ECE270: Embedded Logic Design (ELD)

- The material for this presentation is taken from various books, courses and Xilinx XUP resources. The instructor does not claim ownership of the material presented in this class.

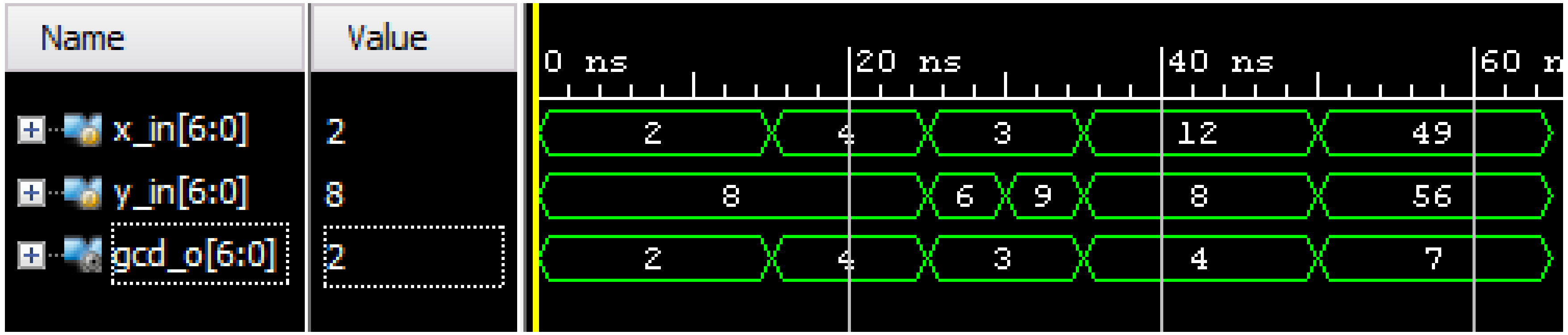
Greatest Common Divisor

- Write Verilog code to find GCD of two numbers X and Y
- Example: GCD (12,8) = 4, GCD(35,49)=7

