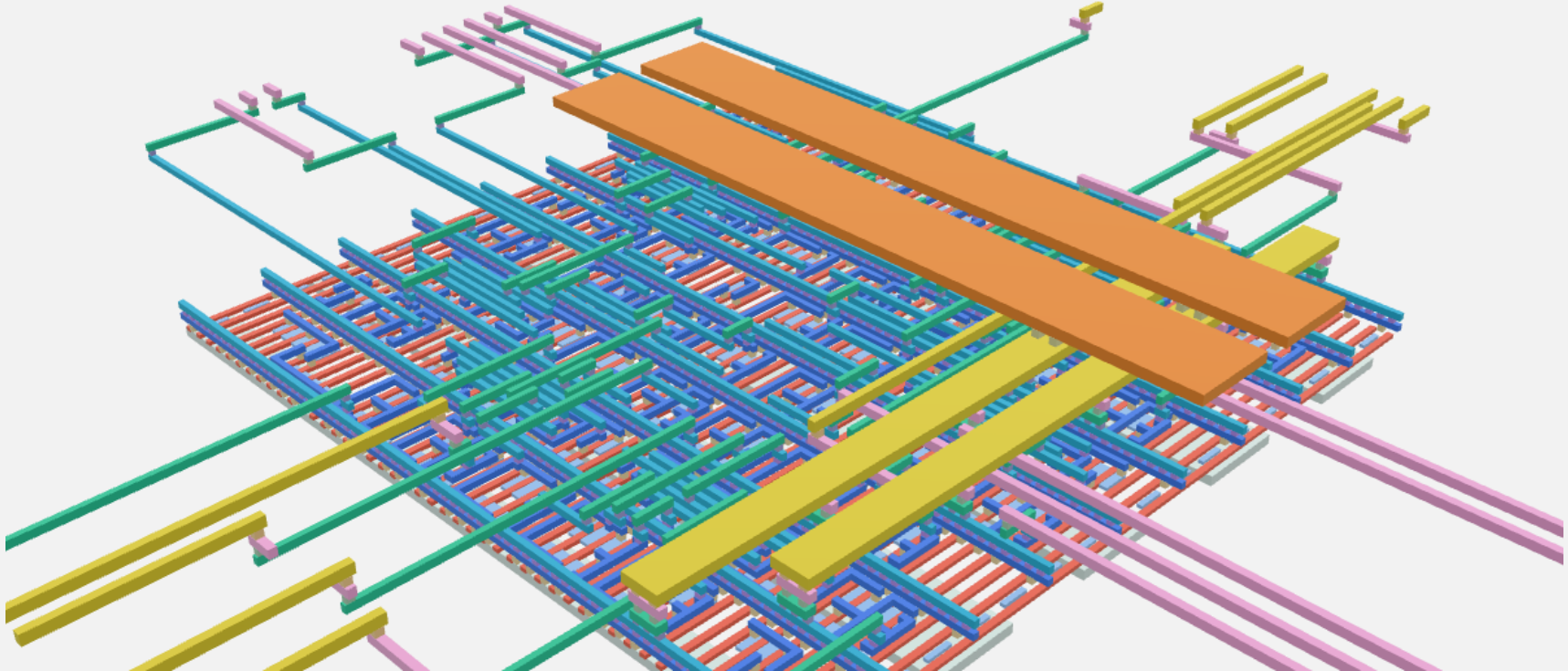
4.1
CSE140 – Digital Design

2026 Summer Session 1

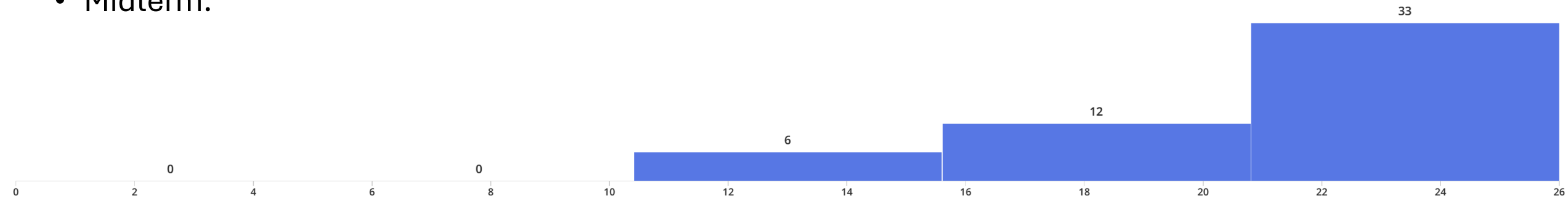
Abarajithan G



Daily reminder:

- Open FeedbackFruits
  Canvas → CSE140 → Assignments → Slides 4.1

- Attendance is mandatory
- Midterm:



Minimum	Median	Maximum	Mean	Std Dev
12.0	23.0	26.0	21.41	4.02

51 Students

Previous Feedback

- I thought that state machines were pretty fun
- I liked the live coding segment. Was nice to follow along.
- I got a bit confused on the live coding portion of it due to the bugs going on.
- the explanation for clocks and how they tie into a circuit was a bit confusing.
- I still feel a little lost on the hold time.

