



Gisselquist  
Technology, LLC

### 3. Finite State Machines

Daniel E. Gisselquist, Ph.D.





# Lesson Overview



## ▷ Lesson Overview

Shift Register  
Wavedrom  
LED Walker  
Wavedrom  
The Need  
Case Statement  
The Need  
The Need  
The addresses  
Simulation  
Finite State Machine  
Simple  
Mealy  
Moore  
One Process FSM  
Two Process FSM  
Which to use?  
Formal Verification  
Assertion  
SymbiYosys  
Integer Clock  
Divider  
Exercise  
Conclusion

- What is a Finite State Machine?
- Why do I need it?
- How do I build one?

## Objectives

- Learn the concatenation operator
- Be able to explain a shift register
- To get basic understanding of Finite State Machines
- To learn how to build and use Finite State Machines



# Shift Register



Lesson Overview

▷ Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

The concatenation operator

```
always @(posedge i_clk)
    o_led <= { o_led[6:0], o_led[7] };
```

Composes a new bit-vector from other pieces



# Shift Register



Lesson Overview

➤ Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

The concatenation operator

```
always @(posedge i_clk)
    o_led <= { o_led[6:0], o_led[7] };
```

Simplifies what otherwise would be quite painful

```
always @(posedge i_clk)
begin
    o_led[0] <= o_led[7];
    o_led[1] <= o_led[0];
    o_led[2] <= o_led[1];
    o_led[3] <= o_led[2];
    o_led[4] <= o_led[3];
    o_led[5] <= o_led[4];
    o_led[6] <= o_led[5];
    o_led[7] <= o_led[6];
end
```



# Shift Register



Lesson Overview

➤ Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

A shift register shifts bits through a register

- Can shift from LSB to MSB

```
always @(posedge i_clk)
    o_led <= { o_led[6:0], i_input };
```

- or from MSB to LSB

```
always @(posedge i_clk)
    o_led <= { i_input, o_led[7:1] };
```



# Shift Register



Lesson Overview

➤ Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

You can use this to create a neat LED display as well

- You just need to mix the shift register

```
initial o_led = 8'h1;  
always @(posedge i_clk)  
if (stb)  
    o_led <= { o_led[6:0], o_led[7] };
```

- With a counter to slow it down

```
reg [26:0] counter;  
reg stb;  
initial { stb, counter } = 0;  
always @(posedge i_clk)  
    { stb, counter } <= counter + 1'b1;
```

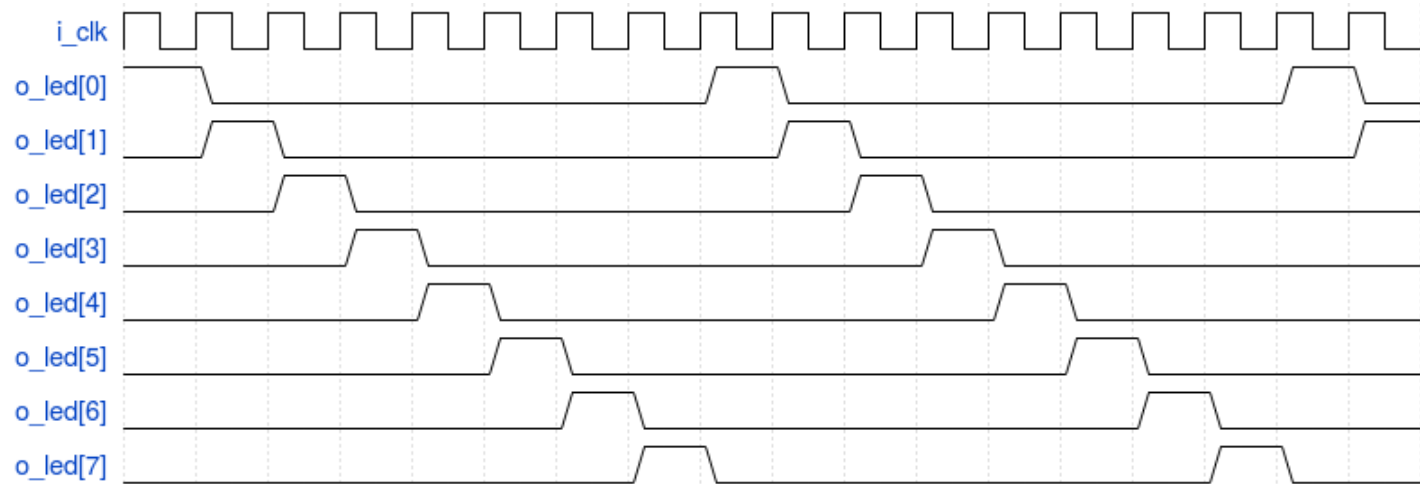
- Note that you can assign to a concatenation as well



# Wavedrom



If you've never seen Wavedrom, it is an awesome tool!  
Here's a waveform description of our shift register



```
initial o_led = 8'h1;  
always @(posedge i_clk)  
    o_led <= { o_led[6:0], o_led[7] };
```

What would it take to make the LED's go *back and forth*?



# LED Walker



Lesson Overview

Shift Register

Wavedrom

▷ LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

Let's build an LED walker!

- Active LED should walk across valid LED's and back

We'll assume 8 LEDs

Shift registers don't naturally go both ways

- Only one LED should be active at any time
- One LED should always be active at any given time