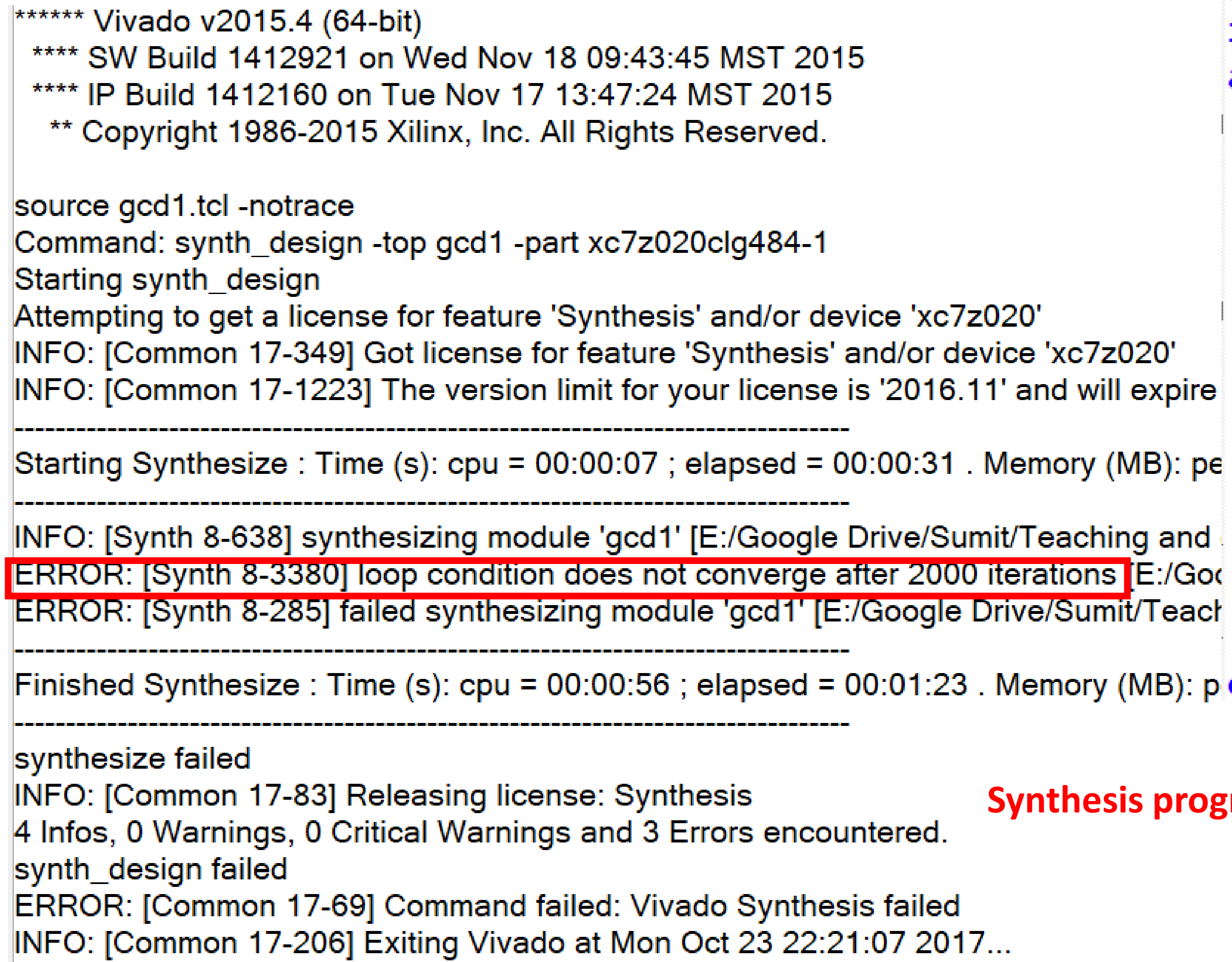
```
module gcd (
input wire [6:0] x_in ,
input wire [6:0] y_in ,
output reg gcd_o
);
reg [6:0] xs , ys;
always @(*)
    begin
      xs = x_in;
      ys = y_in;
      while(xs != ys)
        begin
            if (xs < ys)
                ys = ys - xs;
            else
                xs = xs - ys;
            end
        gcd_o = x_in;
    end
endmodule
```

Greatest Common Divisor (GCD)

```
module gcd (
input wire [6:0] x_in ,
input wire [6:0] y_in ,
output reg gcd_o
);
reg [6:0] xs , ys;
always @(*)
    begin
      xs = x_in;
      ys = y_in;
      while(xs != ys)
        begin
            if (xs < ys)
                ys = ys - xs;
            else
                xs = xs - ys;
            end
        gcd_o = x_in;
    end
endmodule
```

