# ECE 337: ASIC Design Lab 6

Simplified USB Receiver

Individual Mini-Design Project

*Week of 9/28/2015*

## Lab 6 High-Level

- Will be designing a simplified USB Receiver
- Two-week, **individual lab**
- Two main phases with optional source grade check
  - Preparation Diagrams
    - Due late night 2 days prior to your Design deadline
      - Signed-off OR submitted electronically
  - Phase 1: Sub-block code & test benches
    - Due start of lab next week
  - Optional complete source only grade submission
    - 48 hours prior to Phase 2 deadline
  - Phase 2: Complete mapped version
    - Start of your Lab week after next (Week of 10/5)

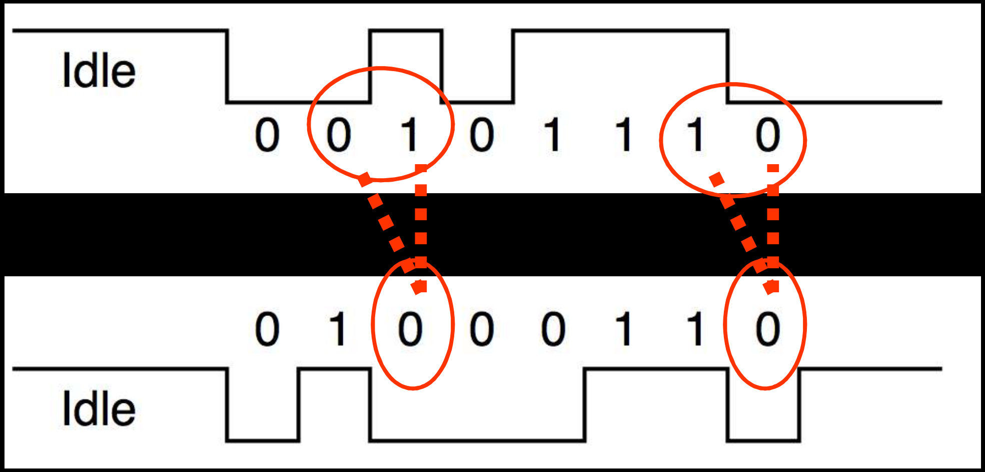
## Simplified USB Interface

- Read-only USB interface, subset of USB 1.1
  - Recognizes USB data packets
  - Puts data into a FIFO
  - Higher level functions handled by external controller (CRC decode, addressing protocol etc.)
- Differential input lines D+, D-
- NRZi (inverted Non Return to Zero) encoding
  - requires synchronous edge detection
- Sync (start) byte
- Clock synchronization (using data transitions)
- Real USB includes “bit-stuffing” (we won’t)
  - used to ensure that clock is resynchronized often enough
- End of packet signal – D+ = D- = 0
- Data written to FIFO Queue for interface to external device (provided)

## NRZi encoding/decoding 0's

NRZi-Encoded Data:

Recovered Data:



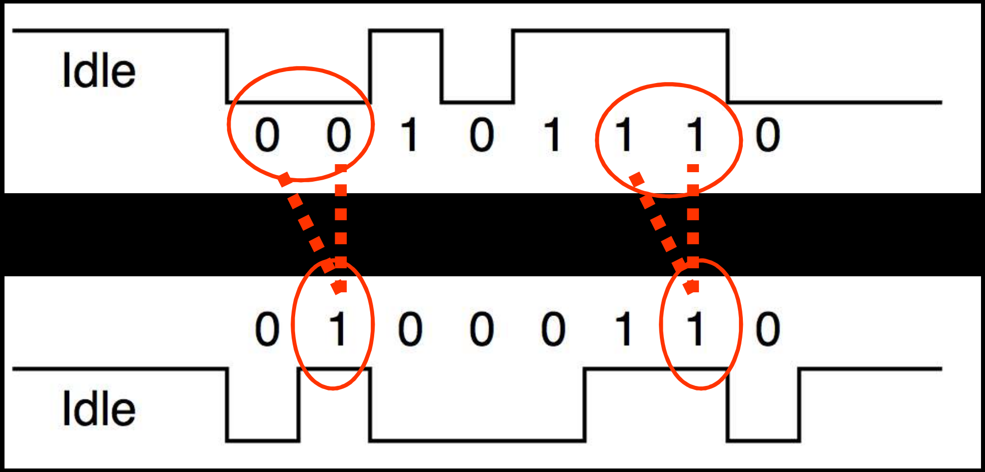
| Original Data | | Encoded data |
|---|---|---|
| **0** | **=>** | **0->1 or 1->0 transition** |
| 1 | => | no transition |

To decode: have to compare current and previous bit.
If same, data = 1. If different, data = 0.

