ENGR 210 / CSCI B441
“Digital Design”

UART I

Andrew Lukefahr

Course Website

fangs-bootcamp.github.io

Write that down!

Announcements

- Saturating Counter: You should be done
- Elevator Controller: Should be mostly done
- UART: Should be up

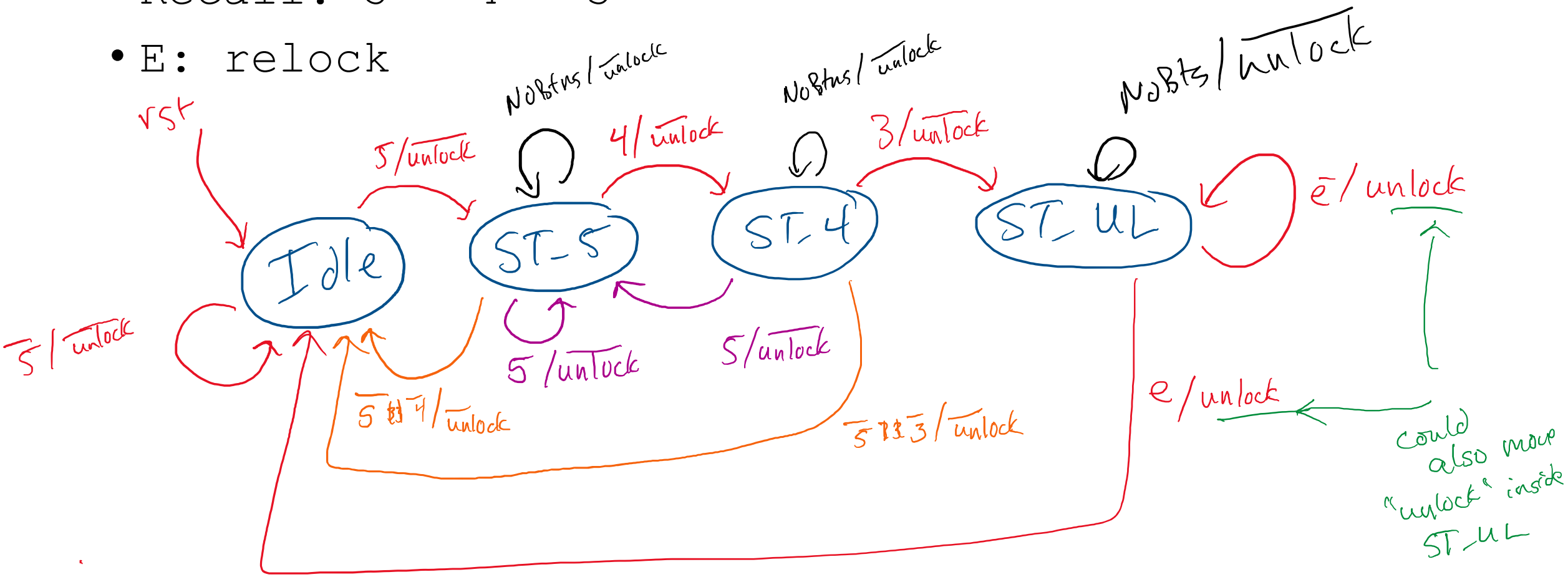
Always specify defaults for `always_comb`!

BLOCKING (=) FOR `always_comb`

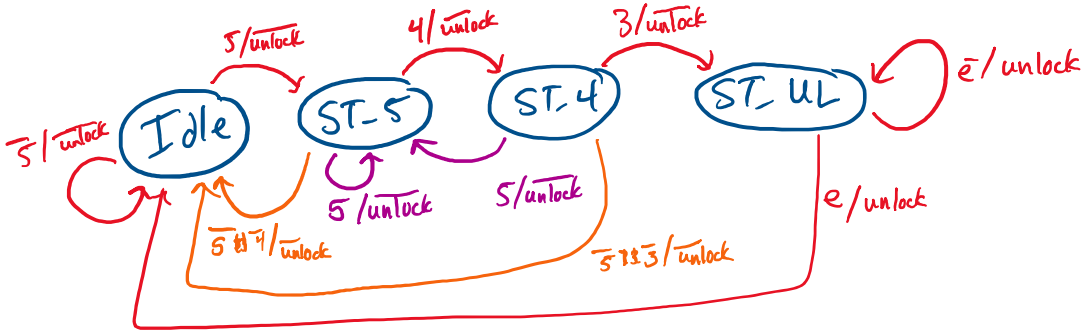
NON-BLOCKING (<=) for `always_ff`

Lock State Machine

- Recall: 5 – 4 – 3
- E: relock

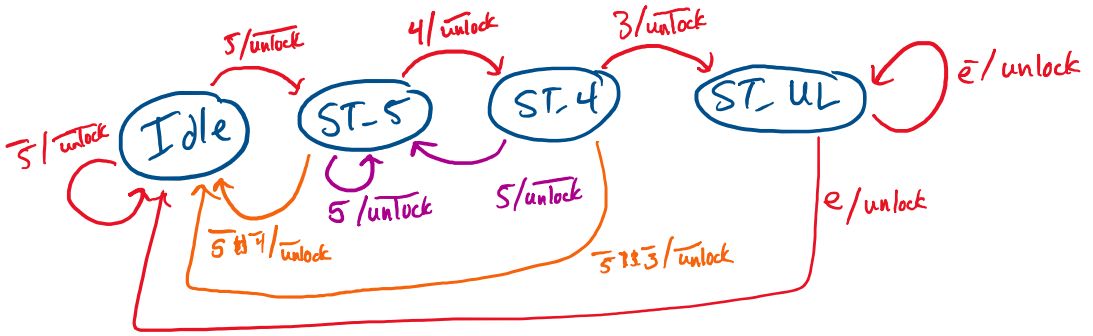


State Machine in Verilog




```
module Lock(
    input clk, rst,
    input [9:0] num,
    input e, //relock
    output unlock
);

enum {ST_IDLE, ST_5, ST_4, ST_UL } state, next_state;

//seq logic
always_ff @(posedge clk) begin
       if (rst) state <= ST_IDLE;
       else     state <= next_state;
end
```

