CHW 469 : Embedded Systems

Instructor:

Dr. Ahmed Shalaby

http://bu.edu.eg/staff/ahmedshalaby14#

Interrupt
Chapter 10

The AVR microcontroller
and embedded
systems
using assembly and c

Contents

- Polling Vs. interrupt
- Interrupt unit
- Steps in executing an interrupt
- Edge trigger Vs. Level trigger in external interrupts
- Timer interrupt
- Interrupt priority
- Interrupt inside an interrupt
- Task switching and resource conflict
- C programming

Polling Vs. Interrupt

- Polling
  - Ties down the CPU

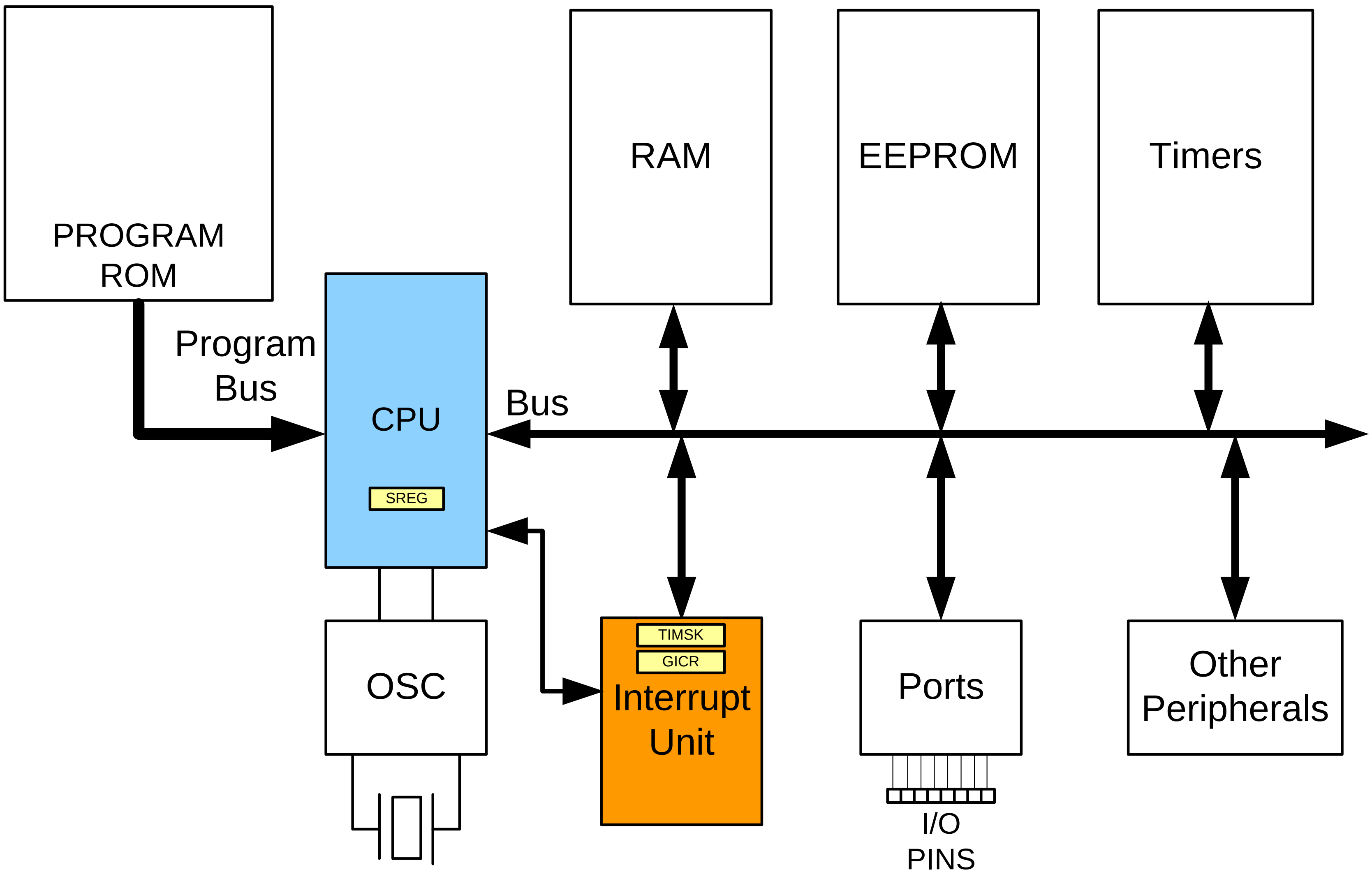
```
while (true)
{
    if(PIND.2 == 0)
        //do something;
}
```

- Interrupt
  - Efficient CPU use
  - Has priority
  - Can be masked

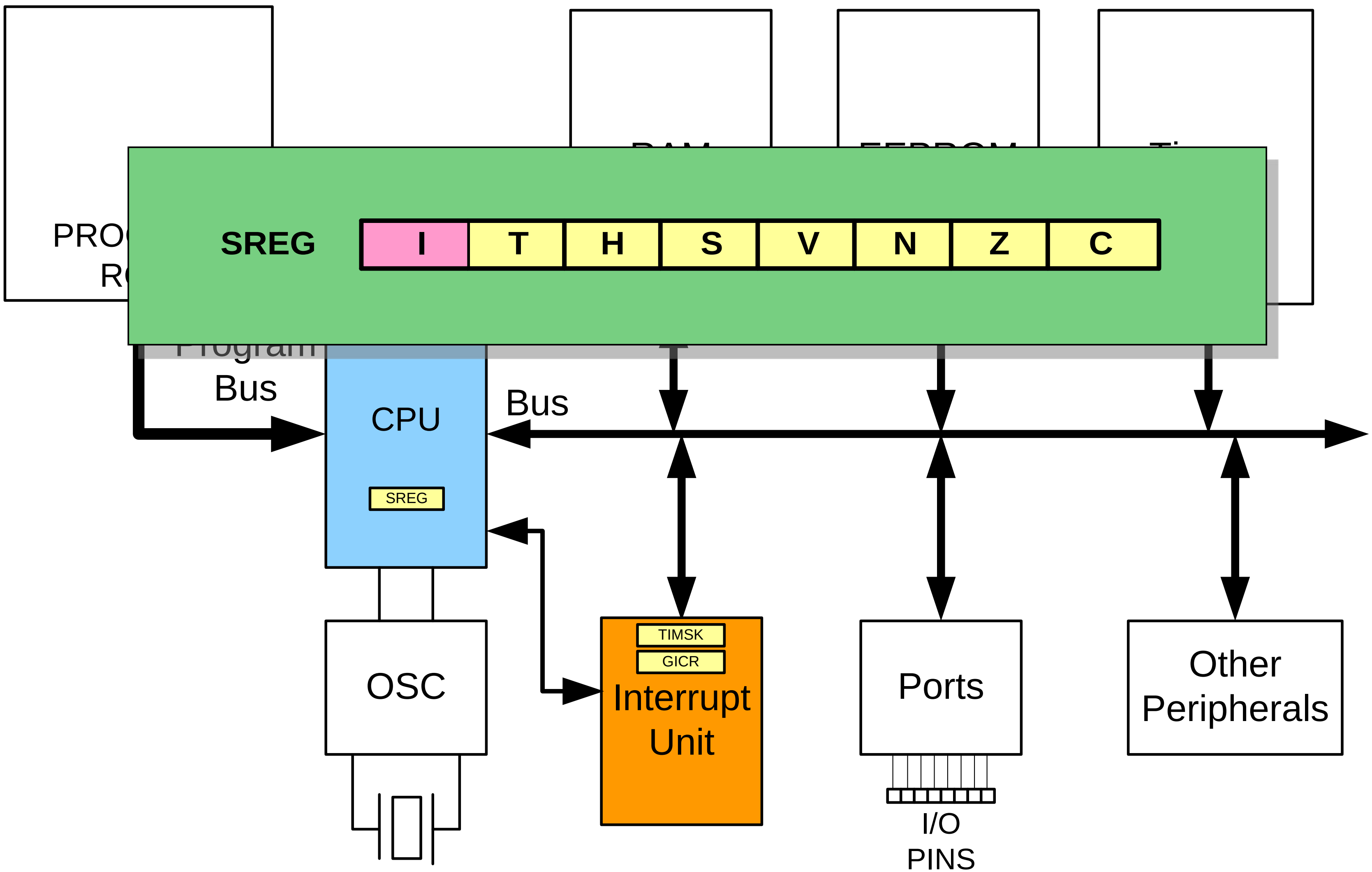
```
main( )
{
    Do your common task
}

whenever PIND.2 is 0 then
    do something
```

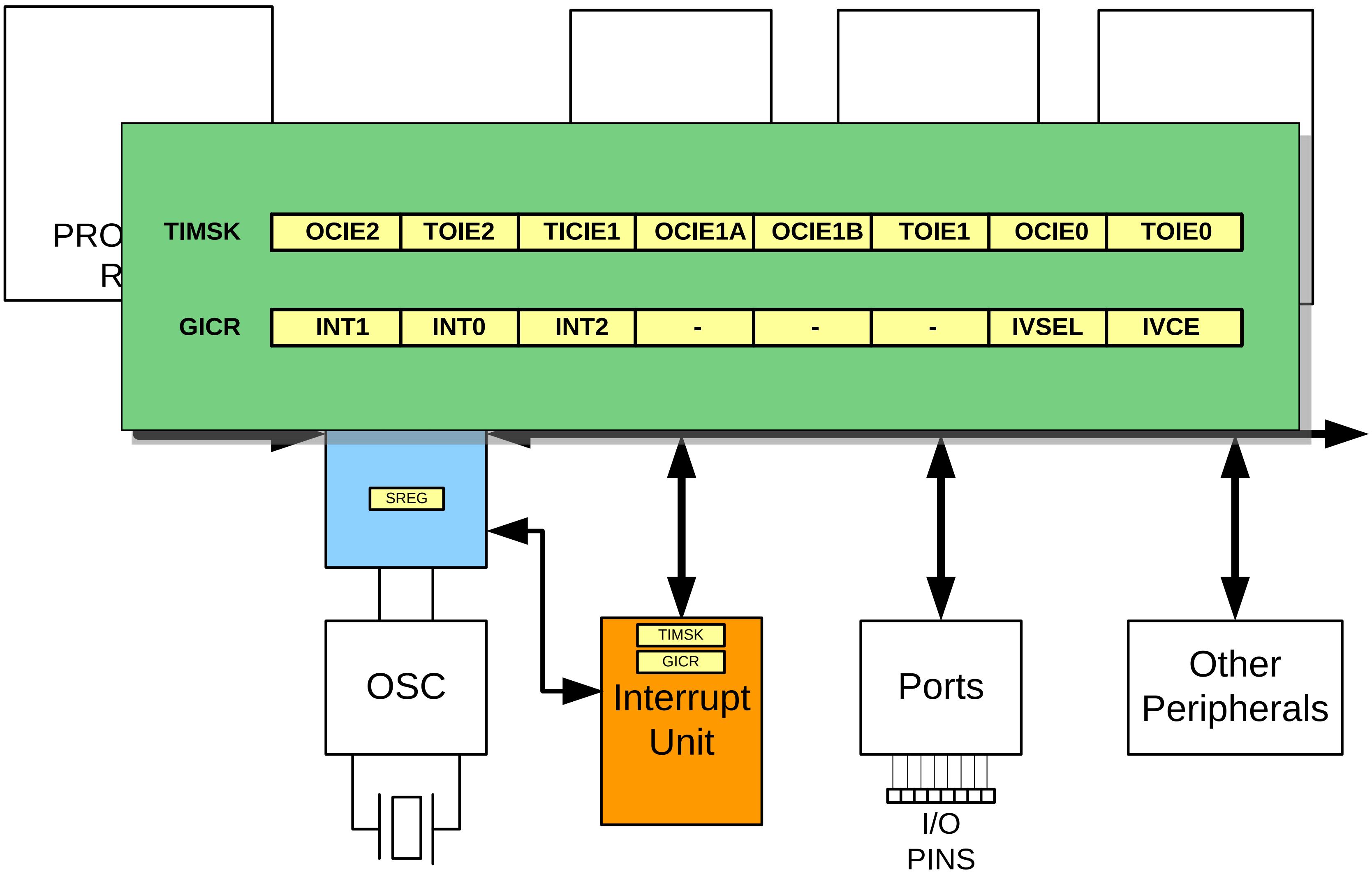
Interrupt unit



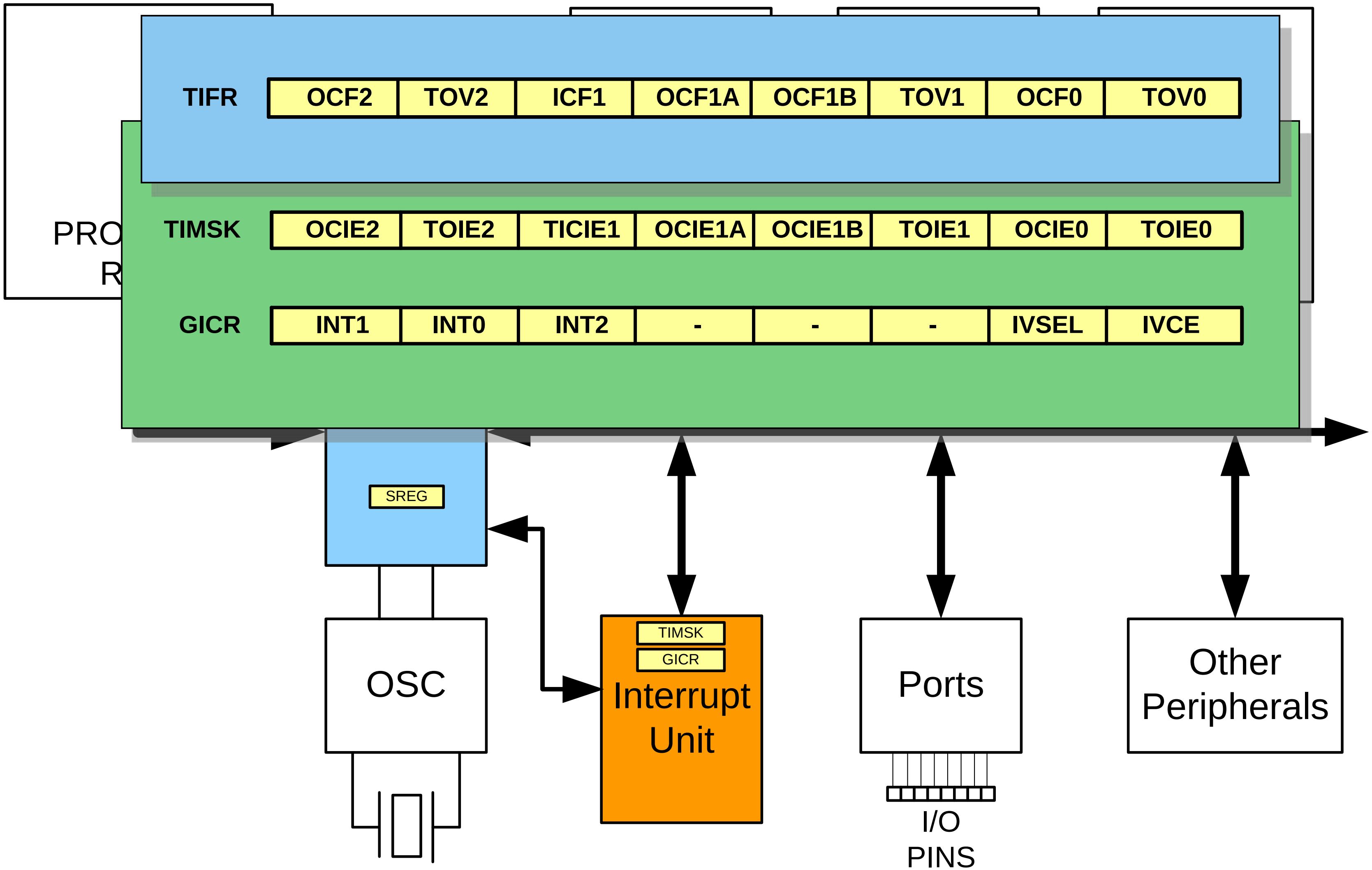
Interrupt unit



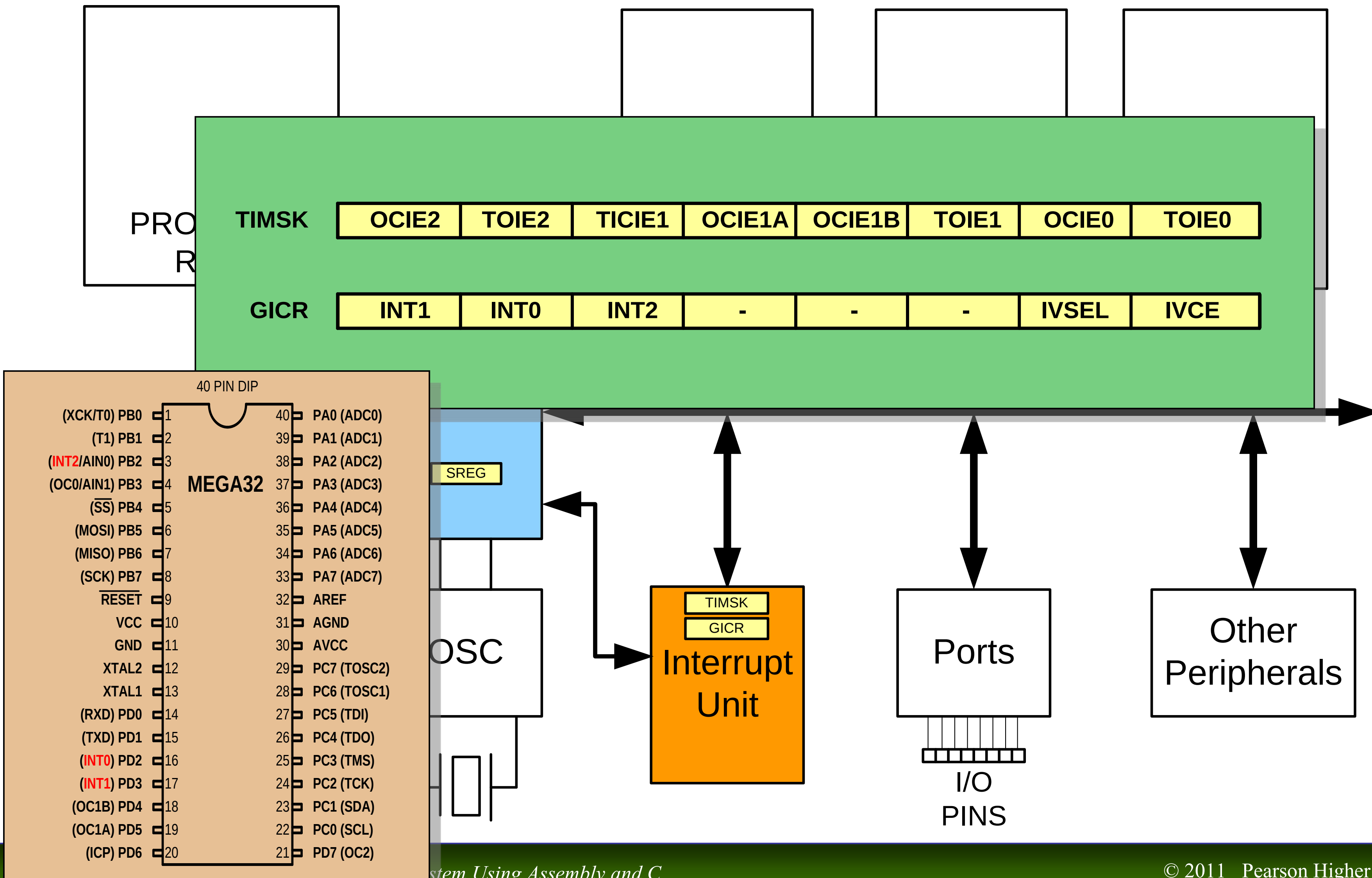
Interrupt unit



Interrupt unit



Interrupt unit



Interrupt unit

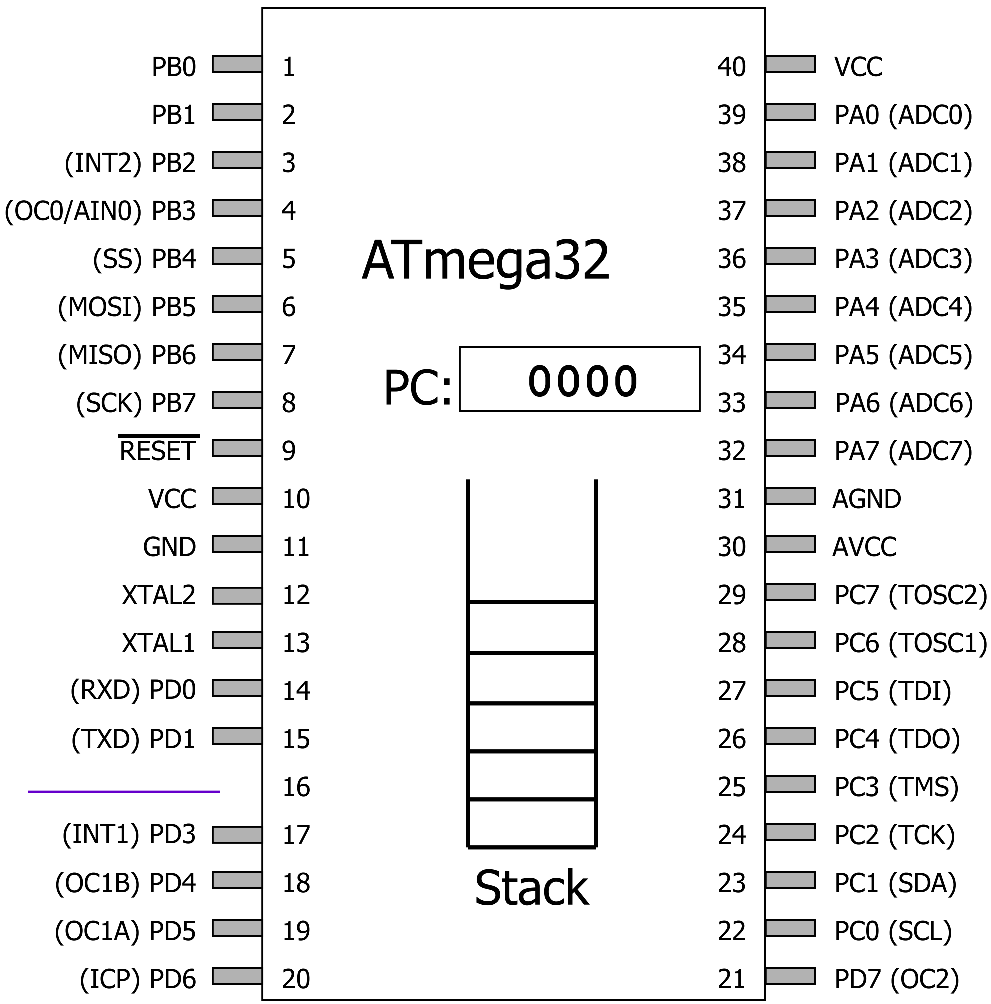
PROGRA
ROM

PINS

Table 10-1: Interrupt Vector Table for the Mega32

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028

Steps in executing an interrupt

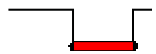
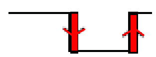
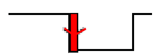
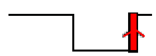


Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

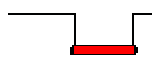
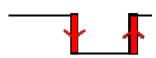
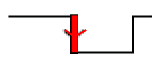
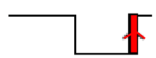
Edge trigger Vs. Level trigger in external interrupts

MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

ISC01, ISC00 (Interrupt Sense Control bits) These bits define the level or edge on the external INT0 pin that activates the interrupt, as shown in the following table:

ISC01	ISC00		Description
0	0		The low level of INT0 generates an interrupt request.
0	1		Any logical change on INT0 generates an interrupt request.
1	0		The falling edge of INT0 generates an interrupt request.
1	1		The rising edge of INT0 generates an interrupt request.

ISC11, ISC10 These bits define the level or edge that activates the INT1 pin.

ISC11	ISC10		Description
0	0		The low level of INT1 generates an interrupt request.
0	1		Any logical change on INT1 generates an interrupt request.
1	0		The falling edge of INT1 generates an interrupt request.
1	1		The rising edge of INT1 generates an interrupt request.

Edge trigger Vs. Level trigger in external interrupts

MCUCR							
SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

ISC01, ISC00 (Interrupt Sense Control bits) These b... external INT0 pin that activates the interrupt, as shown i...

```
LDI     R20,0x02     ;falling
OUT     MCUCR,R20
```

ISC01	ISC00		
0	0		The low level of INT0 generates an interrupt request.
0	1		Any logical change on INT0 generates an interrupt request.
1	0		The falling edge of INT0 generates an interrupt request.
1	1		The rising edge of INT0 generates an interrupt request.

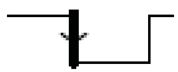
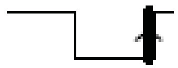
ISC11, ISC10 These bits define the level or edge that activates the INT1 pin.

ISC11	ISC10		Description
0	0		The low level of INT1 generates an interrupt request.
0	1		Any logical change on INT1 generates an interrupt request.
1	0		The falling edge of INT1 generates an interrupt request.
1	1		The rising edge of INT1 generates an interrupt request.

Edge trigger Vs. Level trigger (Cont.)

