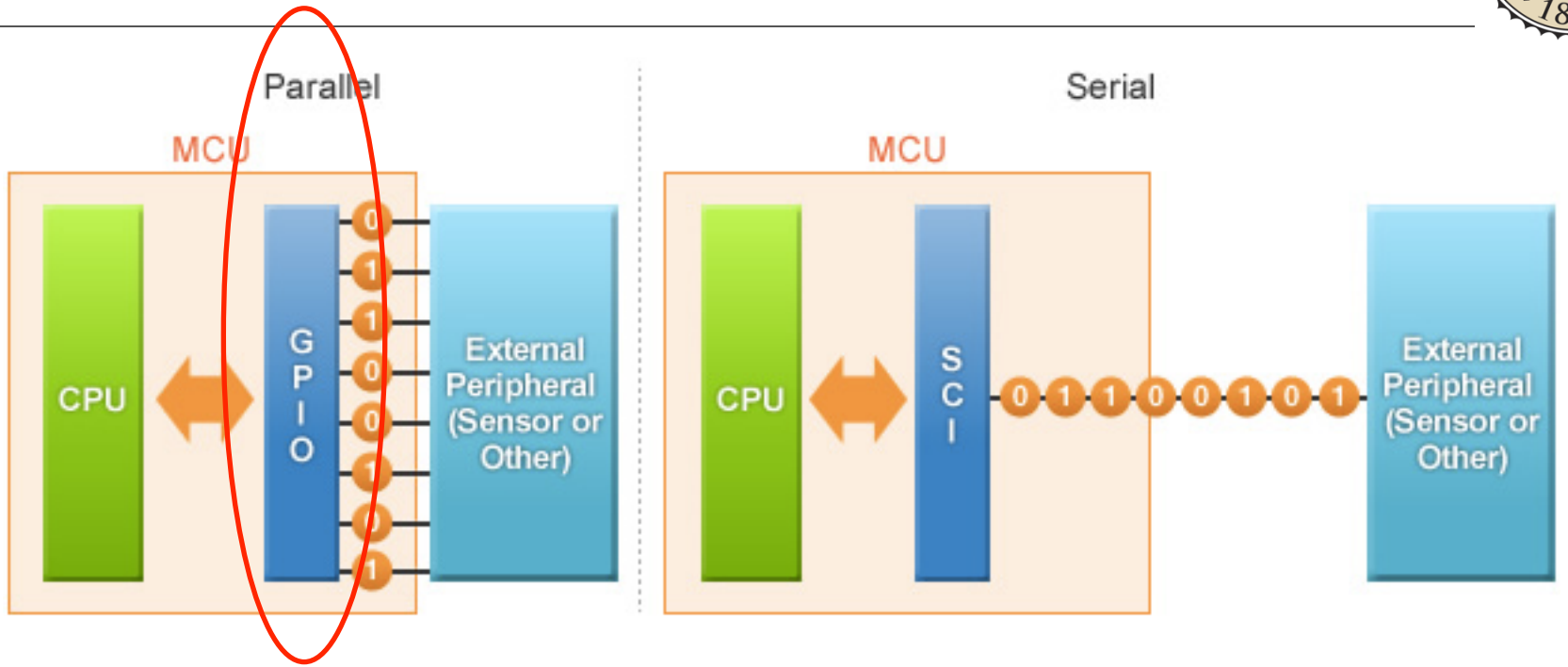
General Purpose I/O in a Microcontroller

Baris Aksanli
Department of Electrical and Computer Engineering
San Diego State University

Outline

- Port read/write and circuit examples
- Switches, switch matrices and multiplexing
- Keypad connections
- General purpose I/O (GPIO) on AVR

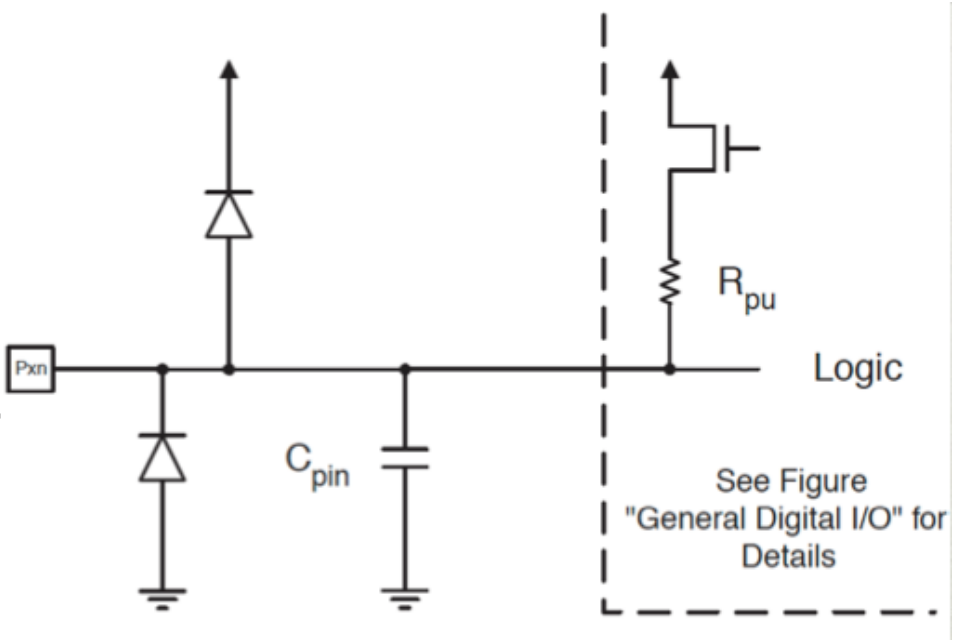
Remember from the last time



- GPIO is the facility that is going to provide us parallel communication
- We have ports and pins to accept inputs from outside or provide outputs to the outer environment