State Machine in Verilog



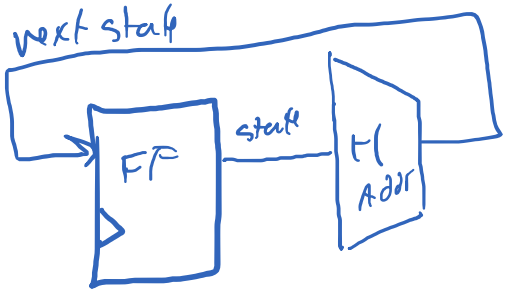

```
case (state)
  ST_IDLE:
    if (num[5])
      next_state = ST_5;
  ST_5: begin
    if (num[4])
      next_state = ST_4;
    else if (num[5])
      next_state = ST_5;
    else if ( (|num) | e ) //other btns
      next_state = ST_IDLE;
  end
 ST_4: begin
    if (num[3])
      next_state = ST_UL;
    else if (num[5])
      next_state = ST_5;
    else if ( (|num) | e) // other btns
      next_state = ST_IDLE;
  end
  ST_UL: begin
    unlock = 1'h1;
    if (e)
      next_state = ST_IDLE;
  end
endcase
```

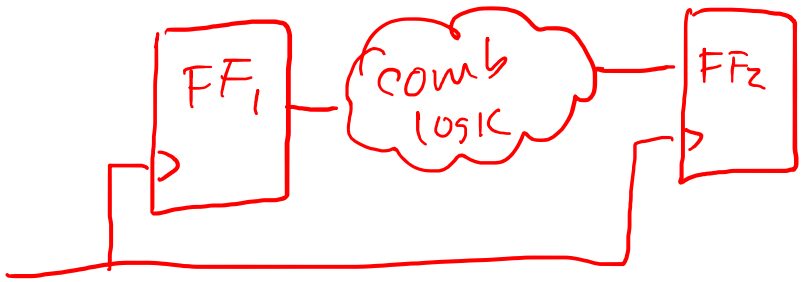
Setup and Hold Time

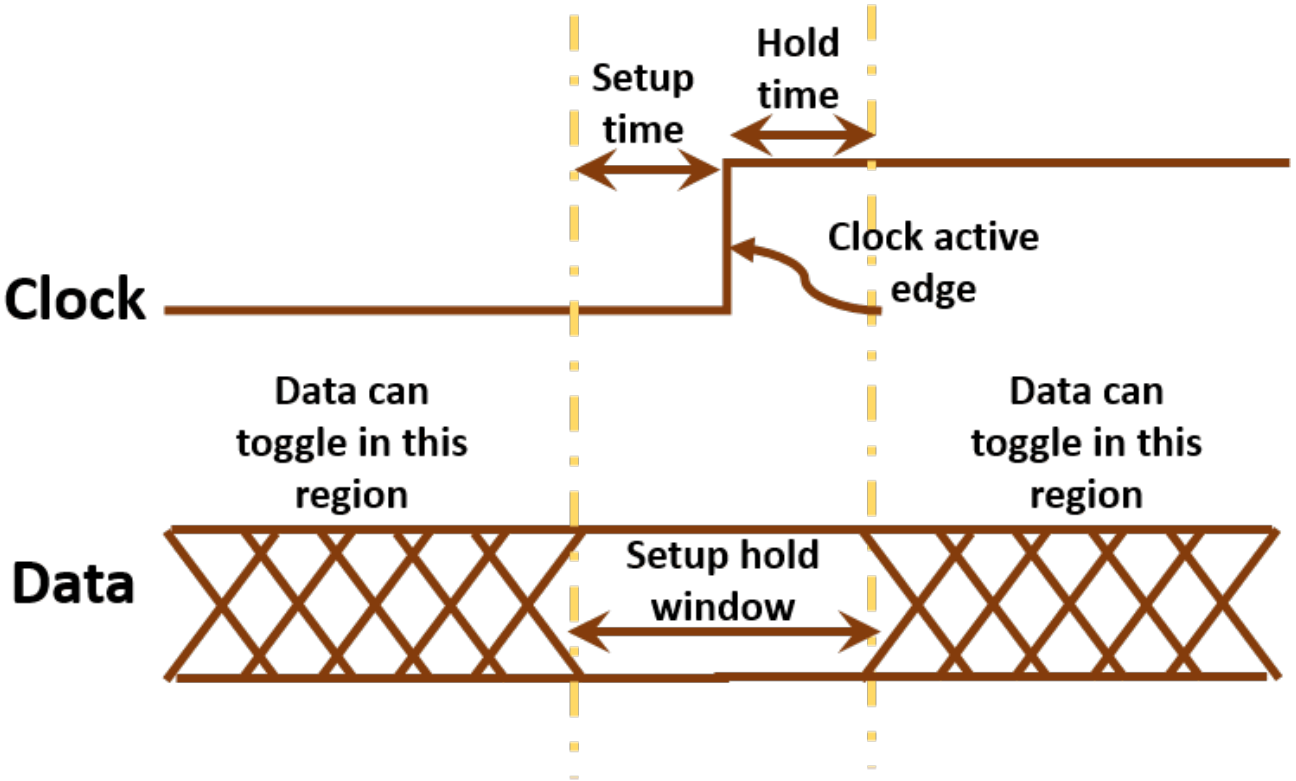
- **Setup Time**: minimum time the inputs to a flip-flop **must** be stable **before** the clock edge

- **Hold Time:** minimum time the inputs to a flip-flop **must** be stable **after** the clock edge

Setup/Hold Time







Slack

- Extra time between combinational propagation delay and setup time
- Time between stable input to Flip-Flop and next clock edge

- Vivado:
  - WNS: Worst-case Negative Slack

- If this number is <0, your circuit will (probably) not work

`for(;;)` Loops in Verilog

- Testbenches – just like C/C++

- Synthesizable modules - more like `#define`

`for(;;)` Loops in Verilog

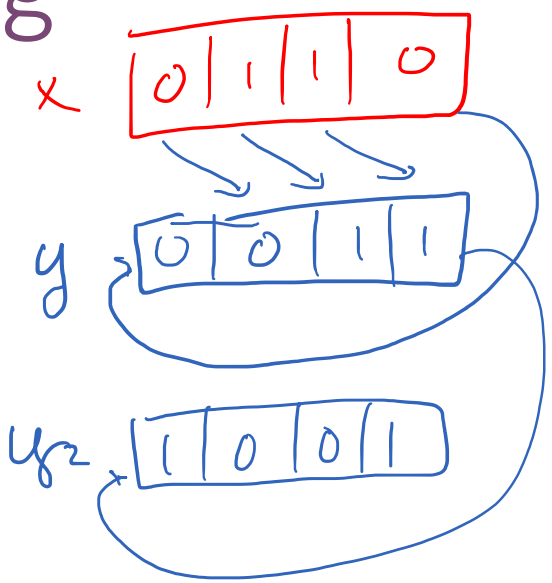
- Testbenches – just like C/C++
- Synthesizable modules - more like `#define`
- `for(;;)` loop is expanded at Compile (Synthesis) Time

Correct `for(;;)` Loops in Verilog

```
module MysteryFunction(
   input[3:0] a,
   input[1:0] right,
   output[3:0] y
);

//correct for loop
integer x; // OR genvar x
for (x = 0; x < 4; x++) begin
     assign y[x] = a[x+right];
end

endmodule
```

Correct `for(;;)` Loops in Verilog

```
module RightBarrelShift(
   input[3:0] a,
   input[1:0] right,
   output[3:0] y
);

//correct for loop
integer x; // OR genvar x
for (x = 0; x < 4; x++) begin
     assign y[x] = a[x+right];
end

endmodule
```



assign y[0] = a[0+right]
assign y[1] = a[1+right]
...

