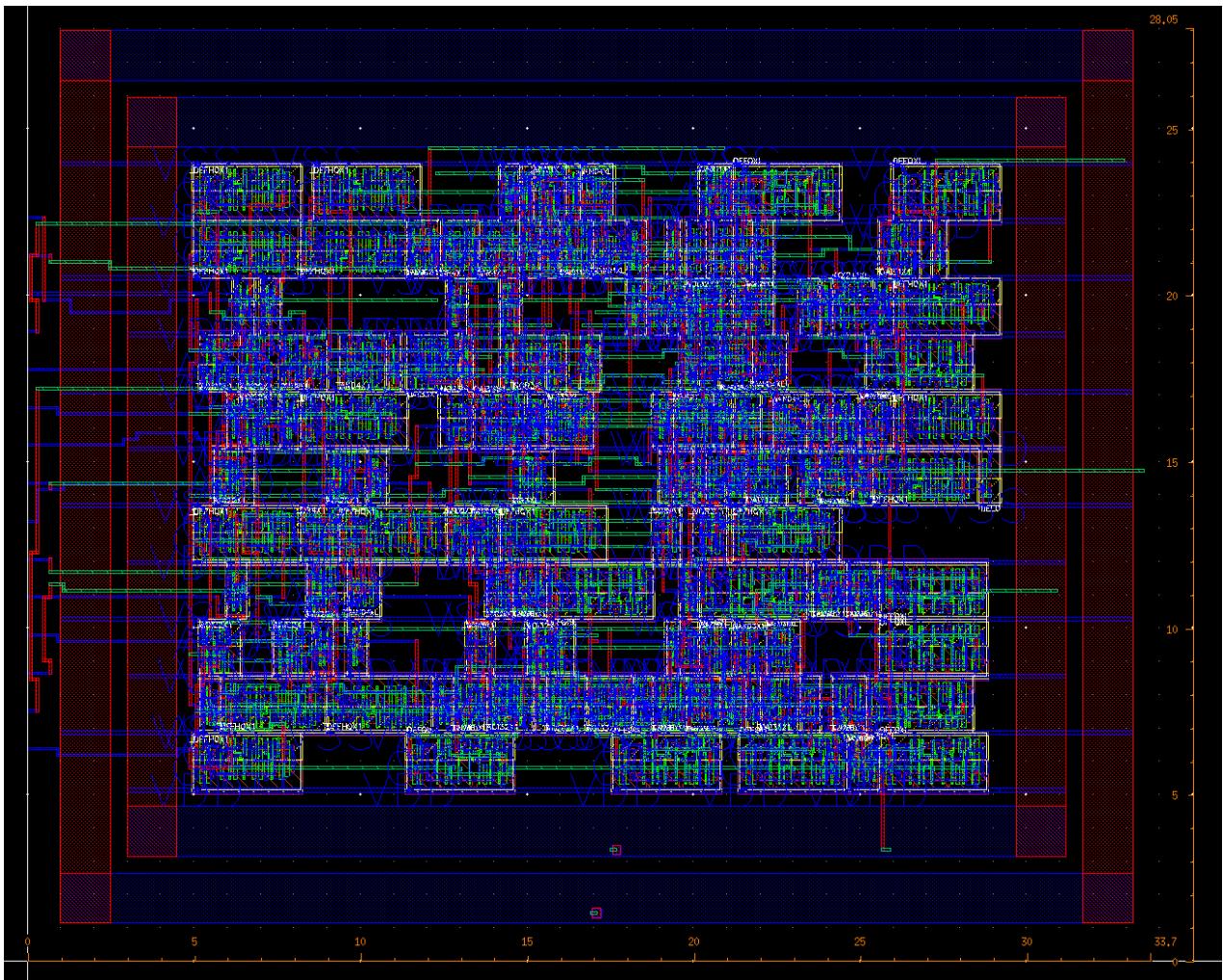


Figure 3: The level_fsm.sv file was synthesized and laid out using the 45nm standard cell library provided. The ports were aligned as such: inputs on the left, outputs on the right, and power on the bottom. The very outer dimensions are 33um x 28um, for a total area of 924um $\hat{2}$.

