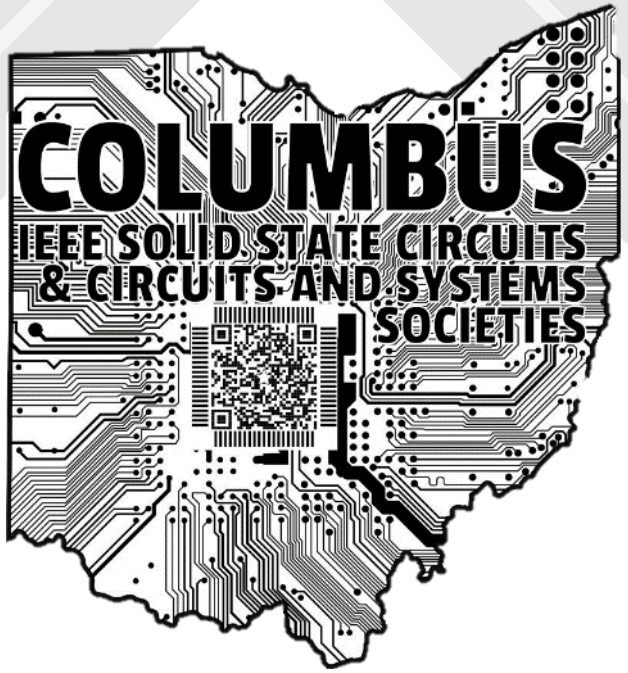


IEEE

SSCS IEEE SOLID-STATE CIRCUITS SOCIETY™
IC Innovation

Columbus Joint Chapter
SSC37 / CAS04

Presentation will begin at 6:15 EST

CAS
IEEE CIRCUITS AND SYSTEMS SOCIETY

SSC37/CAS04
Columbus Joint-Chapter Seminar

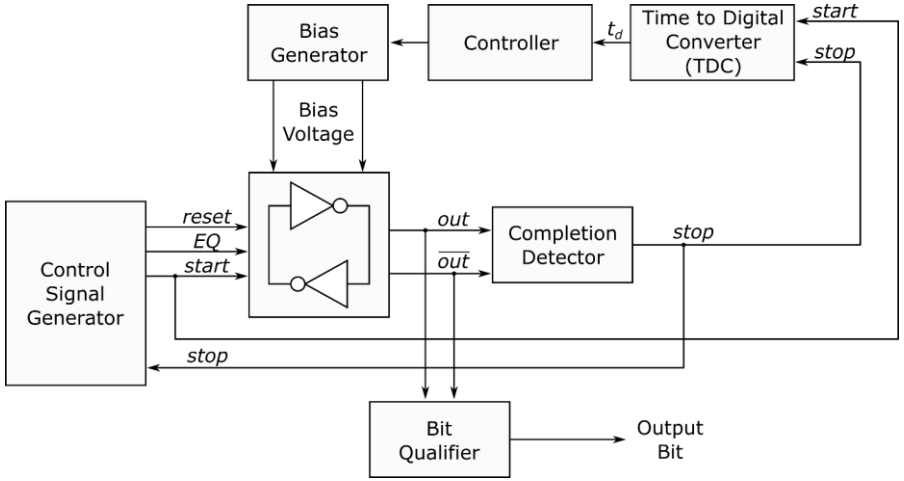
A Practical Approach to Lab-to-Fab series: Tiny Tapeout Workshop (2)

Webinar Location: SenselCs Corporation @ Rev1 Ventures Main Conference Room

Speaker: Sam Ellicott

About Me

- IEEE Columbus SSCS/CAS Webmaster
- PhD Student at The Ohio State University
  - Circuits Laboratory for Advanced Sensors and Systems
  - RF and Mixed-Signal Integrated Circuit (IC) Design
  - True Random Number Generators
- BSEE at Cedarville University (2019)
- Intern at Analog Devices



Workshop Series Goals

- Understand the workflow for open-source tools
  - Demystify the steps required to generate digital designs
- Hands-on introduction to digital design
  - Crash course to Verilog
  - Ability to design/test simple modules
  - Make a simple project
- Have fun!

Workshop Series Outline

- Workshop 1: Introduction
  - Last Semester
- Workshop 2: Tooling and Series Project
  - Today
- ~~Workshop 3: Design Review~~
  - ~~October/November~~

Workshop Series Outline

- Workshop 1: Introduction
  - What are Integrated Circuits (ICs)
  - Brief History of ICs
  - Introduction to Digital Design
  - Introduction to Tiny Tapeout
- Workshop 2: Tooling and Series Project

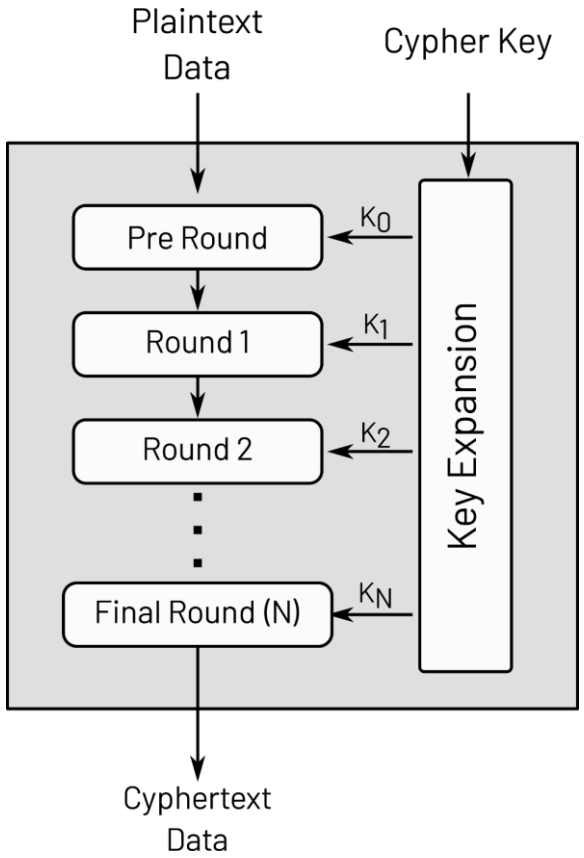
Workshop Series Outline

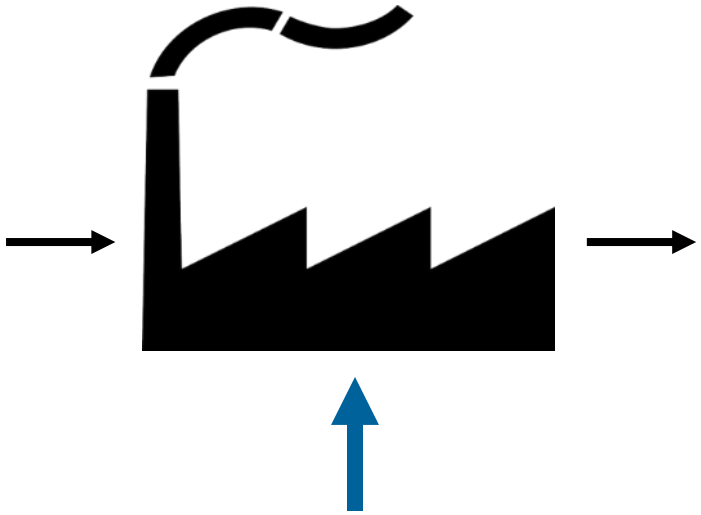
- Workshop 1: Introduction
- Workshop 2: Tooling and Series Project (tonight)
  - Review of Workshop 1
  - Set up a Tiny Tapeout project
  - Write simple Verilog modules
  - Write simple Verilog testbenches

Background/Review

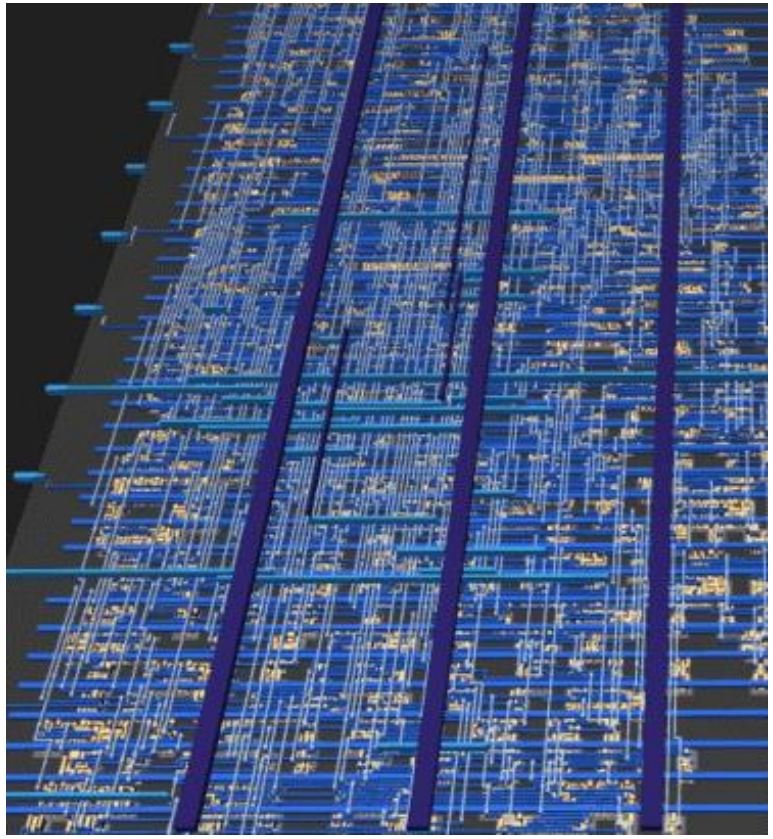
What are we trying to do?

- Concept (algorithm, behavior, etc) → Physical design (gates)





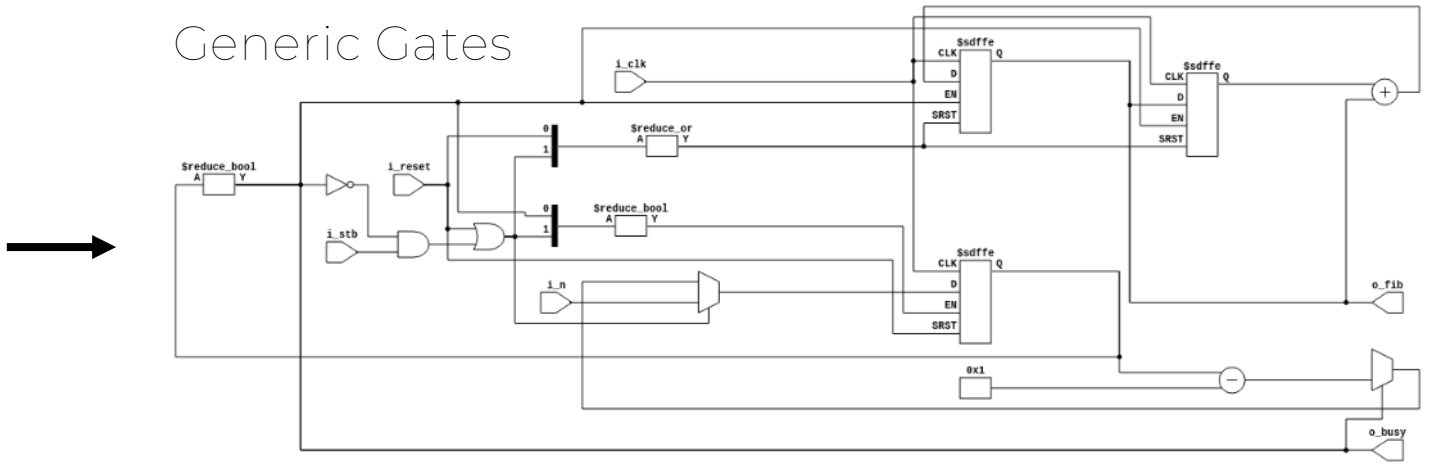
Digital Design Flow (OpenLane)

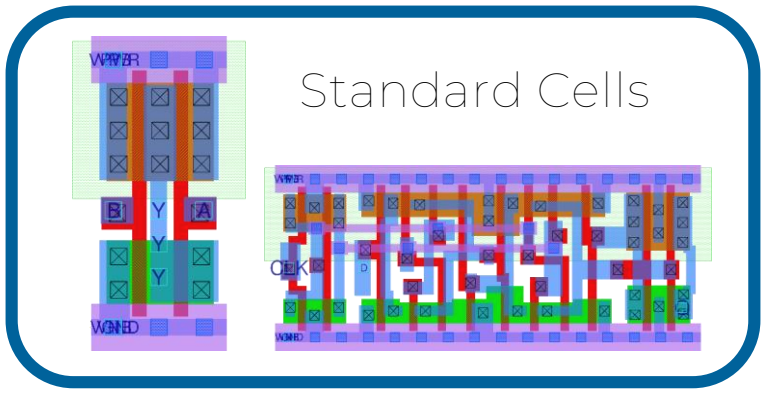


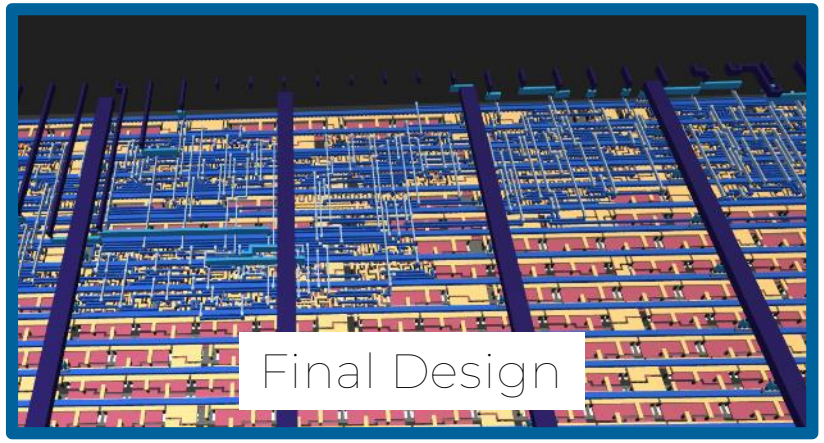
Recap: Digital Design

- Concept (algorithm, behavior, etc) → Physical design (gates)

```
reg [WIDTH-1:0] iteration;
reg [WIDTH-1:0] prev;
reg [WIDTH-1:0] current;

assign o_busy = (iteration != RESET);
assign o_fib  = current;

always @(posedge i_clk) begin
    if (i_reset || (!o_busy && i_stb)) begin
        iteration <= i_n;
        prev    [WIDTH-1:0] <= 1;
        current [WIDTH-1:0] <= 0;
    end
    else if (o_busy) begin
        iteration <= iteration - ONE;
        current   <= prev + current;
        prev      <= current;
    end

    if (i_reset) begin
        iteration <= RESET;
        prev    [WIDTH-1:0] <= 1;
        current [WIDTH-1:0] <= 0;
    end
end

endmodule
~
NORMAL  fib.v
```





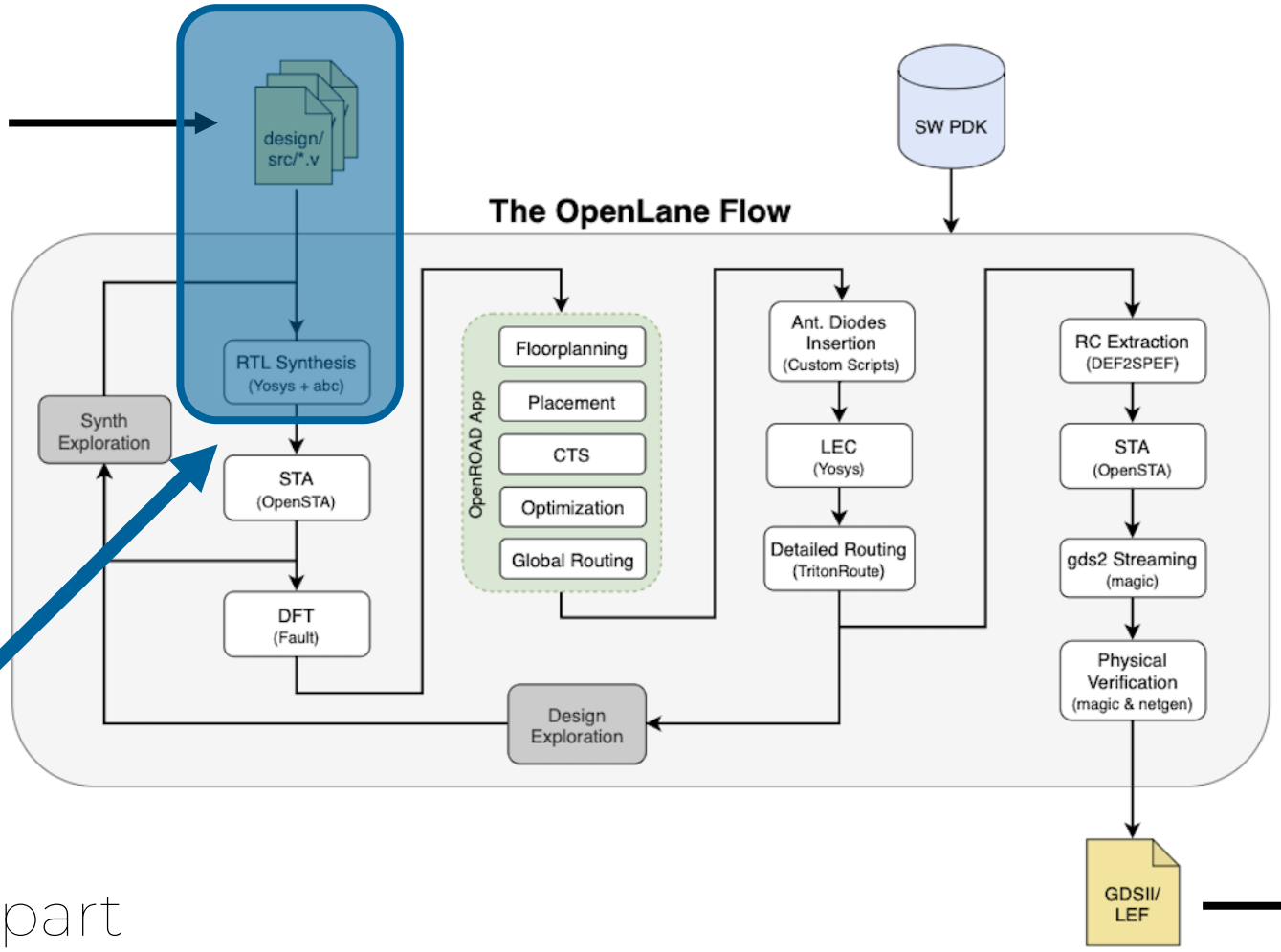


OpenLane Flow

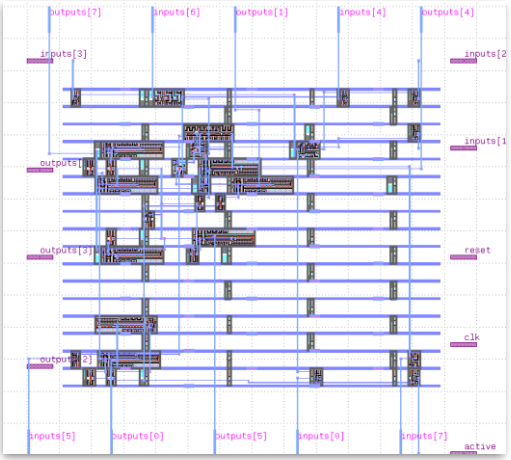
WOKWi

or

Verilog Files



Output Design (GDS File)