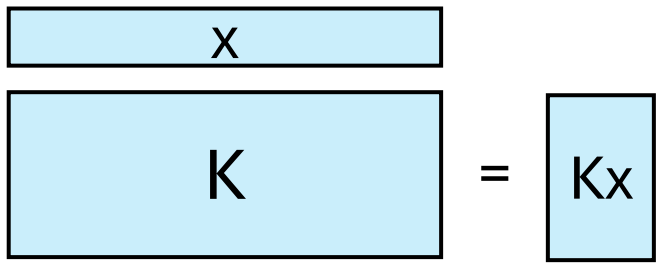
To be covered in 2 weeks

1. FIR Filter
2. Read/valid, Parallel to Serial Converter
3. Down Counter, Nested Counters
4. UART RX, UART TX
5. CPU

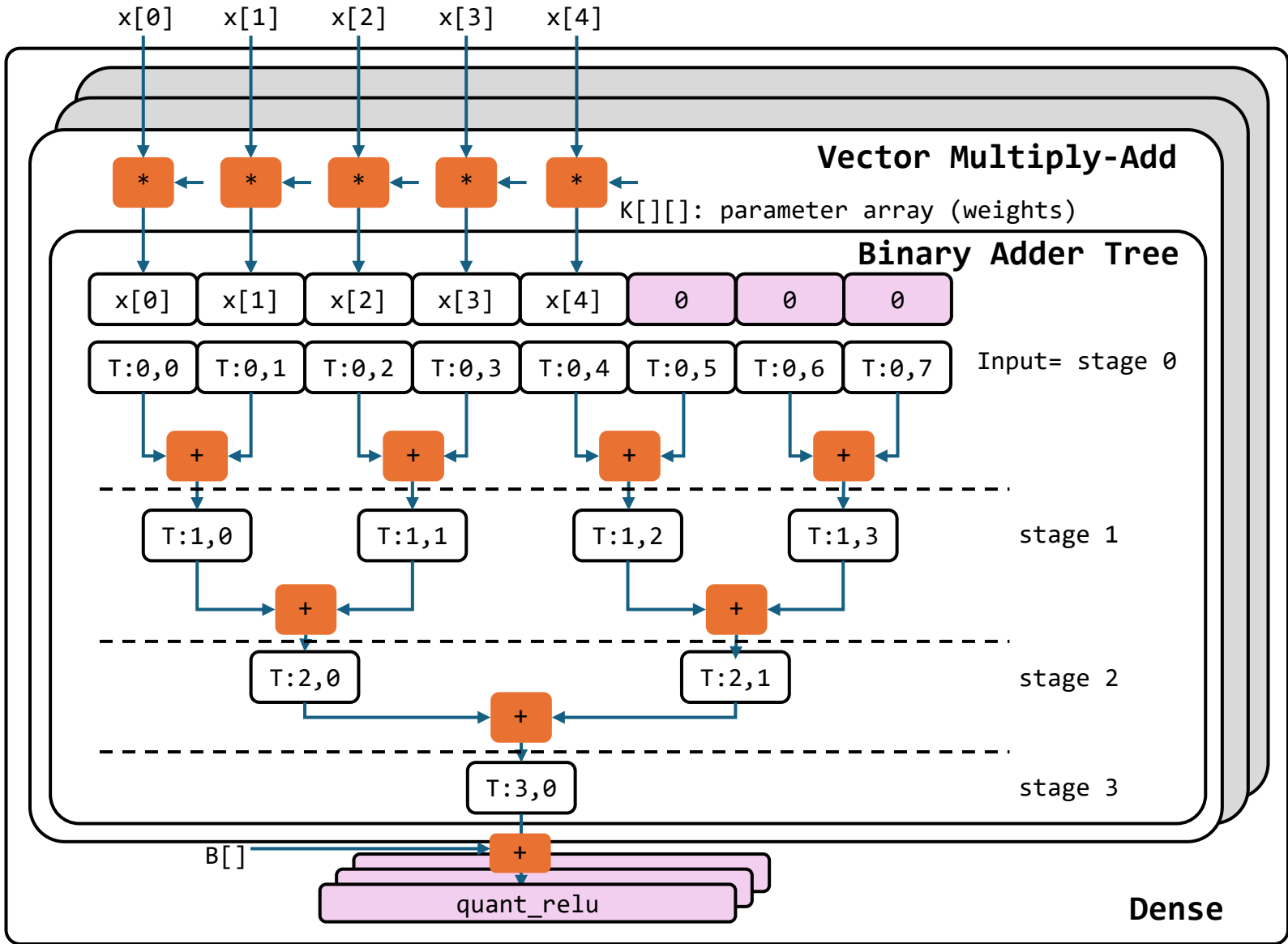
Anything you want this course to cover?

Assignment 3

- Dense layer
- y = ReLU(Kx + B)

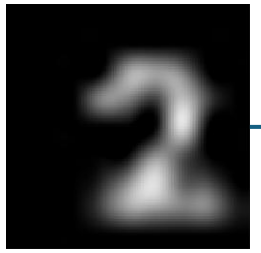






Assignment 4

Webcam input



Your computer



```
MINGW64:/d/digital-design
prediction=0  scores=[-2, -8, -3, -3,
prediction=2  scores=[-5, -8, 0, -5,
prediction=2  scores=[-6, -8, 1, -4,
prediction=2  scores=[-8, -4, 0, -2,
prediction=1  scores=[-8, -2, -2, -5,
prediction=2  scores=[-8, -3, -1, -6,
prediction=2  scores=[-8, -3, -1, -5,
prediction=2  scores=[-8, -4, 0, -4,
prediction=2  scores=[-8, -7, -1, -2,
prediction=2  scores=[-5, -8, -1, -3,
prediction=2  scores=[-4, -8, -1, -2,
prediction=2  scores=[-5, -8, 0, -3,
prediction=2  scores=[-6, -8, -3, -3,
prediction=3  scores=[-8, -8, -6, -5,
prediction=3  scores=[-7, -8, -6, -4,
```

Prediction Output

sys_nn (A4)

UART RX

axis_dense_relu [0]

dense_qrelu [0] (A3)

vector_mac

Multipliers

reduction_tree_add

qrelu (A2)

axis_dense_relu [1]

dense_qrelu [1] (A3)

