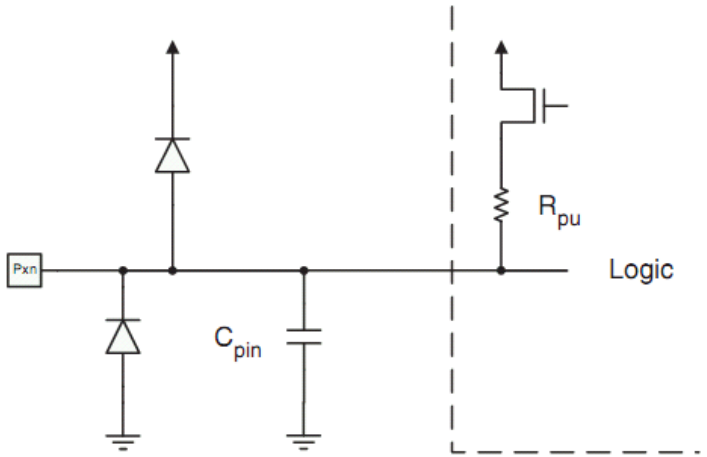
Design with Microprocessors

Year III Computer Science
1-std Semester

Lecture 3: AVR input/output

Input / Output ports

- Input/output ports: PORT**x**
- PORT**A**... **G** can be accessed through **in** and **out** instructions
- PORT **H … L** can be accessed only through **ld**, **st** (extended I/O address space: address > 0x0060h).
- ATmega 328P: PORT B, C, D
- ATmega 2560 – PORT A, B, C, D, E, F, G, H, J, K, L
- Every pin (Pxn) can be configured as input or output by writing the corresponding DDRx (data direction register)
- Writing to the port can be done through the PORTx register
- Reading from the port can be done through PINx register (read-only)
- “Pull up” resistors (http://en.wikipedia.org/wiki/Pull-up_resistor) can be activated/deactivated through “Logic”



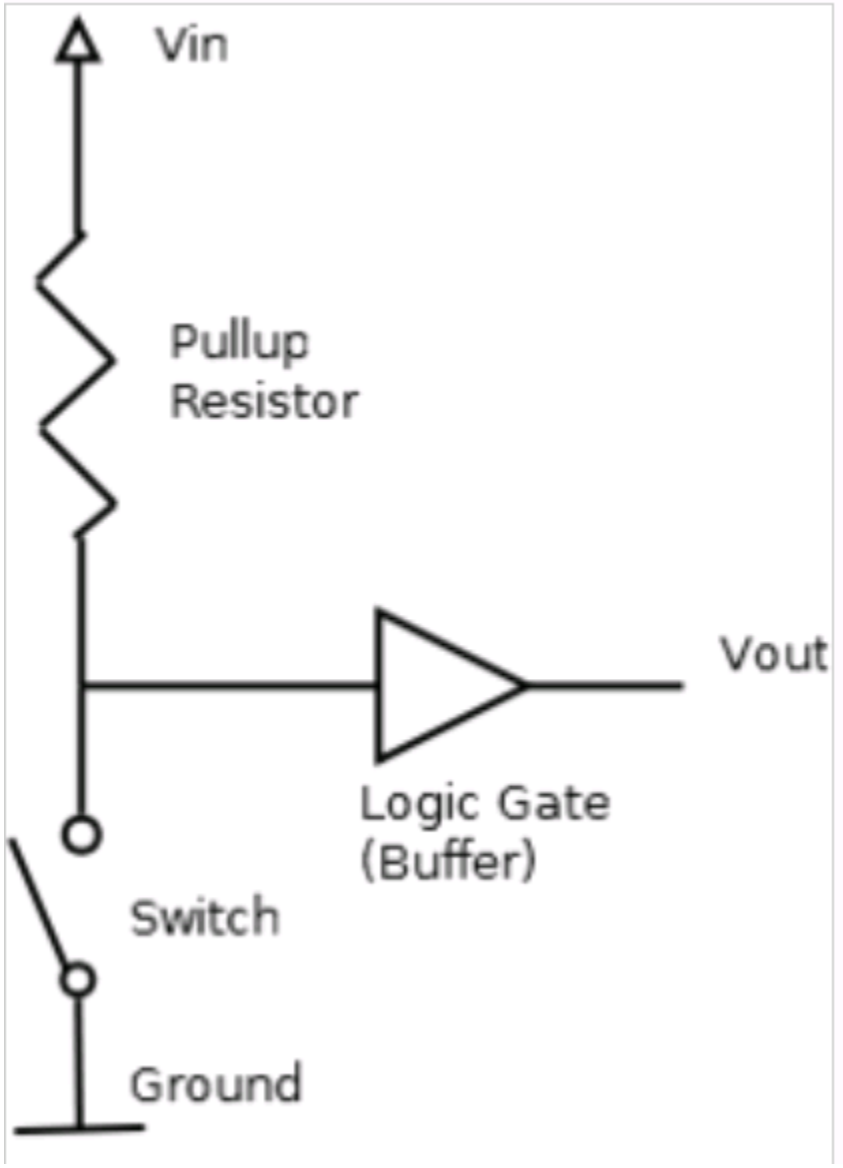
Input / Output ports

Pull-up resistors

(http://en.wikipedia.org/wiki/Pull-up_resistor)

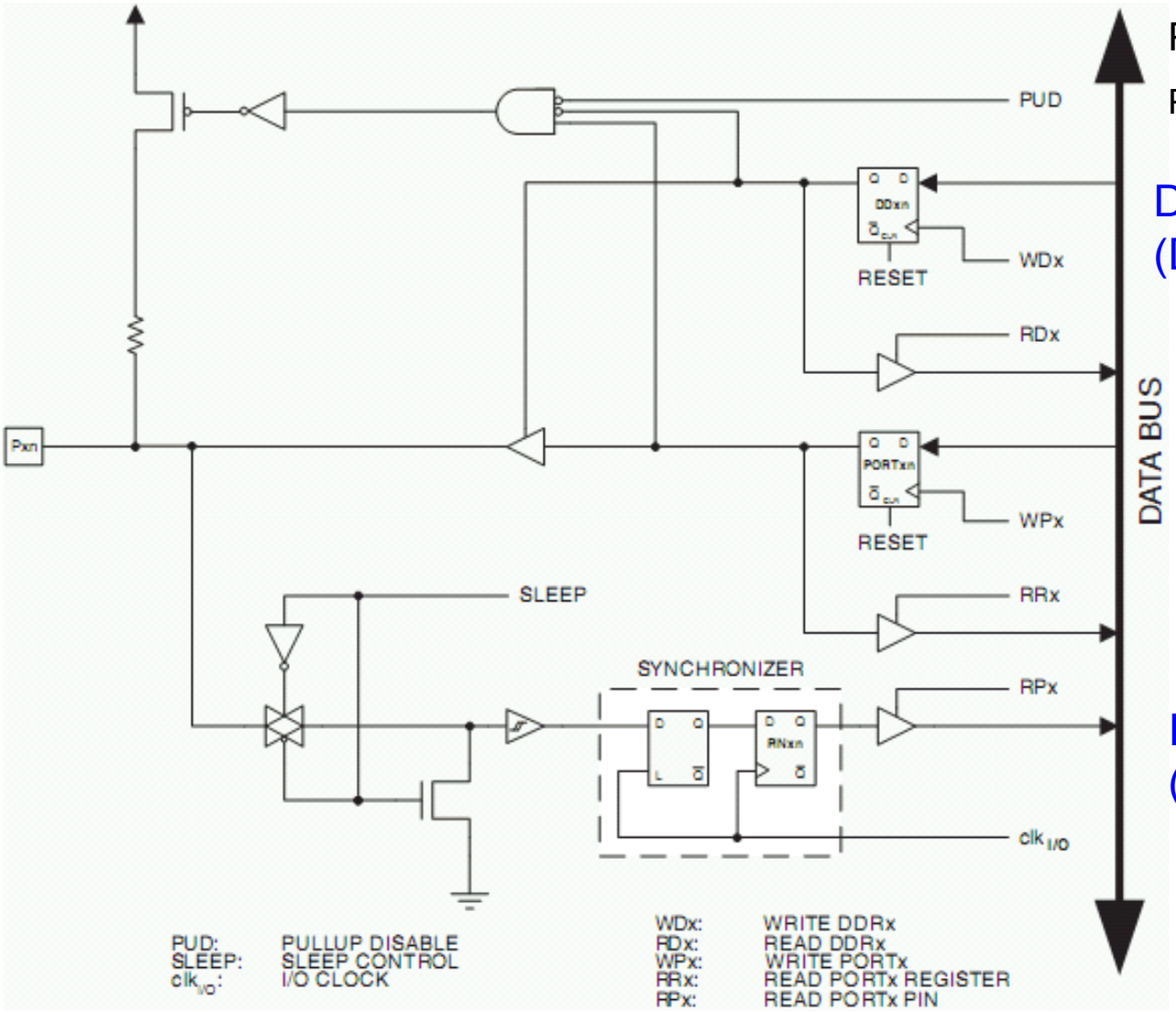
- Used in electronic logic circuits to ensure that inputs to logic systems settle at expected logic levels if external devices are disconnected or high-impedance.