## NRZi encoding/decoding 1's

NRZi-Encoded Data:

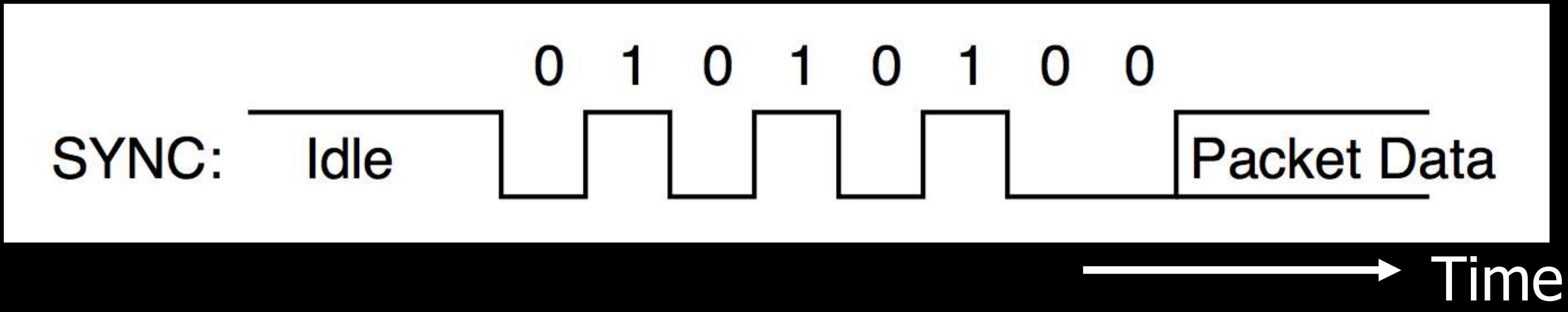
Recovered Data:

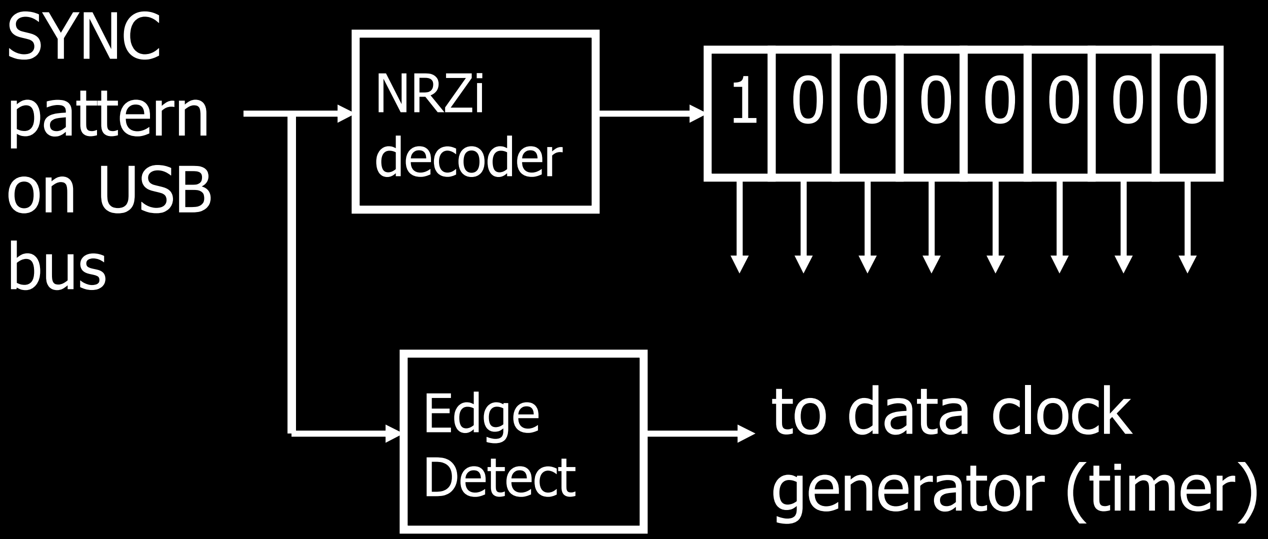


| Original Data | | Encoded data |
|---|---|---|
| 0 | => | 0->1 or 1->0 transition |
| **1** | **=>** | **no transition** |

To decode: have to compare current and previous bit.
If same, data = 1. If different, data = 0.