vector_mac

Multipliers

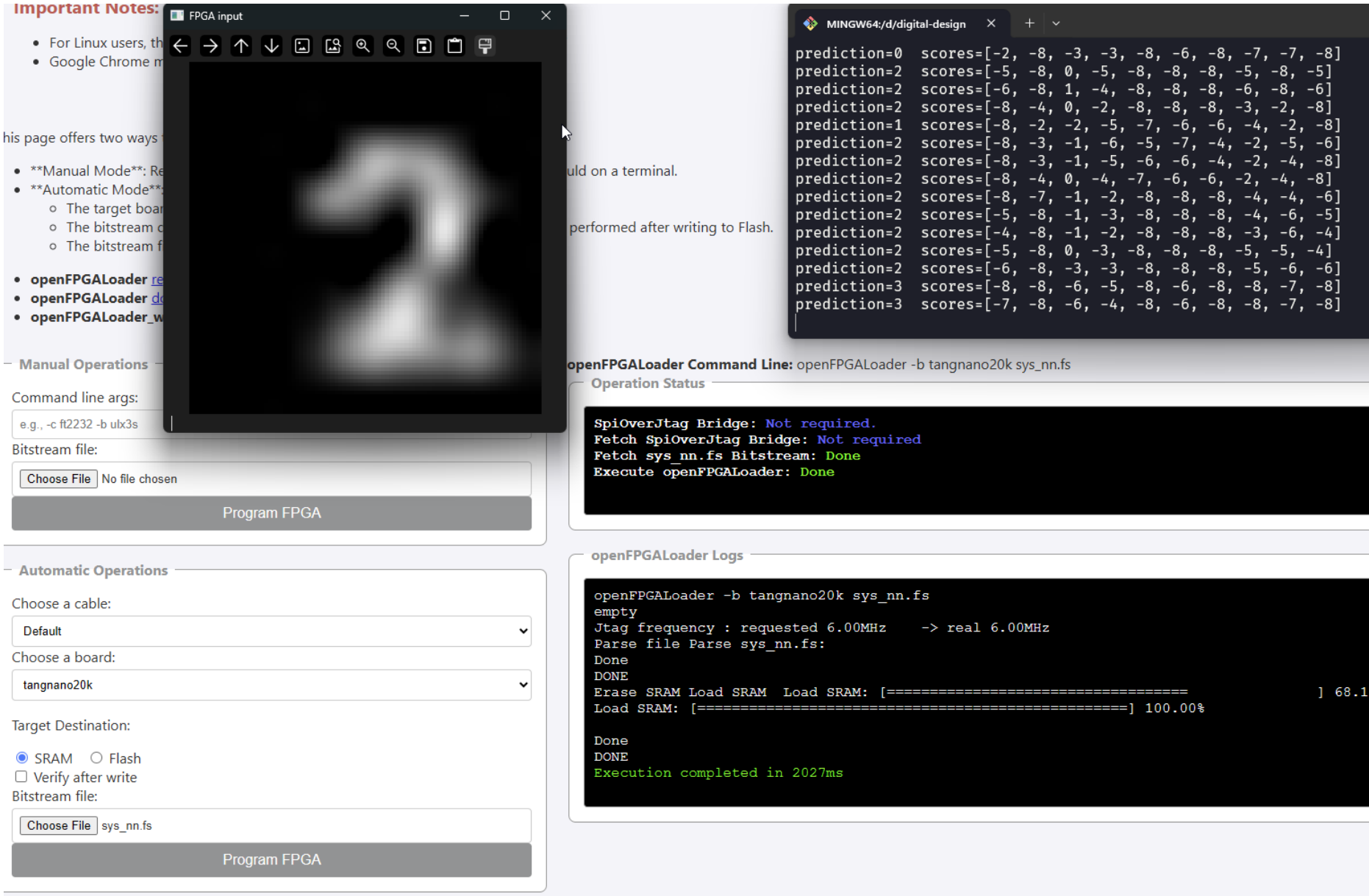
reduction_tree_add

qrelu (A2)

UART TX

NN Accelerator on FPGA

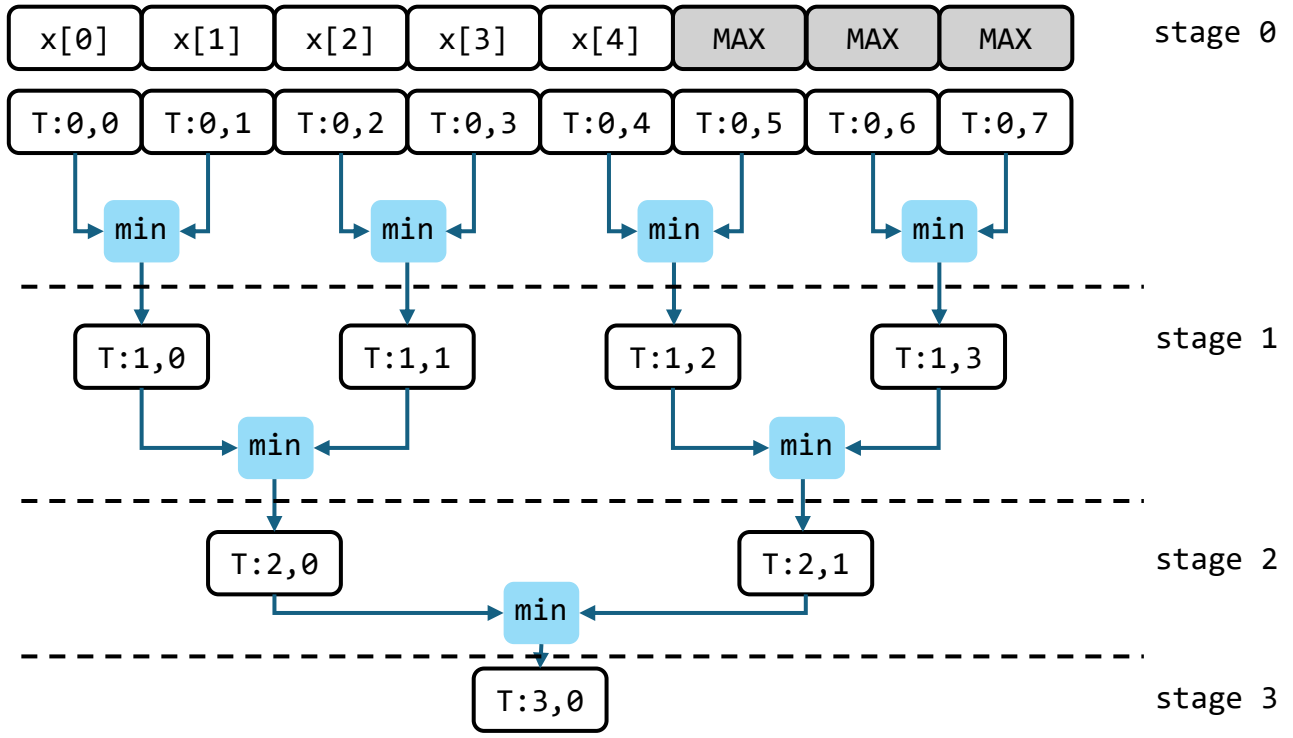
- NN is bad because **fully parallel** inference
- 54 MHz
- One frame per clock
- 54 million frames/s
- 30 fps camera
- We can put a bigger model by reusing resources.