- They may also be used at the interface between two different types of logic devices, possibly operating at different power supply voltages.



Input / Output ports

General schematic for 1 pin (Pxn) of the I/O ports



PUD – bit 4 in MCUCR

PUD=1 ⇒ pull-up disabled

Direction pin (DDRx)

Output data (PORTx)

Input data (PINx)

Input / Output ports

Three I/O memory address locations are allocated for each port, one for each

- Data Register – PORTx,
- Data Direction Register – DDRx
- Port Input Pins – PINx

Example (PORTA):

PORTA – Port A Data Register

Bit	7	6	5	4	3	2	1	0	
0x02 (0x22)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRA – Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x01 (0x21)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PINA – Port A Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x00 (0x20)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Notation: PORTxn = pin n of PORTx (ex: PORTB3 - for bit no. 3 in Port B).

Input / Output ports

Notations (related o the lecture):

- Code with “Arial” font is assembly language
- `Code with “CourierNew” font is the equivalent in C`

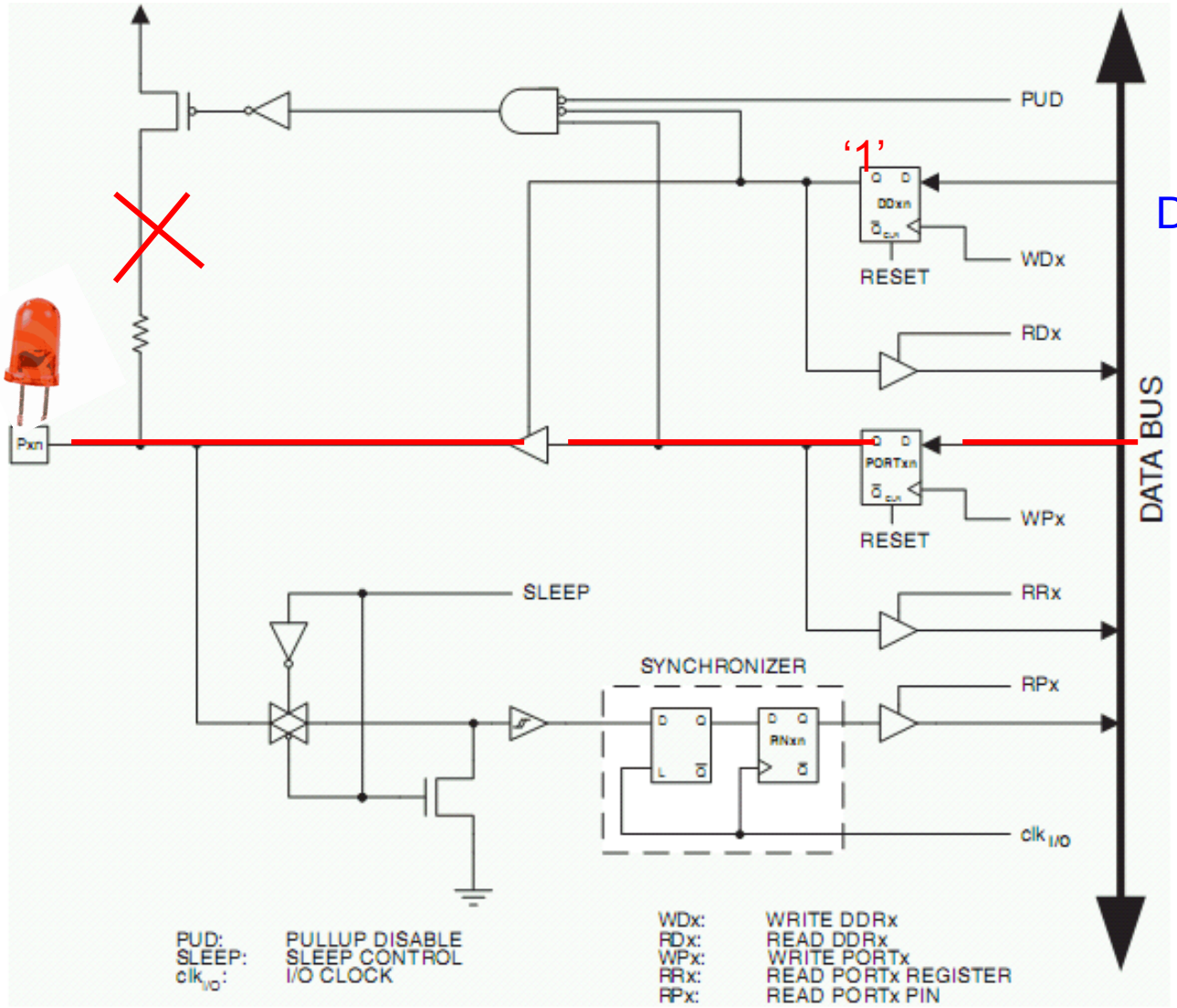
Use the one allowed by the IDE compiler.

Example: If you are working on an Arduino board with Arduino IDE use the C language

Most of the examples in the ATmega datasheets are written both in Assembly and in C !!!

