

# Basics of digital systems

Notes by Annie Guo

1

## Overview

- Basics of computing with digital systems
  - Hardware fundamentals
    - Logic gates
    - Functional blocks
    - Processor structures

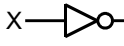
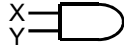

2

## Logic gates

- Virtually all problems can be solved by digital circuits and systems
- The basic elements of digital circuits are logic gates
- Logic gates
  - ideally have signals of two levels: high and low
  - perform logic functions, such as NOT, AND, OR, NAND, NOR
- Logic gates can be represented by symbols and their functions can be described using truth tables.

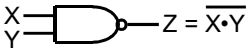
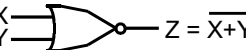
3

## NOT, AND & OR gates

	<i>Symbol</i>	<i>Truth Table</i>																															
– NOT	$X \rightarrow \text{NOT} \rightarrow Z = \overline{X}$ 	<table border="1" style="border-collapse: collapse; width: 40px; height: 40px;"> <tr><th>X</th><th>Z</th></tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </table>	X	Z	0	1	1	0	<table border="1" style="border-collapse: collapse; width: 40px; height: 40px;"> <tr><th>X</th><th>Z</th></tr> <tr><td>low</td><td>high</td></tr> <tr><td>high</td><td>low</td></tr> </table>	X	Z	low	high	high	low																		
X	Z																																
0	1																																
1	0																																
X	Z																																
low	high																																
high	low																																
– AND	$X, Y \rightarrow \text{AND} \rightarrow Z = X \cdot Y$ 	<table border="1" style="border-collapse: collapse; width: 60px; height: 60px;"> <tr><th>X</th><th>Y</th><th>X·Y</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	X	Y	X·Y	0	0	0	0	1	0	1	0	0	1	1	1	<table border="1" style="border-collapse: collapse; width: 60px; height: 60px;"> <tr><th>X</th><th>Y</th><th>X·Y</th></tr> <tr><td>low</td><td>low</td><td>low</td></tr> <tr><td>low</td><td>high</td><td>low</td></tr> <tr><td>high</td><td>low</td><td>low</td></tr> <tr><td>high</td><td>high</td><td>high</td></tr> </table>	X	Y	X·Y	low	low	low	low	high	low	high	low	low	high	high	high
X	Y	X·Y																															
0	0	0																															
0	1	0																															
1	0	0																															
1	1	1																															
X	Y	X·Y																															
low	low	low																															
low	high	low																															
high	low	low																															
high	high	high																															
– OR	$X, Y \rightarrow \text{OR} \rightarrow Z = X + Y$ 	<table border="1" style="border-collapse: collapse; width: 60px; height: 60px;"> <tr><th>X</th><th>Y</th><th>X+Y</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	X	Y	X+Y	0	0	0	0	1	1	1	0	1	1	1	1	<table border="1" style="border-collapse: collapse; width: 60px; height: 60px;"> <tr><th>X</th><th>Y</th><th>X+Y</th></tr> <tr><td>low</td><td>low</td><td>low</td></tr> <tr><td>low</td><td>high</td><td>high</td></tr> <tr><td>high</td><td>low</td><td>high</td></tr> <tr><td>high</td><td>high</td><td>high</td></tr> </table>	X	Y	X+Y	low	low	low	low	high	high	high	low	high	high	high	high
X	Y	X+Y																															
0	0	0																															
0	1	1																															
1	0	1																															
1	1	1																															
X	Y	X+Y																															
low	low	low																															
low	high	high																															
high	low	high																															
high	high	high																															

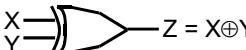
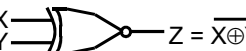
4

