USB Receiver Unit

- Two-week, individual lab
- Three phases
  - Diagrams, code and sub-block test benches (start of lab, Feb 24-26)
  - Submit source and top-level test bench (11:59 PM, March 1-3)
  - Submit mapped version (start of lab, March 3-5)

Project

- Think of a design that is applicable for an ASIC (or a FPGA). A microcontroller should not be adequate due to high speed/low power requirements (NO ELEVATOR CONTROLLERS)
- Think about a parallel design
- Blackboard contains past project ideas. Good ideas include protocol interfacing and signal processing… (no math co-processors)
- Due to time limitations, a subset of a given protocol may be acceptable
- Work in groups of 3-4 ONLY IN YOUR SECTION
- See Project Idea Guidelines posted online (due next week), you are encouraged to submit multiple ideas.

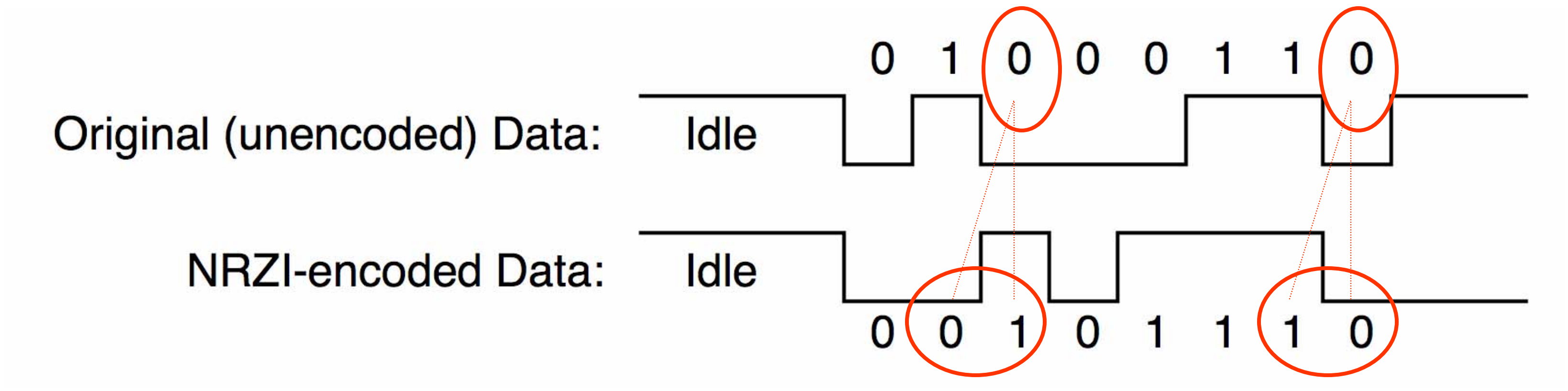
Simplified USB Interface

- 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

how well would this work for UART?

- 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
- Date written to FIFO Queue for interface to external device (provided)

NRZi encoding/decoding



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

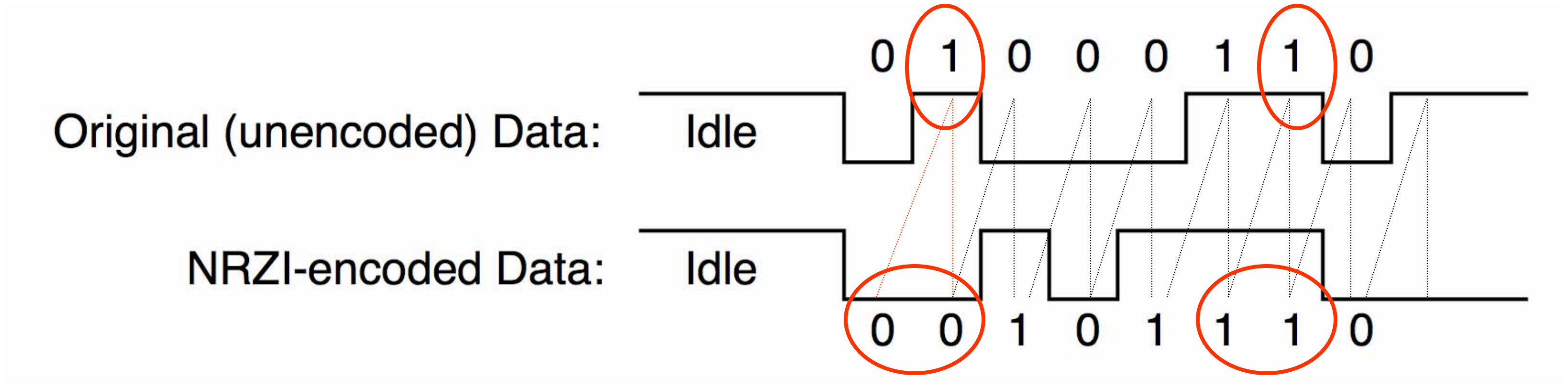
To decode: have to compare current and previous bit.
If same, data = 1. If different, data = 0.

What logic gate has similar behavior?

a. NAND
b. XOR
c. XNOR
d. INV

NRZi encoding/decoding

Note – the output of an actual encoder or decoder will be delayed by one bit period.



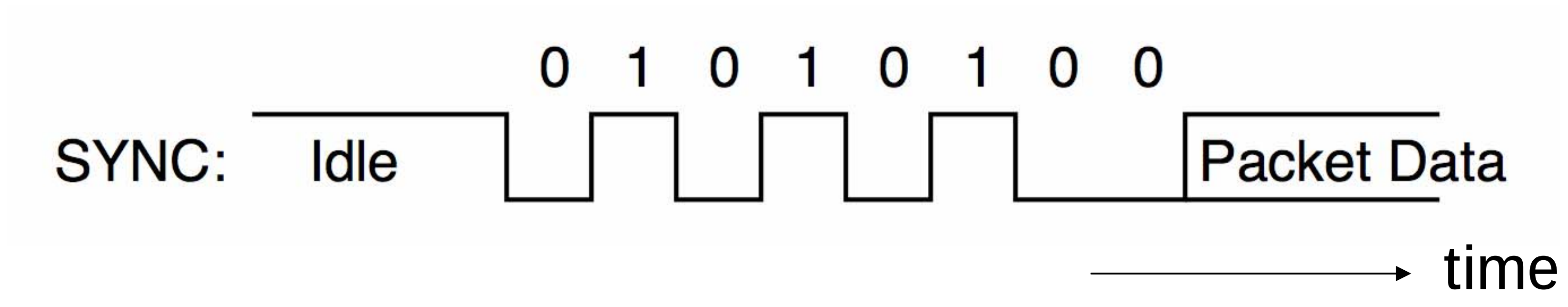
Original Data		Encoded data
0	=>	0->1 or 1->0 transition
1	**=>**	**no transition**

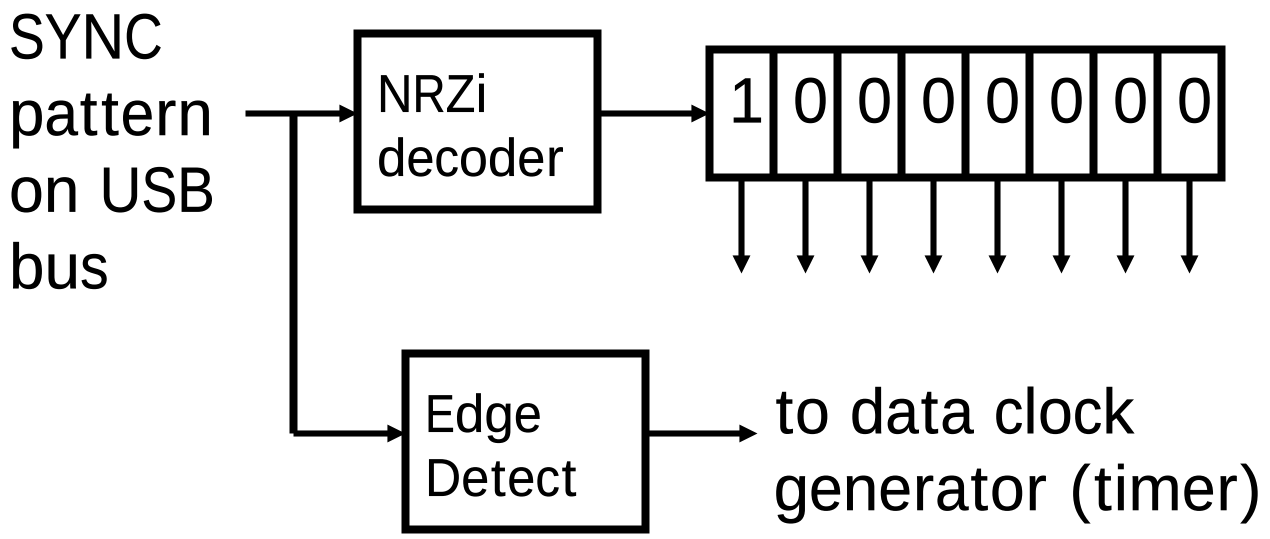
To decode: have to compare current and previous bit.
If same, data = 1. If different, data = 0.

