ENGR 210 / CSCI B441
“Digital Design”

UART II

Andrew Lukefahr

Course Website

fangs-bootcamp.github.io

Write that down!

Announcements

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

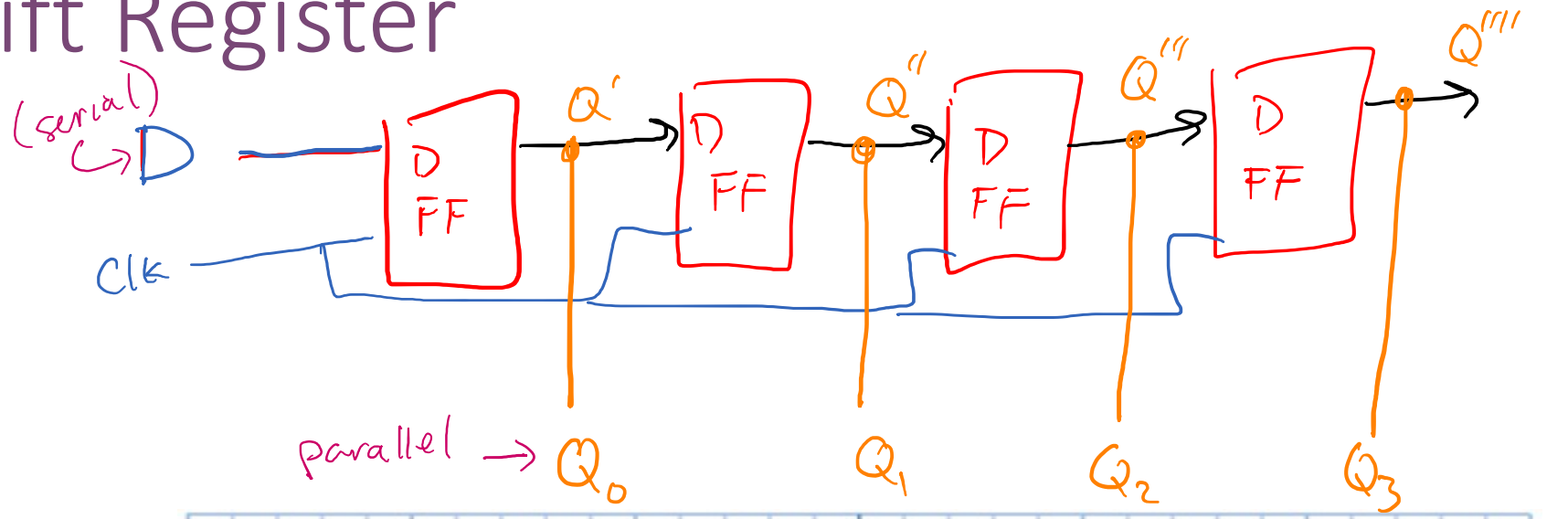
Always specify defaults for `always_comb`!