Input / Output ports

Output configuration



Direction = 1

ldi ri, 0xFF

out DDRx, ri

```
DDRx=0xFF
```

Data written into PORTx are sent to the output (Pxn)

out PORTx, rj

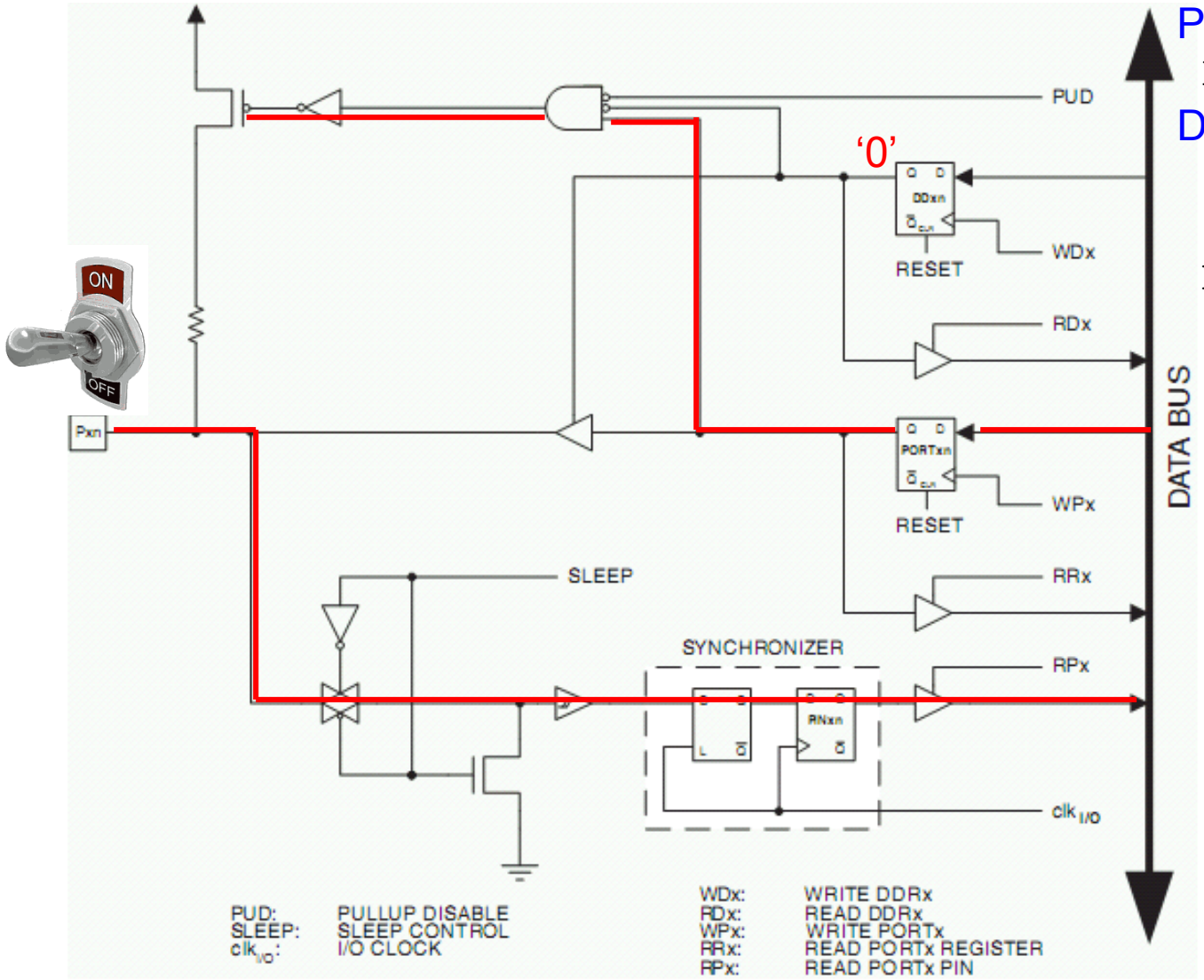
```
PORTx = val
```

Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn.

Input / Output ports

Input configuration (with pull-up resistor enabled)



PUD = 0 (pull-up enabled)

```
MCUCR &= B11101111;
```

Direction = 0 (input)

ldi ri, 0x00
out DDRx, ri

```
DDRx = 0x00
```

'1' written to PORTx activates the pull-up resistor

ldi ri, 0xFF
out PORTx, ri

```
PORTx = 0xFF
```

Data become available on PINx

in ri, PINx

```
val = PINx
```

Input / Output ports

Basic output example – C:

```
void setup() { // put your setup code here, to run once:
  DDRB=0xFF; //set port B as output
}
void loop() {// put your main code here, to run repeatedly:
  PORTB = B01010101; // write (output) to port B a constant
}
```

Basic input example – C:

```
byte val;
void setup() {// put your setup code here, to run once:
  MCUCR &= 0b11101111;// pull up enable (globally): PUD = 0
  PORTA = 0xFF; // activates pull-up resistors
}
void loop() {// put your main code here, to run repeatedly:
  val = PINA;
}
```

Input / Output ports

Stats of the I/O pins (Pxn)

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)

PUD – Pull Up Disable/Enable – GLOBAL

- Value ‘1’ of bit 4 from MCUCR deactivates all pull-up resistors

MCUCR – MCU Control Register

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

To set (PUD=1)

```
in r17, MCUCR
ori r17, 0b00010000
out MCUCR, r17
```

```
MCUCR |=(1<<PUD)
```

To clear (PUD=0)

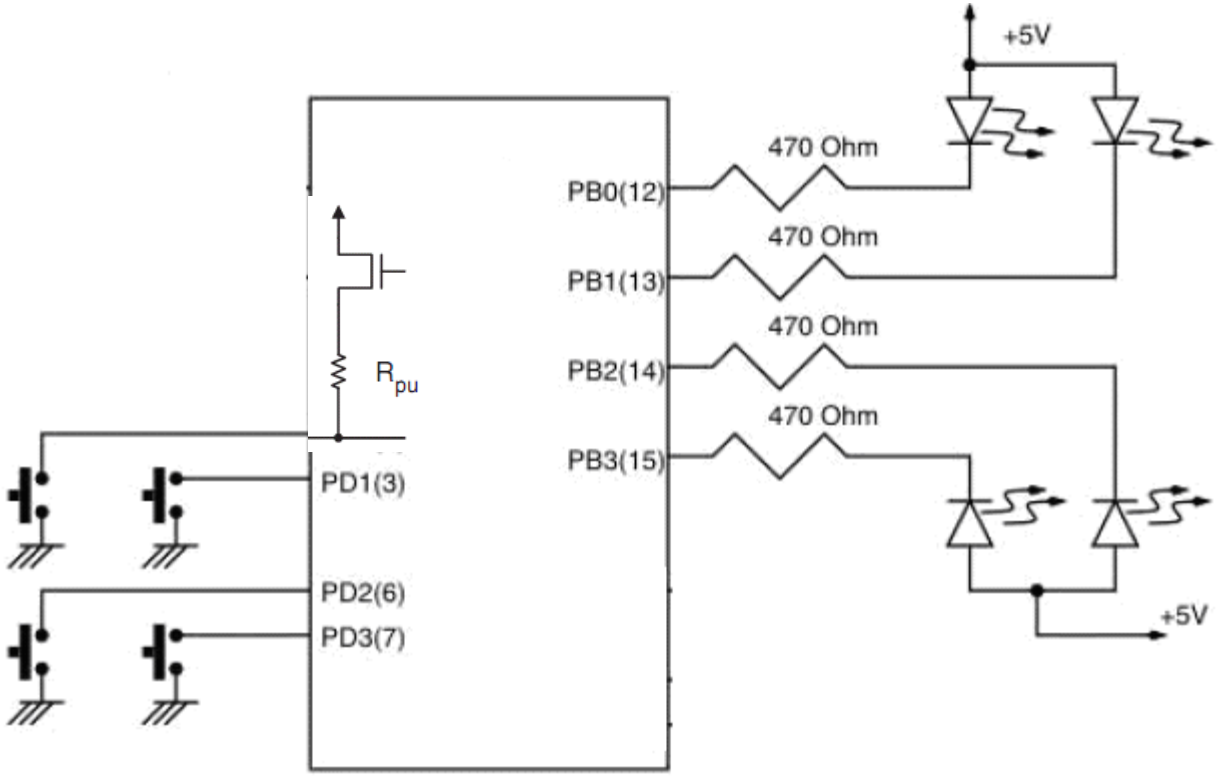
```
in r17, MCUCR
andi r17, 0b11101111
out MCUCR, r17
```

