Verilog Lab 2

Data Types for Signals or Variables

- Wire type: physical connections between ports
  (most popular type of input/internal signals)
  - wire
    ```
    wire reset, clock;
    wire [7:0] address;
    ```
- Register types: abstract data storage elements
  (only these types of signals can be on the left-hand side of assignments in procedural blocks)
  - reg: unsigned, varying width (most popular type of input/internal signals)
    ```
    reg carry_out;
    reg [31:0] data_a, data_b;
    ```

Example

```
~~wire out ;~~
reg  out ;
always @(sel or in1 or in2) begin
    if (sel == 1'b1) begin
            out = in1;
    end else begin
            out = in2;
    end
end
```

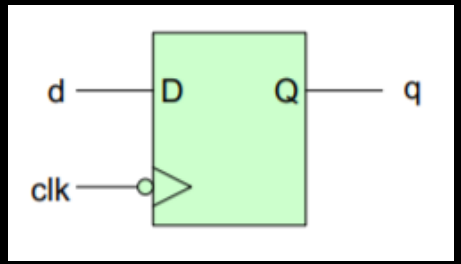
Combinational vs Sequential Circuits

- Sequential circuits
  - contain memory elements and logic gates
  - the outputs are a function of the current inputs and the state of the memory elements.
- Combinational circuits
  - consist of logic gates
  - the outputs at any time are determined from only the present combination of inputs.

Two types of always blocks

- Sequential block
  - is triggered by clock and other signals.

```
always @(posedge clk or negedge rst_n) begin
    if (rst_n== 1'b1) begin
            q<=0;
    end else begin
            q<=d;
    end
end
```



Two types of always blocks

- Combinational block
  - is triggered by the signals in the sensitivity list

Traditional style: full sensitivity list of the always block

```
always @(sel or in1 or in2) begin
    if (sel == 1'b1) begin
        out = in1;
    end else begin
        out = in2;
    end
end
```

Modern coding style:

```
always @* begin
```

You may also use combinational assignment. eg. assign out =(sel)?in1:in2;

Procedural Assignment

- Blocking procedural assignment: =
  - An assignment is completed before the next assignment starts.
  - (assume a = 0)

    a = 1;

    c = a; // c = 1
- Non-blocking procedural assignment: <=
  - Assignments are executed in parallel.
  - (assume a = 0)

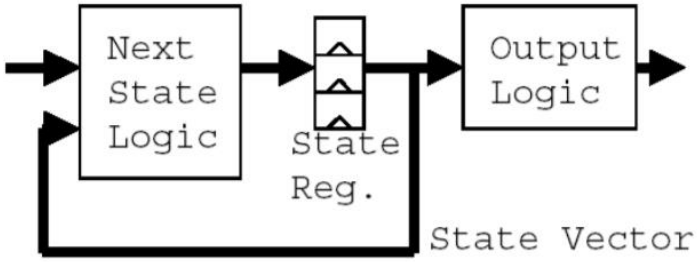
    a <= 1;

    c <= a; // c = 0

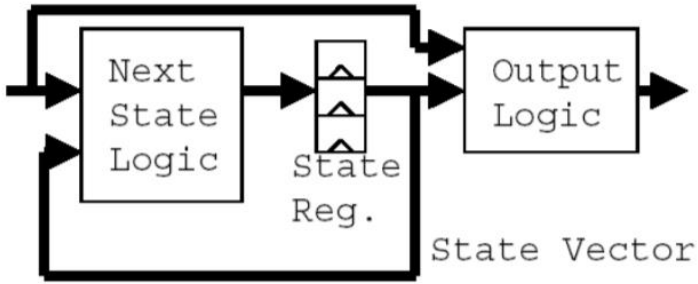
Procedural Assignment

- Do not mix blocking and non-blocking in the same always block.
- Sequential circuits usually use non-blocking.
- Combinational circuits usually use blocking.
- Use blocking assignment for assign.

Moore / Mealy machine



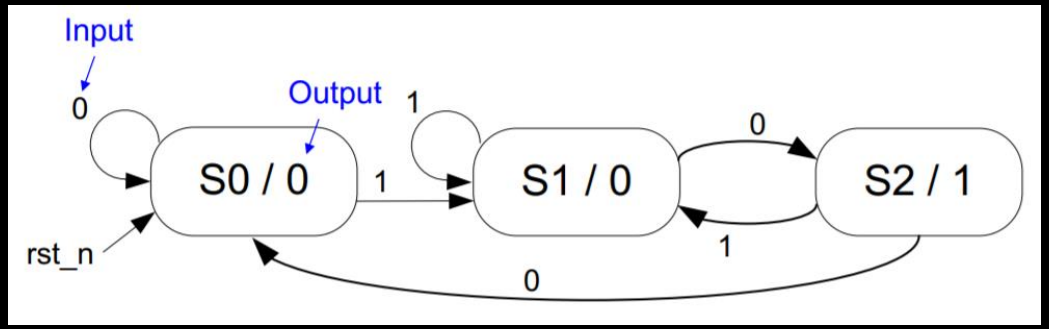
Moore Outputs
Outputs depend solely on state vector (generally, a Moore FSM is the simplest to design)



Mealy Outputs
Outputs depend on inputs and state vector (only use if it is significantly smaller or faster)

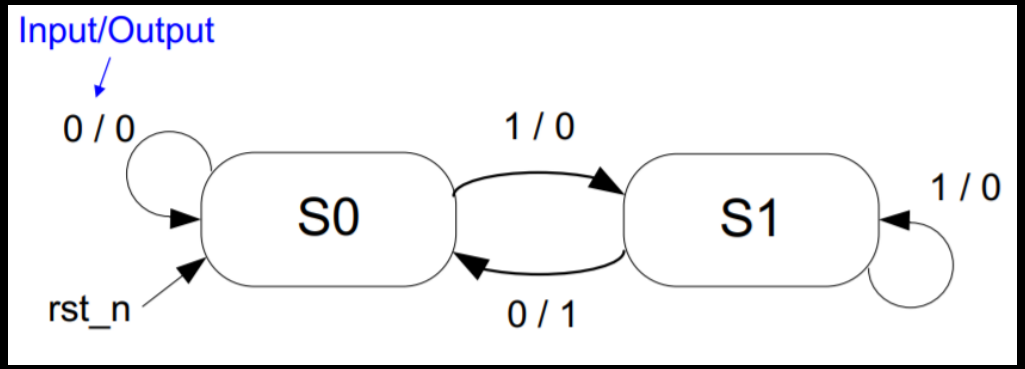
Moore Machine Example

- Recognizing the “10” sequence among the input bit stream



Mealy Machine Example

- Recognizing the “10” sequence among the input bit stream

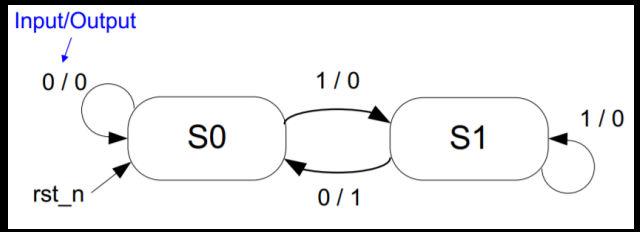


FSM design guideline

- An always block for updating state registers
  - Sequential block
- An always block for next state evaluation (state transition)
  - Combinational block
- Optional always blocks for output generation
  - Combinational block
- Parameterize all state encoding