What logic gate has similar behavior?

a. NAND
b. XOR
c. XNOR
d. INV

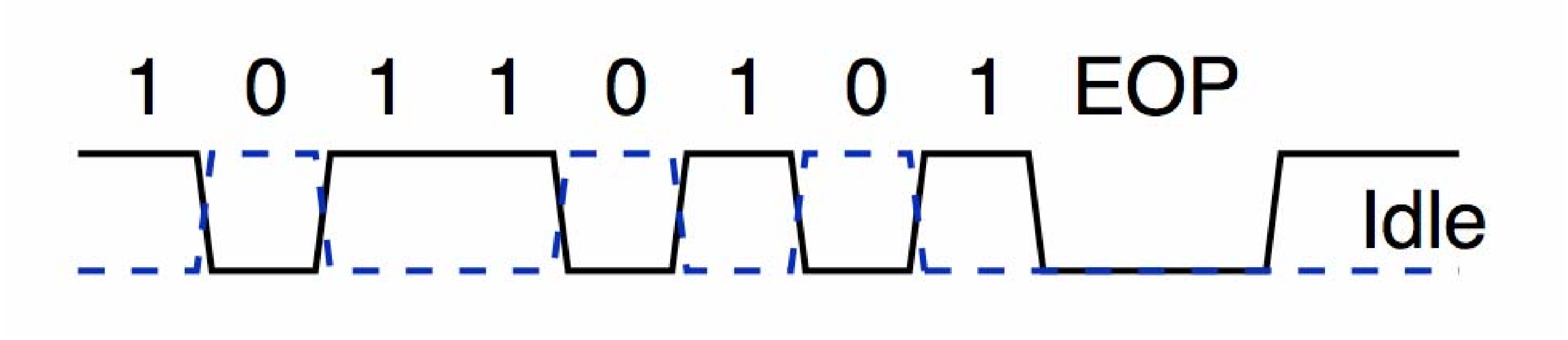
Sync Byte (encoded)





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

End of Packet (EOP) Signal



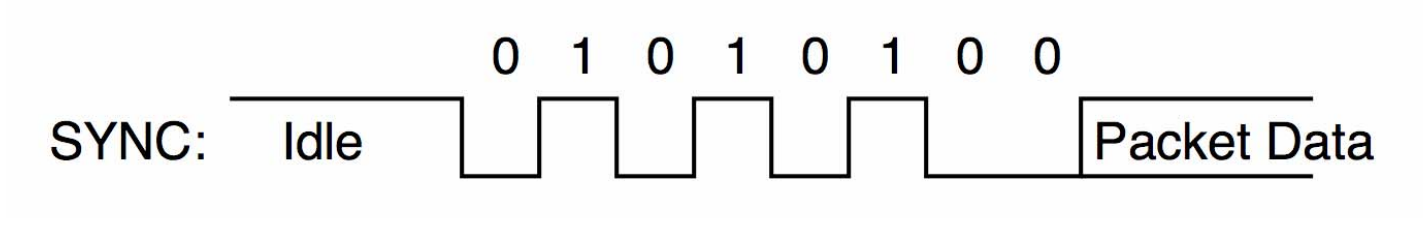
USB uses two data lines

- normally one is the complement of the other
- eliminates common mode noise

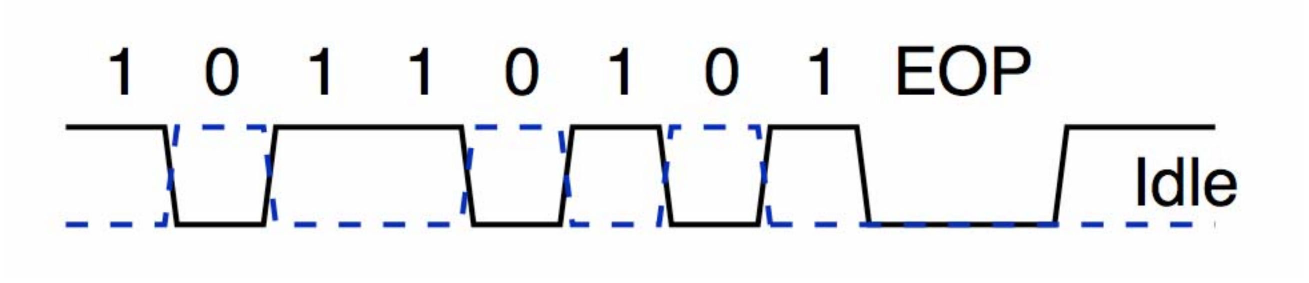
Both data lines are pulled low to signal EOP
Can be detected by a single logic gate