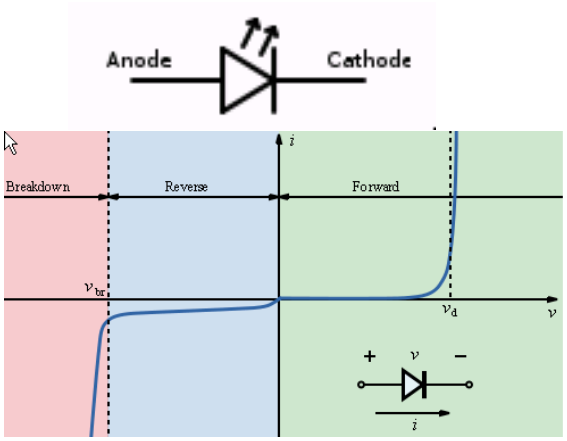
```
MCUCR &= 0b11101111;
```

Input / Output ports

Example 1 – BTNs & LEDs

- "Pull-up" resistors are providing '1' on the pin (Pxn) when the push-button is not pressed (Pxn – high/*source*)
- When the button is pressed, the pin (Pnx) level is '0' by the connection to GND (Pxn = high/*sink*)
- A level of '0' on the output pins (port B) determine the LEDs to lit
- A level of '1' on the B pins turns off the LEDs





http://en.wikipedia.org/wiki/Light-emitting_diode

Input / Output ports

Example 1 – BTNs & LEDs

```
in r17, MCUCR
andi r17, 0b11101111
out MCUCR, r17        ; pull-up enable: PUD = 0;

ldi r16, 0x00
out DDRD, r16         ; Set Direction of port D: input

ldi r16, 0x0F         ; '1' to PORTD: pull-up resistors are activated
out PORTD, r16

ldi r16, 0xFF
out DDRB, r16         ; Set Direction of port B - output

loop:
    in r16, PIND      ; Read from port D
    out PORTB, r16    ; Write to port B (LEDs are "set" until the next
                      ;                  out PORTB, r16 )
rjmp loop
```

Input / Output ports

Example 1 – BTNs & LEDs

```
byte val;

void setup() {
  MCUCR &= 0b11101111; // pull up enable: PUD=0
  DDRD = 0x00; // Set Direction of port D: input
  PORTD = 0x0F; // Activates pullup resist.(pins 0 ..3)
  DDRB = 0xFF; // Set Direction of port B: output
}
void loop() {
  val = PIND; // Read input from buttons (PORTD)
  PORTB=val;  // Write to port B (activate LEDs)
  // or you can use: PORTB=PIND
}
```

- **Attention**

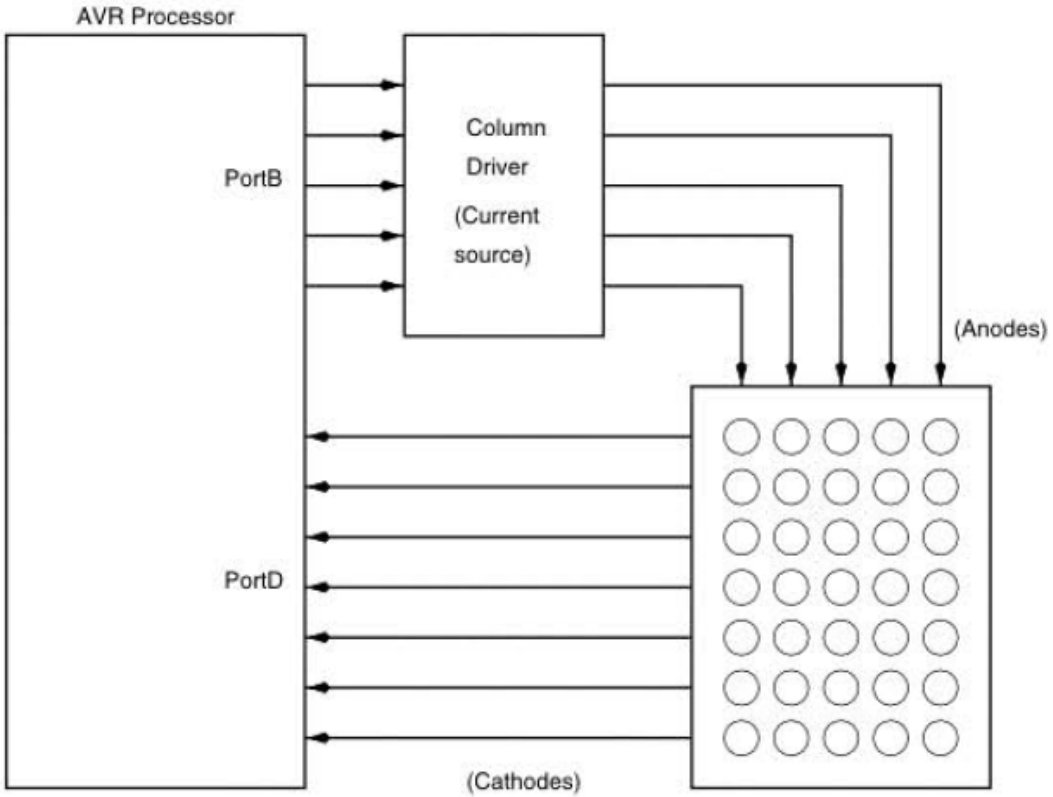
`val = PIND` Read the status of the external pins changed by the previous activity (INPUT value from port)

`val = PORTD` Reads the status of PORTD (ex: pull-up resistors activation status)

