ECE243

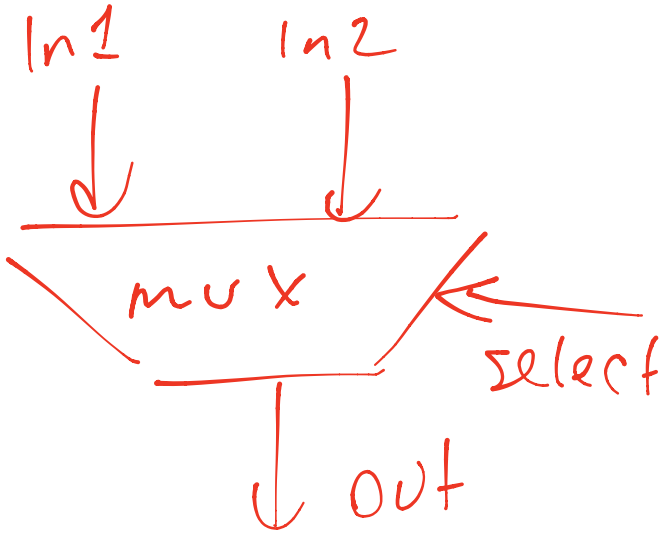
I/O Hardware

Prof. Enright Jerger

Basic Components

MULTIPLEXER

select	out
0	In 1
1	In 2

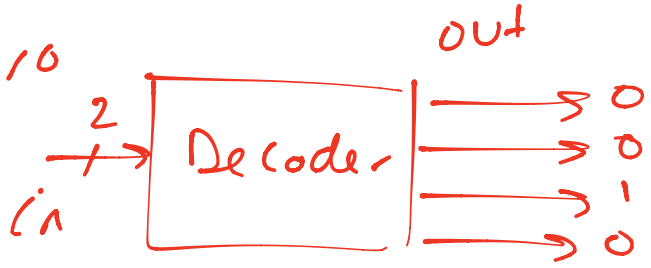


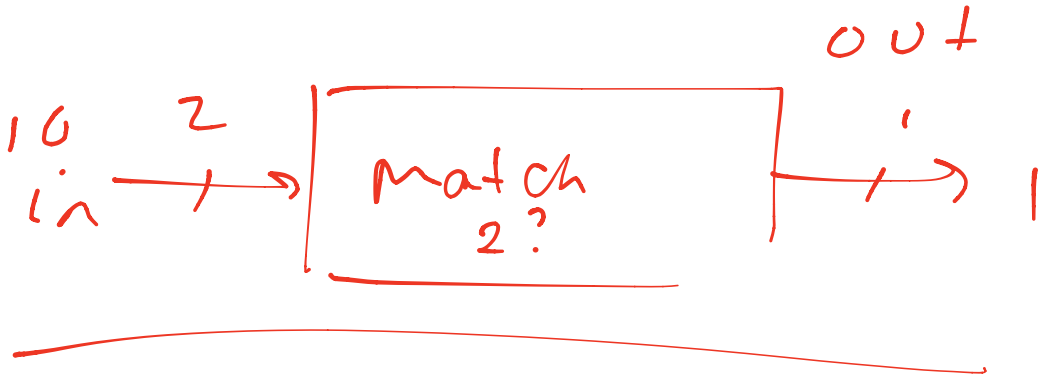
DECODER

- Example: a 2->4 decoder

In(1)	In(0)	Out(3)	Out(2)	Out(1)	Out(0)
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Can be used to match a specific value: eg., in==2?

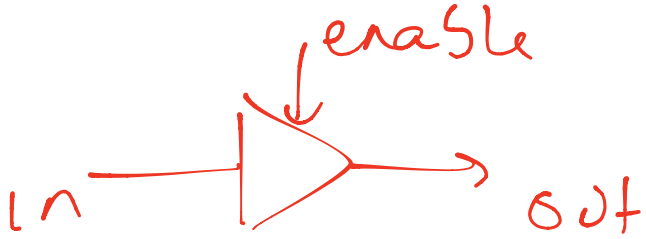




TRI STATE INTERFACE

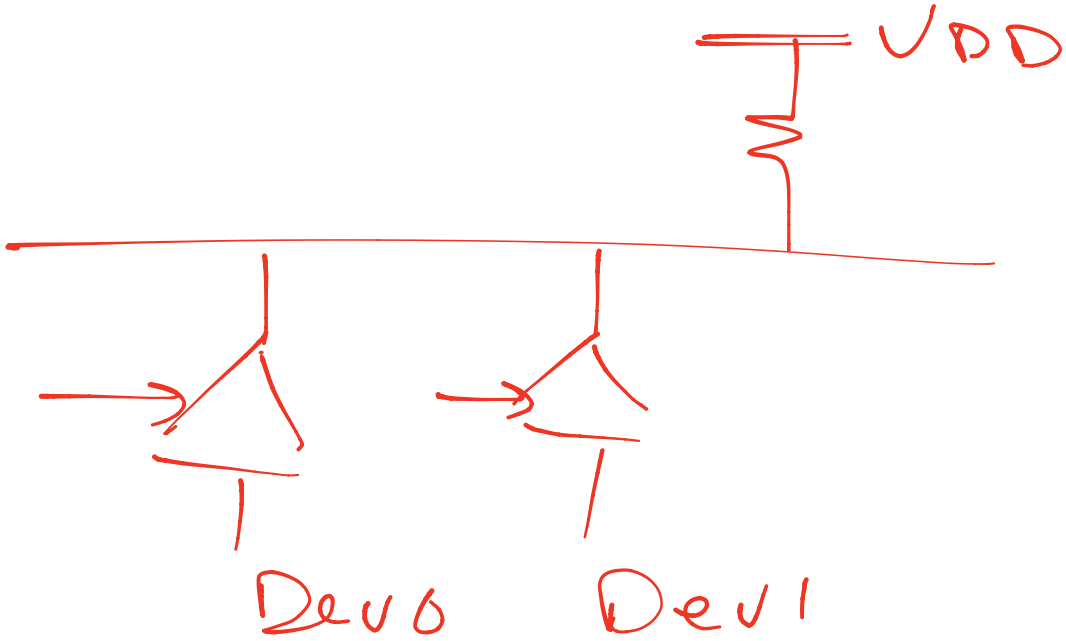
- aka tri-state buffer
- used for attaching to shared wires
  - eg a bus
- Z = “high impedence”
  - ie no impact on outgoing wire