Correct `for(;;)` Loops in Verilog

This Code:

```
integer x; //or genvar x
for (x = 0; x < 4; x++) begin
     assign y[x] = a[x+right];
end
```

Expands to:

```
assign y[0] = a[0 + right];
assign y[1] = a[1 + right];
assign y[2] = a[2 + right];
assign y[3] = a[3 + right];
```

Incorrect `for(;;)` Loops in Verilog

```
integer i;
wire carry;

for(i = 0; i < 4; i++) begin
        if (i == 0) begin
                assign sum[i] = a[i] ^ b[i] ^ c_in;
                assign carry = a[i] & b[i] | a[i] & c_in | b[i] & c_in;
        end else begin
                assign sum[i] = a[i] ^ b[i] ^ carry;
                assign carry = a[i] & b[i] | a[i] & carry | b[i] & carry;
        end
end
assign c_out = carry;
```

What is wrong here?

Incorrect `for(;;)` Loops in Verilog

```
integer i;
wire carry;

for(i = 0; i < 4; i++) begin
        if (i == 0) begin
                assign sum[i] = a[i] ^ b[i] ^ c_in;
                assign carry = a[i] & b[i] | a[i] & c_in | b[i] & c_in;
        end else begin
                assign sum[i] = a[i] ^ b[i] ^ carry;
                assign carry = a[i] & b[i] | a[i] & carry | b[i] & carry;
        end
end
assign c_out = carry;
```

What is wrong here?

Incorrect `for(;;)` Loops in Verilog

```
integer i;
wire carry;
assign sum[0] = a[0] ^ b[0] ^ c_in;
assign carry = a[0] & b[0] | a[0] & c_in | b[0] & c_in;
assign sum[1] = a[1] ^ b[1] ^ carry;
assign carry = a[1] & b[1] | a[1] & carry | b[1] & carry;
assign sum[2] = a[2] ^ b[2] ^ carry;
assign carry = a[2] & b[2] | a[2] & carry | b[2] & carry;
assign sum[3] = a[3] ^ b[3] ^ carry;
assign carry = a[3] & b[3] | a[3] & carry | b[3] & carry;
assign c_out = carry;
```

Correcting `for(;;)` Loops in Verilog

```
integer i;
wire carry;

for(i = 0; i < 4; i++) begin
        if (i == 0) begin
                assign sum[i] = a[i] ^ b[i] ^ c_in;
                assign carry = a[i] & b[i] | a[i] & c_in | b[i] & c_in;
        end else begin
                assign sum[i] = a[i] ^ b[i] ^ carry;
                assign carry = a[i] & b[i] | a[i] & carry | b[i] & carry;
        end
end
assign c_out = carry;
```

How do we fix this?

Correcting `for(;;)` Loops in Verilog

```
integer i;
wire [3:0] carry;

for(i = 0; i < 4; i++) begin
        if (i == 0) begin
                assign sum[i] = a[i] ^ b[i] ^ c_in;
                assign carry[i] = a[i] & b[i] | a[i] & c_in | b[i] & c_in;
        end else begin
                assign sum[i] = a[i] ^ b[i] ^ carry[i-1];
                assign carry[i] = a[i] & b[i] | a[i] & carry[i-1] | b[i] & carry[i-1];
        end
end
assign c_out = carry[3];
```

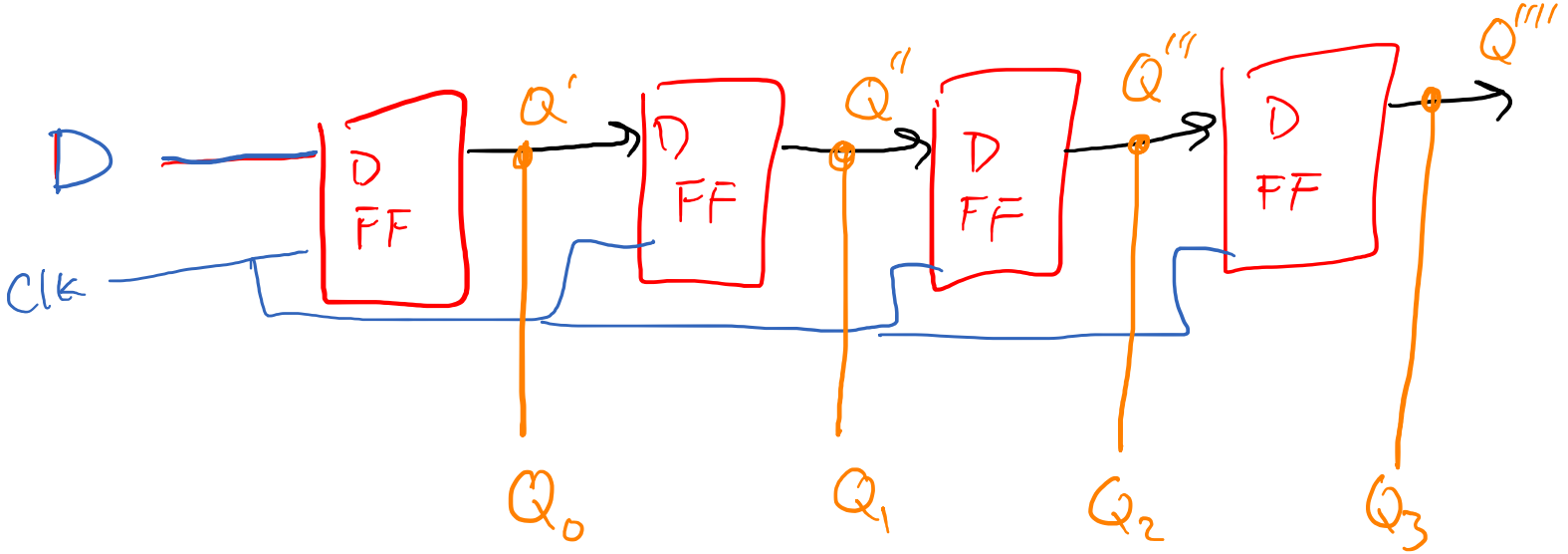
How do we fix this?