## NAND & NOR gates

	<i>Symbol</i>	<i>Truth Table</i>															
- NAND	 $Z = \overline{X \cdot Y}$	<table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th><math>\overline{X \cdot Y}</math></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	X	Y	$\overline{X \cdot Y}$	0	0	1	0	1	1	1	0	1	1	1	0
X	Y	$\overline{X \cdot Y}$															
0	0	1															
0	1	1															
1	0	1															
1	1	0															
- NOR	 $Z = \overline{X + Y}$	<table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th><math>\overline{X + Y}</math></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	X	Y	$\overline{X + Y}$	0	0	1	0	1	0	1	0	0	1	1	0
X	Y	$\overline{X + Y}$															
0	0	1															
0	1	0															
1	0	0															
1	1	0															

5

## XOR & XNOR gates

	<i>Symbol</i>	<i>Truth Table</i>															
- XOR	 $Z = X \oplus Y$	<table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th><math>X \oplus Y</math></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	X	Y	$X \oplus Y$	0	0	0	0	1	1	1	0	1	1	1	0
X	Y	$X \oplus Y$															
0	0	0															
0	1	1															
1	0	1															
1	1	0															
- XNOR	 $Z = \overline{X \oplus Y}$	<table border="1"> <thead> <tr> <th>X</th> <th>Y</th> <th><math>\overline{X \oplus Y}</math></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	X	Y	$\overline{X \oplus Y}$	0	0	1	0	1	0	1	0	0	1	1	1
X	Y	$\overline{X \oplus Y}$															
0	0	1															
0	1	0															
1	0	0															
1	1	1															

6

## Functional blocks

- With basic logic gates we can build up different functional blocks such as
  - Adders
  - Multiplexers
  - Decoders
  - Latches
  - Registers
  - Counters

7

## Adders (1/3)

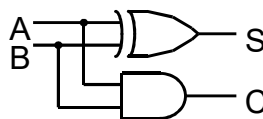
- One bit adder

- Truth table
- Logic function

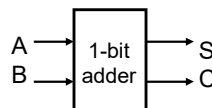
Sum:  $S = A \text{ xor } B$   
Carry:  $C = AB$

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Digital circuit



- Symbol



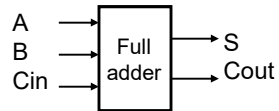
8

## Adders (2/3)

- One bit adder with carry

- Called Full Adder

- Symbol



- Function

- Adding three 1-bit numbers

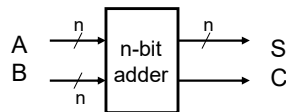
A	B	C <sub>in</sub>	S	C <sub>out</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

9

## Adders (3/3)

- n-bit adder

- Symbol



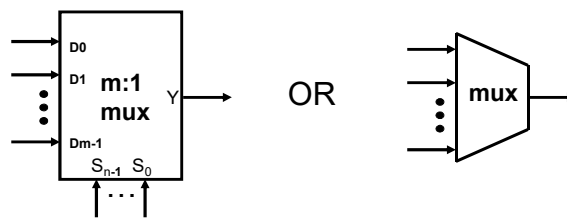
- Function

- Adding two n-bit numbers
      - The result is n-bit sum and 1-bit carry

10

## Multiplexer

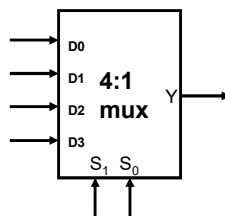
- Function:
  - A multiplexer selects one input among multiple inputs and passes it to output.
  - The selection is controlled by control signal  $S_{n-1} \sim S_0$
- The symbol:



11

## Example

- 4:1 multiplexer

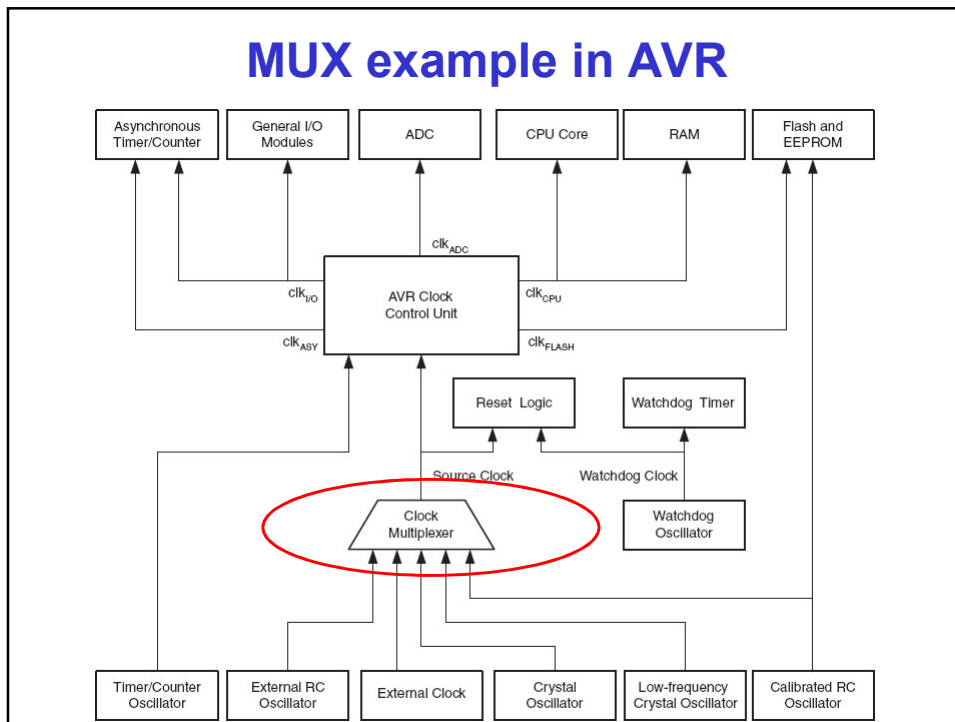


- Function:

**When  $S_1S_0 = 00$ ,  $Y=D0$**   
**When  $S_1S_0 = 01$ ,  $Y=D1$**   
**When  $S_1S_0 = 10$ ,  $Y=D2$**   
**When  $S_1S_0 = 11$ ,  $Y=D3$**

12

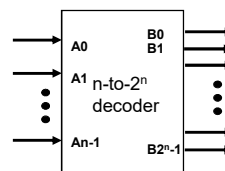
## MUX example in AVR



13

## Decoder

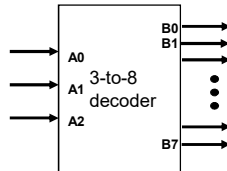
- Function:
  - A decoder converts an  $n$ -bit input code to an  $m$ -bit output code
    - $n \leq m \leq 2^n$
    - each valid input code word produces a unique output code
  - Typical  $n$ -to- $2^n$  decoder
    - One line of outputs represents a specified input combination
- The symbol:



14

## Example

- 3-to-8 register file address decoder



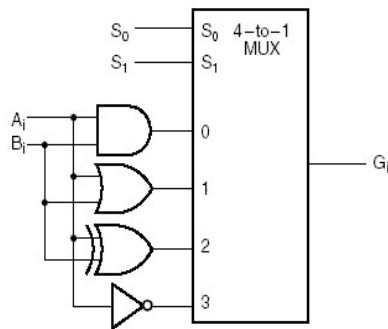
- Function:

Address			Output							
A2	A1	A0	B0	B1	B2	B3	B4	B5	B6	B7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

15

## Multi-operation unit

- Perform 1-bit logic operations:
  - AND, OR, XOR, NOT



S <sub>1</sub>	S <sub>0</sub>	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \bar{A}$	NOT