Mealy Machine Example

```
parameter S0 = 1'b0;
parameter S1 = 1'b1;
reg state, next_state;
always @(posedge clk, negedge rst_n)
begin
        if (rst_n == 1'b0)
                state = S0;
        else
                state = next_state;
end
```




```
always @* begin
        next_state = S0;
        case(state)
            S0: begin
                if (in == 1)
                    next_state = S1;
                else
                    next_state = S0;
                end
            S1: begin
                if (in == 0)
                    next_state = S0;
                else
                    next_state = S1;
                end
         endcase // case end
end // always end
assign out = (state == S1 && in == 0) ? 1 : 0;
```

Assignemt 2

Outline

- Function Description
- Framework Introduction in Block Diagram
- Testbench template
- nWave
- Precautions and Timeline

GCD Engine

- Calculate the greatest common divisor (GCD) of two 8-bit positive integers
- Using a START signal to load inputs
- Generate a DONE signal when the calculation is finished
- Assert an ERROR signal when invalid inputs are detected

GCD Engine

- The IO specification is shown below, and modification is unavailable:

```
module GCD (
    input wire CLK,
    input wire RST_N,
    input wire [7:0] A,
    input wire [7:0] B,
    input wire START,
    output reg [7:0] Y,
    output reg DONE,
    output reg ERROR
);
```

GCD Engine

- Inputs:
  - CLK: clock
  - RST_N: reset (low active)
  - A, B: two 8-bit input numbers
  - START: indicate the valid input with one-cycle pulse
- Outputs:
  - Y: the answer
  - DONE: indicate the valid output with one-cycle pulse
  - ERROR: 0 means valid result while 1 means invalid one (invalid: either A or B is 0 at the start)

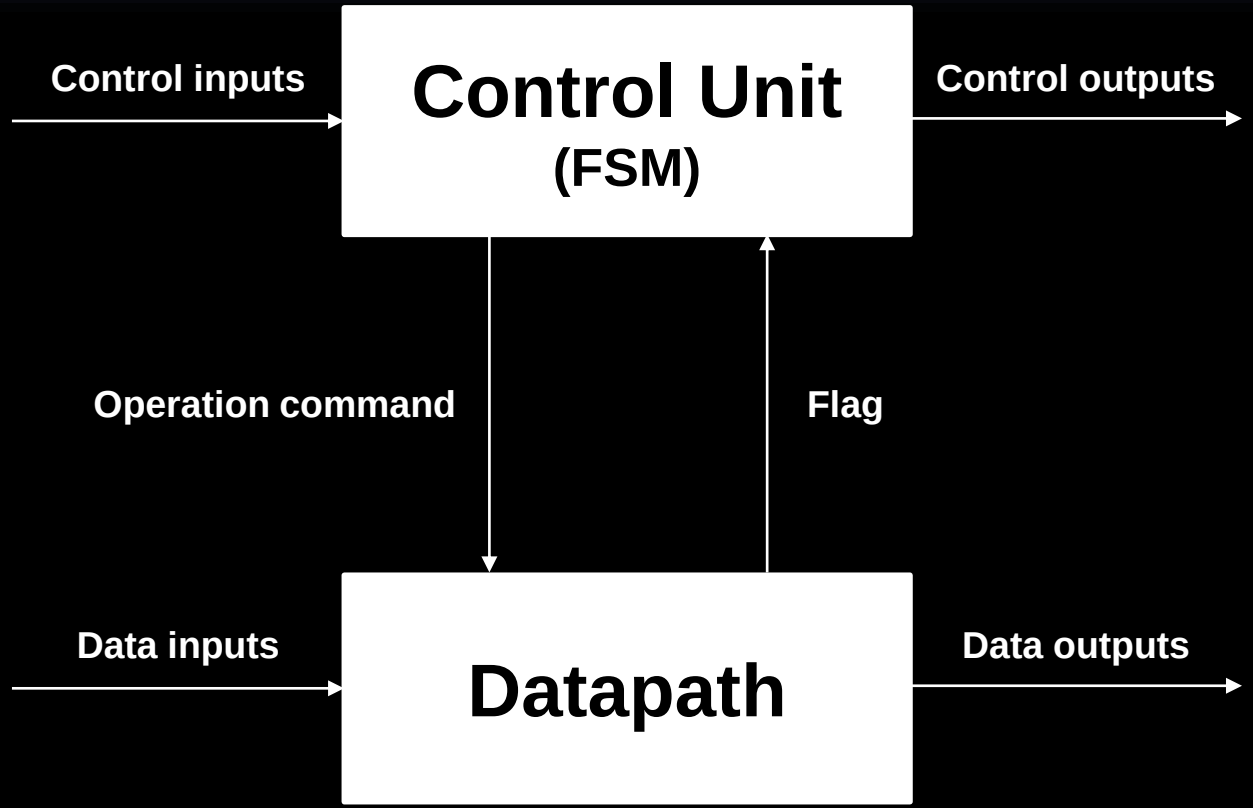
Outline

- Function Description
- Framework Introduction in Block Diagram
- Testbench template
- nWave
- Precautions and Timeline