4 Simulation/Verification

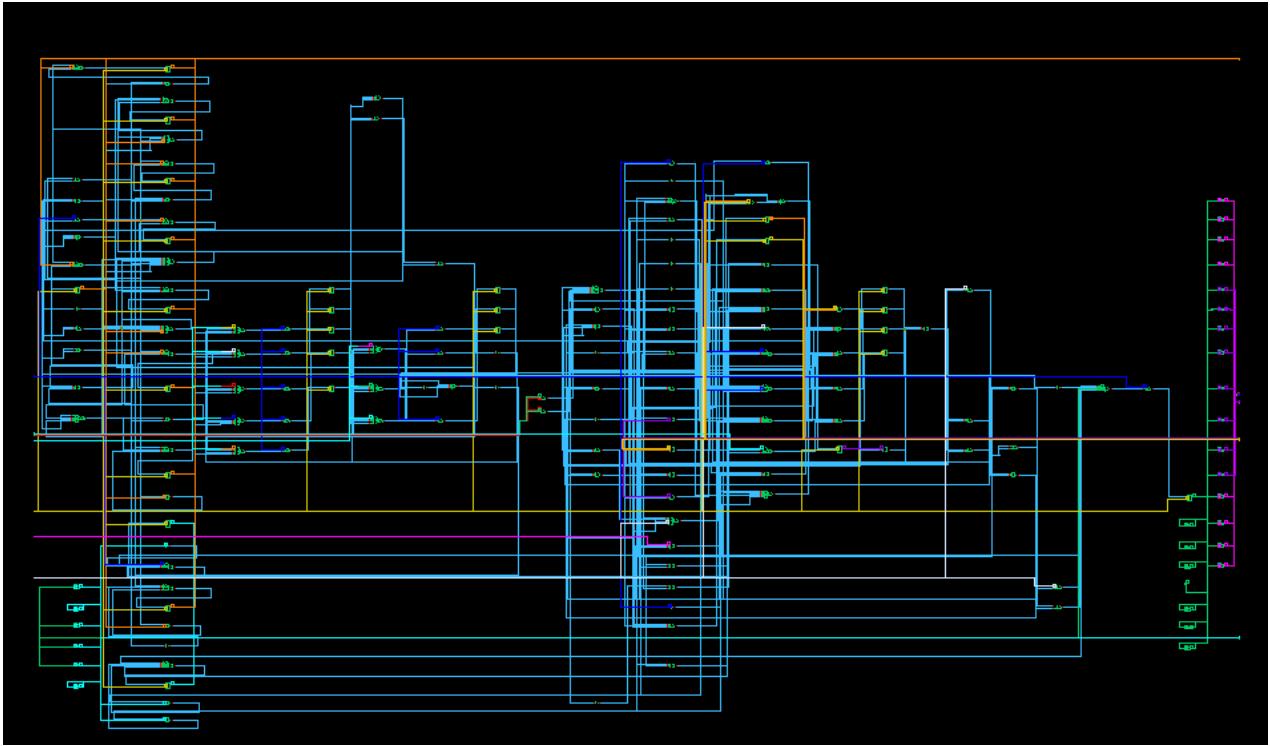


Figure 4: The Verilog file was also translated to a RTL layout and displayed in viewable form. Here we see the entire FSM schematic in terms of logic gates.