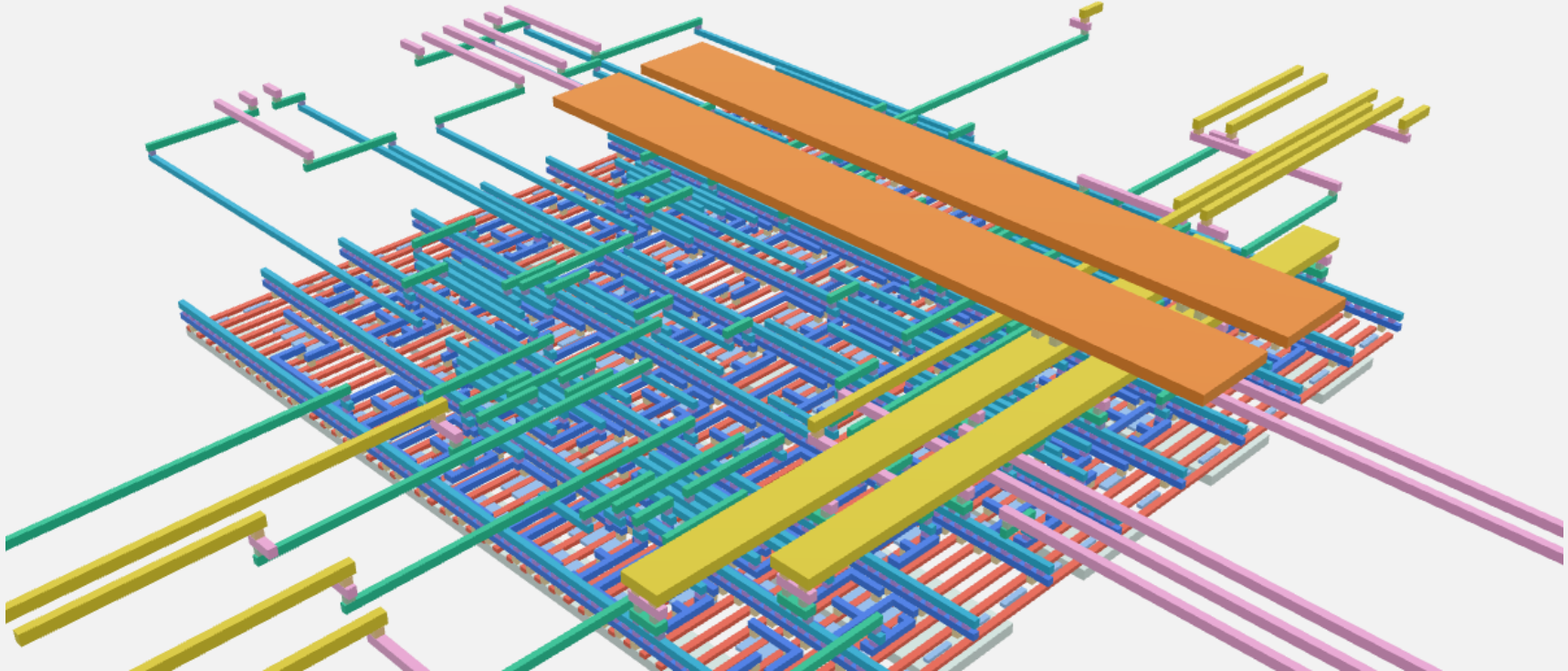
Optional Optimization – Bonus Points

- Must email me before Thursday 11:59pm
- Top 6 designs will be chosen for presentation
- Present in front of the class on Friday 31st
- Questions on code & design choices
- Must do without AI generation – risk "F"
- Bonus points: email + top 6 present + grading
- **Possible optimizations:**
  - Reuse multipliers & adders
  - Initialize acc with bias
  - FIR – share resources for repeated coef.
  - Anything else
- More info on Syllabus

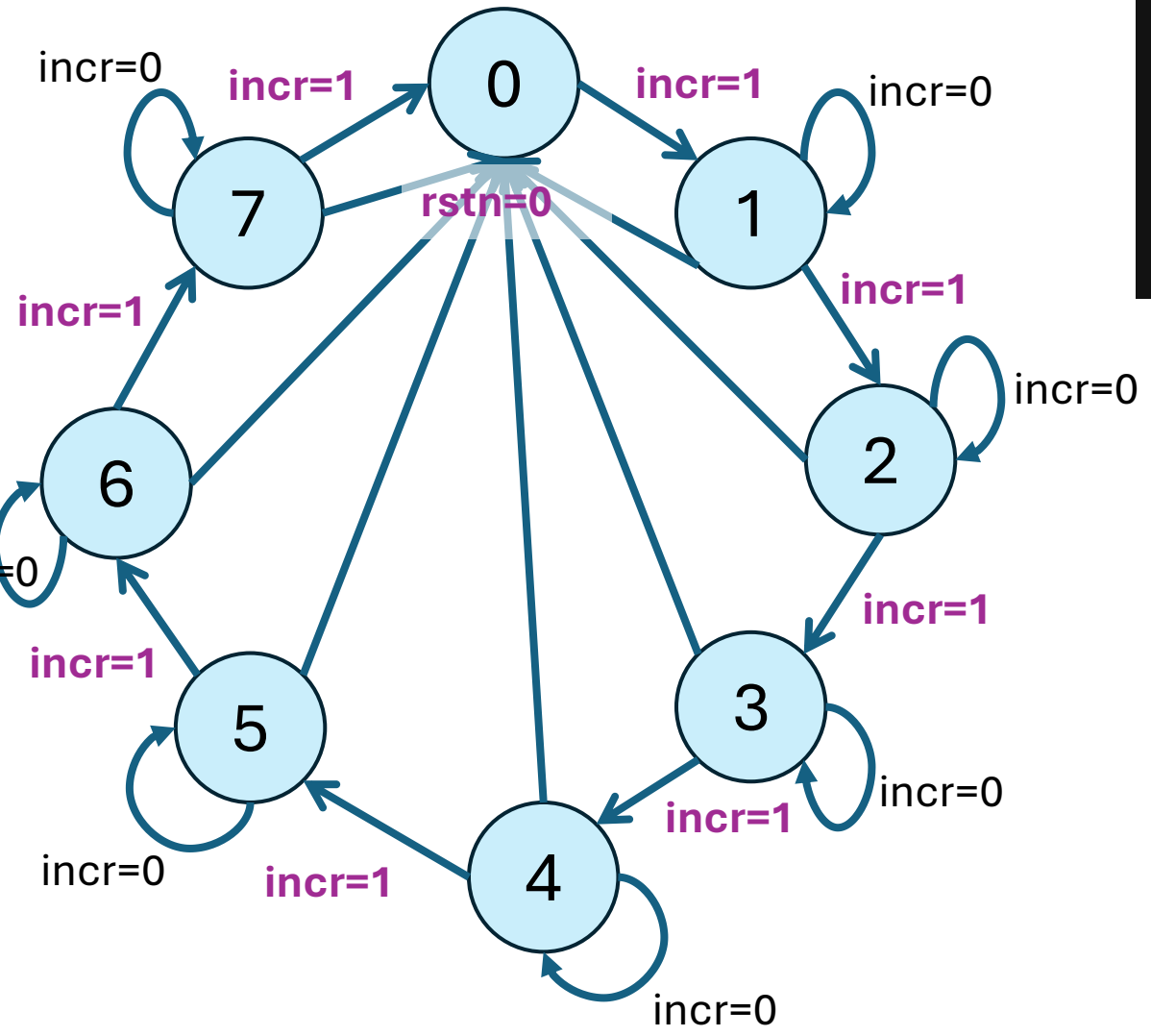
up to +5%	FIR Filter Resource Optimization
up to +10%	Neural Network Resource Optimization