Correcting `for(;;)` Loops in Verilog

```
integer i;
wire carry;
assign sum[0] = a[0] ^ b[0] ^ c_in;
assign carry[0] = a[0] & b[0] | a[0] & c_in | b[0] & c_in;
assign sum[1] = a[1] ^ b[1] ^ carry[0];
assign carry[1] = a[1] & b[1] | a[1] & carry[0] | b[1] & carry[0];
assign sum[2] = a[2] ^ b[2] ^ carry[1];
assign carry[2] = a[2] & b[2] | a[2] & carry[1] | b[2] & carry[1];
assign sum[3] = a[3] ^ b[3] ^ carry[2];
assign carry[3] = a[3] & b[3] | a[3] & carry[2] | b[3] & carry[2];
assign c_out = carry[3];
```

Serial Communication

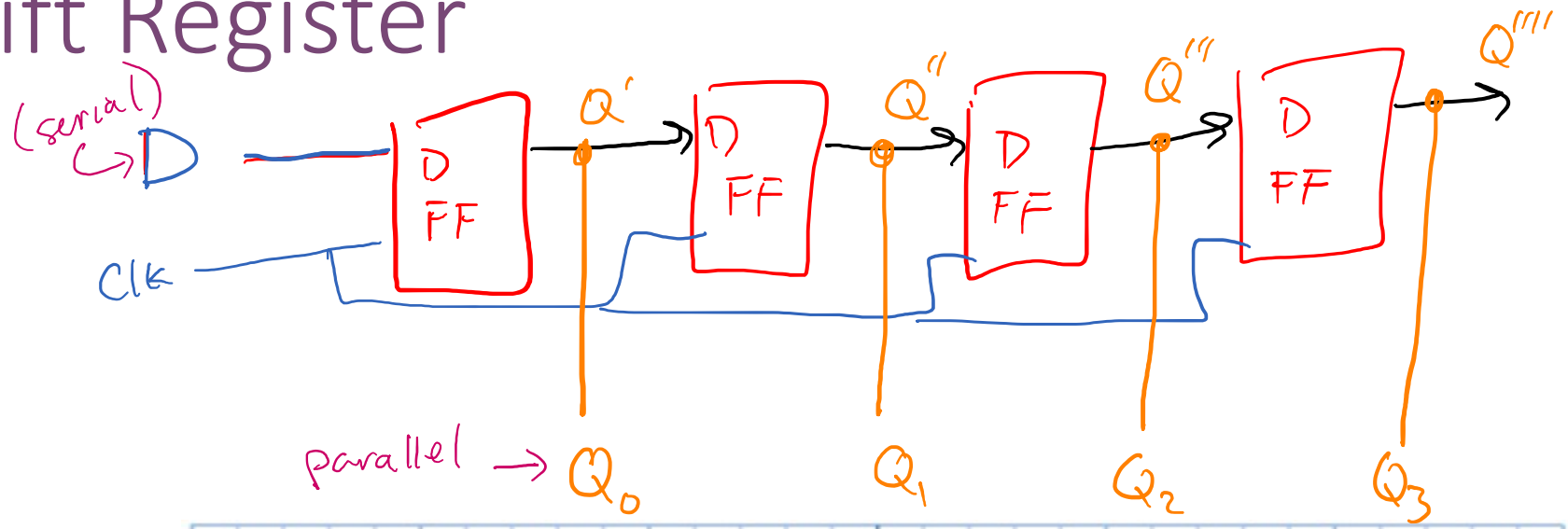
- Do you remember this circuit?

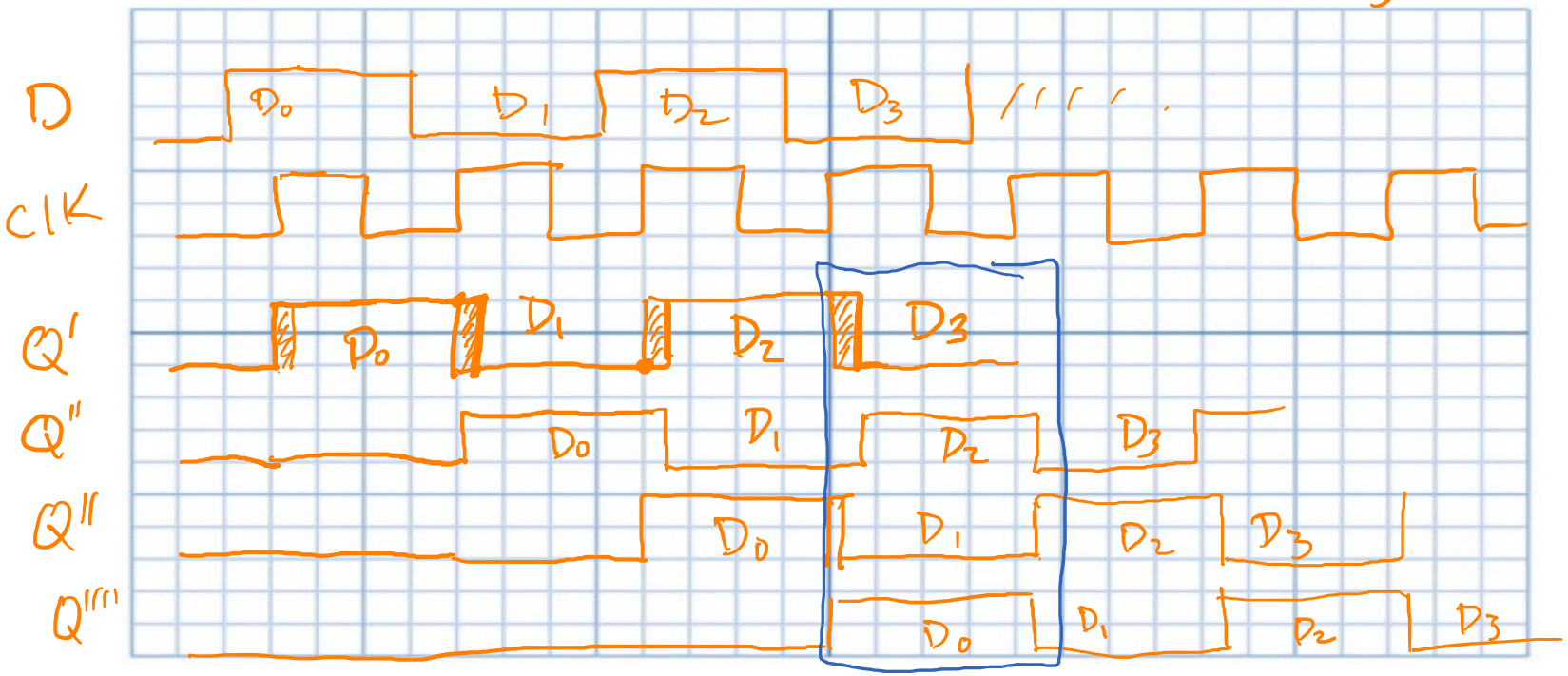


- What did it do?