Most of this project can be done in simulation





# Wavedrom



Lesson Overview

Shift Register

Wavedrom

LED Walker

▷ Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

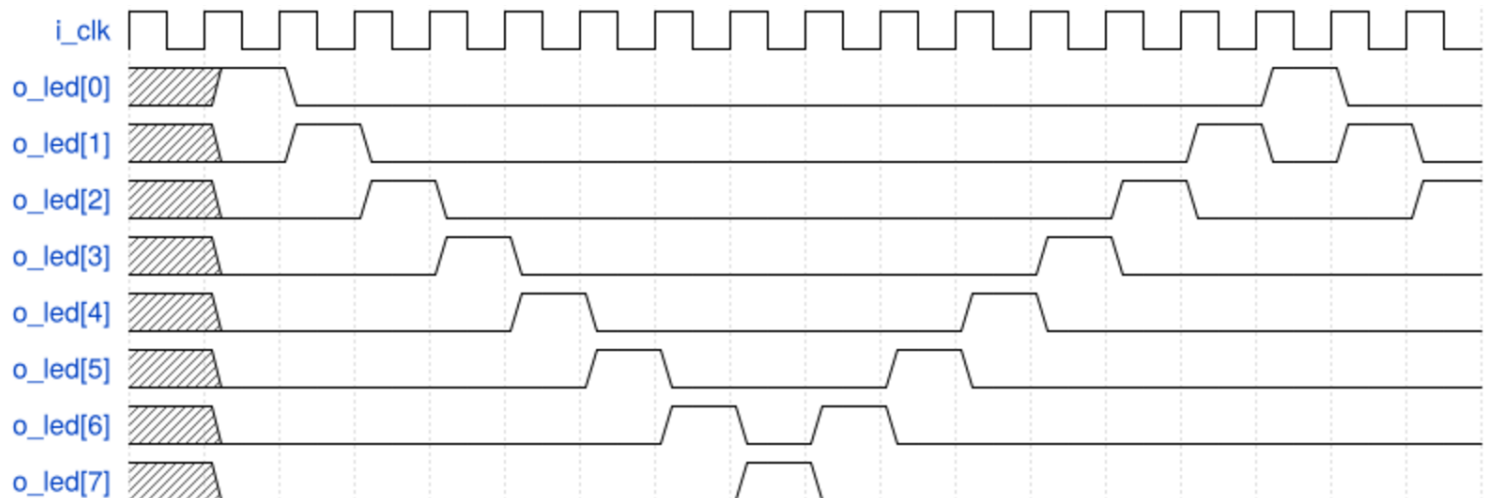
Integer Clock

Divider

Exercise

Conclusion

If you've never seen Wavedrom, it is an awesome tool!  
Here's a waveform description of what I want this design to do



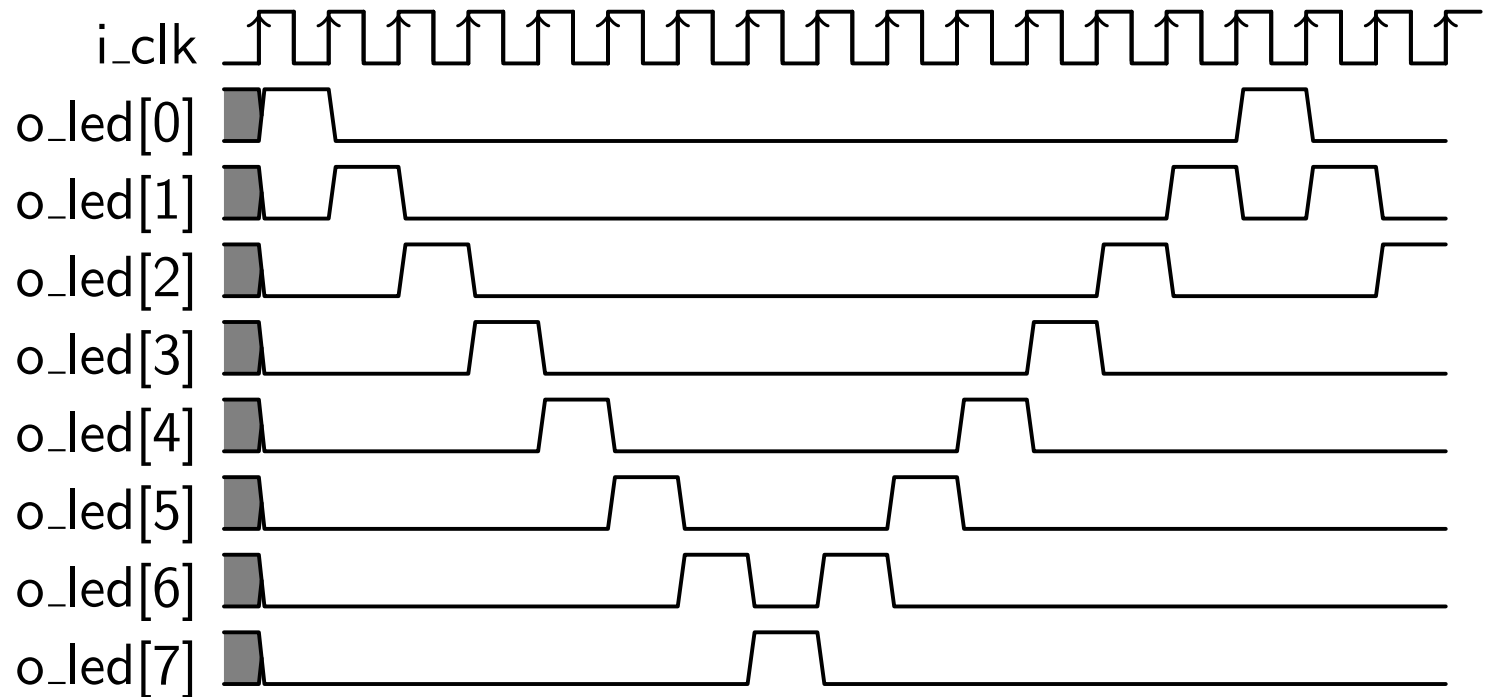
- This “goal” diagram can help mitigate complexity



# Tikz-Timing



Tikz-timing also works nicely for  $\text{\LaTeX}$  users



- Our goal will be to create a design with these outputs
- If successful, you'll see this in GTKwave



# The Need



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

▷ The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

Were we building in C, this would be our program

```
while (1) {  
    o_led = 0x01;  
    o_led = 0x02;  
    o_led = 0x04;  
    // ...  
    o_led = 0x80;  
    o_led = 0x40;  
    // ...  
    o_led = 0x04;  
    o_led = 0x02;  
}
```

How do we turn this code into Verilog?