Finite State Machines



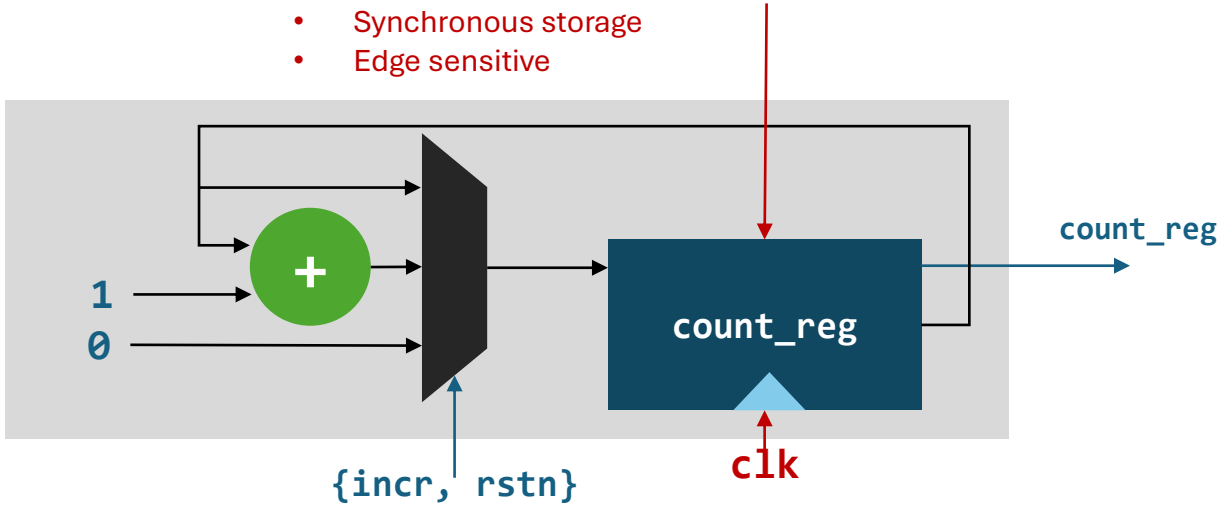
Counter




```
module up_counter #(parameter WIDTH = 3)(
  input  logic clk, rstn, incr,
  output logic [WIDTH-1:0] count
);     Flip flop sensitive to clk's positive edge, with sync active low reset
  always_ff @(posedge clk) begin
    if      (!rstn) count <= 0;
    else if (incr)  count <= count + WIDTH'(1'b1);
  end                                  non-blocking assignment
endmodule
```

Register,

- Made of flip flops
- Synchronous storage
- Edge sensitive



- Moore machine, output only depends on state

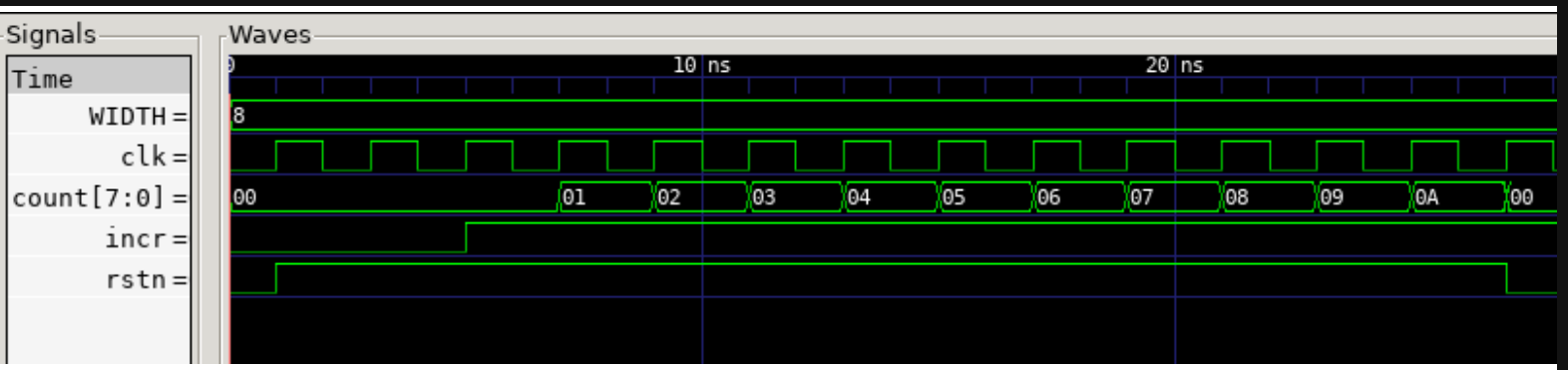
Timing Concepts

- Overall idea: signal should go from register to register within one clock period (T)
- **Tpd** – max delay of combinational circuit (prop delay, comb)
- **Tpcq** – max delay of register's output (prop delay, clock to q)
- **Setup time constraint:** Slowest signal should reach the next FF before setup time
  - T >= Tpcq + Tpd + Tsetup

- **Tcd** – min delay of combinational circuit (cont. delay, comb)
- **Tccq** – min delay of register's output (cont. delay, clock to q)
- **Hold time constraint:** Fastest signal should reach the next FF after the hold time
  - tccq + tcd >= Thold

Counter

```
module up_counter #(parameter WIDTH = 3)(
  input  logic clk, rstn, incr,
  output logic [WIDTH-1:0] count
);
  always_ff @(posedge clk or negedge rstn) begin
    if      (!rstn) count <= 0;
    else if (incr)  count <= count + WIDTH'(1'b1);
  end
endmodule
```




```
`timescale 1ns/1ps

module tb_up_counter;
  localparam WIDTH = 8;
  logic clk=0, rstn=0, incr;
  logic [WIDTH-1:0] count;

  initial forever #1ns clk = ~clk;
  task automatic posedge_clk(int n = 1);
    repeat (n) @(posedge clk); #1ps;
  endtask

  up_counter #(.WIDTH(WIDTH)) dut (.*);

  initial begin
    $dumpfile(`FST_PATH); $dumpvars;

    posedge_clk;    rstn = 1;
    posedge_clk(2); incr = 1;
    posedge_clk(10);
    posedge_clk;    rstn = 0; incr = 1;
    posedge_clk(4); incr = 0;
    posedge_clk(4); rstn = 1;

    $finish();
  end
endmodule
```