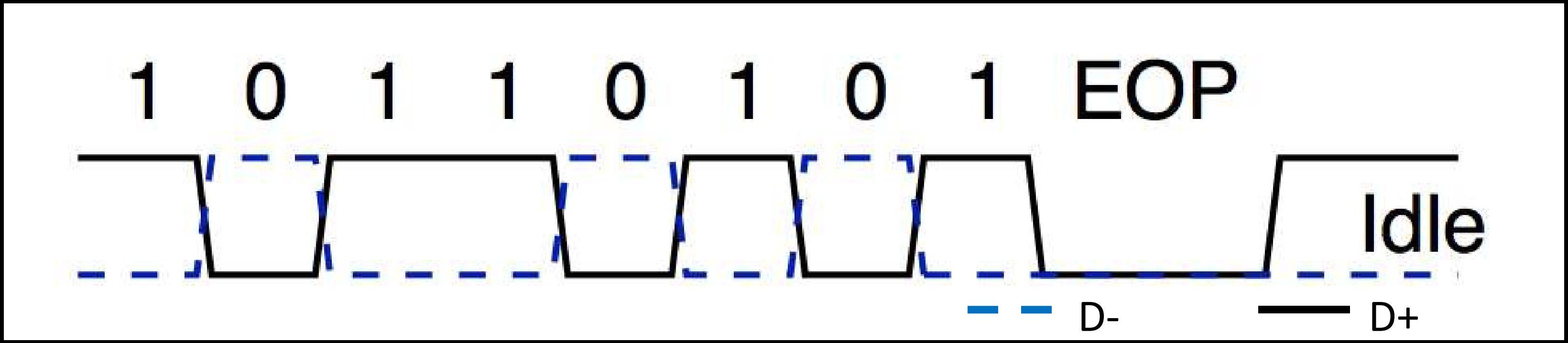
# Sync Byte (encoded)





Which bits of the incoming waveform were decoded to produce the MSB in the shift register? the LSB?

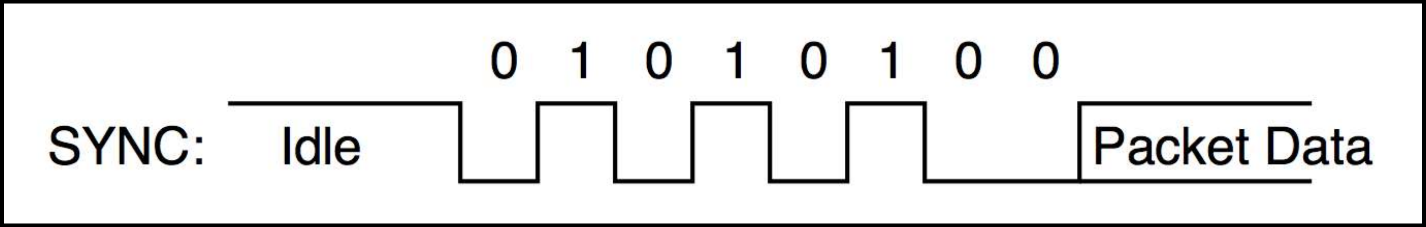
# End of Packet (EOP) Signal



- USB uses two data lines
  - Normally one is the complement of the other
  - Eliminates common mode noise
- Both data lines are pulled low to signal EOP
  - Can be detected by a single logic gate
  - EOP lasts for 2 data bit periods