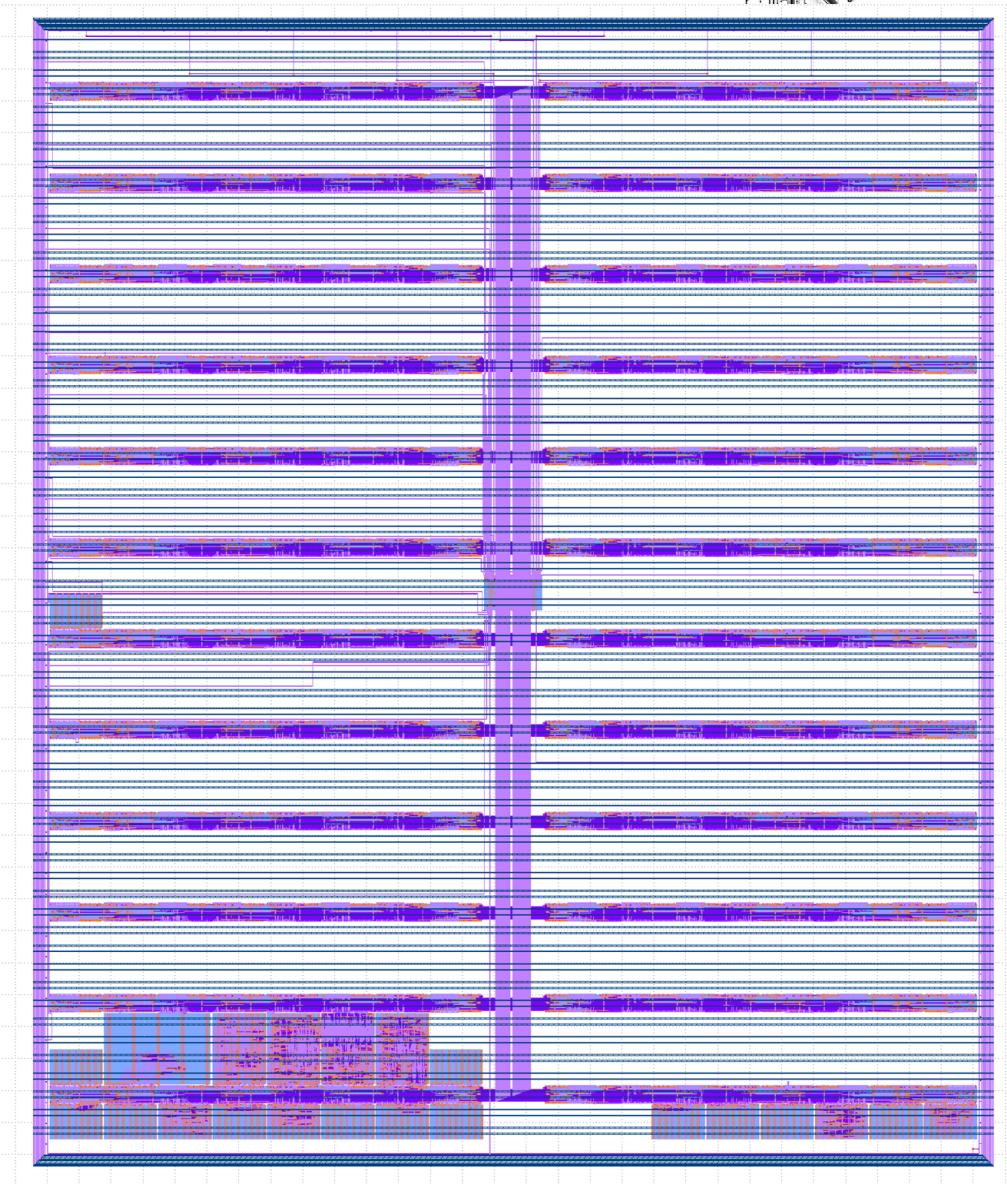


We're mostly dealing with this part

Tiny Tapeout (via Openlane) handles most of this for us!

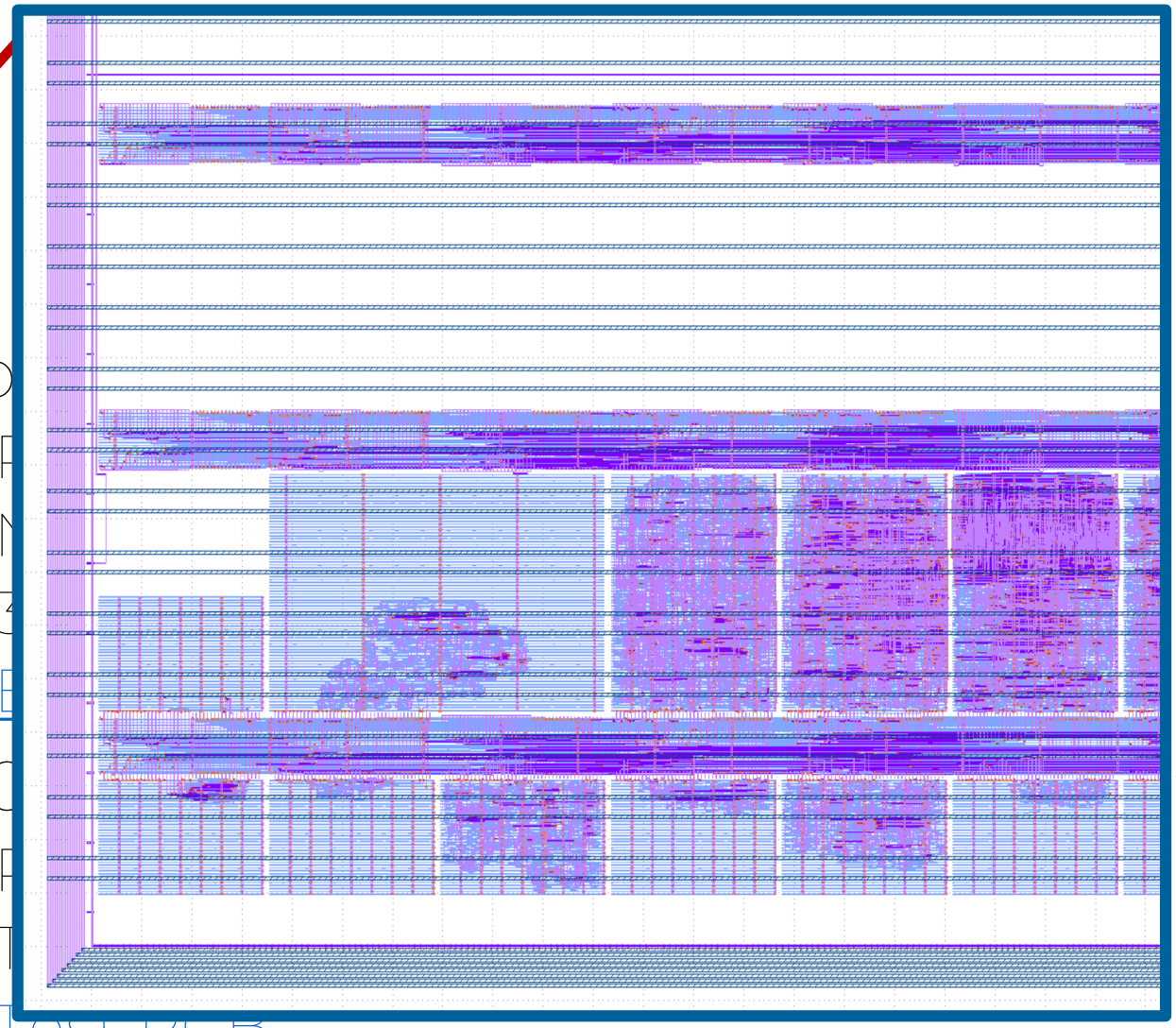
How Tiny Tapeout Works

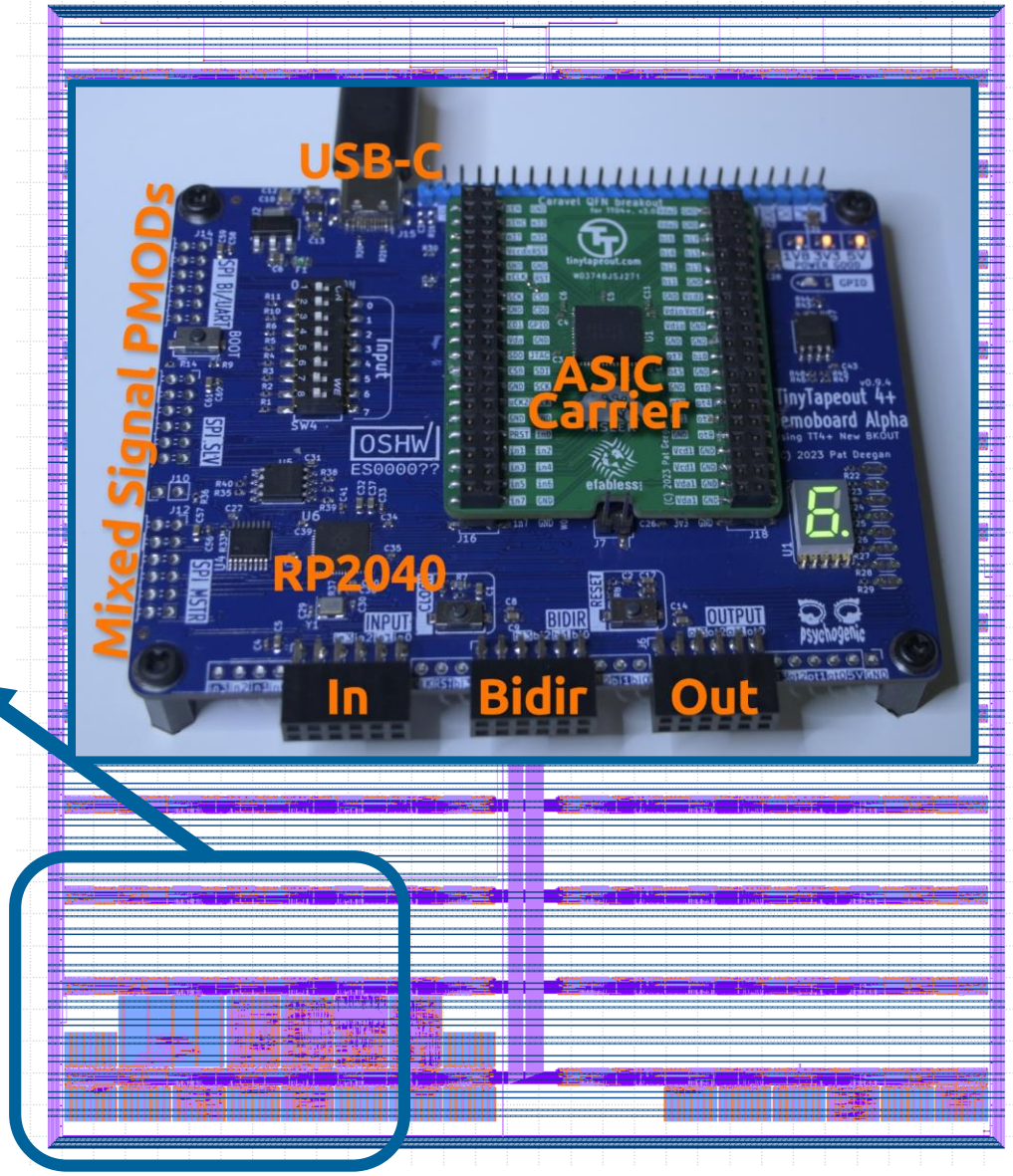
- Cloud based design
  - Runs OpenLane in Github actions
  - No tool install or download
  - 3D viewer / explorer
  - Example Design
- ~500 Projects merged into one IC
  - Reduced cost
  - Try other peoples designs
  - Test PCB



Hov

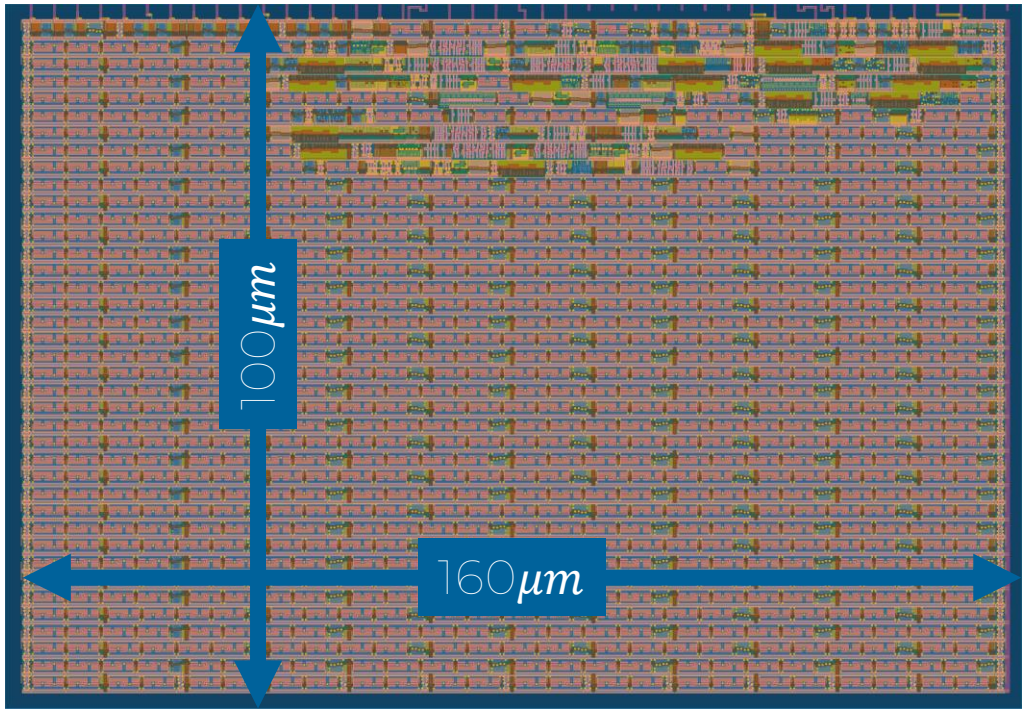
- Clo
  - F
  - M
  - 3
  - E
- ~50
  - F
  - T
  - Test PCB

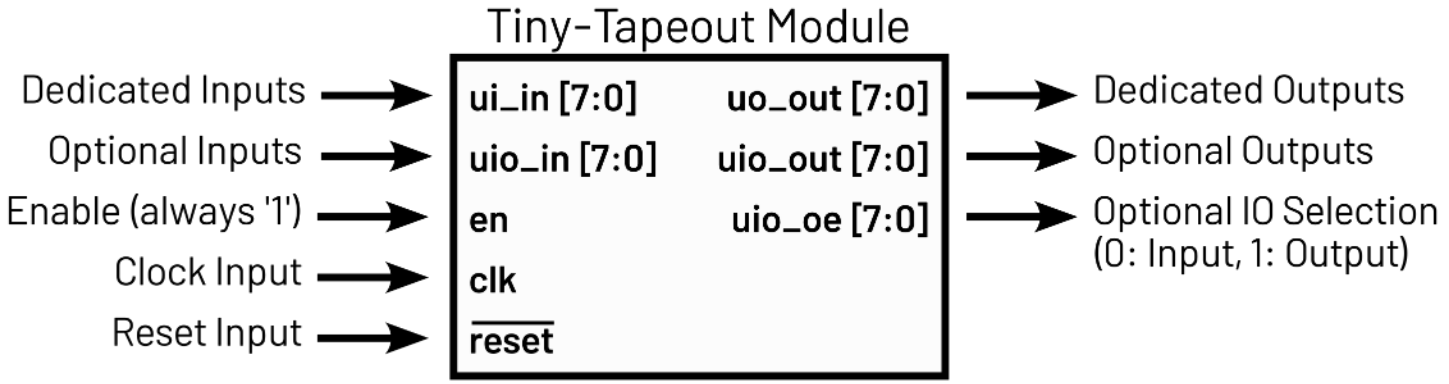


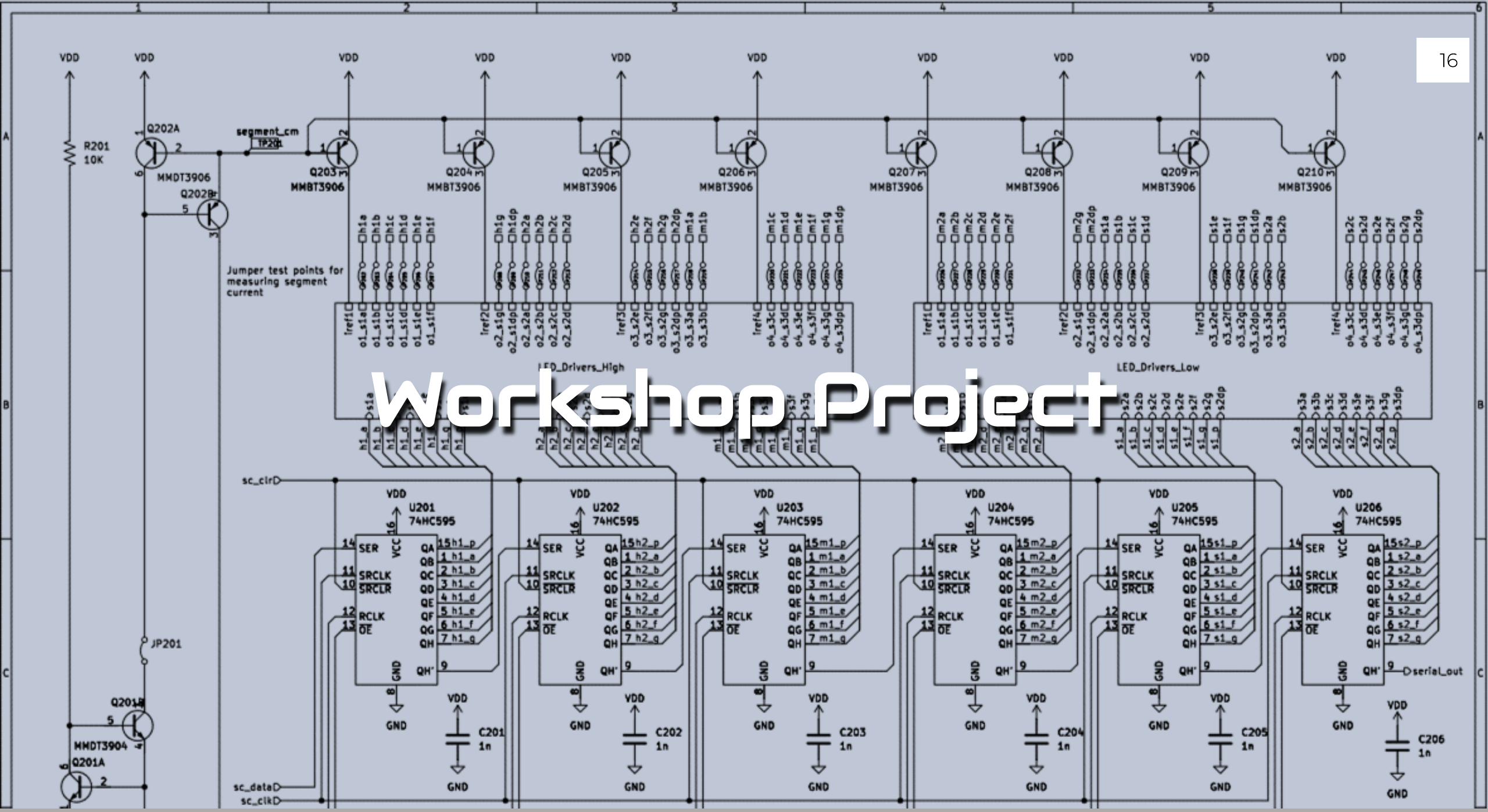


Physical Interface

- 1x Tile
  - 160 x 100μm size
  - ~1000 gates
  - Can buy more than 1 tile
- Pins
  - Clock (~50MHz)
  - Reset
  - 8x Inputs
  - 8x Outputs
  - 8x Bidirectional