Activity – Draw the waveform

Find the value of count (unsigned decimal) in
- 50 ns
- 60 ns
- 70 ns

```
`timescale 1ns/1ps

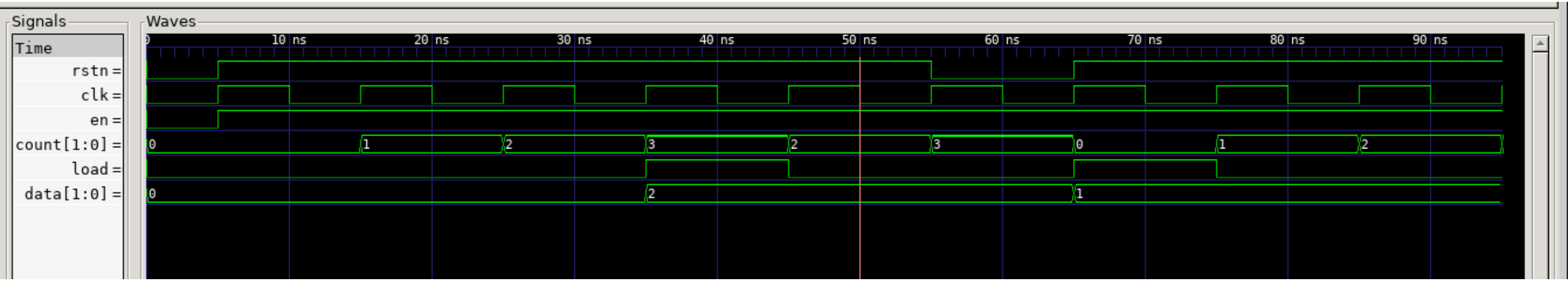
module counter (
  input  logic clk, rstn, load, en,
  input  logic [1:0] data,
  output logic [1:0] count);


  always_ff @(posedge clk)
    if (!rstn)     count <= 2'd0;
    else if (load) count <= data;
    else if (en)   count <= count + 1'b1;
endmodule
```

```
module tb;
  logic clk=0, rstn=0, load=0, en=0;
  logic [1:0] data=0, count;
  counter dut (.*);
  initial forever #5 clk = ~clk;
  task automatic pclk(int n=1);
     repeat(n) @(posedge clk); #1ps;
  endtask

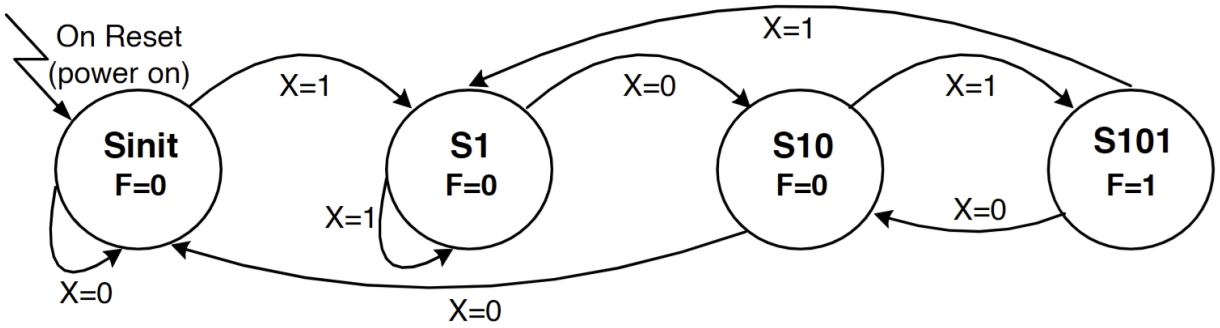
  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
    pclk;    rstn=1; en=1;                  // 5 ns
    pclk(3); load=1; data=2;                // 15,25,35
    pclk;    load=0;                        // 45
    pclk;    rstn=0;                        // 55
    pclk;    rstn=1; load=1; data=1;        // 65
    pclk;    load=0;                        // 75
    pclk(2); $finish;                       // 85,95 ns
  end
endmodule
```

Activity – Draw the waveform



Pattern Detector

"101" Sequence Detector should output F=1 when sequence 101 is found in consecutive order



State Diagram for "101" Sequence Detector

https://ee.usc.edu/~redekopp/ee209/slides/EE209Spiral1-7.pdf

```
module detector_101 (
  input  logic clk, rstn, x,
  output logic f
);
```

typedef, enum

```
typedef enum logic [1:0]
        {SINIT, S1, S10, S101} state_t;
state_t state, next;
```

```
always_ff @(posedge clk or negedge rstn)
  if (!rstn) state <= SINIT;
  else       state <= next;
```

State sequencing

State transition

```
always_comb begin
  case (state)
    SINIT: next = x ? S1   : SINIT;
    S1:    next = x ? S1   : S10;
    S10:   next = x ? S101 : SINIT;
    S101:  next = x ? S1   : S10;
    default: next = SINIT;
  endcase
end
```

```
always_ff f = (state == S101);
```