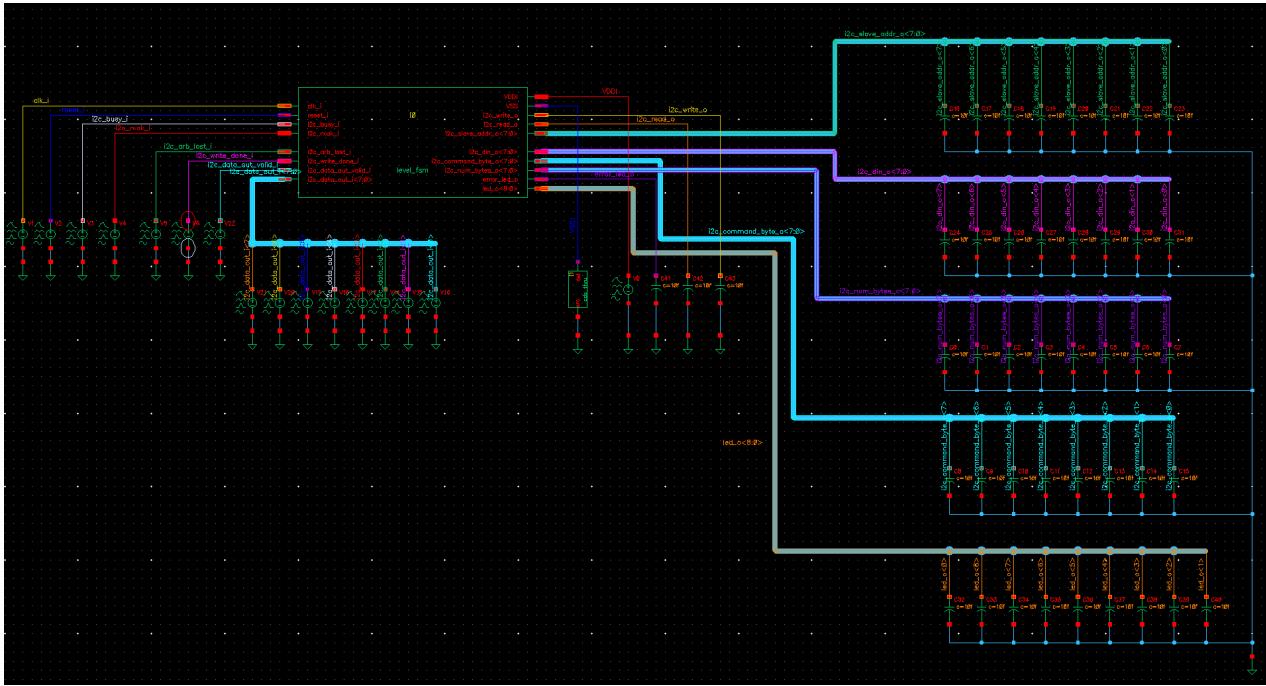


Figure 5: The previous RTL design was transformed into a new cell view. This cell view was placed in a test-bench schematic. Each output of the FSM was loaded with a 10fF capacitor. An array of 1V supplies was used to drive the input signals.

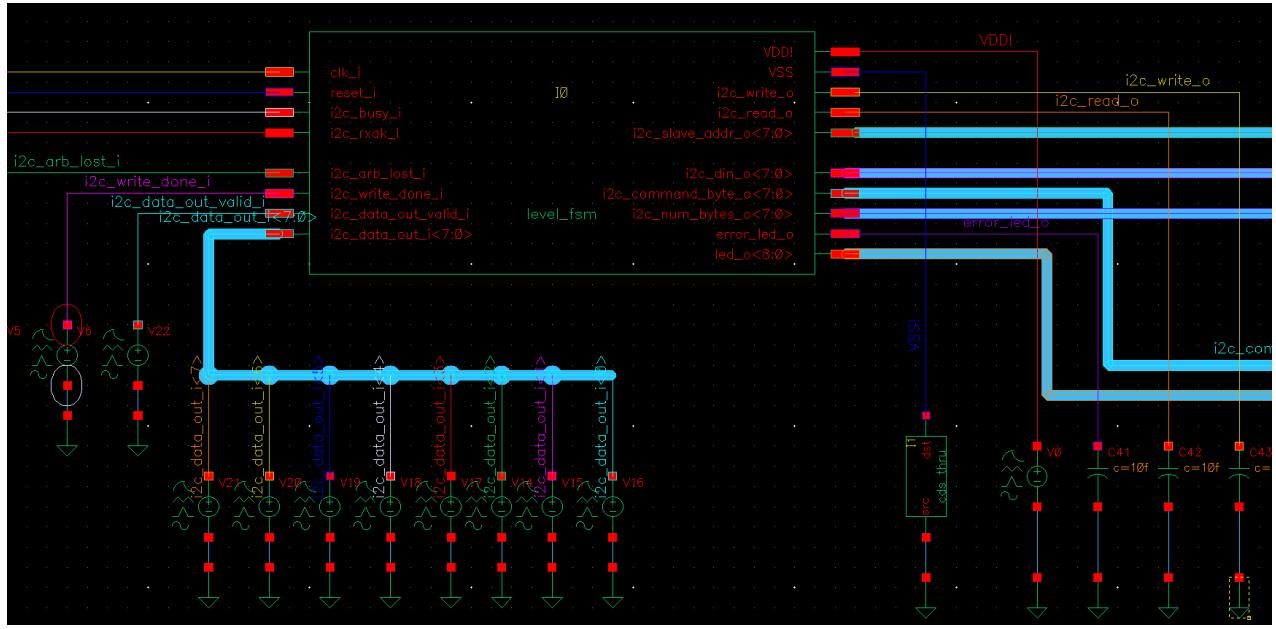


Figure 6: Here is a closeup of the test-bench schematic. The voltage input signals were set up manually using an array of input voltage levels that corresponded to the input vectors used in the original Verilog test bench.