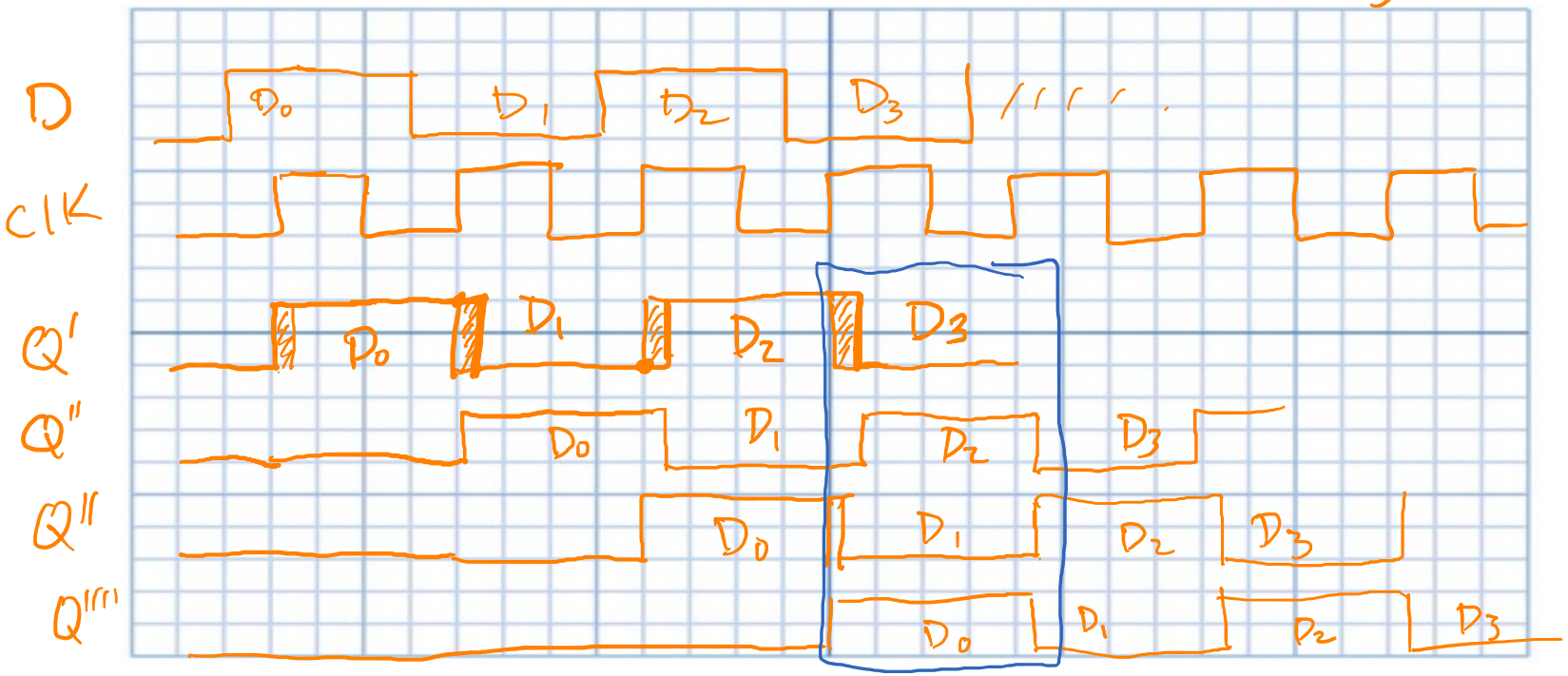
BLOCKING (=) FOR `always_comb`

NON-BLOCKING (<=) for `always_ff`

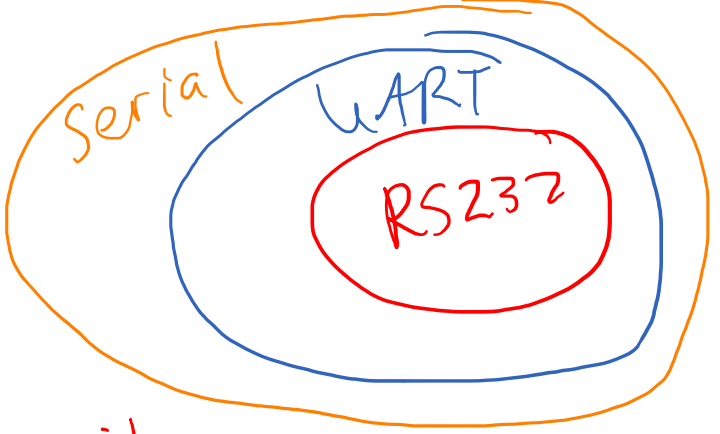
Converts serial input to parallel output