Complete Packets

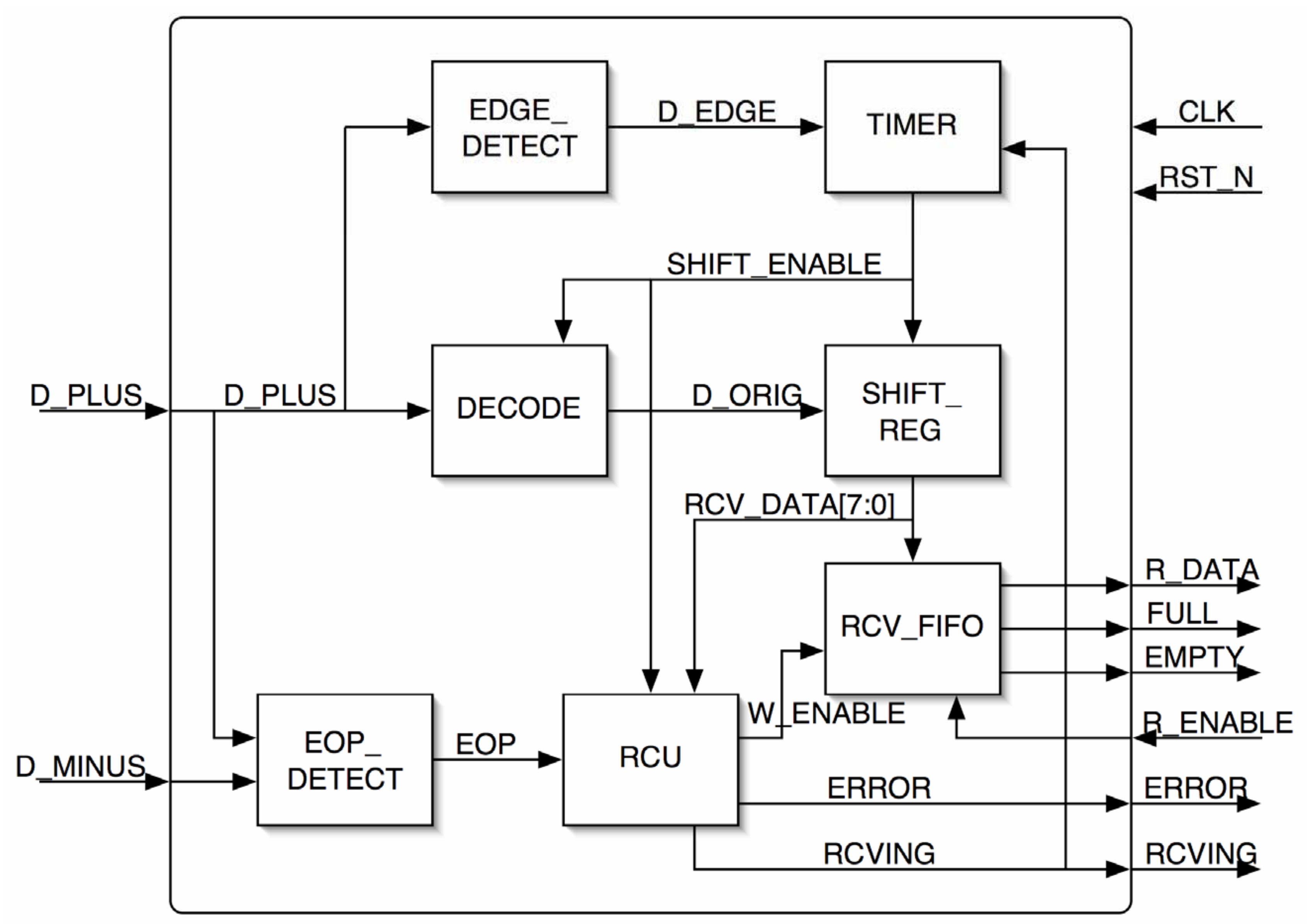
- Begins with SYNC byte



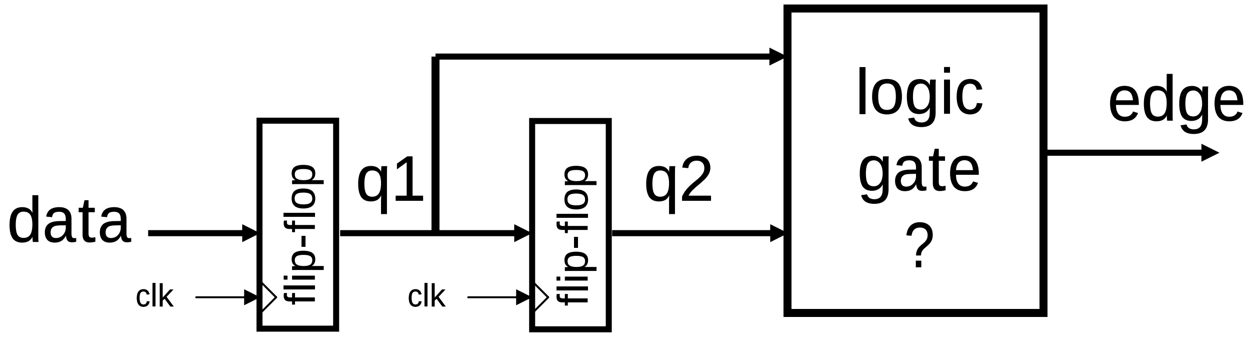
- Packet body is decoded and stored in FIFO, one byte at a time
- Ends with EOP signal



System Overview



Synchronous Edge Detection



Hint: this is useful for NRZi decoding and for clock sync.

Depending on choice of logic gate, this will generate a one clock cycle pulse for:

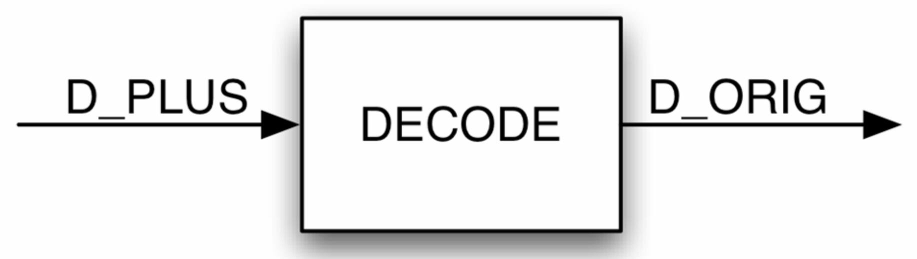
- input rising transition
- input fall transition
- either a rising or falling transition
- if there is no transition

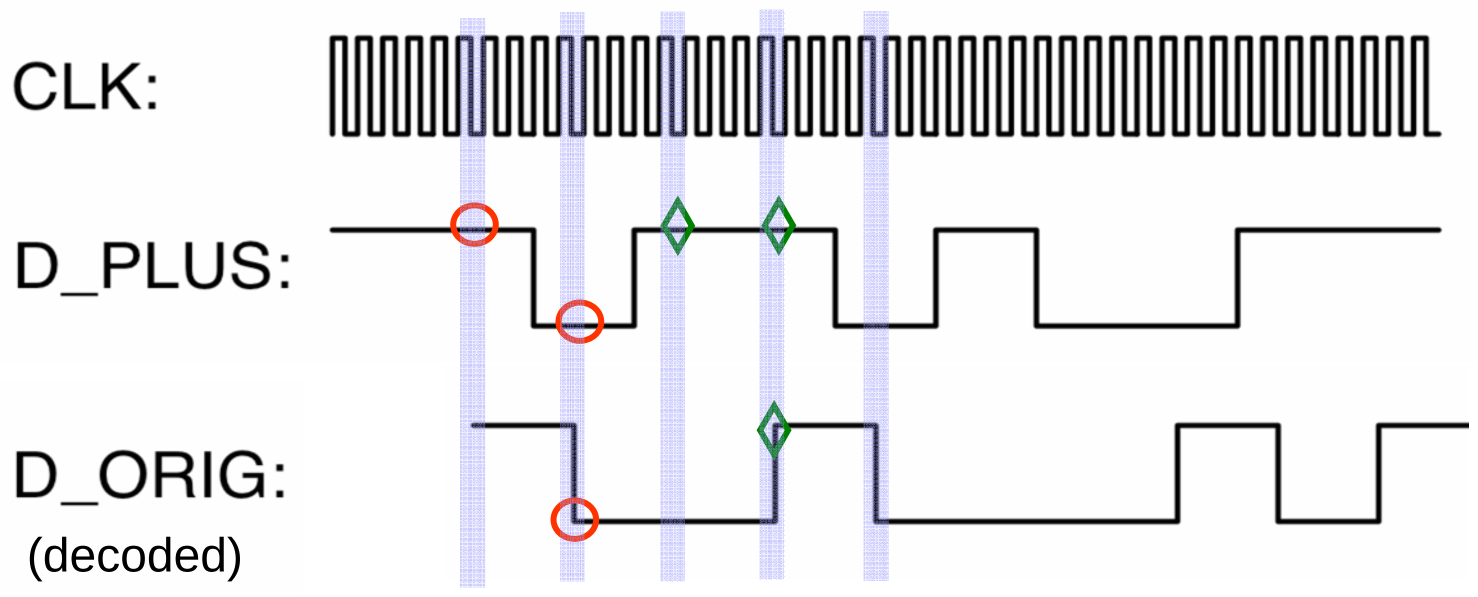
Thought question: is this circuit suitable as a synchronizer?

This assumes system clock period is shorter than time between input data transitions

DECODE

- Reverses NRZI encoding

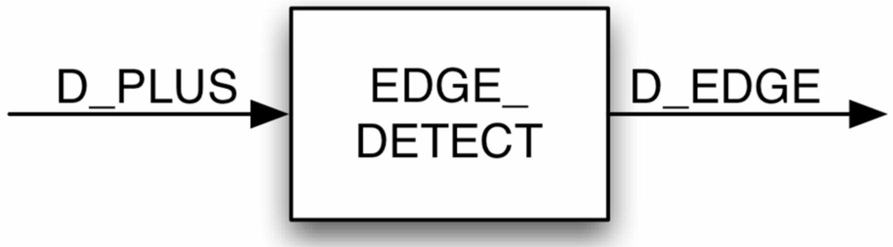




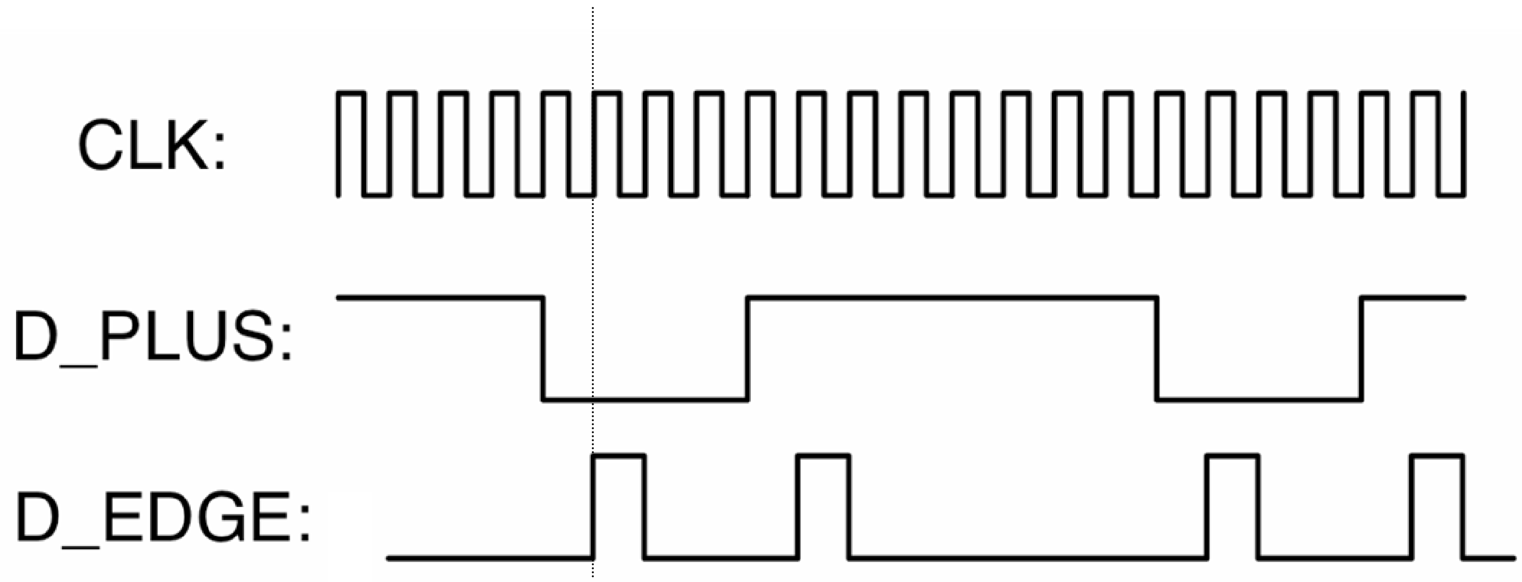
the shapes indicate examples of the correspondence between the NRZI input and the decoded output

the vertical bars indicate shift strobes which cause input data to shift to the right by one position in the decoder flip-flops