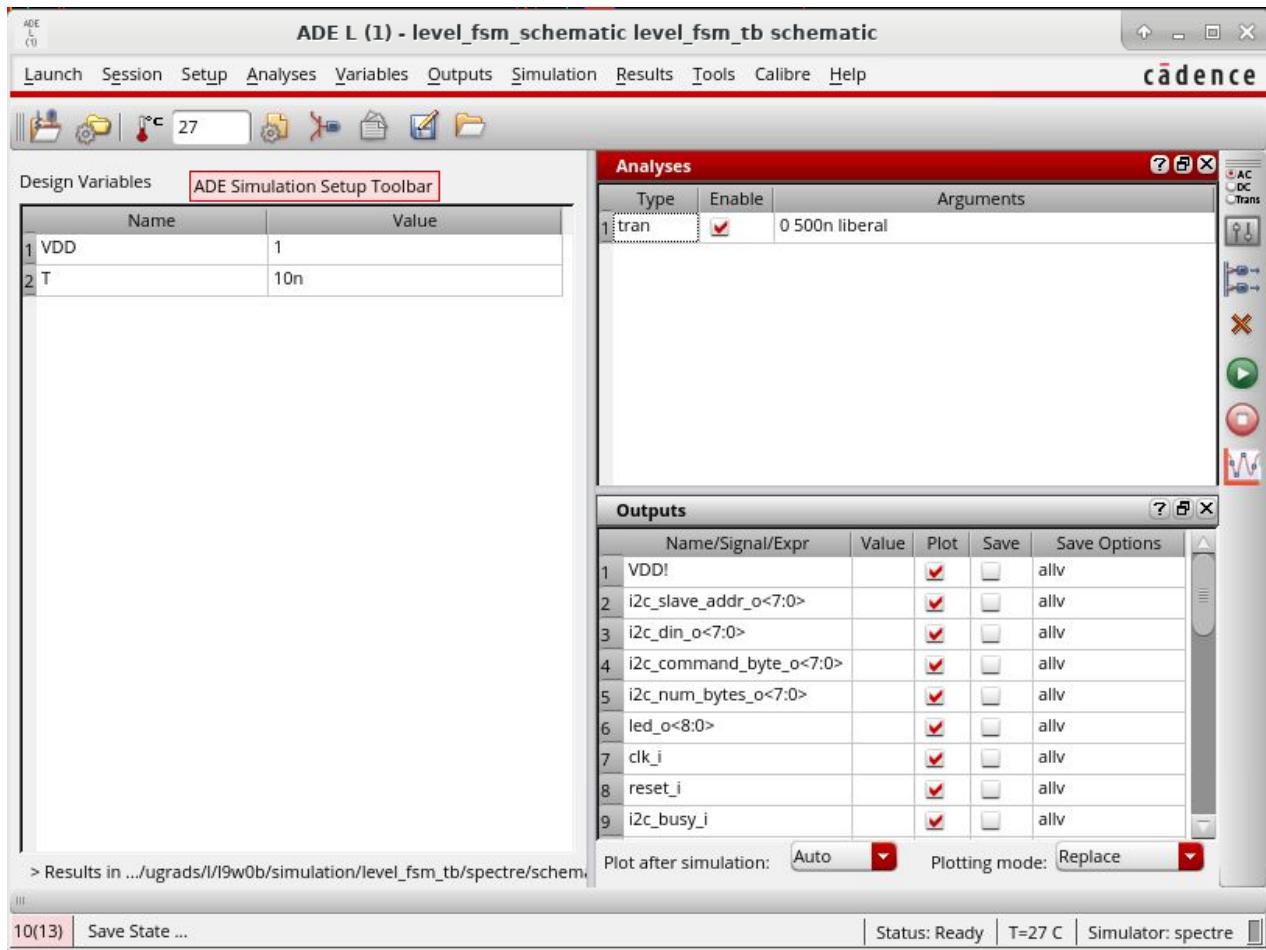


Figure 7: The simulation was set up using ADE. The simulation duration was the same length as the original Verilog test bench file.

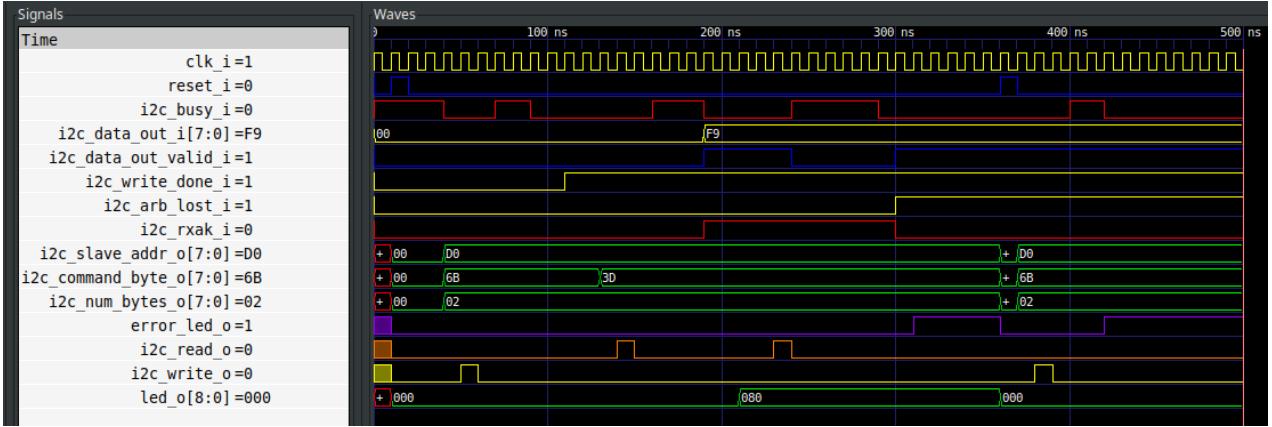


Figure 8: For comparison, here is the input and output vector of the original Verilog simulation. This simulation tests does the following in this order: 1) FSM reset 2) Write to accelerometer to wake it up 3) Wait for valid response 4) Read from accelerometer register 5) Display level on LEDs. 6) Read from accelerometer register, but this time receive an error 7) Verify error_led output 8) Reset FSM 9) Write to accelerometer to wake it up, but this time receive an error 10) Verify error_led output 11) Stall forever. This procedure tests an overwhelming majority of the FSM.