Converts serial input to parallel output

Shift Register

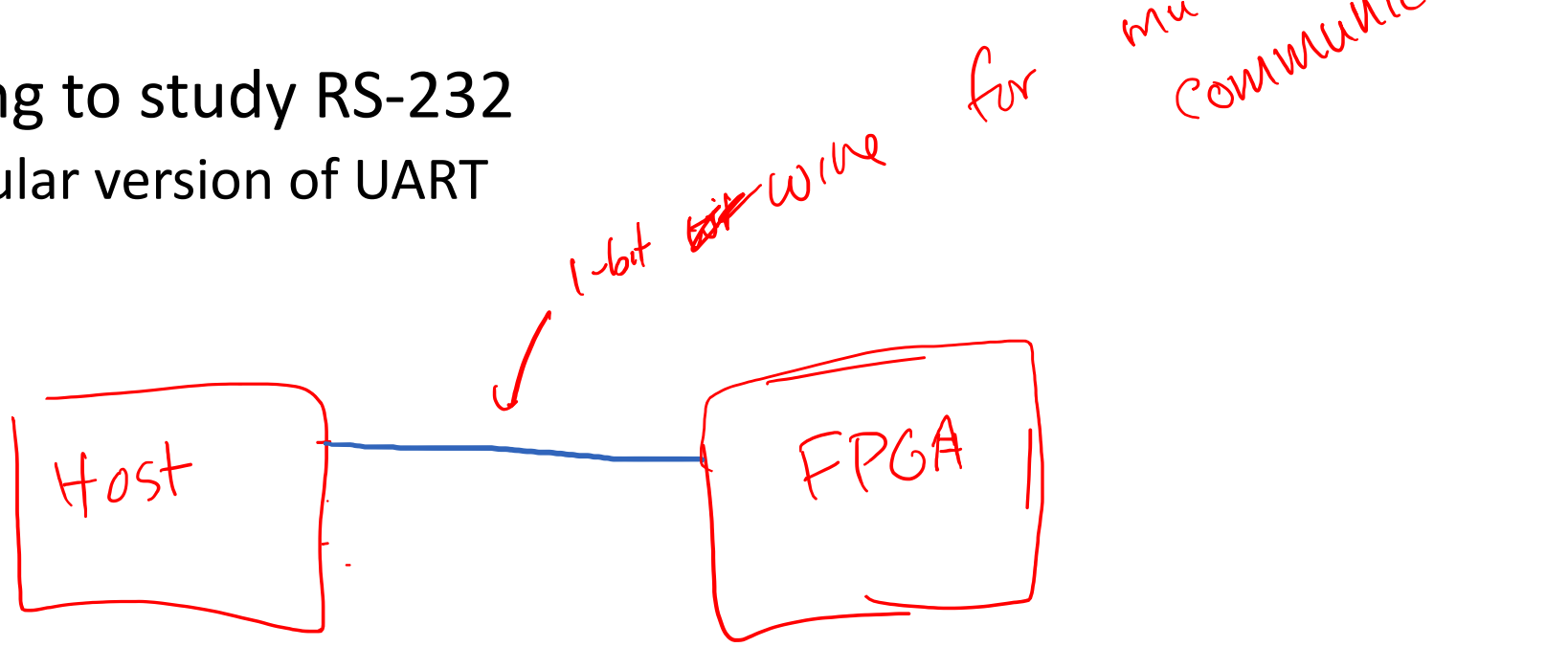




UART

- **U**niversal **A**synchronous **R**eceive-**T**ransmit

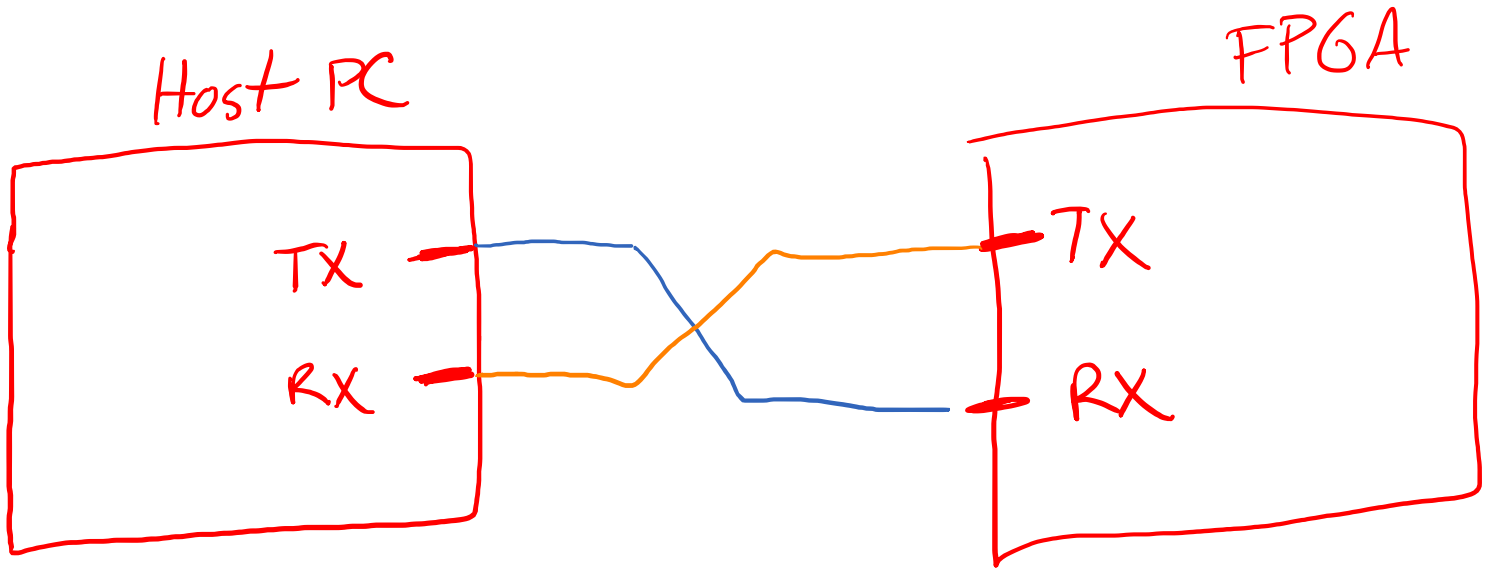
- We're going to study RS-232
  - A particular version of UART