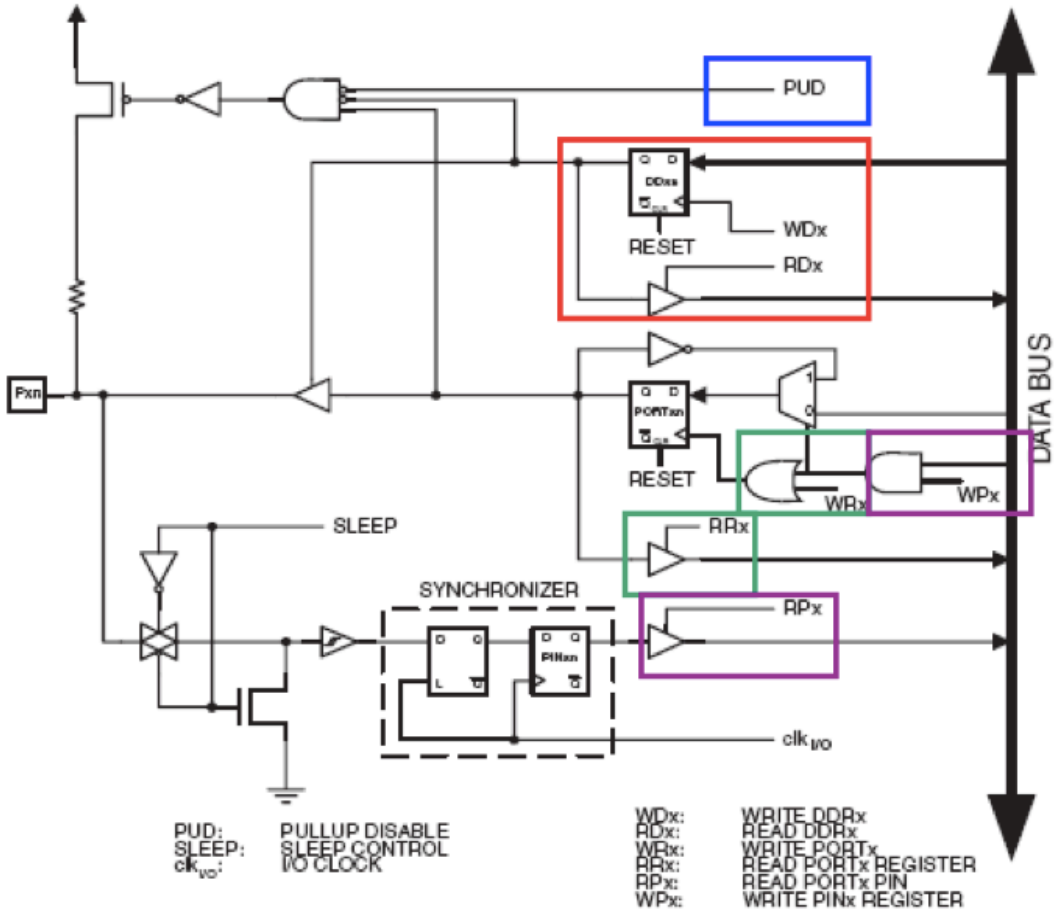
GPIO Overview

- Each pin represents a logic 0 or logic 1 value
  - Logic 0 is defined by low voltage (ground)
  - Logic 1 is defined by high voltage (Vcc)
- All GPIO pins have true Read-Modify-Write ability!
- The direction of one port pin can be changed independently of the other pins



Circuit Diagram for Each "Pin" for AVR

Figure 33. General Digital I/O(1)



- Data Direction Register (DDRn)
- PORT Output (PORTn)
- Port Input (PINn)
- Pull-Up Resistor for Input
  - Setting PORT = 1 while DDR = 0

ATMEGA General Purpose Digital I/O Ports

- The ATmega328P has 23 General Purpose Digital I/O Pins assigned to 3 Ports (8-bit Ports B, D and 7-bit Port C)
- Each I/O port pin may be configured as an **output** with symmetrical drive characteristics
  - Each pin driver is strong enough (20 mA) to drive LED displays directly
- Each I/O port pin may be configured as an input with or without a pull-up resistors
  - The values for the pull up resistors can range from 20 - 50 K ohms
- Each I/O pin has protection diodes to both VCC and Ground

Pin Configuration

- DDRx: for configuring Data Direction (input/output) of the port pins
- PORTx: for writing the values to the port pins in output mode
  - Configuring the port pins in input mode
- PINx: reading data from port pins in input mode

Pull-up Resistor

- First, we need to understand tri-state logic
- Think of each pin as a switch, where when the switch is:
  - ON: the value on the pin is transmitted as output
  - OFF: we cannot know about the value that is transmitted (high impedance)
- But it might be necessary to have a default value for this high-impedance state

INPUT		OUTPUT
A	B	C
0	1	0
1		1
X	0	Z (high impedance)

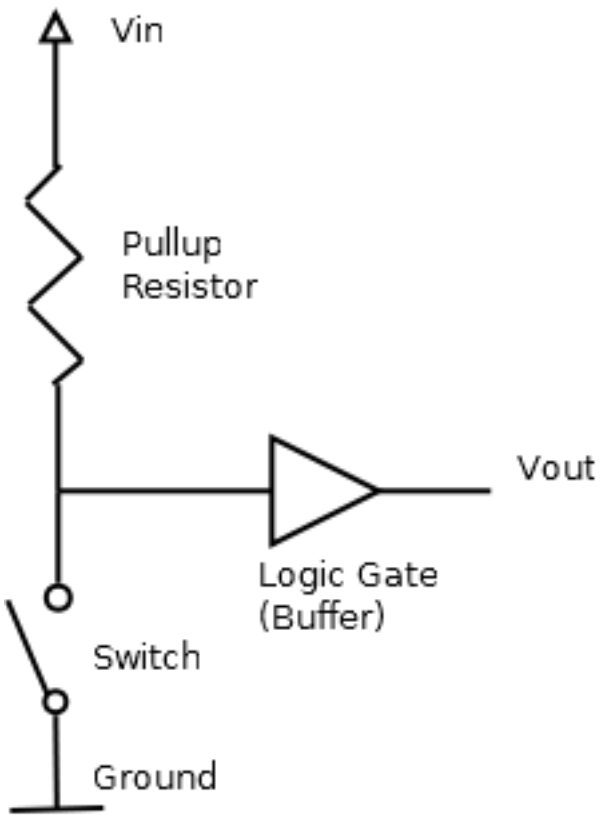
A tristate buffer can be thought of as a switch. If *B* is on, the switch is closed. If B is off, the switch is open.

Additional Points on High-Impedance State

- In high-impedance state, pins draw minuscule current, and don't load the circuit to which they are attached
- If the pin is not connected to a circuit while in this High-Z state, it is "floating" and read unpredictable values
- Unconnected inputs consume more power than inputs that are pulled high or low through resistors

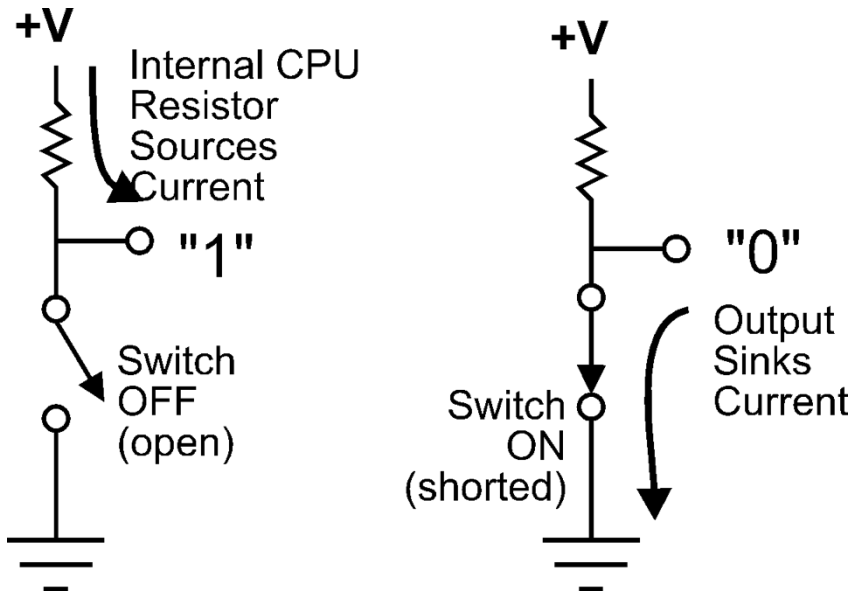
Pull-up Resistor

- Pull-up resistor is used to ensure that tri-stated input always reads HIGH (1) when it is not driven by any external entity
- Pull-up is very important when you are using tri-stated input buffers
- Tri-state input pin offers very high impedance and thus can read as logic 1/ logic 0 because of static charges on nearby objects
- Pin state changes rapidly and this change is unpredictable
- This may cause your program to go haywire if it depends on input from such tri-state pin
- Similarly, there is also "pull-down" resistor