Enable	In	Out
1	0	0
1	1	1
0	0/1	Z



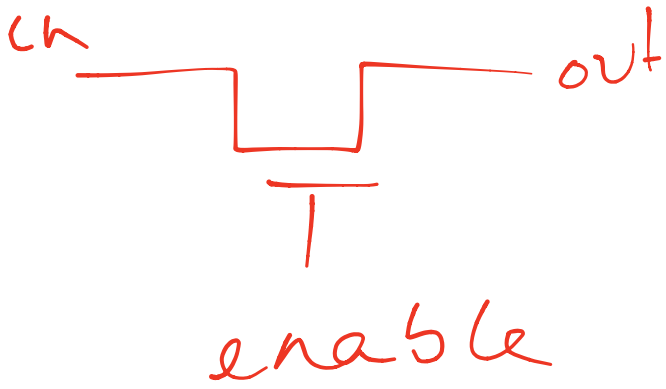
PULL DOWN LINE

- normally made with a pull-up resistor
- resistor connected to power
  - pulls the line ‘up’ to 1 by default
- devices can pull the line ‘down’ to 0



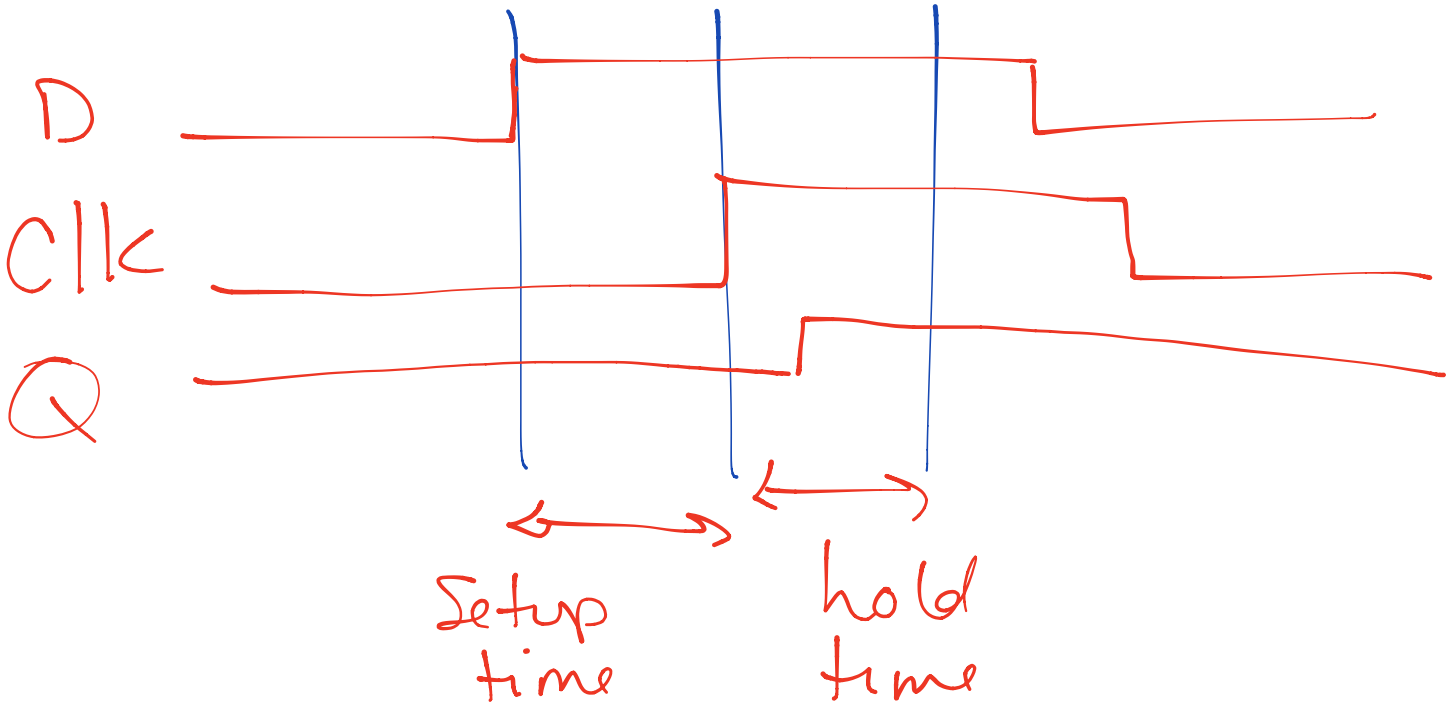
PASS TRANSISTOR

Enable	In	Out
1	0	0
1	1	1
0	0/1	Z



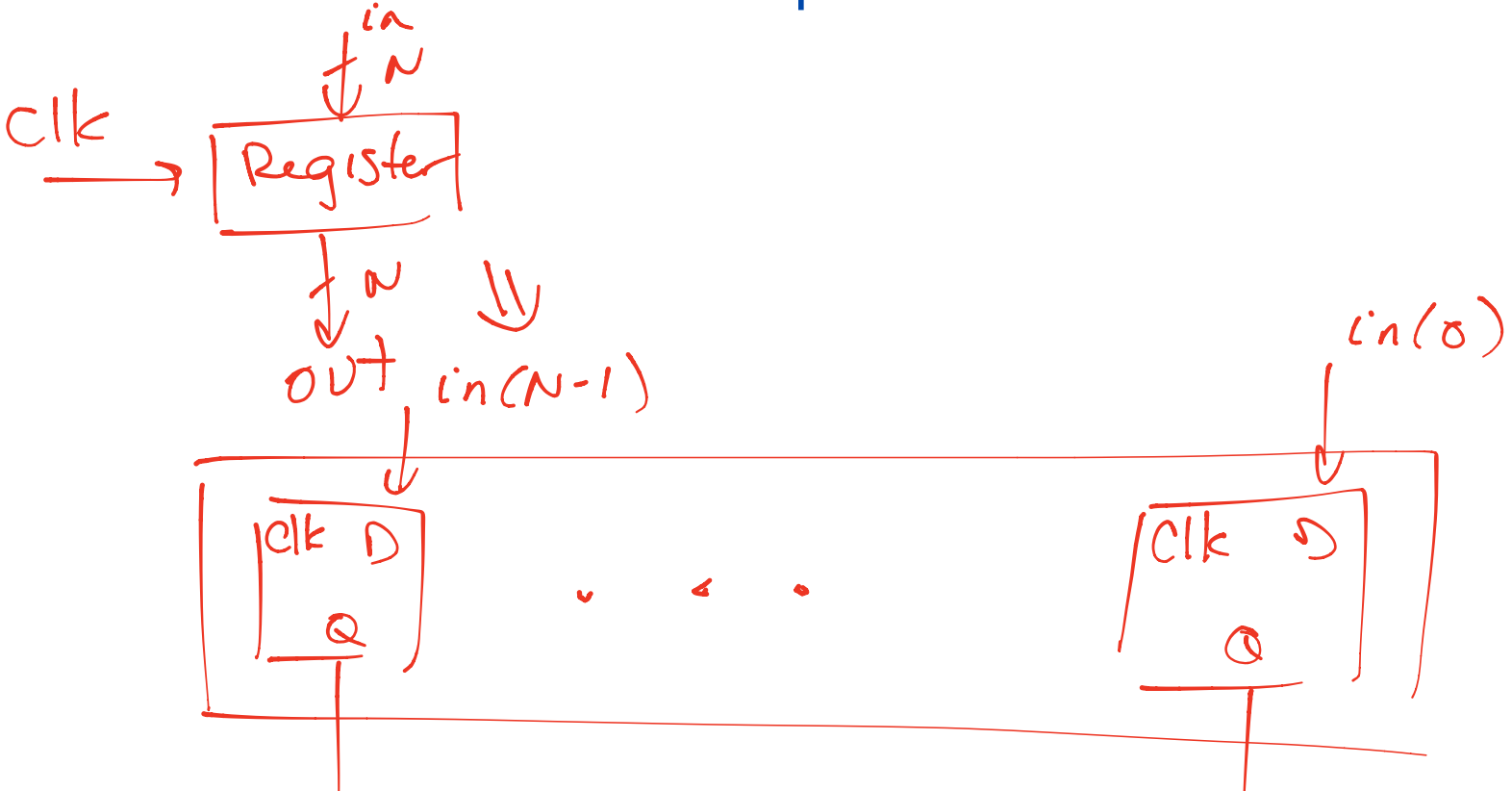
D/Q Flip Flop

- eg., rising edge triggered
- Like posing for a picture
  - Set-up time