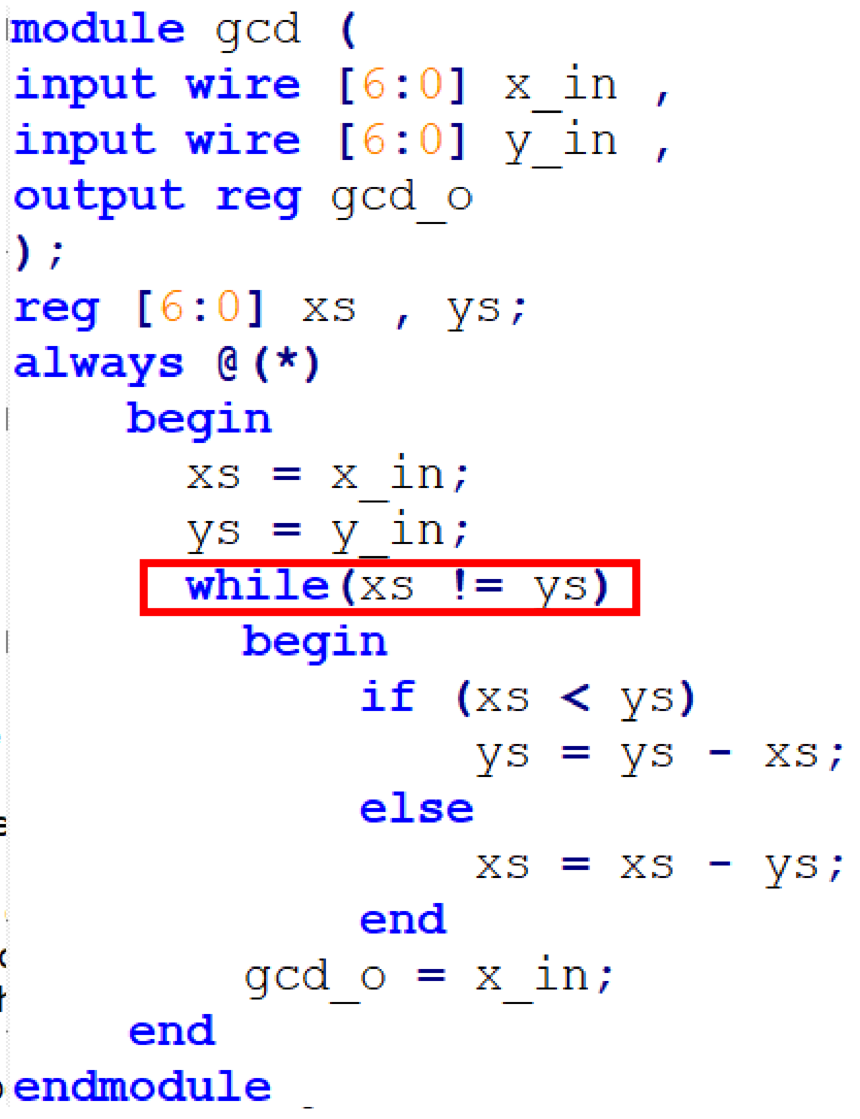


Greatest Common Divisor



```
****** Vivado v2015.4 (64-bit)
  **** SW Build 1412921 on Wed Nov 18 09:43:45 MST 2015
  **** IP Build 1412160 on Tue Nov 17 13:47:24 MST 2015
    ** Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.

source gcd1.tcl -notrace
Command: synth_design -top gcd1 -part xc7z020clg484-1
Starting synth_design
Attempting to get a license for feature 'Synthesis' and/or device 'xc7z020'
INFO: [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7z020'
INFO: [Common 17-1223] The version limit for your license is '2016.11' and will expire
---------------------------------------------------------------------------------
Starting Synthesize : Time (s): cpu = 00:00:07 ; elapsed = 00:00:31 . Memory (MB): pe
---------------------------------------------------------------------------------
INFO: [Synth 8-638] synthesizing module 'gcd1' [E:/Google Drive/Sumit/Teaching and
ERROR: [Synth 8-3380] loop condition does not converge after 2000 iterations [E:/Goo
ERROR: [Synth 8-285] failed synthesizing module 'gcd1' [E:/Google Drive/Sumit/Teach
---------------------------------------------------------------------------------
Finished Synthesize : Time (s): cpu = 00:00:56 ; elapsed = 00:01:23 . Memory (MB): p
---------------------------------------------------------------------------------
synthesize failed
INFO: [Common 17-83] Releasing license: Synthesis
4 Infos, 0 Warnings, 0 Critical Warnings and 3 Errors encountered.
synth_design failed
ERROR: [Common 17-69] Command failed: Vivado Synthesis failed
INFO: [Common 17-206] Exiting Vivado at Mon Oct 23 22:21:07 2017...
```



```
module gcd (
input wire [6:0] x_in ,
input wire [6:0] y_in ,
output reg gcd_o
);
reg [6:0] xs , ys;
always @(*)
    begin
      xs = x_in;
      ys = y_in;
      while(xs != ys)
        begin
          if (xs < ys)
            ys = ys - xs;
          else
            xs = xs - ys;
          end
        gcd_o = x_in;
    end
endmodule
```

Synthesis program tried to iterate 2000 times before giving up.