(b) Function Table

(a) Logic Diagram

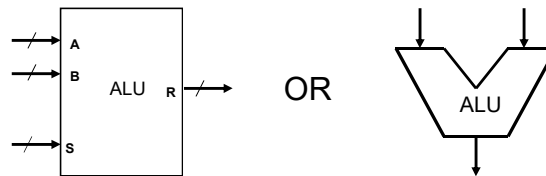
Constructed with functional components

16



## ALU

- Perform arithmetic and logic operations
  - such as addition, subtraction, logic AND, Logic OR
- Symbol:
  - A, B are operands, S selects one of operations in ALU



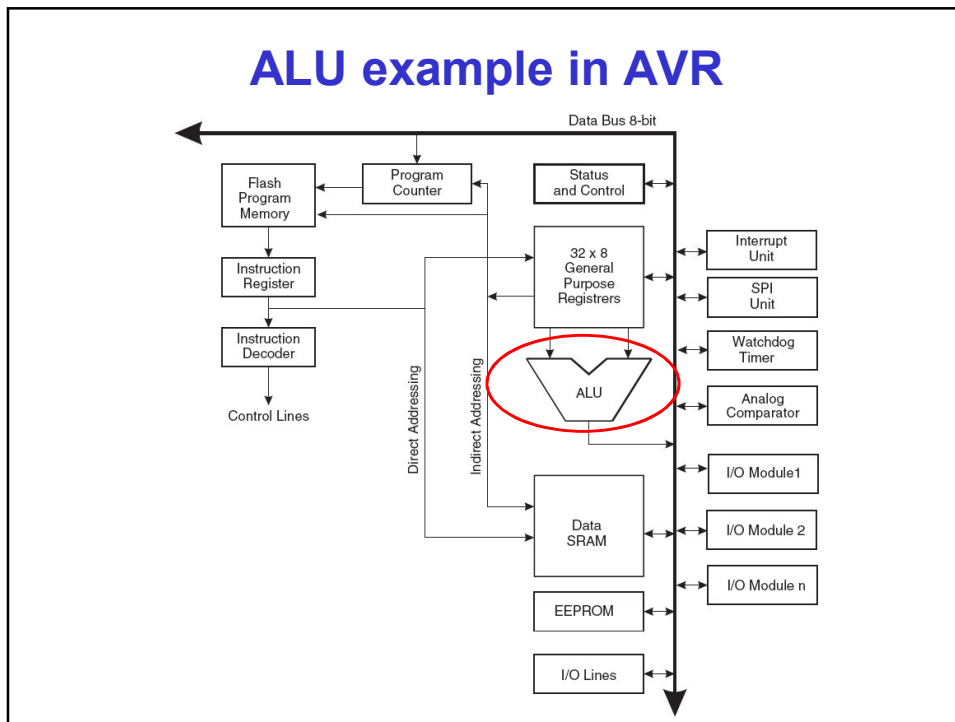
17

## Example

Operation selection S2 S1 S0	Operation
0 0 0	Addition
0 0 1	Subtraction
0 1 0	AND
0 1 1	OR
1 0 0	XOR
1 0 1	NOT
1 1 0	Increment
1 1 1	Transfer

18

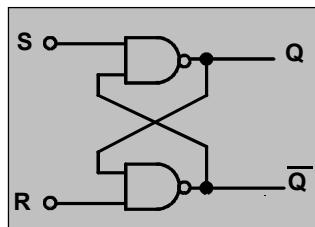
## ALU example in AVR



19

## Latches and Flip Flops (1/3)

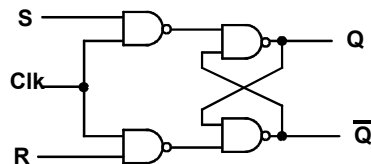
- A latch can store one bit information.
- Can be constructed in many ways.
- 2-NAND gate latch
  - R=0, reset the latch
  - S=0, set latch
  - S = R = 1, store the data



20

## Latches and Flip Flops (2/3)

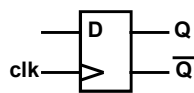
- Clocked latch uses clock to control the latch operation
  - When Clk=1,
    - S=1, set the latch
    - S=1, reset the latch
    - S=R=0, store data
  - When Clk=0,
    - Data is retained