EDGE_DETECT

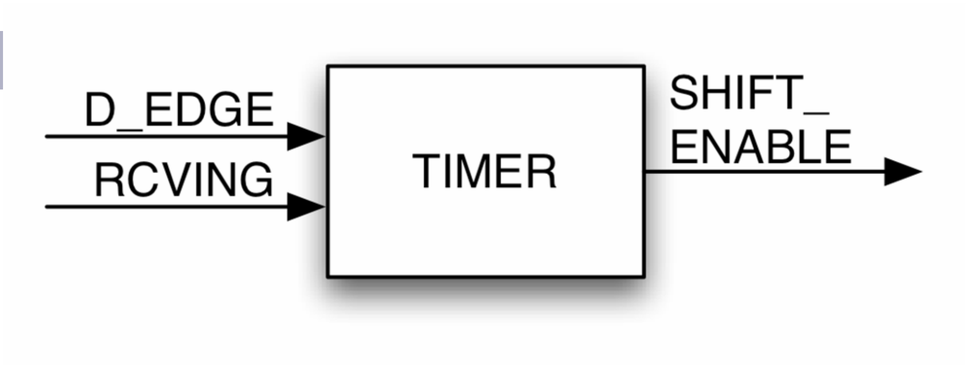


- Detects transitions on the D_PLUS input
- Synchronizes TIMER block

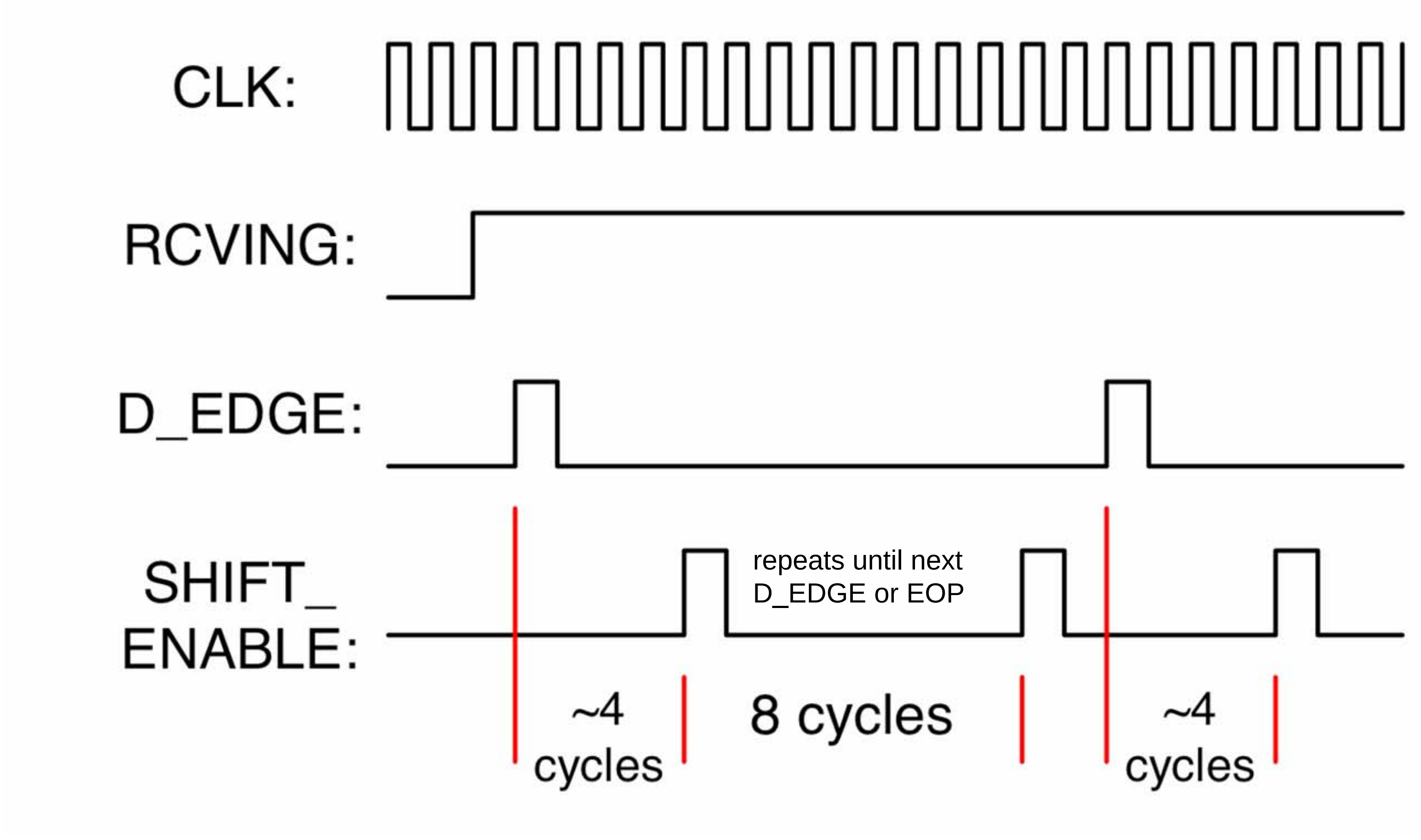


note that edge detect pulse may be delayed by one clock cycle

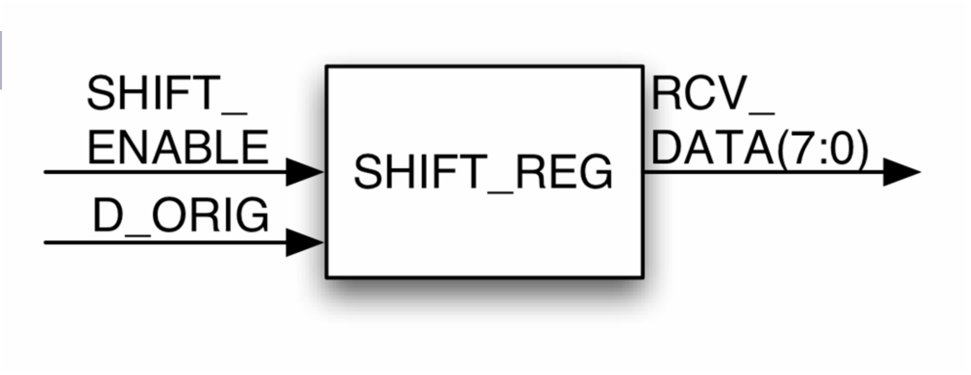
TIMER



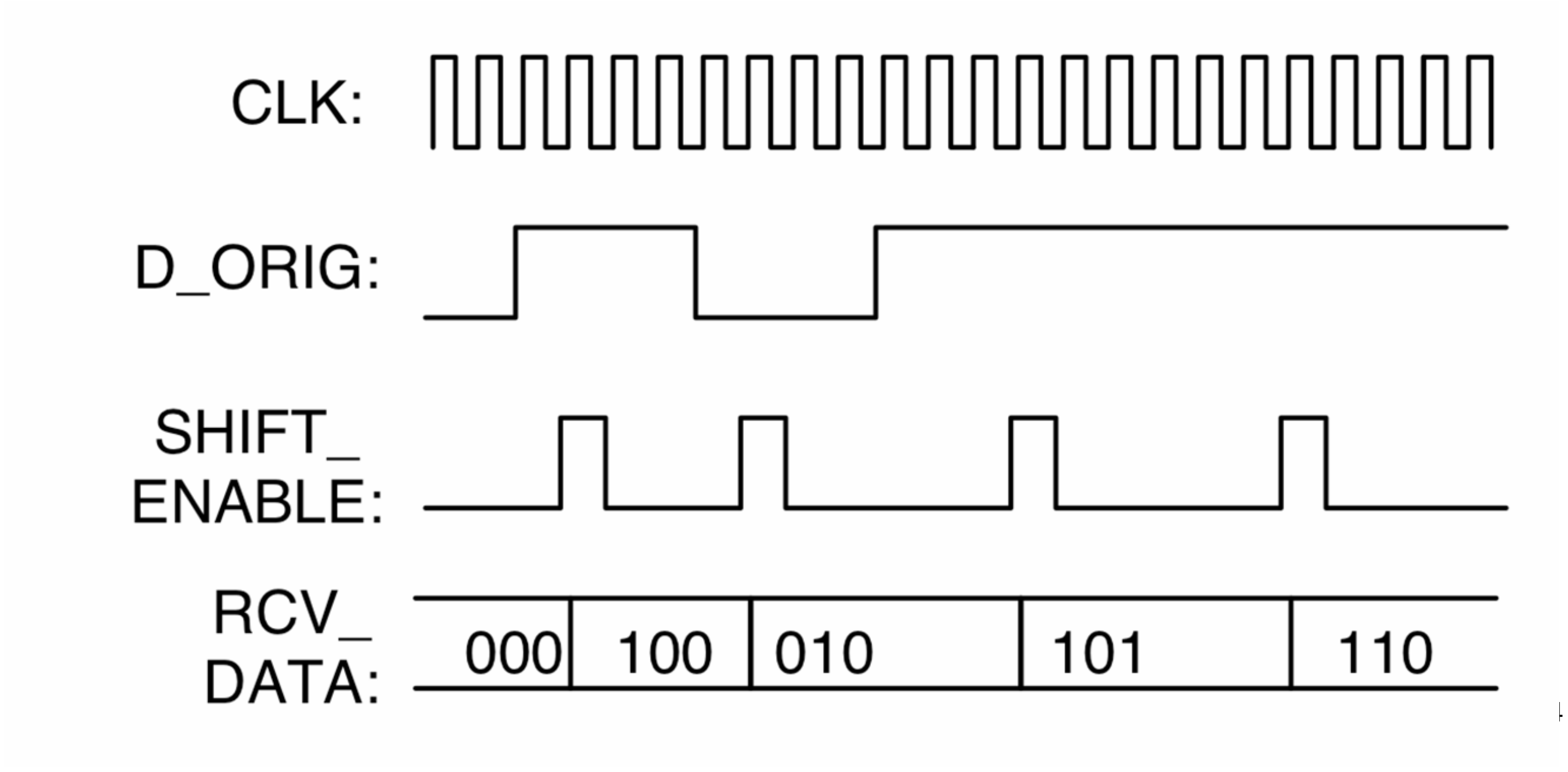
- Shifts next bit every 8 clock cycles



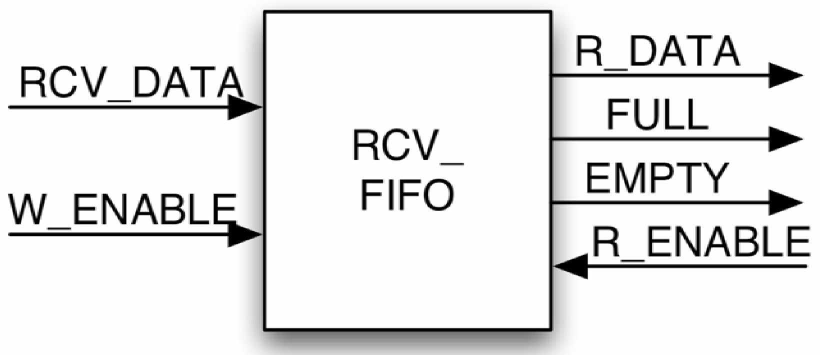
SHIFT_REG



- Shifts data in on each clock cycle when SHIFT_ENABLE is asserted.
- 3-bit shift register illustrated below, actual is 8 bit.