# Case Statement



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

▷ Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

We could use a giant cascaded **if** statement

```
always @(posedge i_clk)
if (o_led == 8'b0000_0001)
    o_led <= 8'h02;
else if (o_led == 8'b0000_0010)
    o_led <= 8'h04;
else if (o_led == 8'b0000_0100)
    o_led <= 8'h08;
else if (o_led == 8'b0000_1000)
    o_led <= 8'h08;

// ...
// Don't forget a final else!
else // if (o_led == 8'b0000_0010)
    o_led <= 8'h01
```



# Case Statement



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

▷ Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

We could use a giant case statement

```
always @(posedge i_clk)
case(o_led)
8'b0000_0001: o_led <= 8'h02;
8'b0000_0010: o_led <= 8'h04;
// ...
8'b0010_0000: o_led <= 8'h40;
8'b0100_0000: o_led <= 8'h80;
8'b1000_0000: o_led <= 8'h40;
// ...
8'b0000_0100: o_led <= 8'h02;
8'b0000_0010: o_led <= 8'h01;
default: o_led <= 8'h01;
endcase
```

Can anyone see a problem with these two approaches?



# The Need



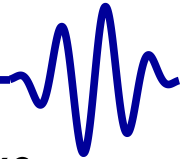
A better way: Let's assign an index to each of these outputs

```
// ... using C++ notation again
o_led = 0x01;    // 1
o_led = 0x02;    // 2
o_led = 0x04;    // 3
// ...
o_led = 0x80;    // 8
o_led = 0x40;    // 9
// ...
o_led = 0x04;    // 13
o_led = 0x02;    // 14
```

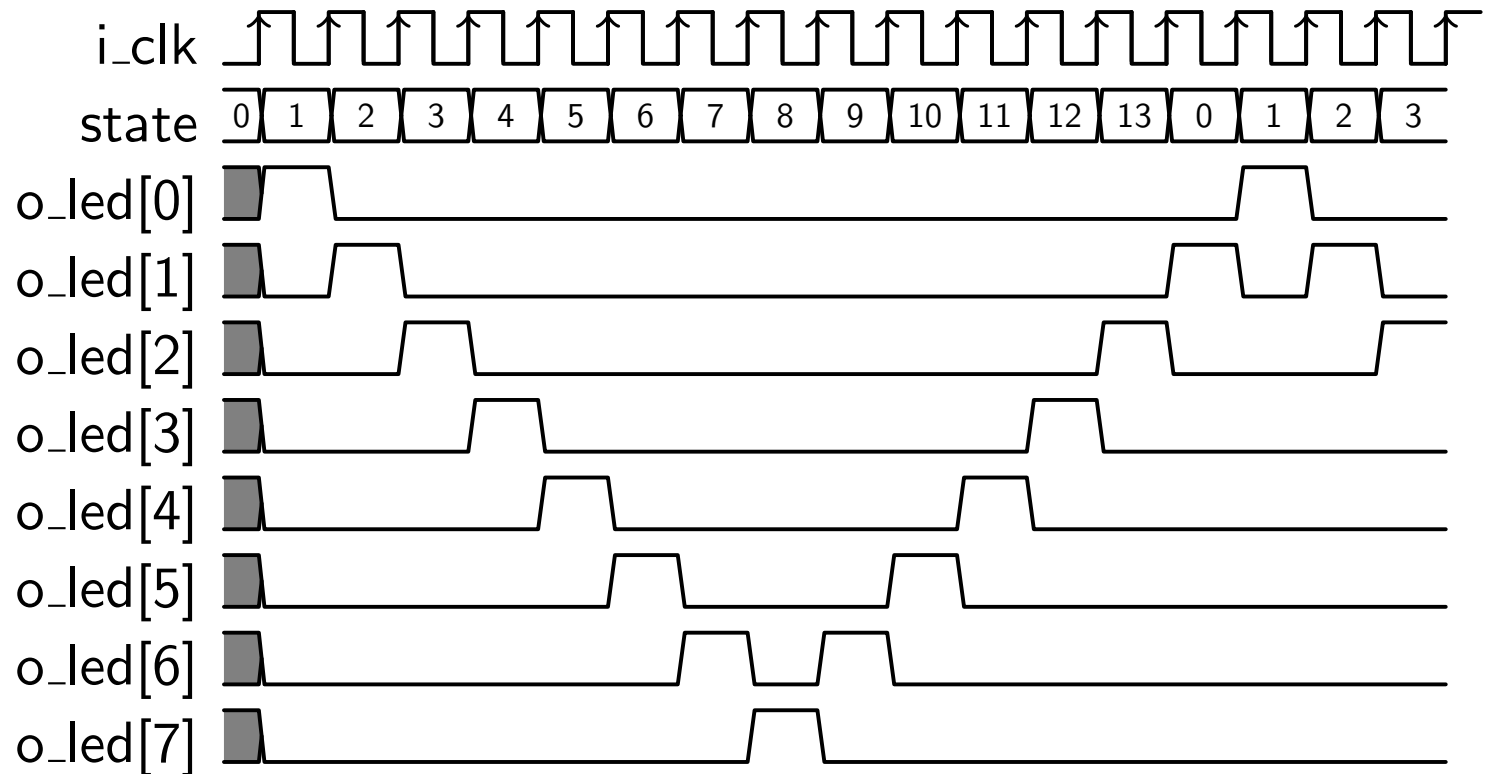
In software, you might think of this as an *instruction address*



# Tikz-Timing



Here's what an updated waveform diagram might look like



- Our goal will be to create a design with these outputs
- If successful, you'll see this in GTKwave



# The Need



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

▷ The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

We can now set the result based upon the *instruction address*

```
always @(posedge i_clk)
case(led_index)
4'h0:    o_led <= 8'h01;
4'h1:    o_led <= 8'h02;
4'h2:    o_led <= 8'h04;
// ...
4'h7:    o_led <= 8'h80;
4'h8:    o_led <= 8'h40;
// ...
4'hc:    o_led <= 8'h02;
4'hd:    o_led <= 8'h01;
default: o_led <= 8'h01;
endcase
```

- This is an example of a *finite state machine*





# The addresses



All we need now is something to drive the *instruction address*

- This is known as the *state* of our finite state machine

```
initial led_index = 0; // Our "state" variable
always @(posedge i_clk)
if (led_index >= 4'd13)
    led_index <= 0;
else
    led_index <= led_index + 1'b1;
```



# Simulation



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

▷ Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

Go ahead and simulate this design

- Does it work as intended?
- Did we miss anything?