Shift Register



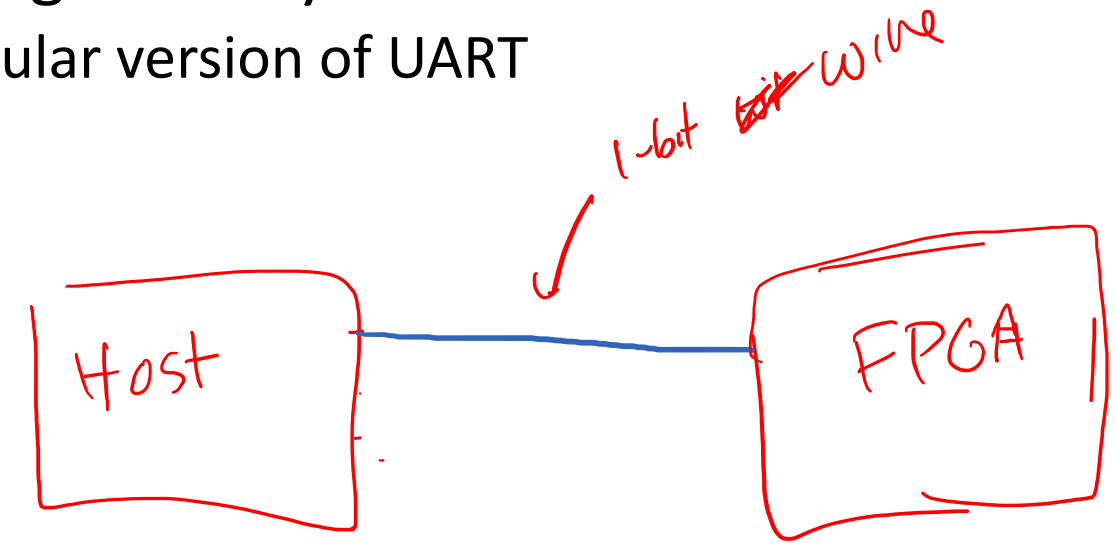


UART == Serial == RS232



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

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

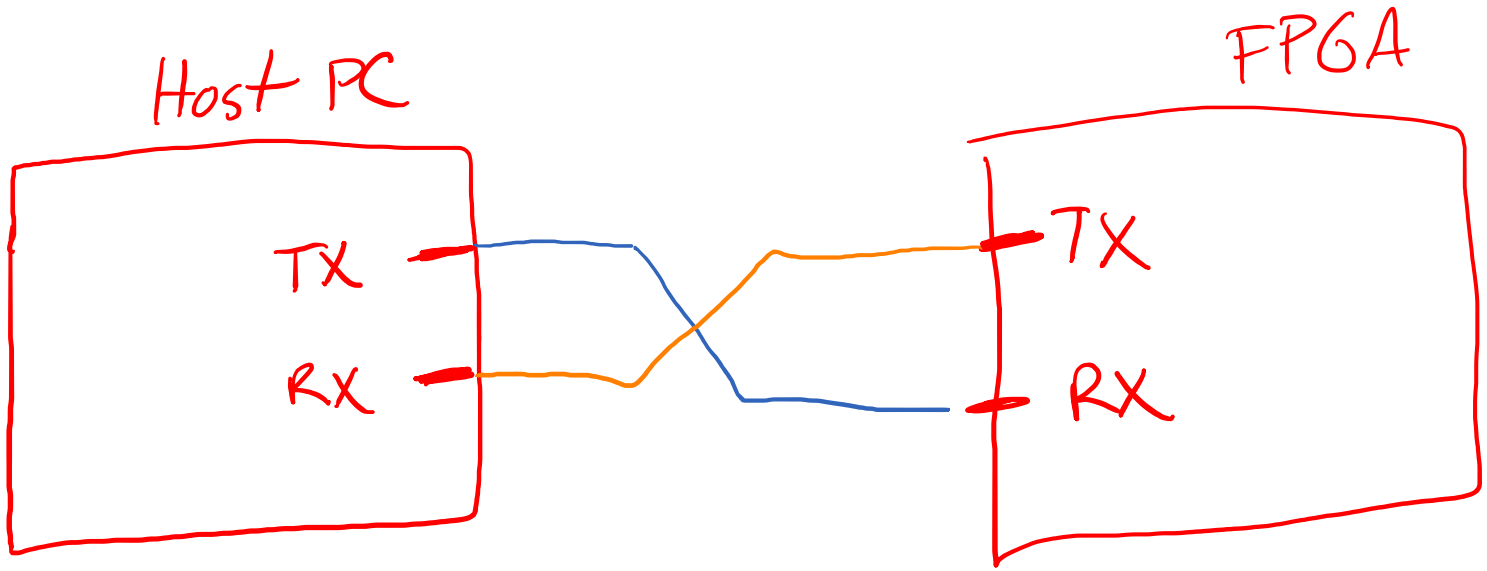


for multi-bit communication

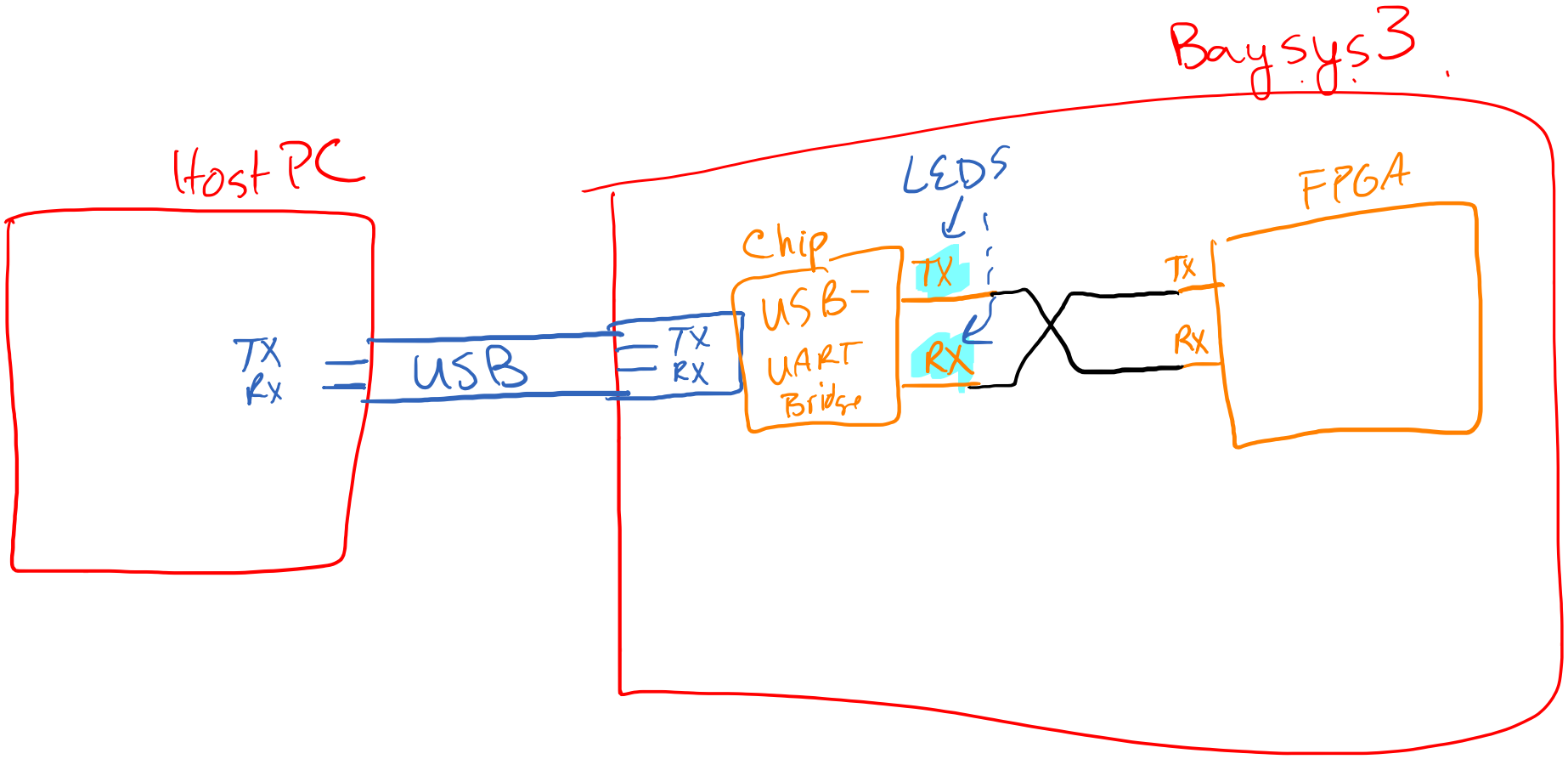
UART RX/TX

RX = Receive

TX = transmit



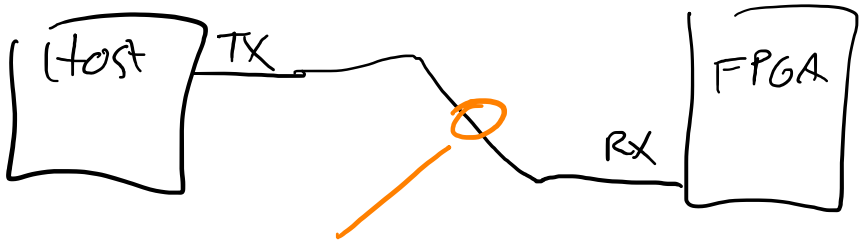
UART RX/TX on Basys3



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



time

"1"