Switch Input Circuit

- Use pull-up R for sourcing current
  - Internal/external R
  - Switch open
  - Logic level = 1 (high)
- Switch ON pulls down pin, sinking current
  - Switch closed
  - Logic level = 0 (low)

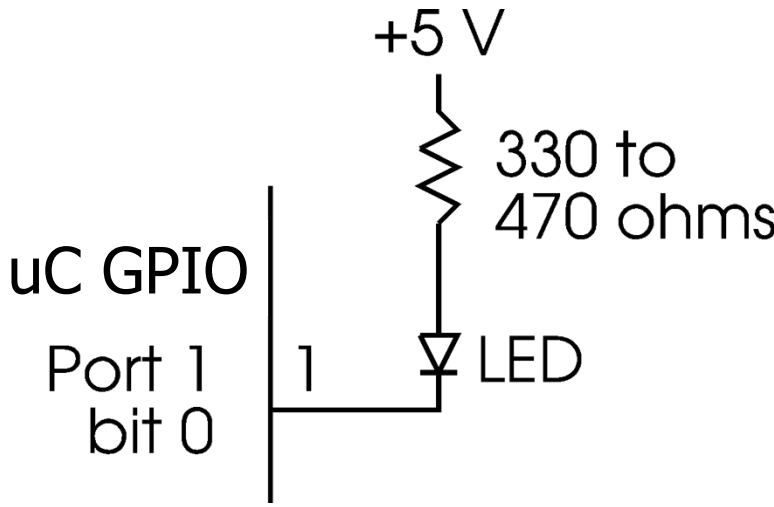


GPIO as Output

- The pin is in a state of low impedance
- The pin can supply current to other circuits (40mA max)
- Shorts or attempting to draw more current can fry your board
- Use external resistors to limit current from the pins

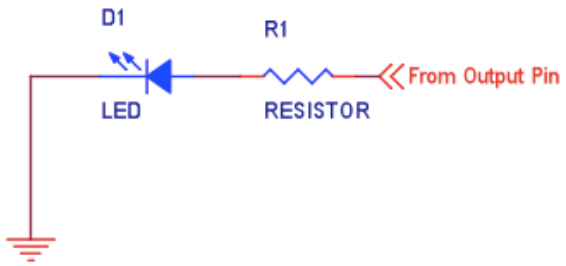
Example: Driving an LED directly

- Drive LED from Vdd
- LED OFF when bit= 1
- LED ON when bit= 0
- For LED with Vf=~2V
  - voltage across R is:
    - +5V - 2V = 3V
  - current in R and LED: 3V/330 ohms = 9 mA
  - What if Vdd = 3V?



Other Examples: LED (cont'd)

- **1) High to light LED (Active High)**



Does it matter?

Soemtimes.

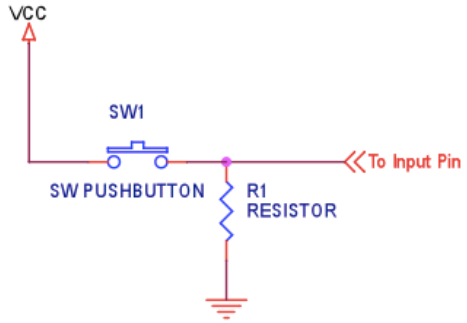
Generally, devices can Sink more current than they can Source (#2 is better)

- **2) Low to light LED (Active Low)**

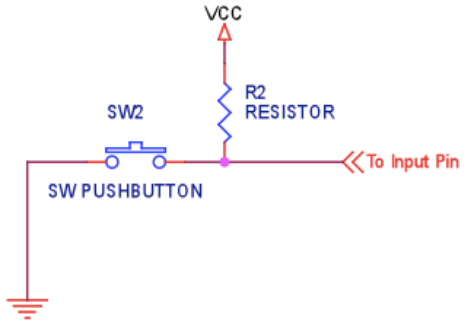
Source: EE192C – Klein, Stanford

Other Examples: Push button

- 1) Normally Low, High on Press



- 2) Normally High, Low on Press



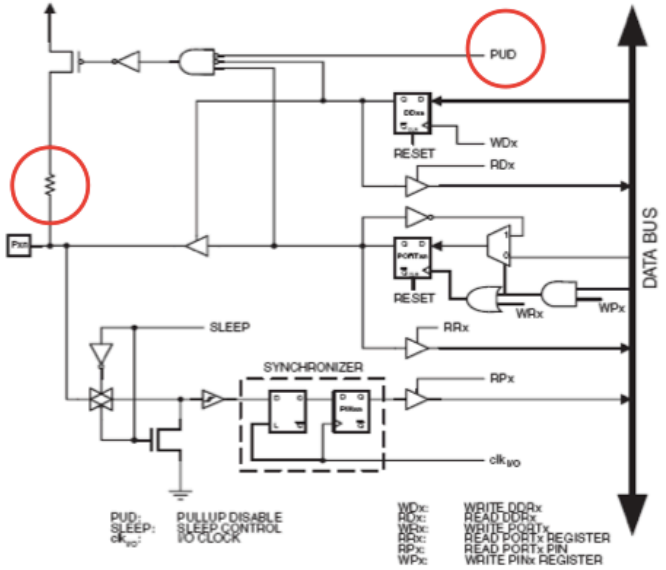
Does it matter?

YES!

#2's resistor can be the internal pull-up:

no extra external components necessary!