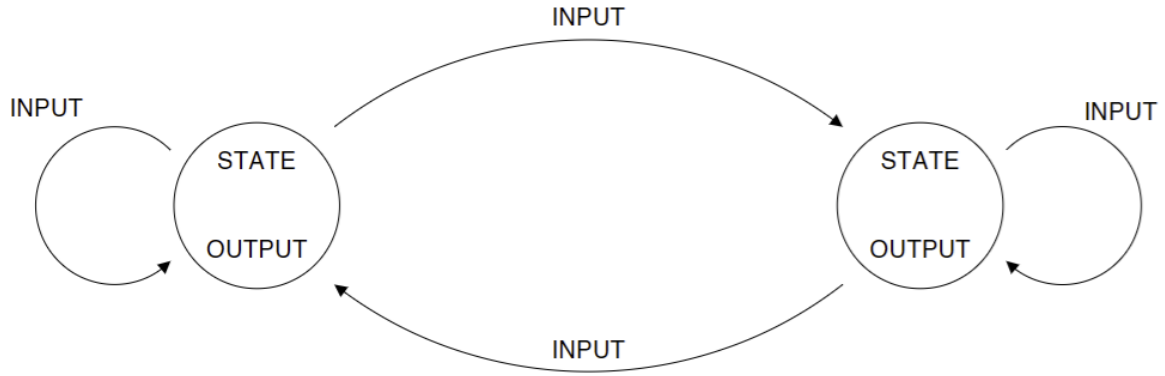
State decoding

```
endmodule
```

Mealy & Moore Machines

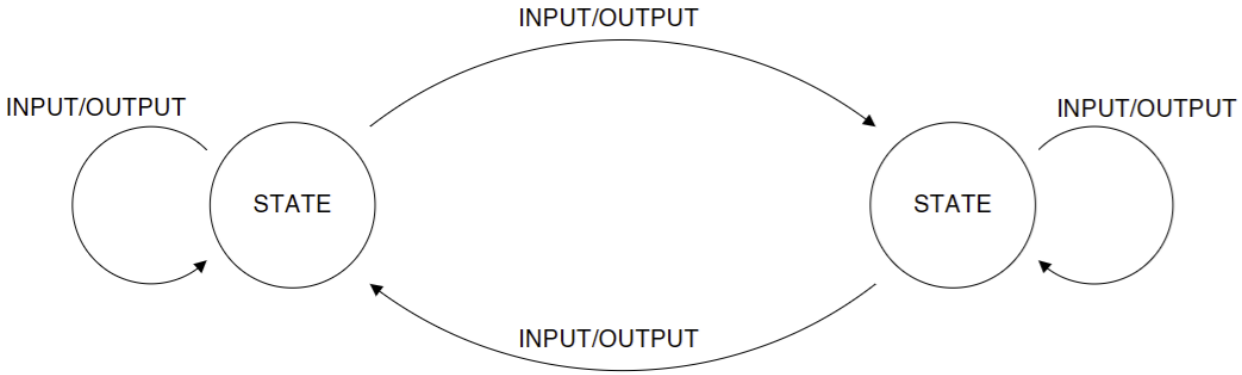
- Moore machine: out = f(state)

```
module up_counter #(parameter WIDTH = 3)(
  input  logic clk, rstn, incr,
  output logic [WIDTH-1:0] count,
  output logic less_than_3
);
  always_ff @(posedge clk or negedge rstn)
    if      (!rstn) count <= 0;
    else if (incr)  count <= count + WIDTH'(1'b1);
  always_comb less_than_3 = count < 3;
endmodule
```



- Mealy machine: out = f(state, inputs)

```
module up_counter #(parameter WIDTH = 3)(
  input  logic clk, rstn, incr,
  output logic [WIDTH-1:0] count, count_next
);
  always_ff @(posedge clk or negedge rstn)
    if      (!rstn) count <= 0;
    else if (incr)  count <= count + WIDTH'(1'b1);
  always_comb count_next = count + incr;
endmodule
```



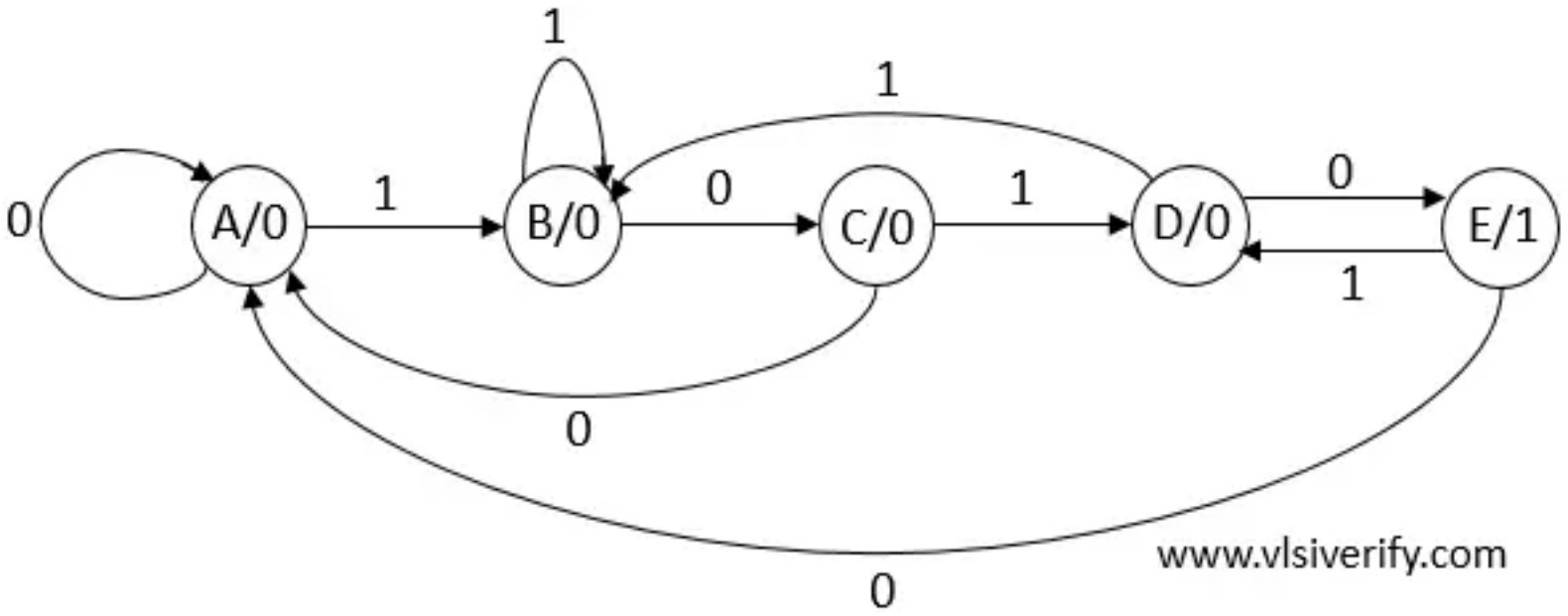
Mealey & Moore Machines

	Moore Machine	Mealy Machine
Output depends on	Current state only	Current state and current input
Output location in diagram	Inside states	On transitions
Transition label	Input	Input / output
Output timing	Changes after state changes	Can change immediately when input changes
Number of states	Often more	Often fewer
Output stability	More stable	More sensitive to input glitches
Response speed	Usually one clock later	Can respond in the same clock cycle
Typical output logic	`z = f(state)`	`z = f(state, input)`