"0"

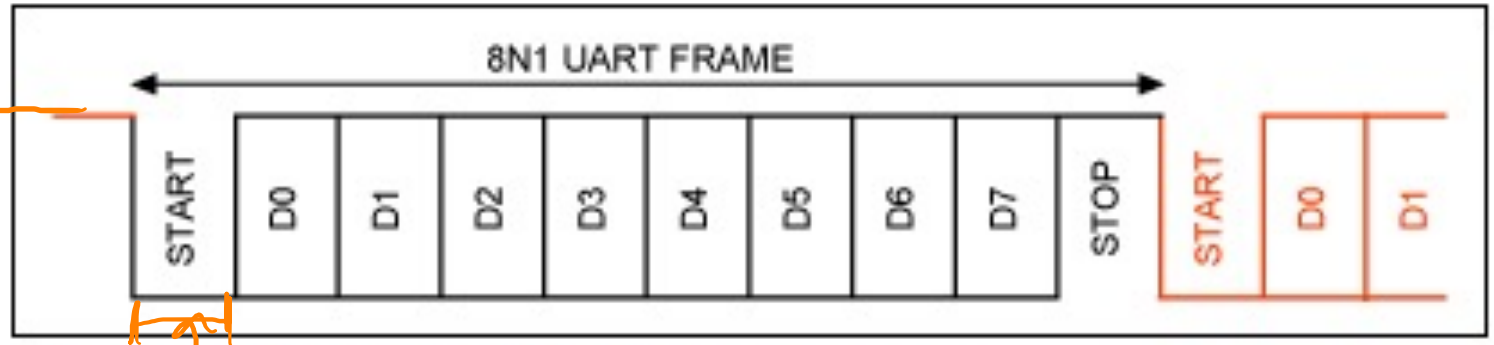


Figure 1. A typical UART data frame.

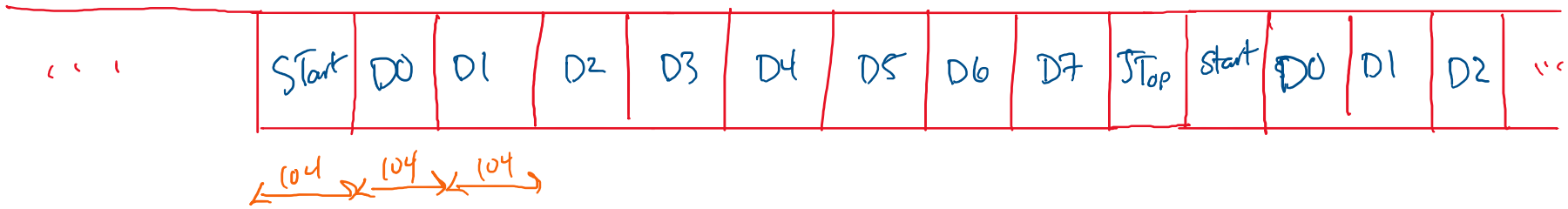
104 μs

UART Frame Rates

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

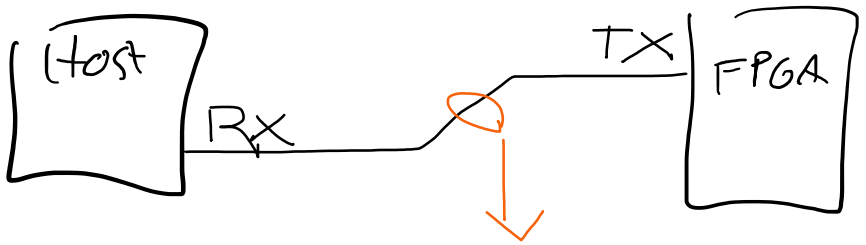
frequency = 9600 bits/second
= 9600 Hz

Period = $\frac{1}{9600}$ ≈ 104 μsec /bit

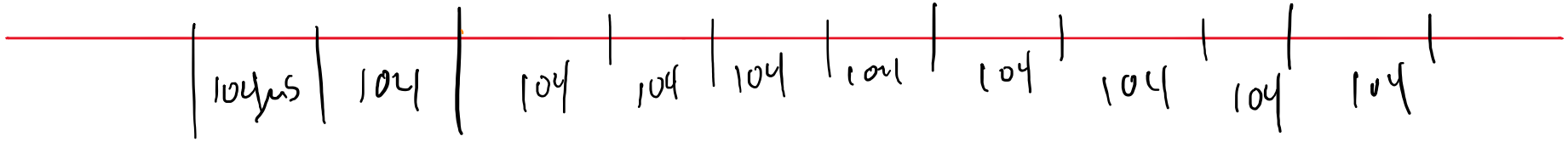


UART: TX

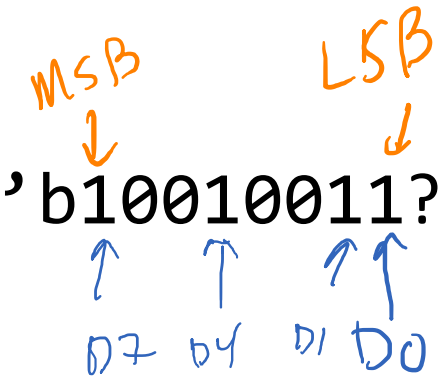
- How to transmit `8'b01101010`?