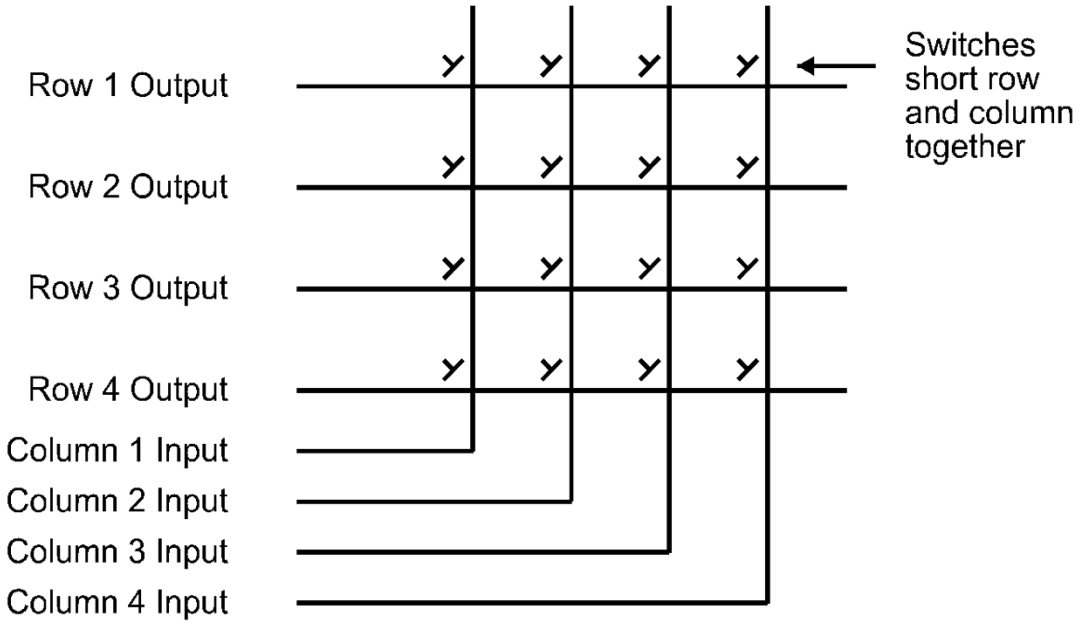
Figure 33. General Digital I/O(1)



Source: EE192C – Klein, Stanford

Other Examples: Matrix Keypad

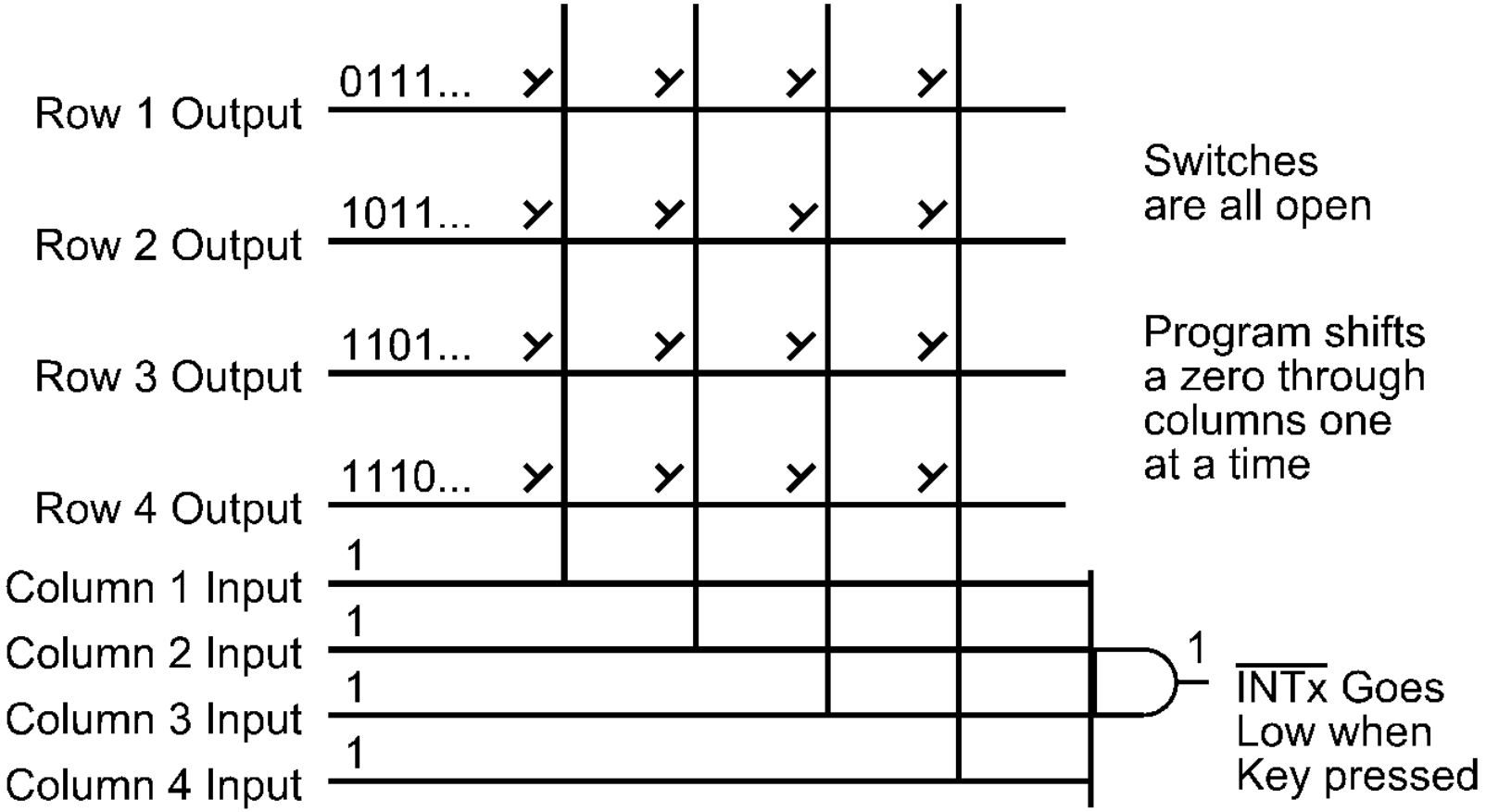
- Read by scanning each col/ row and checking for shunts / key press
- Expandable to NxM keys
  - Gives **(n/2)²** keys using only **n** I/O lines
  - Takes time to scan
  - It saves the limited I/O pins on a uC
- Often impossible / ambiguous with multiple simultaneous key presses