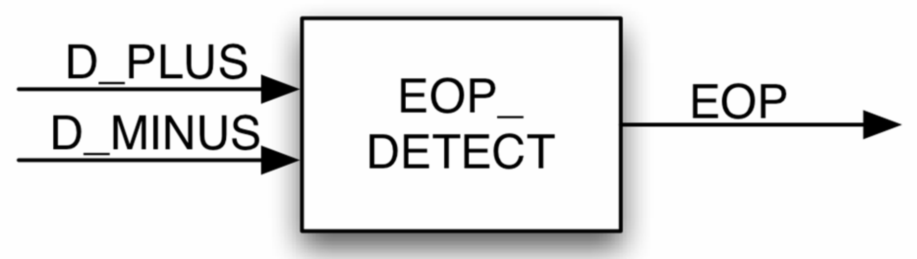


RCV_FIFO

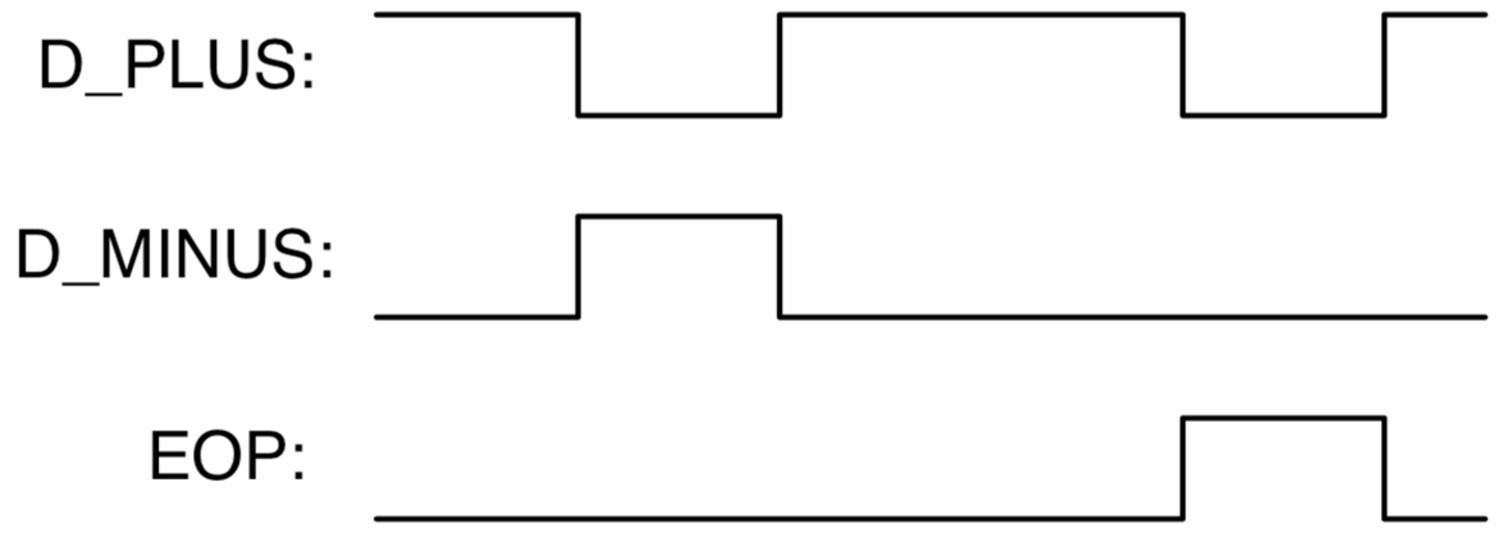


- The FIFO is **provided**, "wrapper" file.
- FIFO writes data from RCV_DATA when W_ENABLE is high on a clock edge.
- FIFO outputs data on R_DATA, increments to next byte when R_ENABLE is high.
- FULL and EMPTY indicate FIFO status
- Provided FIFO has two clocks
  - Why? This FIFO was designed to provided interface between circuits running on different clocks
  - you will tie them to same system clock