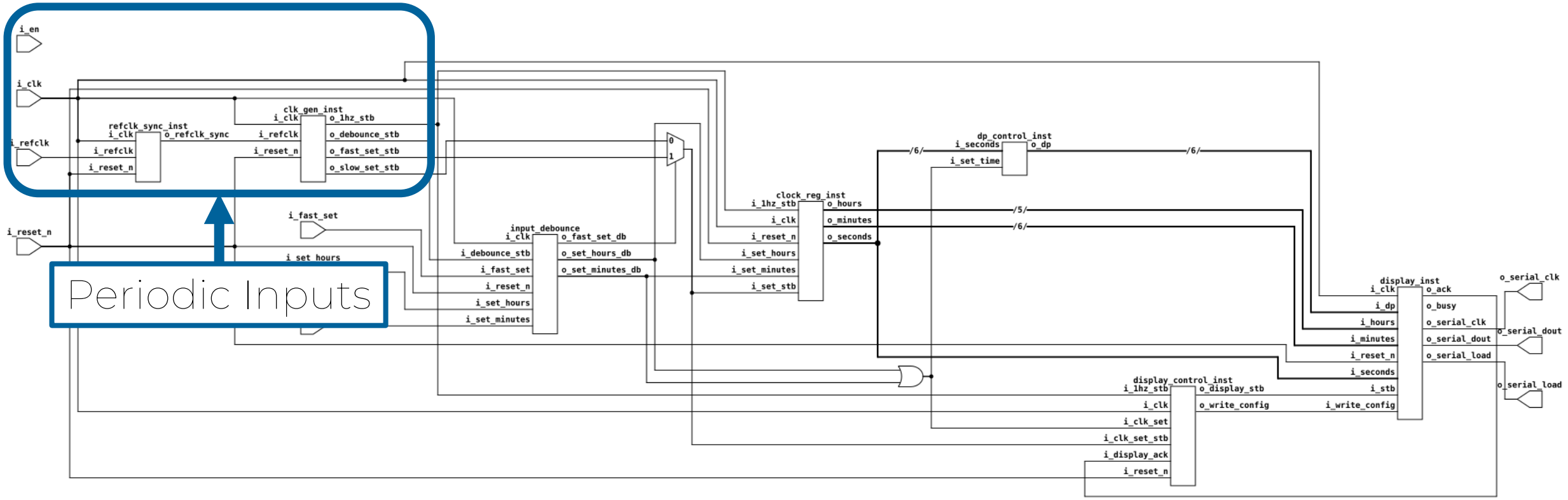



Workshop Project
VDD
R201
10K
Q202A
MMDT3906
Q202B
segment_cm
TP201
Q203
MMBT3906
Q204
MMBT3906
Q205
MMBT3906
Q206
MMBT3906
Q207
MMBT3906
Q208
MMBT3906
Q209
MMBT3906
Q210
MMBT3906
Jumper test points for measuring segment current
LED_Drivers_High
LED_Drivers_Low
JP201
Q201B
MMDT3904
Q201A
sc_clr
sc_data
sc_clk
U201
74HC595
U202
74HC595
U203
74HC595
U204
74HC595
U205
74HC595
U206
74HC595
SER
SRCLK
SRCLR
RCLK
OE
VCC
GND
QA
QB
QC
QD
QE
QF
QG
QH
QH'
C201
1n
C202
1n
C203
1n
C204
1n
C205
1n
C206
1n
serial_out

Project Introduction

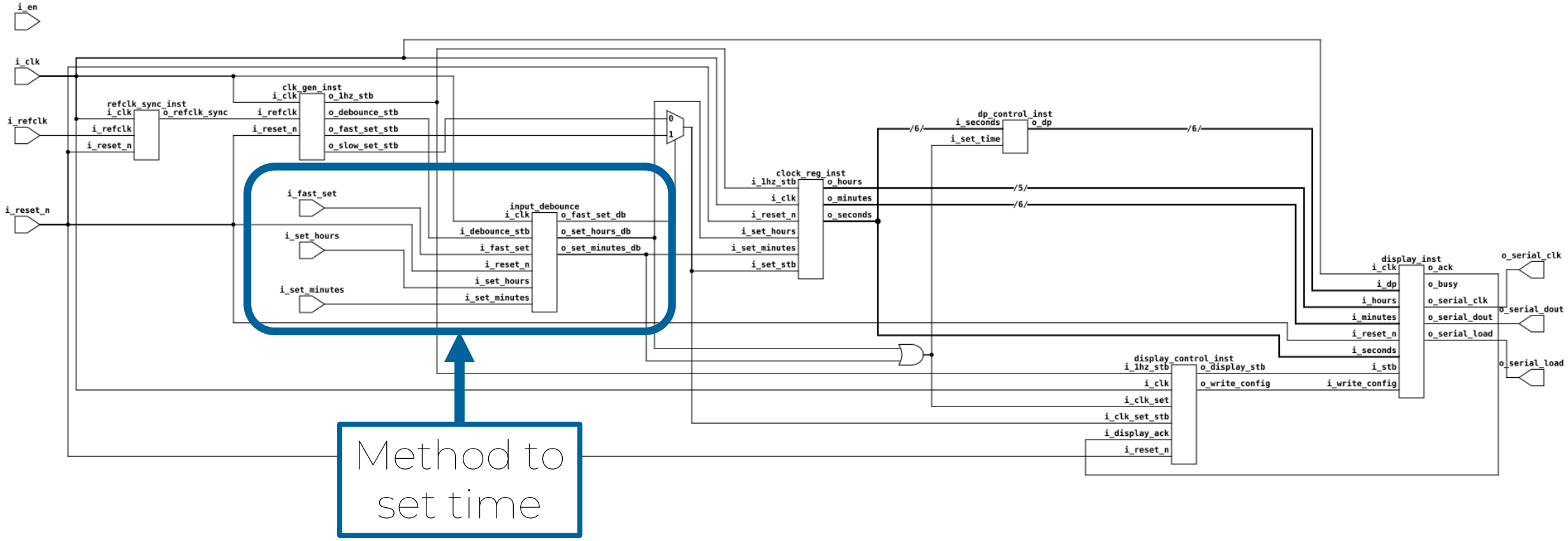
- Design a 7-Segment Digital Desk Clock
  - 24 Hour time format
  - Time set
  - 7-Segment Display Output
  - Easily extensible for customization
- Broken into modules
  - Some provided
  - Some are yours to complete ☺
- Testbenches for each module

Desk Clock Block Diagram Overview



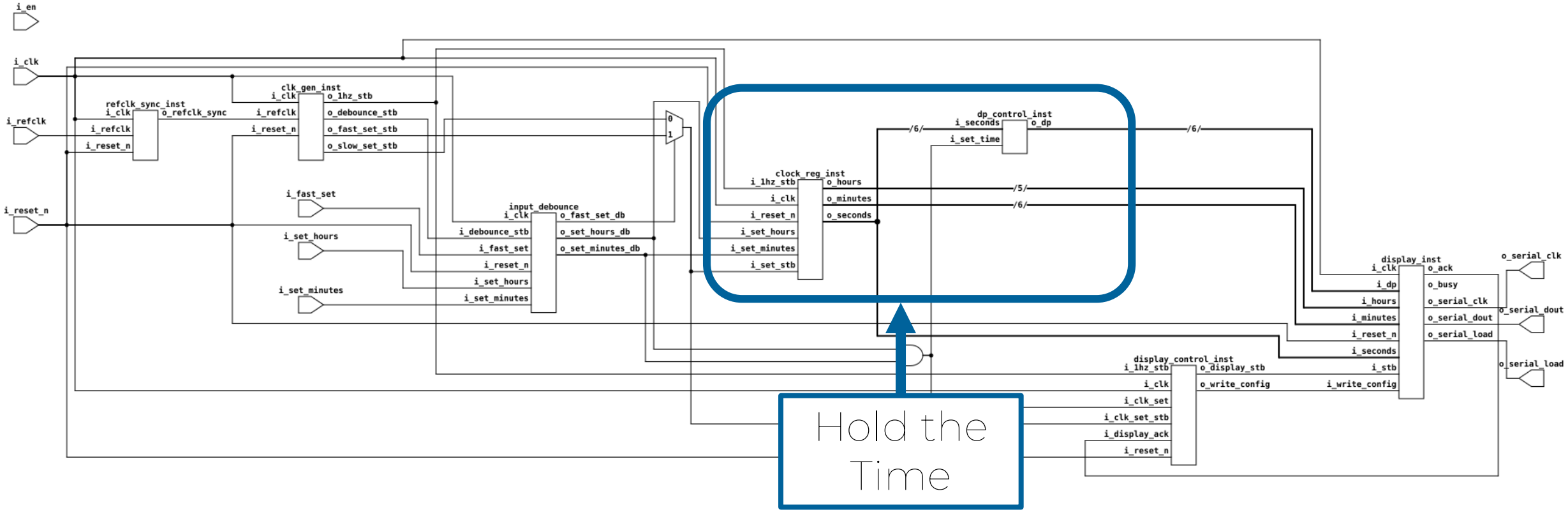
- Generate “strobe” signals to trigger update events
  - Strobe signal is a single system clock wide pulse

Desk Clock Block Diagram Overview



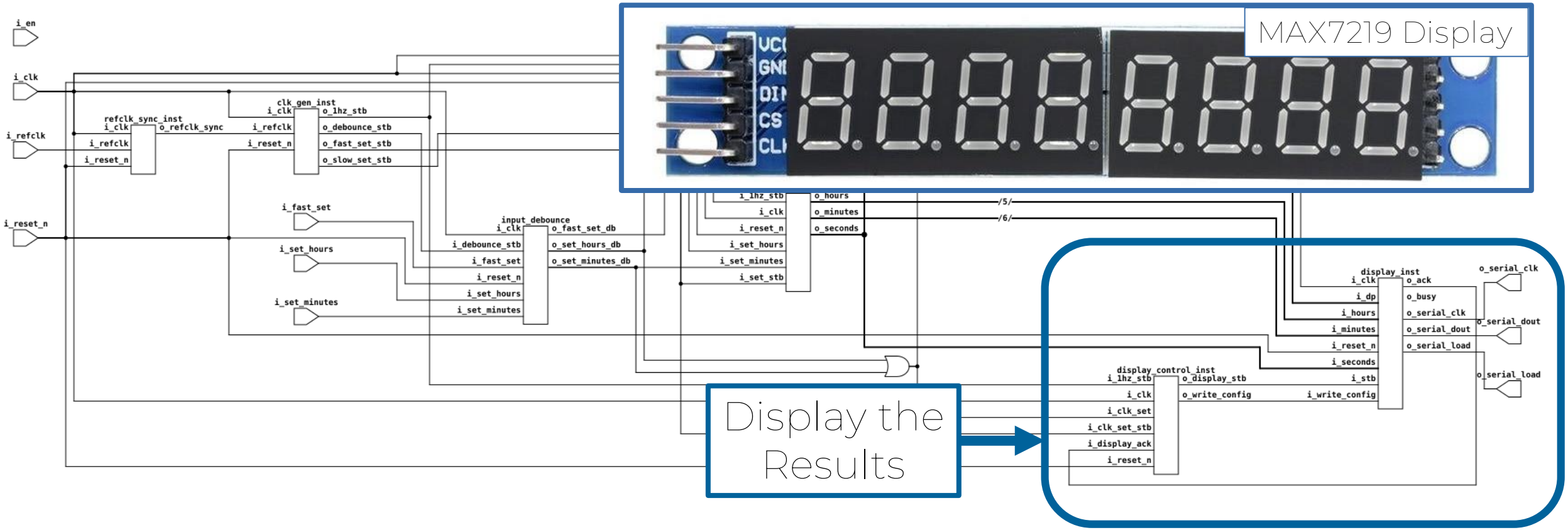
- Synchronize and Debounce Button Inputs
  - Clean up noisy signals

Desk Clock Block Diagram Overview



- Hold the time in 24-Hour format
- Wrap Hours, Minutes, Second appropriately
- Handle time setting functionality

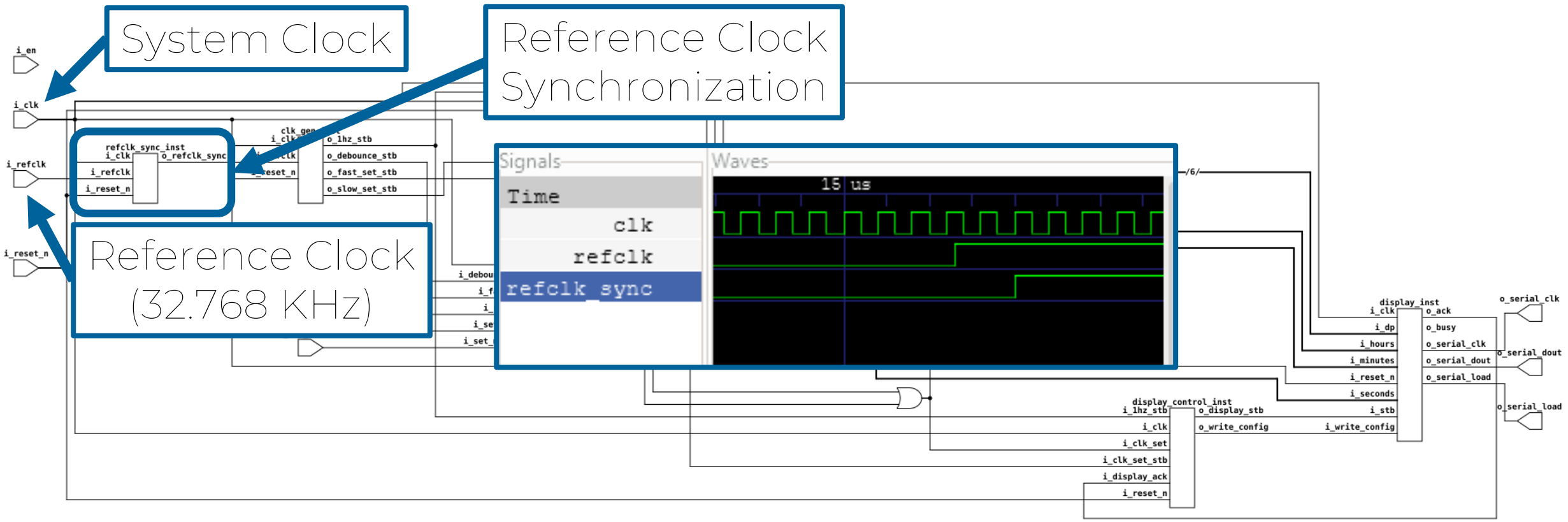
Desk Clock Block Diagram Overview



- Shift out time over SPI
  - Drive MAX7219 Display

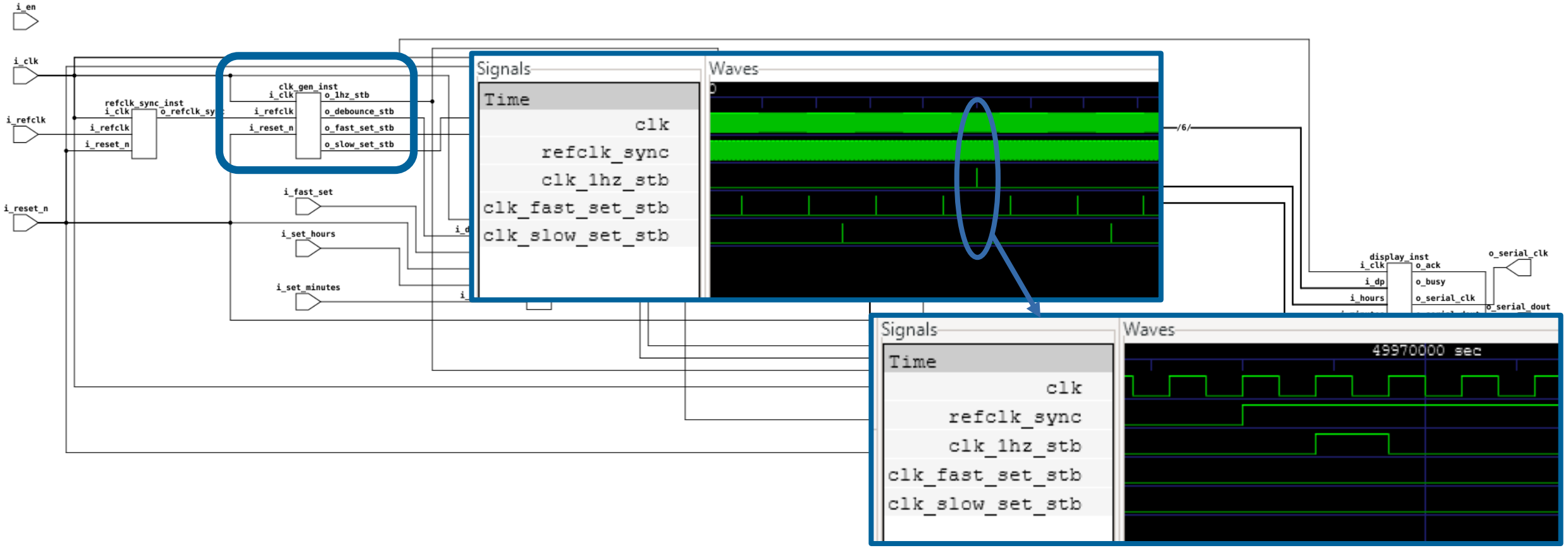
https://www.diymore.cc/products/max7219-7-segment-led-dot-matrix-8-digit-digital-tube-display-control-module-for-arduino-3-3v-5v-microcontroller-serial-driver