21

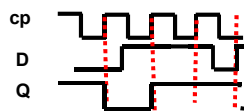
## Latches and Flip Flops (3/3)

- Flip Flops use clock edges to trigger the data-store operation.
  - A very commonly used Flip Flop is D FF
    - On the rising edge of clock, the input data D is locked into the D flip flop



D	Q(n+1)
0	0
1	1

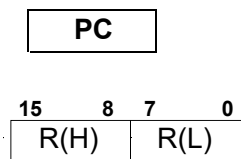
- Timing diagram



22

## Registers (1/3)

- A register is a collection of latches/FFs
  - storing a vector of bit values
- symbol

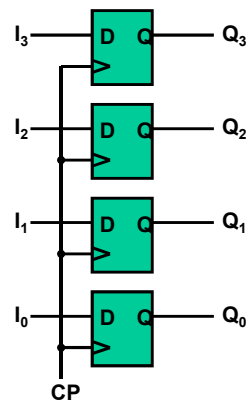


23

## Registers (2/3)

- 4-bit Parallel In Parallel Out (PIPO) registers.

The 4-bit input  $I_3I_2I_1I_0$  is "loaded" (copied to the output  $Q_3Q_2Q_1Q_0$  of the **D FFs**) on the rising clock edge, and that output is held until the next clock edge.

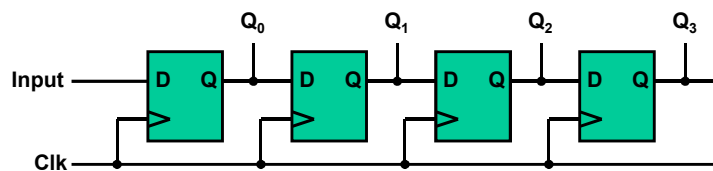


24

## Registers (3/3)

- 4-bit Serial In Parallel Out (PISO) registers.

On the clock edge, the output of each flip-flop is passed to the next flip-flop in the chain. The input signal is fed serially (one bit at a time) into the first flip-flop. The flip-flop outputs are available in parallel



25

## Counters (1/2)

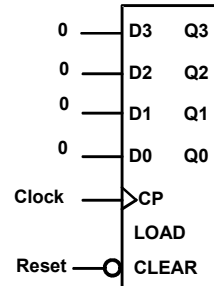
- A counter increases/decrease its value every clock cycle.

26

## Counters (2/2)

- 4-bit counter

- with a **synchronous load**
- an **asynchronous clear**
- counts through 0, 1, 2, ..., 15, 0