    - say cheese and hold the pose
  - Hold-time
    - like taking a long exposure shot at night
- Q is set to D's value
  - when clk goes from low to high
- D must be stable for setup-time seconds
  - before the clock edge
- D must remain stable for hold-time seconds
  - after clock edge



REGISTER

- stores an N-bit value
- is composed of N flip flops
- value is read/written in parallel

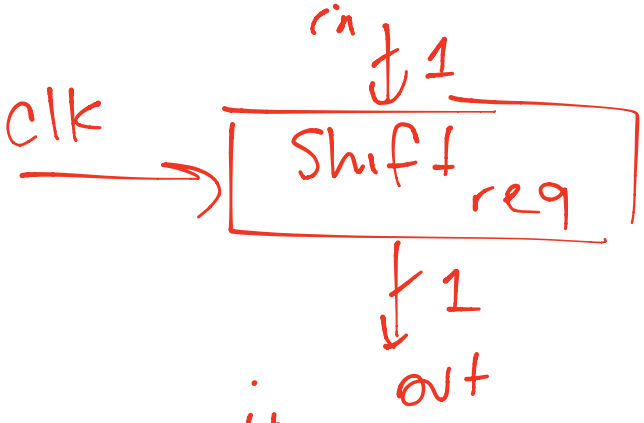


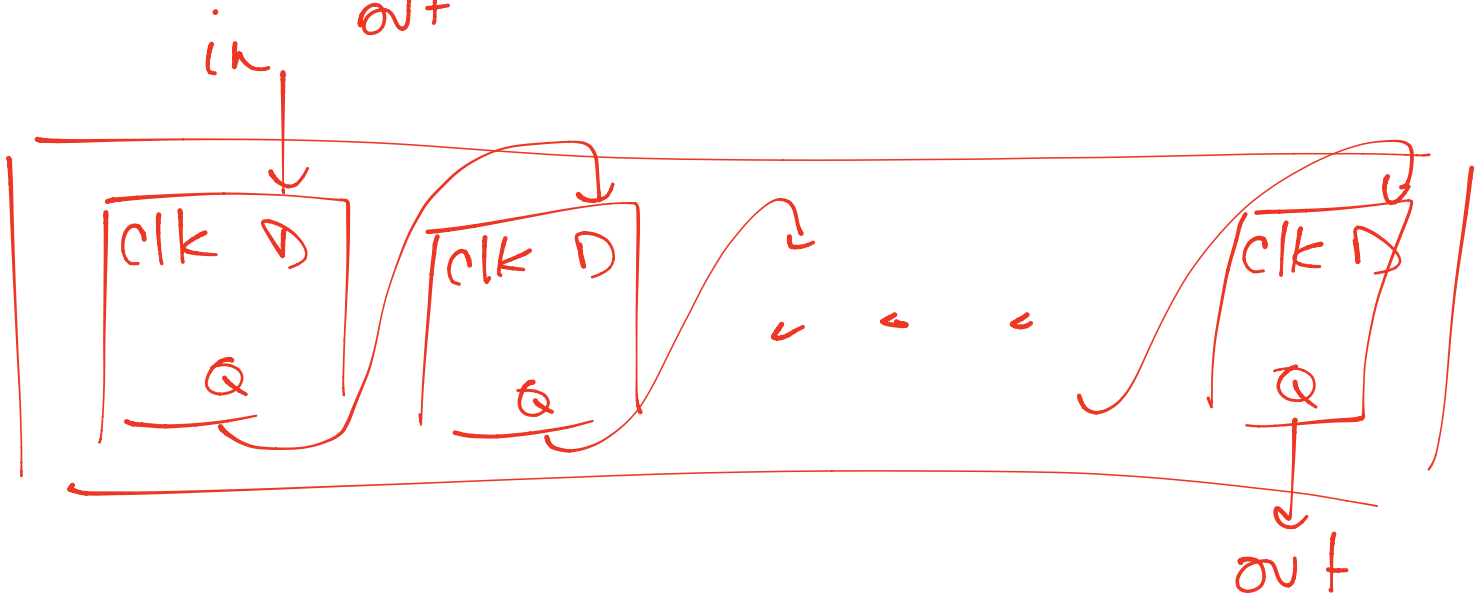
out(N-1)