Desk Clock Block Diagram



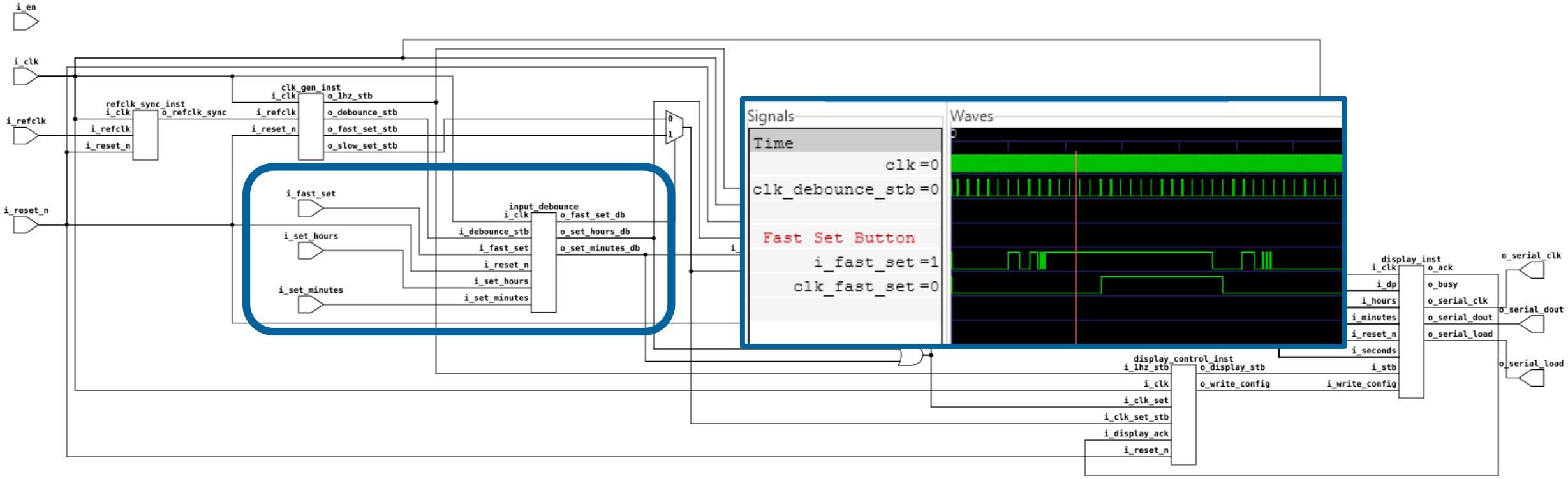
- Everything runs off a single system clock
- Reference clock provides accurate timing ($2^{15}Hz$)

Desk Clock Block Diagram



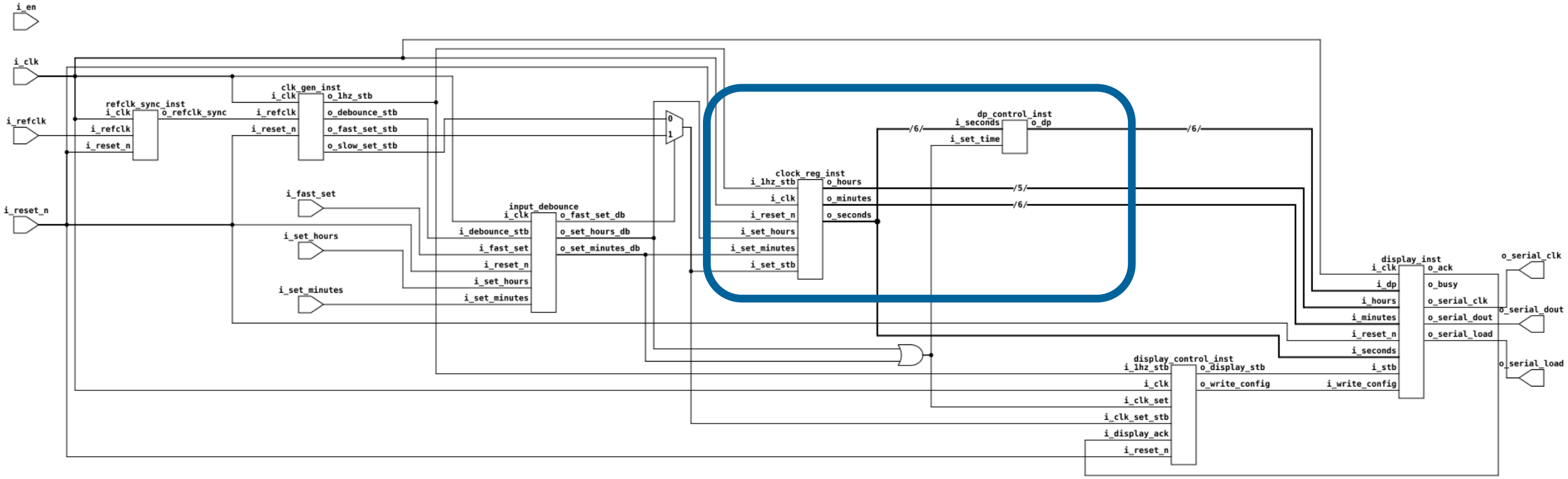
- Generate “strobe” signals to trigger update events
  - Strobe signal is a single system-clock wide pulse

Desk Clock Block Diagram



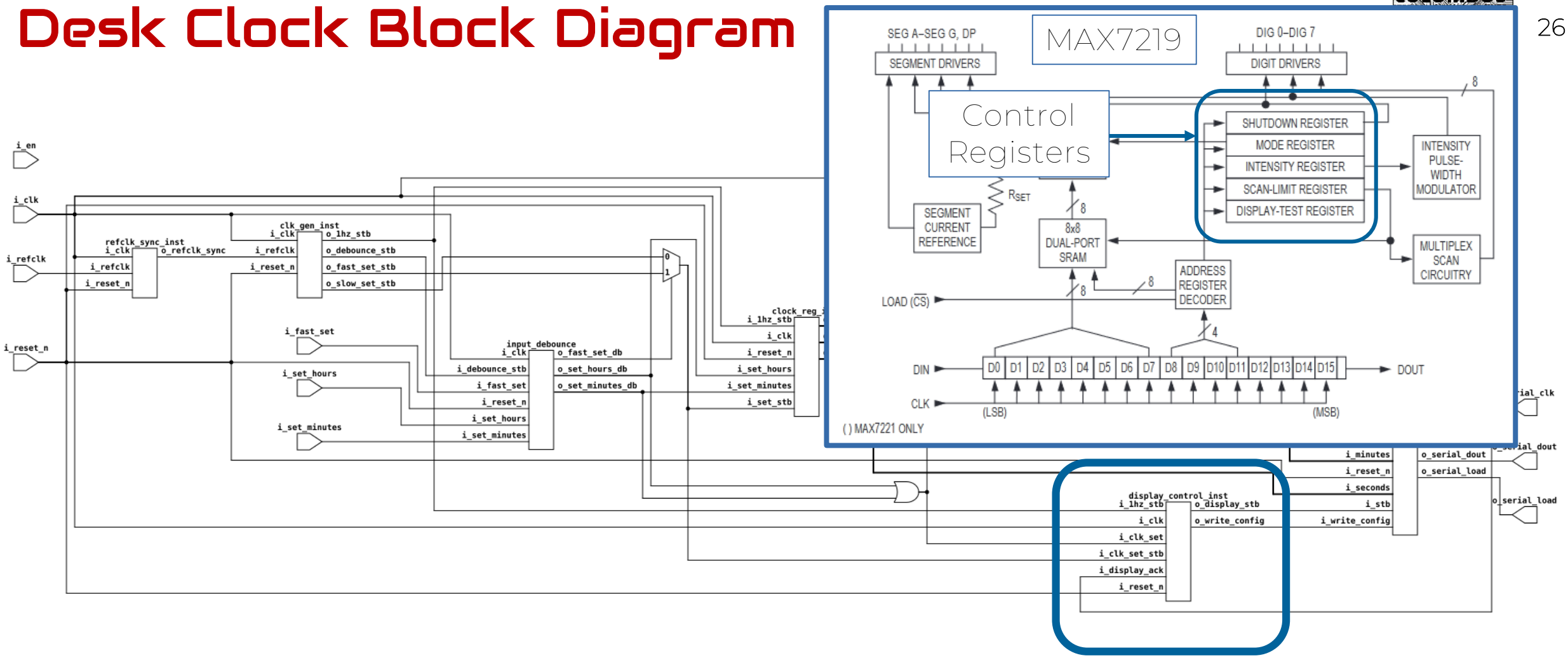
- Synchronize and Debounce Button Inputs
  - Clean up noisy signals

Desk Clock Block Diagram Overview



- Hold the time in 24-Hour format
- Wrap Hours, Minutes, Second appropriately
- Handle time setting functionality

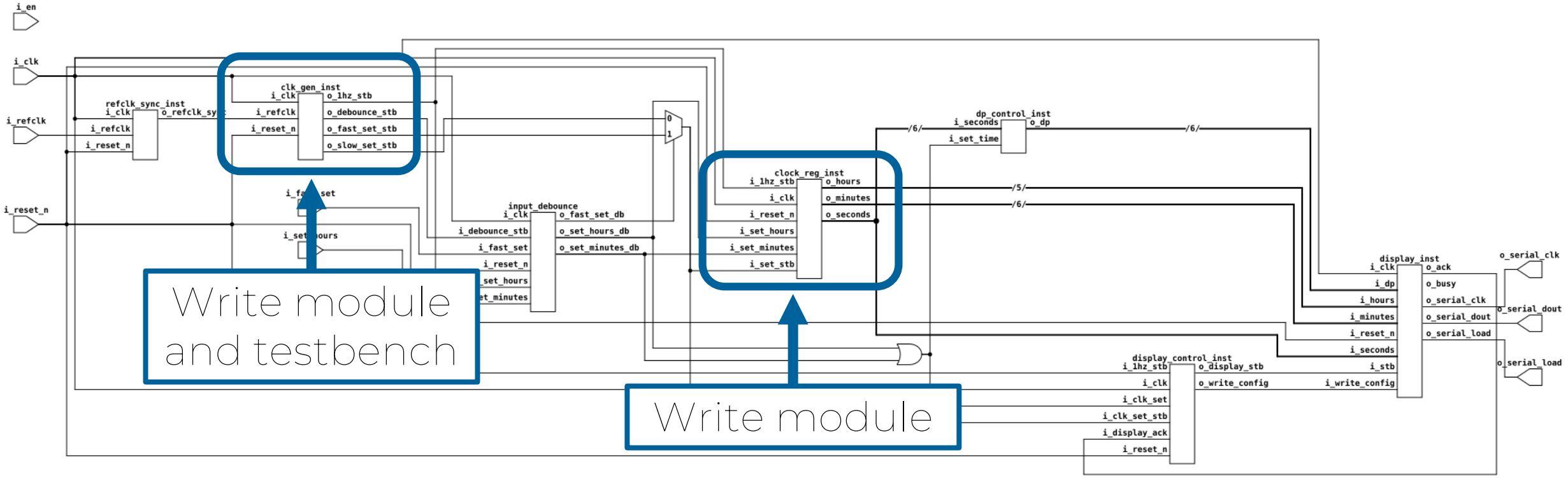
Desk Clock Block Diagram



- MAX7219 Needs Configured
  - Write display settings on startup

https://www.analog.com/media/en/technical-documentation/data-sheets/MAX7219-MAX7221.pdf

What are we doing?

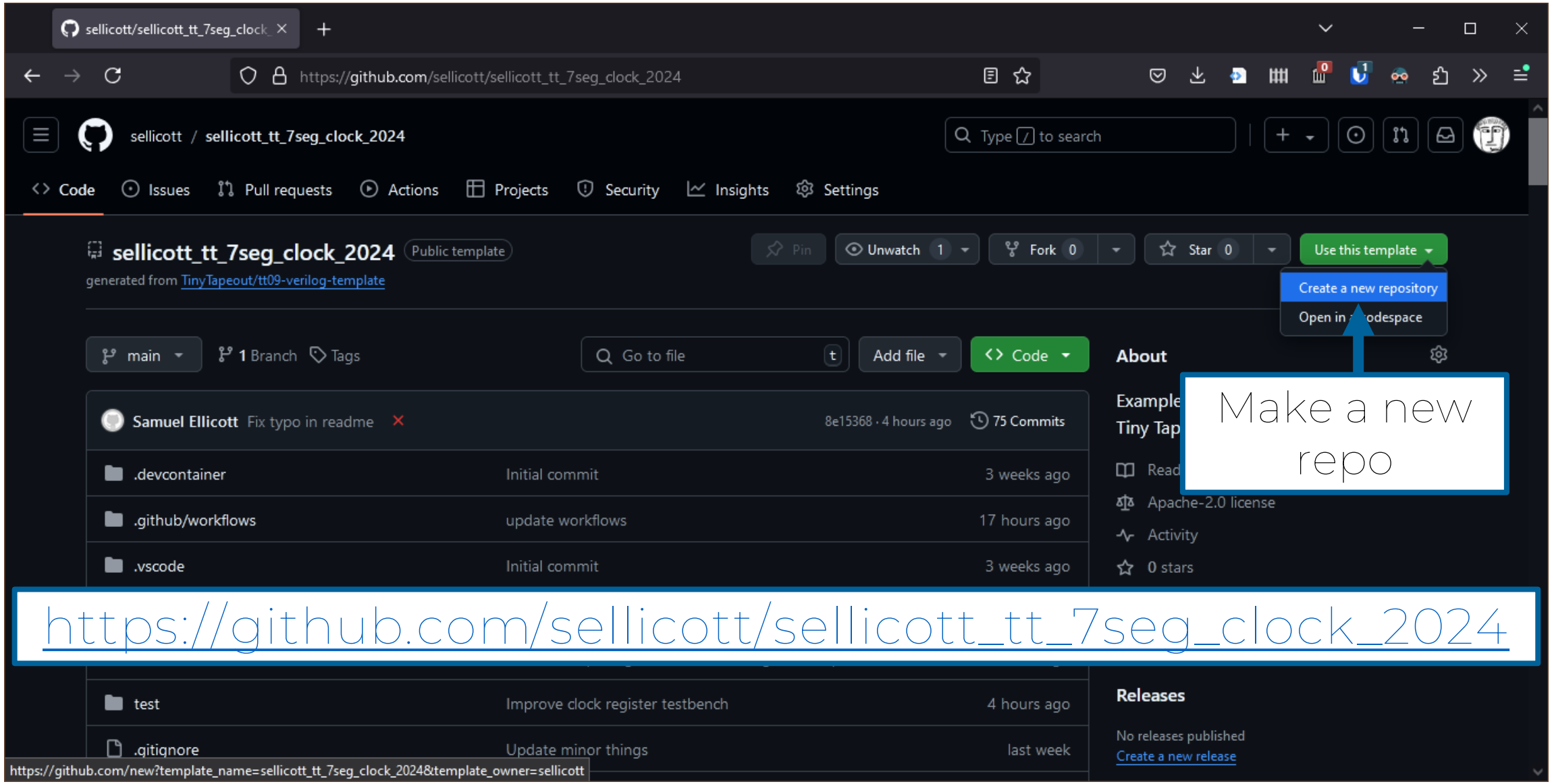


- Implement a few blocks
  - Strobe signal generator
  - Clock register

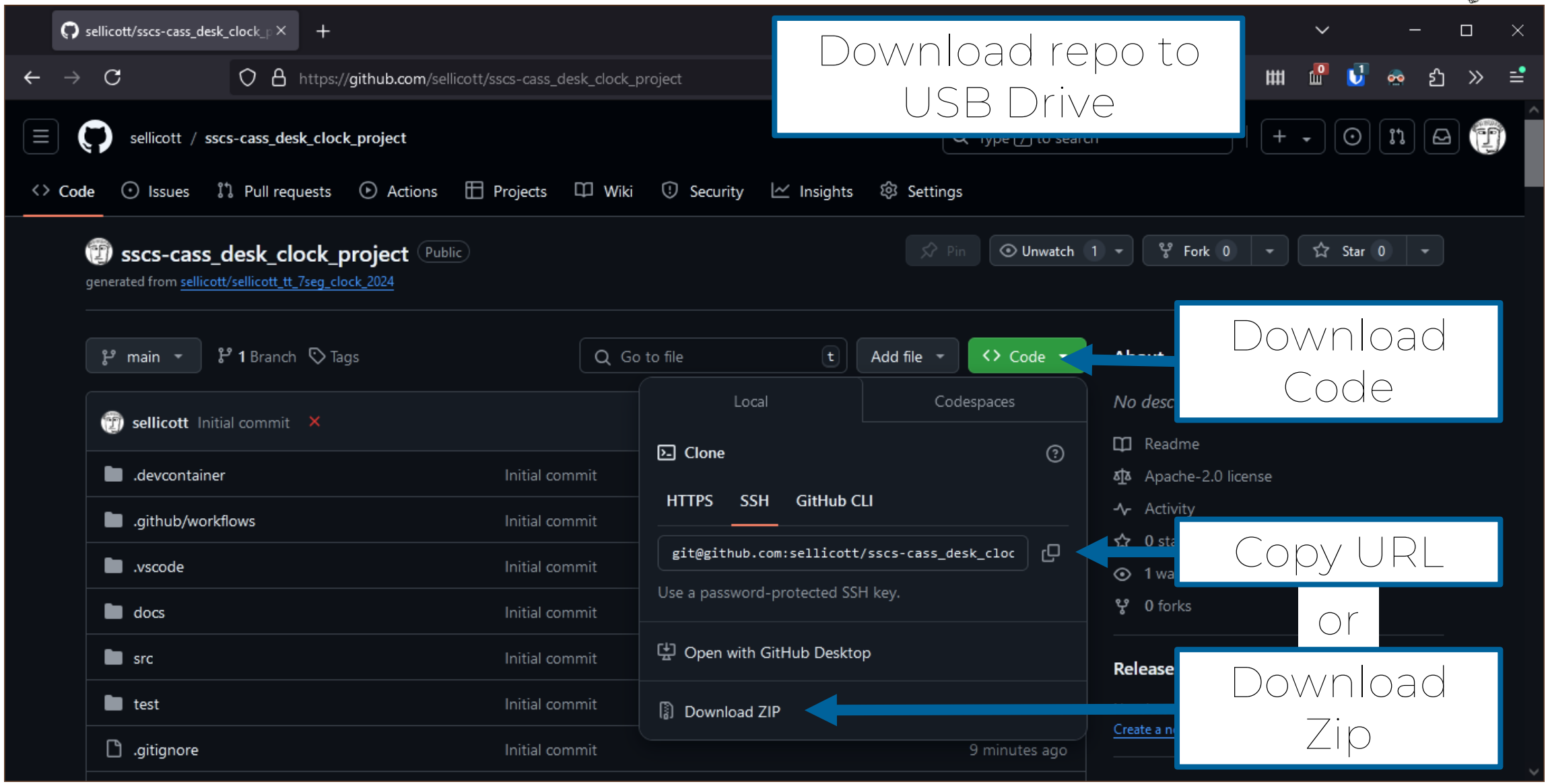
Where do we go from here?