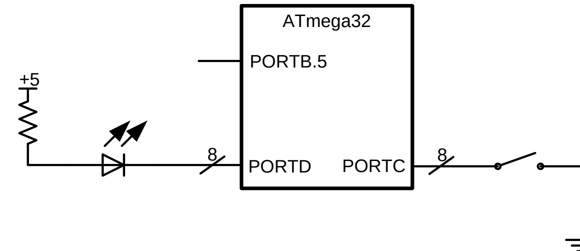
MCUCSR	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF

ISC2 This bit defines whether the INT2 interrupt activates on the falling edge or the rising edge.

ISC2		Description
0		The falling edge of INT2 generates an interrupt request.
1		The rising edge of INT2 generates an interrupt request.

Using Timer0 overflow interrupt

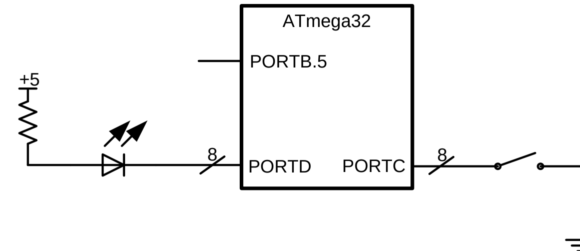
- This program uses Timer0 to generate a square wave on pin PORTB.5, while at the same time data is being transferred from PORTC to PORTD.




```
;Program 10-1
.INCLUDE "M32DEF.INC"
.ORG        0x0         ;location for reset
            JMP         MAIN
.ORG        0x16        ;loc. for Timer0 over.
            JMP         T0_OV_ISR
;----main program for initialization
.ORG        0x100
MAIN:       LDI         R20,HIGH(RAMEND)
            OUT         SPH,R20
            LDI         R20,LOW(RAMEND)
            OUT         SPL,R20
            SBI         DDRB,5    ;output
            LDI         R20,0
            OUT         DDRC, R20
            LDI         R20,0xFF
            OUT         DDRD, R20
            LDI         R20,(1<<TOIE0)
            OUT         TIMSK,R20
            SEI
            LDI         R20,-32   ;value for 4µs
            OUT         TCNT0,R20
            LDI         R20,0x01
            OUT         TCCR0,R20
HERE:       IN          R20,PINC
            OUT         PORTD,R20
            JMP         HERE
;--------ISR for Timer 0
T0_OV_ISR:
            IN          R16,PORTB
            LDI         R17,0x20
            EOR         R16,R17
            OUT         PORTB,R16
            RETI
```

Using Timer0 overflow interrupt

- This program uses Timer0 to generate a square wave on pin PORTB.5, while at the same time data is being transferred from PORTC to PORTD.



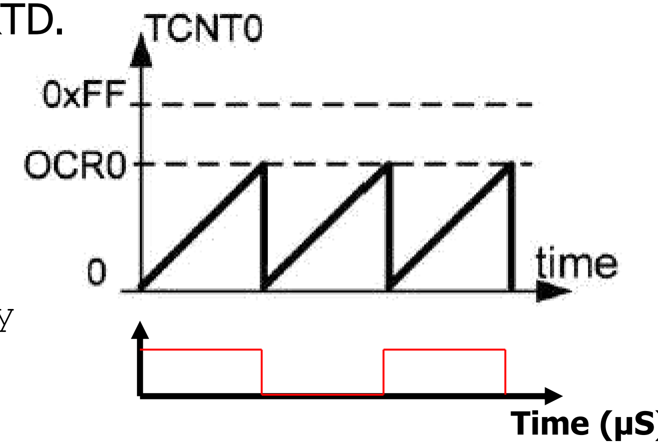

```
;Program 10-1
.INCLUDE "M32DEF.INC"
.ORG      0x0       ;location for reset
          JMP       MAIN
.ORG      0x16      ;loc. for Timer0 over.
          JMP       T0_OV_ISR
;----main program for initialization
.ORG      0x100
MAIN:     LDI       R20,HIGH(RAMEND)
          OUT       SPH,R20
          LDI       R20,LOW(RAMEND)
          OUT       SPL,R20
          SBI       DDRB,5    ;output
          LDI       R20,0
          OUT       DDRC, R20
          LDI       R20,0xFF
          OUT       DDRD, R20
          LDI       R20,(1<<TOIE0)
          OUT       TIMSK,R20
          SEI
          LDI       R20,-32   ;value for 4µs
          OUT       TCNT0,R20
          LDI       R20,0x01
          OUT       TCCR0,R20
HERE:     IN        R20,PINC
          OUT       PORTD,R20
          JMP       HERE
;--------ISR for Timer 0
T0_OV_ISR:
          IN        R16,PORTB
          LDI       R17,0x20
          EOR       R16,R17
          OUT       PORTB,R16
          RETI
```

Timer int. init. (LDI R20,(1<<TOIE0) / OUT TIMSK,R20 / SEI)

Timer init. (LDI R20,-32 / OUT TCNT0,R20 / LDI R20,0x01 / OUT TCCR0,R20)

Timer0 compare match interrupt

- using Timer0 and CTC mode generate a square wave on pin PORTB.5, while at the same time data is being transferred from PORTC to PORTD.




```
.INCLUDE "M32DEF.INC"
.ORG      0x0       ;location for reset
          JMP       MAIN
.ORG      0x14      ;location for Timer0 compare match
          JMP       T0_CM_ISR
;-main program for initialization and keeping CPU busy
.ORG      0x100
MAIN:     LDI       R20,HIGH(RAMEND)
          OUT       SPH,R20
          LDI       R20,LOW(RAMEND)
          OUT       SPL,R20
          LDI       R20,39
          OUT       OCR0,R20      ;OCR0 = 39
          LDI       R20,0x09
          OUT       TCCR0,R20      ;Start Timer0
          SBI       DDRB,5        ;PB5 as an output
          LDI       R20,(1<<OCIE0) ;Timer0 compare match
          OUT       TIMSK,R20
          SEI                    ;Set I
          LDI       R20,0x00
          OUT       DDRC,R20     ;make PORTC input
          LDI       R20,0xFF
          OUT       DDRD,R20     ;make PORTD output
```

Timer init. (LDI R20,39 … OUT TCCR0,R20)

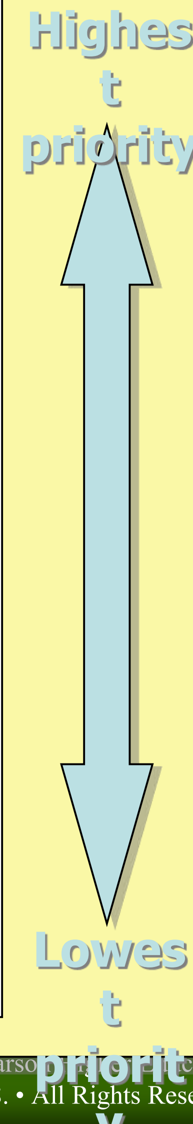
Timer int. init. (LDI R20,(1<<OCIE0) … SEI)

```
HERE:     IN        R20,PINC
          OUT       PORTD,R20
          JMP       HERE
;--------ISR for Timer 0
T0_CM_ISR:
          IN        R16,PORTB
          LDI       R17,0x20
          EOR       R16,R17
          OUT       PORTB,R16
          RETI
```

Interrupt priority

Table 10-1: Interrupt Vector Table for the Mega32

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028



Interrupt inside an interrupt

- The I flag is cleared when the AVR begins to execute an ISR. So, interrupts are disabled.
- The I flag is set when RETI is executed.

Task switching and resource conflict

- Does the following program work?

```
.INCLUDE "M32DEF.INC"
.ORG      0x0        ;location for reset
          JMP        MAIN
.ORG      0x14       ;Timer0 compare match
          JMP        T0_CM_ISR
;--------main program------------------------
----
.ORG      0x100
MAIN:     LDI        R20,HIGH(RAMEND)
          OUT        SPH,R20
          LDI        R20,LOW(RAMEND)
          OUT        SPL,R20    ;set up stack
          SBI        DDRB,5     ;PB5 =
output
          LDI        R20,160
          OUT        OCR0,R20
          LDI        R20,0x09
          OUT        TCCR0,R20
          LDI        R20,(1<<OCIE0)
          OUT        TIMSK,R20
          SEI
          LDI        R20,0xFF
          OUT        DDRC,R20
          OUT        DDRD,R20
          LDI        R20, 0
HERE:     OUT        PORTC,R20
          INC        R20
          JMP        HERE
;----------------------ISR for Timer0
T0_CM_ISR:
          IN         R20,PIND
          INC        R20
          OUT        PORTD,R20
          RETI
```

Task switching and resource conflict

- Does the following program work?

```
.INCLUDE "M32DEF.INC"
.ORG      0x0       ;location for reset
          JMP       MAIN
.ORG      0x14      ;Timer0 compare match
          JMP       T0_CM_ISR
;--------main program------------------------
----
.ORG      0x100
MAIN:     LDI       R20,HIGH(RAMEND)
          OUT       SPH,R20
          LDI       R20,LOW(RAMEND)
          OUT       SPL,R20   ;set up stack
          SBI       DDRB,5    ;PB5 =
output
          LDI       R20,160
          OUT       OCR0,R20
          LDI       R20,0x09
          OUT       TCCR0,R20
          LDI       R20,(1<<OCIE0)
          OUT       TIMSK,R20
          SEI
          LDI       R20,0xFF
          OUT       DDRC,R20
          OUT       DDRD,R20
          LDI       R20, 0
HERE:     OUT       PORTC,R20
          INC       R20
          JMP       HERE
;----------------------ISR for Timer0
T0_CM_ISR:
          IN        R20,PIND
          INC       R20
          OUT       PORTD,R20
          RETI
```

Solution 1: different registers

- Use different registers for different tasks.

```
.INCLUDE "M32DEF.INC"
.ORG      0x0        ;location for reset
          JMP        MAIN
.ORG      0x14       ;Timer0 compare match
          JMP        T0_CM_ISR
;---------main program-------------------------
----
.ORG      0x100
MAIN:     LDI        R20,HIGH(RAMEND)
          OUT        SPH,R20
          LDI        R20,LOW(RAMEND)
          OUT        SPL,R20    ;set up stack
          SBI        DDRB,5     ;PB5 =
output
          LDI        R20,160
          OUT        OCR0,R20
          LDI        R20,0x09
          OUT        TCCR0,R20
```

```
          LDI        R20,(1<<OCIE0)
          OUT        TIMSK,R20
          SEI
          LDI        R20,0xFF
          OUT        DDRC,R20
          OUT        DDRD,R20
          LDI        R20, 0
HERE:     OUT        PORTC,R20
          INC        R20
          JMP        HERE
;------------------------ISR for Timer0
T0_CM_ISR:
          IN         R21,PIND
          INC        R21
          OUT        PORTD,R21
          RETI
```

Solution 2: Context saving

- Save the contents of registers on the stack before execution of each task, and reload the registers at the end of the task.

```
.INCLUDE "M32DEF.INC"
.ORG      0x0       ;location for reset
          JMP       MAIN
.ORG      0x14      ;Timer0 compare match
          JMP       T0_CM_ISR
;---------main program------------------------
----
.ORG      0x100
MAIN:     LDI       R20,HIGH(RAMEND)
          OUT       SPH,R20
          LDI       R20,LOW(RAMEND)
          OUT       SPL,R20   ;set up stack
          SBI       DDRB,5    ;PB5 =
output
          LDI       R20,160
          OUT       OCR0,R20
          LDI       R20,0x09
          OUT       TCCR0,R20
          LDI       R20,(1<<OCIE0)
```

```
          OUT       TIMSK,R20
          SEI
          LDI       R20,0xFF
          OUT       DDRC,R20
          OUT       DDRD,R20
          LDI       R20, 0
HERE:     OUT       PORTC,R20
          INC       R20
          JMP       HERE
;-----------------------ISR for Timer0
T0_CM_ISR:
          PUSH      R20       ;save R20
          IN        R20,PIND
          INC       R20
          OUT       PORTD,R20
          POP       R20       ;restore R20
          RETI
```

Saving SREG

- We should save SREG, when we change flags in the ISR.

```
PUSH    R20
IN      R20,SREG
PUSH    R20
...
POP     R20
OUT     SREG,R20
POP     R20
```

C programming

- Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

```
#include "avr/io.h"
#include "avr/interrupt.h"
int main ()
{
       DDRB |= 0x20;             //DDRB.5 = output
       TCNT0 = -32;              //timer value for 4 µs
       TCCR0 = 0x01;             //Normal mode, int clk, no prescaler
       TIMSK = (1<<TOIE0);       //enable Timer0 overflow interrupt
       sei ();                   //enable interrupts
       DDRC = 0x00;              //make PORTC input
       DDRD = 0xFF;              //make PORTD output
       while (1)                 //wait here
         PORTD = PINC;
}
ISR (TIMER0_OVF_vect)            //ISR for Timer0 overflow
{
       TCNT0 = -32;
       PORTB ^= 0x20;            //toggle PORTB.5
}
```

C programming

- Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

```
#include "avr/io.h"
#include "avr/interrupt.h"
int main ()
{
        DDRB |= 0x20;               //DDRB.5 = output
        TCNT0 = -32;                //timer value for 4 µs
        TCCR0 = 0x01;               //Normal mode, int clk, no prescaler
        TIMSK = (1<<TOIE0);         //enable Timer0 overflow interrupt
        sei ();                     //enable interrupts
        DDRC = 0x00;                //make PORTC input
        DDRD = 0xFF;                //make PORTD output
        while (1)                   //wait here
          PORTD = PINC;
}
ISR (TIMER0_OVF_vect)               //ISR for Timer0 overflow
{
        TCNT0 = -32;
        PORTB ^= 0x20;              //toggle PORTB.5
}
```

C programming

- Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

```
#include "avr/io.h"
#include "avr/interrupt.h"
int main ()
{
        DDRB |= 0x20;               //DDRB.5 = output
        TCNT0 = -32;                //timer value for 4 µs
        TCCR0 = 0x01;               //Normal mode, int clk, no prescaler
        TIMSK = (1<<TOIE0);                                  interrupt
        sei ();
        DDRC = 0x00;
        DDRD = 0xFF;                //make PORTD output
        while (1)                   //wait here
          PORTD = PINC;
}
ISR (TIMER0_OVF_vect)               //ISR for Timer0 overflow
{
        TCNT0 = -32;
        PORTB ^= 0x20;              //toggle PORTB.5
}
```

```
sei ( );      //set I
cli  ( );     //clear I
```

C programming

- Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

```
#include "avr/io.h"
#include "avr/interrupt.h"
int main ()
{
        DDRB |= 0x20;             //DDRB.5 = output
        TCNT0 = -32;              //timer value for 4 µs
        TCCR0 = 0x01;             //Normal mode, int clk, no prescaler
        TIMSK = (1<<TOIE0);       //enable Timer0 overflow interrupt
        sei ();                   //enable interrupts
        DDRC = 0x00;              //make PORTC input
        DDRD = 0xFF;              //make PORTD output
        while (1)                 //wait here
          PORTD = PINC;
}
ISR (TIMER0_OVF_vect)             //ISR for Timer0 overflow
{
        TCNT0 = -32;
        PORTB ^= 0x20;            //toggle PORTB.5
}
```

C programming

- Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

```
#include "avr/io.h"
#include "avr/interru
int main ()
{
        DDRB |= 0x20;
        TCNT0 = -32;
        TCCR0 = 0x01;
        TIMSK = (1<<TO
        sei ();
        DDRC = 0x00;
        DDRD = 0xFF;
        while (1)
          PORTD = PINC
}
ISR (TIMER0_OVF_vect)
{
        TCNT0 = -32;
        PORTB ^= 0x20;
}
```

Table 10-3: Interrupt Vector Name for the ATmega32/ATmega16 in WinAVR

Interrupt	Vector Name in WinAVR
External Interrupt request 0	INT0_vect
External Interrupt request 1	INT1_vect
External Interrupt request 2	INT2_vect
Time/Counter2 Compare Match	TIMER2_COMP_vect
Time/Counter2 Overflow	TIMER2_OVF_vect
Time/Counter1 Capture Event	TIMER1_CAPT_vect
Time/Counter1 Compare Match A	TIMER1_COMPA_vect
Time/Counter1 Compare Match B	TIMER1_COMPB_vect
Time/Counter1 Overflow	TIMER1_OVF_vect
Time/Counter0 Compare Match	TIMER0_COMP_vect
Time/Counter0 Overflow	TIMER0_OVF_vect
SPI Transfer complete	SPI_STC_vect
USART, Receive complete	USART0_RX_vect
USART, Data Register Empty	USART0_UDRE_vect
USART, Transmit Complete	USART0_TX_vect
ADC Conversion complete	ADC_vect
EEPROM ready	EE_RDY_vect
Analog Comparator	ANA_COMP_vect
Two-wire Serial Interface	TWI_vect
Store Program Memory Ready	SPM_RDY_vect

C programming

- Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

```
#include "avr/io.h"
#include "avr/interrupt.h"
int main ()
{
       DDRB |= 0x20;              //DDRB.5 = output
       TCNT0 = -32;               //timer value for 4 µs
       TCCR0 = 0x01;              //Normal mode, int clk, no prescaler
       TIMSK = (1<<TOIE0);        //enable Timer0 overflow interrupt
       sei ();                    //enable interrupts
       DDRC = 0x00;               //make PORTC input
       DDRD = 0xFF;               //make PORTD output
       while (1)                  //wait here
         PORTD = PINC;
}
ISR (TIMER0_OVF_vect)             //ISR for Timer0 overflow
{
       TCNT0 = -32;
       PORTB ^= 0x20;             //toggle PORTB.5
}
```

C programming

- Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

```
#include "avr/io.h"
#include "avr/interrupt.h"
int main ()
{
        DDRB |= 0x20;             //DDRB.5 = output
        TCNT0 = -32;              //timer value for 4 µs
        TCCR0 = 0x01;             //Normal mode, int clk, no prescaler
        TIMSK = (1<<TOIE0);       //enable Timer0 overflow interrupt
        sei ();                   //enable interrupts
        DDRC = 0x00;              //make PORTC input
        DDRD = 0xFF;              //make PORTD output
        while (1)                 //wait here
          PORTD = PINC;
}
ISR (TIMER0_OVF_vect)             //ISR for Timer0 overflow
{
        TCNT0 = -32;
        PORTB ^= 0x20;            //toggle PORTB.5
}
```

C programming Example 2

- Using Timer1 and CTC mode write a program that toggles pin PORTB.5 every second, while at the same time transferring data from PORTC to PORTD. Assume XTAL = 8 MHz.

```
#include "avr/io.h"
#include "avr/interrupt.h"

int main ()
{
       DDRB |= 0x20;            //make DDRB.5 output
       OCR0 = 40;
       TCCR0 = 0x09;            //CTC mode, internal clk, no prescaler
       TIMSK = (1<<OCIE0);      //enable Timer0 compare match int.
       sei ();                  //enable interrupts
       DDRC = 0x00;             //make PORTC input
       DDRD = 0xFF;             //make PORTD output
       while (1)                //wait here
         PORTD = PINC;
}

ISR (TIMER0_COMP_vect)          //ISR for Timer0 compare match
{
       PORTB ^= 0x20;           //toggle PORTB.5
}
```

C programming Example 3

- Assume that the INT0 pin is connected to a switch that is normally high. Write a program that toggles PORTC.3, whenever INT0 pin goes low. Use the external interrupt in level-triggered mode.

```
#include "avr/io.h"
#include "avr/interrupt.h"

int main ()
{
        DDRC = 1<<3;              //PC3 as an output
        PORTD = 1<<2;             //pull-up activated
        GICR = (1<<INT0);         //enable external interrupt 0
        sei ();                   //enable interrupts

        while (1);                //wait here
}

ISR (INT0_vect)                   //ISR for external interrupt 0
{
        PORTC ^= (1<<3);          //toggle PORTC.3
}
```

ADC and DAC Programming in AVR

The AVR microcontroller and embedded systems
using assembly and c

Topics

- What is ADC and why do we need it?
- ADC major characteristics
- ADC in AVR
- Hardware Consideration
- AVR ADC Programming
  - ADCH and ADCL
  - ADMUX
  - ADCSRA
- DAC
- Signal conditioning and sensors

What is ADC? Do we need it?

- Analogue vs. digital signal

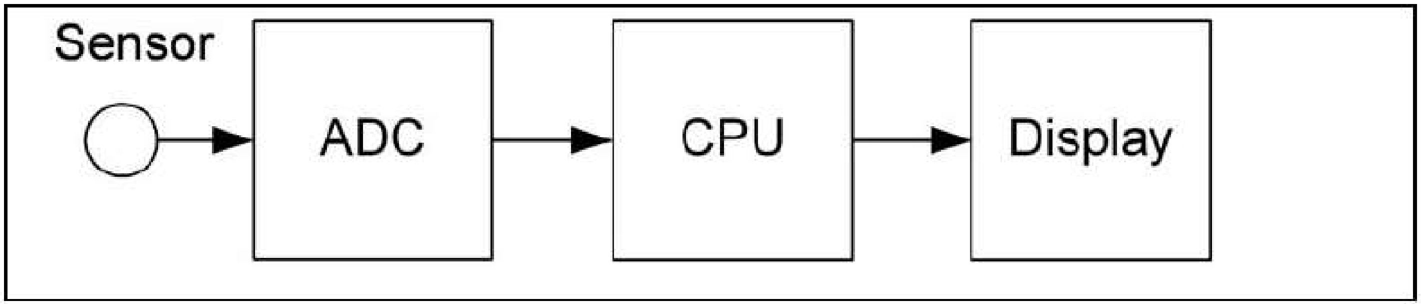


Figure 13-1. Microcontroller Connection to Sensor via ADC

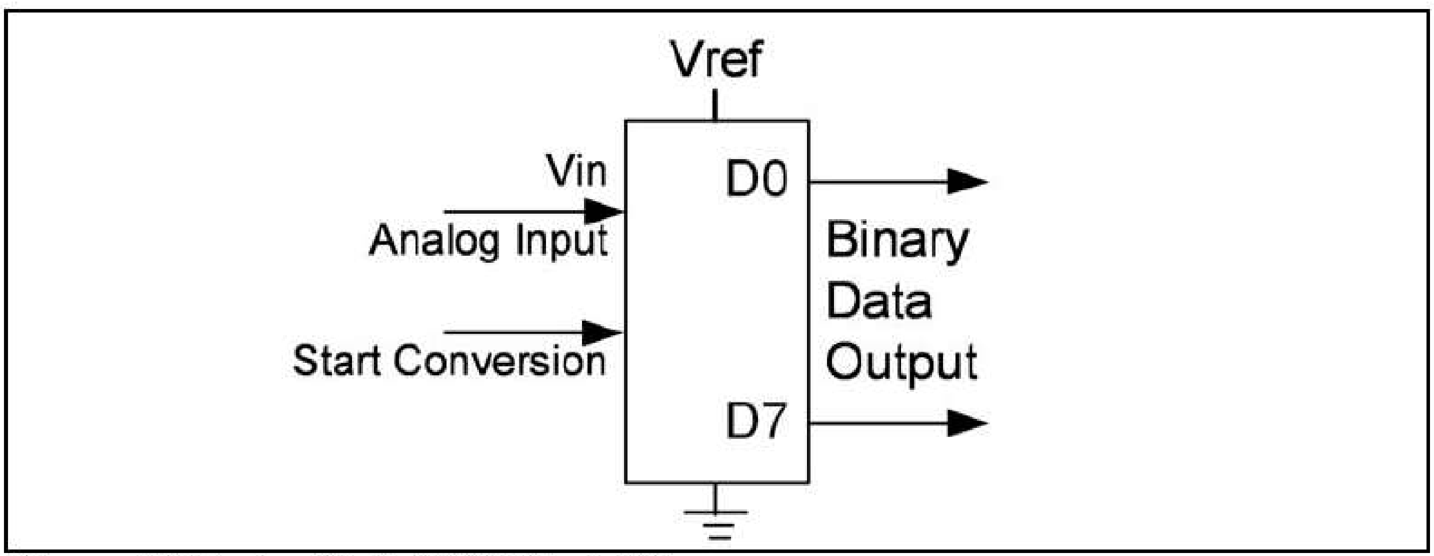


Figure 13-2. An 8-bit ADC Block Diagram

ADC major characteristics

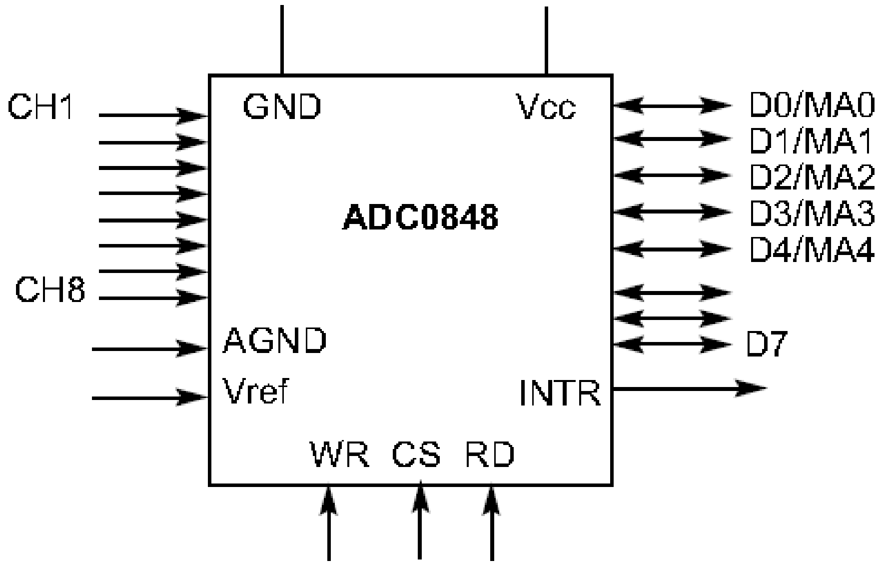
- Conversion Time
- Resolution
- Vref
- Parallel vs. serial
- Input channels

(Resolution, Vref) → Step Size

Some of ADC Signals

- Dout = Vin / Step size

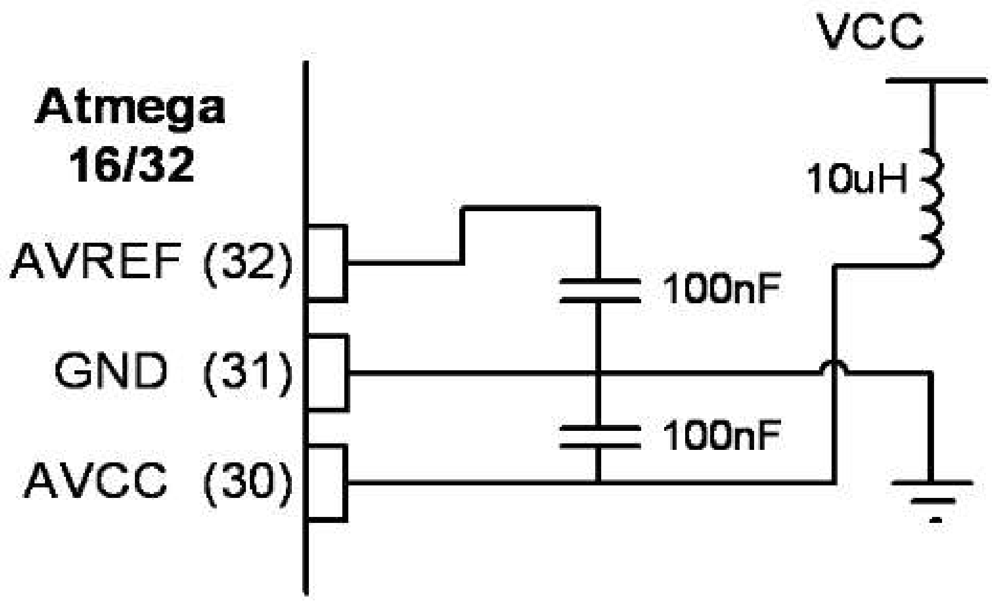
  (in 8 bit ADC and Vref = 2.56 what is D out for 20mV input voltage?)
- Start of Conversion
- Channel Selector



ADC in AVR

- Atmega 16/32 have internal ADC
  - 8 analogue input channel
  - 7 differential input channel
  - 2 differential input channel with 10x or 200x gain
  - 3 source of Vref
  - Internal 2.56V Vref generator

Hardware Consideration



ADCH and ADCL Data registers

- ADCH:ADCL store the results of conversion.
- The 10 bit result can be right or left justified:

ADLAR = 0

ADCH							
-	-	-	-	-	-	ADC9	ADC8

ADCL							
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

ADLAR =1

ADCH							
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2

ADCL							
ADC1	ADC0	-	-	-	-	-	-

ADMUX

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

REFS1:0- Bit7:6 Reference Selection Bits

These bits select the voltage reference for the ADC.