## Complete Packets

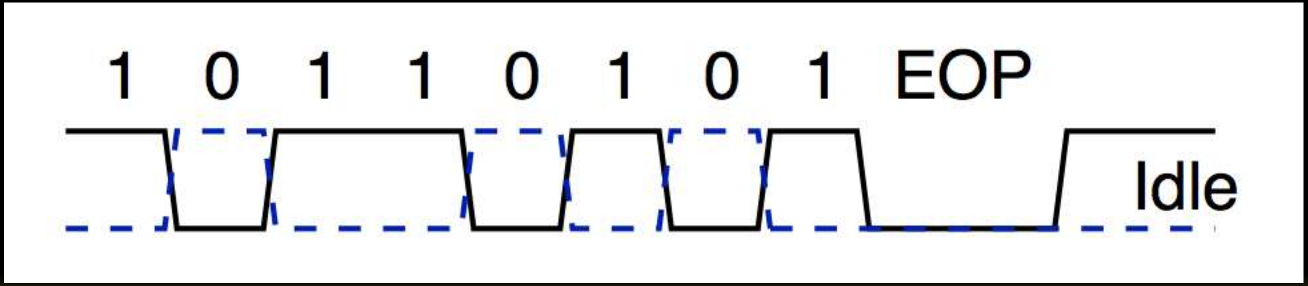
Begins with SYNC byte



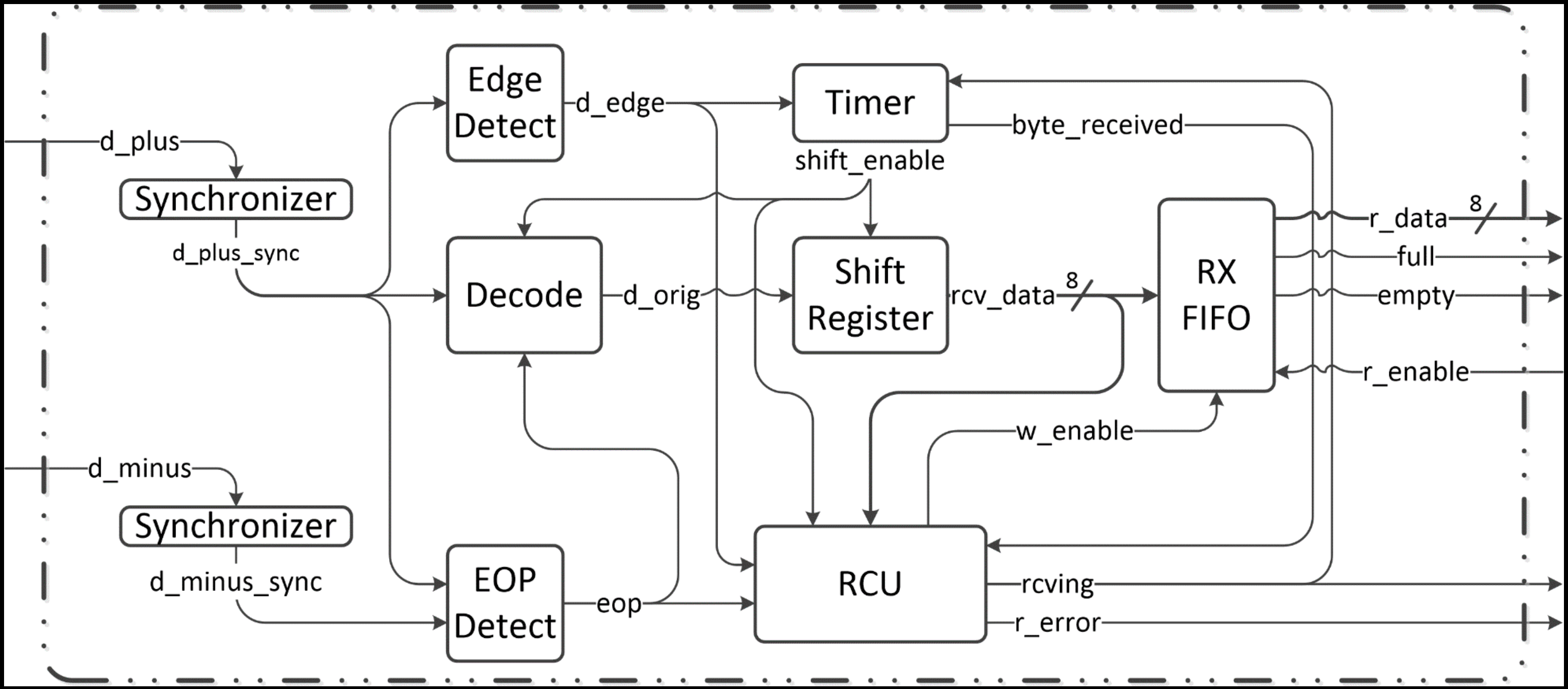
Packet body is decoded and stored in FIFO