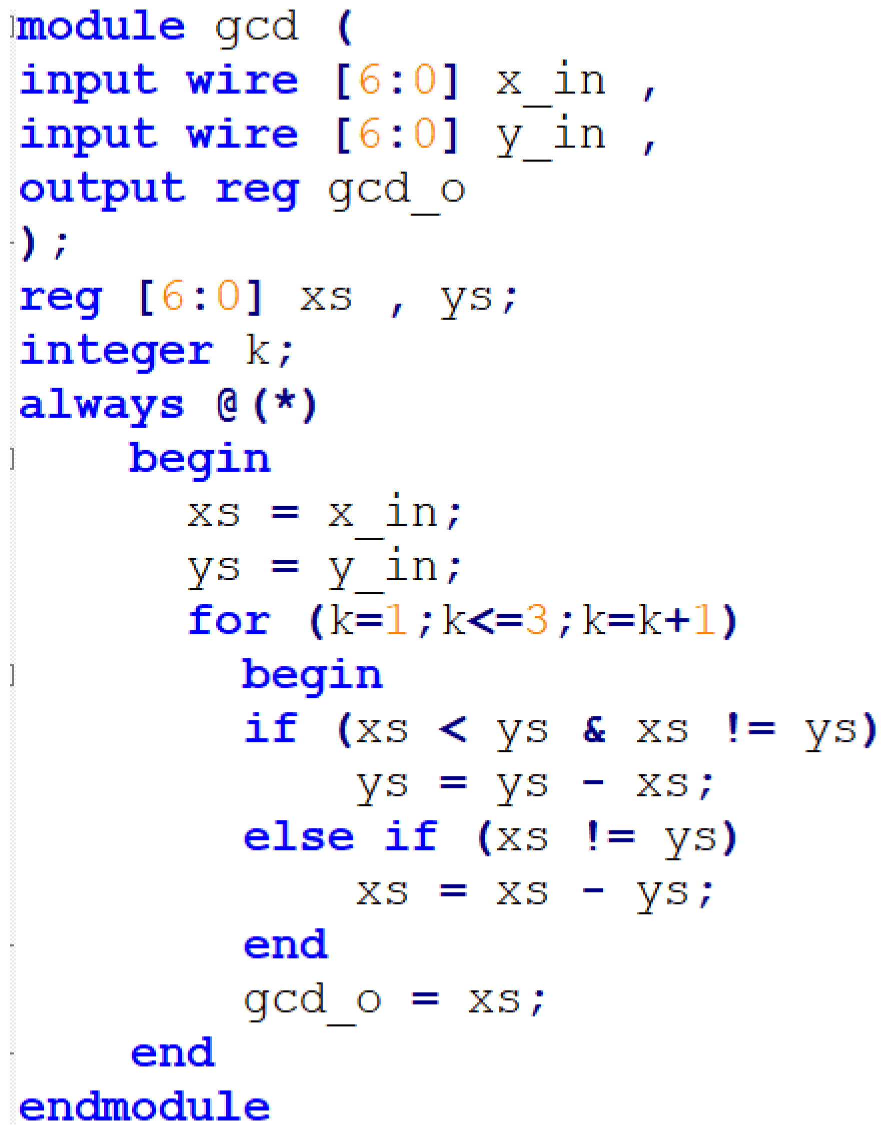
While Loop

- For while loop, number of times the program goes through the loop depends on the given condition which in turn depends on real time values of variables related to the condition.
- Hence, we can’t use while loop if we want to create hardware on the FPGA