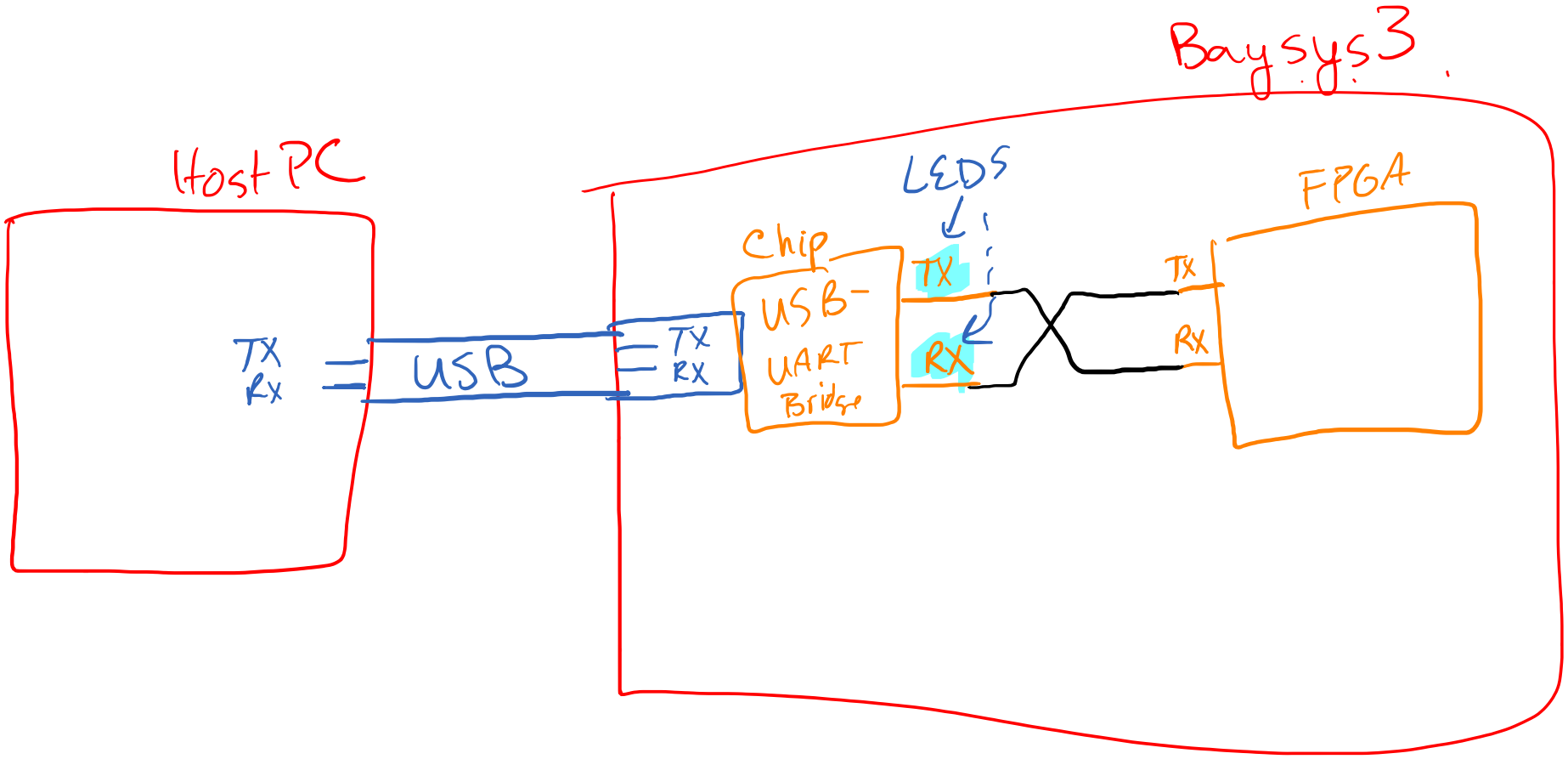
UART RX/TX

RX = Receive

TX = transmit



UART RX/TX on Basys3

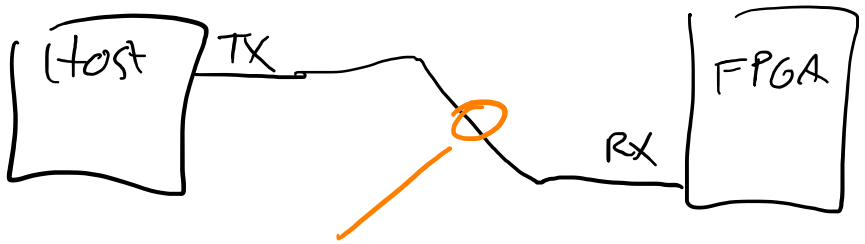


UART RX/TX LEDs on Basys3

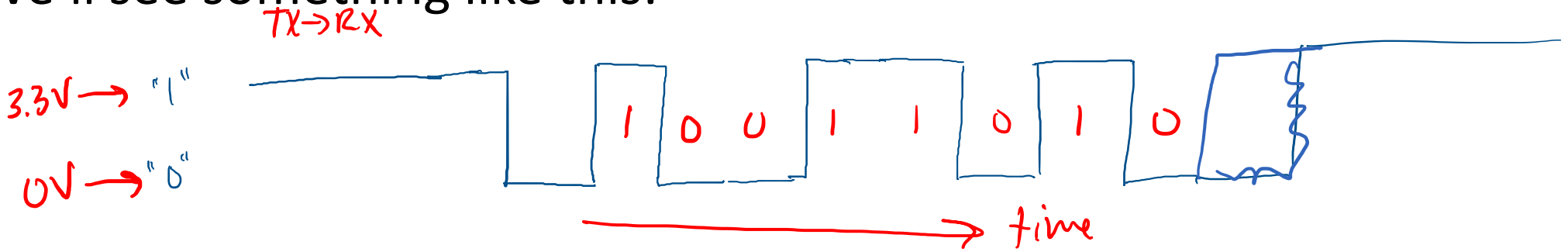
- A word of caution:
- The Basys3's RX + TX LEDs are backwards from what you expect.
- They are the USB adaptor chip's RX+TX, not the FPGAs.