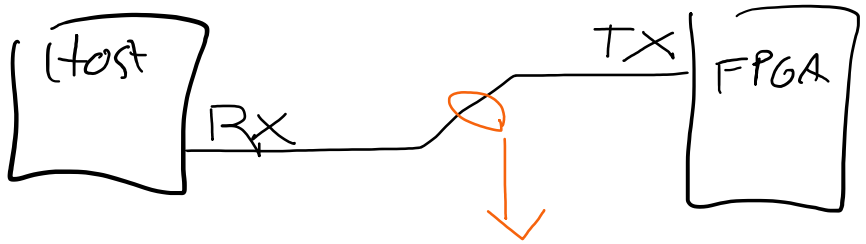
- Draw the **packet!**
  - Hint: UART is transmitted LSB -> MSB



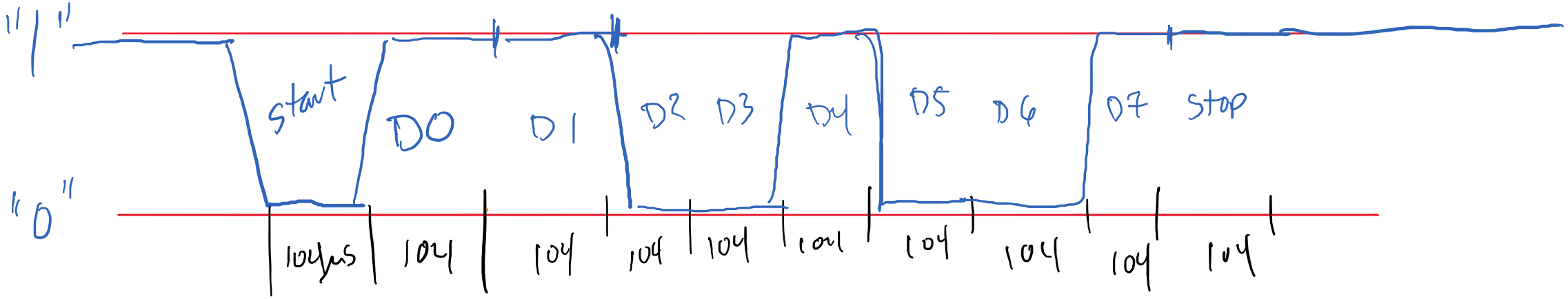
UART: TX



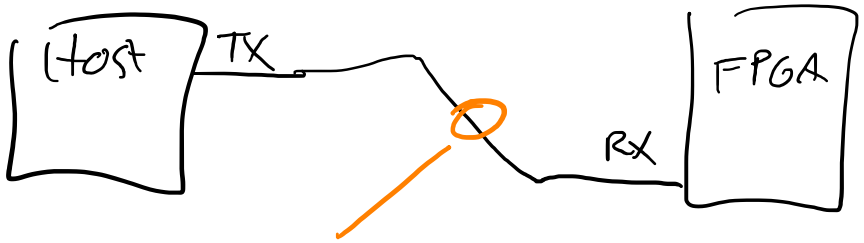
- How to transmit `8'b10010011`?



- Draw the **packet!**
  - Hint: UART is transmitted LSB -> MSB



UART Frame RX



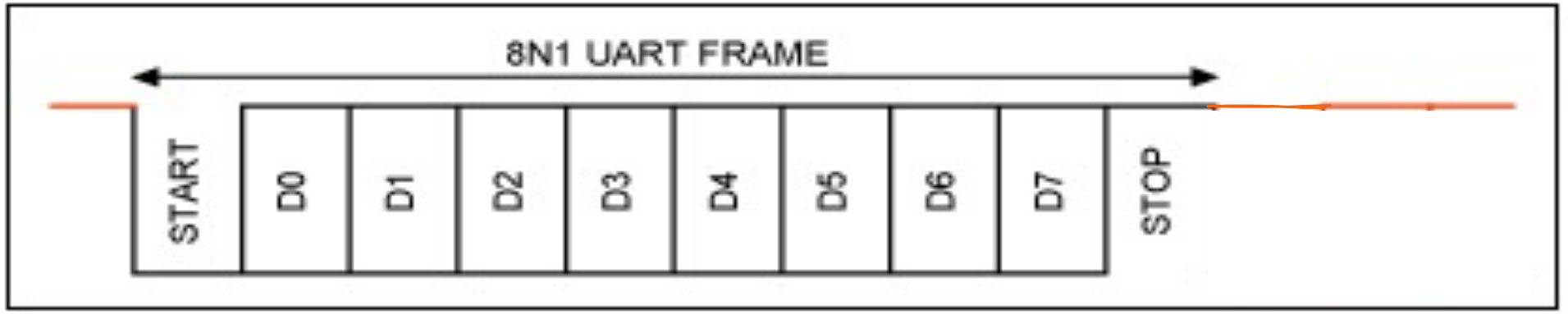


Figure 1. A typical UART data frame.