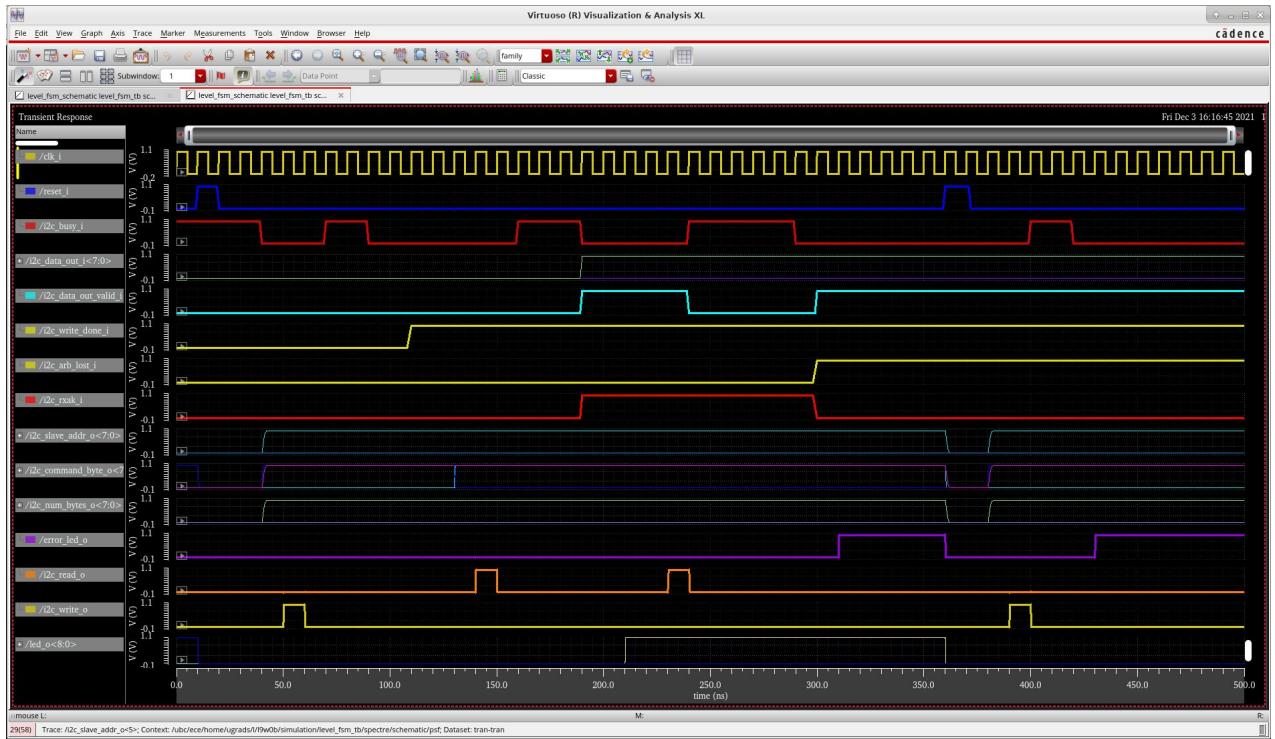


Figure 9: Here is the same test bench but now it is executed in Virtuoso. As we can see the waveforms are the same, but we see characteristic RC delays. The multi-bit vector values were the same in both cases. Rise time for input signals was 2ns. The layout passed the test!