Input / Output ports

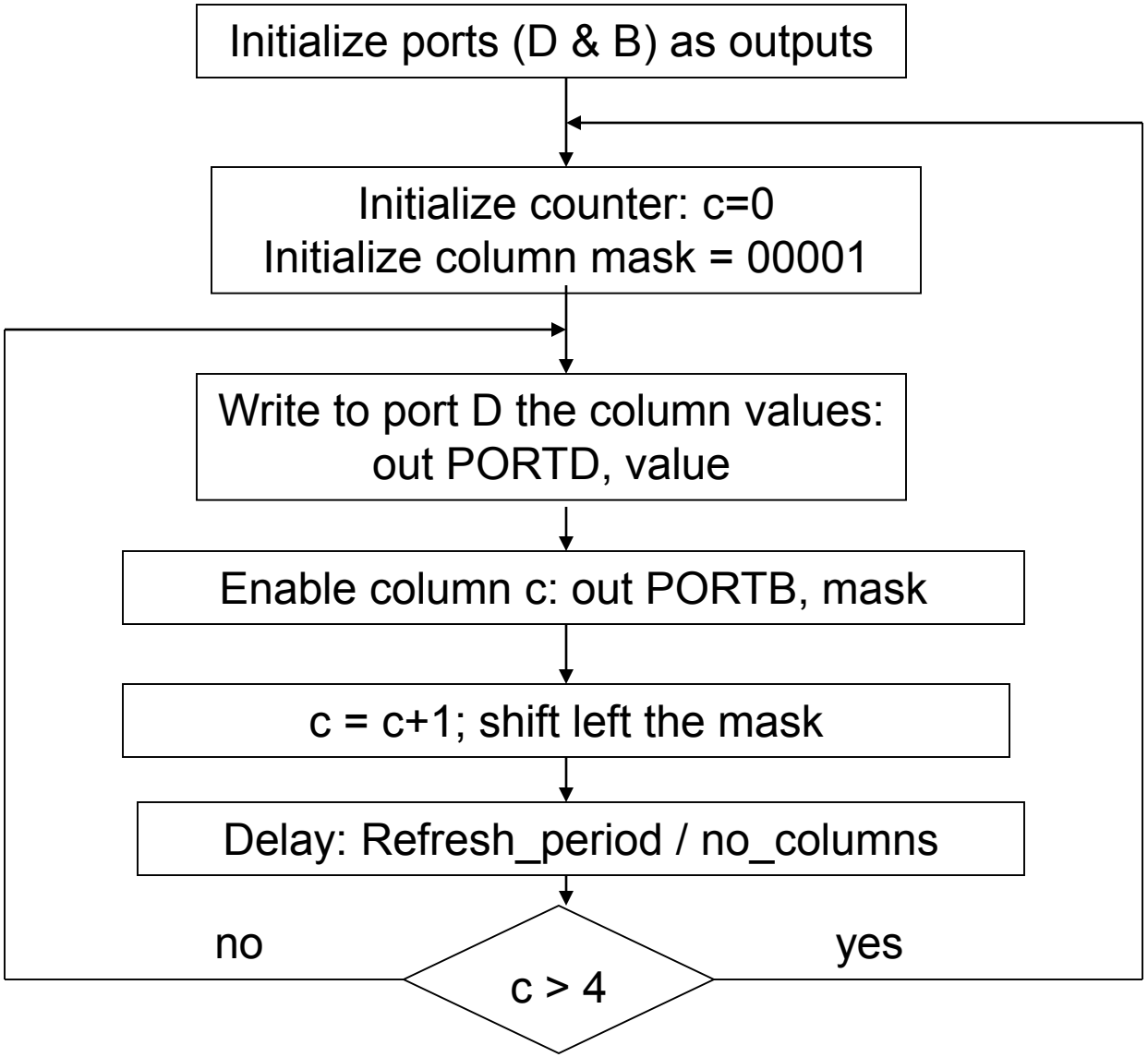
Example 2 – LEDs matrix display

- Both ports (D & B) are outputs
- To lit up a LED, to its “anode” a value of ‘1’ (Vcc) should be applied and to its cathode a value of ‘0’ (Gnd) should be applied
- Only one row or one column can be lit at a time
- For displaying on the whole matrix, the rows or columns are lit up in successive order with a specified frequency (Refresh_period:1 ms … 16 ms)



Input / Output ports

Example 2 – LEDs matrix display (column shifting approach)



Input / Output ports

Example 2 – LEDs matrix display

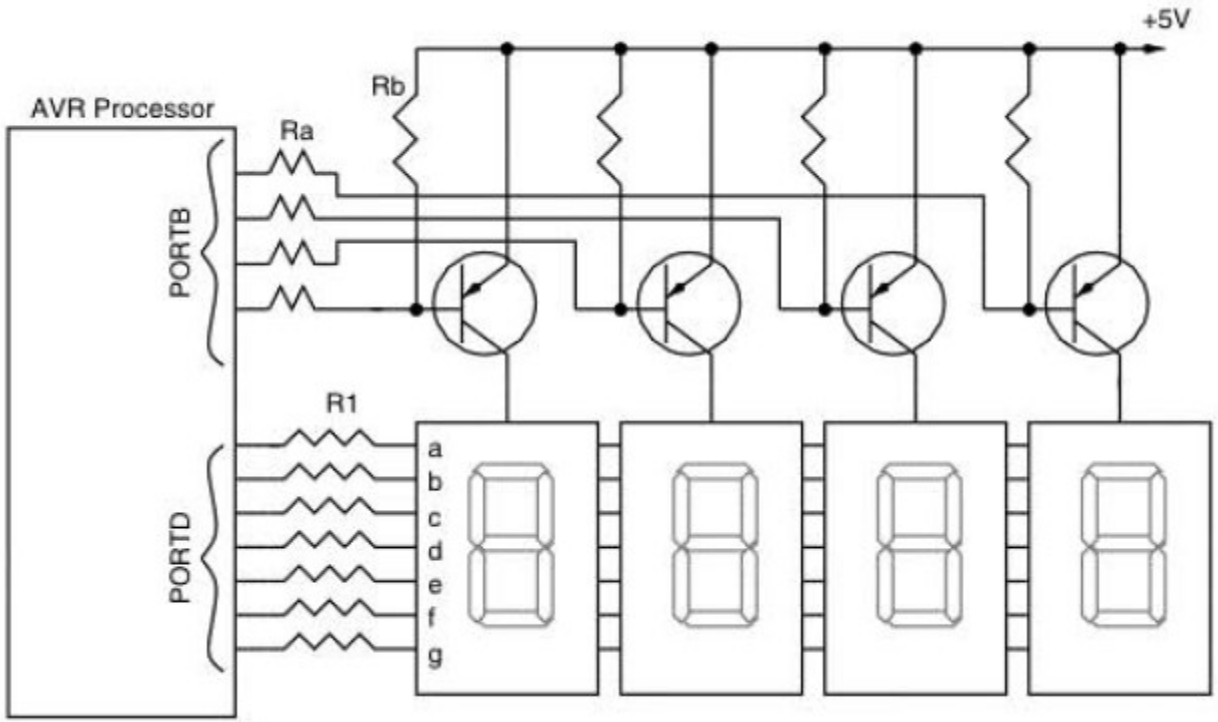
- Homework: write the above program in AVR assembly, then change the approach (use row shifting) and write the updated program in AVR assembly & C

```
byte mask; // column mask
byte c;   //  column counter
byte happy[5] = { // Each value (byte) is a column
        B0110010,B0110001,B0000001,B0110001,B0110010};
void setup() {
  DDRD = 0xFF; // set direction of port D: output
  DDRB = 0xFF; // set direction of port B: output
}
void loop() {
  c=0;
  mask= 1<<c;
   while (c <= 4){
    PORTD = happy[c]; // write to port D the column values
    PORTB = mask; // enable column c
    c++; // increment the column cunter
    mask = 1<<c; // left shift the mask
    delay(2); // Matrix refresh period is 5x2=10 ms (100Hz)
   }
}
```