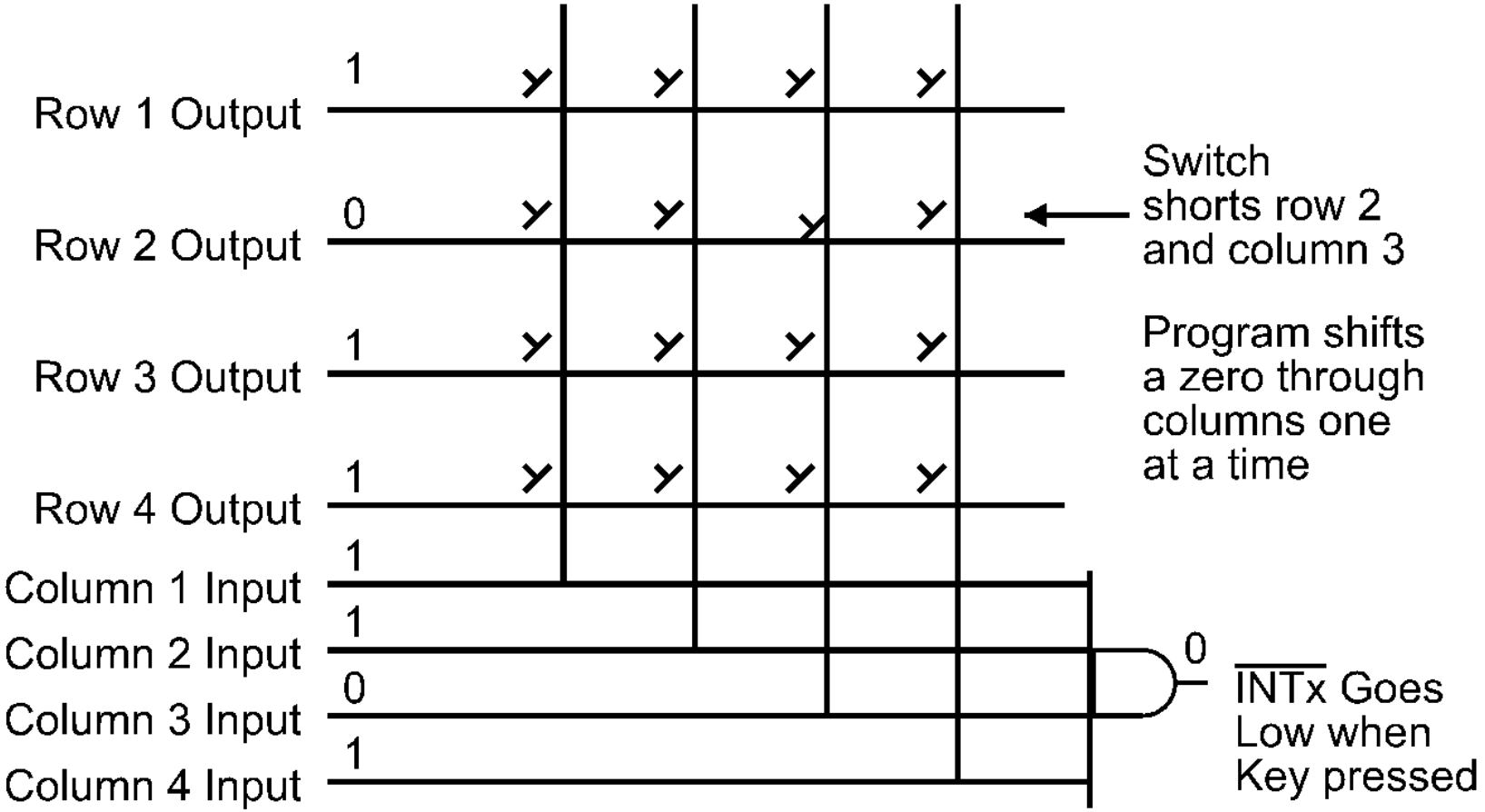
How does it work?

- Switches organized as row/column
- Drive rows, read columns, pull-up on cols
- Switch shorts row line to column line
- Walking zero on columns to activate one column at a time
- Check for low level on row inputs to determine which key in this column is pressed

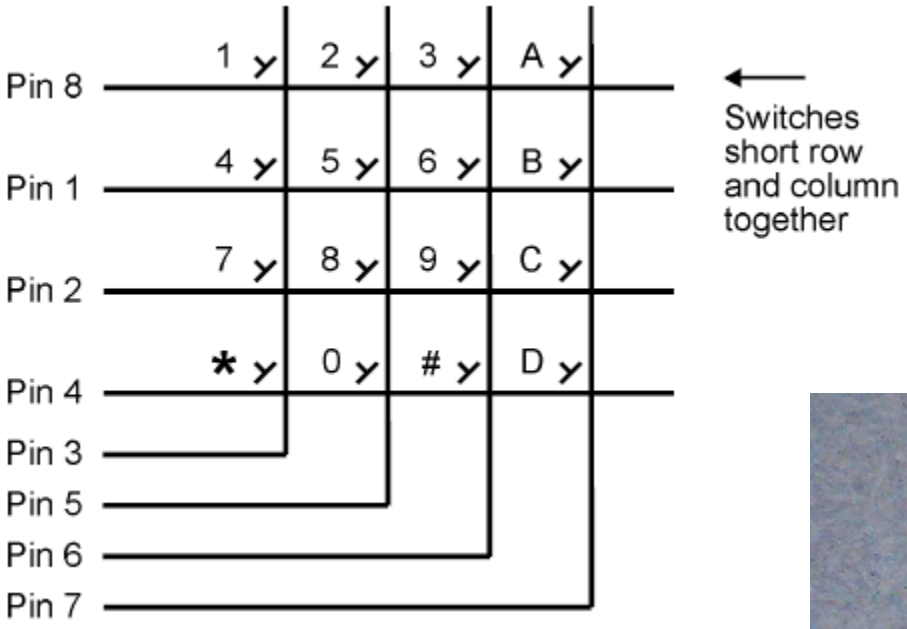
Scanning the Switch Matrix

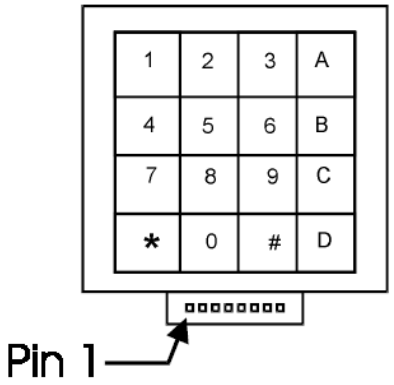


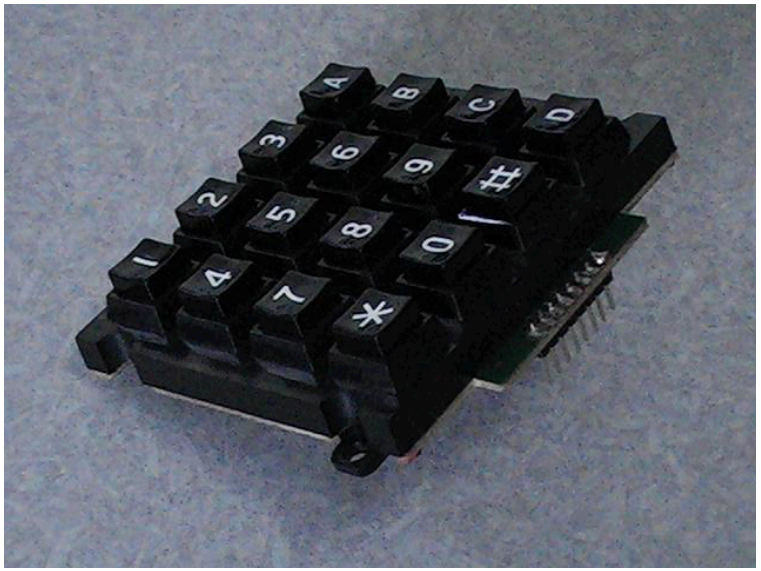
Multiplexed Switch Matrix



Keypad Connections







Keypad is jameco.com part #169244, with an 8 pin single row male header, with pin spacing = 0.1" soldered to the keypad, so it can be plugged directly into the solderless proto board