UART “Frame”

- If we monitor the RX line



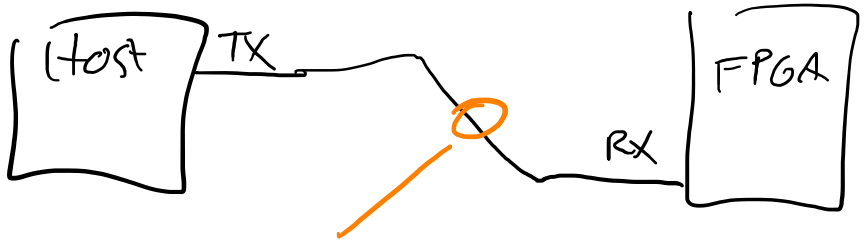
- We’ll see something like this:



- But what does it “mean”?

0 1 0 1 1 0 0 1 = 0x59

UART Frame



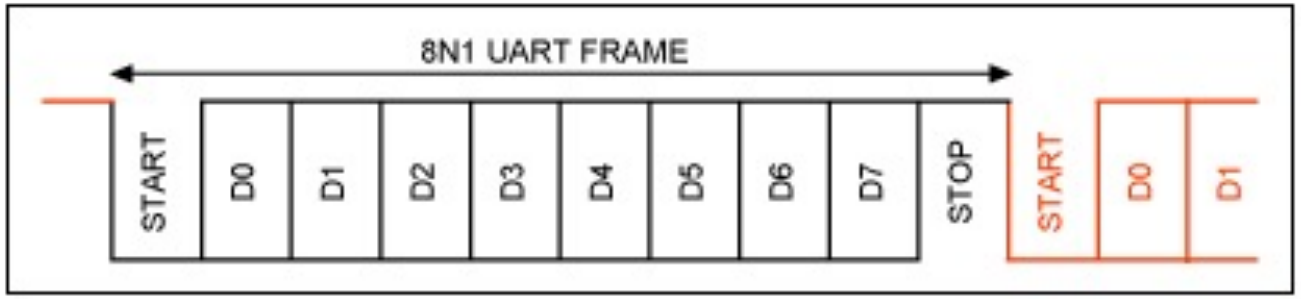


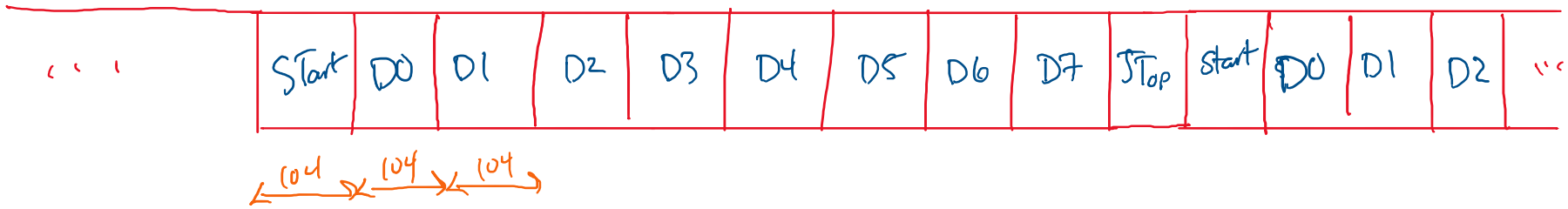
Figure 1. A typical UART data frame.