While Loop

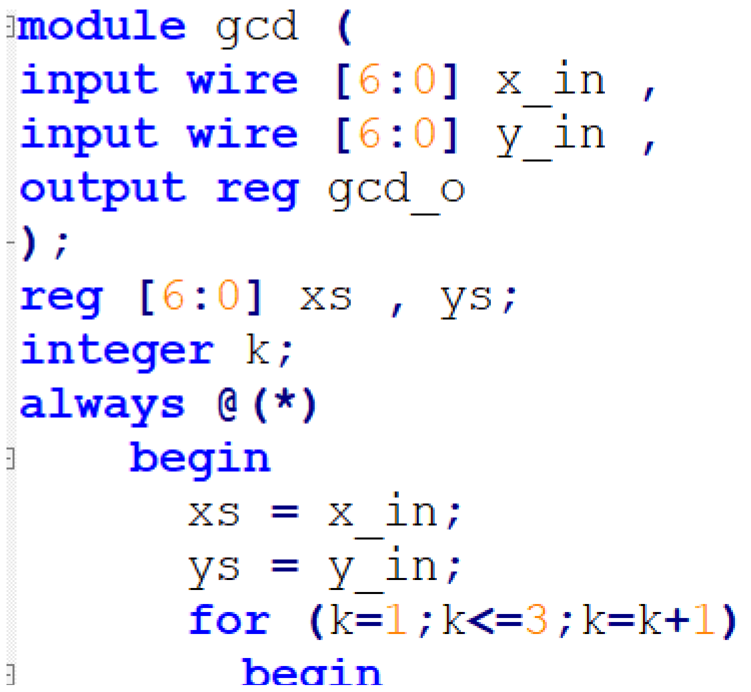
- For while loop, number of times the program goes through the loop depends on the given condition which in turn depends on real time values of variables related to the condition.
- Hence, we can’t use while loop if we want to create hardware on the FPGA
- Without the while loop, most of the algorithms won’t run on FPGA. Hence, we must figure out how to implement these algorithms in hardware

Greatest Common Divisor

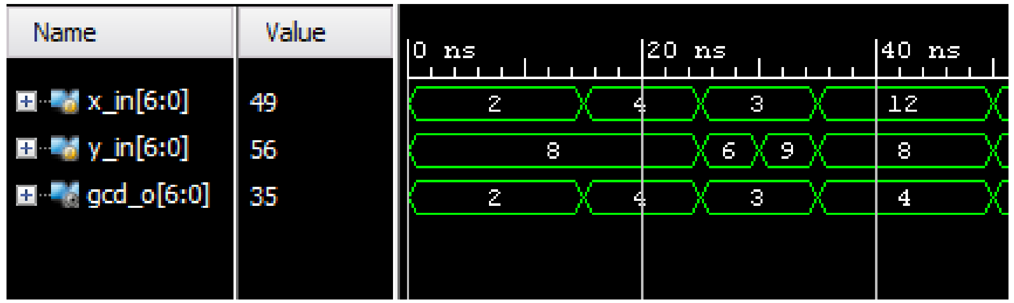


```
module gcd (
input wire [6:0] x_in ,
input wire [6:0] y_in ,
output reg gcd_o
);
reg [6:0] xs , ys;
integer k;
always @(*)
    begin
      xs = x_in;
      ys = y_in;
      for (k=1;k<=3;k=k+1)
        begin
        if (xs < ys & xs != ys)
            ys = ys - xs;
        else if (xs != ys)
            xs = xs - ys;
        end
        gcd_o = xs;
    end
endmodule
```