1. Tools/Project Setup
2. Write “Strobe Generation” Module
   - Verilog Crash Course
   - Generating Block Diagrams
3. Write “Strobe Generation” Testbench
   - View results in gtkwave
4. Repeat for “Clock Register” (If we have time)
   - Testbench is already done for this module
5. Run Tiny Tapeout flow to generate hardware

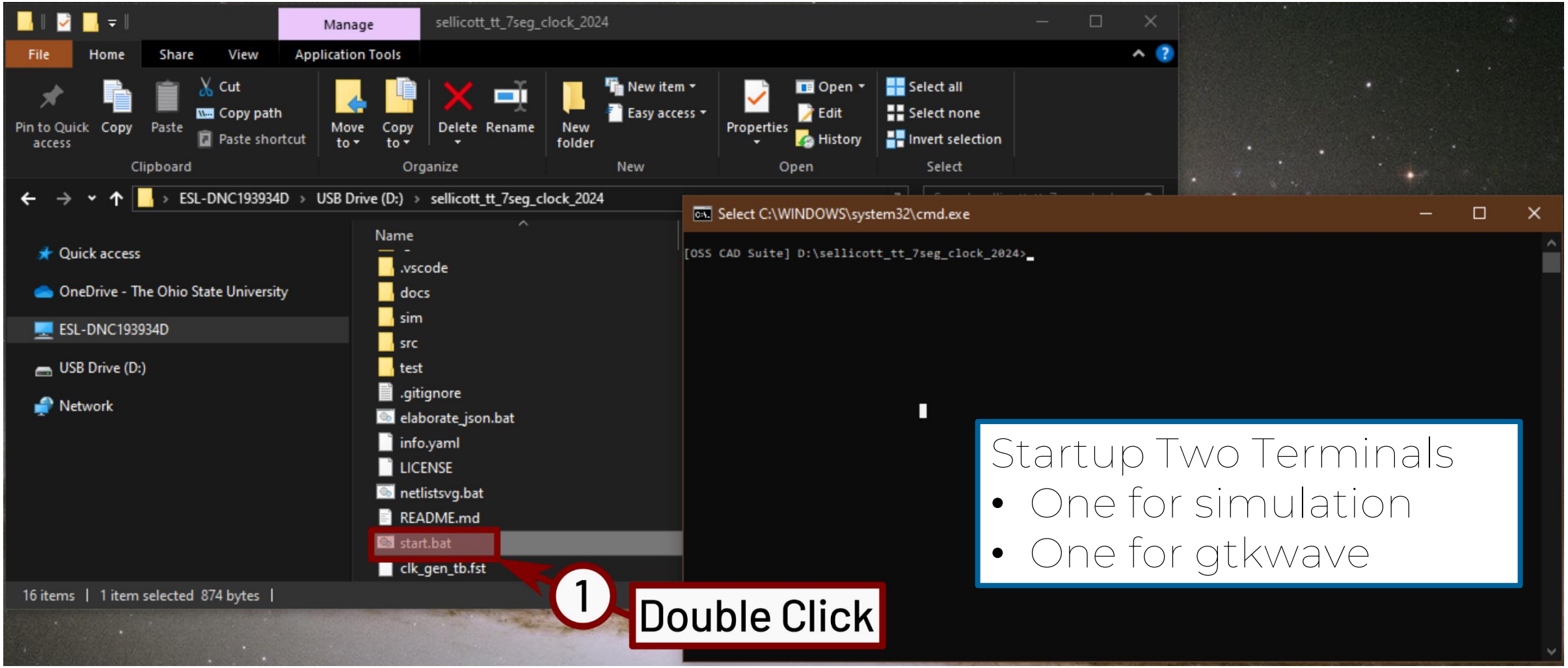
Start a Project (1/2)



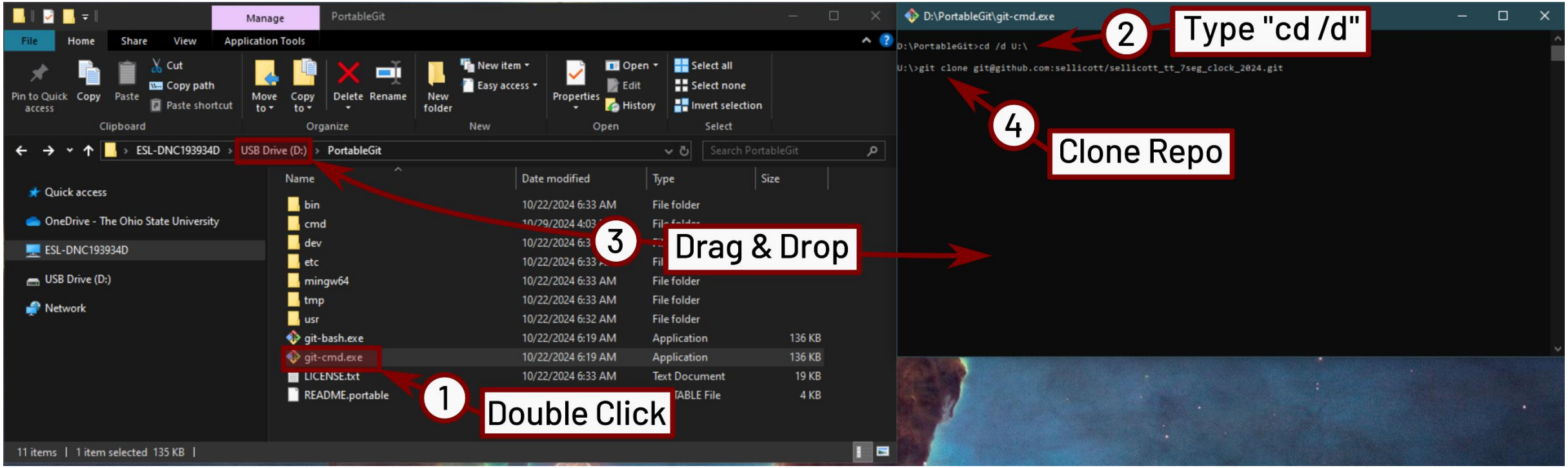
Start a Project (2/2)



Start up tools



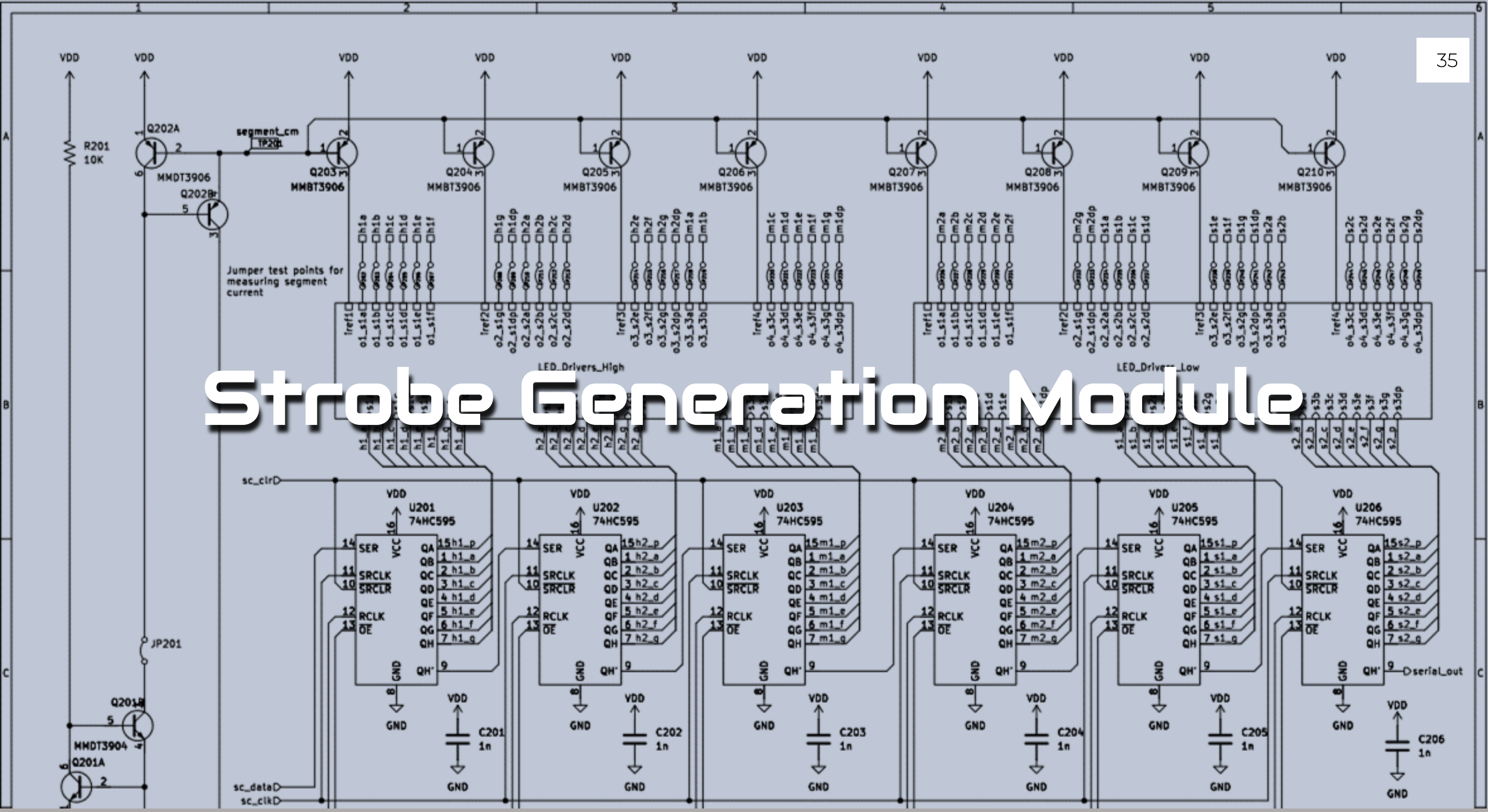
Clone the repo



https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens

Modify Project

- Look at github project page
  - Look at log messages to see why builds are failing
  - Setup github pages
- Move template files to their base names
  - Move `src\input\clk_gen_template.v` to `src\input\clk_gen.v`
  - Move `src\core\clock_register_template.v` to `src\core\clock_register.v`


Strobe Generation Module
VDD
R201
10K
Q202A
MMDT3906
Q202B
segment_cm
TP201
Q203
MMBT3906
Q204
MMBT3906
Q205
MMBT3906
Q206
MMBT3906
Q207
MMBT3906
Q208
MMBT3906
Q209
MMBT3906
Q210
MMBT3906
Jumper test points for measuring segment current
LED_Drivers_High
LED_Drivers_Low
sc_clr
U201
74HC595
U202
74HC595
U203
74HC595
U204
74HC595
U205
74HC595
U206
74HC595
SER
SRCLK
SRCLR
RCLK
OE
VCC
GND
QA
QB
QC
QD
QE
QF
QG
QH
QH'
C201
1n
C202
1n
C203
1n
C204
1n
C205
1n
C206
1n
JP201
Q201B
MMDT3904
Q201A
sc_data
sc_clk
serial_out

Writing a Verilog Module

- We are going to write the strobe generation module

- Generate 1-clock wide pulses to trigger updates
  - Allow everything to run off the system clock
  - Rest of the system uses these strobe pulses

- Goals for this section
  - Learn to write a Verilog module
  - Learn how to generate block diagrams
  - Learn how to run Icarus verilog

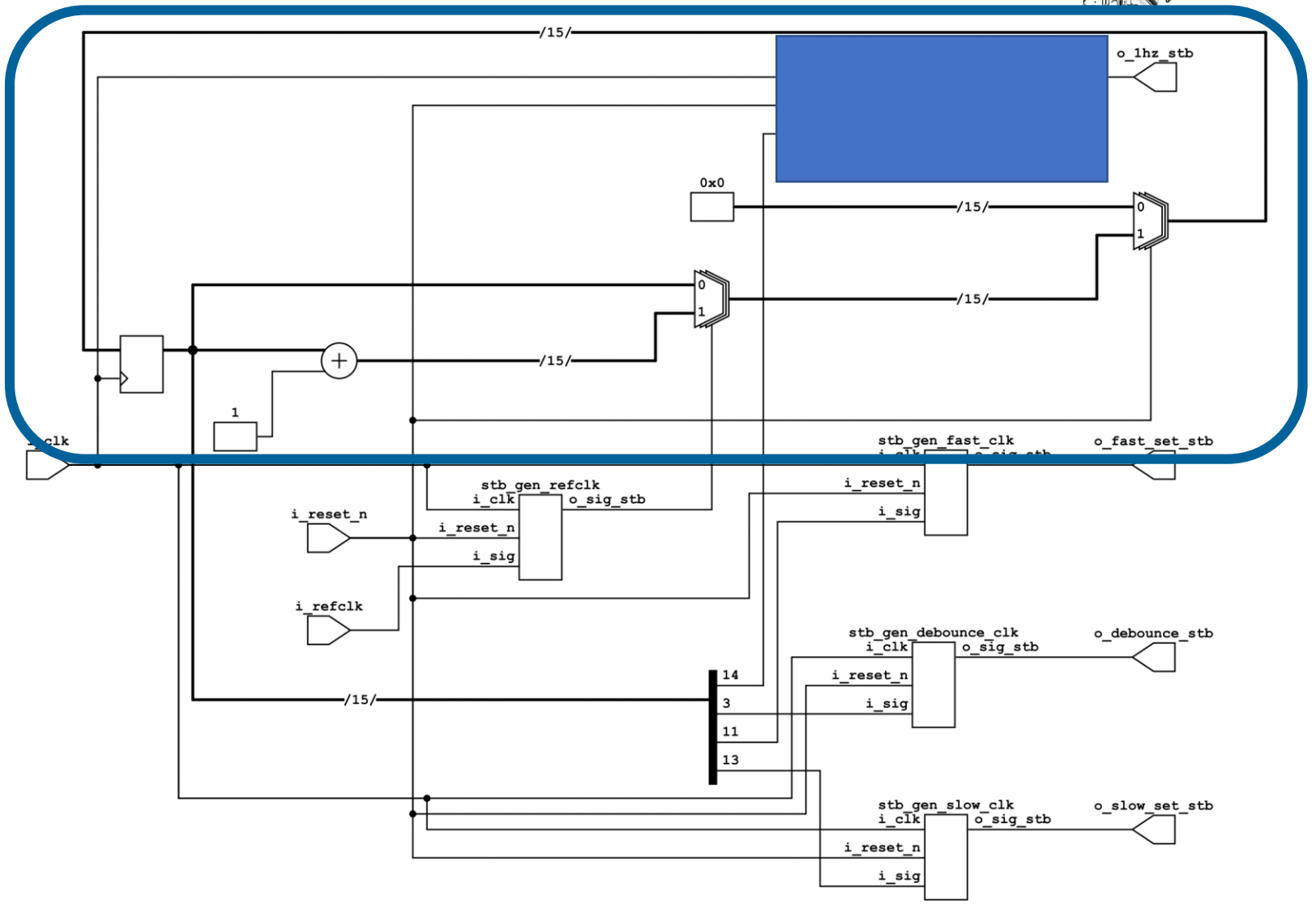
Writing a Verilog Module

- Open `src\input\clk_gen.v`
- We have a defined set of IO Pins
  - We want to generate strobe signals
  - Base the strobe signals off the refclk
- How do we do this?
  - Before we write code, what should the hardware look like?

```
`default_nettype none

module clk_gen (
    // global signals
    i_reset_n,
    i_clk,
    // Strobe from 32,768 Hz reference clock
    i_refclk,
    // output strobe signals
    o_1hz_stb,      // refclk / 2^15 -> 1Hz
    o_slow_set_stb, // refclk / 2^14 -> 2Hz
    o_fast_set_stb, // refclk / 2^12 -> 8Hz
    o_debounce_stb  // refclk / 2^4  -> 4.096KHz
);

input wire i_reset_n;
input wire i_clk;
input wire i_refclk;

output wire o_1hz_stb;
output wire o_slow_set_stb;
output wire o_fast_set_stb;
output wire o_debounce_stb;

// TODO: Implement strobe output signals

endmodule
```

The Completed Block (diagram)

- Looks daunting
- Break it down
  - Counter
  - Strobe generators
- How do we write a counter?



Crash course in Verilog

- For more detailed tutorials
  - NandLand
  - ChipVerify
  - ZipCPU
  - Verilog Reference Card (pdf)
  - Verilog Reference Manual (pdf)
  - Verilog RTL Synthesis Standard (pdf)
- For now you have me…
  - This will only cover the bare necessities for today's project
  - I hope to motivate you to learn more

Verilog Basics

- What are we going to talk about
  - Modules
  - Sequential Logic (Flip-Flops)
  - Combinational Logic
- Preliminaries
  - Sort of "C like" syntax (if you squint)
    - Statements end with "**;**"
    - Case-sensitive names
    - Comments with "**//**" and "**/* stuff */**"
  - Blocks start/stop with **begin** and **end**
  - Give blocks a name using
    - **begin: block_name**

7.1. Unary Operators

+, -	Positive, Negative
!	Logical negation
~	Bitwise negation
&, ~&	Bitwise and, nand
|, ~|	Bitwise or, nor
^, ~^, ^~	Bitwise xor, xnor

7.2. Binary Operators

Increasing precedence:

?:	if/else
||	Logical or
&&	Logical and
|	Bitwise or
^, ^~	Bitwise xor, xnor
&	Bitwise and
==, != , ===, !==	Equality
<, <=, >, >=	Inequality
<<, >>	Logical shift
+, -	Addition, Substraction
*, /, %	Multiply, Divide, Modulo

From: https://www.ece.lsu.edu/v/refcard.pdf

Lets write a counter!