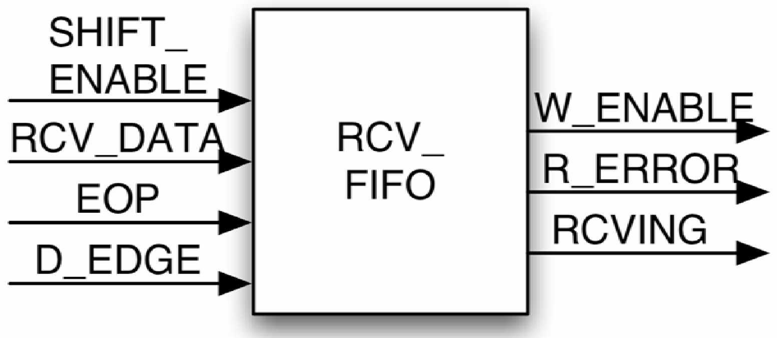
EOP_DETECT



- High when D_PLUS and D_MINUS are both low.
- Ridiculously easy



RCU



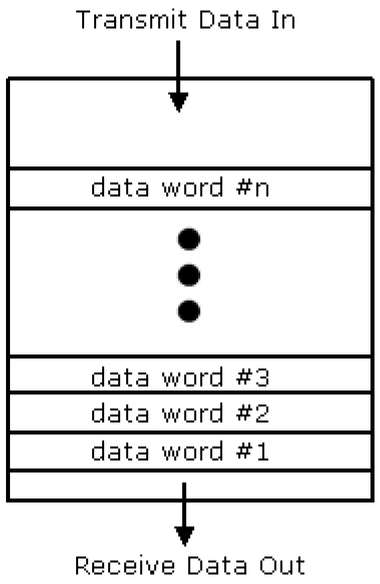
- Controls USB Receiver circuit
- Begins receiving when an edge is detected.
- Checks to see if the SYNC byte was received, error condition if not.
- Stop receiving when the EOP is detected.

Sequence of Operation

- On reset, the RCU goes to the idle (waiting) state.
- When an edge is first detected,
  - begin receiving data through the DECODE block to the SHIFT_REG.
  - Set the RCVING line high.
- 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.
- 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.

FIFO RAM

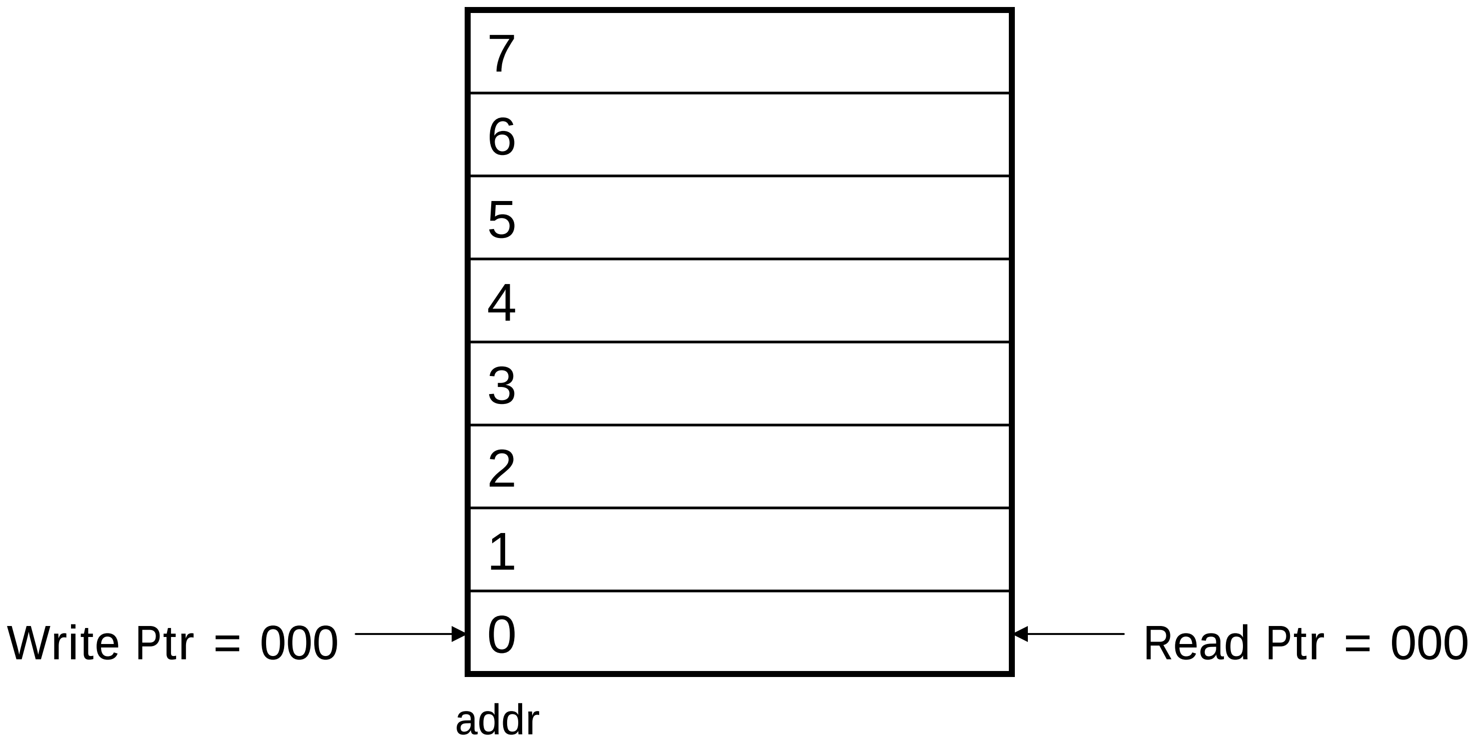
FIFO Design (provided)



Why can't this be done with a simple shift register?

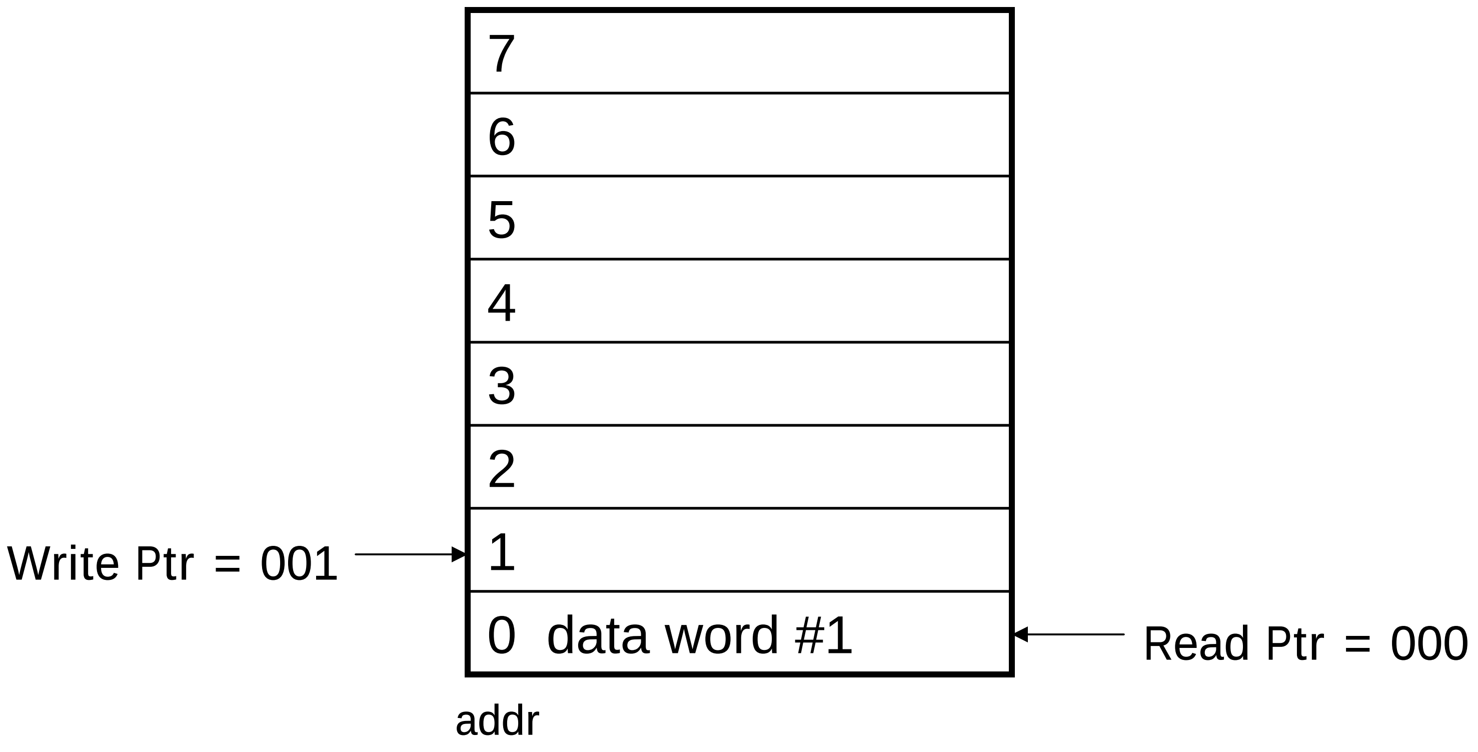
- Purpose:
  - Efficiently handle data transfer between two systems
  - Allows source & receiver of data to operate at different speeds (& clock rates)
- Some Requirements:
  - Data must be read out in same order as written in
  - Can't assume synchronization between input and output
  - Next word in FIFO must be immediately available to read.