Scanning a Keypad

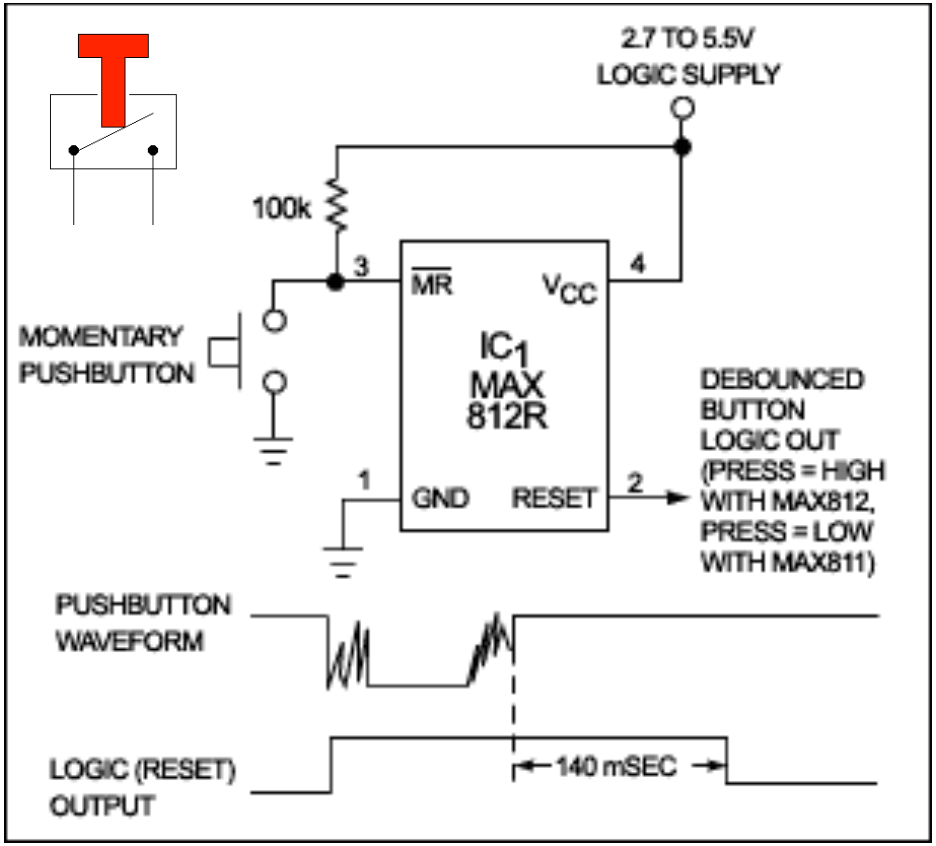

```
unsigned char code row[] = {0xEF, 0xDF, 0xBF, 0x7F };
       /* for strobing each row */
    /* 4 MSBs are row outputs, 4 LSBs are col inputs */

unsigned int keys; /* for storing key info */

void keypress (void) interrupt 0 // ISR for /INT0
  {
   unsigned char I;
   keys = 0xFFFF;  // get ready to input key press data
   for (I=0; I <= 3; I++)
    {
     P1 = row[I];  // strobe low one row at a time
     keys = (keys<<4) | (P1&0x0F);
        // read and store info from each column
    }
```

Switch Bounce

- Switches bounce!
  - Typical bounces last 1 to 50 milliseconds
  - Occurs mostly when contacts close
  - Can occur when contacts open
- Mechanical contacts bounce, due to elastic collision
- Can be de-bounced with hardware or software



De-bouncing in Software

- Look for first transition low (ON)
- Ignore additional switch transitions for de-bounce period, typically ~50 mS
- If switch is still on, key is down
- Look for first transition high (OFF)
- Ignore transitions for ~50 mS
- If switch is still high, key is up

Lets Remember the Features of AVR (ATmega32)

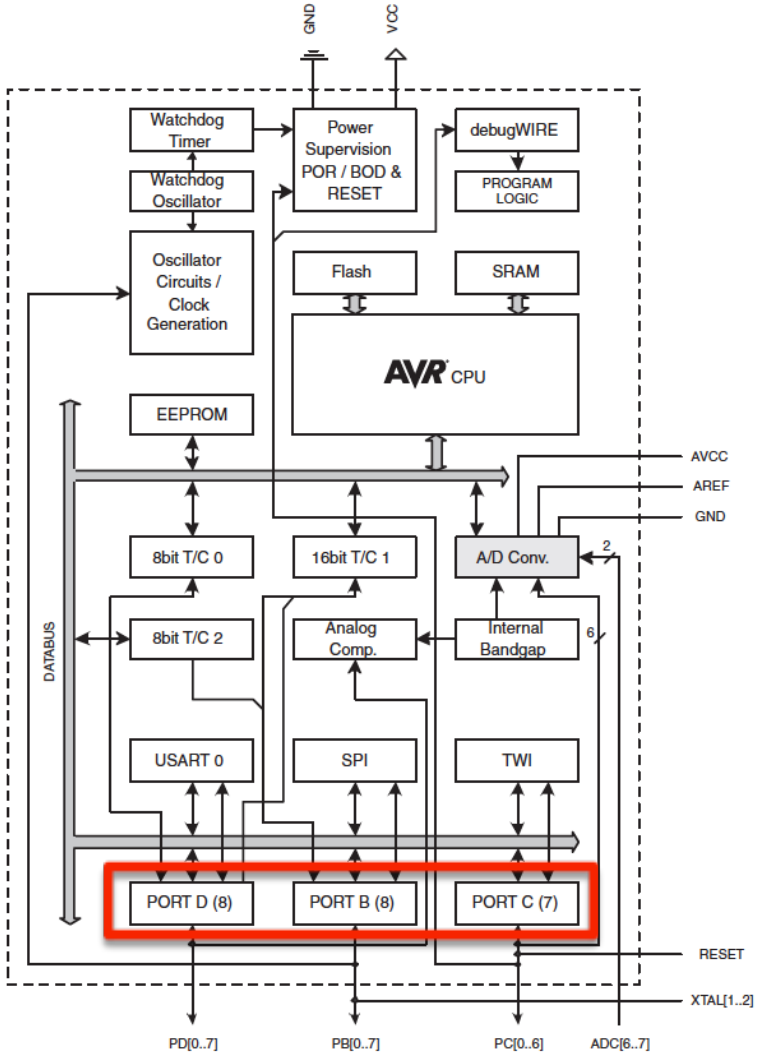
- Harvard architecture
- 8 bit microcontroller
- High performance – 20 MIPS @ 20MHz
- High endurance memory
  - 32KB Flash
  - 1KB EEPROM
  - 2KB internal SRAM
- Two 8-bit, one 16-bit timer with total 4 PWM channels
- On chip 10 bit ADC, 8 channels
- 27 Programmable I/O Lines
- UART, I2C, SPI protocol support

Points to be Noted

- PORT : group of 8 pins, or set of pins used for exchanging data with external world
- Width of almost all registers : 8 bits (some 16 bits)
- In port related registers, every bit corresponds to one pin of the port.
  Bit 0 corresponds to Pin 0
  Bit 0 corresponds to Pin 1 .. etc
- Remember direct one to one correspondence between HEX and BINARY numbers
  0xFF = 1111 1111
  0xAA = 1010 1010
  0x11 = 0001 0001