Input / Output ports

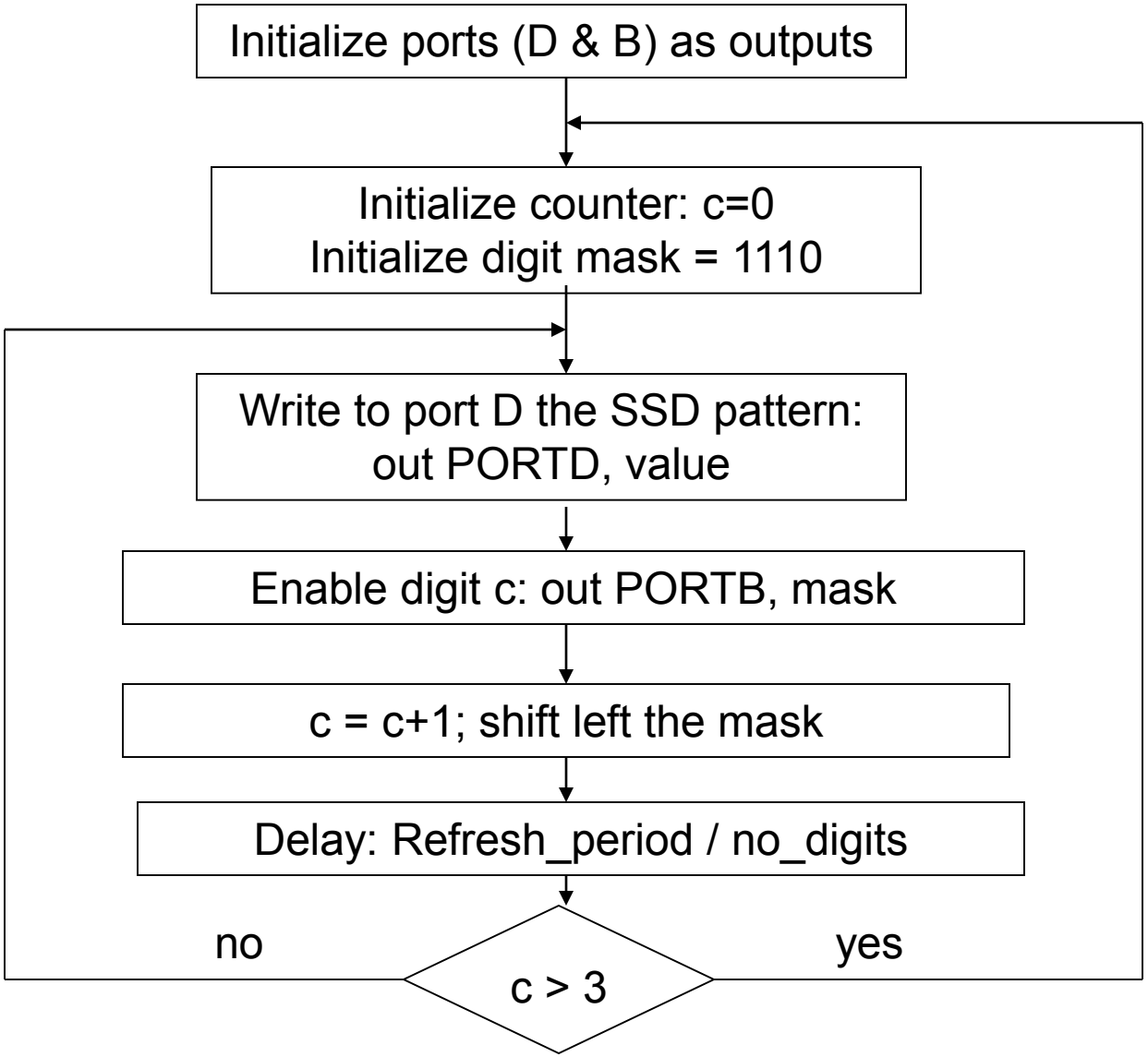
Example 3 – Block of 4 x SSD

- Each digit (SSD) has 7 LEDs, with common anode
- Level ‘1’ on the anode activates the digit (only one digit can be lit at a time)
- Values of ‘0’ on each LED cathode will lit up the corresponding segment of the digit (SSD)
- For displaying on the whole SSD block, digits are activated in successive order with a specified frequency (Refresh_period:1 ms … 16 ms)



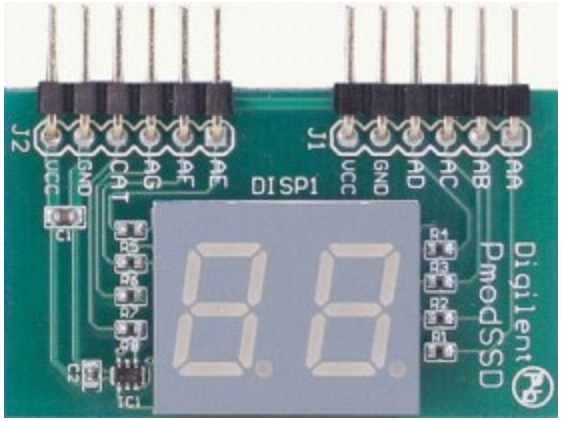
Input / Output ports

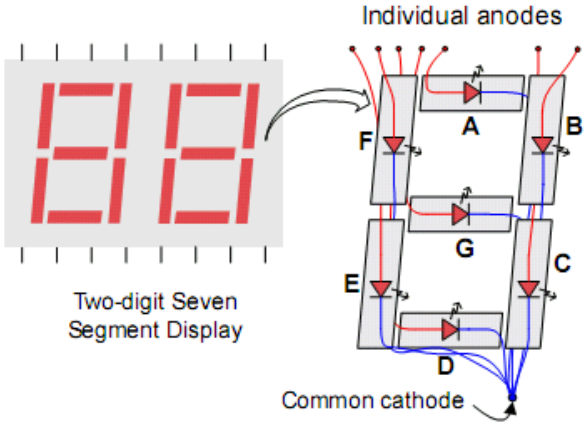
•Example 3 – Block of 4 x SSD

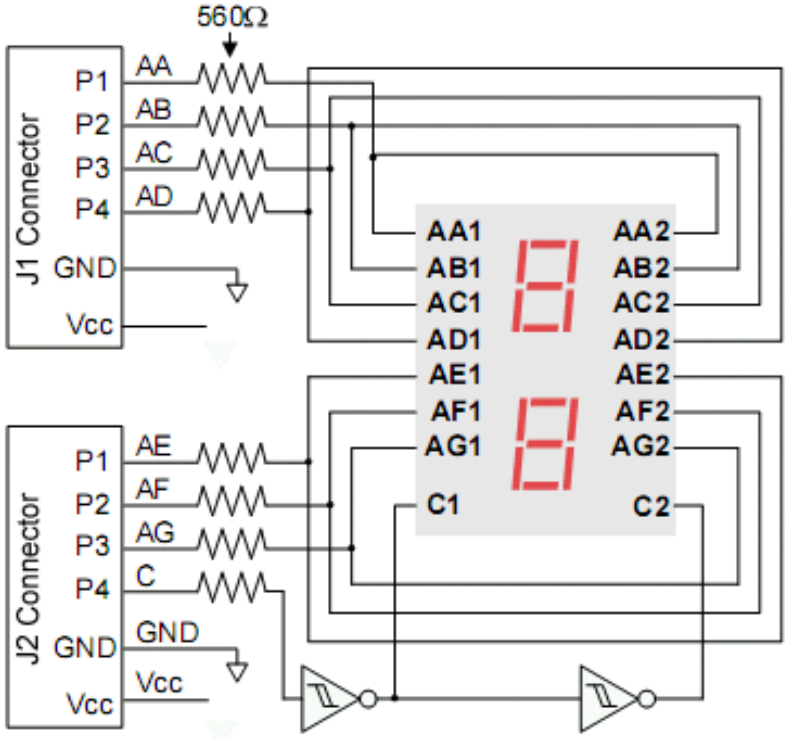


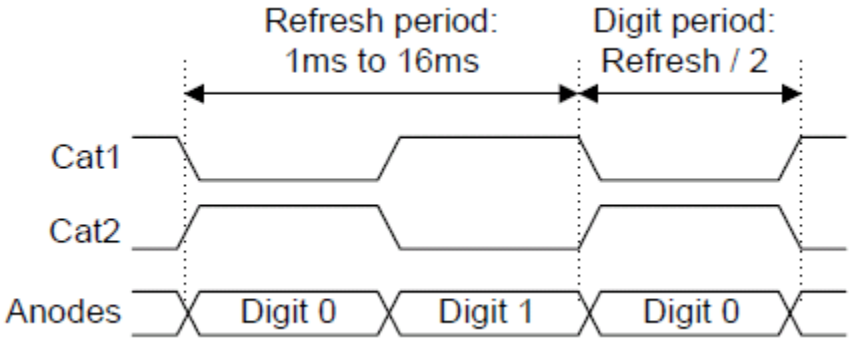
Input / Output ports

Example 4 – Digilent PMOD SSD (2x7 segments)