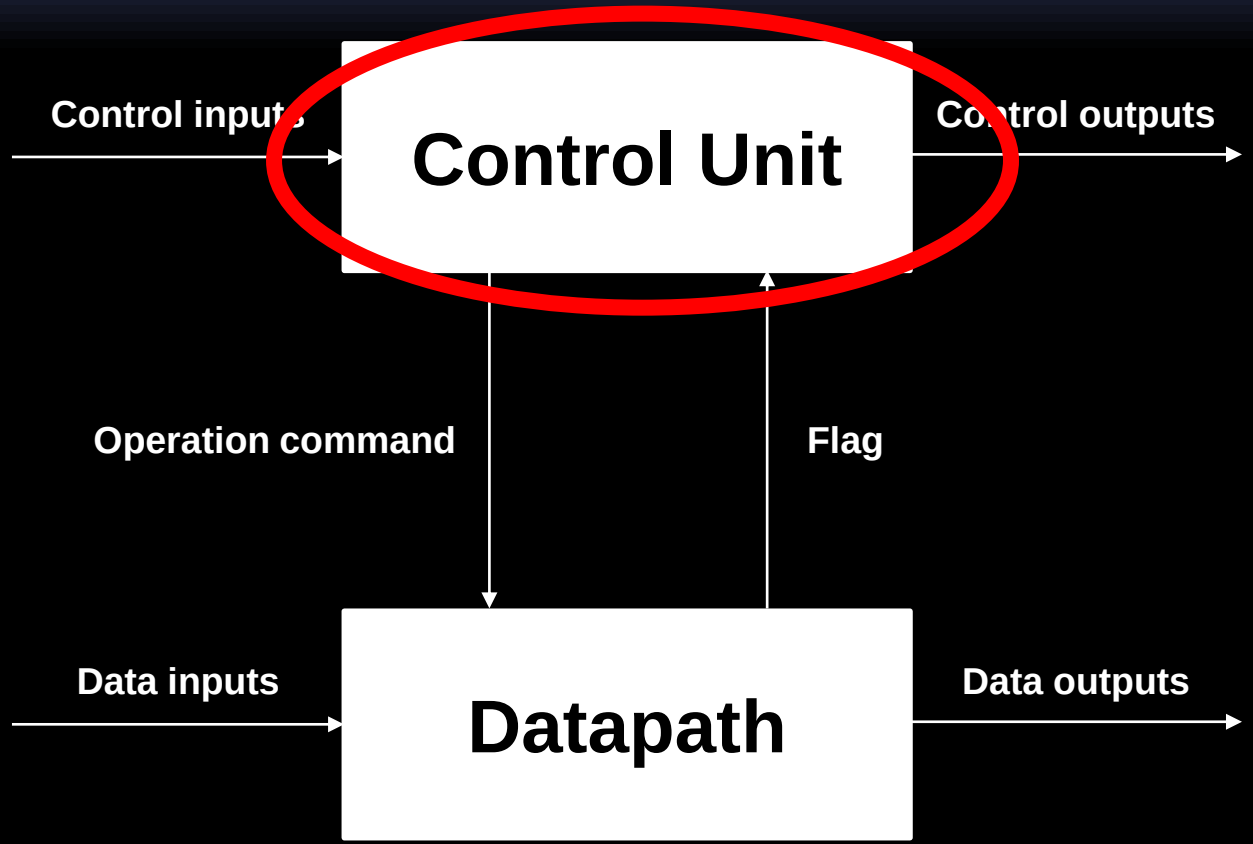
GCD Engine

- Sequential circuits can be partitioned into datapath and control unit
  - Datapath: perform data processing, data registering, and data moving
  - Control unit: behavior control, state(mode) switching

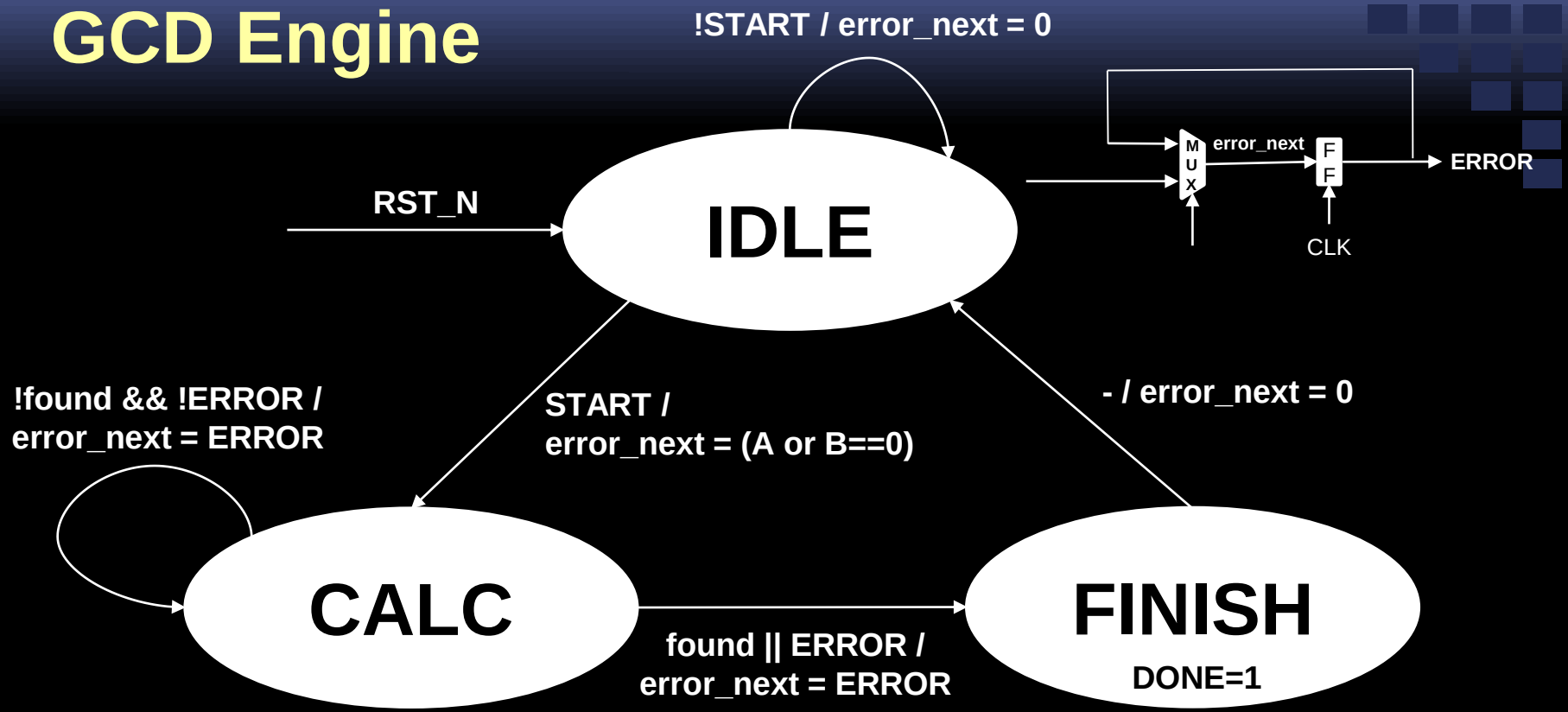
GCD Engine


Control inputs
Control Unit
(FSM)
Control outputs
Operation command
Flag
Data inputs
Datapath
Data outputs