- Edit "**src\input\clk_gen.v**"
- Divide our input **refclk** signal by 2^{15}
  - Generate $1Hz$ clock signal
- We also need $\frac{\text{refclk}}{2^{14}}$, $\frac{\text{refclk}}{2^{12}}$, and $\frac{\text{refclk}}{2^{4}}$
  - Generates $2Hz$, $8Hz$, and $4.096KHz$ clocks respectively
  - Tap off output clocks from the counter
- We need a 15-bit register
  - Reset to 0 with reset signal
  - Count up by 1 on each rising edge of **refclk**
    - We will come back to this…
    - For now, count up on **i_clk** rising edge

Verilog Modules


```
`default_nettype none
```

- Start files with this

```
module clk_gen (
  // global signals
  i_reset_n,
  i_clk,
  // Strobe from 32,768 Hz reference clock
  i_refclk,
  // output strobe signals
  o_1hz_stb,      // refclk / 2^15 -> 1Hz
  o_slow_set_stb, // refclk / 2^14 -> 2Hz
  o_fast_set_stb, // refclk / 2^12 -> 8Hz
  o_debounce_stb  // refclk / 2^4  -> 4.096KHz
);
```

- Module Port List
- List of names separated by commas
- By (my) convention
  - Inputs are **i_<name>**
  - Outputs are **o_<name>**

```
input wire i_reset_n;
input wire i_clk;
input wire i_refclk;

output wire o_1hz_stb;
output wire o_slow_set_stb;
output wire o_fast_set_stb;
output wire o_debounce_stb;

// TODO: Implement strobe output signals

endmodule
```

- Define Signal Wires/Registers
- Signals in the port list are prefixed with
  - **input** or **output**
- We can assign to wires when we create them
- We can define busses too
  - **wire [msb:lsb] bus_name;**

- In general:

```
module module_name ( list_of_signals );
// stuff
endmodule
```

Multi-Bit Signals

- Busses (groups of signals)
  - `wire [msb:lsb] bus_name;`
  - `reg  [msb:lsb] bus_name;`
- Accessing Busses
  - `bus_name[bit]`
  - `bus_name[msb:lsb]`
  - `bus_name[bit1, bit2, msb:lsb]`
- Grouping Signals
  - `wire [2:0] signal_group = {sig_2, sig_1, sig_0};`
- Fixed Width Constants
  - Format: **<width>'<format><value>**
  - **<width>** is a decimal number > 0
  - **<format>** is a letter
    - **b** for binary
    - **o** for octal
    - **d** for decimal
    - **h** for hexadecimal
  - **<value>** is a number (in the appropriate format) that fits the width of the constant
    - Numbers can be separated by "_"
- Examples
  - 16'h5a5a_a5a5
  - 5'd3
  - 3b110

Sequential Logic: Always @(posedge)


```
input wire i_reset_n;
input wire i_clk;
input wire i_refclk;

output wire o_1hz_stb;
output wire o_slow_set_stb;
output wire o_fast_set_stb;
output wire o_debounce_stb;

wire refclk_stb;
reg [14:0] refclk_div;
```

Define Signals

```
always @(posedge i_clk) begin
    if (refclk_stb) begin
        refclk_div <= refclk_div + 1;
    end

    if (!i_reset_n) begin
        refclk_div <= 15'h0;
    end
end
```

Low Priority

High Priority

Sequential Logic Behavior

- Always use system clock!
- Block “fires” on **i_clk** rising edge
- Note the **@(posedge signal)**
- start/stop with **begin**/**end**
- If / else if / else statements
  - Only in always blocks
  - start/stop with **begin**/**end**
- Assignment uses **<=**
  - Only assign to **reg** signals
  - Can read from regs and wires

Counter (as of right now)

- What do we expect?
- Block Diagram
  - 15-bit Flip-Flop
    - Triggered on **i_clk**
  - Add '1' when **i_reset_n** is high
  - Set to '0' when **i_reset_n** is low
  - Assign output to prevent deletion of block diagram
- Simulation
  - Counter should increment every system clock
  - We should see the system clock divided by 4
    - We will get rid of this later

```
module clk_gen (
  // global signals
  i_reset_n,
  i_clk,
  // Strobe from 32,768 Hz reference clock
  i_refclk,
  // output strobe signals
  o_1hz_stb,      // refclk / 2^15 -> 1Hz
  o_slow_set_stb, // refclk / 2^14 -> 2Hz
  o_fast_set_stb, // refclk / 2^12 -> 8Hz
  o_debounce_stb  // refclk / 2^4  -> 4.096KHz
);

input wire i_reset_n;
input wire i_clk;
input wire i_refclk;

output wire o_1hz_stb;
output wire o_slow_set_stb;
output wire o_fast_set_stb;
output wire o_debounce_stb;

wire refclk_stb;

reg [14:0] counter;
always @(posedge i_clk) begin
  counter <= counter + 15'd1;
  if (!i_reset_n) begin
    counter <= 15'd0;
  end
end

assign o_1hz_stb = counter[2];

endmodule
```

Expected Block Diagram

- Generate block diagram
  - **`elaborate_json.bat clk_gen src\input\clk_gen.v`**
  - This copies the block diagram into your clipboard
- Generate Picture
  - Go to https://neilturley.dev/netlistsvg/
  - Paste in code
  - Click “render”

```
C:\WINDOWS\
[OSS CAD Suite] D:
[OSS CAD Suite] D:
[OSS CAD Suite] D:\clock_2024_workshop>
[OSS CAD Suite] D:\clock_2024_workshop>
[OSS CAD Suite] D:\clock_2024_workshop>elaborate_json.bat clk_gen src\input\clk_gen.v
```

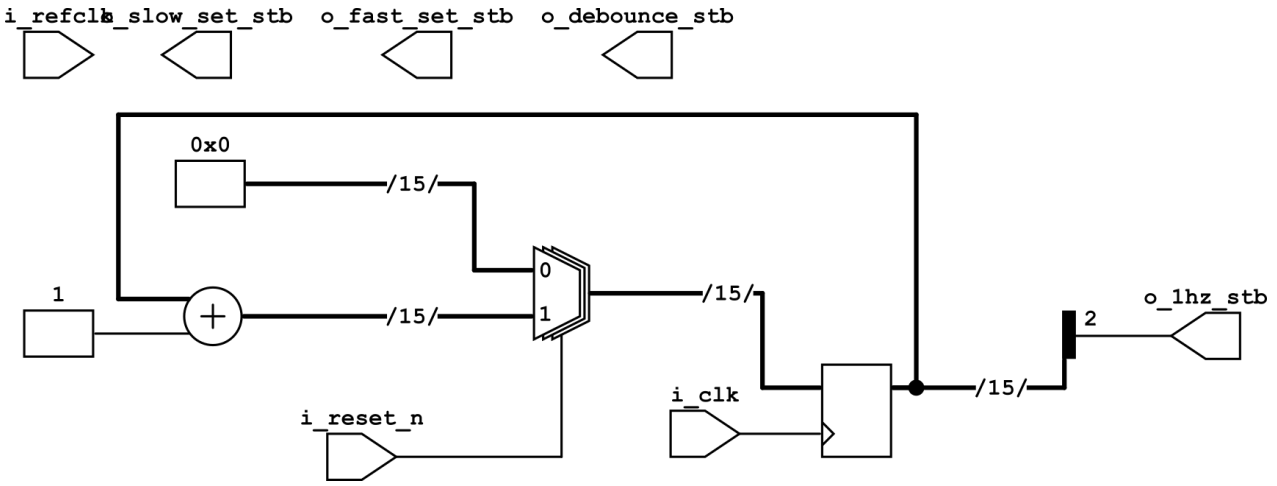
Generate Block Diagram

Commands are run from the project directory

This is a demo of netlistsvg.

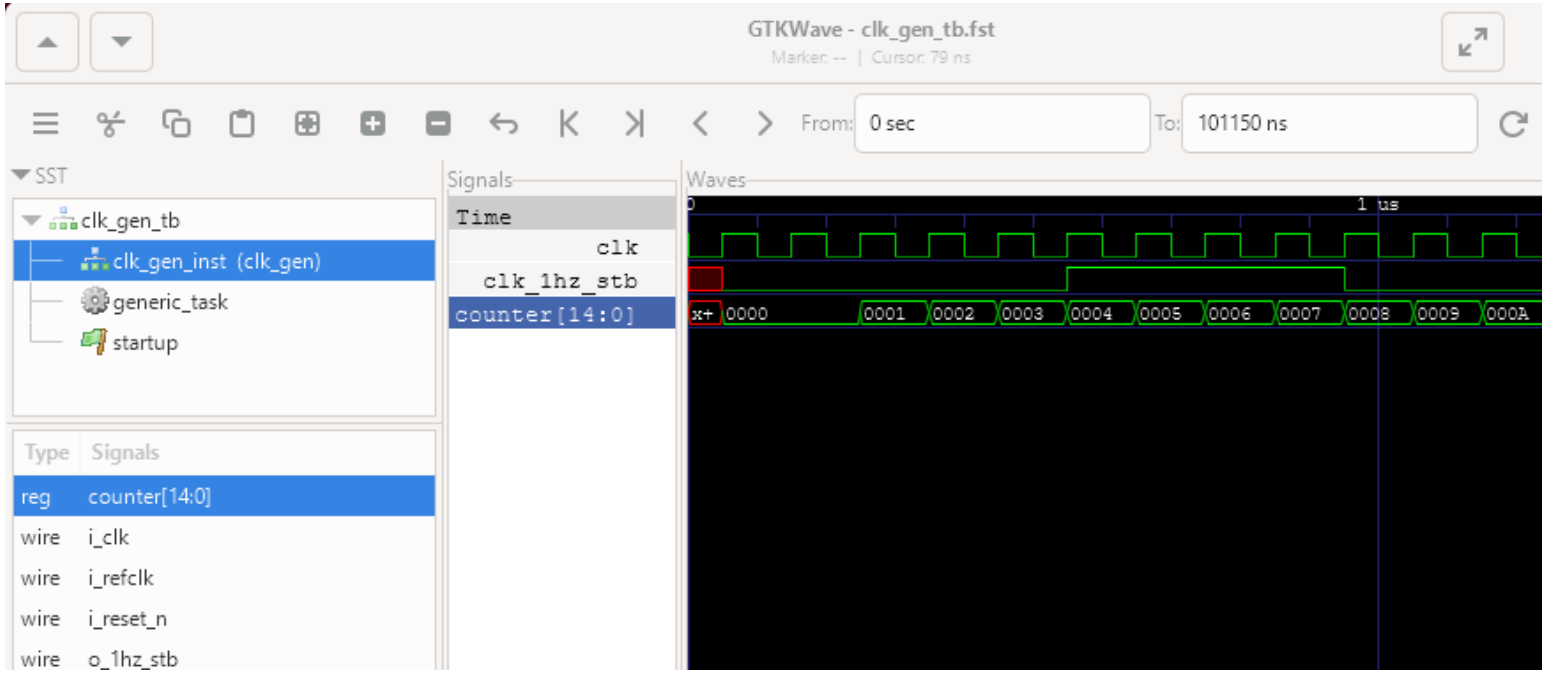
Format JSON | Select skin: lib/default.svg | Render

```
          "attributes": {
            "src": "src\\input\\clk_gen.v:45.1-50.4"
          },
          "port_directions": {
            "CLK": "input",
            "D": "input",
            "Q": "output"
          },
          "connections": {
            "CLK": [ 3 ],
            "D": [ 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
48, 49, 50, 51, 52 ],
            "Q": [ 9, 10, 5, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22 ]
          }
        },
        "$procmux$4": {
```



Expected Counter Output

- Run Simulator
  - `iverilog -o clk_gen_tb.vvp test\clk_gen_tb.v src\input\clk_gen.v`
  - `vvp clk_gen_tb.vvp -fst`
- Open Waveform Viewer
  - Do this in a different terminal
  - `gtkwave clk_gen_tb.fst`



Run Simulator

```
C:\WINDOWS\system32\cmd.exe
[OSS CAD Suite] D:\clock_2024_workshop>
[OSS CAD Suite] D:\clock_2024_workshop>iverilog -o clk_gen_tb.vvp test\clk_gen_tb.v src\input\clk_gen.v

[OSS CAD Suite] D:\clock_2024_workshop>vvp clk_gen_tb.vvp -fst
```

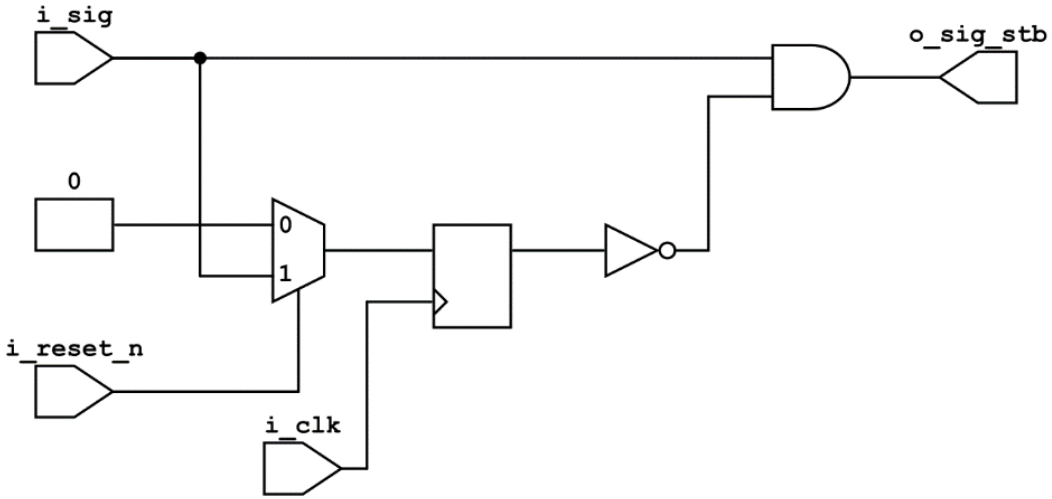
Run gtkwave

```
C:\WINDOWS\s...en_tb.fst
[OSS CAD Suite] D:\clock_2024_workshop>gtkwave clk_gen_tb.fst

GTKWave Analyzer v3.4.0 (w)1999-2022 BSI

FSTLOAD | Processing 26 facs.
FSTLOAD | Built 19 signals and 7 aliases.
FSTLOAD | Building facility hierarchy tree.
FSTLOAD | Sorting facility hierarchy tree.
```

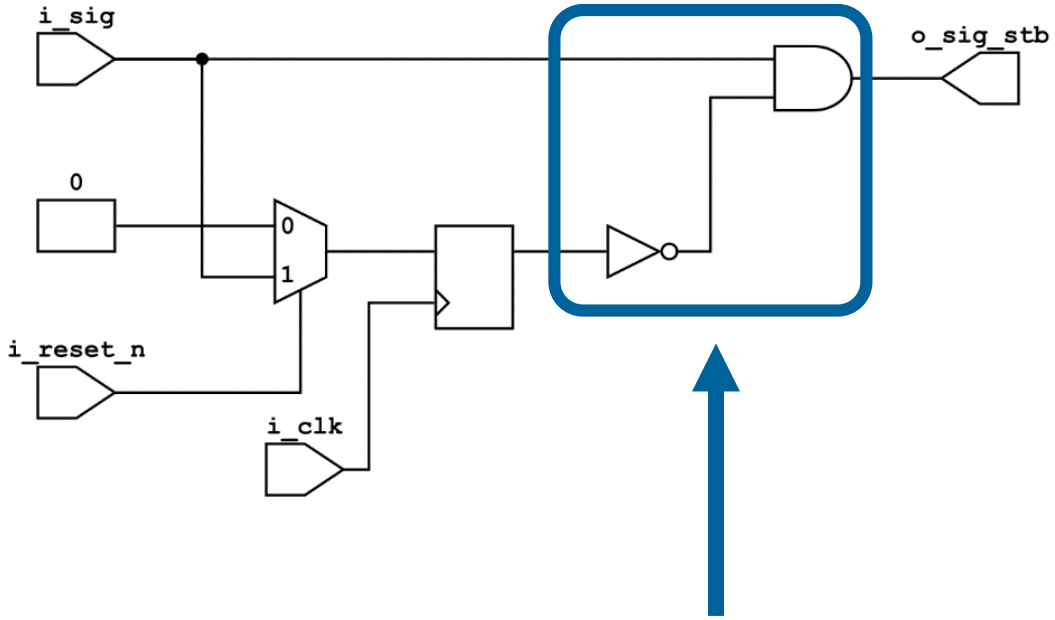
Commands are run from the project directory

Generating Strobe Signals

- Output ‘1’ on signal rising edge
- Detect rising edge
  - Hold the previous signal state
  - Compare to current signal state
  - Output ‘1’ when previous state is ‘0’ and current state is ‘1’
- Reset when **`i_reset_n`** is low
- Put this into a new module
  - The new module should be in the same file as the counter module



Combinational Logic: Assign statement

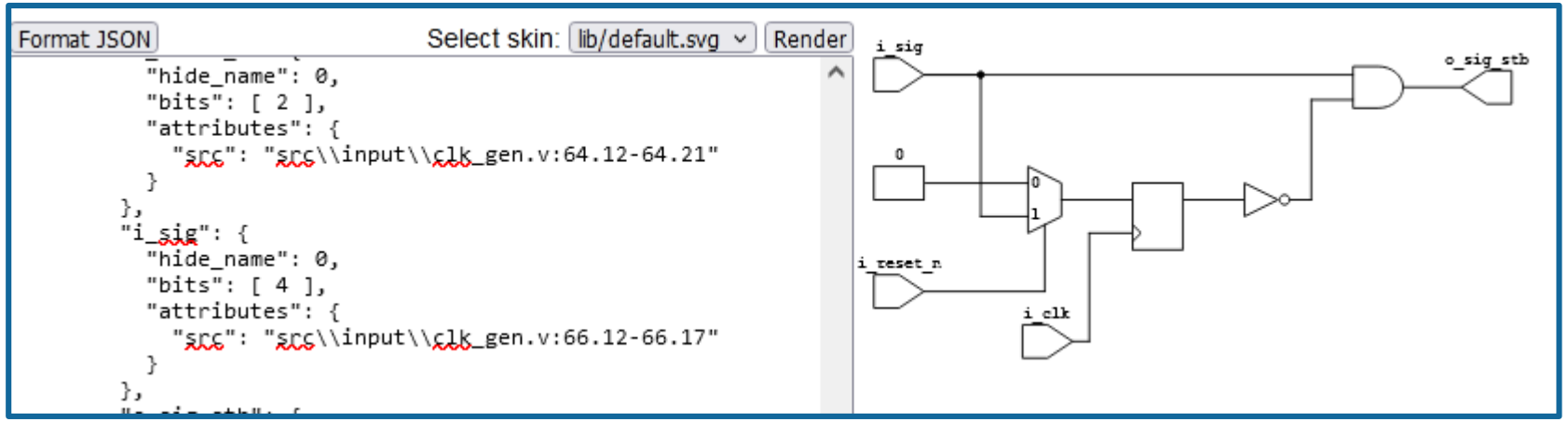



```
assign o_sig_stb = i_sig & ~sig_hold;
```

- In general:
  `assign wire_name = signal1 [operation signal2 …];`
- Signals can be either wires or regs
- Can only assign to wires
- Assign statements cannot be in always blocks

Strobe Generator Code

- Example Strobe Generator
- Generate Block Diagram
  - **`elaborate_json.bat stb_gen src\input\clk_gen.v`**
  - Paste into netlistsvg




```
assign o_1hz_stb = counter[2];

endmodule
```

End of clk_gen

```
module stb_gen (
  i_reset_n,
  i_clk,

  i_sig,
  o_sig_stb
);

input wire i_reset_n;
input wire i_clk;
input wire i_sig;
output wire o_sig_stb;

reg sig_hold;
always @(posedge i_clk) begin
  sig_hold <= i_sig;
  if (!i_reset_n) begin
    sig_hold <= 1'b0;
  end
end

assign o_sig_stb = i_sig & ~sig_hold;

endmodule
```

Fixing the counter

- We only want to count up on the rising edge of refclk
- Generate a strobe signal off of refclk (**`refclk_stb`**)
  - Instantiate a module that we just made
  - Input signal is **`refclk`**, generates **`refclk_stb`**
  - Only count up when **`refclk_stb`** is high
  - This makes our counter work right.
- Generate strobe signals for each output clock.
  - Do the same thing, but take strobe input from counter register

Instantiating Modules

- Back to Modules
- How do we instantiate a module?

```
module_name #(parameter list)
instance_name (
   .port_name1 (connected signal 1),
   .port_name2 (connected signal 2)
);
```

I'm not using this. Just FYI

```
// clock strobe generator signals
wire refclk_sync;
wire clk_1hz_stb;
wire clk_slow_set_stb;
wire clk_fast_set_stb;
wire clk_debounce_stb;
```

Define Signals

```
clk_gen clk_gen_inst (
    .i_reset_n      (reset_n),
    .i_clk          (clk),
    .i_refclk       (refclk_sync),
    .o_1hz_stb      (clk_1hz_stb),
    .o_slow_set_stb (clk_slow_set_stb),
    .o_fast_set_stb (clk_fast_set_stb),
    .o_debounce_stb (clk_debounce_stb)
);
```

Instantiate Clock Generation Module

```
// Generate a strobe signal on reference clock rising edge
wire refclk_stb;
stb_gen refclk_stb_inst (
    .i_reset_n (i_reset_n),
    .i_clk     (i_clk),
    .i_sig     (i_refclk),
    .o_sig_stb (refclk_stb)
);
```