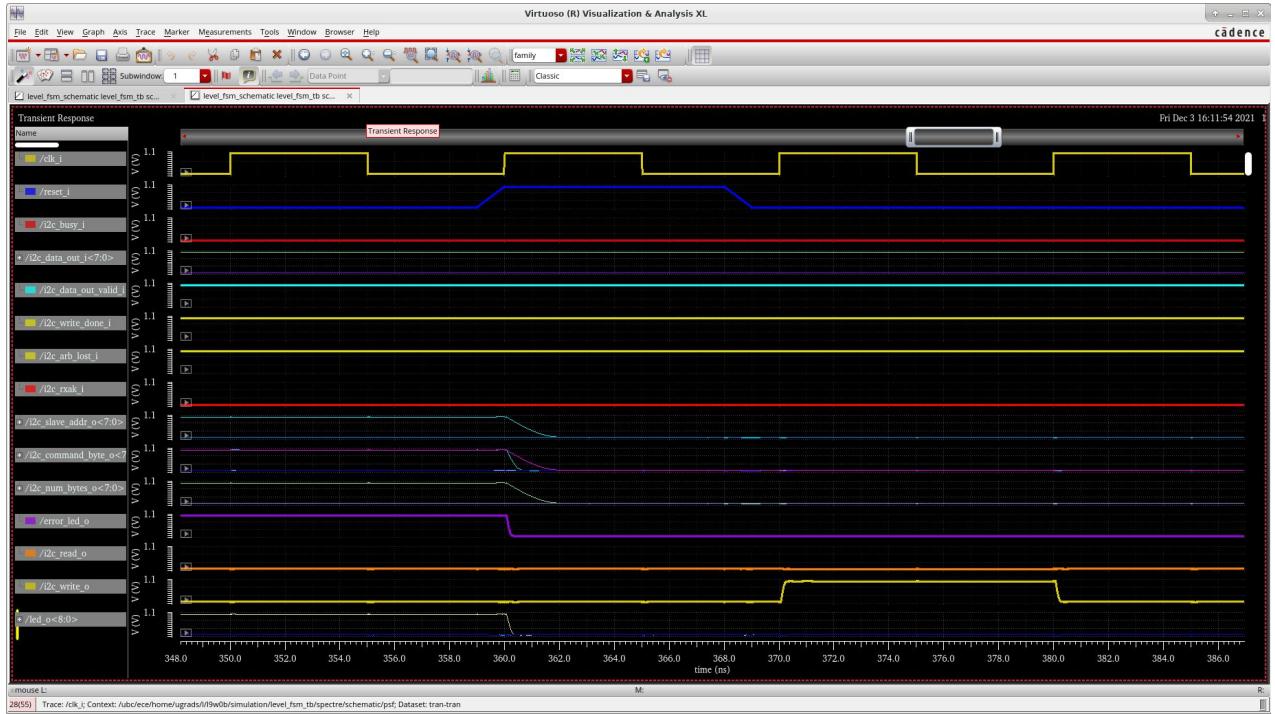


Figure 10: I encountered an interesting problem while troubleshooting the simulation. Part of my vectors from the Verilog and Virtuoso simulations were not lining up. Upon an FSM reset, the slave addr, command byte, and num bytes should be going to 0, then back to their default non-zero values after a reset. This happens on the Verilog simulation, but not on the Virtuoso one. However, the rest of the FSM's functionality would continue as normal. Without those signals though, the FSM would not read/write from the device correctly though. I found that adjusting the length of the reset signal to be slightly MORE than two cycles would solve this issue. However, the root cause of the issue is poor test bench design. It was ultimately a setup-time violation on the reset line that would cause the FSM to skip a state.



Figure 11: What the FSM outputs should look like according to the Verilog test bench. Note how the slave address and other busses change their values at 370ns, but in the Virtuoso they do not.



Figure 12: Extending the reset pulse in Virtuoso solved this bug. Now the simulations are functionally identical.

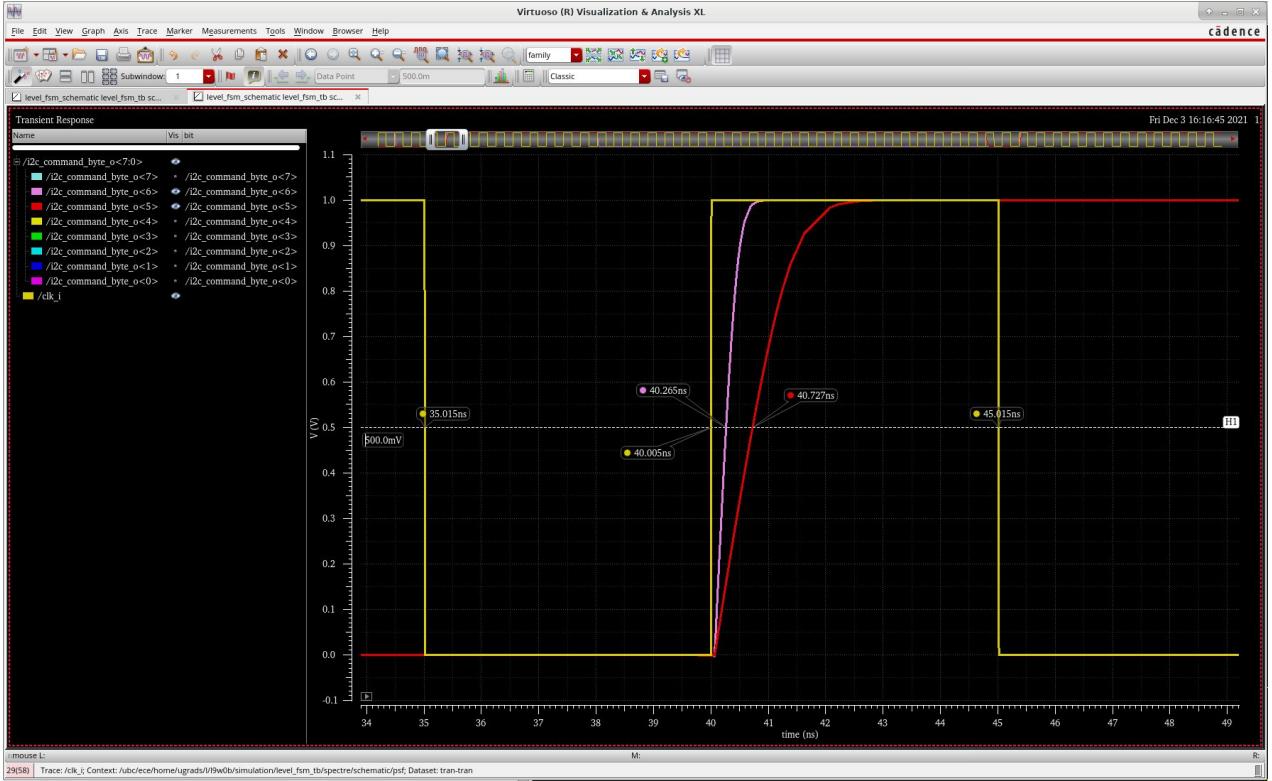
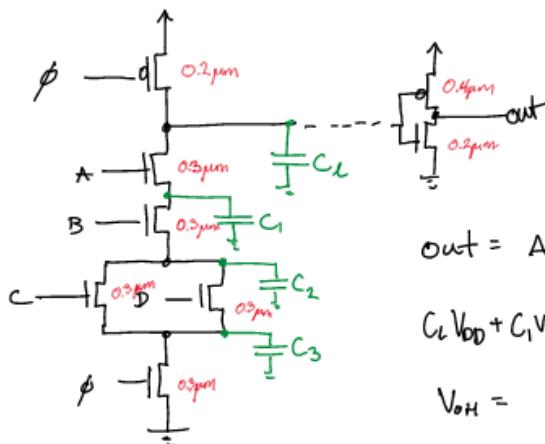