GCD Engine


Control inputs
Control Unit
Control outputs
Operation command
Flag
Data inputs
Datapath
Data outputs

GCD Engine

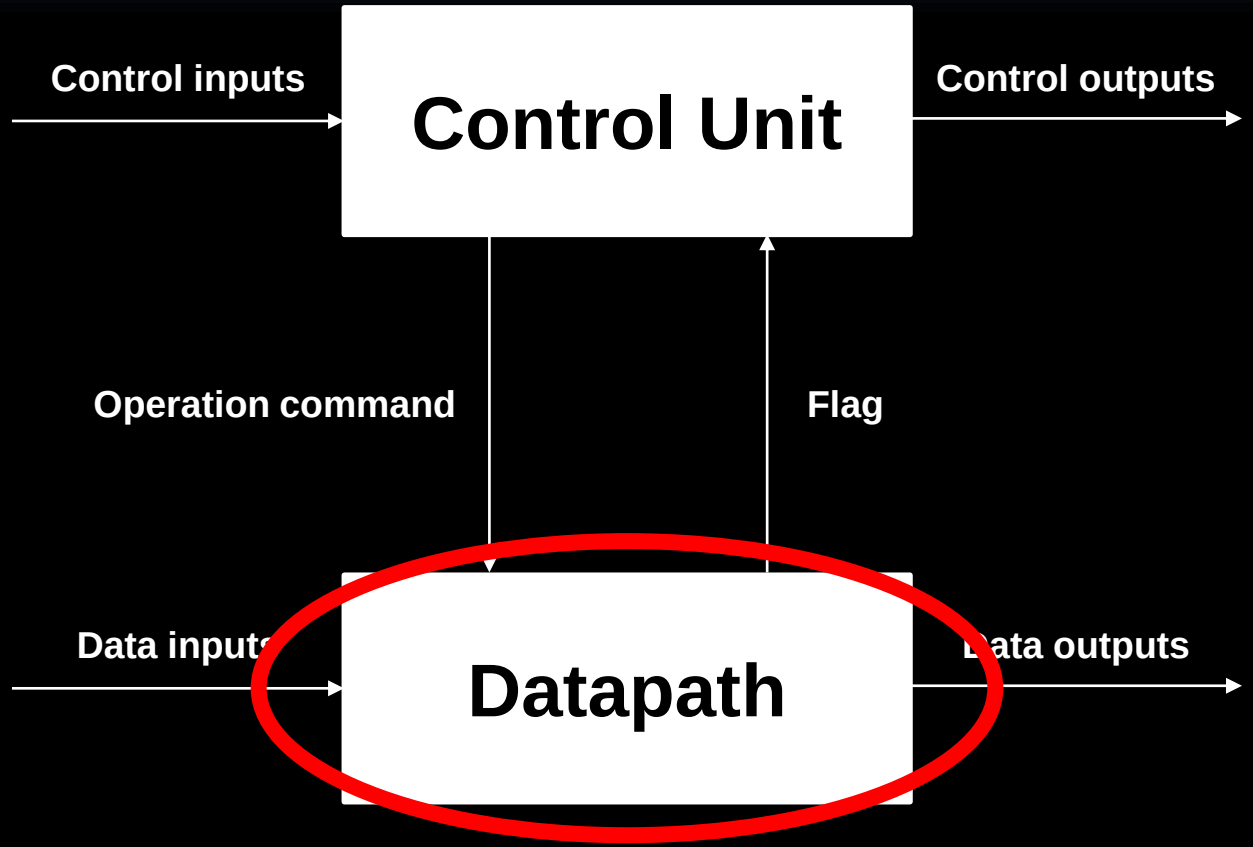

!START / error_next = 0
RST_N
IDLE
MUX
error_next
FF
ERROR
CLK
!found && !ERROR /
error_next = ERROR
START /
error_next = (A or B==0)
- / error_next = 0
CALC
found || ERROR /
error_next = ERROR
FINISH
DONE=1