ADLAR- Bit5 ADC Left Adjust Reslts

This bit dictate either the left bits or the right bits of the result registers ADCH:ADCL are used to store the result. If we write ADLAR to one the result will left adjusted otherwise the result is right adjusted.

MUX4:0- Bit4:0 Analog Channel and gain selection bits

The value of these bits selects the gain for the differential channels and also which combination of analog inputs are connected to the ADC.

ADMUX

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

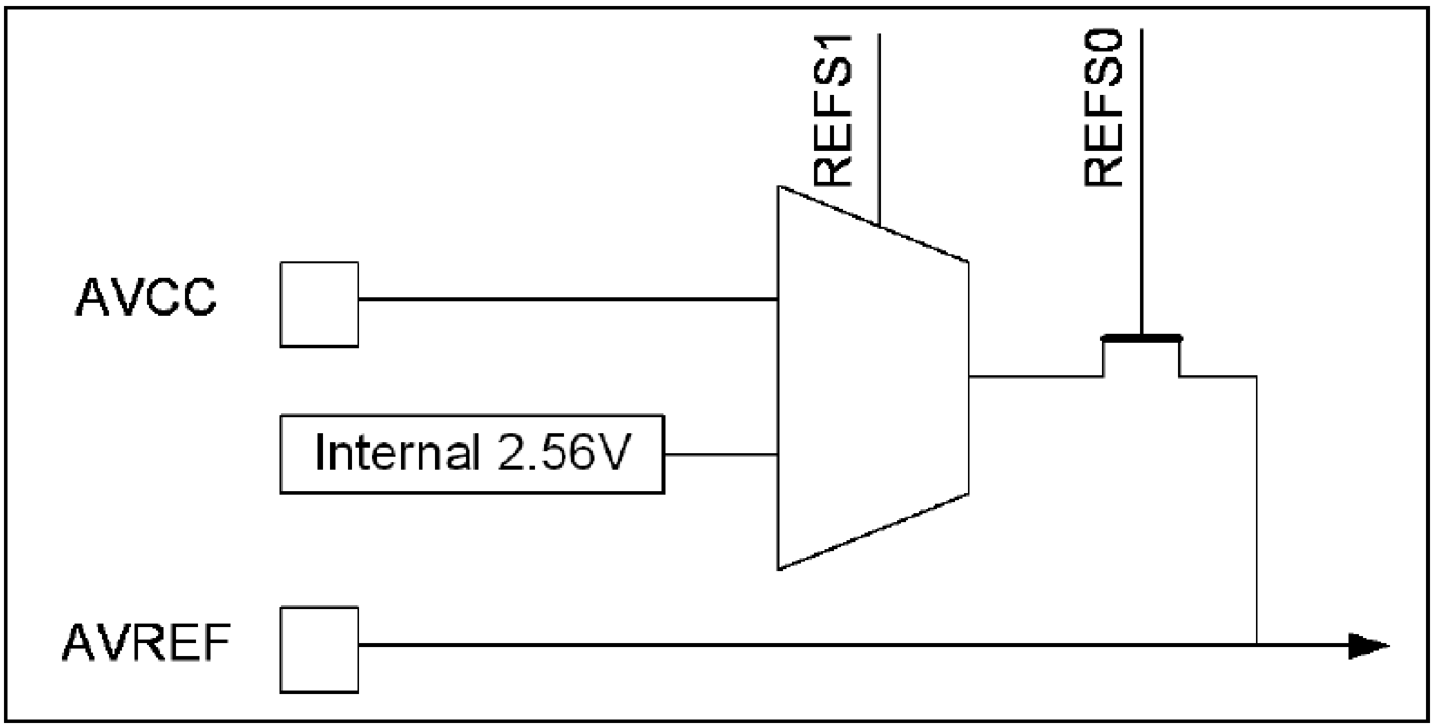


Figure 13-7: ADC Reference Source Selection

ADMUX

Table 13-4: V_{ref} source selection table

REFS1	REFS0	V Reference
0	0	AREF pin
0	1	AVCC pin
1	0	Reserved
1	1	Internal 2.56 V

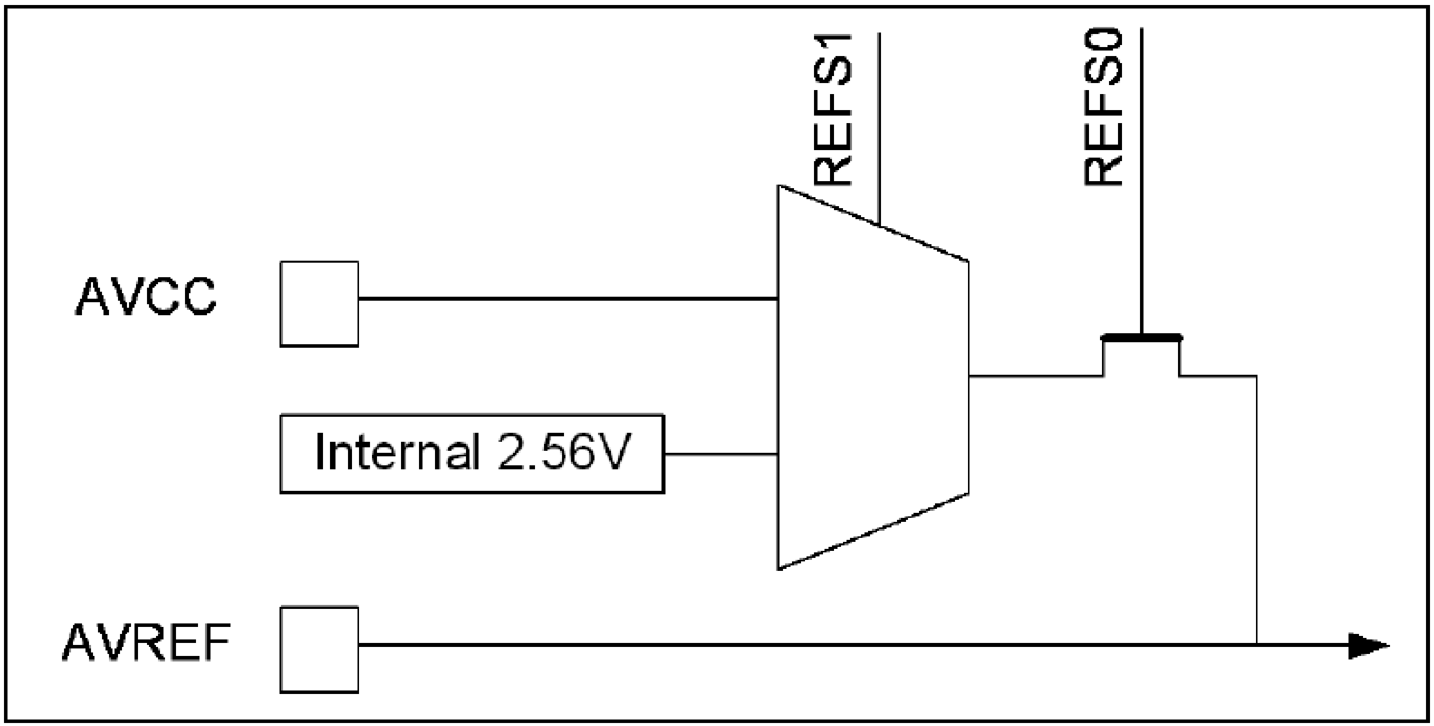
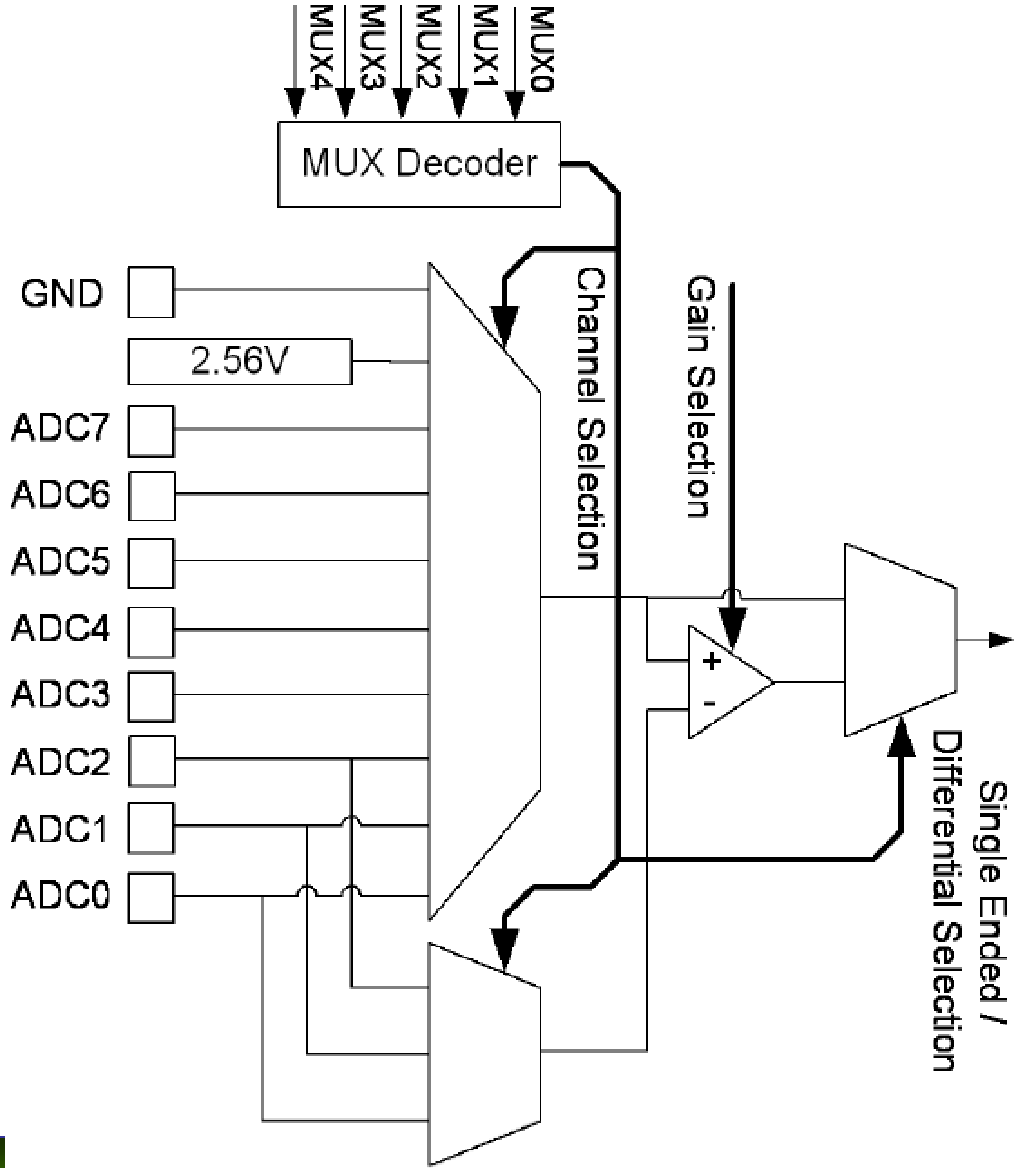


Figure 13-7: ADC Reference Source Selection

ADC input selection



ADC input sel

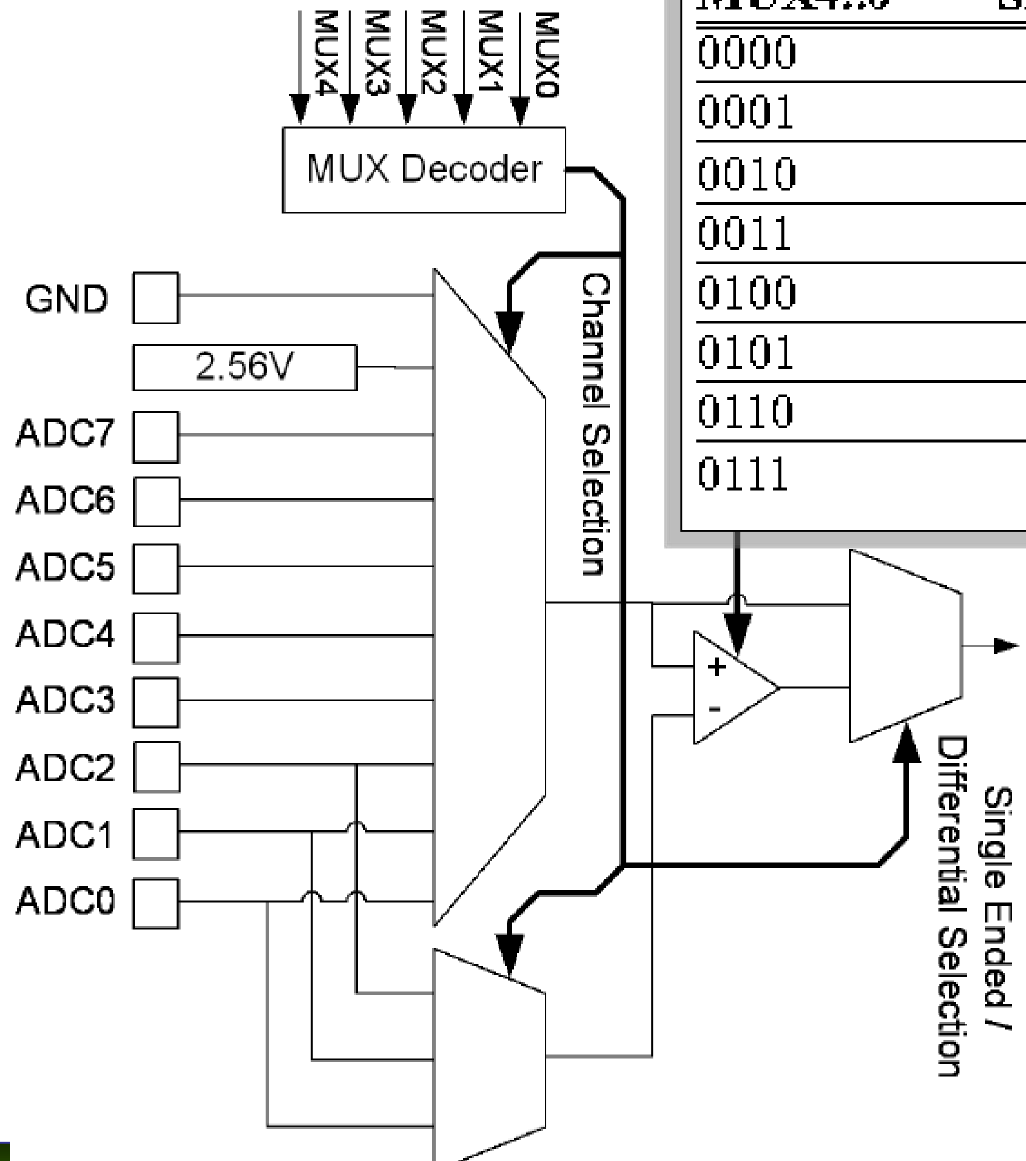


Table 13-6: Single Ended Channels

MUX4..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7

ADC input selection

MUX4..0	+ Differential Input	- Differential Input	Gain
01000	ADC0	ADC0	10x
01001	ADC1	ADC0	10x
01010	ADC0	ADC0	200x
01011	ADC1	ADC0	200x
01100	ADC2	ADC2	10x
01101	ADC3	ADC2	10x
01110	ADC2	ADC2	200x
01111	ADC3	ADC2	200x
10000	ADC0	ADC1	1x
10001	ADC1	ADC1	1x
10010	ADC2	ADC1	1x
10011	ADC3	ADC1	1x
10100	ADC4	ADC1	1x
10101	ADC5	ADC1	1x
10110	ADC6	ADC1	1x
10111	ADC7	ADC1	1x
11000	ADC0	ADC2	1x
11001	ADC1	ADC2	1x
11010	ADC2	ADC2	1x
11011	ADC3	ADC2	1x
11100	ADC4	ADC2	1x
11101	ADC5	ADC2	1x

ADCSA

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN- Bit7 ADC Enable
This bit enables or disables the ADC. Writing this bit to one will enable and writing this bit to zero will disable the ADC even while a conversion is in progress.

ADSC- Bit6 ADC Start Conversion
To start each coversion you have to write this bit to one.

ADATE- Bit5 ADC Auto Trigger Enable
Auto Triggering of the ADC is enabled when you write this bit to one.

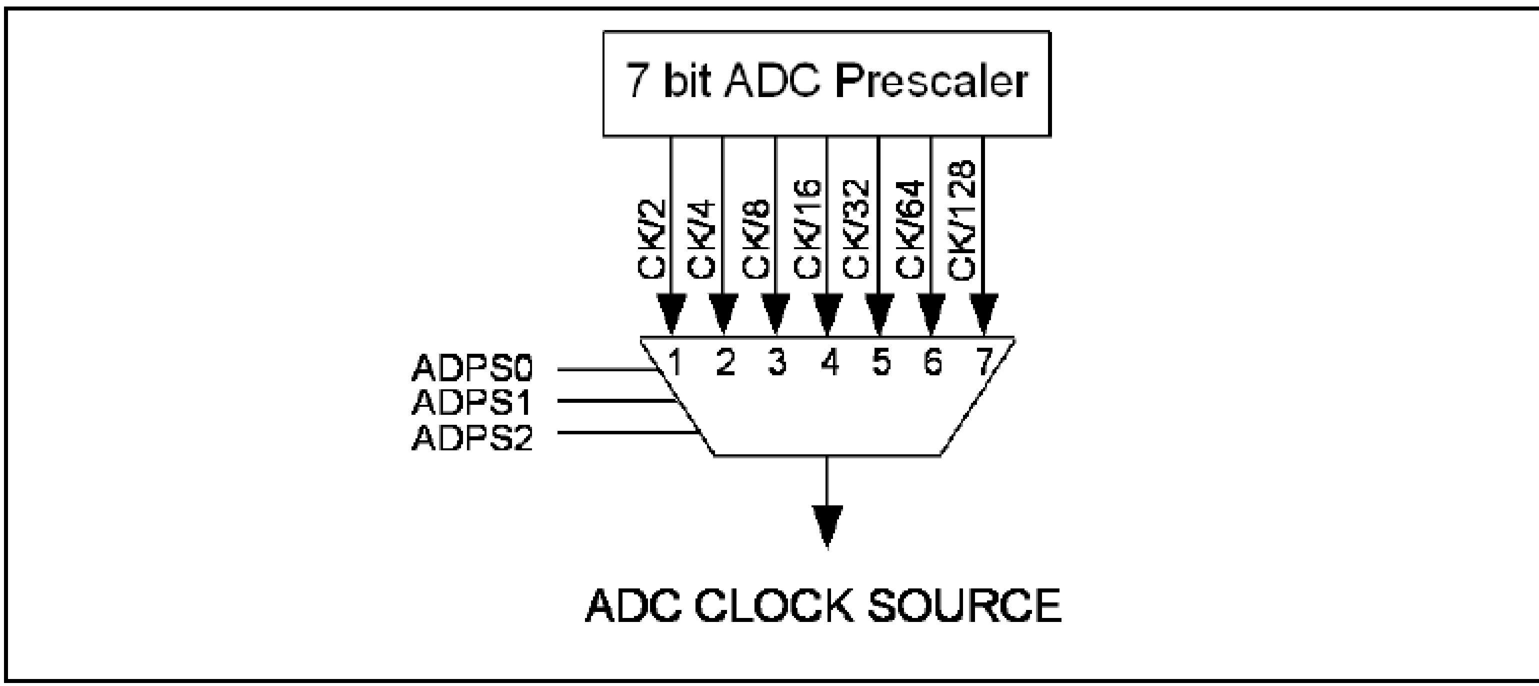
ADIF- Bit4 ADC Interrupt Flag
This bit is set when an ADC conversion completes and the Data Registers are updated

ADIE- Bit3 ADC Interrupt Enable
Writing this bit to one enables the ADC Conversion Complete Interrupt.

ADPS2:0- Bit2:0 ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

ADC Prescaler

- PreScaler Bits let us change the clock frequency of ADC
- The frequency of ADC should not be more than 200 KHz
- Conversion time is longer in the first conversion

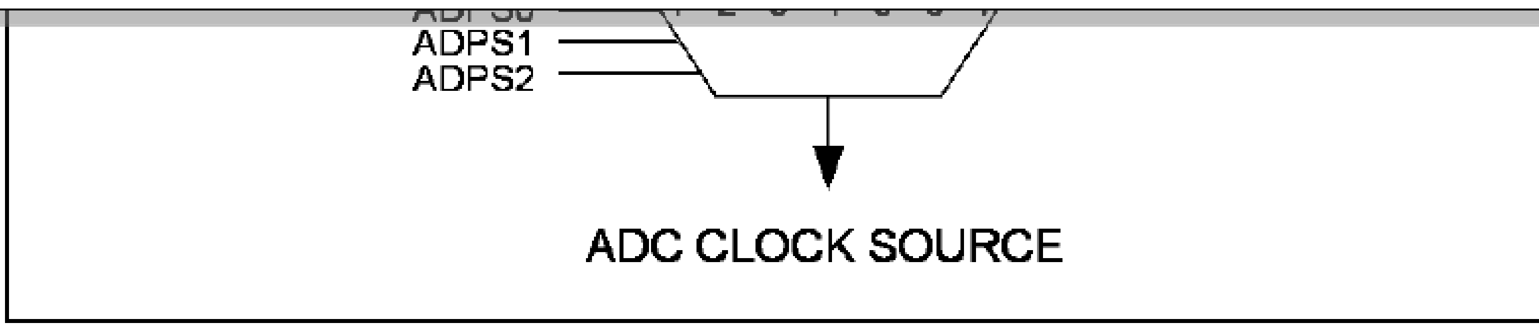


ADC Prescaler

- PreScaler Bits let us change the clock frequency of ADC
- The frequency of ADC should not be more than 200 KHz
- Conversion time is longer in the first conversion

Table 13-3: V_{ref} source selection table

Condition	Sample and Hold Time (Cycles)	Conversion Time (Cycles)
First Conversion	14.5	25
Normal Conversion, Single ended	1.5	13
Normal Conversion, Differential	2	13.5
Auto trigger conversion	1.5 / 2.5	13/14



Programming ADC

1. Make the pin for the selected ADC channel an input pin.
2. Turn on the ADC module of the AVR because it is disabled upon power-on reset to save power.
3. Select the conversion speed. We use registers ADPS2:0 to select the conversion speed.
4. Select voltage reference and A/C input channels. We use REFS0 amd REFS1 bits in ADMUX register to select voltage reference and MUX4:0 bits in ADMUX to select ADC input channel.
5. Activate the start conversion bit by writing ADSC bit of ADCSRA to one.
6. Wait for the conversion to be completed by polling the ADIF bit in ADCSRA register.
7. After the ADIF bit has gone one read the ADCL and ADCH registers to get the digital data output. Note that you have to read ADCL before ADCH otherwise the result may not be valid.
8. If you want to read the selected channel again go back to step 5.
9. If you want to select another Vref source or input channel go back to step 4.