out(0)

SHIFT REGISTER

- stores an N-bit value
- composed of N flip flops
- value is read/written 1-bit at a time (serial)

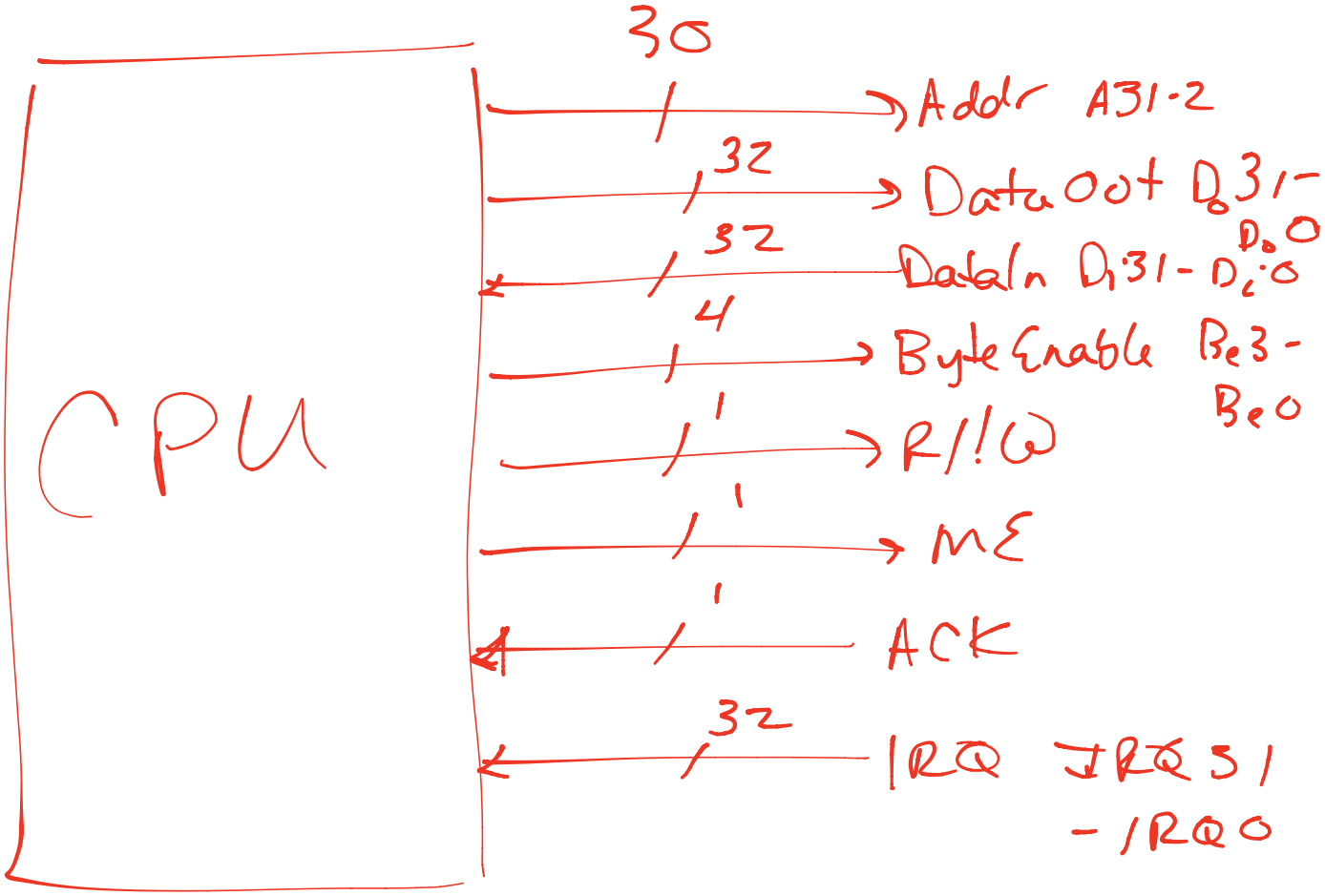




I/O Implementation

NIOS Bus

- addr: only upper 30 bits: A31-A2 (missing A1-A0)
- byte enable: four wires, be3-be0
  - encodes two things: A1,A0 and word/halfword/byte
  - each wire indicates whether that byte is valid
- ME: master enable: one wire
  - do nothing if zero (avoid interpreting transient values)
- Ack: device sets this to one to ack processor request
- IRQ: set to one to request an interrupt



BYTE-ENABLE Examples

	a31-a2	a1-a0	be3-be0	di31-di0
Ldw	0b100001	00	1111	di:31-di:0
Ldb	0b100001	00	0001	di:7-di:0
Ldb	0b101011	01	0010	di:15-di:8
Ldb	0b100101	10	0100	di:23-di:16
Ldb	0b101001	11	1000	di:31-di:24
Ldh	0b110101	00	0011	di:15-di:0
Ldh	0b110001	10	1100	di:31-di:16