FIFO Operation



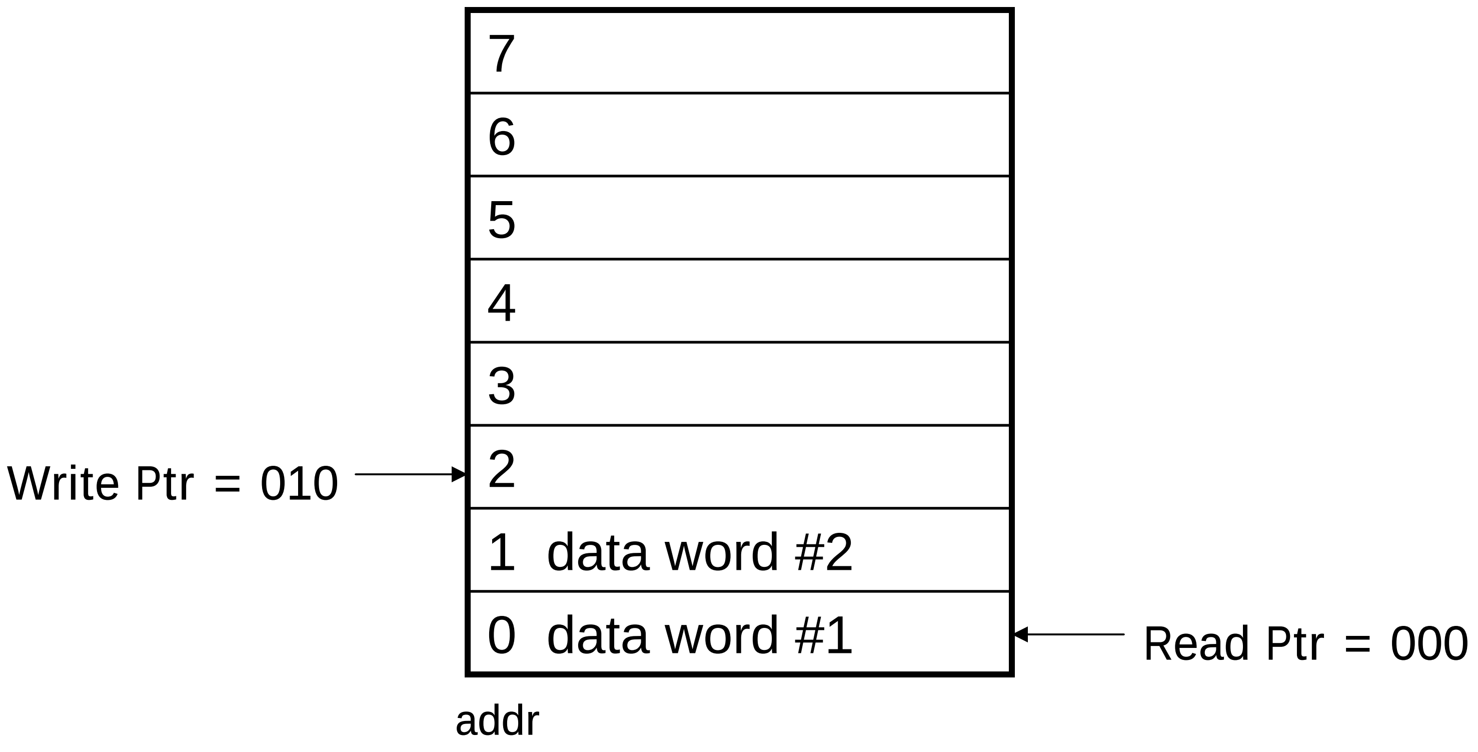
Prior to writing anything
FIFO is empty, Write Ptr = Read Ptr