Programming ADC - Polling

Program 13-1C is the C version of the ADC conversion for Program 13-1.

```
#include <avr/io.h>          //standard AVR header
int main (void)
{
  DDRB = 0xFF;               //make Port B an output
  DDRD = 0xFF;               //make Port D an output
  DDRA = 0;                  //make Port A an input for ADC input
  ADCSRA= 0x87;              //make ADC enable and select ck/128
  ADMUX= 0xC0;               //2.56V Vref, ADC0 single ended input
                             //data will be right-justified
  while (1){
    ADCSRA|=(1<<ADSC);       //start conversion
    while((ADCSRA&(1<<ADIF))==0);//wait for conversion to finish
    PORTD = ADCL;            //give the low byte to PORTD
    PORTB = ADCH;            //give the high byte to PORTB
  }
  return 0;
}
```

Program 13-1C: Reading ADC Using Polling Method in C

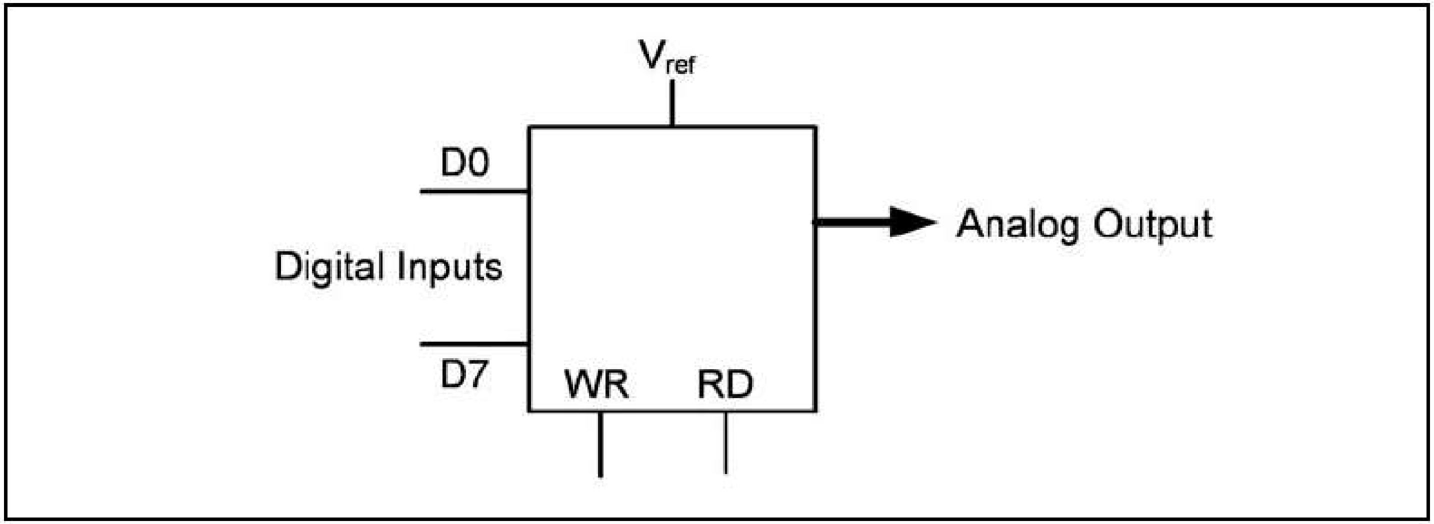
Programming ADC - Interrupts

```
#include <avr\io.h>
#include <avr\interrupt.h>
ISR(ADC_vect){
  PORTD = ADCL;              //give the low byte to PORTD
  PORTB = ADCH;              //give the high byte to PORTB
  ADCSRA|=(1<<ADSC);         //start conversion
}
int main (void){
  DDRB = 0xFF;               //make Port B an output
  DDRD = 0xFF;               //make Port D an output
  DDRA = 0;                  //make Port A an input for ADC input
  sei();                     //enable interrupts
  ADCSRA= 0x8F;              //enable and interrupt select ck/128
  ADMUX= 0xC0;               //2.56V Vref and ADC0 single-ended
                             //input right-justified data
  ADCSRA|=(1<<ADSC);         //start conversion
  while (1);                 //wait forever
  return 0;
}
```

Program 13-2C: Reading ADC Using Interrupts in C

DAC

- What is DAC ?
- How to connect an DAC to AVR?



$$I_{out} = I_{ref}\left(\frac{D7}{2}+\frac{D6}{4}+\frac{D5}{8}+\frac{D4}{16}+\frac{D3}{32}+\frac{D2}{64}+\frac{D1}{128}+\frac{D0}{256}\right)$$

Signal conditioning

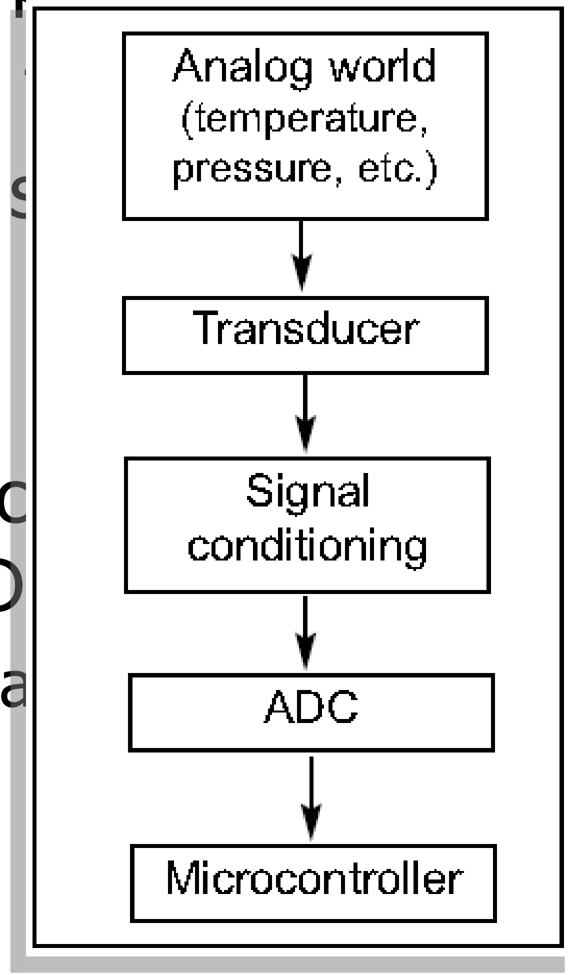
- Thermo couple provides temp in form of uV
- PT100 provides temp in form of resistance
- Some humidity sensor provide the result in form of Capacitance

- -> We have to change these signals to Voltage to convert it by ADC
  - This job is called signal conditioning

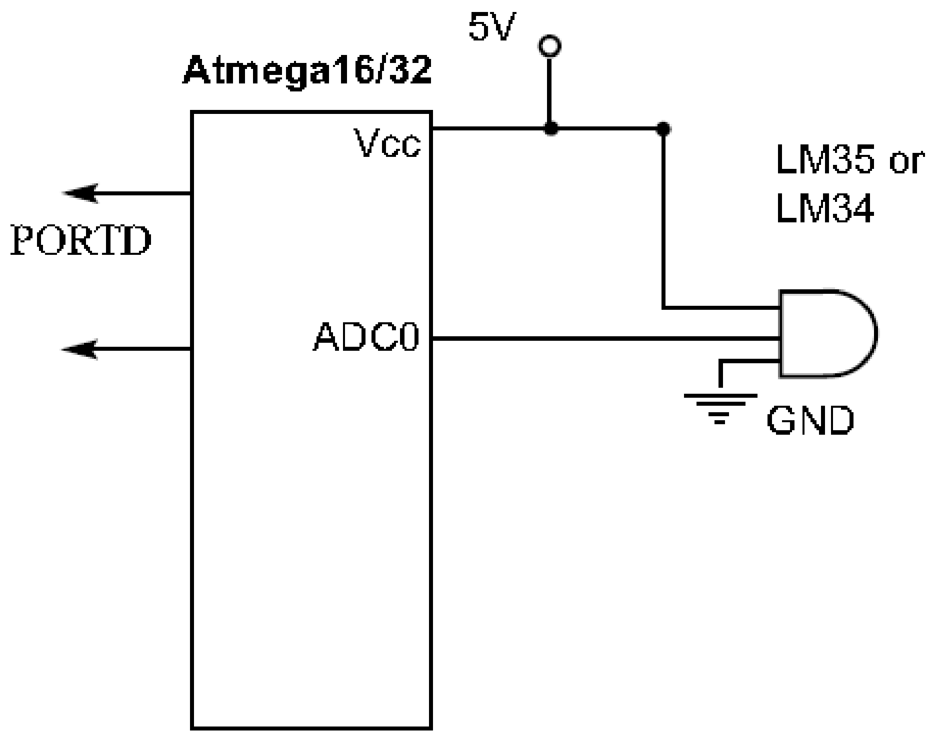
Signal conditioning

- Thermo couple provides temp in form of uV
- PT100 provides resistance
- Some humidity e result in form of Capacitance

- -> We have to c als to Voltage to convert it by AD
  - This job is ca ing



Sensor Interfacing



Output

Relay, Optoisolator, and Stepper motor interfacing with AVR

The AVR microcontroller and embedded systems

using assembly and c

Topics

- AVR Fan-out
- Transistor
- ULN2003
- Relay
- Opto-isolator
- Stepper motor

AVR Fan-out

- Each I/O port can sink 20 mA at VCC = 5V and 10 mA at VCC = 3V

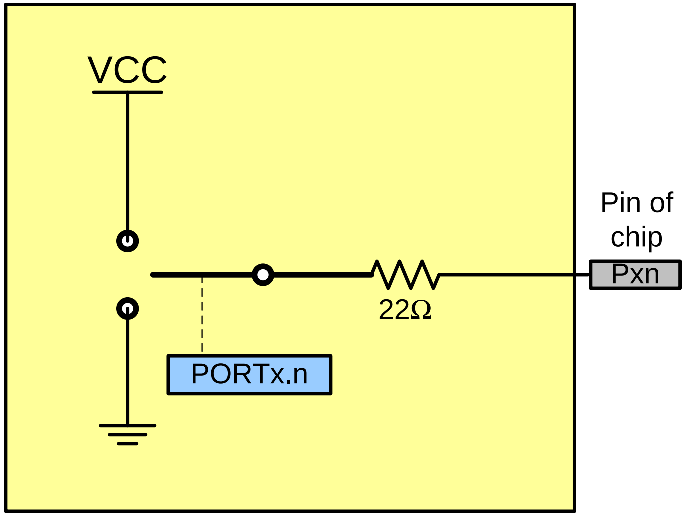
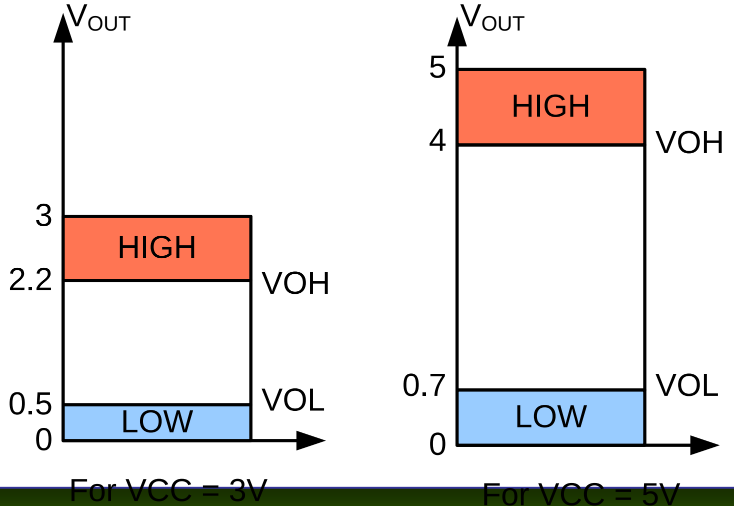


Table C-5: Fan-out for AVR Ports

Pin	Fan-out
IOL	20 mA
IOH	–20 mA
IIL	–1 μA
IIH	1 μA

Note: Negative current is defined as current sourced by the pin.



Transistor

- We can switch devices using transistors.
- Transistor amplifies signals.

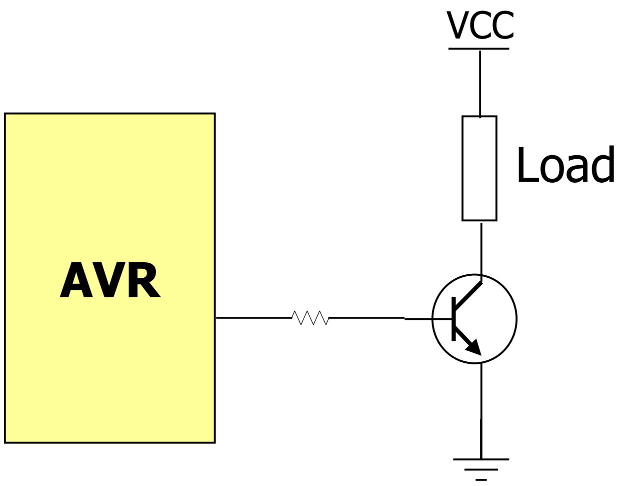
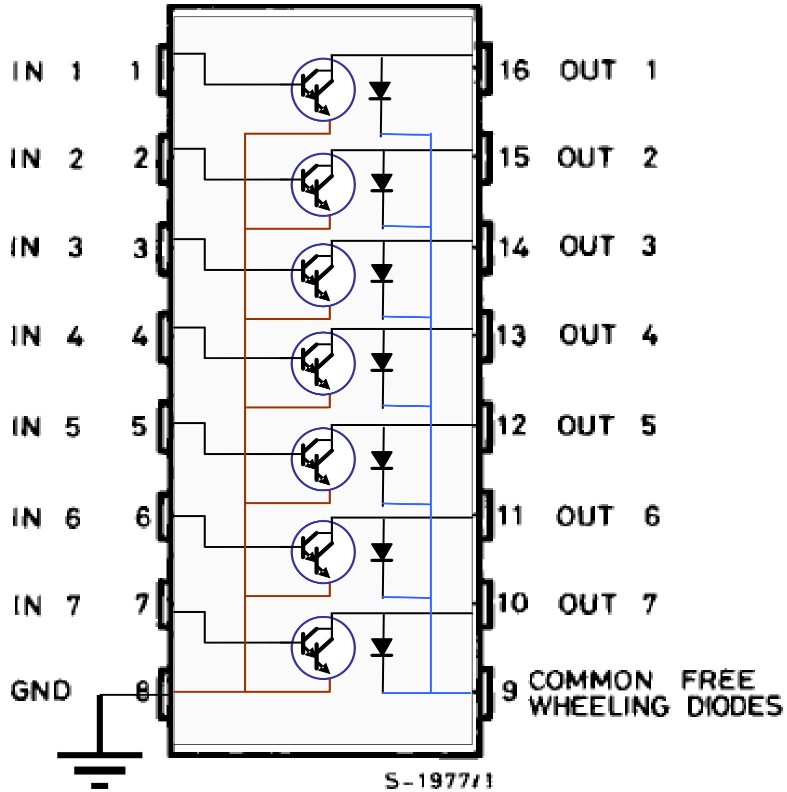


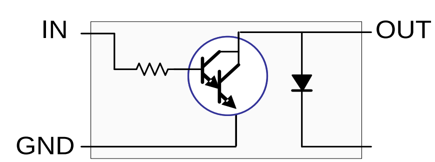
Table 14-8: Darlington Transistor Listing

NPN	PNP	Vceo (volts)	Ic (amps)	hfe (common)
TIP110	TIP115	60	2	1000
TIP111	TIP116	80	2	1000
TIP112	TIP117	100	2	1000
TIP120	TIP125	60	5	1000
TIP121	TIP126	80	5	1000
TIP122	TIP127	100	5	1000
TIP140	TIP145	60	10	1000
TIP141	TIP146	80	10	1000
TIP142	TIP147	100	10	1000

ULN2003

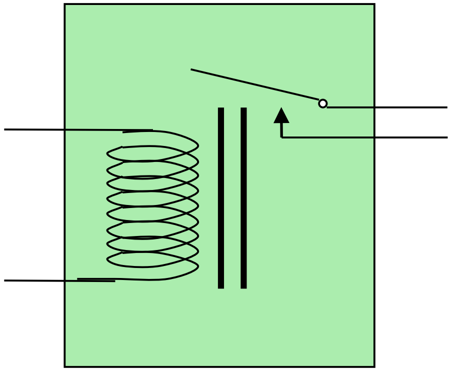
- There are 7 Darlington transistors in a ULN2003.

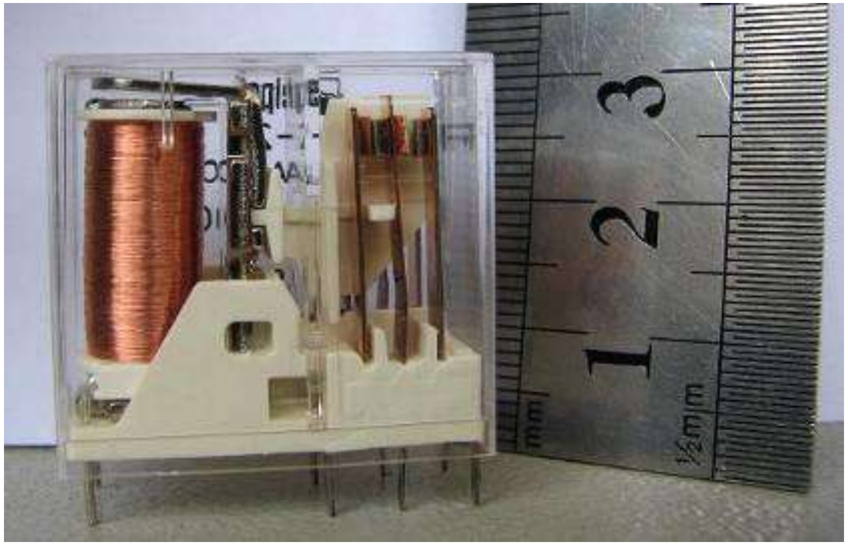




Relay

- Relay is an electronic controlled switch.
- It isolates two parts of a circuit from each other.
- A small amount of current and voltage causes to switch a large amount of voltage and current.