Instantiate Strobe Generation Module

Updated Clock Generator

- Add refclk strobe generator
- Only update counter on **`refclk`** rising edge
  - Check if **`refclk_stb`** is high before incrementing
- Generate slower strobes
  - Strobe on $\frac{refclk}{2^{15}}$, $\frac{refclk}{2^{14}}$, $\frac{refclk}{2^{12}}$, and $\frac{refclk}{2^{4}}$
  - Generates 1Hz, 2Hz, 8Hz, and 4.096KHz strobes

```
// Generate a strobe signal on reference clock rising edge
wire refclk_stb;
stb_gen refclk_stb_inst (
    .i_reset_n (i_reset_n),
    .i_clk     (i_clk),
    .i_sig     (i_refclk),
    .o_sig_stb (refclk_stb)
);
```

refclk

Instantiate Strobe Generation Module

```
reg [14:0] counter;
// TODO Add counter logic
always @(posedge i_clk) begin
    if (refclk_stb) begin
        counter <= counter + 15'd1;
    end
    if (!i_reset_n) begin
        counter <= 15'd0;
    end
end

// generate strobe signals off of the clock divider
// 32,768 / 2^15 -> 1hz
stb_gen stb_gen_1hz (
    .i_reset_n (i_reset_n),
    .i_clk     (i_clk),
    .i_sig     (counter[14]),
    .o_sig_stb (o_1hz_stb)
);

// 32,768 / 2^13 -> 2hz
stb_gen stb_gen_slow_clk (
    .i_reset_n (i_reset_n),
    .i_clk     (i_clk),
    .i_sig     (counter[13]),
    .o_sig_stb (o_slow_set_stb)
);

// 32,768 / 2^11 -> 8hz
stb_gen stb_gen_fast_clk (
    .i_reset_n (i_reset_n),
    .i_clk     (i_clk),
    .i_sig     (counter[11]),
    .o_sig_stb (o_fast_set_stb)
);

// 32,768 / 2^3 -> 4096Hz
stb_gen stb_gen_debounce_clk (
    .i_reset_n (i_reset_n),
    .i_clk     (i_clk),
    .i_sig     (counter[3]),
    .o_sig_stb (o_debounce_stb)
);

endmodule
```

counter

$\div 2^{15}$

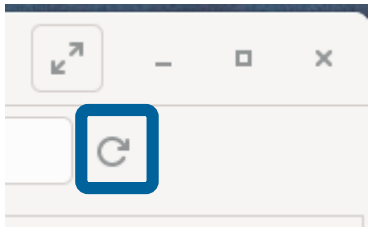
$\div 2^{14}$

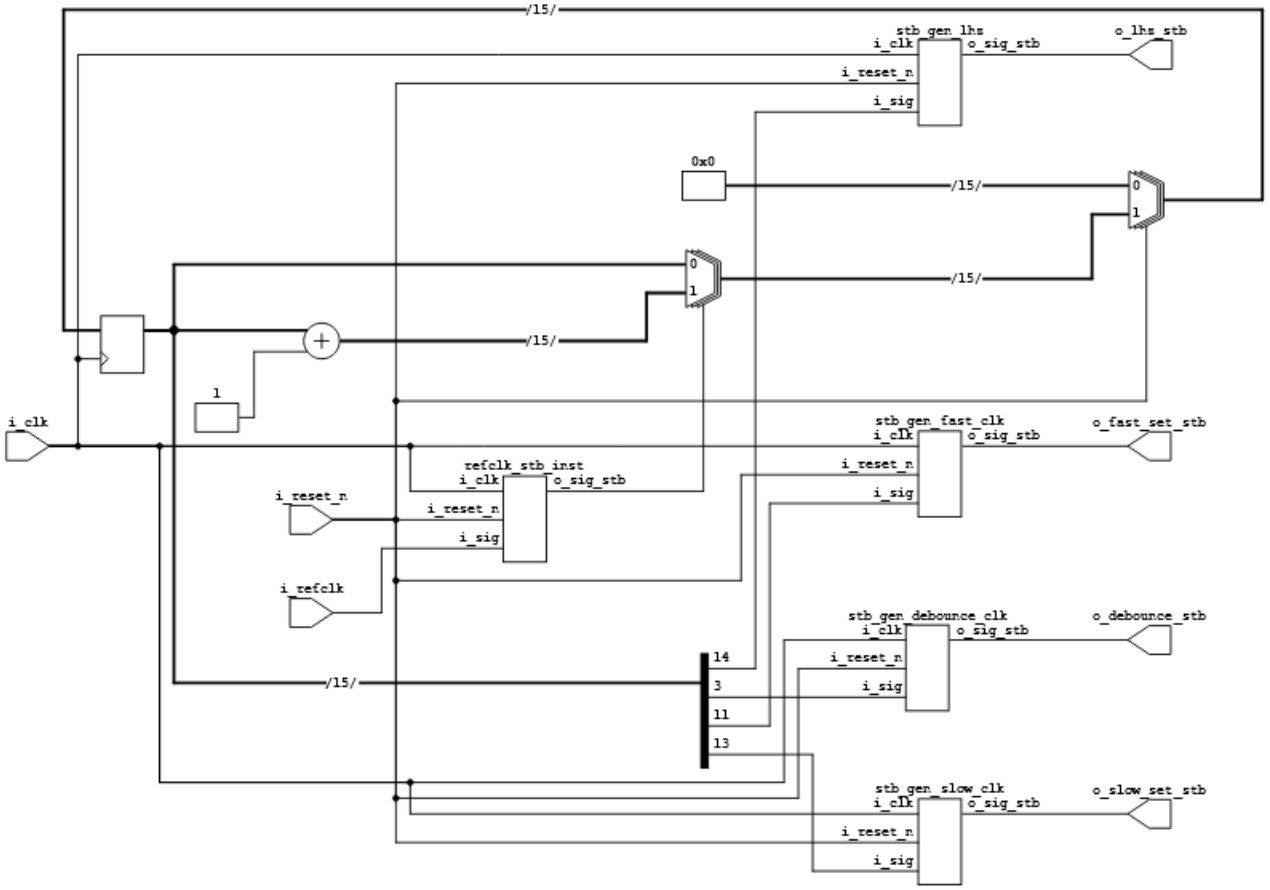
$\div 2^{12}$

$\div 2^{4}$

Clock Generator

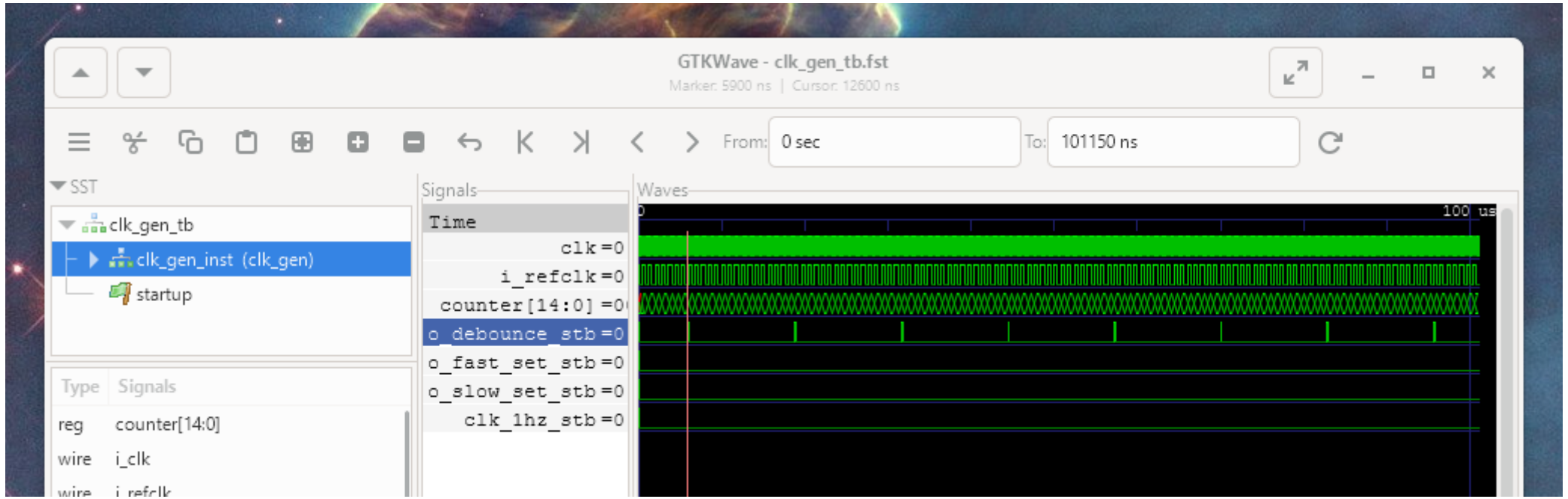
- Generate Block Diagram
  - `elaborate_json.bat clk_gen src\input\clk_gen.v`
  - Paste into netlistsvg
- Run Simulation
  - `iverilog -o clk_gen_tb.vvp test\clk_gen_tb.v src\input\clk_gen.v`
  - `vvp clk_gen_tb.vvp -fst`
- Refresh Simulator Output





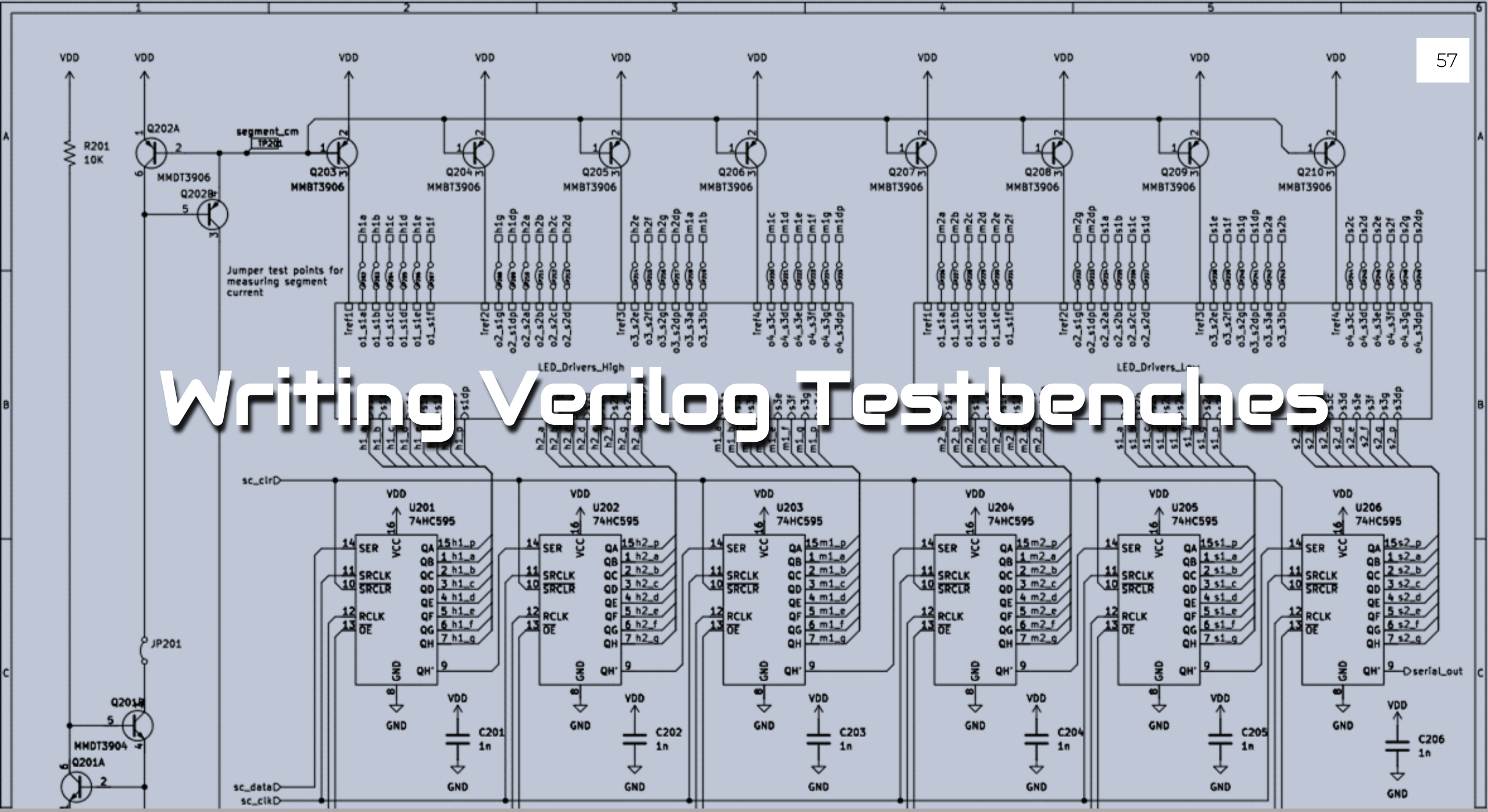
Expected Output

- Should see pulses on **`o_debounce_stb`** signal
- Where are our other pulses?
  - We aren't waiting long enough
  - Take a closer look at testbenches next



What have we learned?

- Write basic modules
  - Sequential logic with “always @(posedge i_clk)”
  - Sequential logic with “assign”
  - Generate Strobe signals to trigger events
- View block diagrams
  - Use yosys to elaborate designs
  - View with netlistsvg
- Run Simulator
  - Use verilator to simulate Verilog module
  - Use gtkwave to view output


Writing Verilog Testbenches
VDD
R201
10K
Q202A
MMDT3906
Q202B
segment_cm
TP201
Q203
MMBT3906
Q204
MMBT3906
Q205
MMBT3906
Q206
MMBT3906
Q207
MMBT3906
Q208
MMBT3906
Q209
MMBT3906
Q210
MMBT3906
Jumper test points for measuring segment current
LED_Drivers_High
LED_Drivers_Low
sc_clr
U201
74HC595
U202
74HC595
U203
74HC595
U204
74HC595
U205
74HC595
U206
74HC595
SER
SRCLK
SRCLR
RCLK
OE
VCC
GND
QA
QB
QC
QD
QE
QF
QG
QH
QH'
C201
1n
C202
1n
C203
1n
C204
1n
C205
1n
C206
1n
JP201
Q201B
MMDT3904
Q201A
sc_data
sc_clk
serial_out

Writing Verilog Testbenches

- So far we've used a very basic testbench
  - It just runs the system clock for 1000 cycles
  - No checks on the output are run
- We want to look at writing better testbenches
  - Failure output if module output doesn't match expectations
  - Similar to unit tests from software
- Using Verilog tasks to make life easier
  - More like using a normal programming language

Open clk_gen_tb Testbench

- Open `test\clk_gen_tb.v`
- Notable things
  - Testbench is a Verilog module
    - Takes no inputs
    - Produces no outputs
  - Less strict rules
    - This doesn't get turned into hardware
    - Use "=" in tasks
- Let's go on a tour
  - Assert definitions
  - Module definition

Assert Definitions

```
// Assert helpers for ending the simulation early in failure
`define assert(signal, value) \
    if (signal !== value) begin \
        $display("ASSERTION FAILED in %m:\nEquality"); \
        $display("\t%d (expected) != %d (actual)", value, signal); \
        close(); \
    end

`define assert_cond(signal, cond, value) \
    if (!(signal cond value)) begin \
        $display("ASSERTION FAILED in %m:\nCondition:"); \
        $display("\n\texpected: %d\n\tactual: %d", value, signal); \
        close(); \
    end

`define assert_timeout(max_count) \
    if (!(timeout_counter < max_count)) begin \
        $display("ASSERTION FAILED in %m\nTimeout:"); \
        $display("\tspecified max count: %d\n\tactual count: %d", max_count,
        timeout_counter); \
        close(); \
    end
```

Module Definition

```
module clk_gen_tb ();
    // cocotb interface signals
    reg test_done = 0;

    // global parameters
    localparam CLK_PERIOD    = 100;
    localparam REFCLK_PERIOD = 203;
    localparam TIMEOUT       = 5000;
```

Testbench: Startup (1/2)


```
33  module clk_gen_tb ();
34      // cocotb interface signals
35      reg test_done = 0;
36
37      // global parameters
38      localparam CLK_PERIOD    = 100;
39      localparam REFCLK_PERIOD = 203;
40      localparam TIMEOUT       = 5000;
41      localparam STARTUP_DELAY = 5;
42
43      // global testbench signals
44      reg clk = 0;
45      reg reset_n = 0;
46      reg ena = 1;
47      reg refclk = 0;
48
49      reg run_timeout_counter;
50      reg [15:0] timeout_counter = 0;
```

- Local parameters act as constants
- Used throughout testbench to reduce "magic numbers"

- Set initial values on registers
- Puts registers into well defined state at startup

Testbench: Module Initilization


```
52  // Output signals from device-under-test (DUT)
53  wire clk_1hz_stb;
54  wire clk_slow_set_stb;
55  wire clk_fast_set_stb;
56  wire debounce_stb;
57
58  // setup top level testbench signals
59  clk_gen clk_gen_inst (
60      // global signals
61      .i_reset_n(reset_n),
62      .i_clk(clk),
63      // Strobe from 32,768 Hz reference clock
64      .i_refclk(refclk),
65      // output strobe signals
66      .o_1hz_stb(clk_1hz_stb),          // refclk / 2^15 -> 1Hz
67      .o_slow_set_stb(clk_slow_set_stb), // refclk / 2^14 -> 2Hz
68      .o_fast_set_stb(clk_fast_set_stb), // refclk / 2^12 -> 8Hz
69      .o_debounce_stb(debounce_stb)     // refclk / 2^4  -> 4.096KHz
70  );
```

- Define signals generated by device-under-test

- Setup DUT
- We can instantiate more modules for helping out
- Any other modules need added to simulator sources

Testbench: Startup (2/2)


```
72    // setup file dumping things
73    initial begin: startup
74      integer i;
75      // Set the output file to dump to
76      $dumpfile("clk_gen_tb.fst");
77      $dumpvars(0, clk_gen_tb);
78      #STARTUP_DELAY;
79
80      $display("Clock Strobe Generation Testbench");
81      init();
82
83      // TODO: Make testbench fancy
84      // This just runs for 1000 clock cycles then quit
85      for (i = 0; i < 1000; i = i + 1) begin
86        @(posedge clk);
87        // $display("%d", clk_1hz_stb);
88      end
89
90      test_done = 1;
91      // exit the simulator
92      close();
93    end
```

- **`initial`** block is run on startup
- Name provided so that variables can be used in block.
  - **`begin: <block name>`**

- **`integer i`** counter variable definition
  - 32-bit local variable

- **`$dumpfile(<string>)`**
  - specify output file name
- **`$dumpvars(0, <modules/variables>)`**
  - Save waveform outputs from a module
  - See Verilog Reference 18.1.2

- **`$display(<format>, <list of vals>)`**
  - Works like printf
- Call **`init()`** task

- Run simulation for 1000 clock cycles

Testbench: Default Tasks


```
117     task init();
118         begin
119             $display("Simulation Start");
120             $display("Reset");
121 
122             repeat (2) @(posedge clk);
123             reset_n = 1;
124             $display("Run");
125         end
126     endtask
```

```
128     task close();
129         begin
130             $display("Closing");
131             repeat (10) @(posedge clk);
132             $finish;
133         end
134     endtask
```