Figure 13: Another thing I played around with is measuring the propagation delay. I found the worst-case delay to be on the i2c_command_byte bus. The worst case delay was a staggering 722ps. Since the FSM is rather simple and it's function is really just to pipe data from the accelerometer to the LEDs, I can be quite confidant that this is in fact the worst-case delay.

5 Written Questions

2)



$$out = AB(C+D)$$

$$C_L V_{DD} + C_1 V_1 + C_2 V_2 = (C_L + C_1 + C_2) V_{OH}$$

$$V_{OH} = \frac{C_L V_{DD} + C_1 V_1 + C_2 V_2}{C_L + C_1 + C_2}$$

for max reduction in V_{OH} , $V_{1,i} = V_{2,i} = 0V$

worst-case-scenario:

ϕ	eval 1		pre.	eval 2		
	X	1	1	O	X	X

$$ABCD = X111 \quad OXXX \quad 1100$$

$$\begin{aligned} C_L &= (0.5\mu m)(1.7\mu m) + (0.6\mu m)(2.0\mu m) \\ &= 0.5fF + 1.2fF \\ &= 1.7fF \end{aligned}$$

$$\begin{aligned} C_1 &= 0.6\mu m \times 1.7fF/\mu m \\ &= 0.6fF \end{aligned}$$

$$\begin{aligned} C_2 &= 0.9\mu m \times 1.7fF/\mu m \\ &= 0.9fF \end{aligned}$$

I'm assuming
no source/drain
sharing here.

Charge sharing voltage reduction %

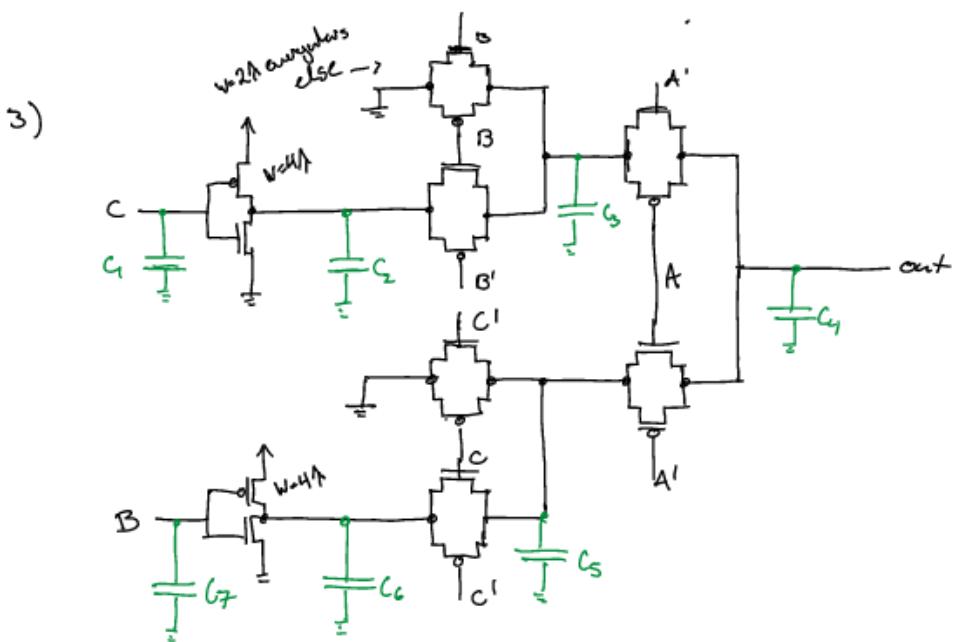
$$= \frac{V_{OH} \times 100\%}{V_{DD}} = \frac{C_L \times 100\%}{C_L + C_1 + C_2} = \frac{1.7 \times 100\%}{1.7 + 0.6 + 0.9} = \frac{1.7 \times 100\%}{3.2} = 53\%$$