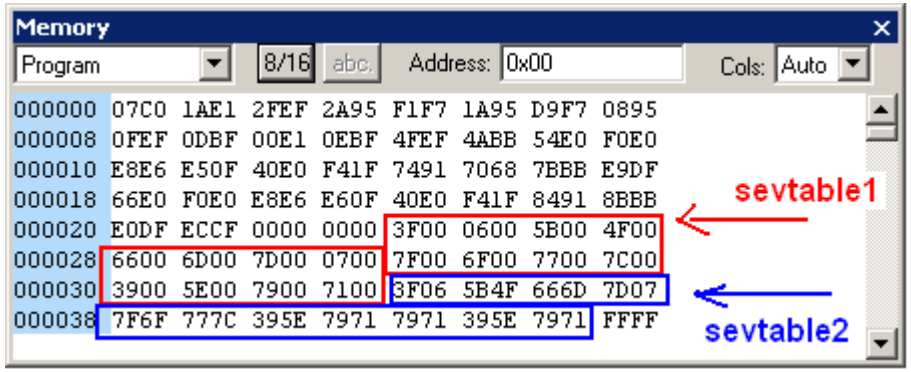
SSD example (1)

Example 4 – The use of the Digilent PMOD SSD (2x7 segments)

Write a program to display a 2 digit number using the PMOD-SSD connected to PORTA of an AVR microcontroller

AVR Assembly code

```
sevstable1:  ; code table in program memory for LED anodes (option 1)
;                    Cgfedbca
            .db    0b00111111      ; (0x3F) 0
            .db    0b00000110       ; (0x06) 1
            .db    0b01011011       ; (0x5B) 2
            .db    0b01001111       ; (0x4F) 3
            .db    0b01100110       ; (0x66) 4
            .db    0b01101101       ; (0x6D) 5
            .db    0b01111101       ; (0x7D) 6
            .db    0b00000111       ; (0x07) 7
            .db    0b01111111      ; (0x7F) 8
            .db    0b01101111       ; (0x6F) 9
            .db    0b01110111       ; (0x77) A
            .db    0b01111100       ; (0x7C) b
            .db    0b00111001       ; (0x39) C
            .db    0b01011110       ; (0x5E) d
            .db    0b01111001       ; (0x79) E
            .db    0b01110001       ; (0x71) F
```




```
sevstable2:  ; code table in program memory for LED anodes  (option 2)
.db 0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71
```

SSD example (2)

```
.org 0x0000
     rjmp main
.def temp = r20
.def index0 = r21              // counter for accessing bytes in program memory
.def index1 = r22              // counter for accessing words in program memory
.def digit0 = r23
.def digit1 = r24
.set tab_size = 16

// macro used to  read a values from vector/table @0 (stored in program memory), index @1. output is placed in @2
.macro rdb     ; memory_address, offset, output register
    ldi zh, high(2*@0)    //load higher bits of 2*memory_address
    ldi zl, low(2*@0)     //load lower bits of 2*memory_address
    add zl, @1            //adds @1 (offset) to zl
    ldi temp,0            //resets temp
    adc zh, temp          //adds carry to zh
    lpm @2, Z             //loads memory location Z into output
.endmacro

delay_refresh:
// performs a 5 ms delay
     ldi r17, 26
     loop1:
                ldi r18, 255
                loop2:
                           dec r18
                           brne loop2
                dec r17
                brne loop1
     ret
```

SSD example (3)

```
main:
        ldi r16, low(RAMEND)
        out SPL, r16
        ldi r16, high(RAMEND)
        out SPH, r16

        //SSD connected to PORTA
        ldi temp, 0xFF
        out DDRA, temp

        loop:
        // infinite loop that displays number 46 on  the SSD
                ldi index0, 4
                rdb sevstable2, index0, digit0
                ori digit0, 0x80
                out PORTA, digit0
                rcall delay_refresh

                ldi index1, 6
                rdb sevstable2, index1, digit1
                out PORTA, digit1
                rcall delay_refresh

                rjmp loop
```

Homework: modify the above example in order to use *sevtable1* as LUT

SSD example for ATMega 2560 (4)

Display on SSD a constant number (ex: 46)connected at PORTA (C code)

```
// Table of values, or look up table (LUT) with the BCD codes for every digit from 0 to 9.
//Every bit corresponds to a LED, 1 means the LED is lit and 0 means its not.
const unsigned char ssdlut[] =
{0b00111111,0b00000110,0b01011011,0b01001111,0b01100110,0b01101101,0b01111101,0b00000111,0b
01111111,0b01101111};

unsigned char outvalue; // cathod mask

void setup() {
  // setting port A as output
  DDRA = 0xFF;
}

void loop() {
  outvalue = 0x80; // select digit 0 (units)
  PORTA = (ssdlut[6] | outvalue);
// we make an OR between the value from the LUT and the selected cathode
// the cathode is wired to bit7 from port A
// through this operation we are setting bit 7 on logical 0 or 1
// the following bits (0 .. 6 - LEDs)will be attached through a logical OR operation:
  delay(5); //delay 5 ms between switching the digit

  outvalue = 0x00; // select digit 1 (decimals)
  PORTA = (ssdlut[4] | outvalue);
// we make an OR between the value from the LUT and the selected cathode
  delay(5); //delay 5 ms between switching the digit

  // total refresh time of the SSD is 10 ms (100 HZ) for eye confort
}
```

Alternative I/O functions

Alternate functions of the I/O pins of AVR MCU

- Most port pins have alternate functions in addition to being general digital I/Os
- How each alternate function interferes with the port pin is described in “Alternate Port Functions” section in the MCU datasheet.
- Refer to the individual module sections for a full description of the alternate functions.
- Note that enabling the alternate function of some of the port pins does not affect the use of the other pins as general digital I/O.

Example for port B (ATmega 2560):

Port Pin	Alternate Functions
PB7	OC0A/OC1C/PCINT7 (Output Compare and PWM Output A for Timer/Counter0, Output Compare and PWM Output C for Timer/Counter1 or Pin Change Interrupt 7)
PB6	OC1B/PCINT6 (Output Compare and PWM Output B for Timer/Counter1 or Pin Change Interrupt 6)
PB5	OC1A/PCINT5 (Output Compare and PWM Output A for Timer/Counter1 or Pin Change Interrupt 5)
PB4	OC2A/PCINT4 (Output Compare and PWM Output A for Timer/Counter2 or Pin Change Interrupt 4)
PB3	MISO/PCINT3 (SPI Bus Master Input/Slave Output or Pin Change Interrupt 3)
PB2	MOSI/PCINT2 (SPI Bus Master Output/Slave Input or Pin Change Interrupt 2)
PB1	SCK/PCINT1 (SPI Bus Serial Clock or Pin Change Interrupt 1)
PB0	$\overline{SS}$/PCINT0 (SPI Slave Select input or Pin Change Interrupt 0)

Alternative I/O functions

Alternate functions of the I/O pins of AVR MCU

- Most port pins have alternate functions in addition to being general digital I/Os
- How each alternate function interferes with the port pin is described in “Alternate Port Functions” section in the MCU datasheet.
- Refer to the individual module sections for a full description of the alternate functions.
- Note that enabling the alternate function of some of the port pins does not affect the use of the other pins as general digital I/O.