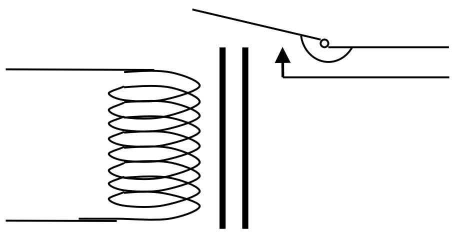




Relay DPDT
Maximum output: 250V 8A ~AC
Input: 24^{V} DC

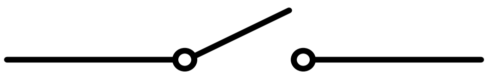
Relay parts

- Relay has the following parts:
  - Coil
  - Contacts
  - Spring

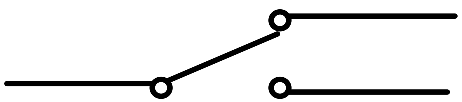


Contacts

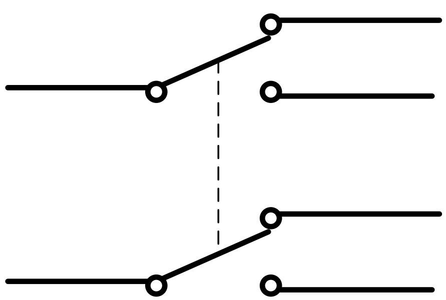
- SPST

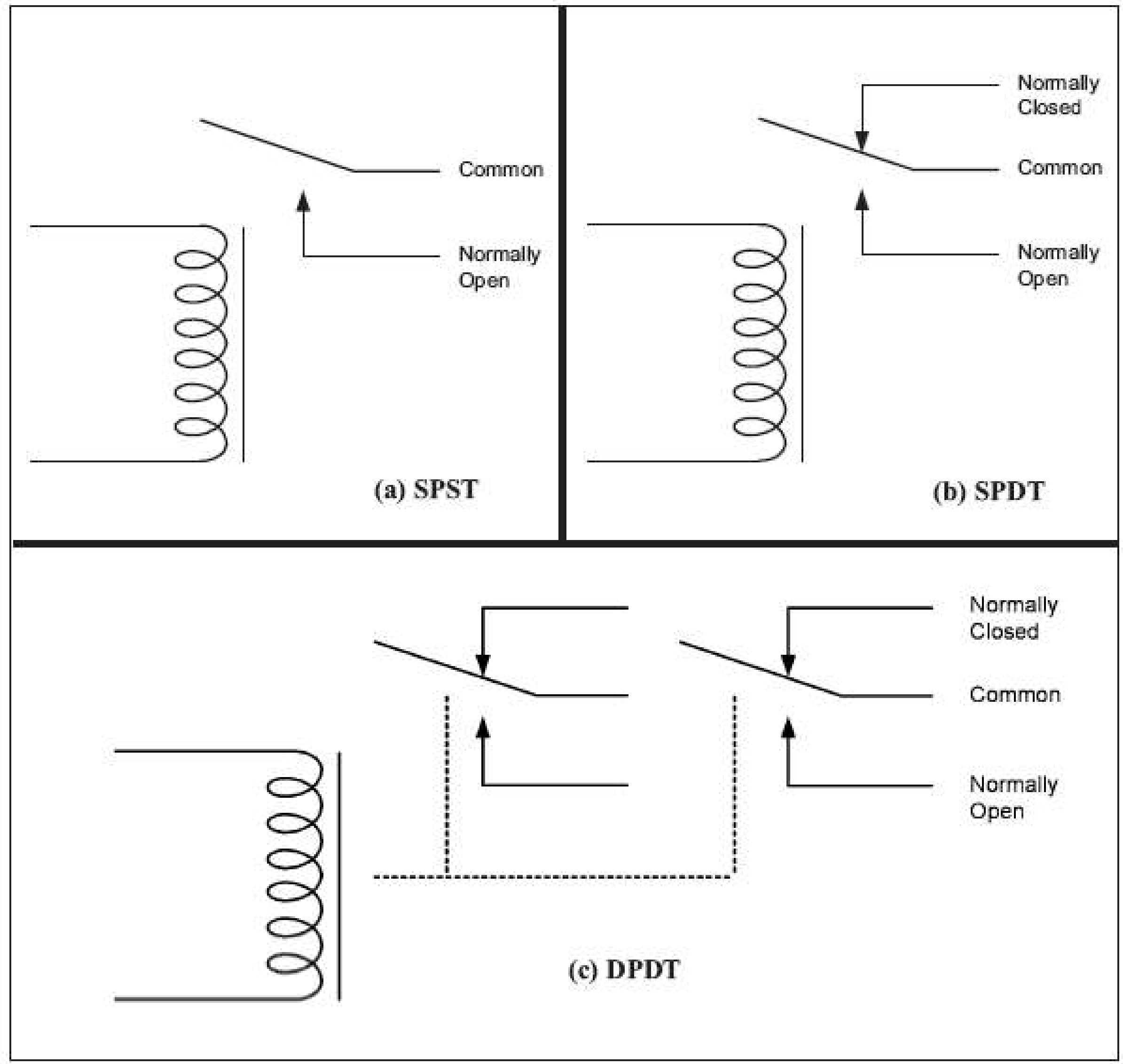


- SPDT

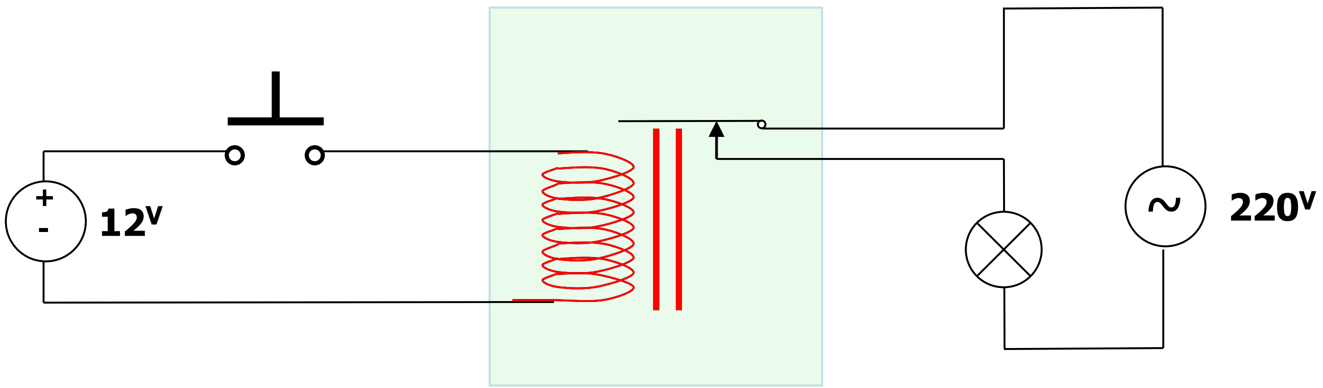


- DPDT

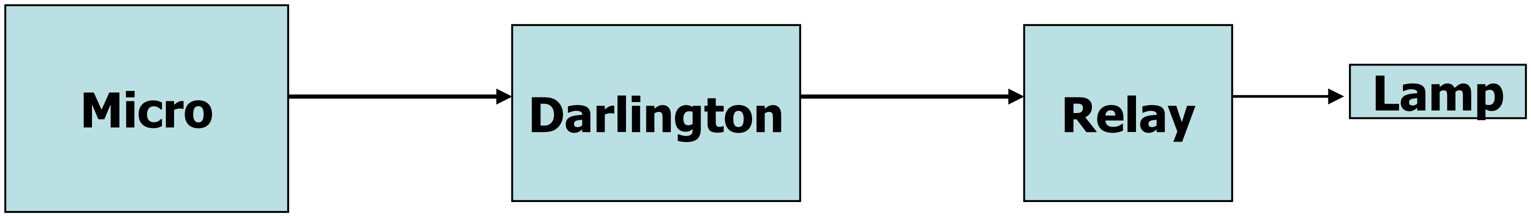


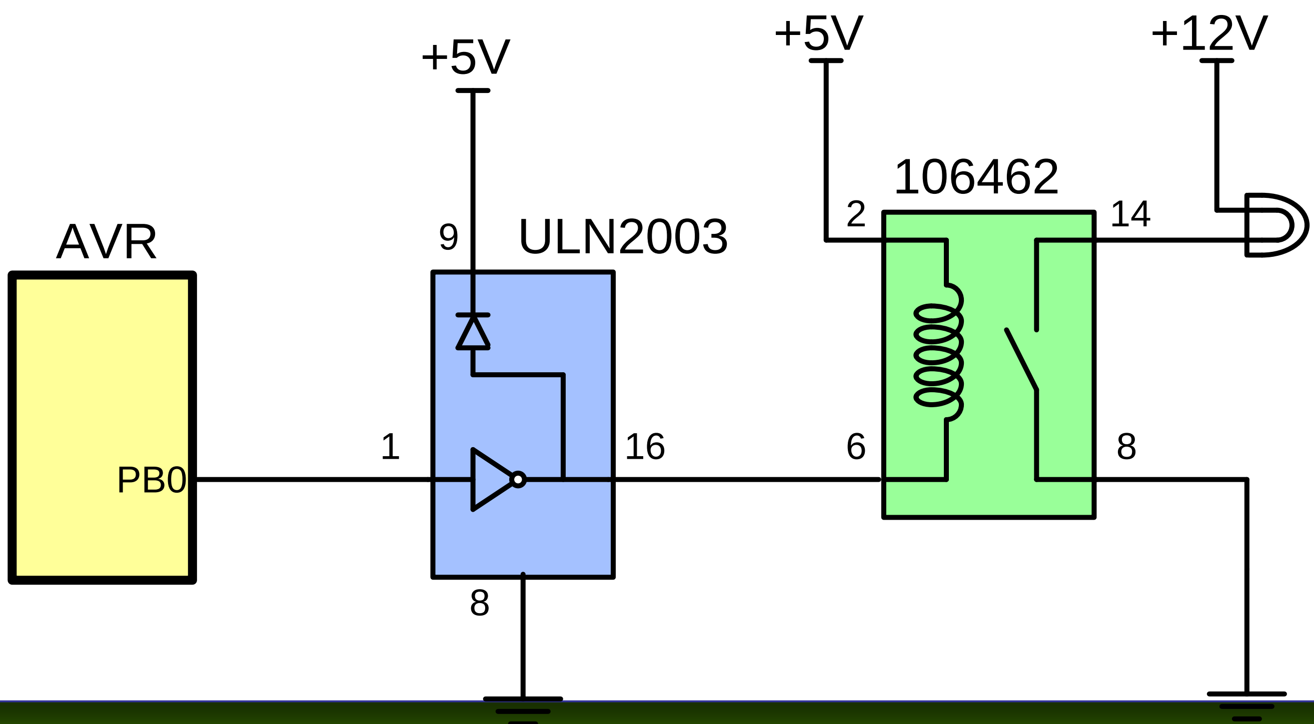


Relay



AVR connection to relay





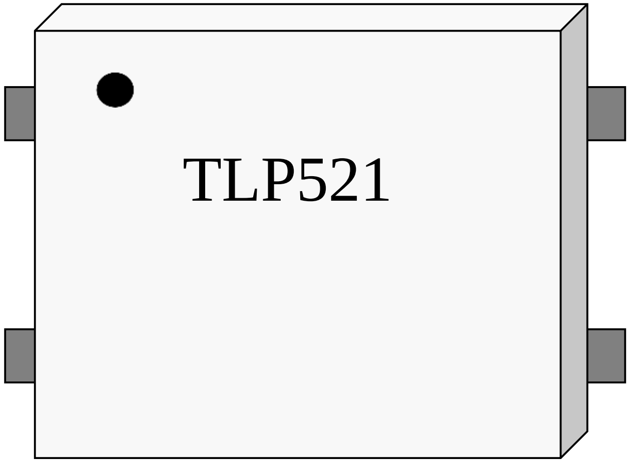
Contactors

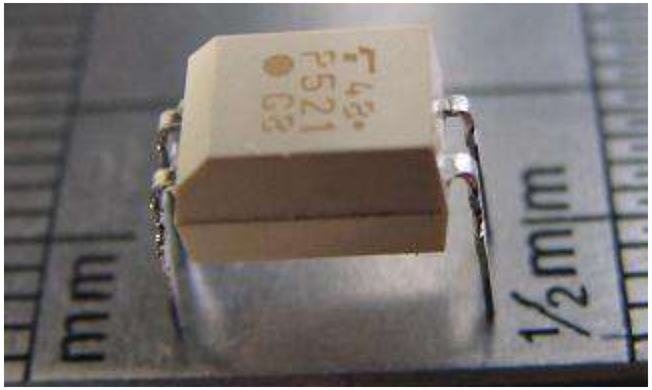
- Similar to relay
- switches larger amounts of current
- Bigger that relays and cannot be used on boards

Contactor
660V 50A

Optoisolator

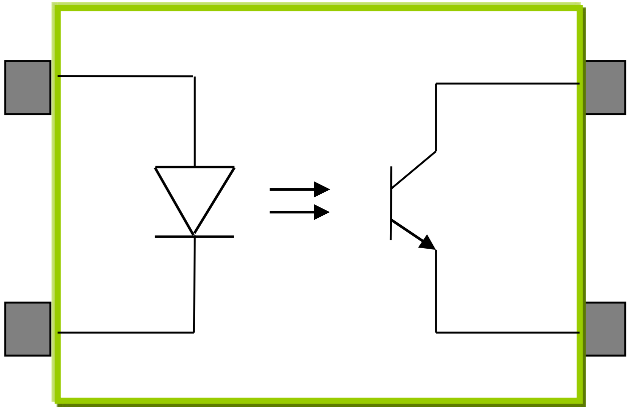
- Opto-isolator isolates two parts of a circuit from each other.
- There is an LED in the input, and a photo-transistor in the output. When the LED lights up, the photo-transistor, senses the light and becomes conductor, and passes the current.
- can be used in input or output circuits.

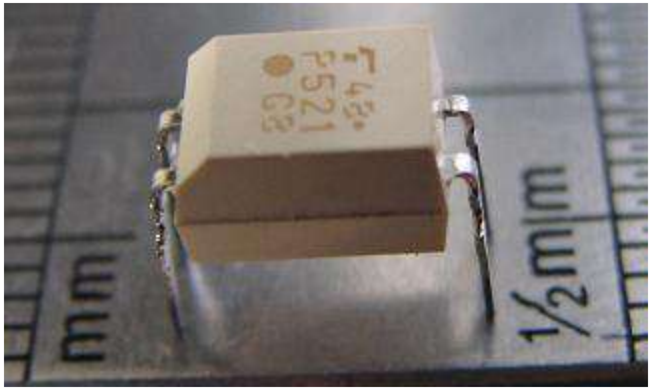




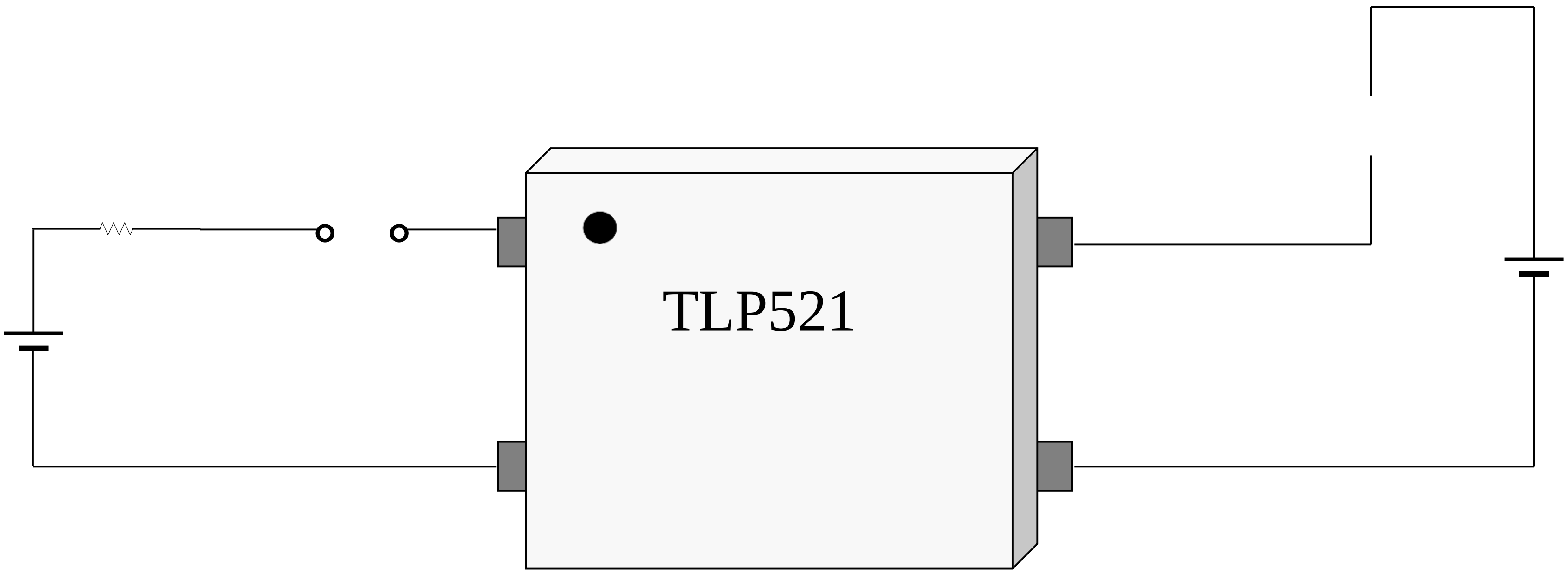
Optoisolator

- Opto-isolator isolates two parts of a circuit from each other.
- There is an LED in the input, and a photo-transistor in the output. When the LED lights up, the photo-transistor, senses the light and becomes conductor, and passes the current.
- can be used in input or output circuits.

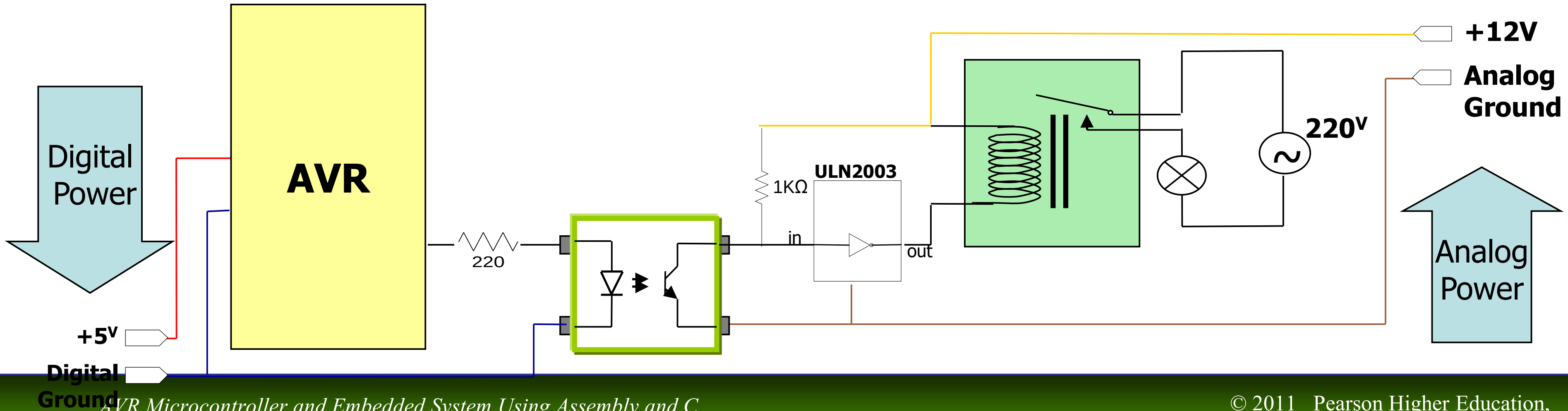




Opto-isolator

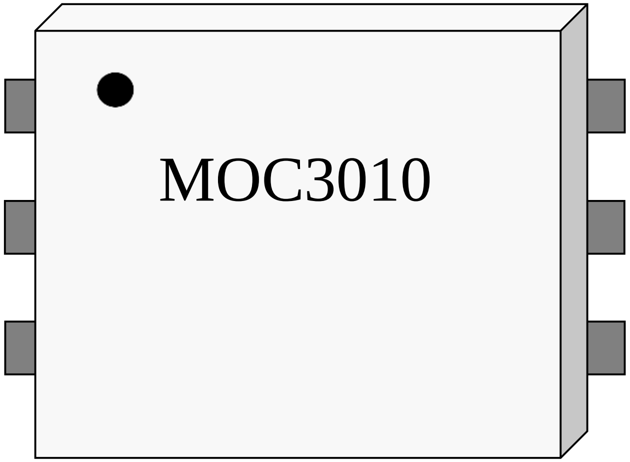


Controlling 220V devices



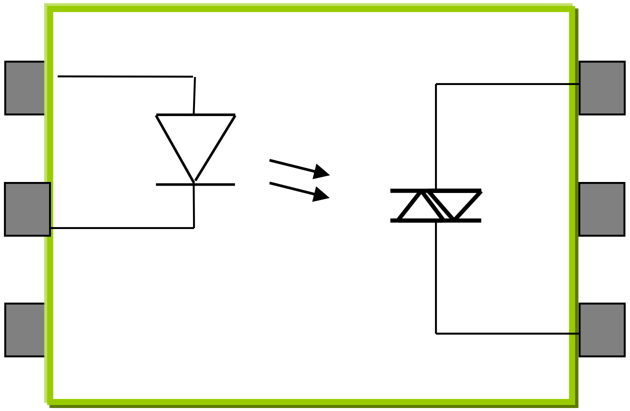
Opto-isolator Triac

- Common opto-isolators can drive only DC currents. But a kind of opto-isolator, called opto-isolator triac can switch AC currents as well.



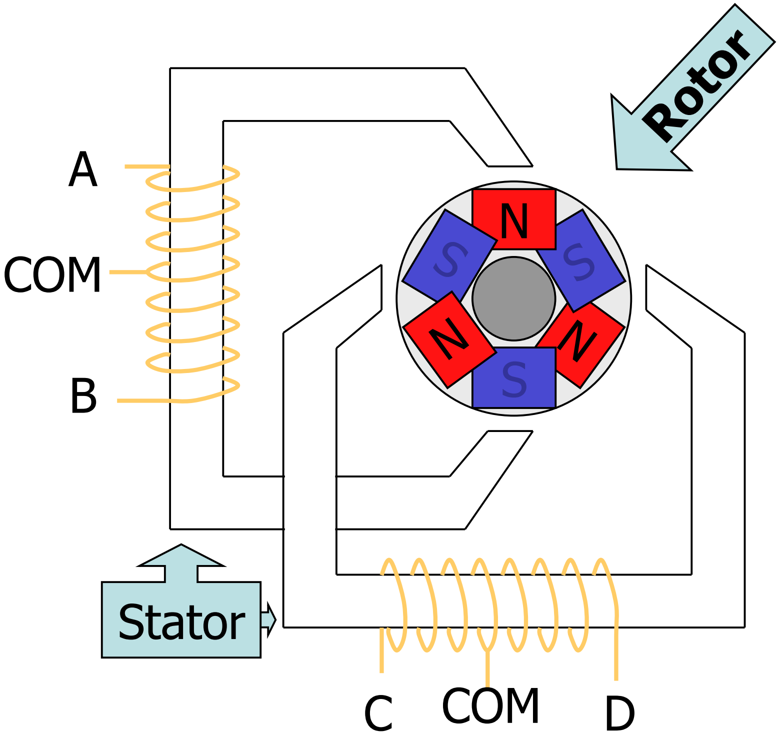
Opto-isolator Triac

- Common opto-isolators can drive only DC currents. But a kind of opto-isolator, called opto-isolator triac can switch AC currents as well.



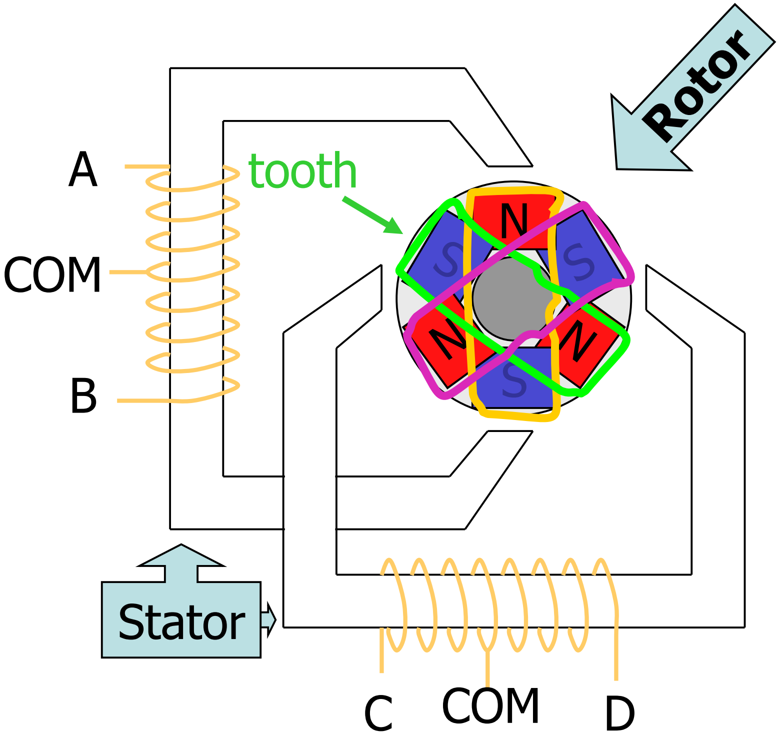
Stepper motor

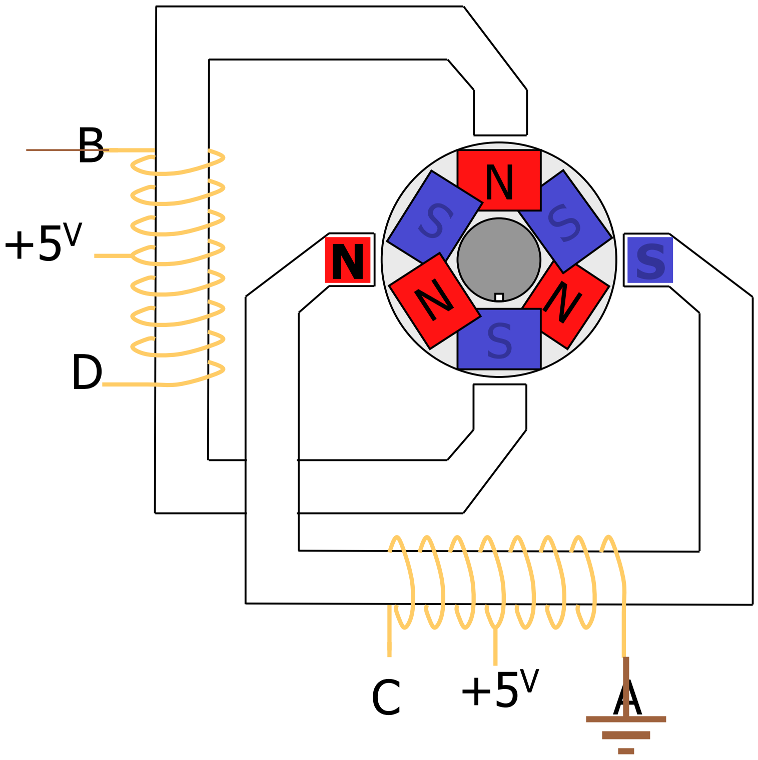
- Stepper motor is a motor, whose rotation angle is proportional to its input pulse.

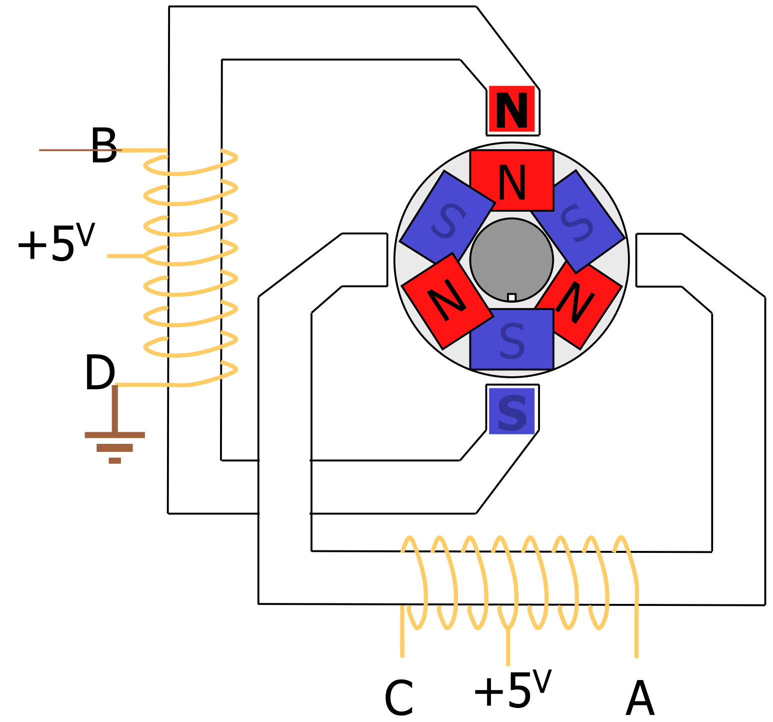


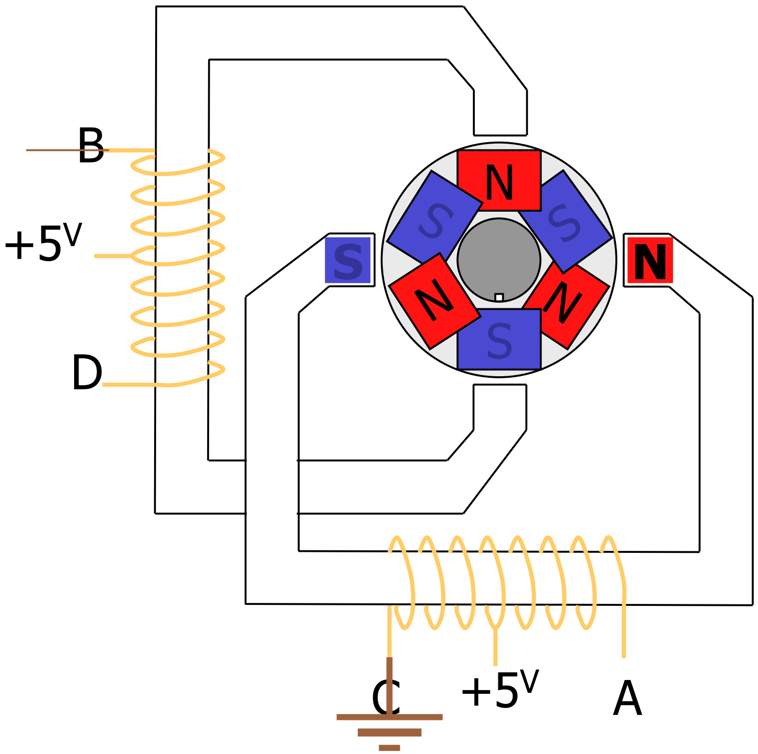
Stepper motor

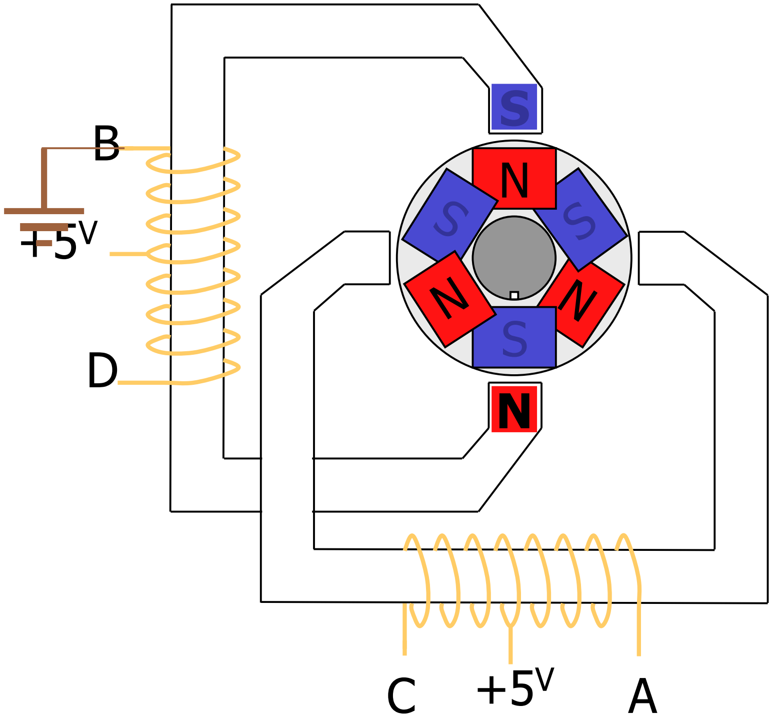
- Stepper motor is a motor, whose rotation angle is proportional to its input pulse.