GCD Engine

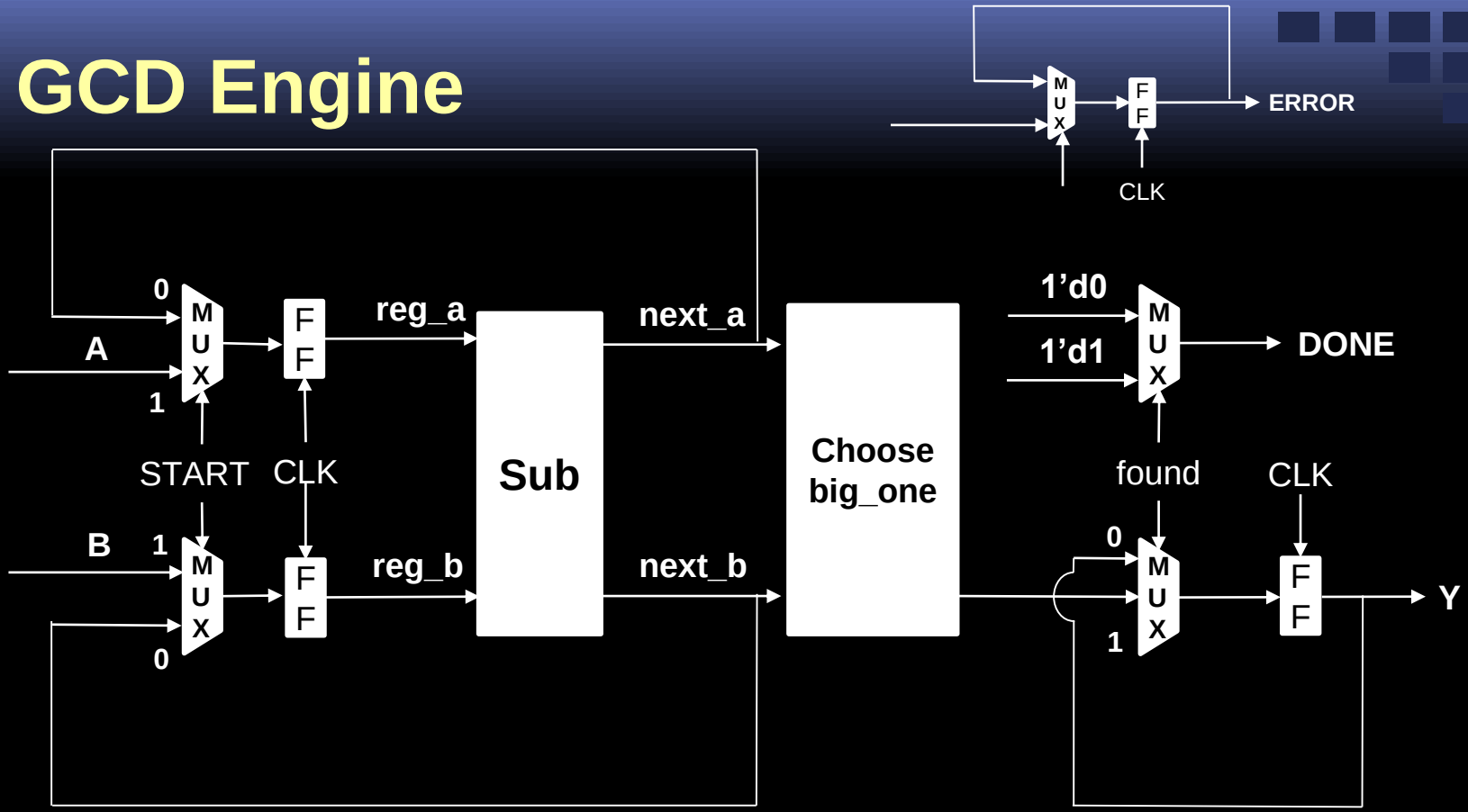
- Your design “must” include these components:

```
parameter [1:0] IDLE = 2'b00;
parameter [1:0] CALC = 2'b01;
parameter [1:0] FINISH = 2'b10;
```

GCD Engine


Control inputs
Control Unit
Control outputs
Operation command
Flag
Data inputs
Datapath
Data outputs

GCD Engine



Warning: This is just a simplified version for easy understanding, the practical implementation can be more complicated.

Example

- (40,12)->(28,12)->(16,12)->(4,12)->(4,8)->(4,4)->(4,0)
- 4 is the answer

```
assign found =(next_a==0 ||next_b==0)?1:0;
```

GCD Engine

- Your design “must” include these components:

```
wire found, err
reg [7:0] reg_a, reg_b, next_a, next_b;
reg [7:0] big_one;
reg error_next;
reg [1:0] state, state_next;
```

Outline

- Function Description
- Framework Introduction in Block Diagram
- Testbench template
- nWave
- Precautions and Timeline

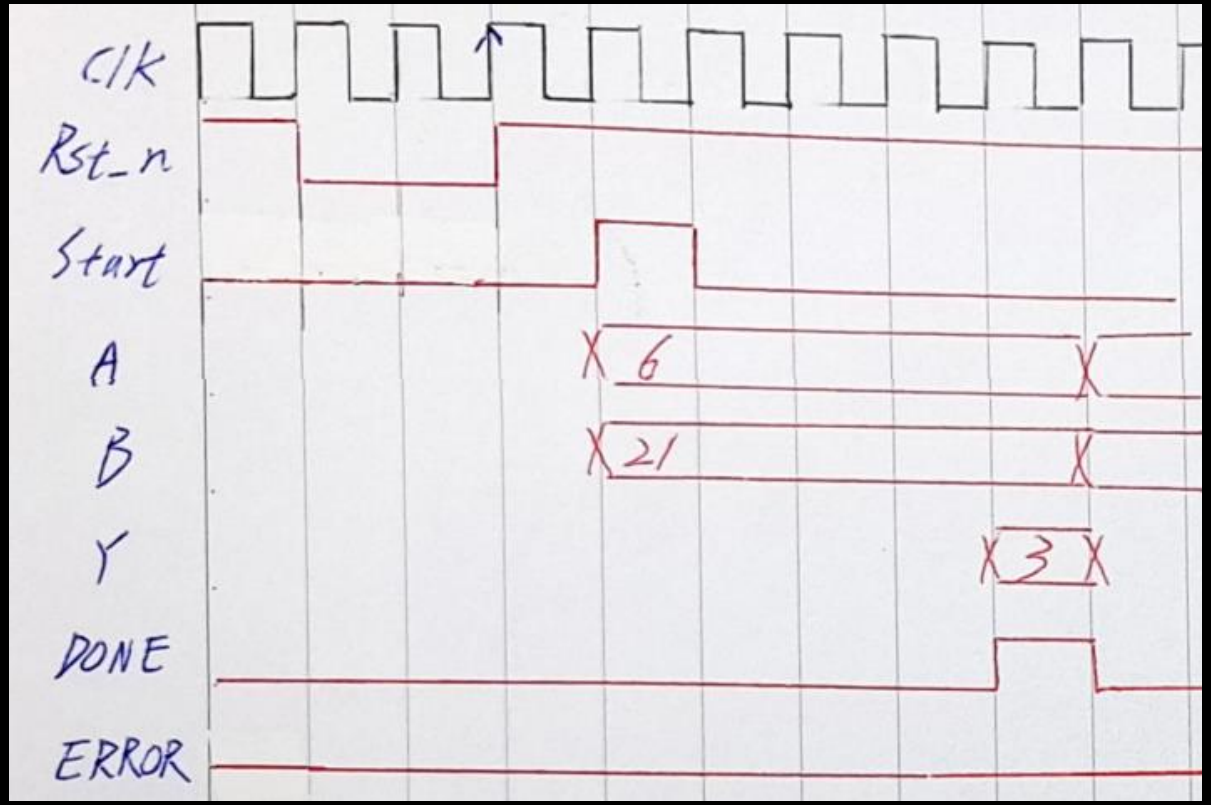
Testbench

- We will provide testbench file for this lab.You don’t need to write it on your own.
- However, you can add your own cases on top of our test cases.
- Testbench file contains 5 basic cases. You will get 60% if you passed all 5 cases.