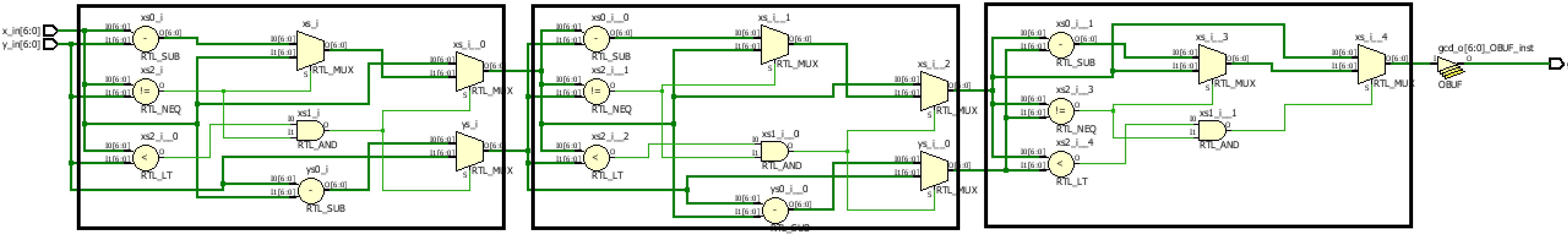
Greatest Common Divisor



```
module gcd (
input wire [6:0] x_in ,
input wire [6:0] y_in ,
output reg gcd_o
);
reg [6:0] xs , ys;
integer k;
always @(*)
    begin
      xs = x_in;
      ys = y_in;
      for (k=1;k<=3;k=k+1)
        begin
```



Name	Value
x_in[6:0]	49
y_in[6:0]	56
gcd_o[6:0]	35



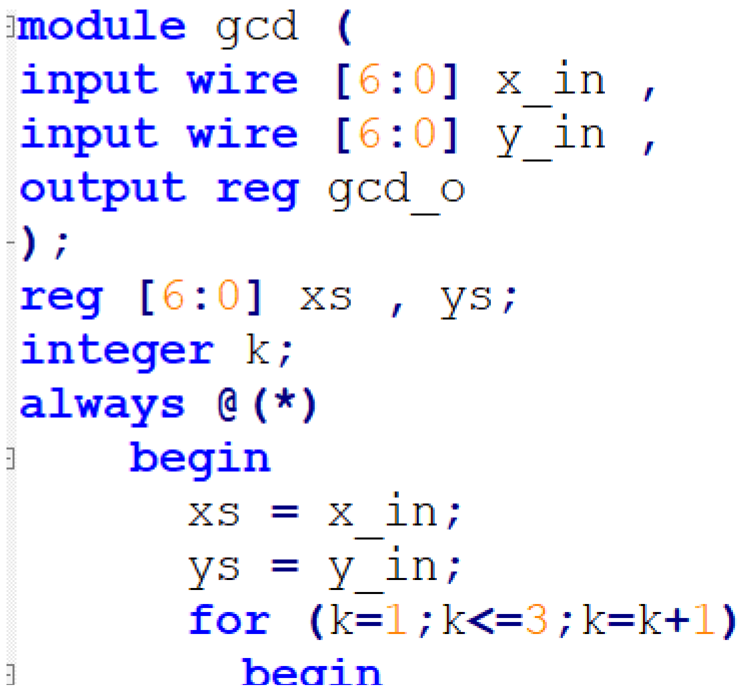
For Loop

- Why for loop is synthesizable?
- When we synthesized *for* loop, the circuits within the *for* loop “unwrapped” to produce new instance of the circuit each time through the loop.
- It is synthesizable because range of *for* loop variable is fixed and known in advance.