B
+5V
D
N
S
S
N
S
N
N
S
C
+5V
A


N
B
+5V
N
S
S
N
N
S
D
S
C
+5V
A


B
+5V
D
N
S
S
S
N
N
S
N
C
+5V
A


S
B
N
S
S
+5V
N
N
S
D
N
C
+5V
A

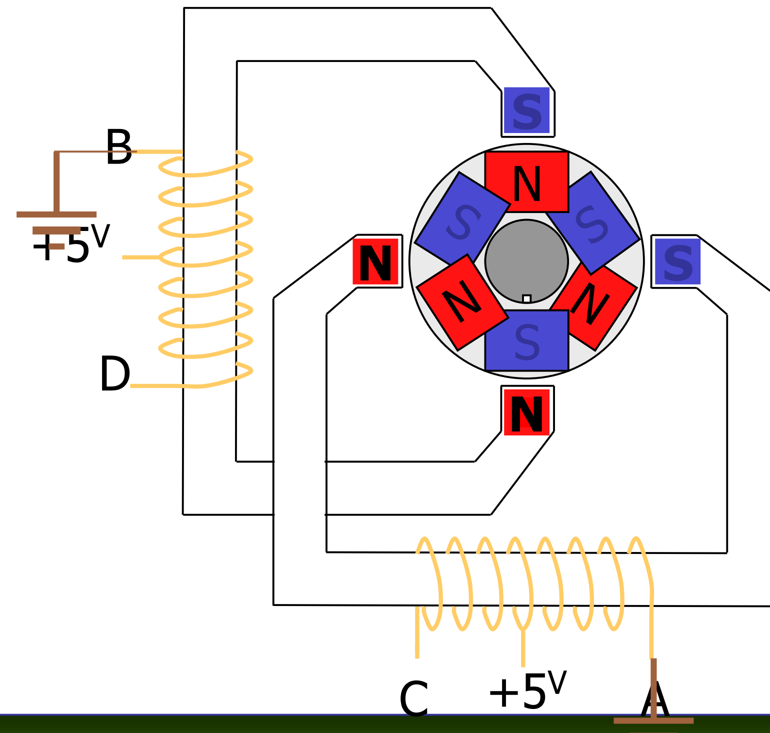
Stepping

Table 14-3: Normal Four-Step Sequence

Clockwise ↓

Step #	Winding A	Winding B	Winding C	Winding D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

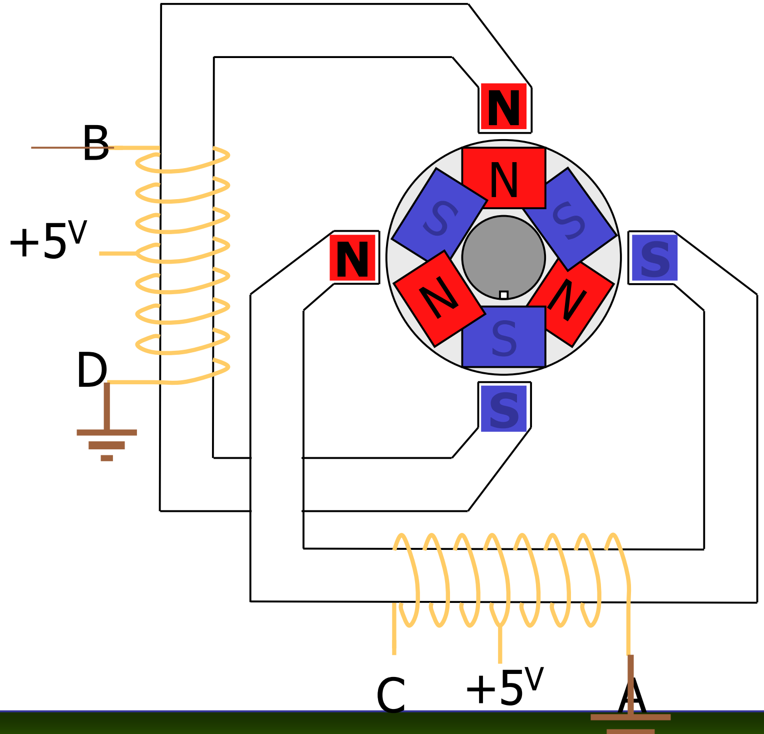
Counter-clockwise ↑



Stepping

Table 14-3: Normal Four-Step Sequence

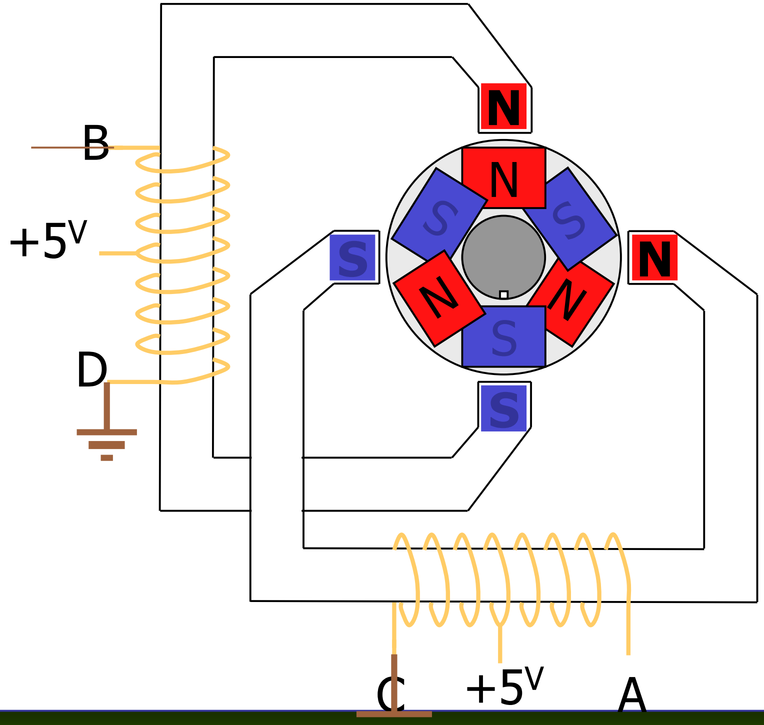
Clockwise	Step #	Winding A	Winding B	Winding C	Winding D	Counter-clockwise
↓	1	1	0	0	1	↑
	2	1	1	0	0	
	3	0	1	1	0	
	4	0	0	1	1	



Stepping

Table 14-3: Normal Four-Step Sequence

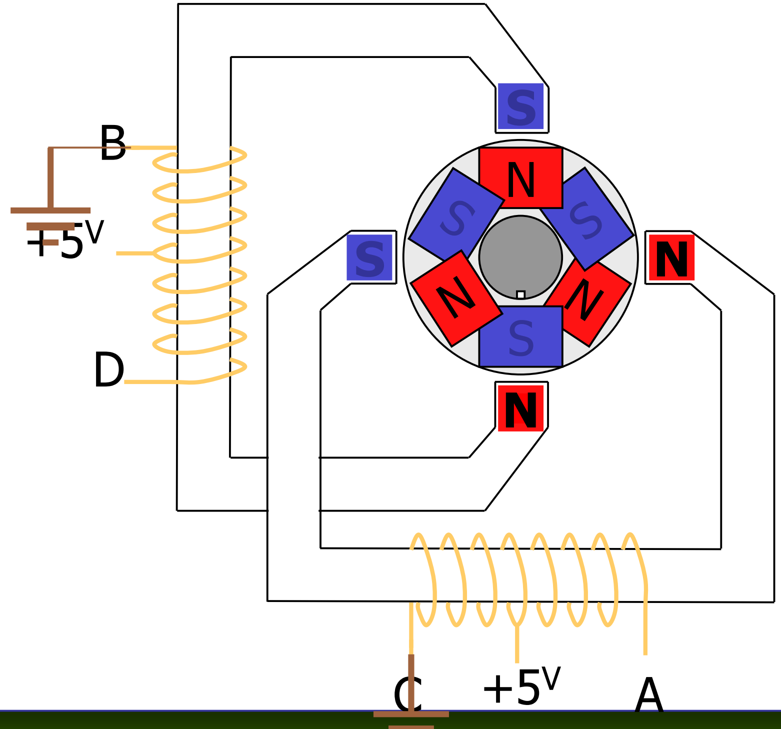
Clockwise	Step #	Winding A	Winding B	Winding C	Winding D	Counter-clockwise
↓	1	1	0	0	1	↑
	2	1	1	0	0	
	3	0	1	1	0	
	4	0	0	1	1	



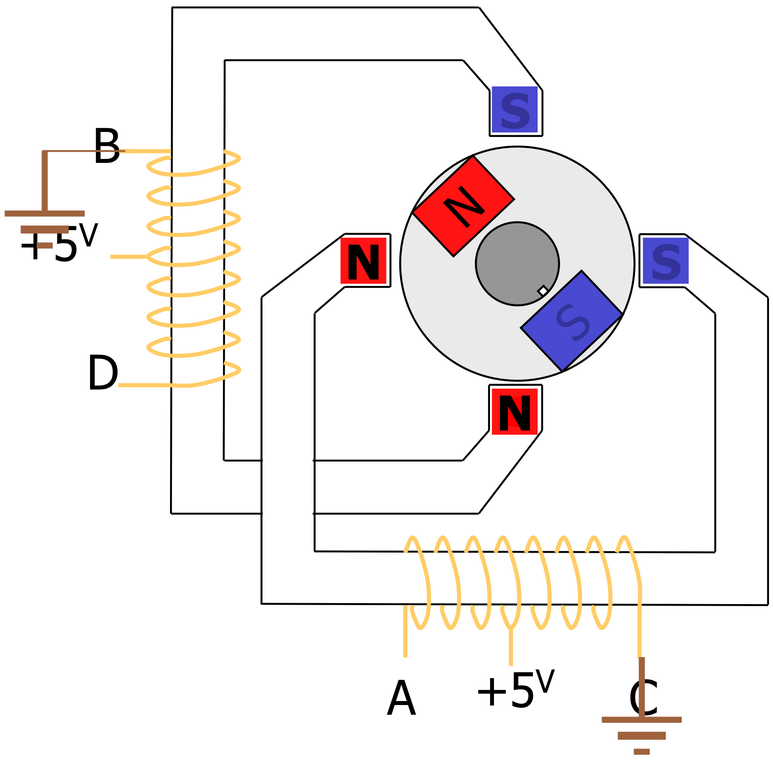
Stepping

Table 14-3: Normal Four-Step Sequence

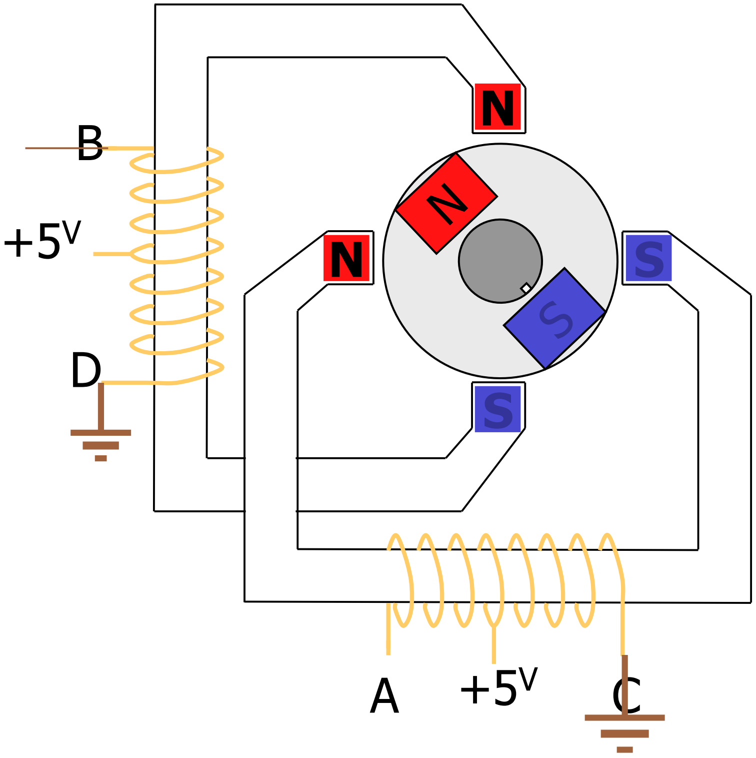
Clockwise	Step #	Winding A	Winding B	Winding C	Winding D	Counter-clockwise
↓	1	1	0	0	1	↑
	2	1	1	0	0	
	3	0	1	1	0	
	4	0	0	1	1	



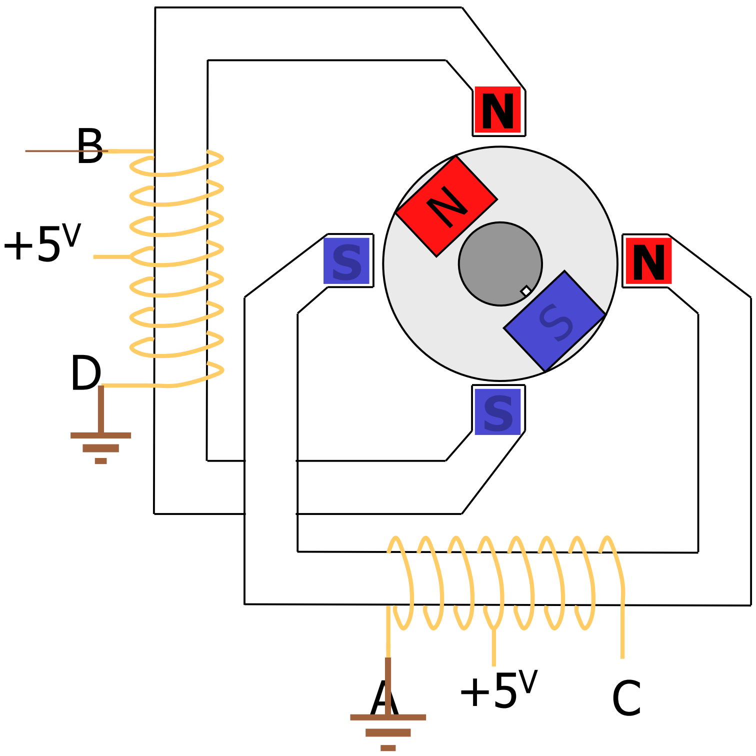
Calculations



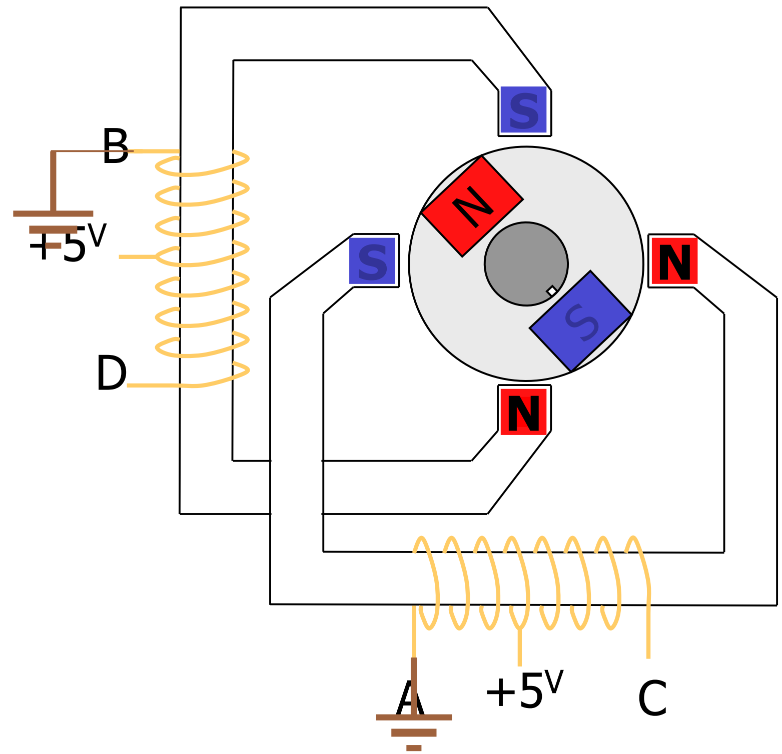
Calculations



Calculations



Calculations

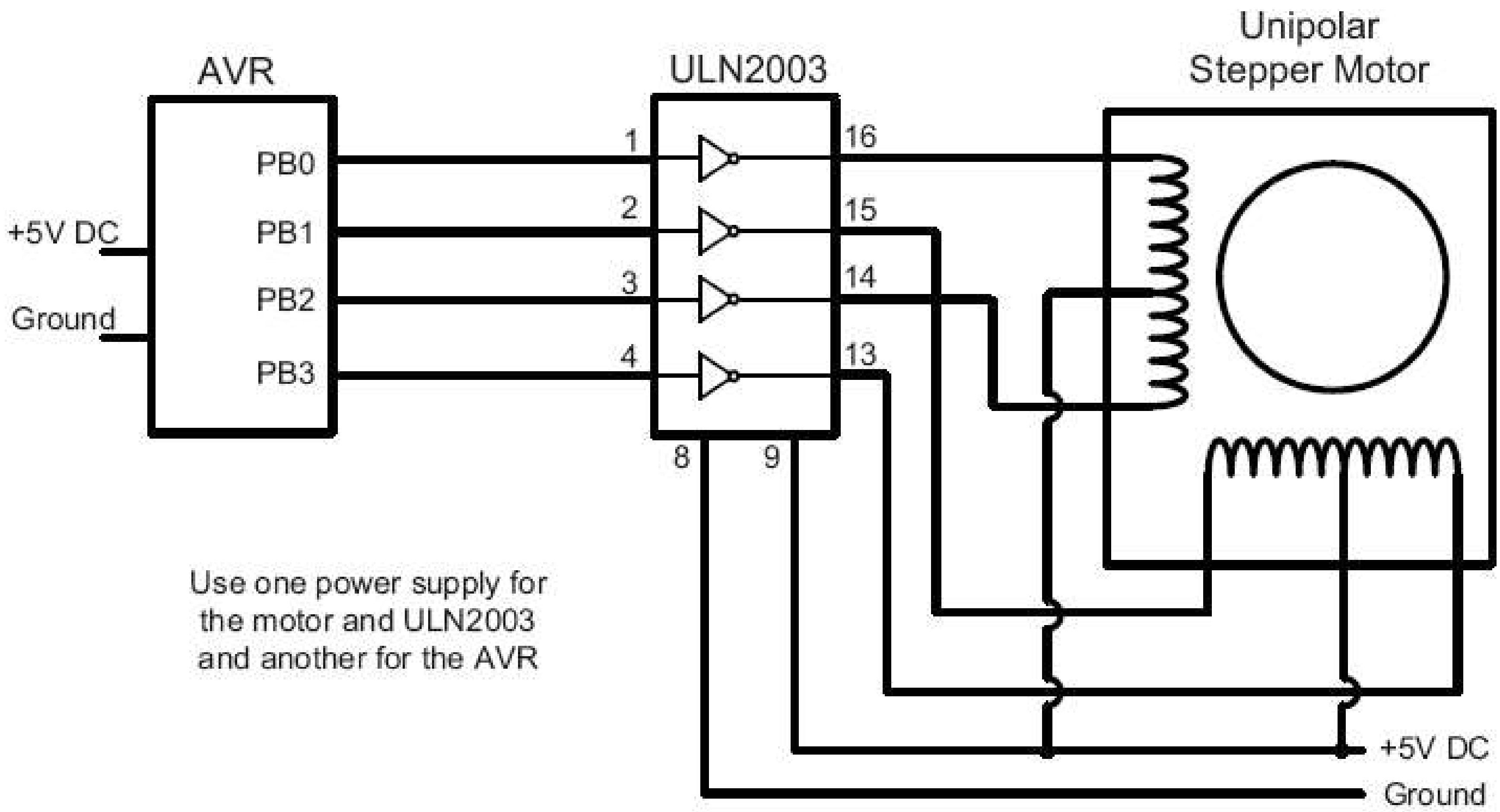


$$\text{Step angle} = \frac{90}{\text{Num of teeth}}$$

$$\text{Steps per revolution} = \frac{360}{\text{Step angle}}$$ = 4 * Num of teeth

$$\text{Steps per second} = \frac{\text{RPM * Steps per revolution}}{60}$$

AVR and a Stepper motor



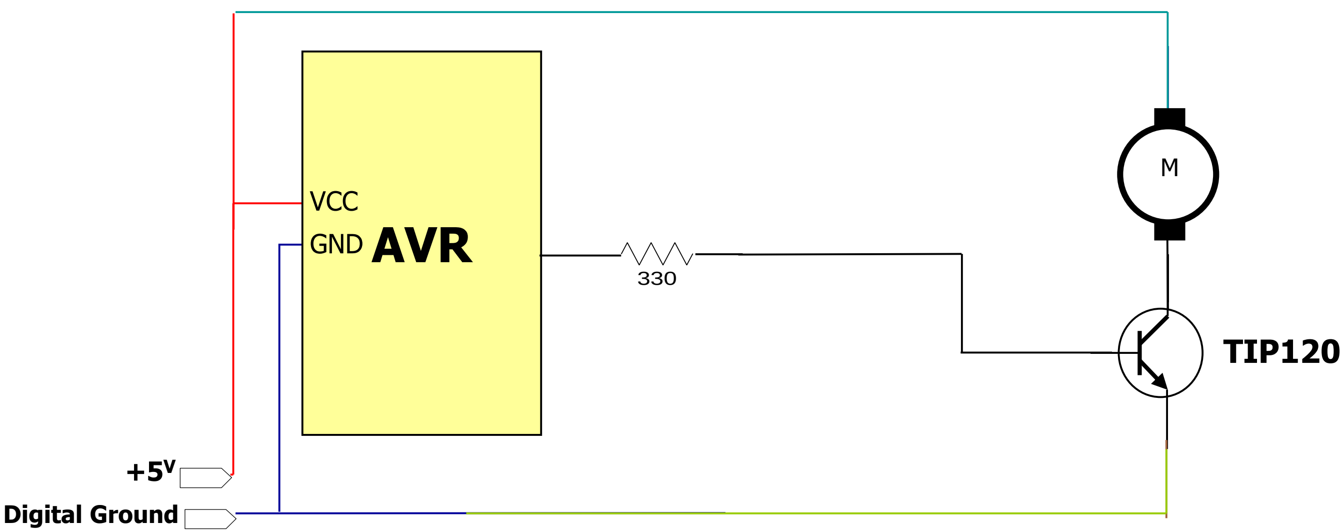
DC motor and PWM

The AVR microcontroller and embedded systems using assembly and c

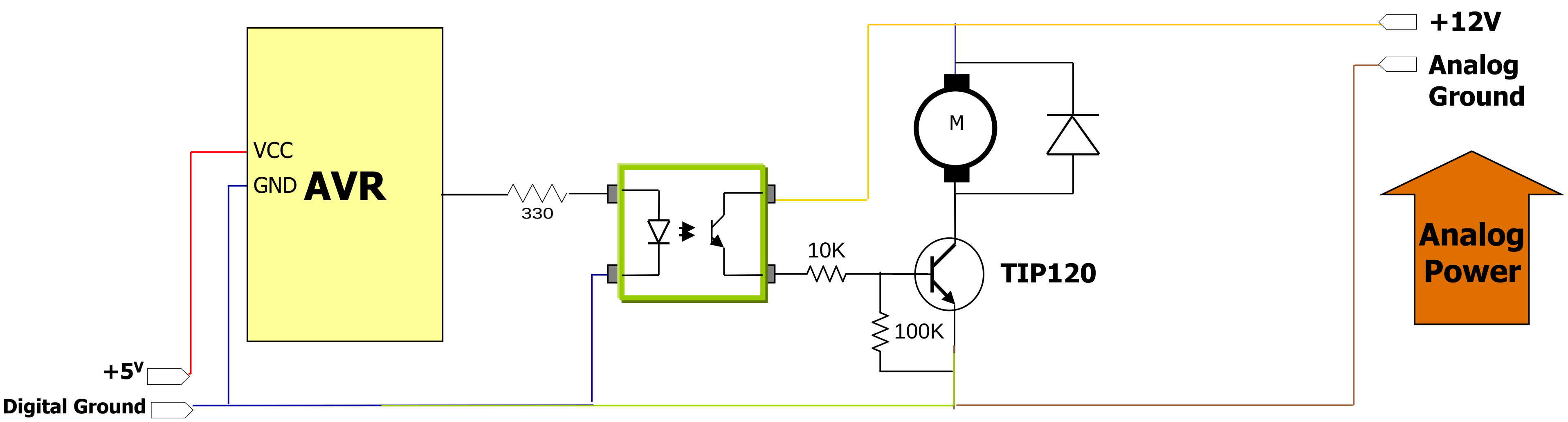
Topics

- DC motor
  - Unidirectional control
  - Bidirectional control
- PWM modes
  - Wave generating using Fast PWM
  - Wave generating using Phase correct PWM