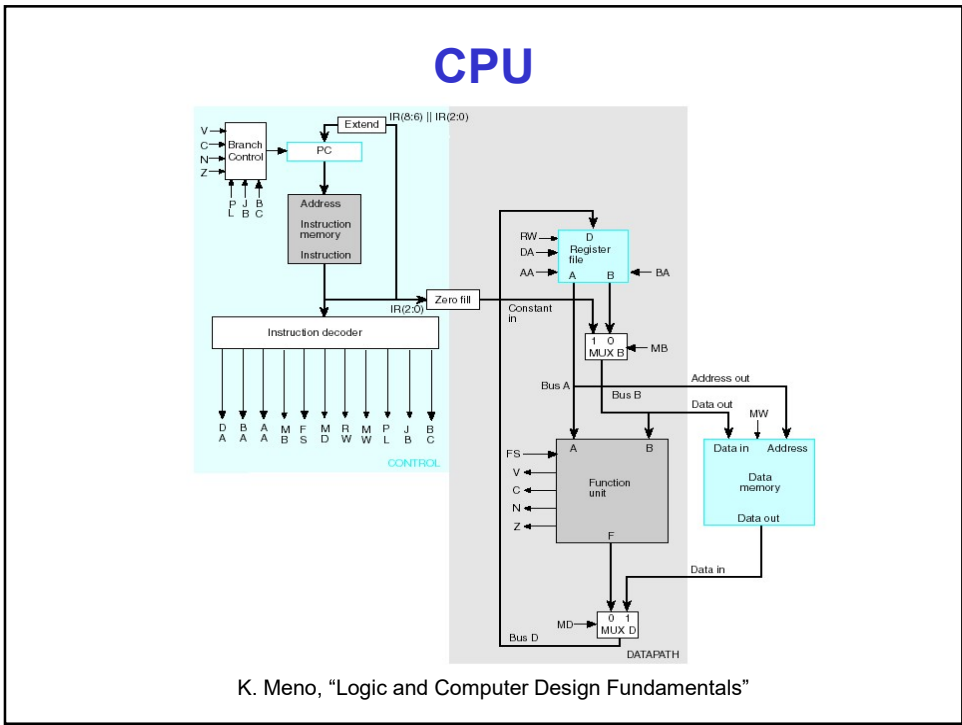


27

## Digital systems

- A digital system generally includes two parts:
  - **Datapath**
    - Performing a variety of operations on data from different sources
  - **Control unit**
    - Controlling the selection of the operation and data

28



29

## Example of datapath operations

DA, AA, BA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
R0	000	Register	0	$F = A$	00000	Function	0	No write	0
R1	001	Constant	1	$F = A + 1$	00001	Data In	1	Write	1
R2	010			$F = A + B$	00010				
R3	011			$F = A + B + 1$	00011				
R4	100			$F = A + \overline{B} - 1$	00100				
R5	101			$F = A + \overline{B} + 1$	00101				
R6	110			$F = A - 1$	00110				
R7	111			$F = A$	00111				
				$F = A \wedge B$	01000				
				$F = A \vee B$	01010				
				$F = A \oplus B$	01100				
				$F = \overline{A}$	01110				
				$F = B$	10000				
				$F = sr B$	10100				
				$F = sl B$	11000				

K. Meno, "Logic and Computer Design Fundamentals"

30

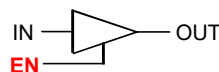
## Control unit

- Control signals determine the operation of the datapath
- Where do the control signals come from?
  - **from control unit**
- Control unit takes the instruction from instruction memory, together with the status values from datapath, to generate the control signals

31

## Some practical designs

- Tri-state buffer
  - Has three output states
    - High level signal (1) passed from the input
    - Low level signal (0) passed from the input
    - High impedance (Hi-Z)
      - Disconnecting the input devices to the output devices
  - Allows multiple logic gates to drive the same output (e.g, bus)



EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

32

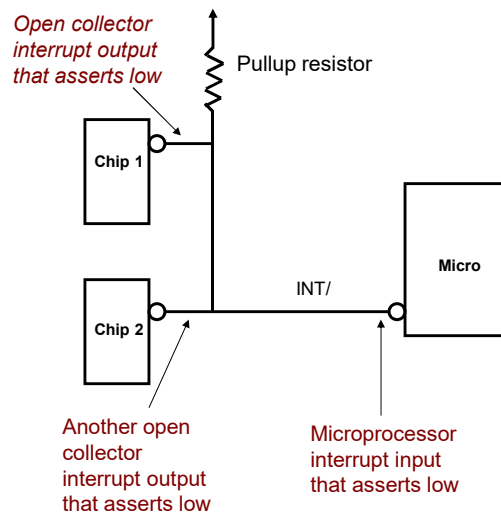


## Some practical designs

- Open collector
  - Act like one-way switch
    - When it is “Open”, no controlling operations

33

## Example



34

For comprehensive coverage of digital systems design, please take COMP3222

- Digital Circuits and Systems
- Offered in S2 each year

35

### **Reading material**

- Appendix B in Computer Organization and Design, The hardware/software interface.

36