STEPS for a LOAD (protocol):

1) CPU:
   - set addr, byte-enable, R/!W to 1;
   - then set ME to 1
2) dev/mem:
   - set DataIn to value
   - set ACK to 1
3) CPU
   - read DataIn lines
   - set ME to 0
4) dev/mem:
   - set ACK to 0

STEPS for a store

1) CPU:
   - set addr, byte-enable, DataOut, R/!W to 0;
   - then set ME to 1
2) dev/mem:
   - use DataOut values to update state
   - set ACK to 1
3) CPU
   - set ME to 0
4) dev/mem:
   - set ACK to 0

IMAGINARY I/O DEVICE

.equ MYDEVICE, 0xffabc0

0(MYDEVICE): 8bit input register

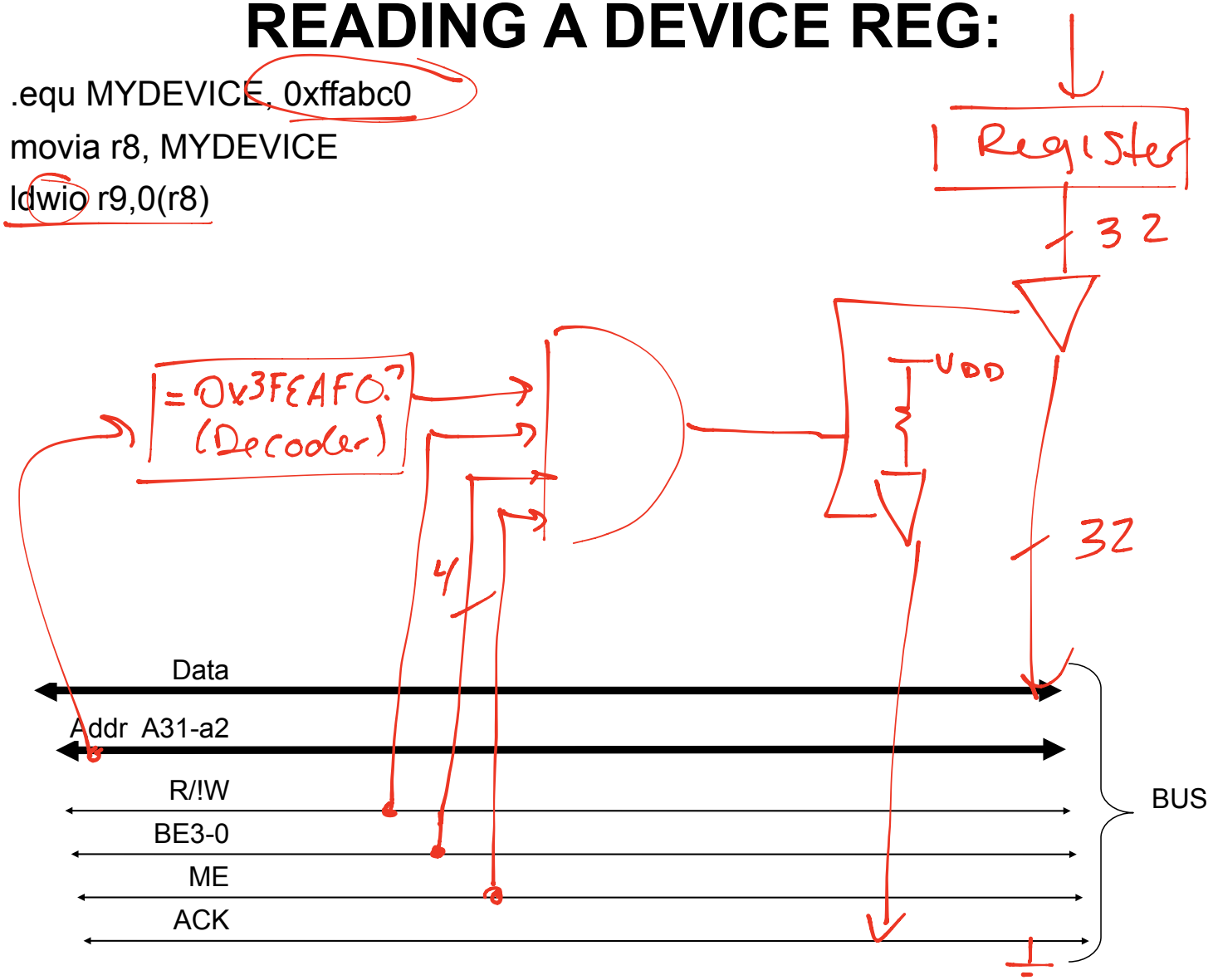
4(MYDEVICE): 8bit output register

Note: 0xffabc0 >> 2 = 0x3feaf0 (dropping A1, A0)

value on A31-A2

READING A DEVICE REG:

```
.equ MYDEVICE, 0xffabc0
movia r8, MYDEVICE
ldwio r9,0(r8)
```