# Finite State Machine



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
  - Finite State
    - ▷ Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

A finite state machine consists of...

- Inputs
- State Variable,

*Finite* means there are a limited number of states

- Outputs,



# Finite State Machine



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
  - Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock Divider
- Exercise
- Conclusion

A finite state machine consists of...

- Inputs // *we didn't have any*
- State Variable, // *led\_index, or addr*

*Finite* means there are a limited number of states

- Outputs, // *o\_led*

Keep it just that simple.



# Simple



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
  - ▷ Simple
  - Mealy
  - Moore
  - One Process FSM
  - Two Process FSM
  - Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

- State machines are conceptually very simple
- We'll ignore the excess math here

## Two classical FSM forms

- Mealy
- Moore

## Two implementation approaches

- One process
- Two process



# Mealy



Outputs depend upon the current state *plus inputs*

```
always @(*)  
if (!i_display_enable)  
    o_led = 0;  
  
else  
    case(led_index)  
        4'h1: o_led = 8'h01;  
        4'h2: o_led = 8'h02;  
        4'h3: o_led = 8'h04;  
        4'h4: o_led = 8'h08;  
        // ...  
    endcase
```

- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
  - ▷ Mealy
  - Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion



# Moore



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- ▷ Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

Outputs depend upon the *current* state *only*

```
// Update the state
always @(posedge i_clk)
    enabled <= i_display_enable;

// Create the outputs
always @(*)
    if (!enabled)
        o_led = 0;
    else
        case(led_index)
            4'h1: o_led = 8'h01;
            4'h2: o_led = 8'h02;
            // ...
        endcase
    endcase
```

The inputs are then used to determine the next state



# One Process FSM



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
  - One Process
    - ▷ FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

A one process state machine

- Created with *synchronous* always block(s)

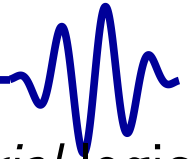
```
initial led_index = 0; // Our "state" variable
always @(posedge i_clk)
begin
    if (led_index >= 4'h0)
        led_index <= 0;
    else
        led_index <= led_index + 1'b1;

    case(led_index)
    4'h0: o_led <= 8'h01;
    // ...
    endcase
end
```





# Two Process FSM



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
  - Two Process
- ▷ FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

Two Process FSM uses both synchronous and *combinatorial* logic

```
always @(*)  
begin  
    if (led_index >= 4'hf)  
        next_led_index = 0;  
    else  
        next_led_index  
            = next_led_index + 1'b1;  
    case(led_index)  
        4'h0: o_led = 8'h01;  
        // ...  
    endcase  
end  
  
always @(posedge i_clk)  
    led_index <= next_led_index;
```



# Which to use?



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
  - Simple
  - Mealy
  - Moore
  - One Process FSM
  - Two Process FSM
  - ▷ Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

Pick whichever finite state machine form ...

- ... you are most comfortable with

There is no right answer here



# Which to use?



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
  - Simple
  - Mealy
  - Moore
  - One Process FSM
  - Two Process FSM
  - ▷ Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

Pick whichever finite state machine form ...

- ... you are most comfortable with

There is no right answer here

*but people still argue about it!*

- Tastes great
- Less Filling

I tend to use one process FSM's



# Formal Verification



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal  
▷ Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

Formal Verification is a process to prove your design “works”

- Fairly easy to use
- Can be faster and easier than simulation
- Most valuable
  - Early in the design process
  - For design *components*, and not entire designs



# Formal Verification



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal

▷ Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

## Formal Verification

- You specify properties your design must have
- A solver attempts to prove if your design has them
- If the solver fails
  - It will tell you what property failed  
By line number
  - It will generate a trace showing the failure
- These traces tend to be much shorter than simulation failure traces



# Assertion



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

▷ Assertion

SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

The free version of Yosys supports immediate assertions

Two types

- Clocked – only checked on clock edges

```
// Remember how we only  
// used some of the states?  
always @(posedge i_clk)  
    assert(led_state <= 4'd13);
```

- Combinational – always checked

```
always @(*)  
    assert(led_state <= 4'd13);
```



# SymbiYosys



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- ▷ SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

To verify this design using SymbiYosys,

- You'll need a script

```
[ options ]
mode prove

[ engines ]
smtbmc

[ script ]
read -formal ledwalker.v
# ... other files would go here
prep -top ledwalker

[ files ]
# List all files and relative paths here
ledwalker.v
```



# Three Basic FV Modes



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- ▷ SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

## 1. BMC

```
[ options ]
```

```
mode bmc
```

```
depth 20
```

- Examines the first  $N$  steps (20 in this case)
- ...looking for a way to break your assertion(s)
- Can find property (i.e. **assert**) failures
- An **assert** is a *safety* property
  - Succeeds only if *no trace* can be found that makes any one of your assertions fail





# Three Basic FV Modes



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
  - ▷ SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

1. BMC
2. Cover

```
[ options ]  
mode cover  
depth 20
```

- Examines the first  $N$  steps (20 in this case)
- ... looking for a way to make any cover statement *pass*

```
always @(posedge i_clk)  
    cover(led_state == 4'he);
```

- No trace will be generated if no way is found
- **cover** is a *liveness* property

Succeeds if one trace, any trace, can be found to make the statement *true*



# Three Basic FV Modes



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- ▷ SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion

1. BMC
2. Cover
3. Full proof using  $k$ -induction

```
[ options ]  
mode prove  
depth 20
```

- Examines the first  $N$  steps (20 in this case)
- Also examines an arbitrary  $N$  steps  
*starting in an arbitrary state*

The induction step will ignore your **initial** statements  
Correct functionality must be guaranteed using **assert** statements

- Can prove your properties hold for all time
- This is also a *safety* property check



# Example property



Assert the design can only contain one of eight outputs

```
always @(*)  
begin  
    f_valid_output = 0;  
    case(o_led)  
        8'h01: f_valid_output = 1'b1;  
        8'h02: f_valid_output = 1'b1;  
        8'h04: f_valid_output = 1'b1;  
        8'h08: f_valid_output = 1'b1;  
        8'h10: f_valid_output = 1'b1;  
        8'h20: f_valid_output = 1'b1;  
        8'h40: f_valid_output = 1'b1;  
        8'h80: f_valid_output = 1'b1;  
    endcase  
    assert(f_valid_output);  
end
```

- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
  - ▷ SymbiYosys
- Integer Clock
- Divider
- Exercise
- Conclusion



# It doesn't work



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

▷ SymbiYosys

Integer Clock

Divider

Exercise

Conclusion

If you try implementing this design as it is now,

- You'll be disappointed
- All the LED's will light dimly

*The LED's will toggle so fast you cannot see them change*

We need a way to slow this down.