Example for port B (ATmega 2560):

Port Pin	Alternate Functions
PB7	OC0A/OC1C/PCINT7 (Output Compare and PWM Output A for Timer/Counter0, Output Compare and PWM Output C for Timer/Counter1 or Pin Change Interrupt 7)
PB6	OC1B/PCINT6 (Output Compare and PWM Output B for Timer/Counter1 or Pin Change Interrupt 6)
PB5	OC1A/PCINT5 (Output Compare and PWM Output A for Timer/Counter1 or Pin Change Interrupt 5)
PB4	OC2A/PCINT4 (Output Compare and PWM Output A for Timer/Counter2 or Pin Change Interrupt 4)
PB3	MISO/PCINT3 (SPI Bus Master Input/Slave Output or Pin Change Interrupt 3)
PB2	MOSI/PCINT2 (SPI Bus Master Output/Slave Input or Pin Change Interrupt 2)
PB1	SCK/PCINT1 (SPI Bus Serial Clock or Pin Change Interrupt 1)
PB0	$\overline{SS}$/PCINT0 (SPI Slave Select input or Pin Change Interrupt 0)

Interrupts

- **Interrupt sources are presented in the Interrupt Vector table**
- Interrupts can be accepted after the finish of the currently executed instruction
- Response time:> = 4 .. 5 cycles

  PC (2/3 bytes) saved on the stack (push)

  Stack Pointer ← Stack Pointer – 2/3;

  Jump according to the interrupt vector table

  Interrupt system is blocked: bit I, SREG(7) ← 0
- After 4..5 cycles begins the execution of the ISR (optimal case)
- If the interrupt is used for waking up from sleep mode, the response time is increased with 4..5 cycles
- Returning from ISR (RETI): 4..5 cycles

  PC ← PC saved (pop from stack)

  Stack Pointer ← Stack Pointer + 2/3;

  Interrupt system is enabled: bit I, SREG(7) ← 1
- Bit I can be set/reset directly through instructions SEI & CLI
- **Priority:** decreases with the increasing of the interrupt number
- **Maximum priority : Reset**

Interrupt Vectors in ATMega 2560

http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

Table 14-1. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016(3)	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B

Interrupt Vectors in ATMega 2560

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega2560 is:

Address	Labels	Code Comments
0x0000	jmp RESET	; Reset Handler
0x0002	jmp INT0	; IRQ0 Handler
0x0004	jmp INT1	; IRQ1 Handler
0x0006	jmp INT2	; IRQ2 Handler
0x0008	jmp INT3	; IRQ3 Handler
0x000A	jmp INT4	; IRQ4 Handler
0x000C	jmp INT5	; IRQ5 Handler
0x000E	jmp INT6	; IRQ6 Handler
0x0010	jmp INT7	; IRQ7 Handler
0x0012	jmp PCINT0	; PCINT0 Handler
0x0014	jmp PCINT1	; PCINT1 Handler
0x0016	jmp PCINT2	; PCINT2 Handler
0X0018	jmp WDT	; Watchdog Timeout Handler
0x001A	jmp TIM2_COMPA	; Timer2 CompareA Handler
0x001C	jmp TIM2_COMPB	; Timer2 CompareB Handler
0x001E	jmp TIM2_OVF	; Timer2 Overflow Handler
0x0020	jmp TIM1_CAPT	; Timer1 Capture Handler

External Interrupts

External Interrupts are triggered by the INT7:0 pin or any of the PCINT23:0 pins.

If enabled, the interrupts will trigger even if the INT7:0 or PCINT23:0 pins are configured as outputs. This feature provides a way of generating a software interrupt.

EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

EICRB – External Interrupt Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x6A)	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

External Interrupts

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – INT7:0: External Interrupt Request 7 - 0 Enable**

When an INT7:0 bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Registers – EICRA and EICRB – defines whether the external interrupt is activated on rising or falling edge or level sensed.

EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – INTF7:0: External Interrupt Flags 7 - 0**

When an edge or logic change on the INT7:0 pin triggers an interrupt request, INTF7:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT7:0 in EIMSK, are set (one), the MCU will jump to the interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. These flags are always cleared when INT7:0 are configured as level interrupt. Note that

External Interrupts

Example 5: binary counter incremented by an external interrupt (generated by a button connected INT0 / pin PD0). Output counter value on LEDs connected to PORTA

```
.org 0x0000 ; vector address for reset interrupt
        rjmp main
.org 0x0002 ; vector address of the ISR for external interrupt INT0
        rjmp isr_INT0 ;
main:
        ldi r16, high(RAMEND) ; initialize the stack – necessary for interrupts
        out SPH, R16
        ldi r16, low(RAMEND)
        out SPL, R16

        ldi r16, 0b00000011 ; configure INT0 activation mode (rising edge)
        sts EICRA, r16      ; EICRA address = 0x006A > 0x0060
        ldi r16, 0b00000001 ; enabling INT0
        out EIMSK, r16      ; EICRA address = 0x0059 < 0x0060

        ldi r16, 0xFF ; set direction of port A as output (binary counter displayed on LED's)
        out DDRA, r16
        ldi r17,0 ; initial value of the counter
```

External Interrupts

Example 5: binary counter incremented by an external interrupt (generated by a button connected INT 0 pin – PD0), continued …

```
            sei ; global activation of the interrupt system
loop:
            out PORTA, r17 ; output the value of the counter to port E (LEDs)
            rjmp loop

isr_INT0: ; INT0 ISR (Interrupt Service Routine for external interrupt 0)
            inc r17 ; increment the counter
            reti ; return from interrupt
```

External Interrupts

Example 6: Modify the previous example in order to use two buttons connected to INT1 and INT2. The INT1 should increment the counter by 2, while the INT2 should decrement the counter by one. Write thecode in AVR assembly.

Hint (how to link the ISR address to the interrupt vector table) in AVR assambly:

```
.org 0x0000 ; vector address for reset interrupt
        rjmp main
.org 0x0004 ; vector address of the ISR for external interrupt INT1
        rjmp isr_INT1
 .org 0x0006 ; vector address of the ISR for external interrupt INT2
        rjmp isr_INT2
```

External Interrupts

Example 6: C implementation using AVR specific instructions (Arduino)

```
// Include the header for the avr interrupt system
#include "avr/interrupt.h"

volatile byte counter; // 8 bit counter - public variable shared between
the ISR and the main program

void setup(void)
{
  counter = 0;
  DDRA = 0xFF; // Output the counter value to port A (ex. drive LEDs)

// Set pin 21 as input (the pin corresponding to INT0)
  pinMode(21 ,INPUT);
// Set pin 20 as input (the pin corresponding to INT1)
  pinMode(20, INPUT);

  EIMSK |= (1 << INT0); // Activate INT0
  EIMSK |= (1 << INT1); // Activate INT1

  EICRA |= (1 << ISC01); // Specify INT0 triggering behavior: falling edge
  EICRA |= (1 << ISC11); // Same behavior for INT1
  sei(); // Global interrupt system activation
}
```

External Interrupts

Example 6: C implementation using MCU specific instructions – cont.

```
void loop()
{
  // PORTA = counter : the inefficient way to update the LEDs
  // Better do something else ...
}

// ISR for INT0 ("INT0_vect" is a predefined name (address) for INT0 ISR
ISR(INT0_vect)
{
  counter = counter + 2;
  PORTA = counter; // Update the LEDs
}
// ISR for INT1
ISR(INT1_vect)
{
  counter-- ; // Decrement the counter
  PORTA = counter; // Update the LEDs
}
```

External Interrupts

Example 6: C implementation using Arduino IDE generic functions

```
// Include the header for the avr interrupt system
#include "avr/interrupt.h"
volatile byte counter; // 8 bit counter - public variable
void setup(void)
{
// The 2 interrupt pins 21 and 20 declared as inputs with pull-up resistors activated
  pinMode(20 ,INPUT);
  pinMode(21 ,INPUT);
  digitalWrite(20, HIGH); // activate pull-up resistors
  digitalWrite(21, HIGH);

// Atach ISRs to the interrupts corresponding to pins 21 and 20 (INT0 and INT1)
  attachInterrupt(digitalPinToInterrupt(21), my_INT0, FALLING);
  attachInterrupt(digitalPinToInterrupt(20), my_INT1, FALLING);
}

void loop()
{
}

void my_INT0()// ISR for INT0
{
  counter = counter + 2;
  PORTA = counter; // Update the LEDs
}

void my_INT1() // ISR for INT1
{
  counter-- ; // Decrement the counter
  PORTA = counter; // Update the LEDs
}
```