UART RX Frame Timing

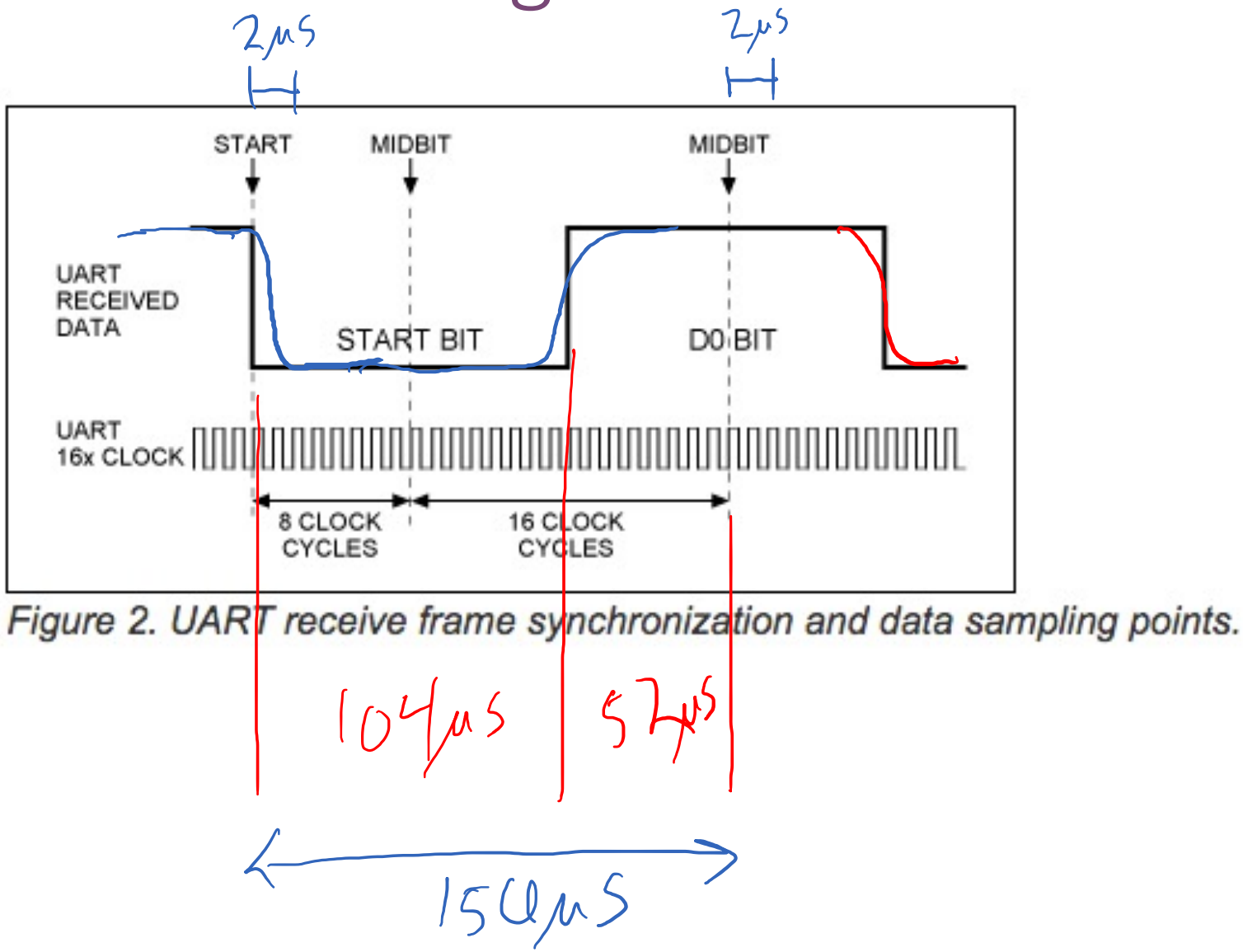
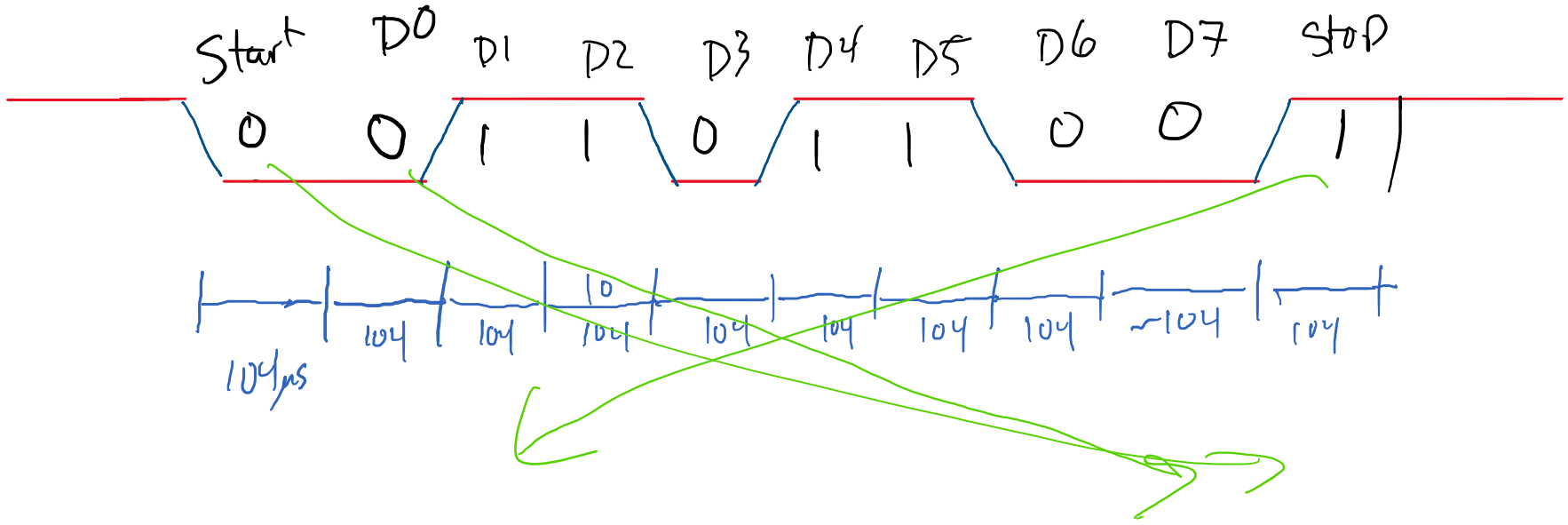


Figure 2. UART receive frame synchronization and data sampling points.