FIFO Operation



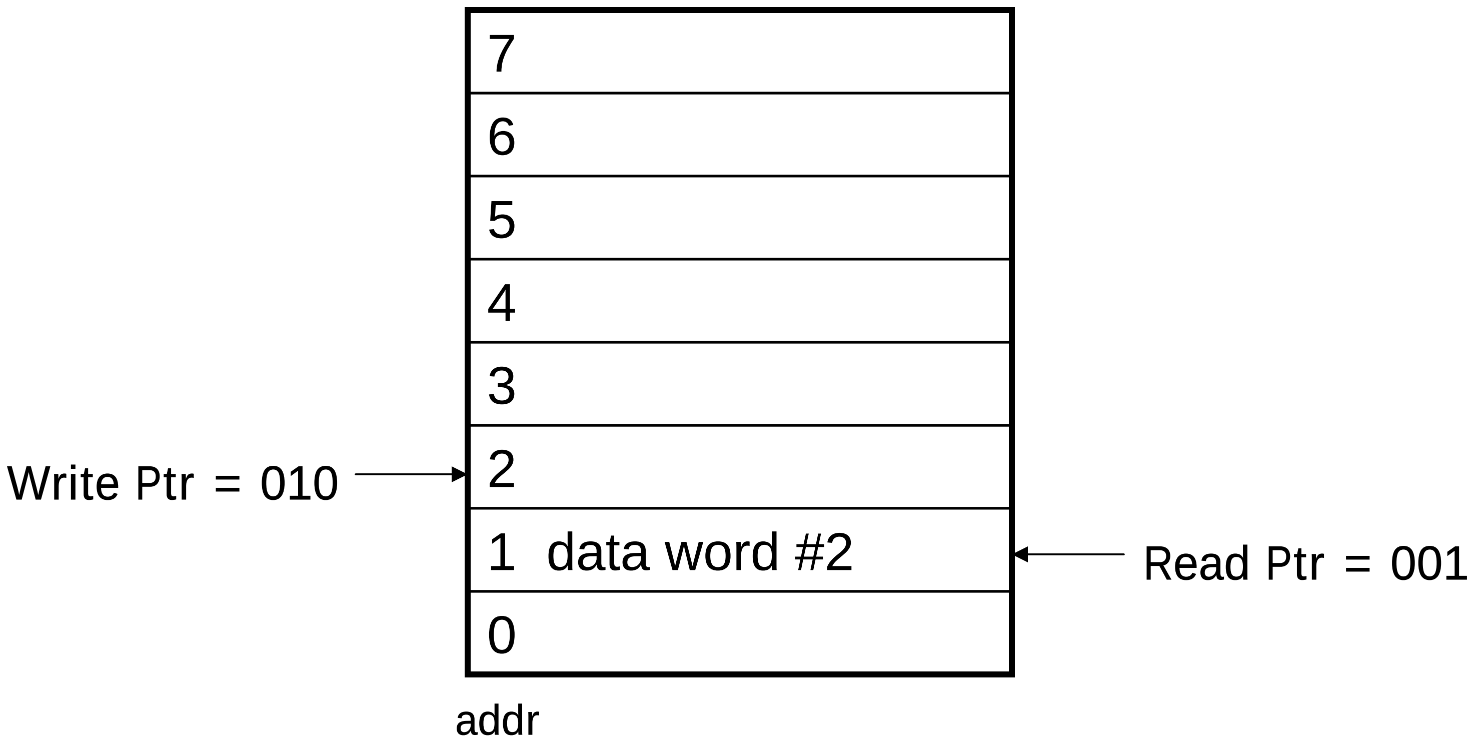
After 1st word written

FIFO Operation



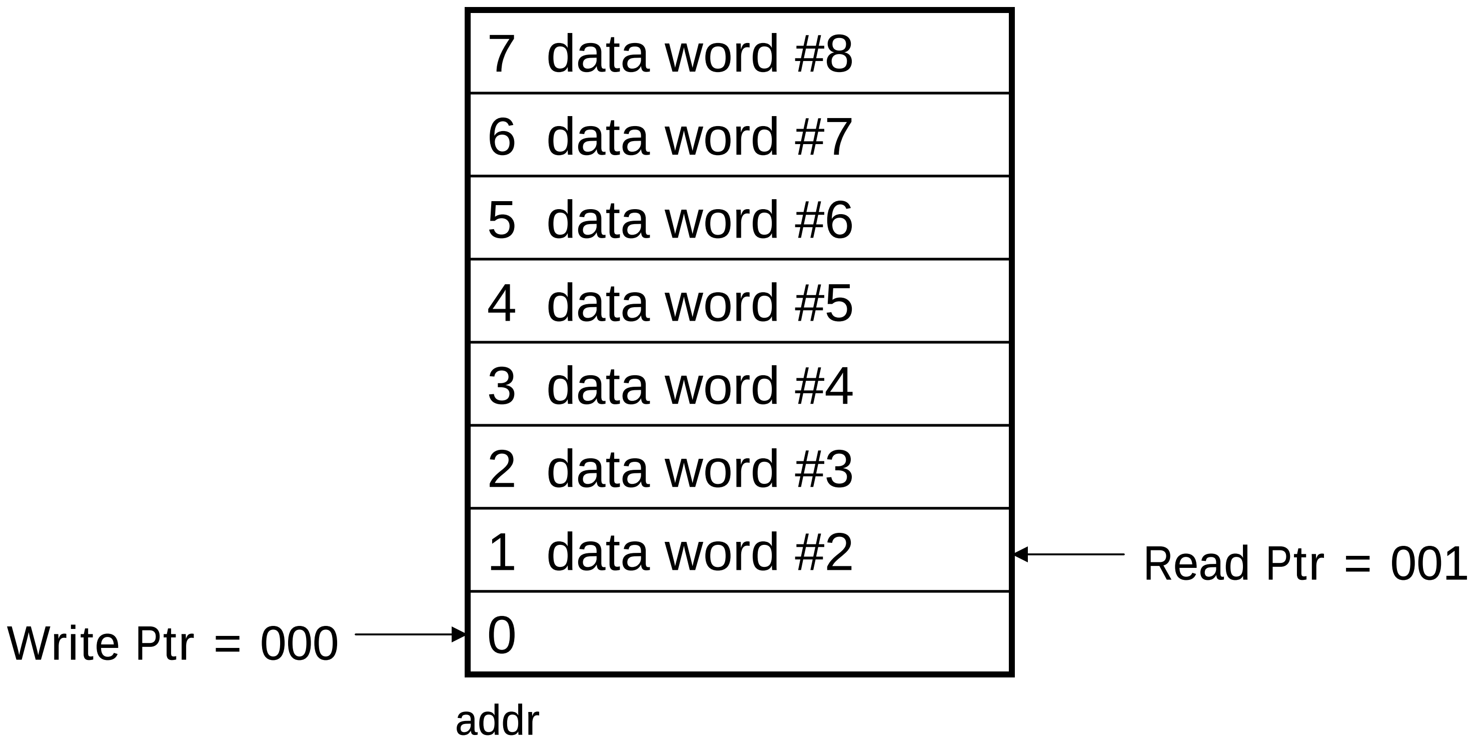
After 2nd word written

FIFO Operation



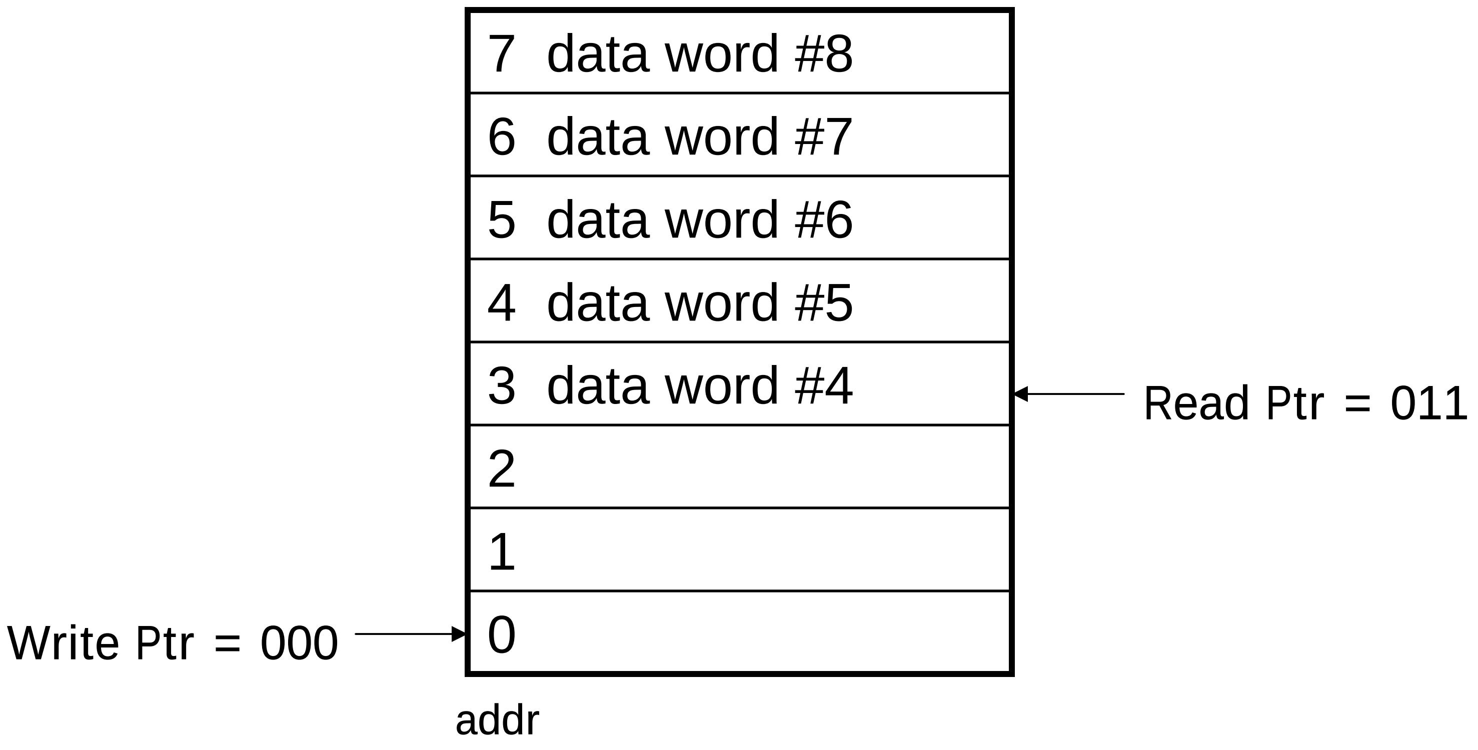
After 1st word read

FIFO Operation



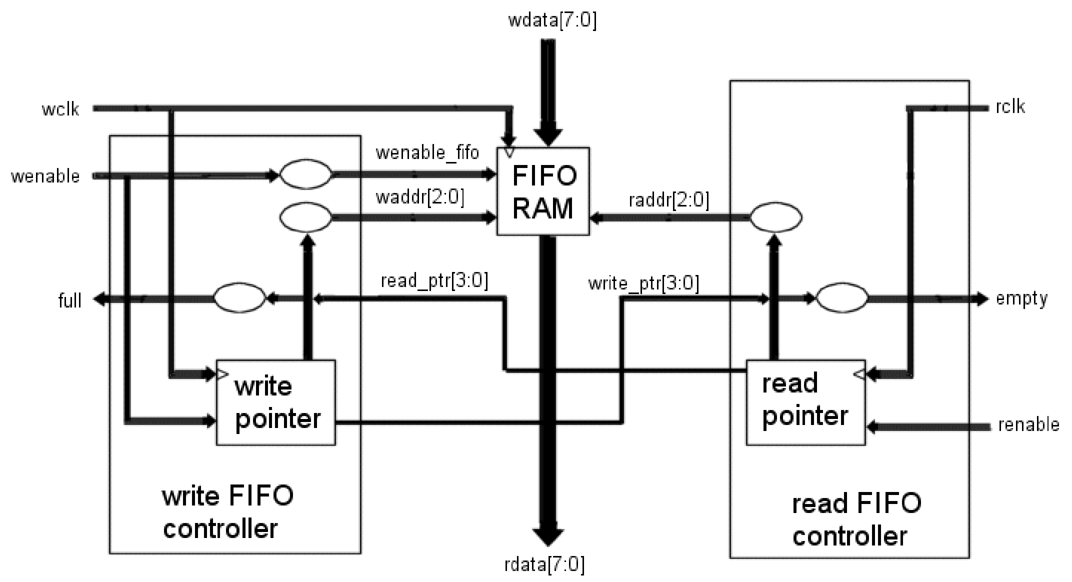
After 7 more words written
Notice write pointer wrapped around

FIFO Operation



After 2 more words read

FIFO Implementation



Write controller: Checks for full condition. If not full, writes data & increments write pointer.

FIFO RAM: A set of 8 registers, 8 bits each, with addressing logic

Read controller: Checks for empty condition. If not full, increments the read pointer.