AVR Architecture

- 8-bit ports, 3 of them (B, C, D)
- Each port is controlled by three 8-bit registers, except Port C (7-bit)
- Next slide



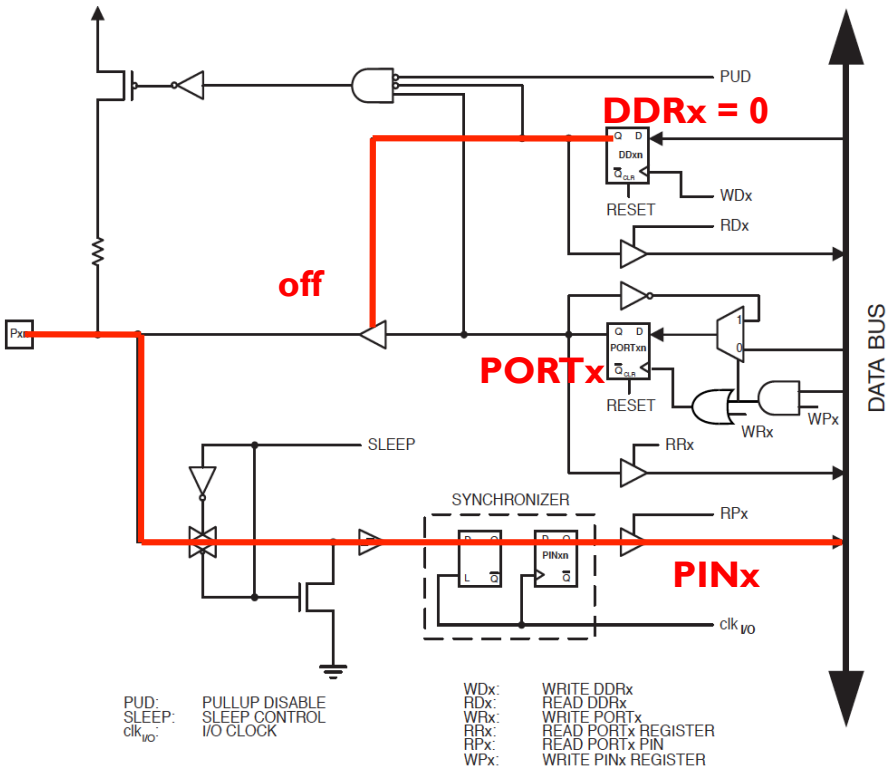
Source: Carl Ebeling, CSE P657

Input/Output Basics

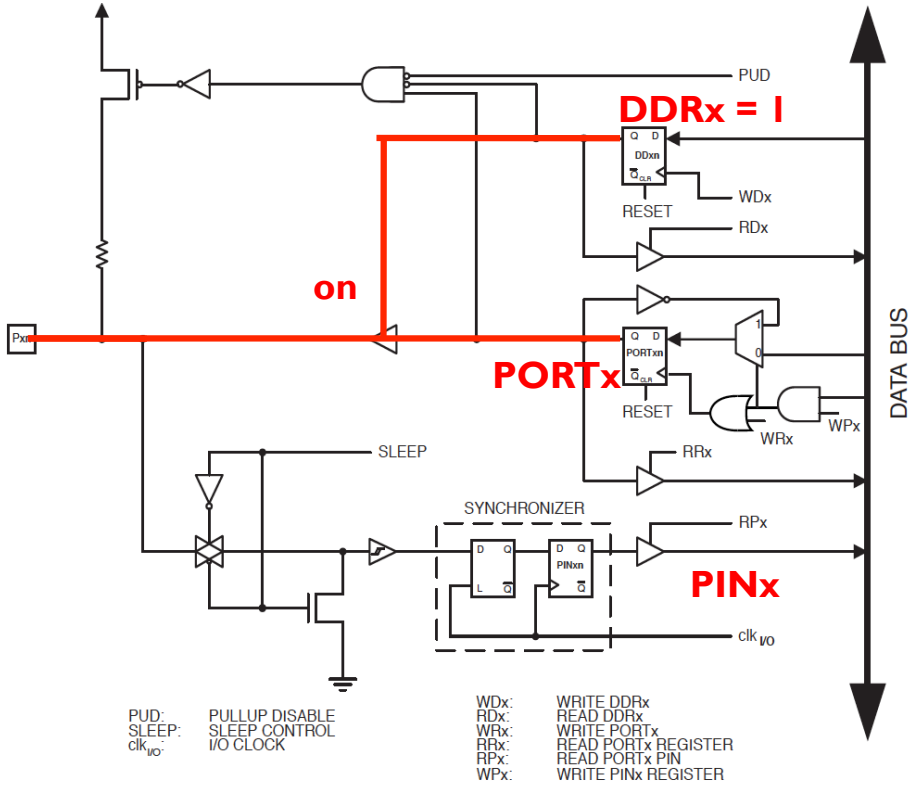
- **Three registers :**
  - DDRx : for configuring Data Direction (input/output) of the port pins
  - PORTx: for writing the values to the port pins in output mode. Configuring the port pins in input mode
  - PINx: reading data from port pins in input mode

- Where x = A,B,C,D… depending on the available ports in your AVR.
- For our device, x = B, C, D

Input/output Pin Configurations



INPUT, DDRX = 0



OUTPUT, DDRX = 1

Source: Carl Ebeling, CSE P657

DDR – Data Direction Register

- Configures data direction of the port - Input / Output
- DDRx.n = 0 > makes corresponding port pin as input
  DDRx.n = 1 > makes corresponding port pin as output

- Examples :
  - to make all pins of port A as input pins :
    ```
    DDRA = 0b00000000;
    ```
  - to make all pins of port A as output pins
    ```
    DDRA = 0b11111111;
    ```
  - to make lower half of port B as output and higher half as input:
    ```
    DDRB = 0b00001111;
    ```

PIN Register

- Used to read data from port pins, when port is configured as input
- First set DDRx to zero, then use PINx to read the value
- If PINx is read, when port is configured as output, it will give you data that has been outputted on port
- There two input modes :
  - Tri-stated input
  - Pull-up input (default value 1 for the high-impedance state)

- Example :

```
DDRA = 0x00;        //Set PA as input
x = PINA;   //Read contents of PA
```

PORT Register

- Used for two purposes …

1) For data output, when port is configured as output:

- Writing to PORTx.n will immediately (in same clock cycle) change state of the port pins according to given value.
- Do not forget to load DDRx with appropriate value for configuring port pins as output.
- Examples :
  - to output oxFF data on PB

```
DDRB = 0b11111111;  //set all pins of port b
                        as outputs
PORTB = 0xFF;      //write data on port
```

  - to output data in variable x on PA

```
DDRA = 0xFF;     //make port a as output
PORTA = x;      //output 8 bit variable on port
```

PORT Register

2) for configuring pin as tri-state/pull-up, when port is configured as input) :