# Integer Clock Divider



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

▷ Divider

Exercise

Conclusion

You may remember the integer clock divider

- Let's use it here

```
always @(posedge i_clk)
  if (wait_counter == 0)
    wait_counter <= CLK_RATE_HZ - 1;
  else
    wait_counter <= wait_counter - 1'b1;

always @(posedge i_clk)
begin
  stb <= 1'b0;
  if (wait_counter == 0)
    stb <= 1'b1;
end
```



# Integer Clock Divider



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

▷ Divider

Exercise

Conclusion

This wait\_counter is limited in range

- It will only range from 0 to CLK\_RATE\_HZ-1
- Don't forget the assertion that wait\_counter remains in range!

```
always @(posedge i_clk)
    assert(wait_counter <= CLK_RATE_HZ - 1);
```

*If your state variable can only take on some values, always make an assertion to that affect*

- Let's also make sure the stb matches the wait\_counter too

```
always @(posedge i_clk)
    assert(stb == (wait_counter == 0));
```



# Integer Clock Divider



Now we can use `stb` to tell us when to adjust our state

```
initial led_index = 0;
always @(posedge i_clk)
if (stb)
begin
    // The logic inside is just
    // what it was before
    // Only the if(stb) changed
    if (led_index >= 4'd13)
        led_index <= 0;
    else
        led_index <= led_index + 1'b1;
end // else nothing changes
// wait for stb to be true before changing state
```

- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
  - Integer Clock Divider
- Exercise
- Conclusion



# Exercise



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- ▷ Exercise
- Conclusion

Try out the tools

1. Recreate this waveform using wavedrom





# Exercise



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

▷ Exercise

Conclusion

Try out the tools

1. Recreate this waveform using wavedrom
2. Simulate this design
  - **printf** o\_led anytime it changes
  - Look at the trace in gtkwave  
*Does it match our design goal?*  
*Don't forget to slow it down!*



# Exercise



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- ▷ Exercise
- Conclusion

## Run the tools

1. Recreate this waveform using wavedrom
2. Simulate this design
3. Run SymbiYosys

*Does this design pass?*

If it passes, try **assert**(led\_index <= 4);

Examine the resulting waveform



# Exercise



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- ▷ Exercise
- Conclusion

## Run the tools

1. Recreate this waveform using wavedrom
2. Simulate this design
3. Run SymbiYosys

*Does this design pass?*

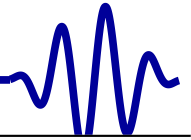
If it passes, try **assert**(led\_index <= 4);

Examine the resulting waveform

*Let's do this one together*



# Running Verilator



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock Divider
- ▷ Exercise
- Conclusion

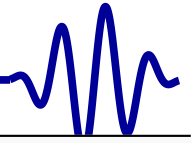
```
% verilator -Wall -cc ledwalker.v
%Error: ledwalker.v:61: Can't find definition
      of variable: o_led
%Error: Exiting due to 1 error(s)
%Error: Command Failed /usr/bin/verilator_bin
      -Wall -cc ledwalker.v
%
```

- ❑ Oops, we misspelled `o_led` in our case statement
- ❑ We also forgot to start our file with **'default\_nettype none**
- ❑ Once fixed, we pass the Verilator check

```
% verilator -Wall -cc ledwalker.v
%
```



# Running SymbiYosys



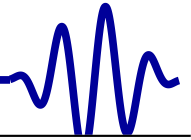
```
% sby -f ledwalker.sby
```

```
/ex-03-walker$ sby -f ledwalker.sby
SBY 21:11:51 [ledwalker] Removing directory 'ledwalker'.
SBY 21:11:51 [ledwalker] Copy 'ledwalker.v' to 'ledwalker/src/ledwalker.v'.
SBY 21:11:51 [ledwalker] engine_0: smtbmc
SBY 21:11:51 [ledwalker] base: starting process "cd ledwalker/src; yosys -ql ../model/design.log ../model/design.js"
SBY 21:11:51 [ledwalker] base: ledwalker.v:71: ERROR: Identifier '\led state' is implicitly declared and 'default nettype is set to none.
SBY 21:11:51 [ledwalker] base: finished (returncode=1)
SBY 21:11:51 [ledwalker] base: job failed. ERROR.
SBY 21:11:51 [ledwalker] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 21:11:51 [ledwalker] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 21:11:51 [ledwalker] DONE ERROR rc=16)
/ex-03-walker$
```

- Another syntax error, mislabeled `led_index` as `led_state`
- Let's try again



# Running SymbiYosys



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock Divider
- ▷ Exercise
- Conclusion

```
% sby -f ledwalker.sby
```

```
SBY 21:14:15 [ledwalker] engine_0.induction: ## 0:00:00 Trying induction in step 5..
SBY 21:14:15 [ledwalker] engine_0.induction: ## 0:00:00 Temporal induction successful.
SBY 21:14:15 [ledwalker] engine_0.induction: ## 0:00:00 Status: PASSED
SBY 21:14:15 [ledwalker] engine_0.induction: finished (returncode=0)
SBY 21:14:15 [ledwalker] engine_0: Status returned by engine for induction: PASS
SBY 21:14:15 [ledwalker] engine_0.basecase: ## 0:00:00 Writing trace to constraints file: engine_0/trace.smtc
SBY 21:14:15 [ledwalker] engine_0.basecase: ## 0:00:00 Status: FAILED (!)
SBY 21:14:15 [ledwalker] engine_0.basecase: finished (returncode=-1)
SBY 21:14:15 [ledwalker] engine_0: Status returned by engine for basecase: FAIL
SBY 21:14:15 [ledwalker] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 21:14:15 [ledwalker] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 21:14:15 [ledwalker] summary: engine_0 (smtbmc) returned PASS for induction
SBY 21:14:15 [ledwalker] summary: engine_0 (smtbmc) returned FAIL for basecase
SBY 21:14:15 [ledwalker] summary: counterexample trace: ledwalker/engine_0/trace.vcd
SBY 21:14:15 [ledwalker] DONE (FAIL, rc=2)
/ex-03-walkers$
```