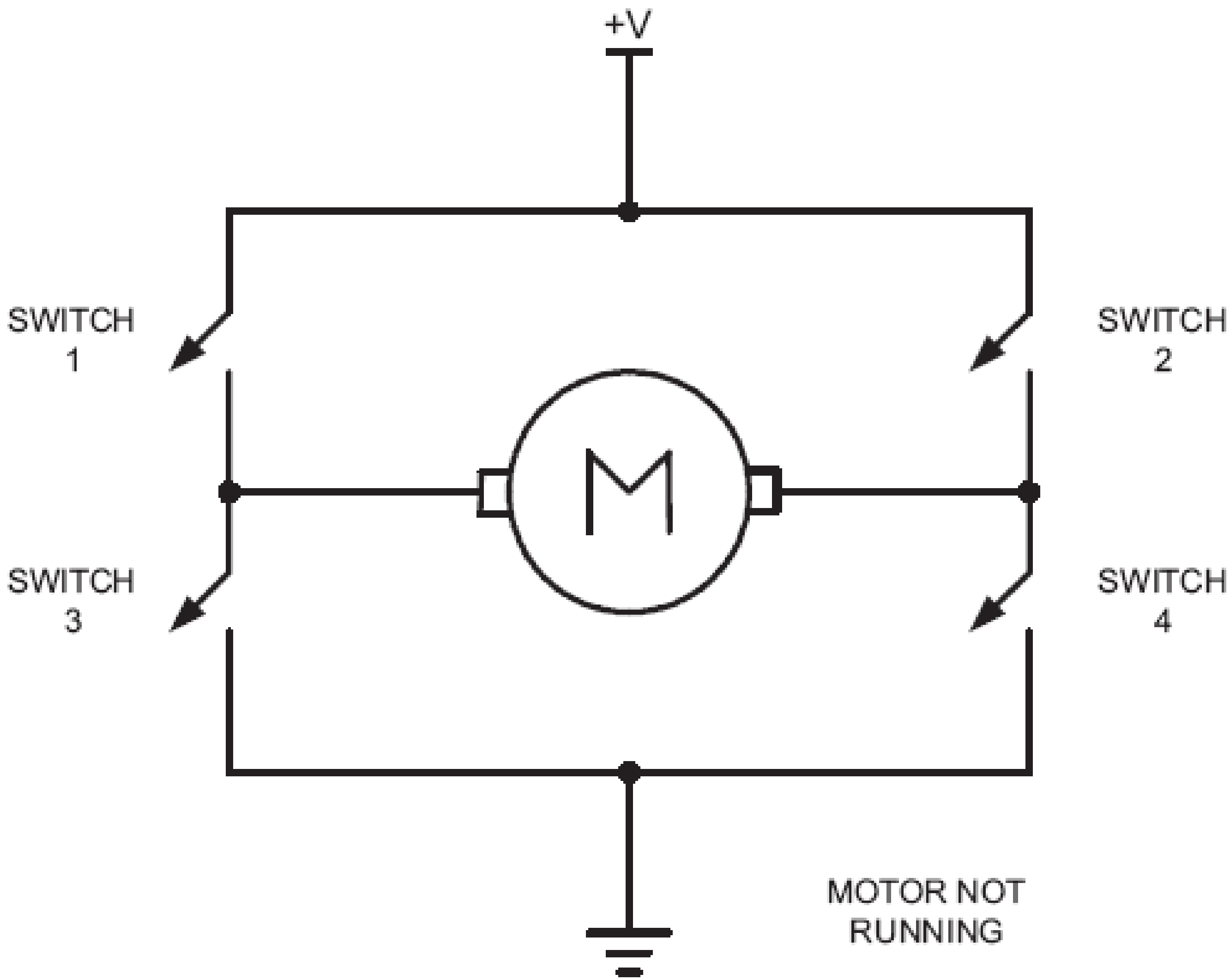
Unidirectional control



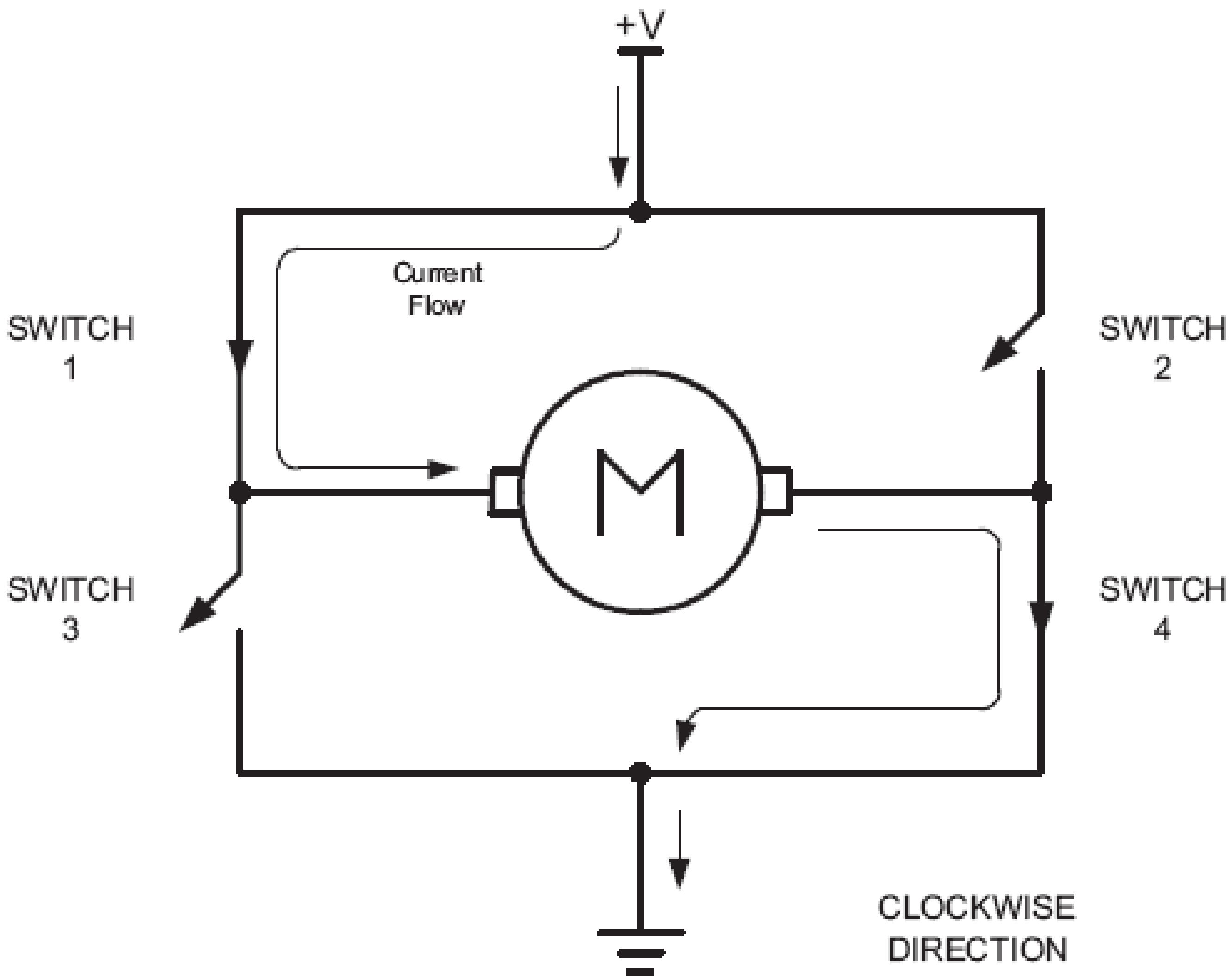
Unidirectional control



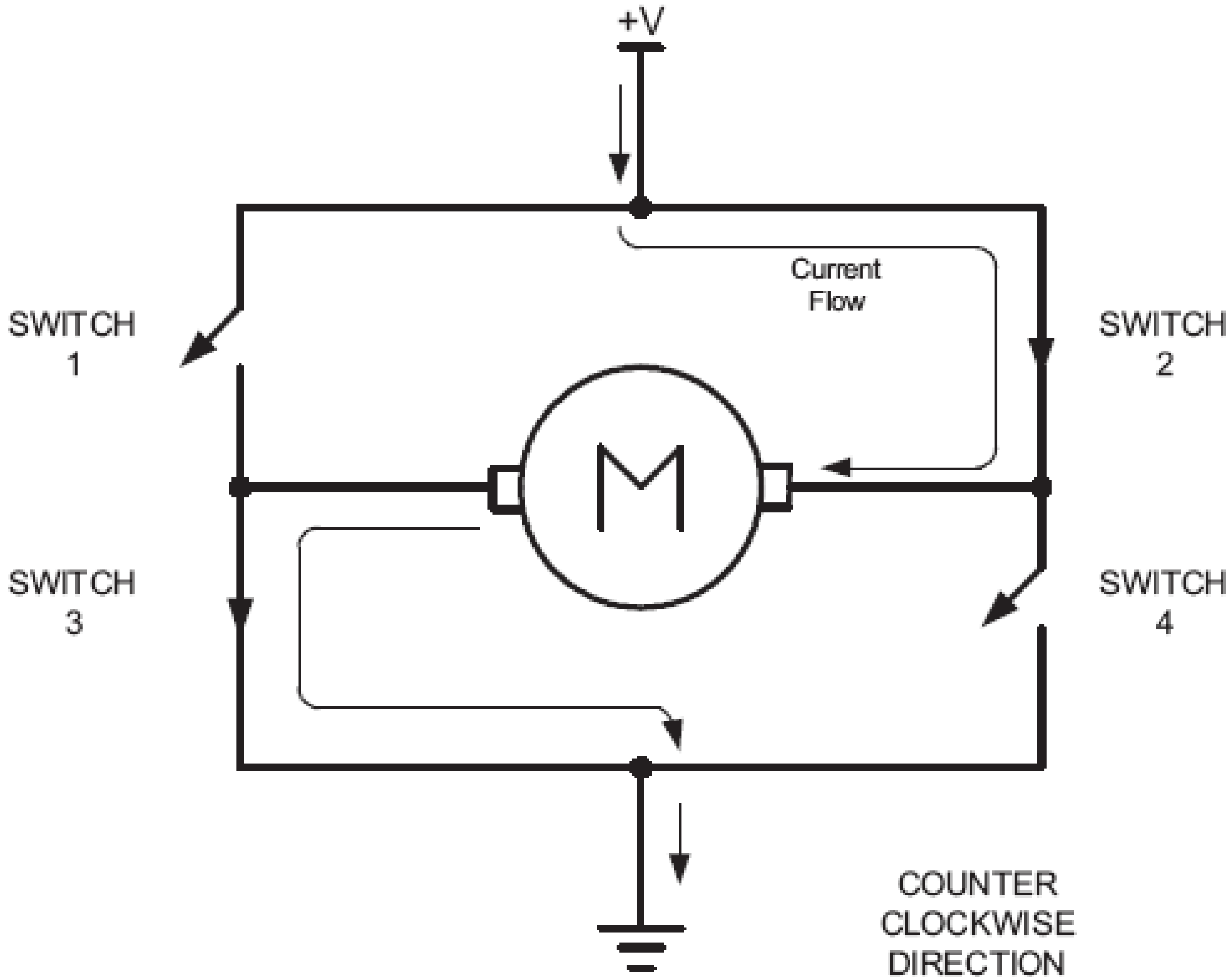
Bidirectional control



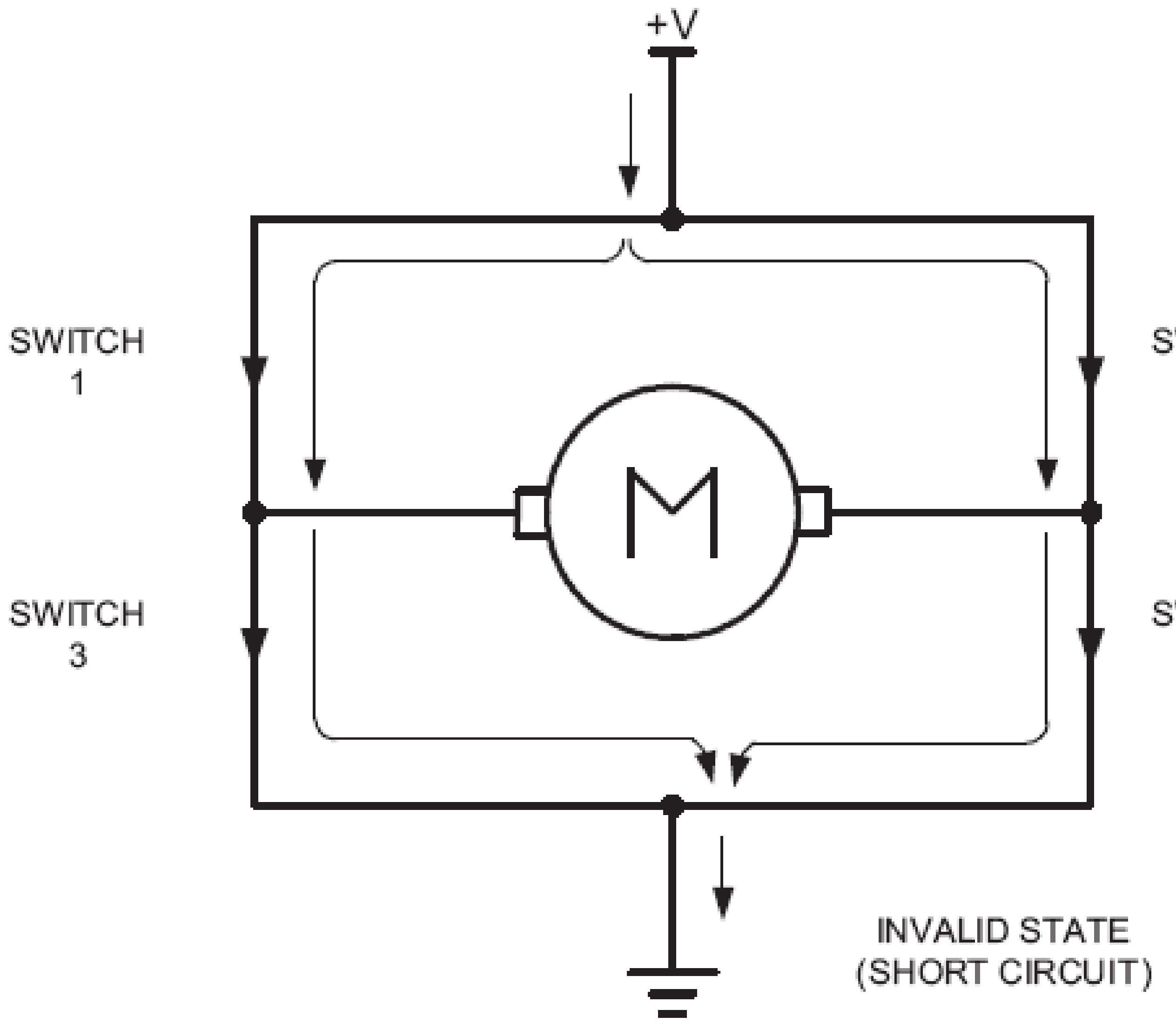
Bidirectional (clock wise)



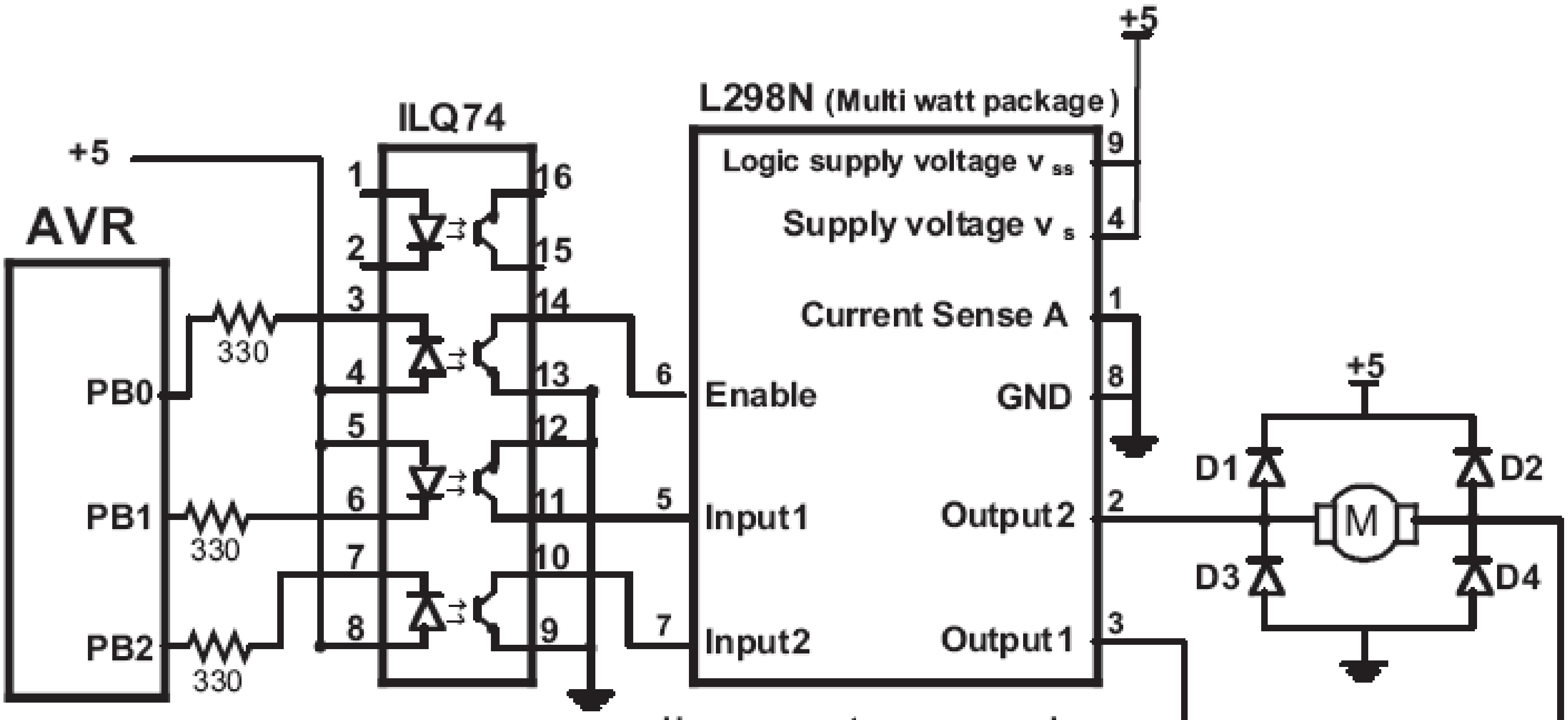
Bidirectional (counter clockwise)



Bidirectional



Using L298N

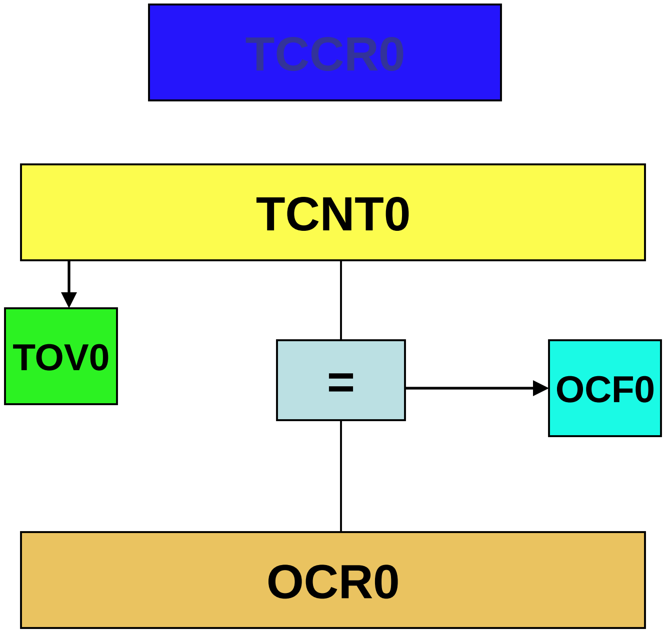


The optoisolator provides additional protection of the AVR

Use a separate power supply for the motor and L 298N than for the AVR

D1, D2, D3, D4 are 1N4004

Timer0 Review



OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

Clock Selector (CS)

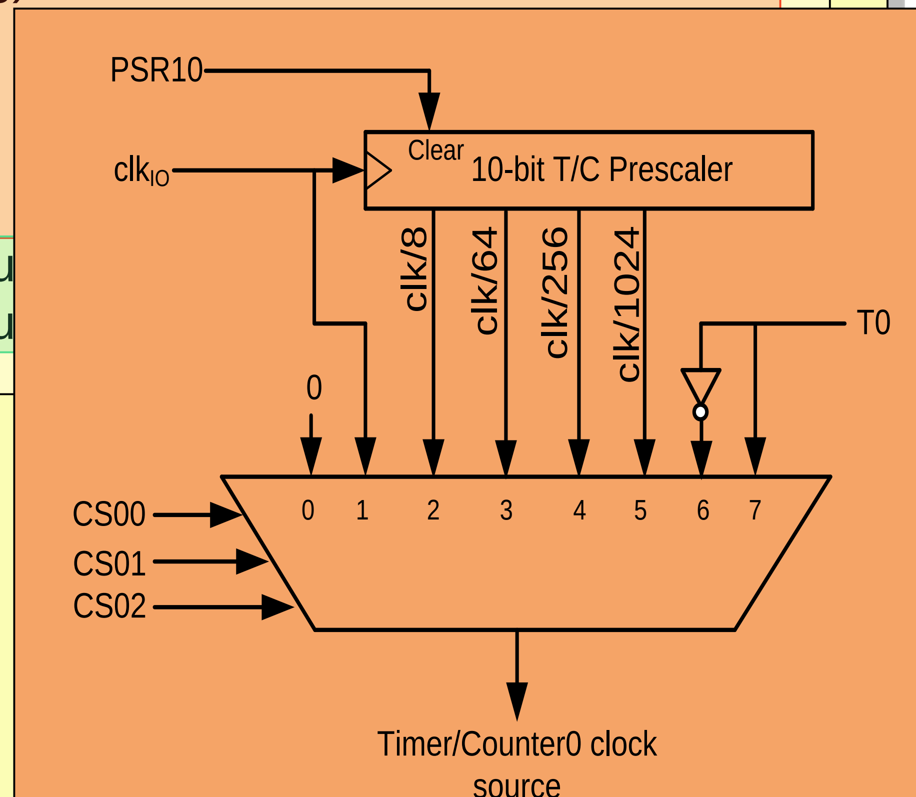
CS02	CS01	CS00	Comment
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Clock Selector (CS)

CS02	CS01	CS00	Comment
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock sou
1	1	1	External clock sou



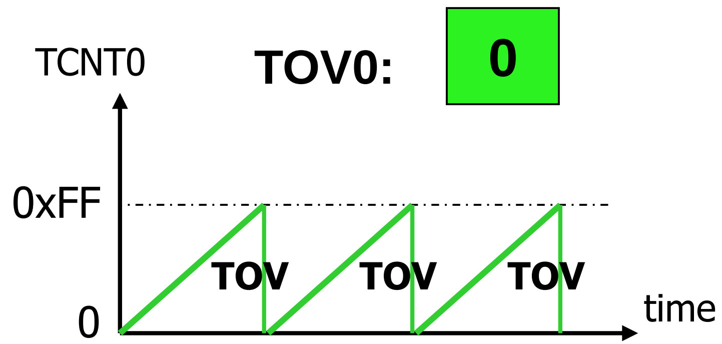
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

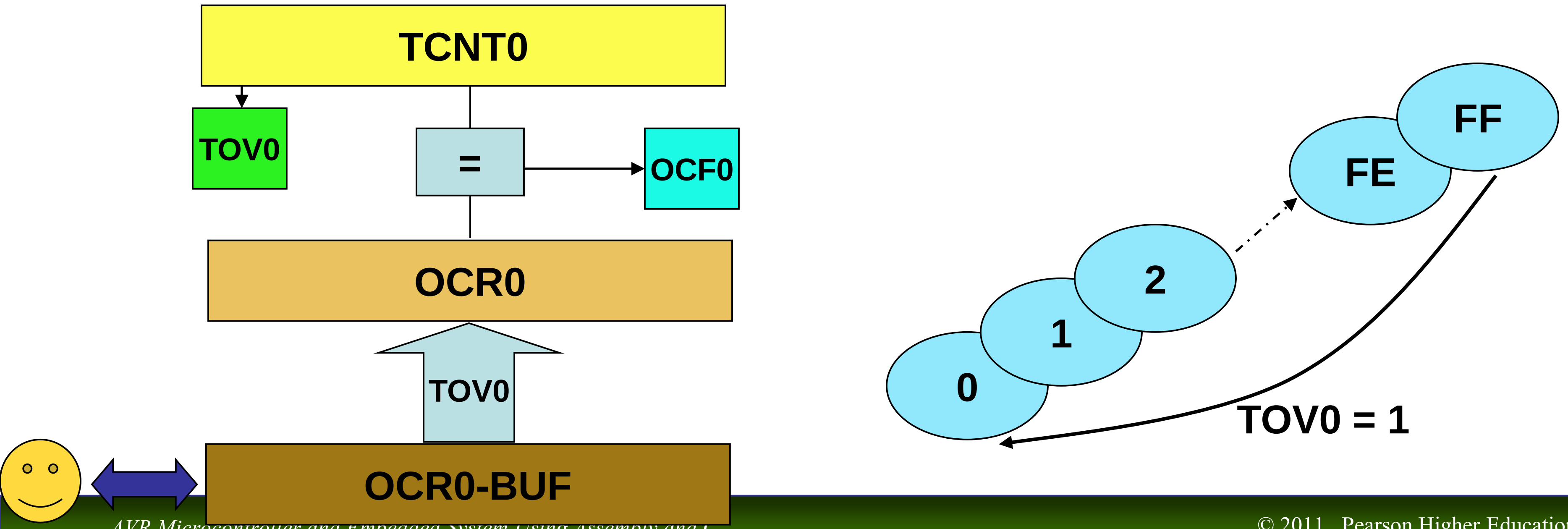
Timer Mode (WGM)

WGM00	WGM01	Comment
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

Fast PWM mode

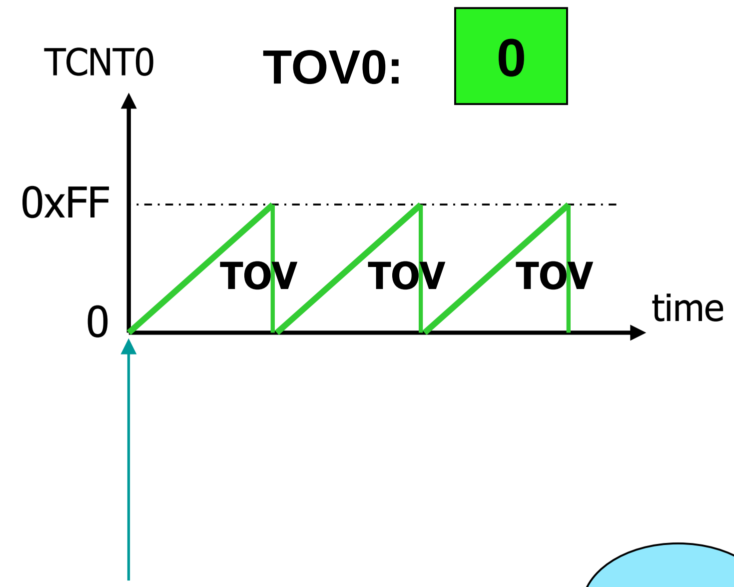
- Similar to Normal mode but OCR0 is buffered.