- One byte at a time

Ends with EOP signal



## System Overview

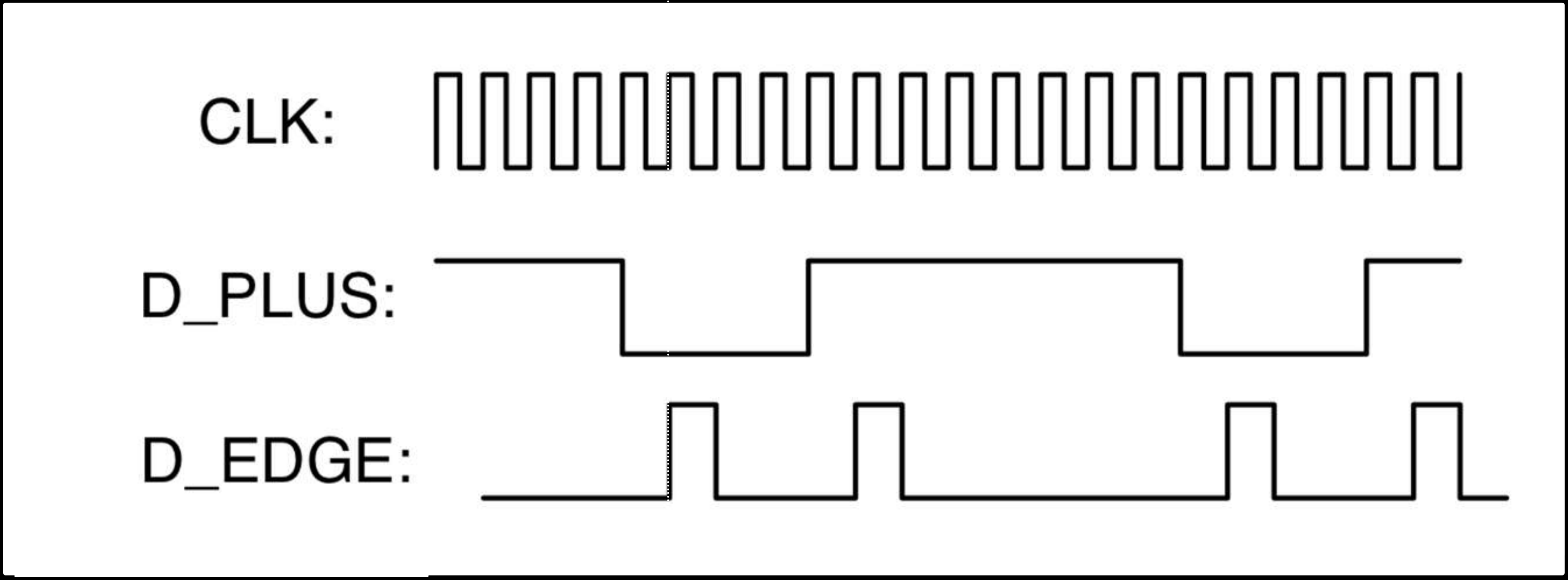


## Decode

- Reverses NRZI encoding
- Done inline with data stream to shift register
- Synchronously compare current value and immediate past data bit