It failed, but how? Need to scroll up for the details



# Running SymbiYosys



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- ▷ Exercise
- Conclusion

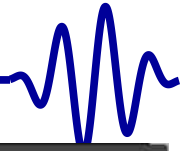
```
trace_induct.smtc model/design_smt2.smt2"
SBY 21:14:15 [ledwalker] engine_0.basecase: ## 0:00:00 Solver: yices
SBY 21:14:15 [ledwalker] engine_0.induction: ## 0:00:00 Solver: yices
SBY 21:14:15 [ledwalker] engine_0.basecase: ## 0:00:00 Checking assumptions i
n step 0..
SBY 21:14:15 [ledwalker] engine_0.basecase: ## 0:00:00 Checking assertions in
step 0..
SBY 21:14:15 [ledwalker] engine_0.induction: ## 0:00:00 Trying induction in s
tep 20..
SBY 21:14:15 [ledwalker] engine_0.basecase: ## 0:00:00 BMC failed!
SBY 21:14:15 [ledwalker] engine_0.basecase: ## 0:00:00 Assert failed in ledwa
lker: ledwalker.v:96
SBY 21:14:15 [ledwalker] engine_0.basecase: ## 0:00:00 Writing trace to VCD f
ile: engine_0/trace.vcd
SBY 21:14:15 [ledwalker] engine_0.induction: ## 0:00:00 Trying induction in s
tep 19..
```

- Fail in line 96
- Trace file in ledwalker/engine\_0/trace.vcd
- Open this in GTKWave, compare to line 96

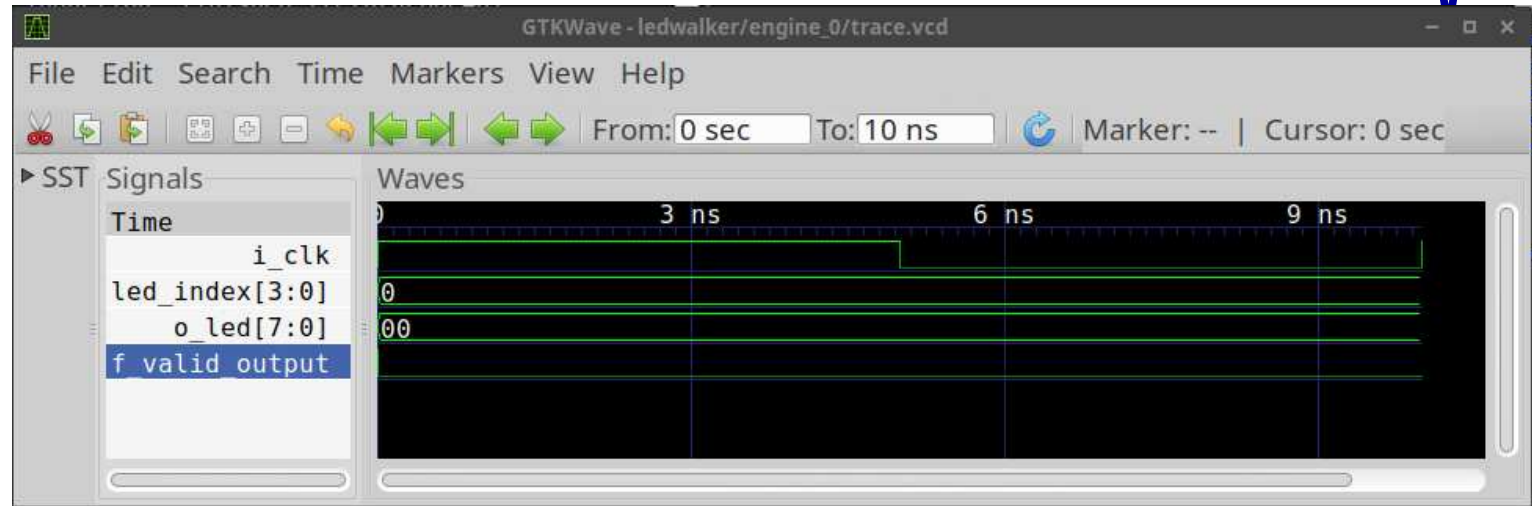




# Running SymbiYosys



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock Divider
- ▷ Exercise
- Conclusion

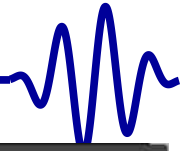


- See the bug?