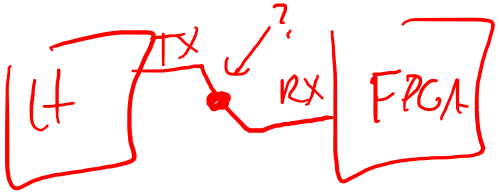
UART Frame Rates

- We're using 9600 baud
- Baud = bits per second
  - Includes start/stop bits

frequency = 9600 bits/second
= 9600 Hz

Period = $\frac{1}{9600} \approx 104\ \mu sec$



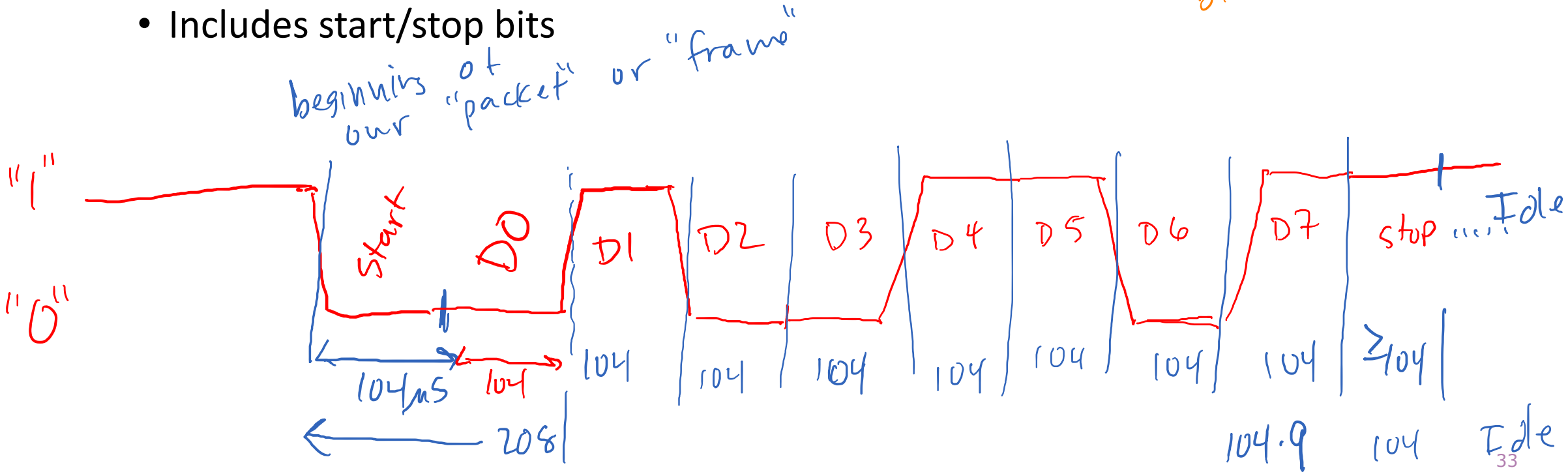


UART Frame Rates

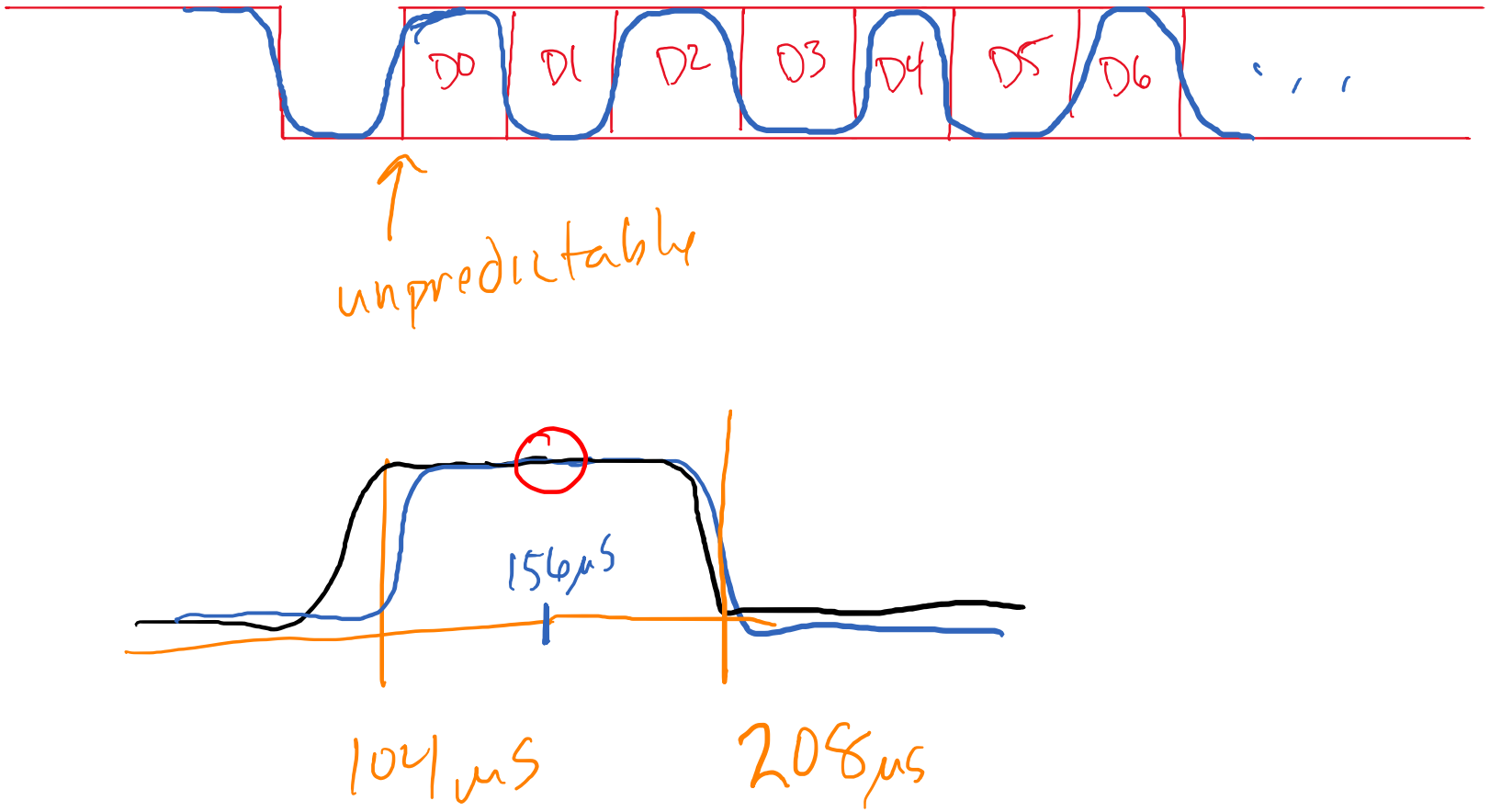
" " ≈ 9kb/sec

- We're using 9600 baud
- Baud = bits per second
  - Includes start/stop bits

10 bits/ 8-useful bits



UART Frame Detection



UART Frame Timing

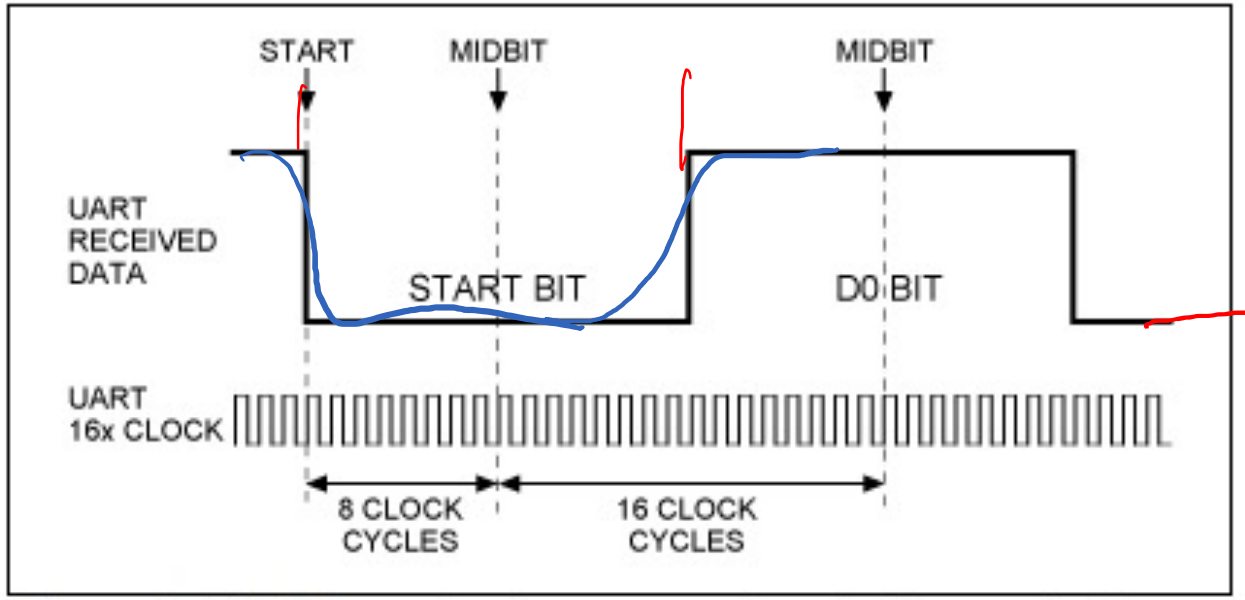


Figure 2. UART receive frame synchronization and data sampling points.

UART RX State Machine