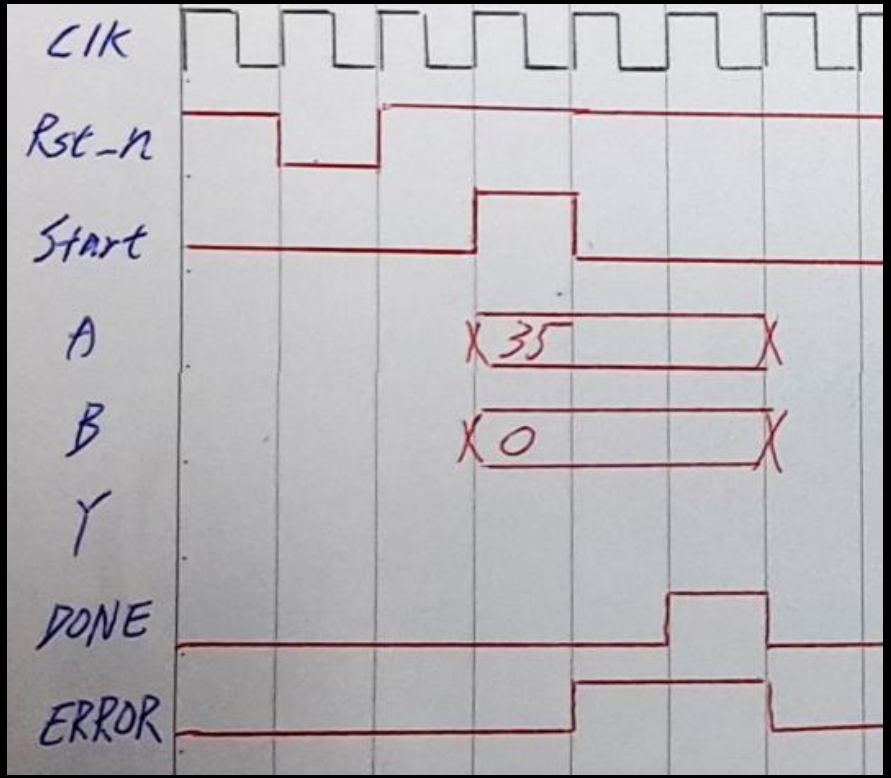
Outline

- Function Description
- Framework Introduction in Block Diagram
- Testbench template
- nWave
- Precautions and Timeline

Timing diagram



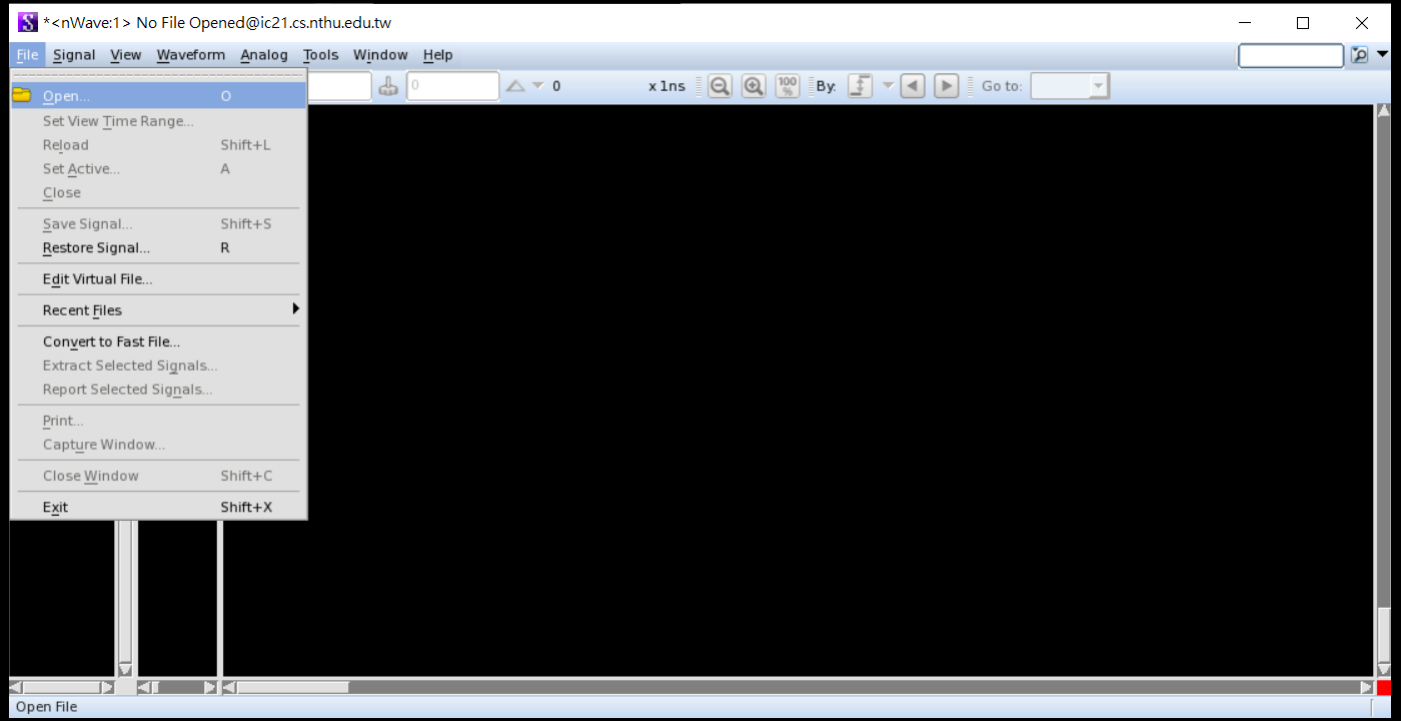
Timing diagram (error case)