- In general:

```
task task_name ( input input1, … );
begin: <block_name>
  // stuff
end
endmodule
```

- Reset the system
  - Assert **reset_n** for two clock cycles

- Delay a bit before ending the simulation
- Makes it easier to read waveforms

$finish closes the simulator

Testbench: Clock Generation


```
136   task reset_timeout_counter();
137     begin
138       @(posedge clk);
139       run_timeout_counter = 1'd0;
140       @(posedge clk);
141       run_timeout_counter = 1'd1;
142     end
143   endtask
144
145   // System Clock
146   localparam CLK_HALF_PERIOD = CLK_PERIOD / 2;
147   always #(CLK_HALF_PERIOD) begin
148     clk <= ~clk;
149   end
150
151   // Reference clock
152   localparam REFCLK_HALF_PERIOD = REFCLK_PERIOD / 2;
153   always #(REFCLK_HALF_PERIOD) begin
154     refclk <= ~refclk;
155   end
156
157   // Timeout Clock
158   always @(posedge clk) begin
159     if (run_timeout_counter) timeout_counter <= timeout_counter + 1'd1;
160     else timeout_counter <= 16'h0;
161   end
```

- Task to clear the timeout counter
  - More to follow later

- Periodically toggle clock registers
  - **`always #(<delay_time>)`**

- Run the timeout counter
  - **`always #(<delay_time>)`**

Testbench: User Task


```
98  task generic_task(
99      input integer timeout
100 );
101     begin : my_generic_task
102         $display("Timeout: %d", timeout);
103
104         // reset our watchdog timer
105         reset_timeout_counter();
106
107         while ( /* some condition */ timeout_counter < timeout) begin
108             // TODO: Implement the generic task we want to accomplish
109             @(posedge clk);
110         end
111
112         // make sure we didn't run out the watchdog timer
113         `assert_timeout(timeout);
114     end
115 endtask
```

- We will edit this task for our testbench

- Input value to task
  - We use this to set the maximum count for the timeout counter

- Reset the timeout counter so that we can check its value from a known state

- Loop until a condition is met
- Exit the loop if we are over the timeout
- One clock cycle per loop iteration

- Assert that we passed the condition and didn't timeout

Edit the testbench

- Call the user task
  - Remove the for loop
- Run the simulator
  - Same command as before
  - Should see timeout failure

```
72  // setup file dumping things
73  initial begin: startup
74      integer i;
75      // Set the output file to dump to
76      $dumpfile("clk_gen_tb.fst");
77      $dumpvars(0, clk_gen_tb);
78      #STARTUP_DELAY;
79
80      $display("Clock Strobe Generation Testbench");
81      init();
82
83      generic_task(TIMEOUT);
84
85      test_done = 1;
86      // exit the simulator
87      close();
88  end
```

```
C:\WINDOWS\system32\cmd.exe
[OSS CAD Suite] D:\clock_2024_workshop>
[OSS CAD Suite] D:\clock_2024_workshop>
[OSS CAD Suite] D:\clock_2024_workshop>
[OSS CAD Suite] D:\clock_2024_workshop>iverilog -o clk_gen_tb.vvp test\clk_gen_tb.v src\input\clk_gen.referenc

[OSS CAD Suite] D:\clock_2024_workshop>vvp clk_gen_tb.vvp -fst
FST info: dumpfile clk_gen_tb.fst opened for output.
Clock Strobe Generation Testbench
Simulation Start
Reset
Run
Timeout:        5000
ASSERTION FAILED in clk_gen_tb.generic_task
Timeout:
        specified max count:        5000
        actual count:  5000
Closing
test\clk_gen_tb.v:127: $finish called at 501450 (1ns)

[OSS CAD Suite] D:\clock_2024_workshop>
```

Make our test snazzy

- Check that we have the correct number of pulses
  - Add a counters for each pulse generator
- Update task to check correct operation
  - Assert statements to generate failure logs
  - "===" checks against invalid and tristate conditions

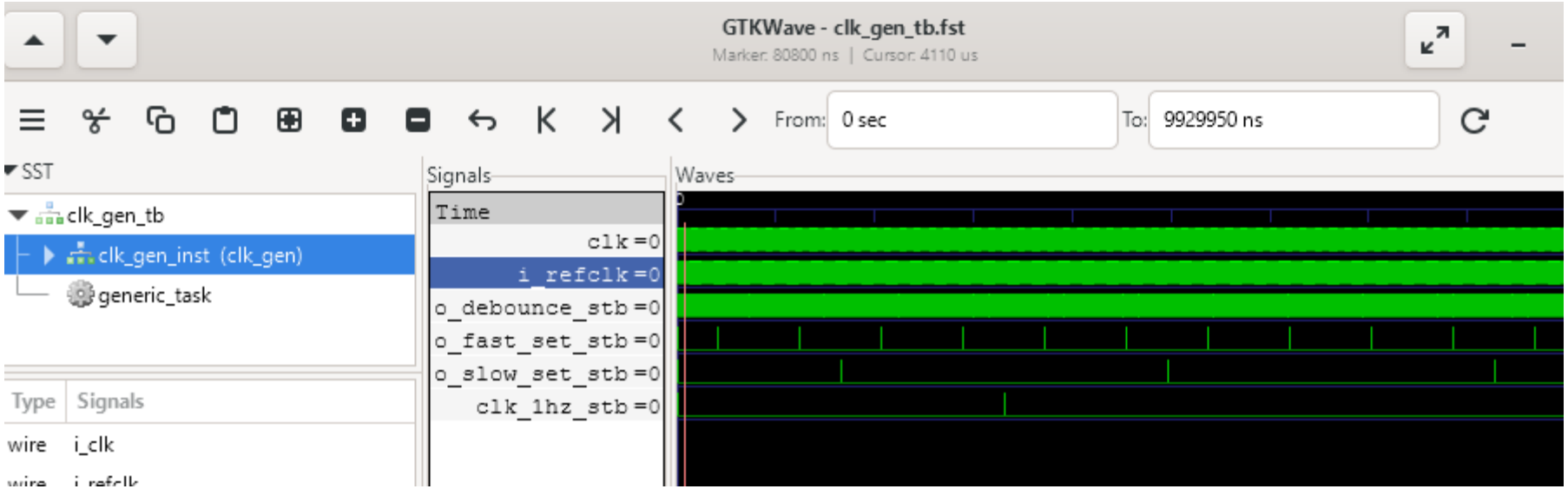
```
72  reg [15:0] slow_stb_count = 0;
73  reg [15:0] fast_stb_count = 0;
74  reg [15:0] debounce_stb_count = 0;
75  always @(posedge clk) begin
76      if (clk_slow_set_stb) begin
77          slow_stb_count <= slow_stb_count + 1;
78      end
79
80      if (clk_fast_set_stb) begin
81          fast_stb_count <= fast_stb_count + 1;
82      end
83
84      if (debounce_stb) begin
85          debounce_stb_count <= debounce_stb_count + 1;
86      end
87  end
```

```
110 task generic_task(
111     input integer timeout
112 );
113     begin : my_generic_task
114         $display("Timeout: %d", timeout);
115
116         // reset our watchdog timer
117         reset_timeout_counter();
118
119         // wait for the first 1hz strobe signal
120         $display("Wait for 1hz Strobe Signal");
121         while ( !clk_1hz_stb && timeout_counter < timeout) begin
122             @(posedge clk);
123         end
124         // make sure we didn't run out the watchdog timer
125         `assert_timeout(timeout);
126
127         // reset our watchdog timer
128         reset_timeout_counter();
129         // clear our counter registers
130         slow_stb_count = 0;
131         fast_stb_count = 0;
132         debounce_stb_count = 0;
133
134         $display("Wait for 2nd 1hz Strobe Signal");
135         while ( !clk_1hz_stb && timeout_counter < timeout) begin
136             @(posedge clk);
137         end
138
139         // make sure we didn't run out the watchdog timer
140         `assert_timeout(timeout);
141
142         // assert that we get the correct number of counts
143         // in our counting registers
144         `assert_cond(slow_stb_count, ===, 2);
145         `assert_cond(fast_stb_count, ===, 8);
146         `assert_cond(debounce_stb_count, >=, 1000);
147
148     end
149 endtask
```

To the Simulator!

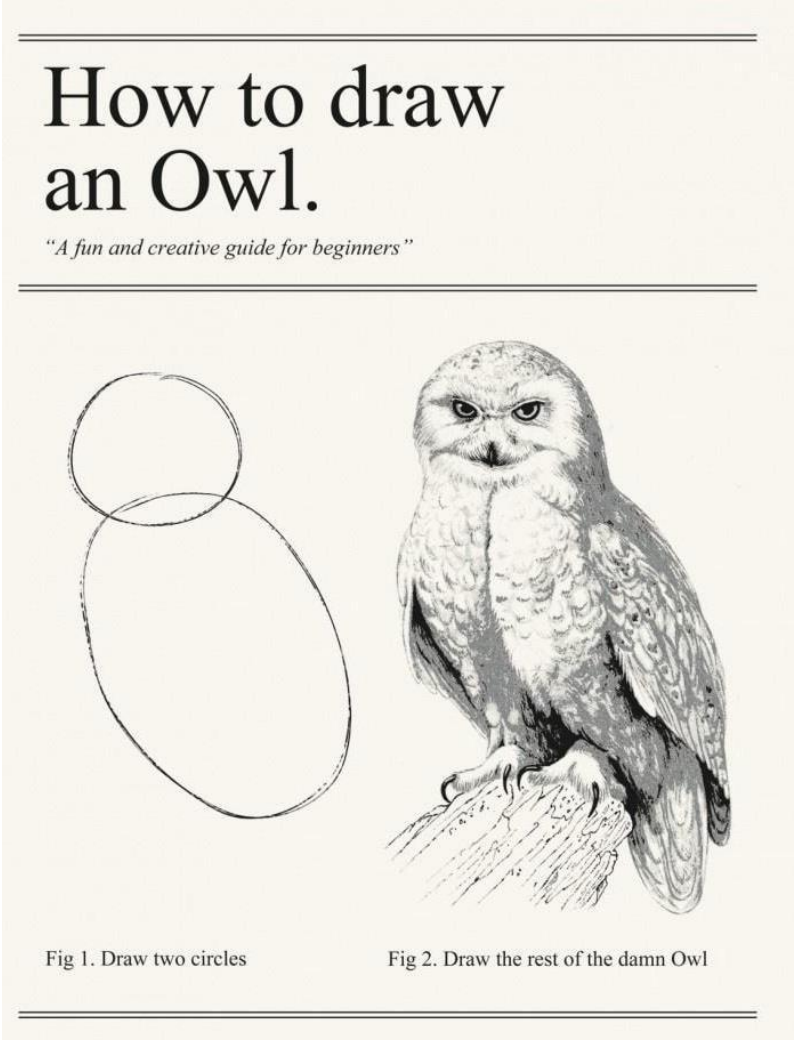
- Run simulator
- Expected Simulator log
- Should see our pulses in gtkwave

```
C:\WINDOWS\system32\cmd.exe
Closing
test\clk_gen_tb.v:166: $finish called at 3001450 (1ns)

[OSS CAD Suite] D:\clock_2024_workshop>iverilog -o clk_gen_tb.vvp test\clk_gen_tb.v src\input\clk_gen.reference.v

[OSS CAD Suite] D:\clock_2024_workshop>vvp clk_gen_tb.vvp -fst
FST info: dumpfile clk_gen_tb.fst opened for output.
Clock Strobe Generation Testbench
Simulation Start
Reset
Run
Timeout:      300000
Wait for 1hz Strobe Signal
Wait for 2nd 1hz Strobe Signal
Closing
test\clk_gen_tb.v:166: $finish called at 9929950 (1ns)

[OSS CAD Suite] D:\clock_2024_workshop>
```



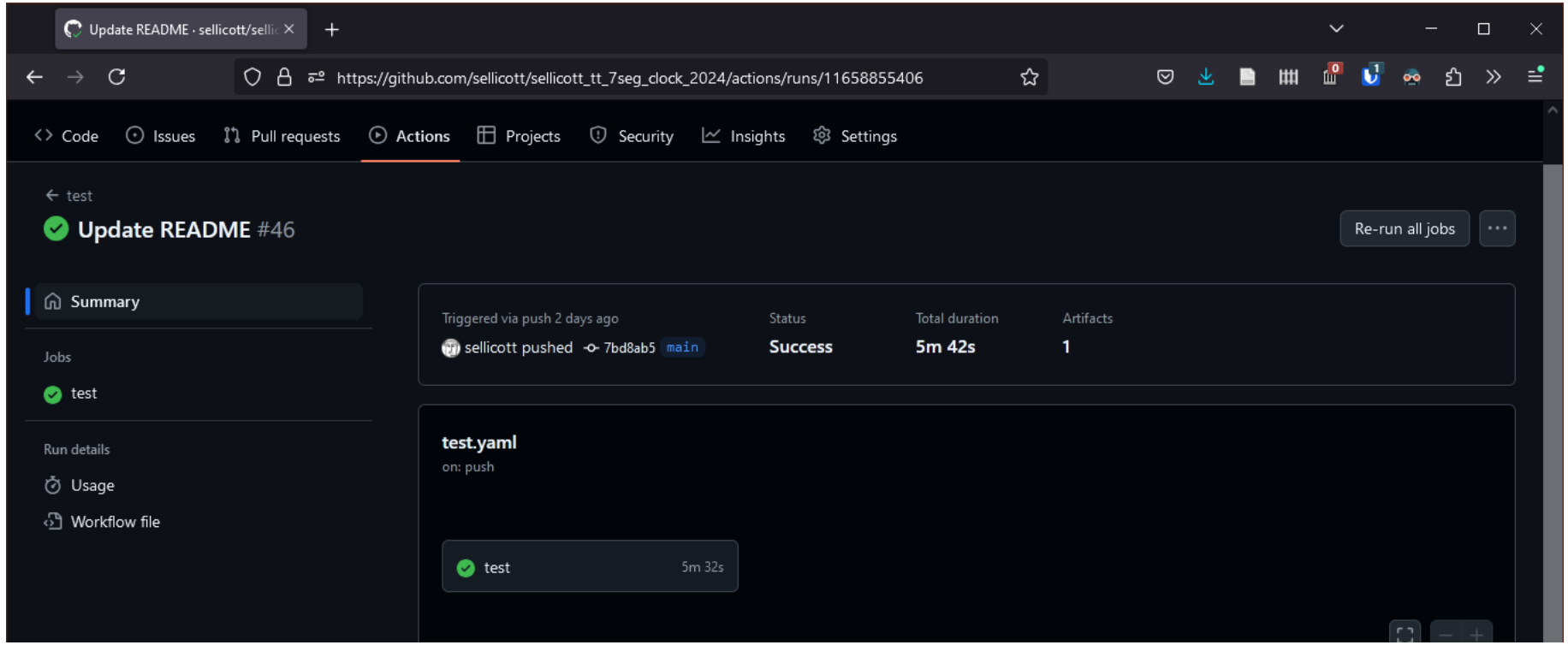
Write the Clock Register Module

- Store Hours, Minutes, Seconds
- Normal Operation
  - Count up when 1Hz strobe signal is high
  - Overflow Seconds and count up minutes
  - Overflow Minutes and count up hours
- Time Set
  - When **`i_set_hours`** is asserted
    - Switch to using the **`i_set_stb`** signal
    - Increment hours when **`i_set_stb`** is high
  - When **`i_set_minutes`** is asserted
    - Switch to using the **`i_set_stb`** signal
    - Increment hours when **`i_set_stb`** is high
  - When both are asserted
    - Clear the seconds register



We need to update the project

- Push changes to Github
- The Github actions will run
  - We (hopefully) see cool output



What have we learned?

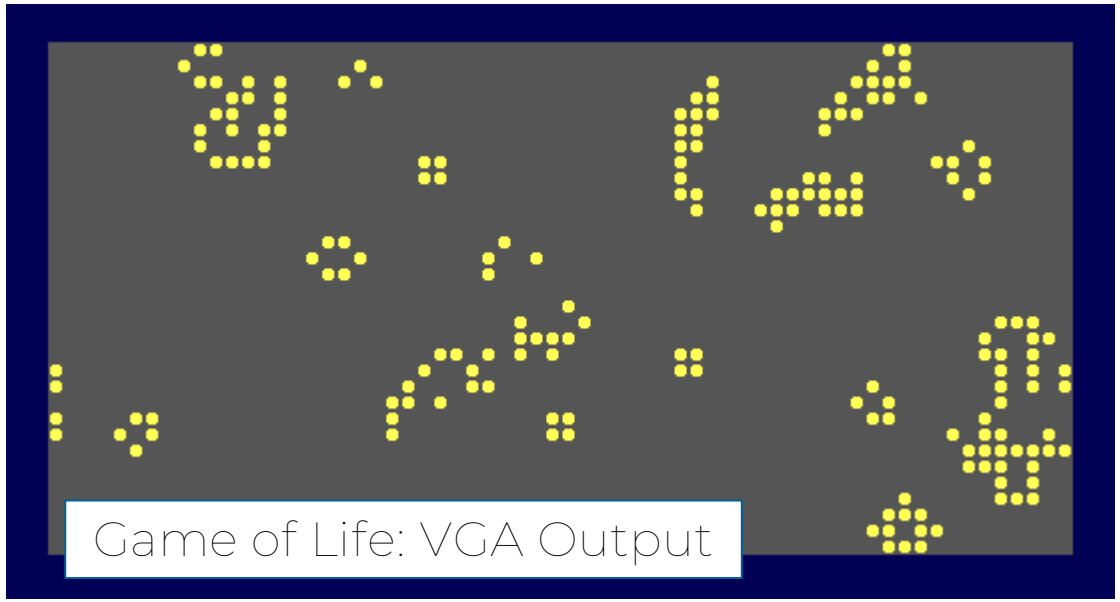
- Write basic modules
  - Sequential logic with “always @(posedge i_clk)”
  - Sequential logic with “assign”

- View block diagrams
  - Use yosys to elaborate designs
  - View with netlistsvg

- Write testbench to simulate modules
  - Simulate with Icarus Verilog
  - View results with gtkwave

Things to look at

- Detailed Verilog Tutorials
  - NandLand
  - ChipVerify
  - ZipCPU



Game of Life: VGA Output

- Tiny Tapeout Demoscene
  - https://tinytapeout.com/competitions/demoscene/
  - https://vga-playground.com/

Where To Get Help

Here!

- Tiny Tapeout Discord
  - Lots of people with deeper knowledge than me
  - I'm there too (@sellicott)

- Learning Resources
  - https://tinytapeout.com/digital_design/
  - https://tinytapeout.com/hdl/
  - https://r2.ieee.org/columbus-ssccas/resources/

- Questions about this workshop
  - Chapter: columbus.sscs.cas@gmail.com

Tiny Tapeout Discord
https://discord.gg/BspcX9SB2h

Acknowledgements

Matt Venn

- Original workshop slides
- Advice on running a workshop

IEEE CIRCUITS AND SYSTEMS SOCIETY

SenseICs

Everyone who worked on Tiny Tapeout!

efabless.com

https://tinytapeout.com/credits/

Extra Slide

This page unintentionally left blank