Pattern Detector - Activity

- Design a moore machine to detect 1010

Pattern Detector



1010 Overlapping Moore Sequence Detector

FIR Filter – Audio processing circuits

Digital Signals

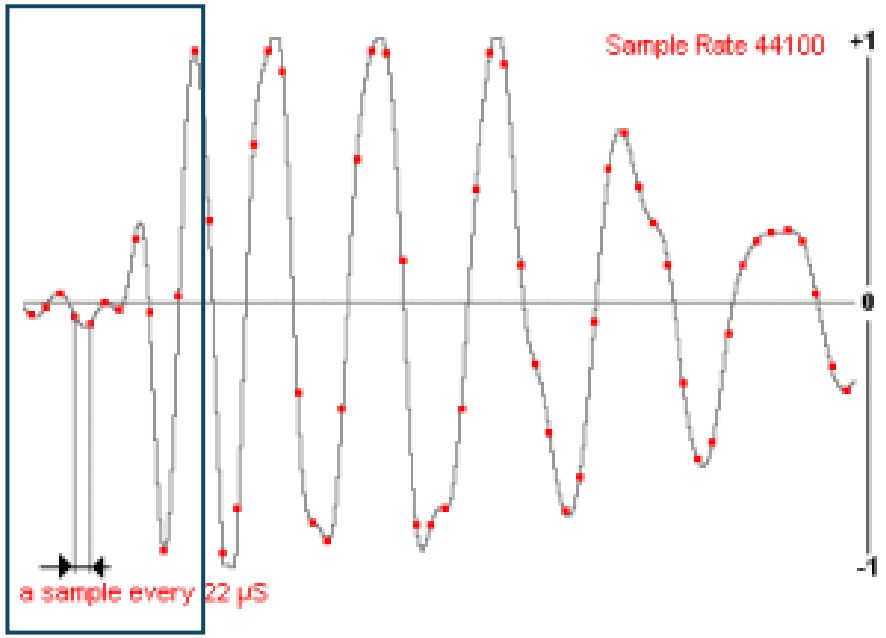
- Audio (songs you listen, my voice…) is a 1D signal
- Stored and transmitted as a series of numbers: [20, 5, 3, 8, 17,…]
- In uncompressed form, the numbers are proportional to sound amplitude & voltage of mic/speaker in known intervals.

Filtering

- We often need to extract, remove or boost different frequency ranges (bass, treble, midtones)
- Inside your mics, headphones & guitar amplifiers, this is done in custom chips, that apply various filters

FIR Filters

- Finite Impulse Response (FIR) filters are series of numbers multiplied & added with the sound signal for filtering.
- Numbers are mathematically computed – entire field of DSP




```
[-2,-1, 2,-2 ,-3, 0,-1, 7,-1,-23,1,23] – samples: x[n]
[ 0, 1, 2, 5 , 2, 1, 0]                – filter : k[n]
```

Dot product:

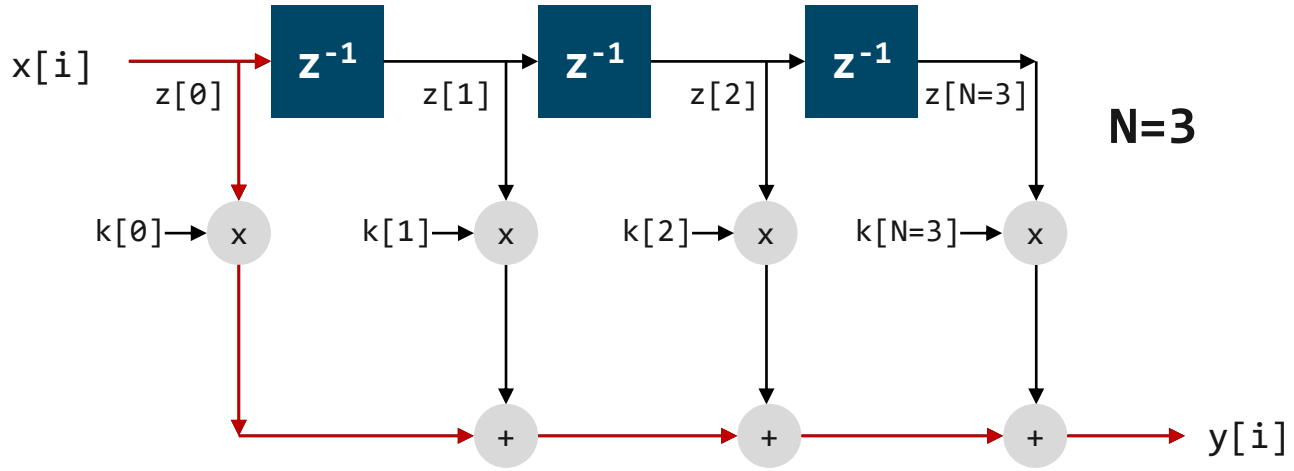
```
[ 0,-1, 4,-10,-6, 0, 0]  – multiplied: m[n]
[-13]                    – added     : y[0]
```

Shift x[n] & dot product:

```
[-2,-1, 2,-2, -3, 0,-1, 7,-1,-23,1,23]    – samples: x[n]
   [ 0, 1, 2,  5, 2, 1, 0]                – filter : k[n]
   [ 0, 2,-4,-15, 0,-1, 0]  – multiplied: m[n]
   [-18]                    – added     : y[1]
```

Repeat shift x[n] & dot product: Convolution

FIR Filter



$$y[n] = x[n]k[0] + x[n-1]k[1] + x[n-2]k[2] + \ldots + x[n-N]k[N]$$

$$y[n] = \mathbf{x[n]}k[0] + \mathbf{z^{-1}x[n]}k[1] + \mathbf{z^{-2}x[n]}k[2] + \ldots + \mathbf{z^{-N}x[n]}k[N]$$

Implementation:

- Every clock cycle, an input sample goes in, output sample comes out
- N+1 multipliers and N adders work in parallel for dot product
- Long path through adders. Fmax for N=100: 687.6 MHz

```
module fir_filter_naive #(
  parameter N = 5, W_X = 8, W_K = 4,
  parameter logic [(N+1)*W_K-1:0] K = {
    4'sd6, 4'sd5, 4'sd4, 4'sd3, 4'sd2, 4'sd1
    },//'{default:0},
  localparam W_Y = W_X + W_K + $clog2(N+1))(
    input  clk, rstn, en,
    input  logic [W_X-1:0] x,
    output logic [W_Y-1:0] y
  );
  logic [N:0][W_K-1:0] k_arr;
  always_comb k_arr = K;

  genvar n;
  localparam W_M = W_X + W_K;
  logic [N  :0][W_M-1:0] m;
  logic [N  :0][W_Y-1:0] a;
  logic [N  :0][W_X-1:0] z;

  always_ff @(posedge clk or negedge rstn)
    if (!rstn)   z[N:1] <= '0;
    else if (en) z[N:1] <= z[N-1:0];

  always_comb begin
    z[0] = x;
    y = 0;
    for (int n=0; n < N+1; n=n+1)
      y = $signed(y)+$signed(k_arr[n])*$signed(z[n]);
  end
endmodule
```

Flat array: K*W

Constant coefficients, baked into the circuit

Flat array [K*W] can be assigned to packed array [K][W]

Delay registers to hold older x values

MAC Tree - combinational

Daily Feedback

- One thing I liked about today’s class:
- One thing that could be improved:
- One place where I felt lost (if any):