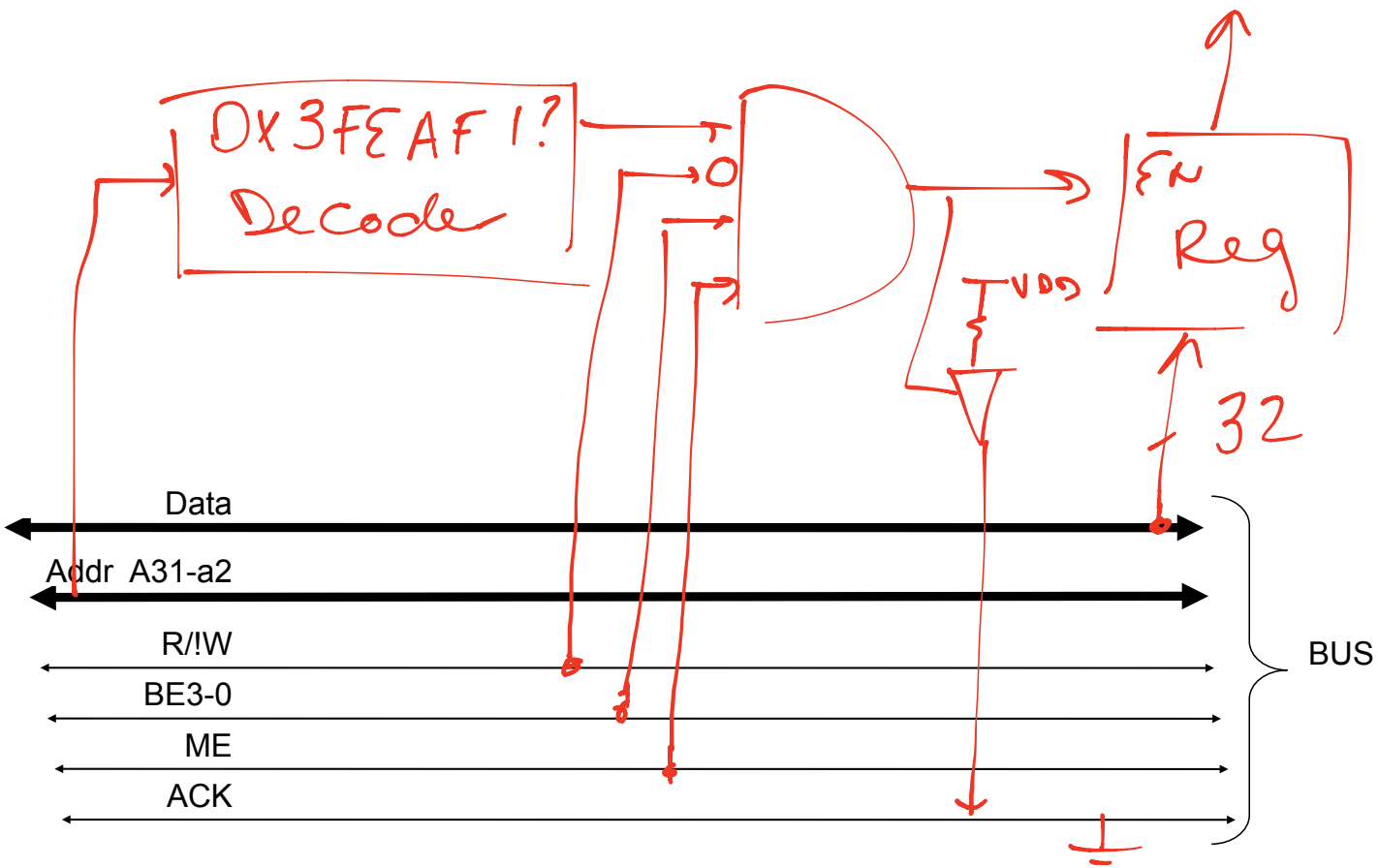


WRITING A DEVICE REG:

equ MYDEVICE, 0xffabc0

movia r8, MYDEVICE

stwio r9,4(r8)



PARALLEL INTERFACE

- **RECALL:**

.equ JP1, 0x10000060

0(JP1): data in/out DIO (~~8~~ 32 bits)

4(JP1): data direction register DIR, each bit configures data pin as in or out (~~8~~ 32 bits)