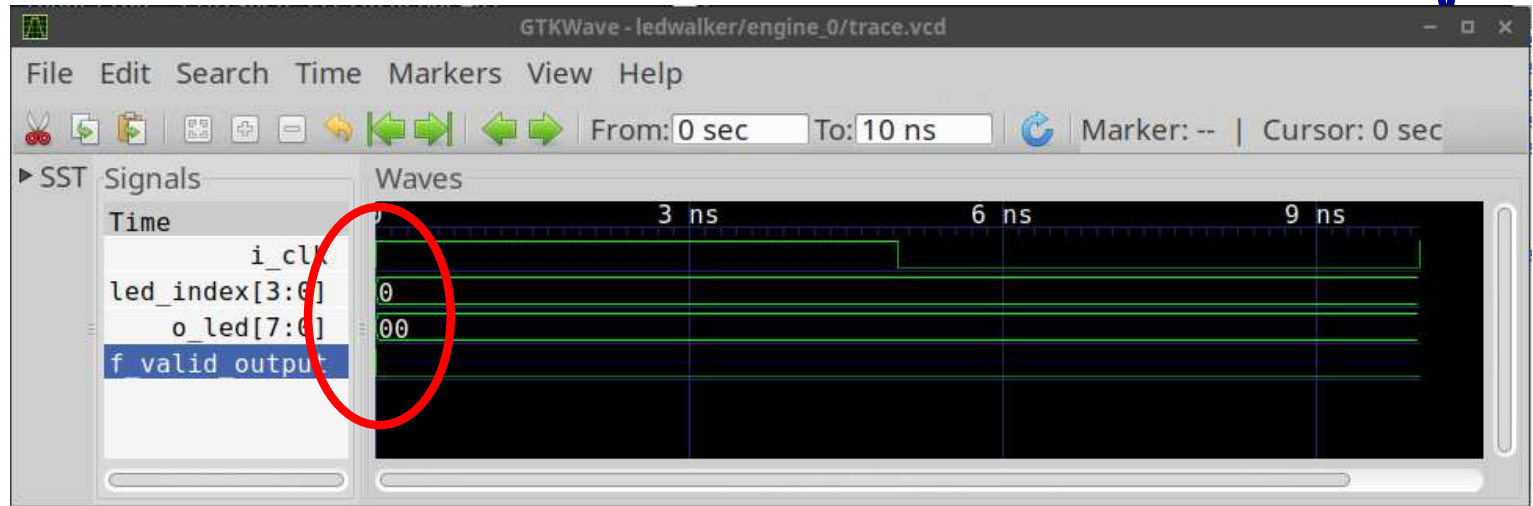




# Running SymbiYosys



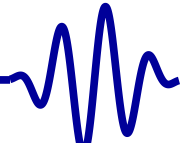
Lesson Overview  
Shift Register  
Wavedrom  
LED Walker  
Wavedrom  
The Need  
Case Statement  
The Need  
The Need  
The addresses  
Simulation  
Finite State Machine  
Simple  
Mealy  
Moore  
One Process FSM  
Two Process FSM  
Which to use?  
Formal Verification  
Assertion  
SymbiYosys  
Integer Clock  
Divider  
▷ Exercise  
Conclusion



- See the bug? `o_led` starts at `8'h00`
- We never initialized `o_led` to a valid value
- **initial** `o_led = 8'h01`; fixes this



# Running SymbiYosys



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- ▷ Exercise
- Conclusion

```
SBY 21:21:37 [ledwalker] engine_0.basecase: ## 0:00:00 Checking assertions in
step 14..
SBY 21:21:37 [ledwalker] engine_0.basecase: ## 0:00:00 BMC failed!
SBY 21:21:37 [ledwalker] engine_0.basecase: ## 0:00:00 Assert failed in ledwa
lker: ledwalker.v:72
SBY 21:21:37 [ledwalker] engine_0.basecase: ## 0:00:00 Writing trace to VCD f
ile: engine_0/trace.vcd
SBY 21:21:37 [ledwalker] engine_0.basecase: ## 0:00:00 Writing trace to Veril
og testbench: engine_0/trace_tb.v
SBY 21:21:37 [ledwalker] engine_0.basecase: ## 0:00:00 Writing trace to const
raints file: engine_0/trace.smtc
SBY 21:21:37 [ledwalker] engine_0.basecase: ## 0:00:00 Status: FAILED (!)
SBY 21:21:37 [ledwalker] engine_0.basecase: finished (returncode=1)
SBY 21:21:37 [ledwalker] engine_0: Status returned by engine for basecase: FAIL
SBY 21:21:37 [ledwalker] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (
0)
SBY 21:21:37 [ledwalker] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00
(0)
SBY 21:21:37 [ledwalker] summary: engine_0 (smtbmc) returned PASS for induction
SBY 21:21:37 [ledwalker] summary: engine_0 (smtbmc) returned FAIL for basecase
SBY 21:21:37 [ledwalker] summary: counterexample trace: ledwalker/engine_0/trace
.vcd
SBY 21:21:37 [ledwalker] DONE (FAIL rc=2)
```

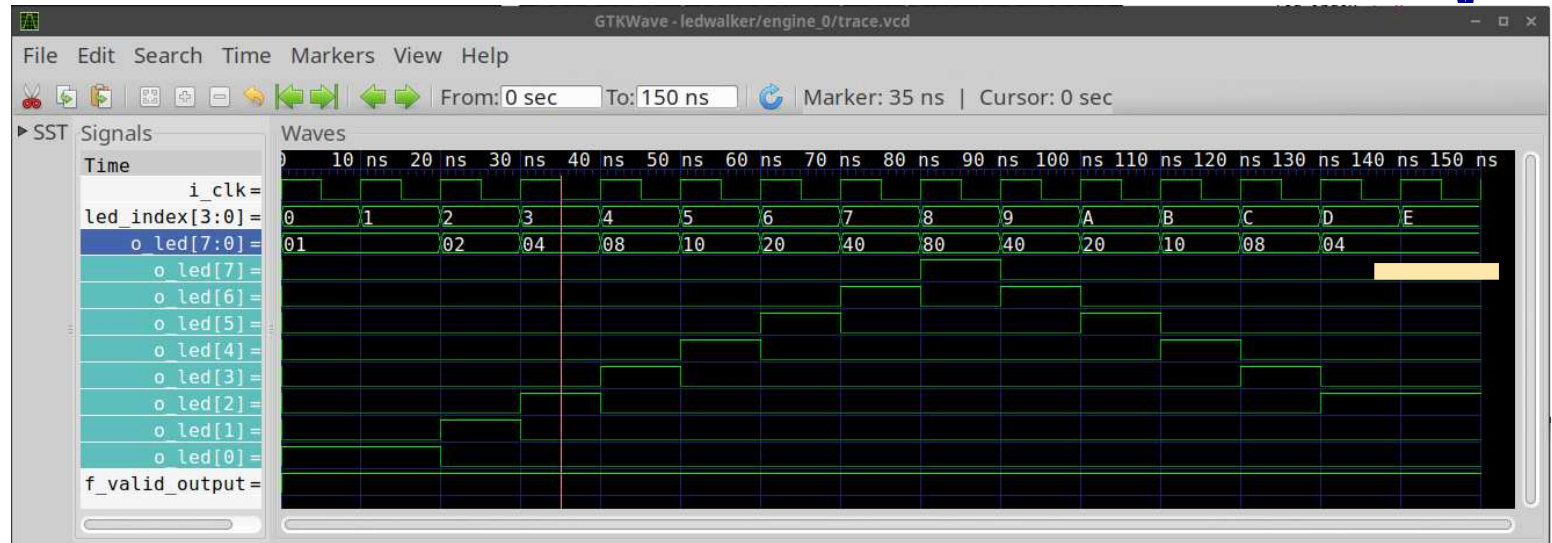
- Same trace file name
- Assertion failed in line 72