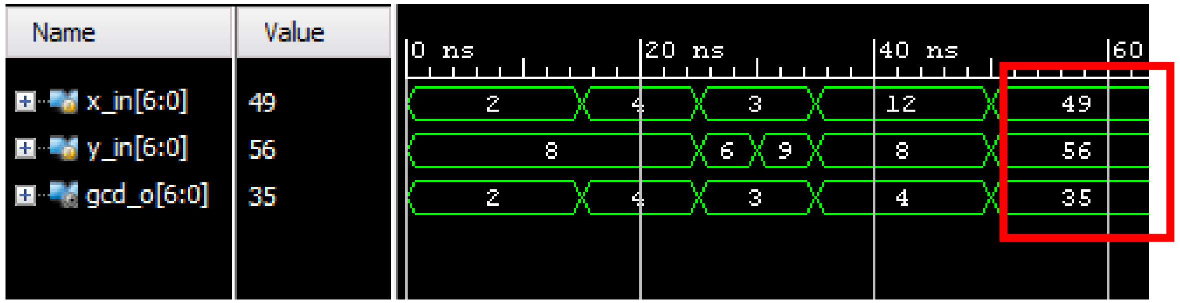
For Loop

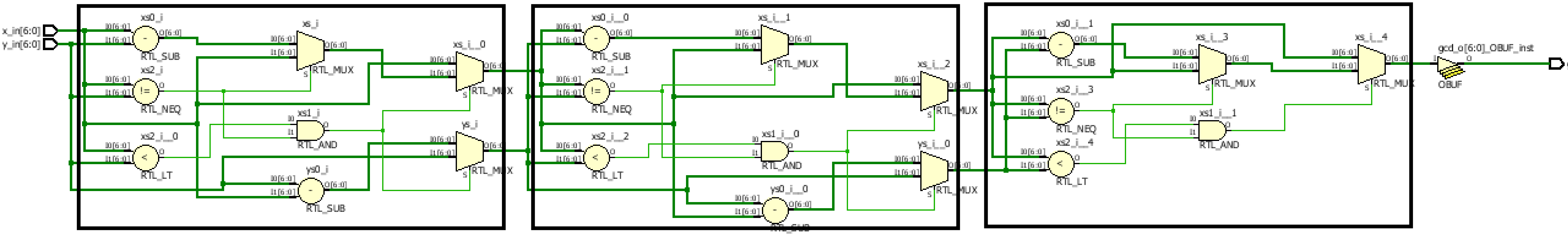
- Why for loop is synthesizable?
- When we synthesized *for* loop, the circuits within the *for* loop “unwrapped” to produce new instance of the circuit each time through the loop.
- It is synthesizable because range of *for* loop variable is fixed and known in advance.
- But while using *for* loops, we must know how many times the program goes through the loop ahead of time so as to synthesize it correctly.
- This may limit the dynamic range of input for a given design.

Verilog: Greatest Common Divisor



```
module gcd (
input wire [6:0] x_in ,
input wire [6:0] y_in ,
output reg gcd_o
);
reg [6:0] xs , ys;
integer k;
always @(*)
    begin
      xs = x_in;
      ys = y_in;
      for (k=1;k<=3;k=k+1)
        begin
```





Lab Homework: Due Nov. 3, 2017

```
module gcd (
input wire clk ,
input wire clr ,
input wire go ,
input wire [3:0] xin ,
input wire [3:0] yin ,
output reg done ,
output reg [3:0] gcd
);
```

- Design the feasible circuit to compute GCD of two number using nested if-else statements and always block.
- Hint: When ‘go’ signal is 1, read the inputs. When ‘go’ signal is zero, start the GCD calculation. You may use the signal ‘calc’ to indicate that GCD operation is going on. When GCD operation is completed, assign the value to output, reset ‘calc’ signal and make ‘done’ signal 1.
- Whenever ‘go’ signal is pressed, inputs should be updated irrespective of the value of ‘calc’ and GCD calculation should restart.
- Everything should happen at clock edge except clear which is asynchronous.

Lab Homework: Due Nov. 3, 2017

- Implement the GCD code discussed in previous slide on Basys 3 board.
- Display X_in and Y_in on the 7-segment display on the board. Display output on PMOD 7-segment display.
- Use pushbuttons for go and clr and switches for inputs (6 switches per input).
- Display done signal on led.
- Use appropriate use of debounce, clock pulse and clock division circuits.