0 means input, 1 means out

0x10000060 >> 2 = 0x4000018

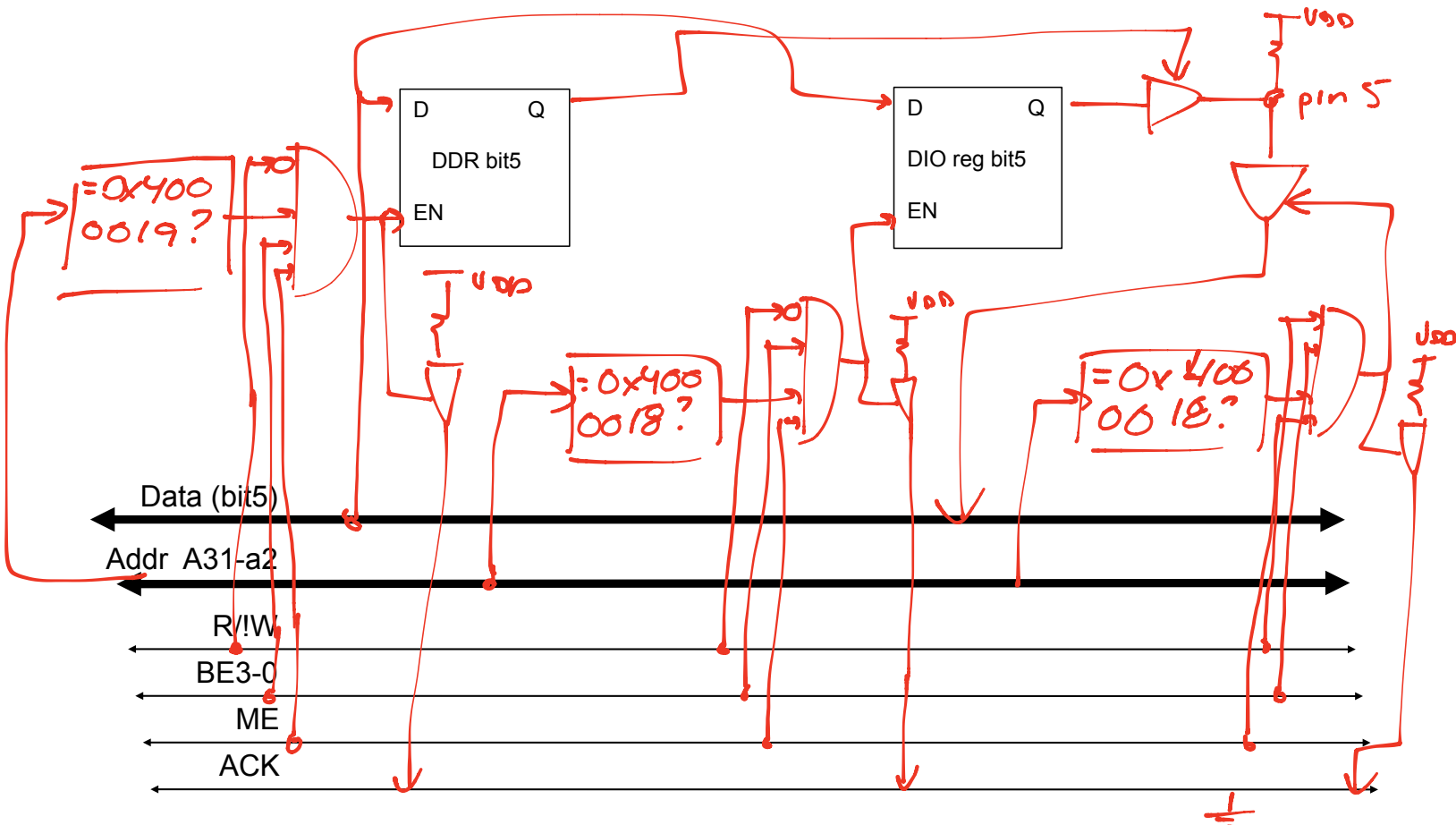
0x10000064 >> 2 = 0x4000019

PARALLEL INTERFACE

JP1:

0(0x10000060): DIO # 0x10000060 >> 2 = 0x4000018

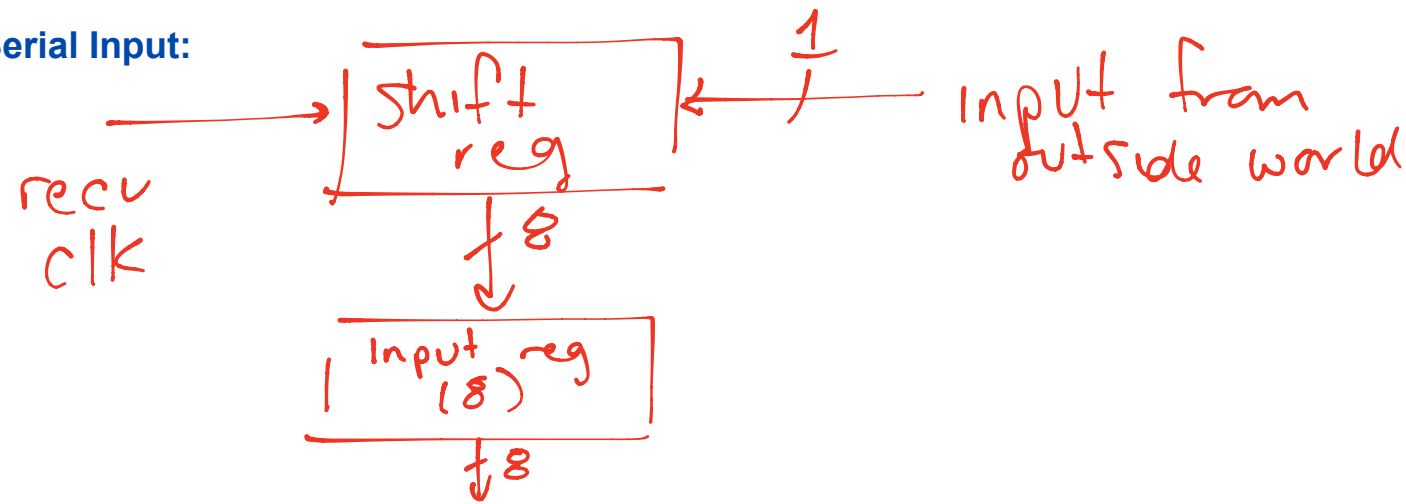
4(0x10000060): DIR # 0x10000060 >> 2 = 0x4000019