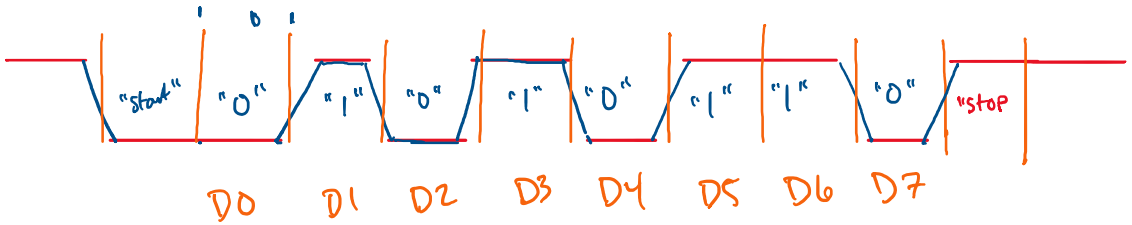
UART RX

- What **data** is this?
  - Recall: LSB first



8'b 0011 0110 = 8'h=36

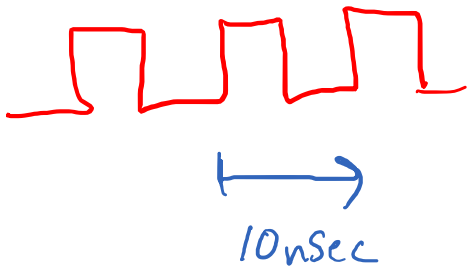
UART TX State Machine



- How to transmit `8'b01101010`?

UART TX: Timing

CLK100MHZ



- Basys3 CLK100MHz = 100MHz clock
- How do we create a 104uS delay?

$f = 100\,MHz = 100_{10}{}^{6}\ cycles/sec$

$P = \frac{1}{f} = \frac{1}{100_{10}{}^{6}} = \frac{1_{10}{}^{-6}}{100} = 1_{10}{}^{-8}\ sec = 10_{10}{}^{-9}\ sec = 10\,nSec$

$10\,nSec \cdot X = 104\,\mu Sec \Rightarrow 10\,nSec \cdot X = 104000\,nSec$

$10_{10}{}^{-6} \cdot X = 104_{10}{}^{-3}$

$X = \frac{104000}{10} = 10400\ cycles$

Simple Countdown Timer

```
timer tim0 (
    .clk(clk),
    .load(load),
    .data(data),
    .trigger(trigger)
);
```

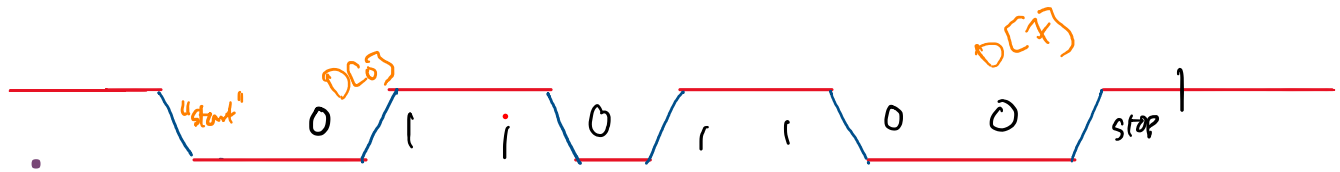
```
module timer (
        input clk,
        input load,              // load_request
        input [31:0] data,       // <- 32-bit timer
        output trigger
);

reg [31:0] count;

always_ff @(posedge clk) begin
        if (load)       count <= data;
        else if (count != 0)
                        count <= count - 32'h1;
end

assign trigger = (count == 0);

endmodule
```

Fun Extra: Parameterizable Modules

```
timer #(
   .TMR_BITS(16) //16-bit
) tim0 (
    .clk(clk),
    .load(load),
    .data(data),
    .trigger(trigger)
);
```

```
module timer #(
        parameter TMR_BITS = 32
        ) (
        input clk,
        input load,
        input [TMR_BITS-1:0] data,    // <- 32-bit timer
        output trigger
);

reg [TMR_BITS-1:0] count;

always_ff @(posedge clk) begin
        if (load)       count <= data;
        else if (count != 0)
                        count <= count - 'h1;
end

assign trigger = (count == 0);

endmodule
```