# Running SymbiYosys



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- ▷ Exercise
- Conclusion



- **if** (led\_index > 4'd12) in line 39 fixes this

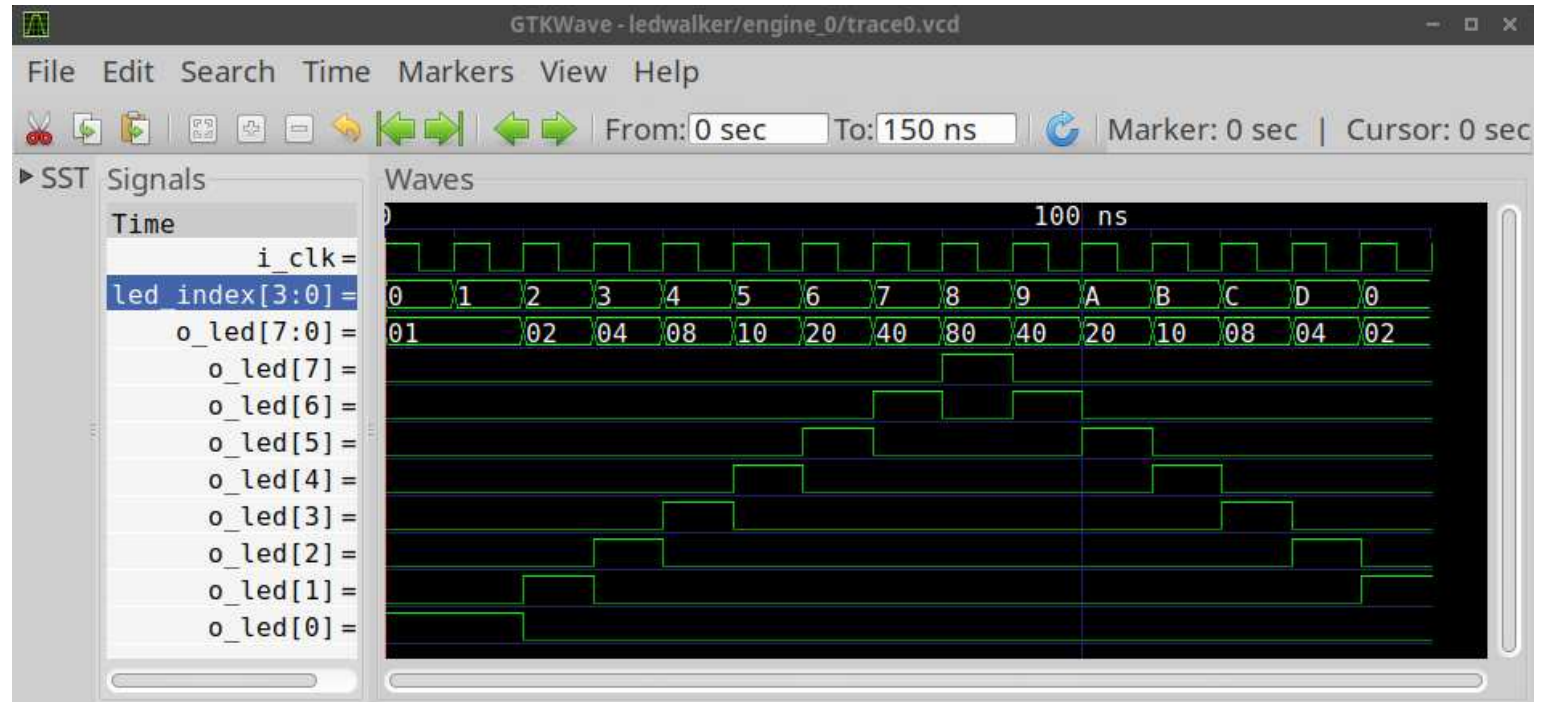


# Cover Property



Let's add a quick cover property

```
always @(*)  
    cover((led_index == 0)&&(o_led == 4'h2));
```





# Exercise



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock Divider
- ▷ Exercise
- Conclusion

Your turn! Run the tools

1. Recreate this waveform using wavedrom
2. Simulate this design
3. Run SymbiYosys
4. Run your device's Synthesis tool
  - Make sure your design ...
    - Passes a timing check
    - Fits within your device
5. Now repeat with the clock divider



# Bonus



- Lesson Overview
- Shift Register
- Wavedrom
- LED Walker
- Wavedrom
- The Need
- Case Statement
- The Need
- The Need
- The addresses
- Simulation
- Finite State Machine
- Simple
- Mealy
- Moore
- One Process FSM
- Two Process FSM
- Which to use?
- Formal Verification
- Assertion
- SymbiYosys
- Integer Clock
- Divider
- ▷ Exercise
- Conclusion

**Bonus:** If you have hardware and more than one LED

- Adjust this design for the number of LEDs you have
  - Implement this on your hardware
- Does it work?*



# Conclusion



Lesson Overview

Shift Register

Wavedrom

LED Walker

Wavedrom

The Need

Case Statement

The Need

The Need

The addresses

Simulation

Finite State Machine

Simple

Mealy

Moore

One Process FSM

Two Process FSM

Which to use?

Formal Verification

Assertion

SymbiYosys

Integer Clock

Divider

Exercise

▷ Conclusion

What did we learn this lesson?

- What a Finite State Machine (FSM) is
- Why FSM's are necessary
- Verilog **case** statement
- Verilog cascaded **if**
- Formal **assert** statement
- How to run SymbiYosys
- How to run slow down an FSM
- Verilog is fun!