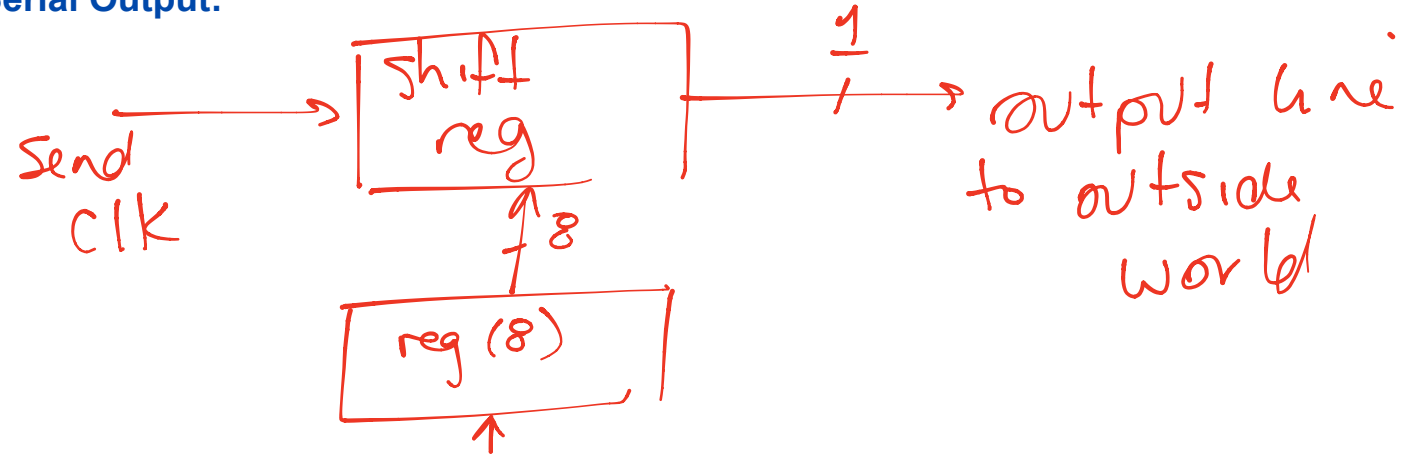
Serial Interface

- Problem: single pin, but read/write bytes
- Solution: use shift registers

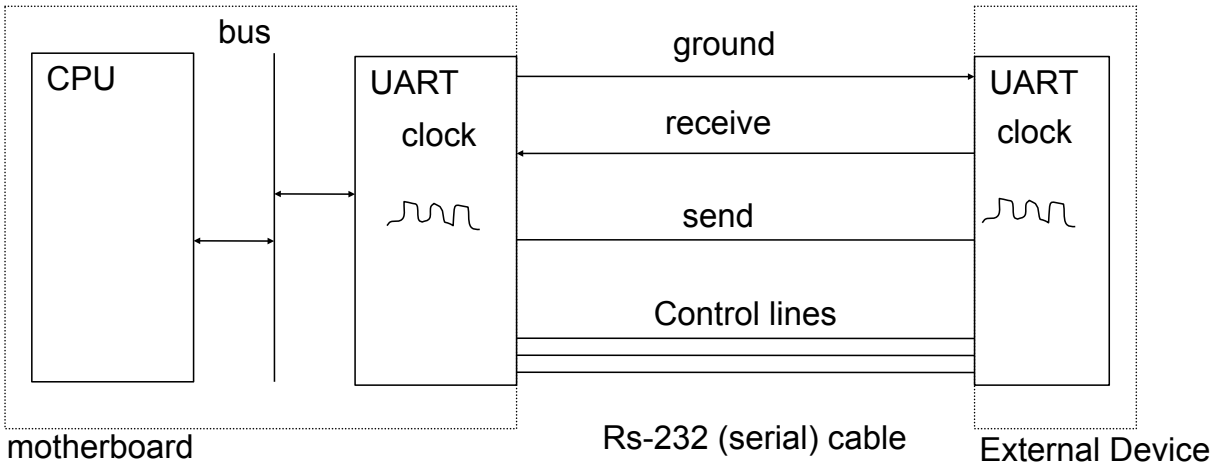
Serial Input:



Serial Output:



Serial Interface: RS232



SYNCHRONIZATION:

- each side has its own clock
- this causes problems:
  - clocks may not be exactly same speed

    - ie., there is a frequency difference
  - clocks may be out-of-sync
    - ie., there is a phase difference
- hardware has to handle these difficulties

SERIAL TRANSMISSION

1. Both sides agree on a configuration:
   - baud rate (bits per second)
   - number of bits per group (7 or 8)
   - is the MS-bit a parity bit? (odd or even)
   - number of stop bits (1, 2…)
     - normally zero's
2. Can then send frames of bits
   - start bit: normally a zero
   - frame: 1 start bit + bit group + stop bits
   - expect a new bit every period
     - period = 1 / baud-rate

PARITY:

- Even parity:
  - the number of bits that are one is even
- Odd parity:
  - the number of bits that are one is odd
- Parity bit:
  - a bit that is added to ensure even or odd parity
  - typically the MSbit

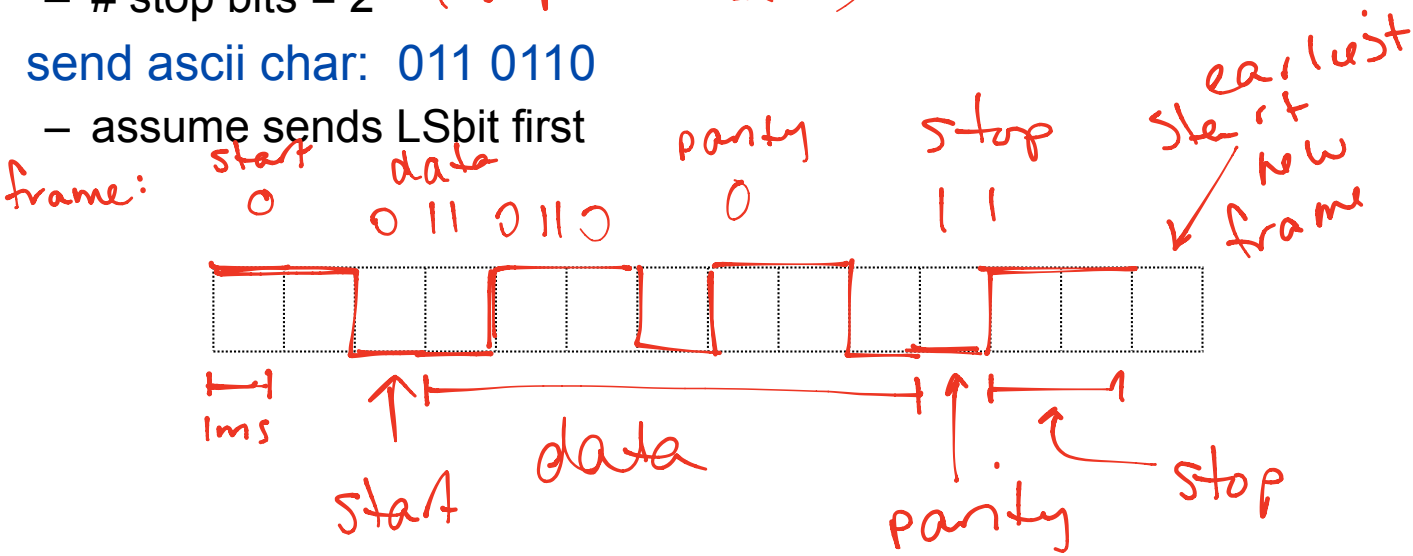
- Ex: what is the parity bit for _1001110
  - even parity: 0
  - odd parity: 1

- Parity bit can be used to detect errors
  - Eg., if expecting even parity and get odd parity

EXAMPLE CONFIGURATION:

- Baud rate: 1Kbaud = 1000 bits/s
- #bits = 8
- parity? = yes, even
- # stop bits = 2 (stop bits are 1)

- send ascii char: 011 0110
  - assume sends LSbit first



EFFECTIVE DATA RATE

- of 1k bits/s, some are wasted:
  - start bits, stop bits, parity bits
- Effective data rate:
  - the bit rate of non-wasted bits

- Ex: what is the effective data rate from previous example?

frac of useful bits = useful bits / total bits

7/(1+7+1+2) = 7/11

7/11 kbits/sec