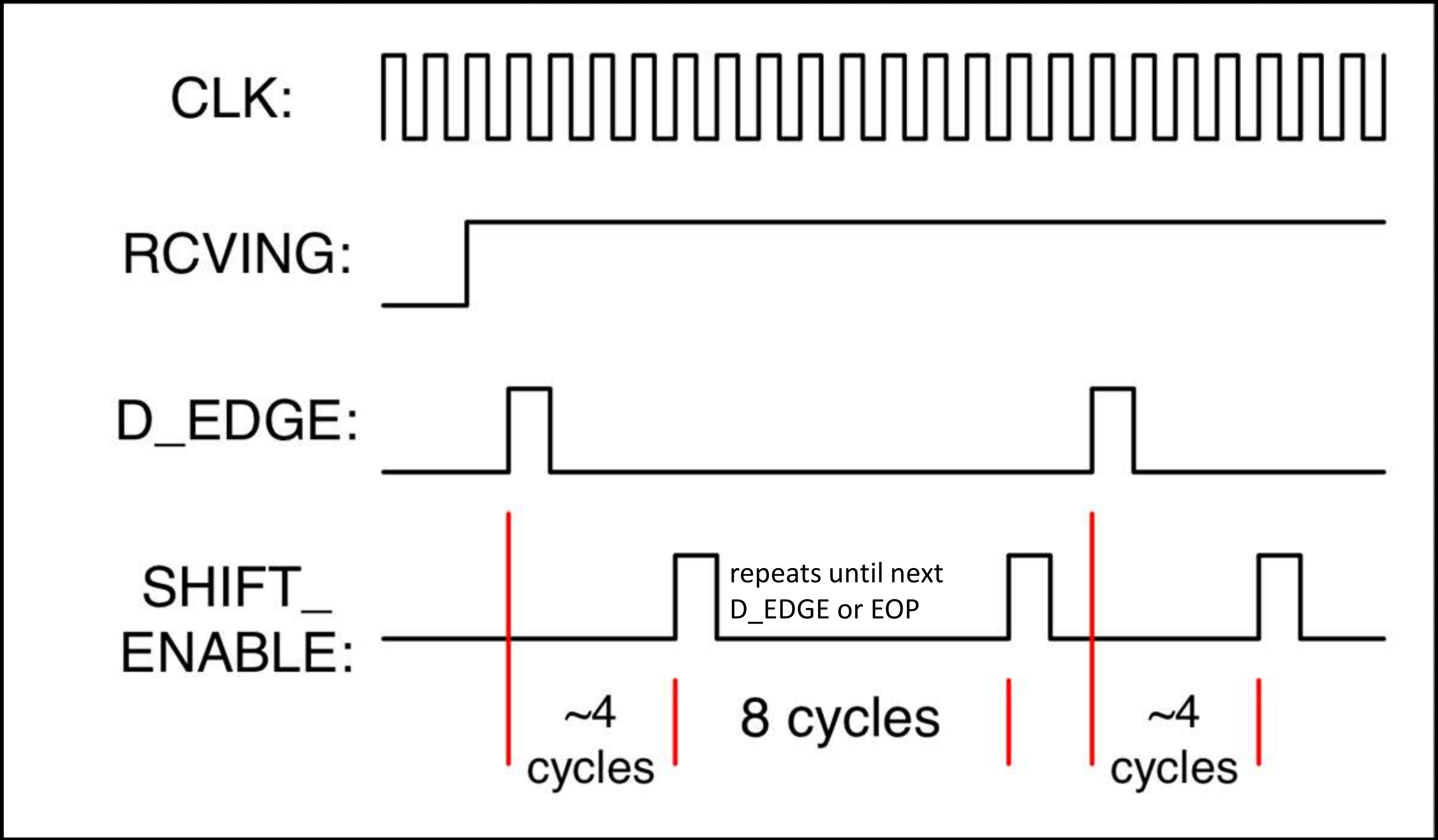
## EDGE_DETECT

- Detects transitions on the D_PLUS input
- Synchronizes TIMER block



## TIMER

- Shifts next bit every 8 clock cycles

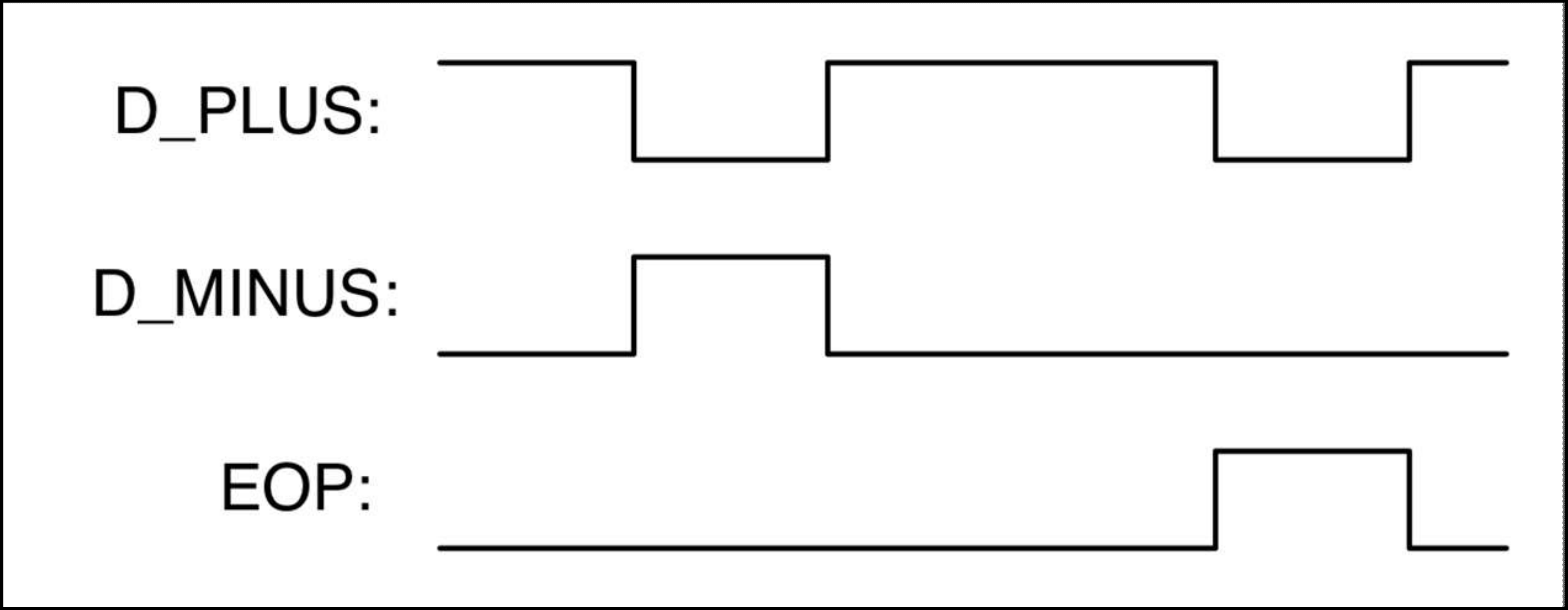


## RX FIFO

- The FIFO is **provided**, you need to make a “wrapper” file.
- FIFO writes data from write_data on a clock edge, while write_enable is high.
- FIFO outputs data on read_data, increments to next byte when read_enable is high and there is a clock edge.
- fifo_full and fifo_empty indicate FIFO status
- Provided FIFO has two clocks
  - Why? This FIFO was designed to provided interface between circuits running on different clocks
  - you will tie them to same system clock

## EOP_DETECT

- High when D_PLUS and D_MINUS are both low.
- Can be pure combinational block



## RCU

- Controls USB Receiver circuit
- Begins receiving when an edge is detected.
- Checks to see if the SYNC byte was received
  - Error condition if not
- Activate the storing of each data byte into the FIFO
  - Told by timer when a new byte is ready
- Stop receiving when the EOP is detected.

## Sequence of Operation

1. On reset, the RCU goes to the idle state
2. When an edge is first detected
    - Begin receiving data through the DECODE block to the SHIFT_REG.
    - Set the RCVING line high.

## Sequence of Operation Continued

3. After the first byte is shifted in
   - Check that byte matches USB SYNC byte
   - If byte matches SYNC, do not store to FIFO, but begin receiving and storing data from next byte
   - If the byte does *not* match the SYNC byte
     - Set R_ERROR flag to 1
     - Disregard any input until the next EOP is reached
     - RCVING line should remain high until the EOP is reached
     - R_ERROR flag should remain high until the next packet begins

## Sequence of Operation Continued

4. Continue receiving and storing data to FIFO until the EOP is detected
    - Then set the RCVING line low.
    - If an EOP is reached with an incomplete byte in the shift register
        - Set the ERROR flag high and do not store the last byte.
        - Leave the R_ERROR flag high until the next packet begins.
    - Note: Do not worry about FIFO overflow as it will not be graded