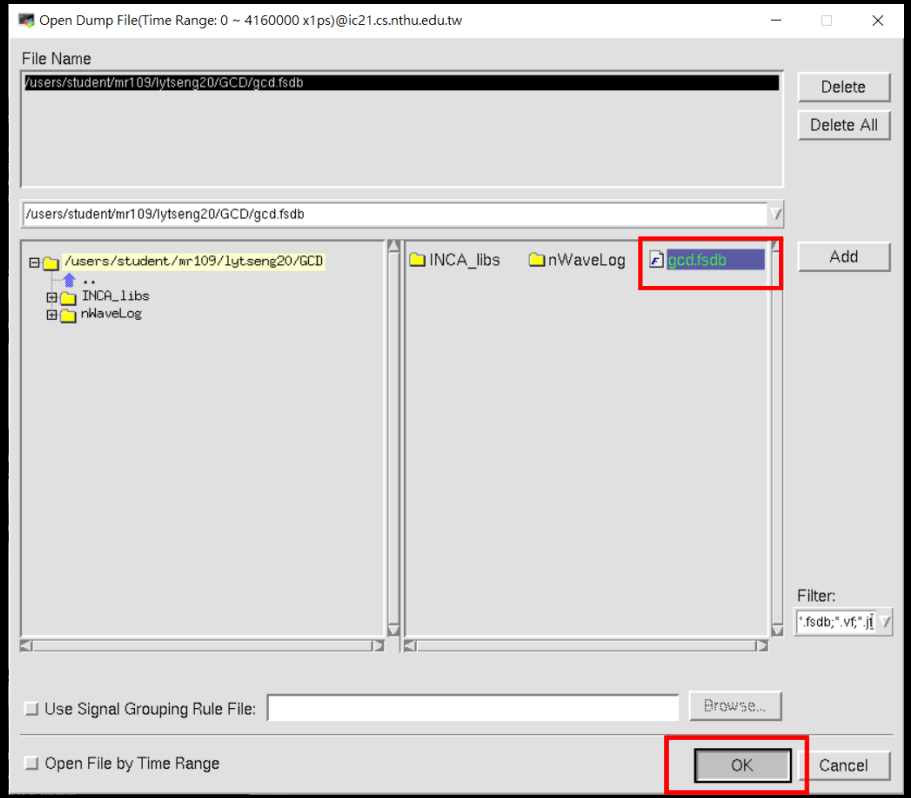
nWave

- You can use waveform viewer to debug.
- Command
  - $ncverilog tb.v design.v +access+r
  - $nWave