- When port is configured as input (i.e DDRx.n=0), then PORTx.n controls the internal pull-up resistor
- PORTx.n = 1 : Enables pull-up for n^{th} bit
  PORTx.n = 0 : Disables pull-up for n^{th} bit, thus making it tri-state
- Examples :
  - to make PA as input with pull-ups enabled and read data from PA

```
DDRA  = 0x00;    //make port A as input
PORTA = 0xFF;    //enable all pull-ups
y = PINA;        //read data from port A pins
```

  - to make PB as tri stated input

```
DDRB  = 0x00;  //make port B as input
PORTB = 0x00;  //disable pull-ups and make it
                 tri-state
```

Input/output Basics Summary

- **Following table lists register bit settings and resulting function of port pins**

register bits → pin function↓	**DDRx.n**	**PORTx.n**	**PINx.n**

tri-stated input	0	0	read data bit(s)
			x = PINx.n;
			y = PINx;

pull-up input	0	1	read data bit(s)
			x = PINx.n;
			y = PINx;

output	1	write data bit(s) PORTx.n = x; PORTx = y;	n/a

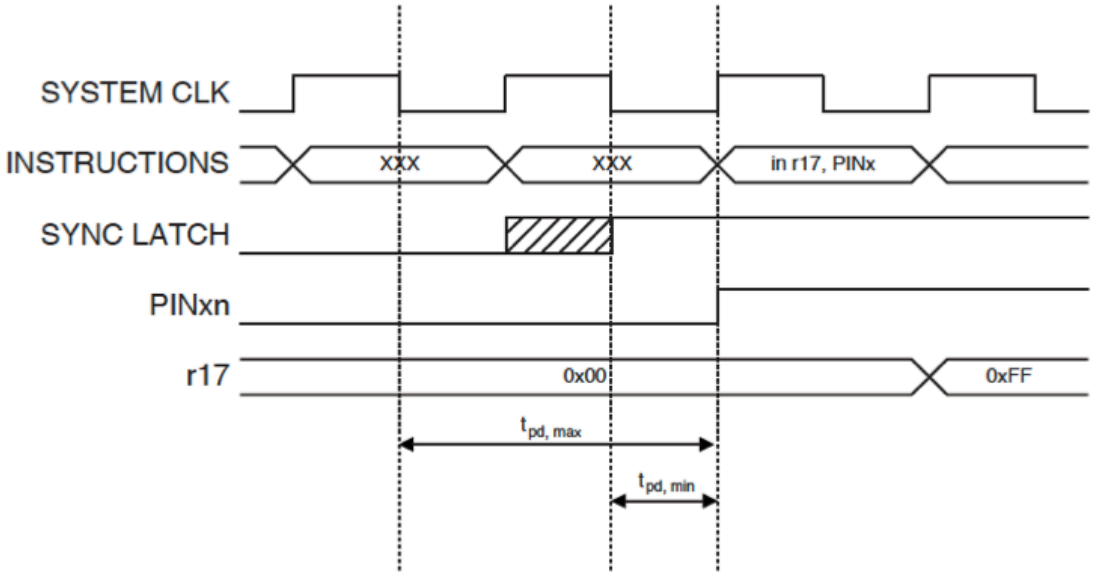
MCUCR (MCU Control Register)

- PUD: Pull-Up Disable!
- When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDRxn and PORTxn Registers are configured to enable the pull-ups
  - (DDxn = 0, PORTxn = 1).

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS(1)	BODSE(1)	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Synchronization when Reading a Pin Value

- You read a port pin by reading the corresponding **PINxn** Register bit. The PINxn Register bit and the preceding latch (to keep the value) constitute a synchronizer.
- This is needed to avoid **metastability** if the physical pin changes value near the edge of the internal clock, but it also introduces a delay as shown in the timing diagram.



Source: EE444, CSULB

Synchronization when Reading a Pin Value

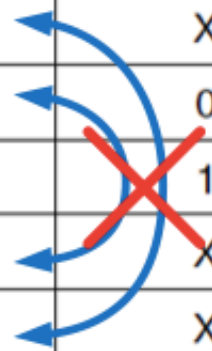
- Consider the clock period starting shortly after the first falling edge of the system clock
- The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the "SYNC LATCH" signal
- The signal value is latched when the system clock goes low
- It is clocked into the PINxn Register at the succeeding positive clock edge
- As indicated by the two arrows tpd,max and tpd,min, a single signal transition on the pin will be delayed between 1/2 and 11/2 system clock period depending upon the time of assertion

Source: EE444, CSULB

Switching Between Configurations

- When switching between **tri-state ({DDRxn, PORTxn} = 0b00)** and **output high ({DDRxn, PORTxn} = 0b11)**, an intermediate state with either pull-up enabled **({DDRxn, PORTxn} = 0b01)** or **output low ({DDRxn, PORTxn} = 0b10)** must occur
- Switching between input with pull-up **({DDRxn, PORTxn} = 0b01)** and output low **({DDRxn, PORTxn} = 0b10)** generates the same problem. You must use either the tri-state **({DDRxn, PORTxn} = 0b00)** or the output high state **({DDRxn, PORTxn} = 0b11)** as an intermediate step

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)



Source: EE444, CSULB

Alternative Functions of Ports

- Each pin in Ports B, C, and D has alternative functions
- An example (for Port D) is shown below, see the device manual for full details

Port Pin	Alternate Function
PD7	AIN1 (Analog Comparator Negative Input) PCINT23 (Pin Change Interrupt 23)
PD6	AIN0 (Analog Comparator Positive Input) OC0A (Timer/Counter0 Output Compare Match A Output) PCINT22 (Pin Change Interrupt 22)
PD5	T1 (Timer/Counter 1 External Counter Input) OC0B (Timer/Counter0 Output Compare Match B Output) PCINT21 (Pin Change Interrupt 21)
PD4	XCK (USART External Clock Input/Output) T0 (Timer/Counter 0 External Counter Input) PCINT20 (Pin Change Interrupt 20)
PD3	INT1 (External Interrupt 1 Input) OC2B (Timer/Counter2 Output Compare Match B Output) PCINT19 (Pin Change Interrupt 19)
PD2	INT0 (External Interrupt 0 Input) PCINT18 (Pin Change Interrupt 18)
PD1	TXD (USART Output Pin) PCINT17 (Pin Change Interrupt 17)
PD0	RXD (USART Input Pin) PCINT16 (Pin Change Interrupt 16)

I/O Basics – Exercise

1. Configure PB as output port.
2. Configure PC as tri-stated input and read value into variable x.
3. Configure PA as pull-up input and read lower nibble into variable y and higher nibble into variable x.
4. Make higher nibble of PA as pull-up inputs and lower nibble as output.
5. Read, only input pins of above mentioned port and get the binary number present on those pins into variable x.
6. Write four bit number 0x5 onto output pins of above mentioned port.

Answers to Exercise

1. DDRB = oxFF (or) ob11111111 (or) 255;
2. DDRC = 0x00; PORTC = 0x00; x = PINC;
3. DDRA = 0x00; PORTA = oxFF; y = PINA & ob00001111; x = (PINA & ob11110000) / 2^4
4. DDRA = 0x0F; PORTA = oxF0;
5. x = (PINA & ob11110000) / 2^4
6. PORTA = PORTA | 0x05;