$$C_L = 1.7fF$$

$$C_1 = 0.6fF$$

$$C_2 = 0.9fF$$

$$\% = \frac{1.7}{1.7 + 0.6 + 0.9} \times 100\% = \frac{1.7}{3.2} \approx 53\%$$



$$C_1 = 6A \times C_g$$

$$C_2 = 6A \times C_{eff} + 4A C_{eff} + B \cdot (2A C_g)$$

$$C_3 = 4A C_{eff} + B(2A C_g) + 4A C_{eff} + B'(2A C_g) + 4A C_{eff} + A'(2A C_g)$$

$$= 12A C_{eff} + 2A C_g + A'(2A C_g)$$

$$C_4 = A(2A C_g) + A'(2A C_g) + 8A C_{eff}$$

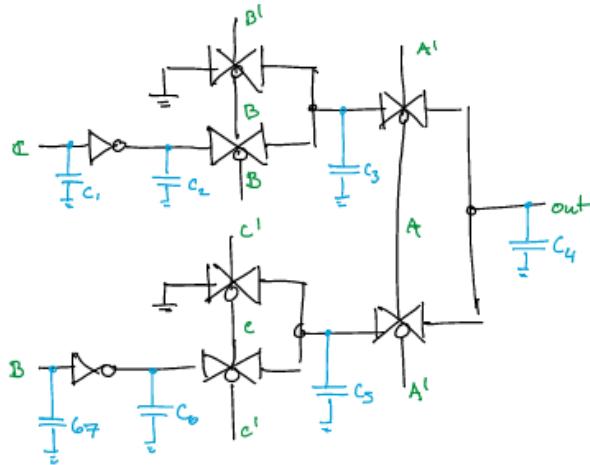
$$= 8A C_{eff} + 2A C_g$$

$$C_5 = 12A C_{eff} + C(2A C_g) + C'(2A C_g) + A(2A C_g)$$

$$= 12A C_{eff} + 2A C_g + A(2A C_g)$$

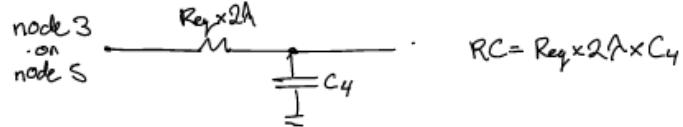
$$C_6 = 10A C_{eff} + C(2A C_g)$$

$$C_7 = 6A C_g$$



$$out = \bar{C}BA' + \bar{B}CA$$

For minimum delay I will assume we can use elmore delay and ignore charged capacitors.

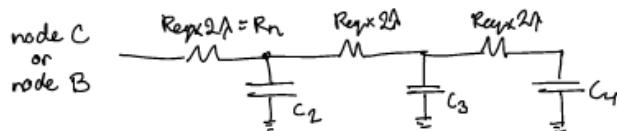


$$RC = \tau \approx R_{eq} \times 2A \times (8C_{eff} + 2AC_g)$$

$$\text{assuming: } R_{eq} = 12.5k/\square \quad C_{eff} = 1fF/\mu m \\ \lambda = 0.1\mu m \quad C_g = 2fF/\mu m$$

$$\tau \approx 12.5k \times 2 \times 0.1\mu m \times (8 \times 0.1 \times 1f + 2 \times 0.1 \times 2f) \\ \approx 12 \text{ ps}$$

longest delay:



$$RC \approx \tau = R_n(C_2 + C_3 + C_4) + 2R_n(C_3 + C_4) + 3R_n(C_4)$$

$$= R_n [C_2 + 3C_3 + 6C_4]$$

$$= R_{eq} \times 2A \cdot [10A C_{eff} + B2A C_g + 36A C_{eff} + 6A C_g + A'6AC_g + 48AC_{eff} + 12AC_g]$$

$$= R_{eq} \times 2A [94A C_{eff} + 26A C_g]$$

$$= 12.5k \times 2 \times 0.1 [94 \times 0.1 \times 1f + 26 \times 0.1 \times 2] \approx 36.5 \text{ ps}$$