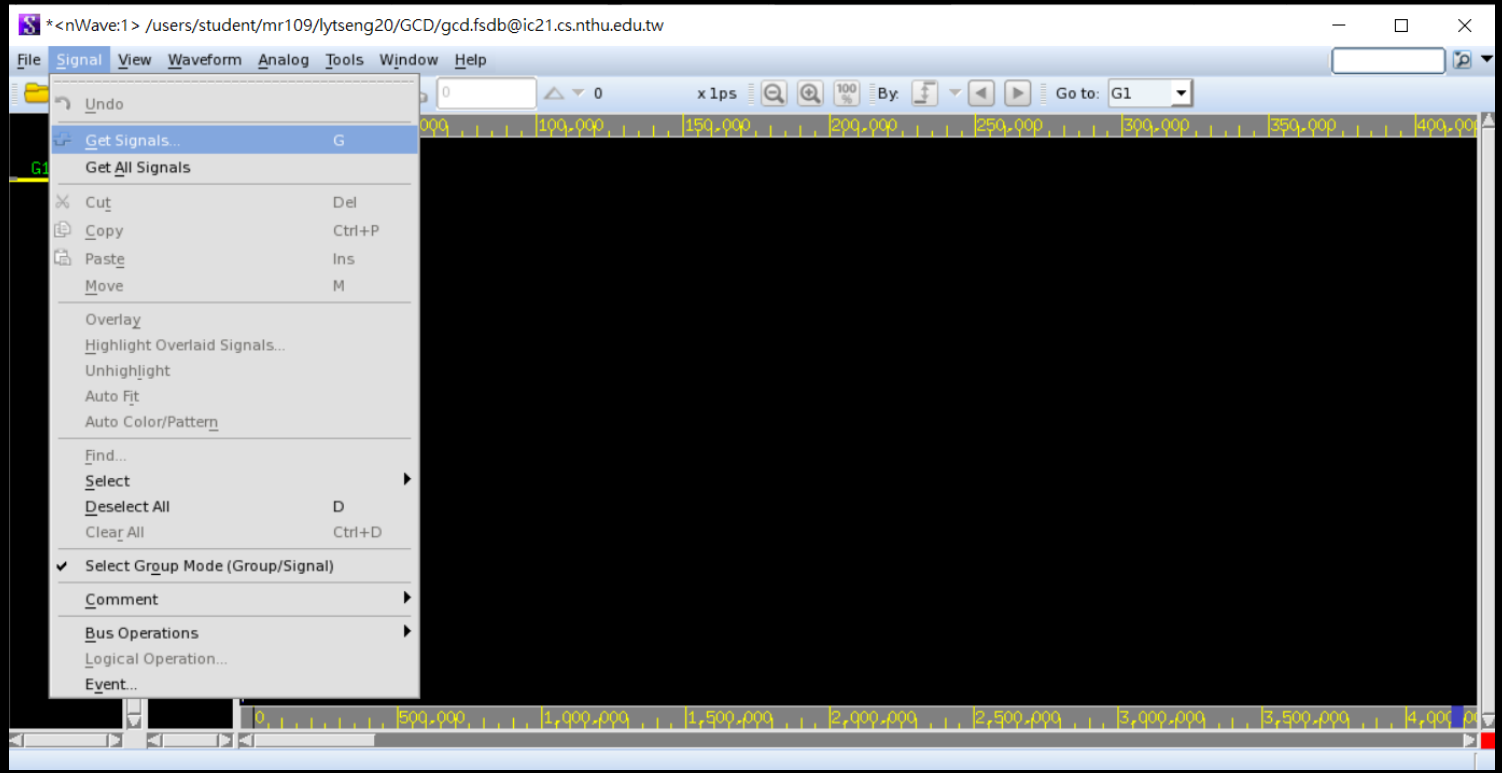
Open your waveform file



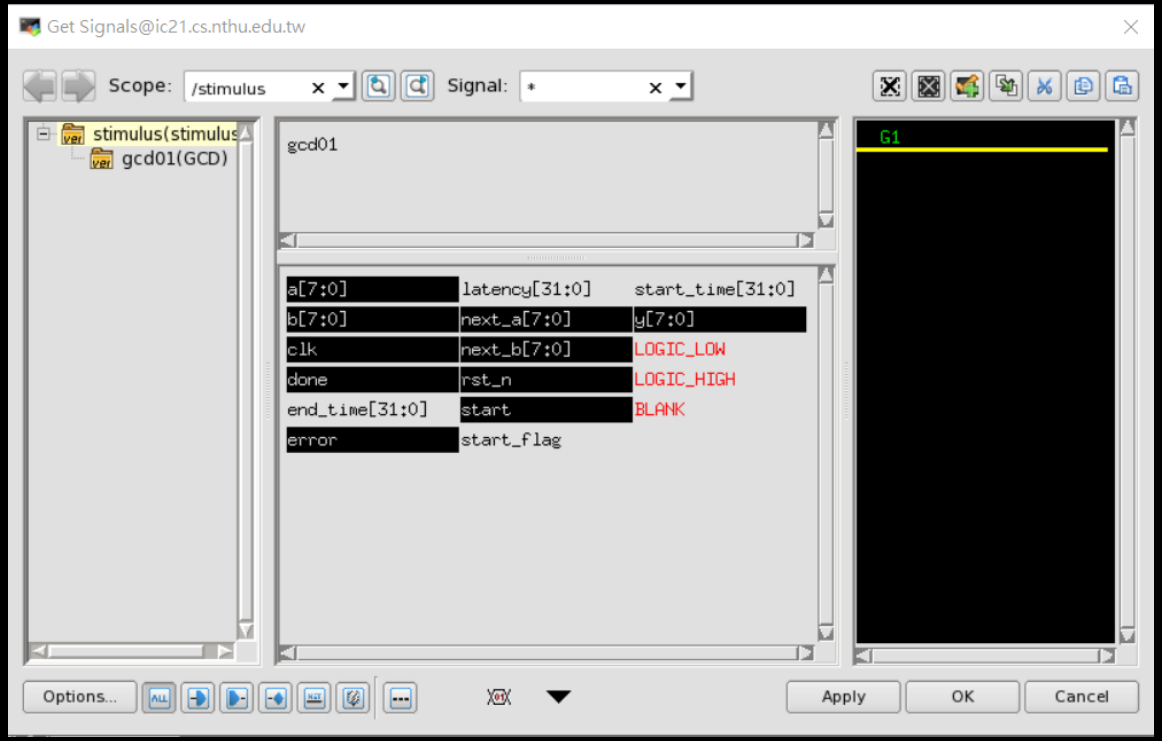
Open your waveform file



Select signals you want to check

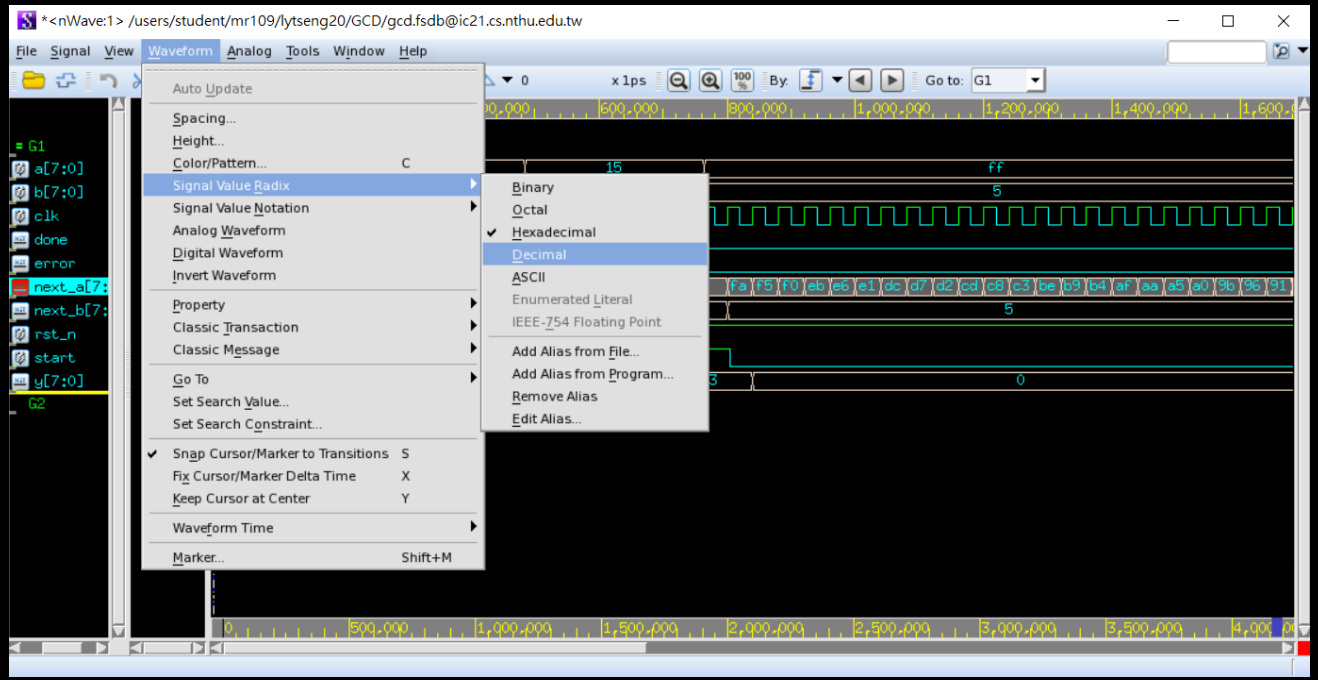


Select signals you want to check



Change signal value radix

- Example: next_a (Hex->Dec)



Change signal value radix

- Before

- After

Vim (text editor)

- Command
  - vim your_file.v
  - Press i to enter insert mode
    - You can edit your file in insert mode
    - Press Esc to leave insert mode
  - type :wq to exit vim

```
 1 module GCD (
 2   input wire CLK,
 3   input wire RST_N,
 4   input wire [7:0] A,
 5   input wire [7:0] B,
 6   input wire START,
 7   output reg [7:0] Y,
 8   output reg DONE,
 9   output reg ERROR
10 );
11
12 wire found, err;
13 reg [7:0] reg_a, reg_b, next_a, next_b;
14 reg [7:0] big_one;
15 reg error_next;
16 reg [1:0] state, state_next;
17
18 parameter [1:0] IDLE = 2'b00;
19 parameter [1:0] CALC = 2'b01;
20 parameter [1:0] FINISH = 2'b10;
21
22
23 endmodule
```

Outline

- Function Description
- Framework Introduction in Block Diagram
- Testbench template
- nWave
- Precautions and Timeline

Grading

- 60%: basic test cases (5)
- 40%: hidden test cases

Warning

- Be careful of handling boundary conditions and make sure your testbench check for that
- For simplicity, use only subtraction for GCD calculation
- Use waveform viewer nWave to debug your timing issue
- Plagiarism is forbidden and gets you 0 point
  - Also punished by NTHUCS(drop out etc.)
- Discussion is pleasant, but no coding detail.

Submission rule

- Lab 2 code submission due date & time: 2021/05/27,23:59pm
- Please submit your Verilog codes to ILMS Lab2_YourStudentID_Codes.v

- *Important*: If you want to email us, please sent it to this email address “bibi1483070@gmail.com”