UART RX: State Machine



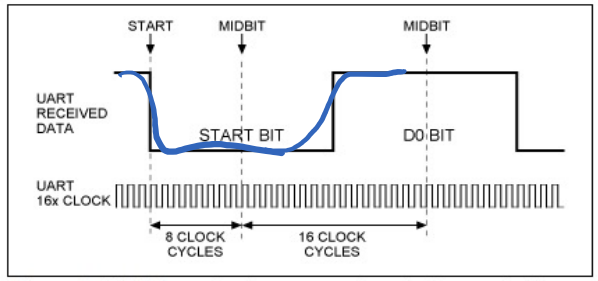
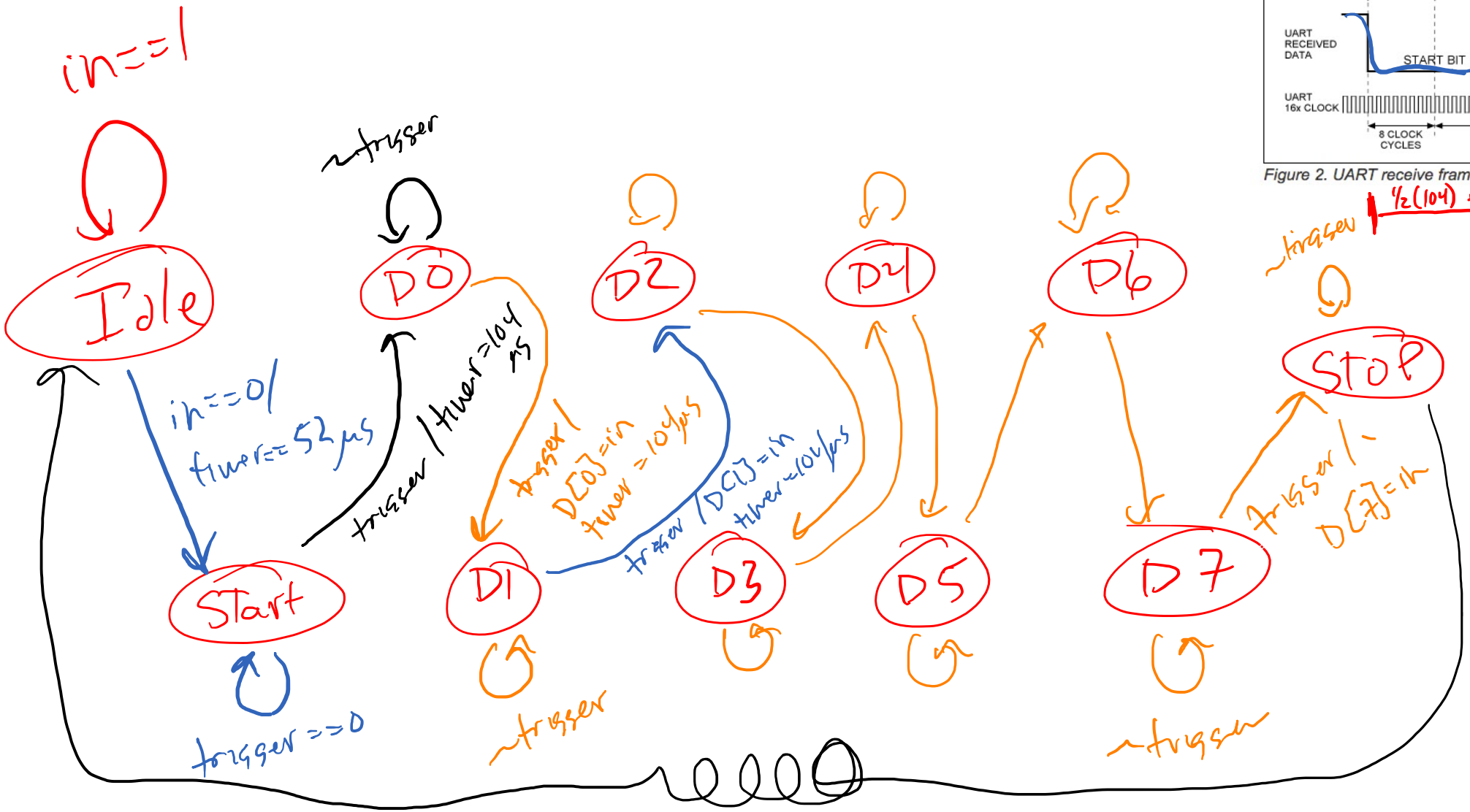
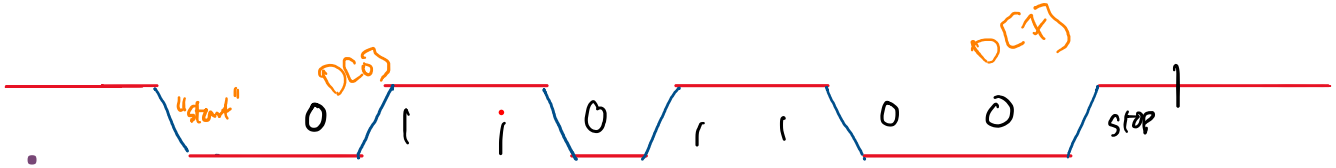


Figure 2. UART receive frame synchronization and data sampling points.



UART RX: State Machine



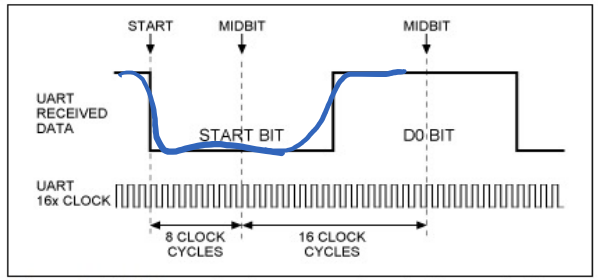


Figure 2. UART receive frame synchronization and data sampling points.

½(104) + 104ns

UART RX: Shift Registers

- Rather than explicitly assign destination indexes, can also use a shift register

```
always_ff @(posedge clk) begin
     //other code here!
     if (rst)
               shift_reg <= 8'h0;
     else if (shift_in)
               shift_reg <= {in, shift_reg[7:1]};
     else //optional
               shift_reg <= shift_reg;
end
```

UART TX: Shift Registers

- Rather than explicitly assign destination indexes, can also use a shift register

```
always_ff @(posedge clk) begin
      //other code here!
      if (load)
                  shift_reg <= load_value;
      else if (shift_out)
                  shift_reg <= {1'h0, shift_reg[7:1]};
      else //optional
                  shift_reg <= shift_reg;
end

assign out = shift_reg[0];
```

P5: UART

- You get to build a UART interface
- P5: just echo RX back over TX

- Connect your FPGA to your PC
- Allows you to “talk” to your FPGA with keyboard

- P6: we add python UART interface

Next Time

- Memory