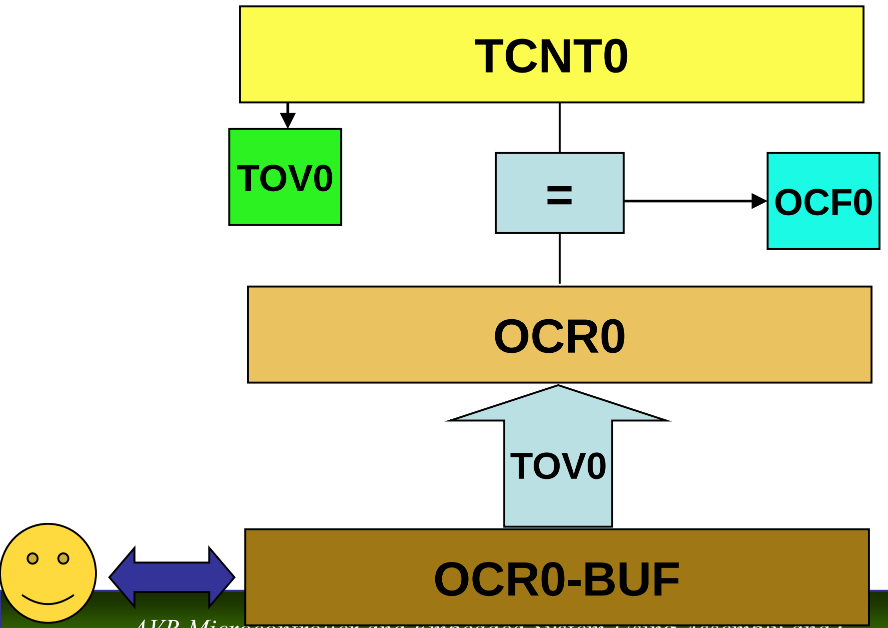


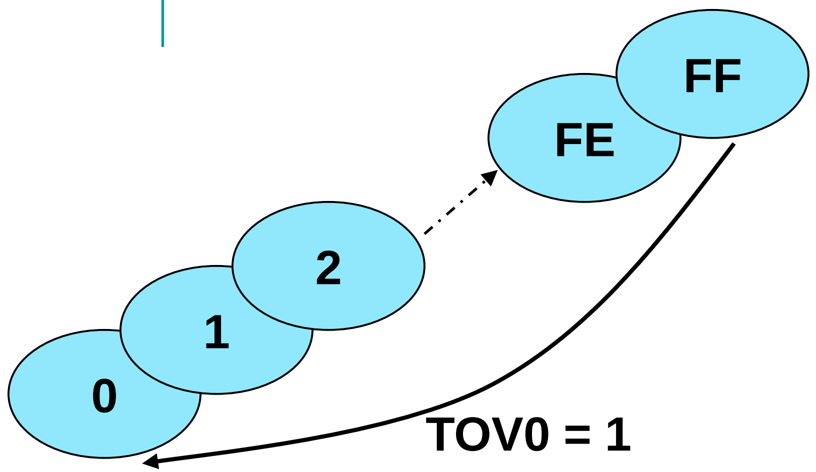


Fast PWM mode

- Similar to Normal mode but OCR0 is buffered.

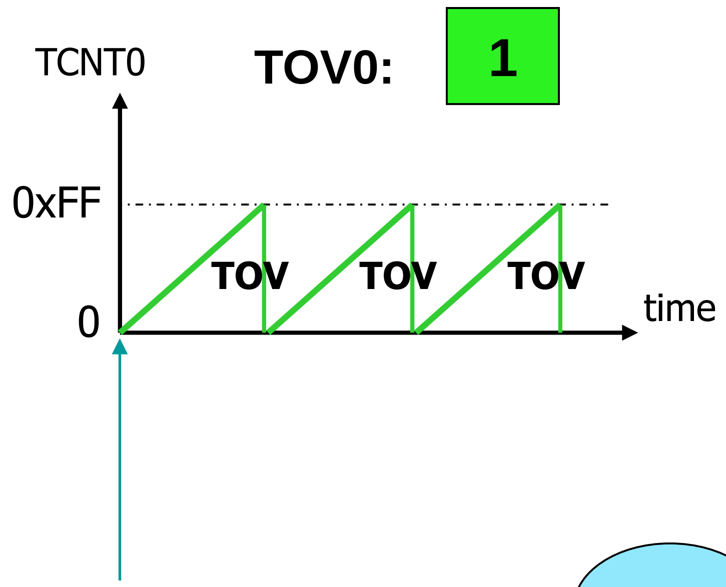


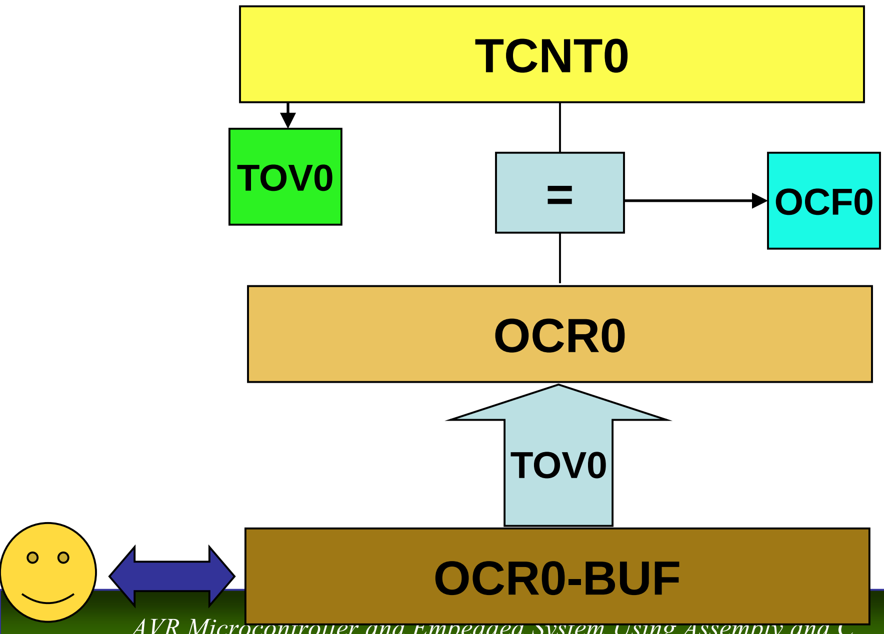


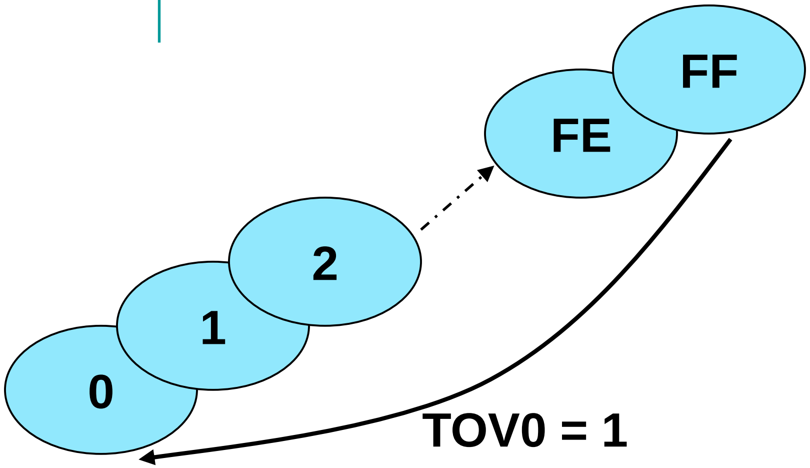


Fast PWM mode

- Similar to Normal mode but OCR0 is buffered.

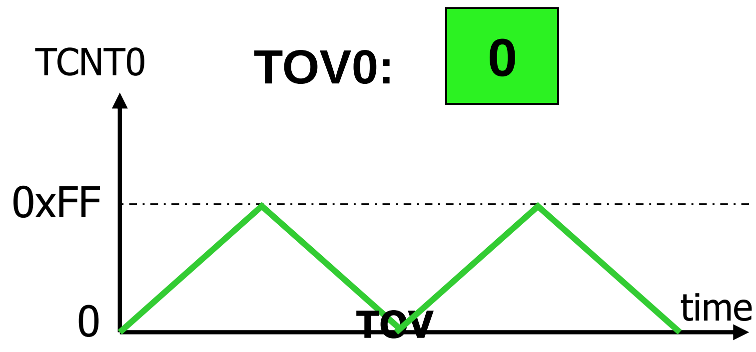


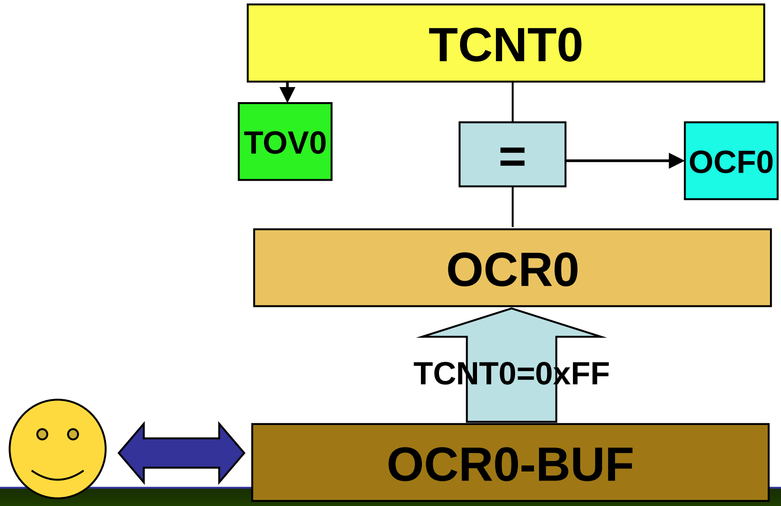


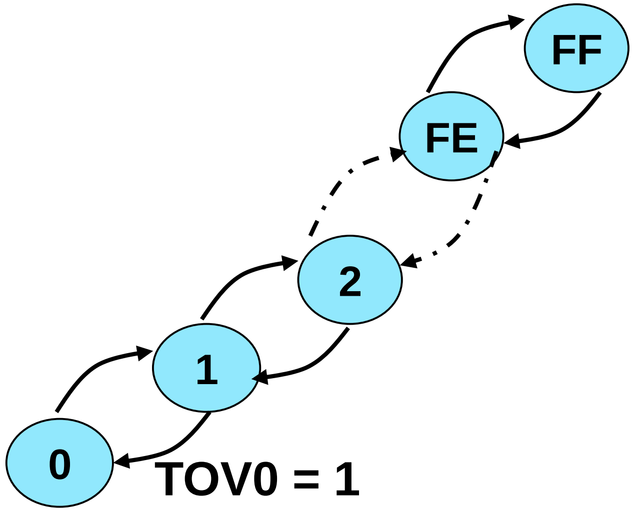


Phase Correct PWM mode

- Goes up and down like a yo-yo
- When TCNT becomes zero, the TOV0 flag sets.

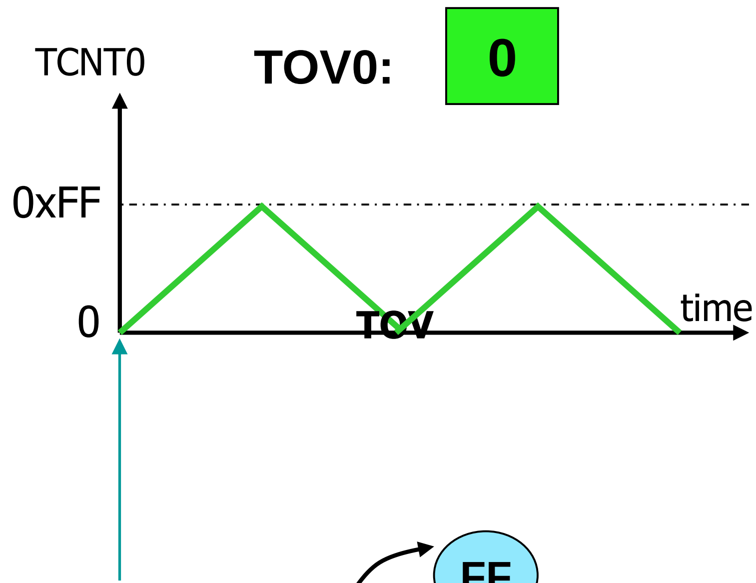


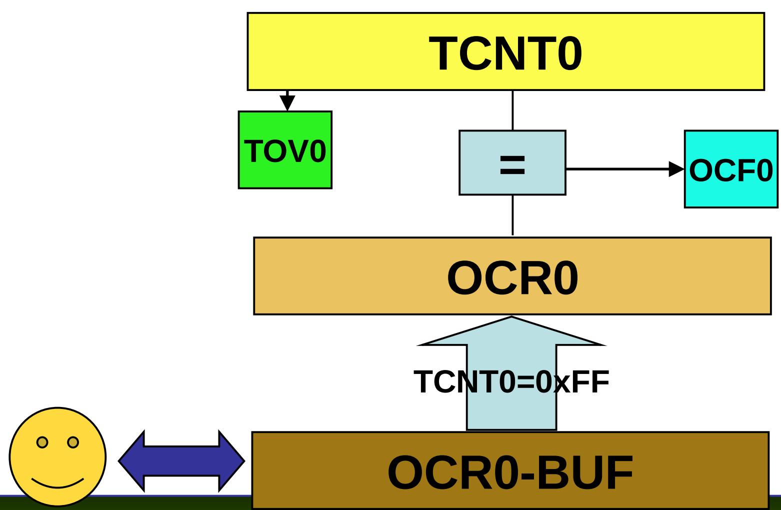


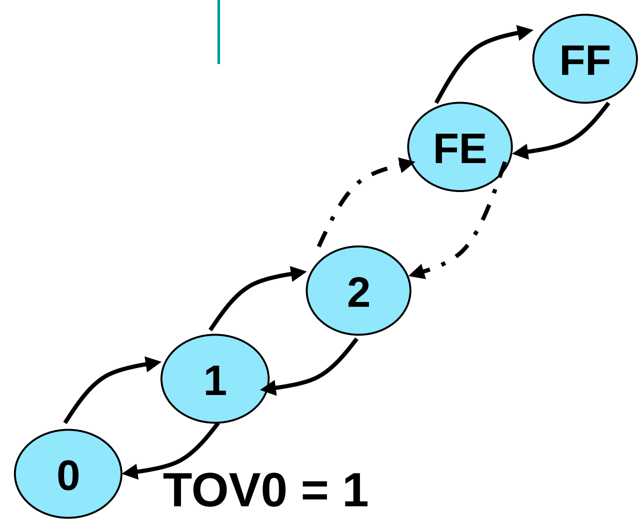


Phase Correct PWM mode

- Goes up and down like a yo-yo
- When TCNT becomes zero, the TOV0 flag sets.

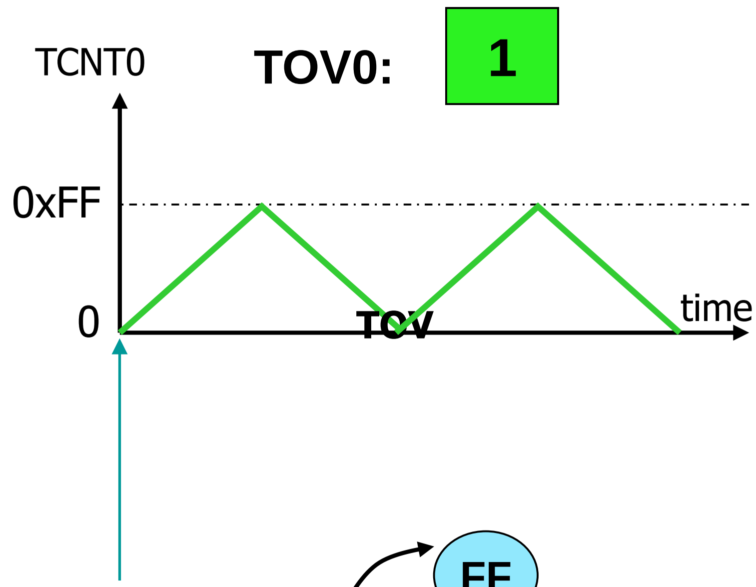


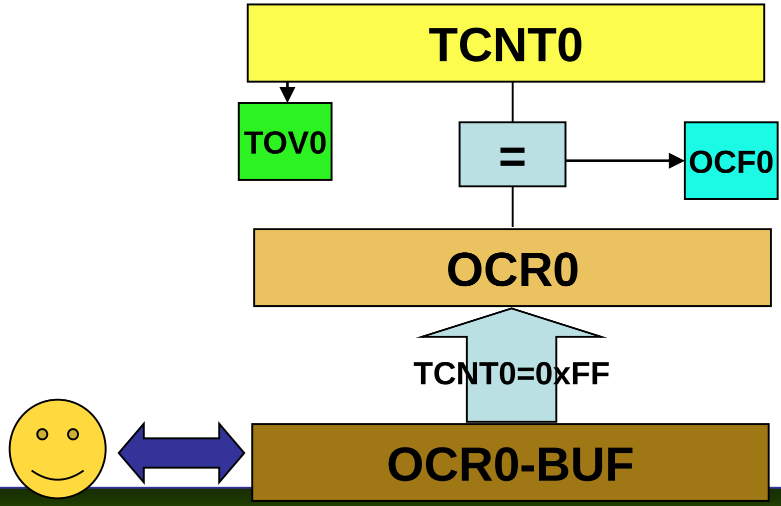


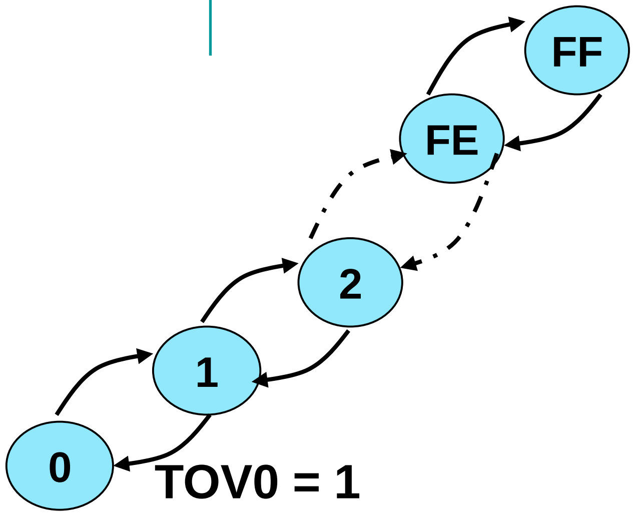


Phase Correct PWM mode

- Goes up and down like a yo-yo
- When TCNT becomes zero, the TOV0 flag sets.







FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Compare Output Mode (COM)

CTC or Normal (Non PWM)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Fast PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

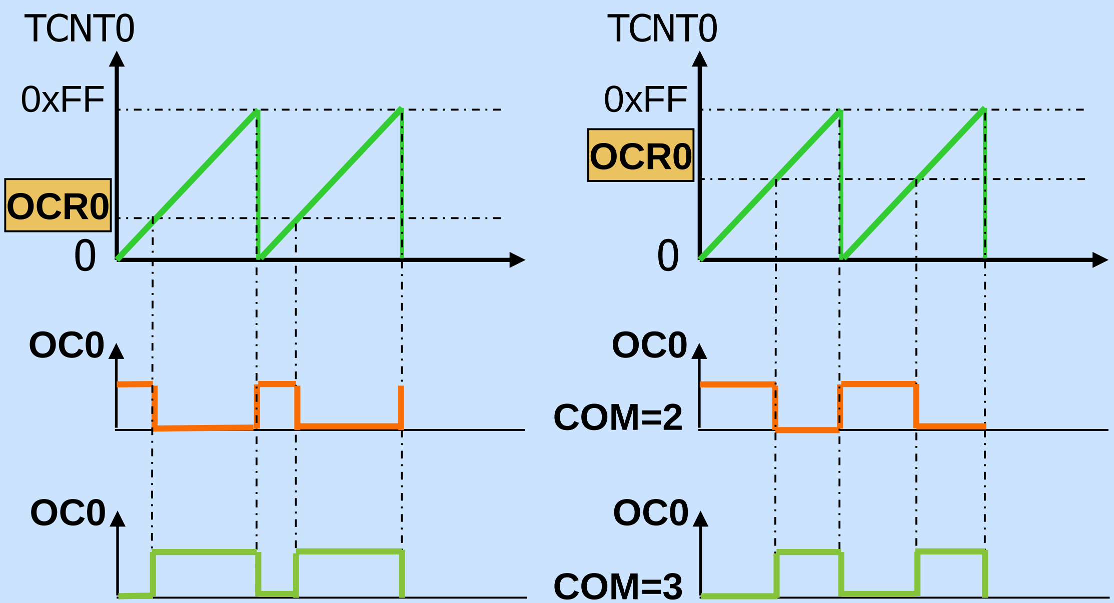
Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 73 for more details.

Phase Correct PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

Fast PWM
Duty cycle = changeable (0% to 100%)
Frequency = selectable between limited choices



$$\text{Duty Cycle} = \frac{OCR0 + 1}{256} \times 100$$

$$\text{Duty Cycle} = \frac{255 - OCR0}{256} \times 100$$

$$F_{OC0} = \frac{f_{clk}}{N(256)}$$

Assuming XTAL = 8 MHz, make the following pulse duty cycle = 75% and frequency = 31.250KHz

$$F_{OC0} = \frac{f_{clk}}{N(256)} \Rightarrow 31.250KHz = \frac{8MHz}{N(256)} \Rightarrow N = \frac{8MHz}{31.250K*256} = 1$$

75/100 = (OCR0+1)/255 → OCR0+1 = 191 = 0xBF → OCR0 = 0xBE

```
LDI R20,0xBE
OUT OCR0,R20
LDI R20,0x79
OUT TCCR0,R20
```

```
OCR0 = 0xBE;
TCCR0 = 0x79;
```

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Compare Output Mode (COM)

CTC or Normal (Non PWM)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Fast PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

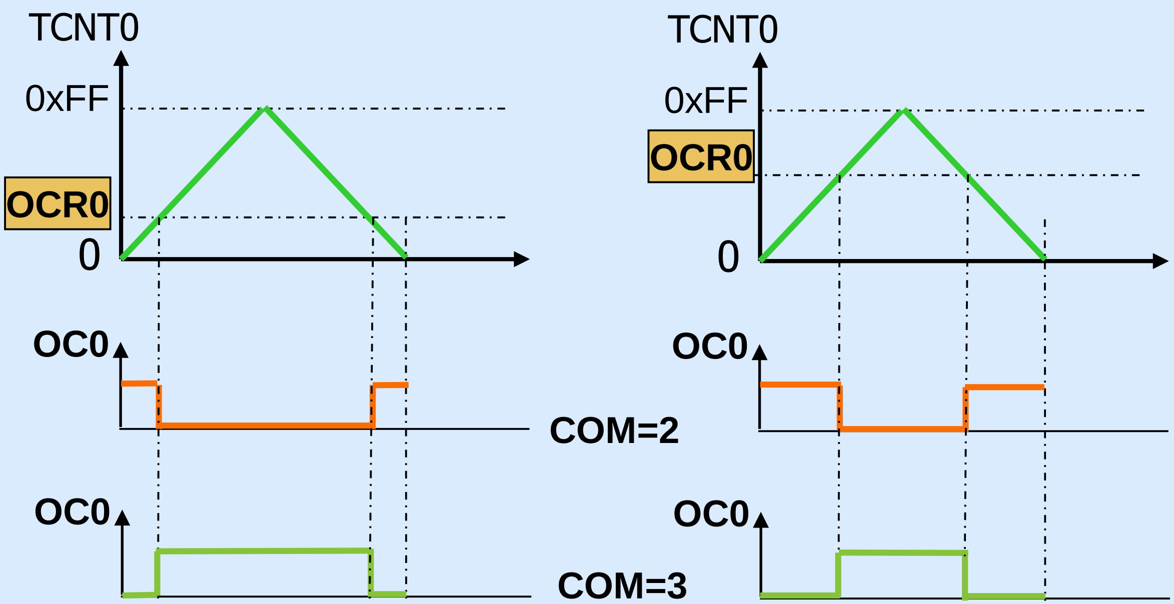
Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 73 for more details.

Phase Correct PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

Phase Correct PWM
Duty cycle = changeable (0% to 100%)
Frequency = selectable between limited choices



$$\text{Duty Cycle} = \frac{OCRx}{255} \times 100$$

$$\text{Duty Cycle} = \frac{255 - OCR0}{255} \times 100$$

$$F_{OC0} = \frac{f_{clk}}{N(510)}$$

LCD and Keyboard

The AVR microcontroller and embedded systems

using assembly and c

About LCD

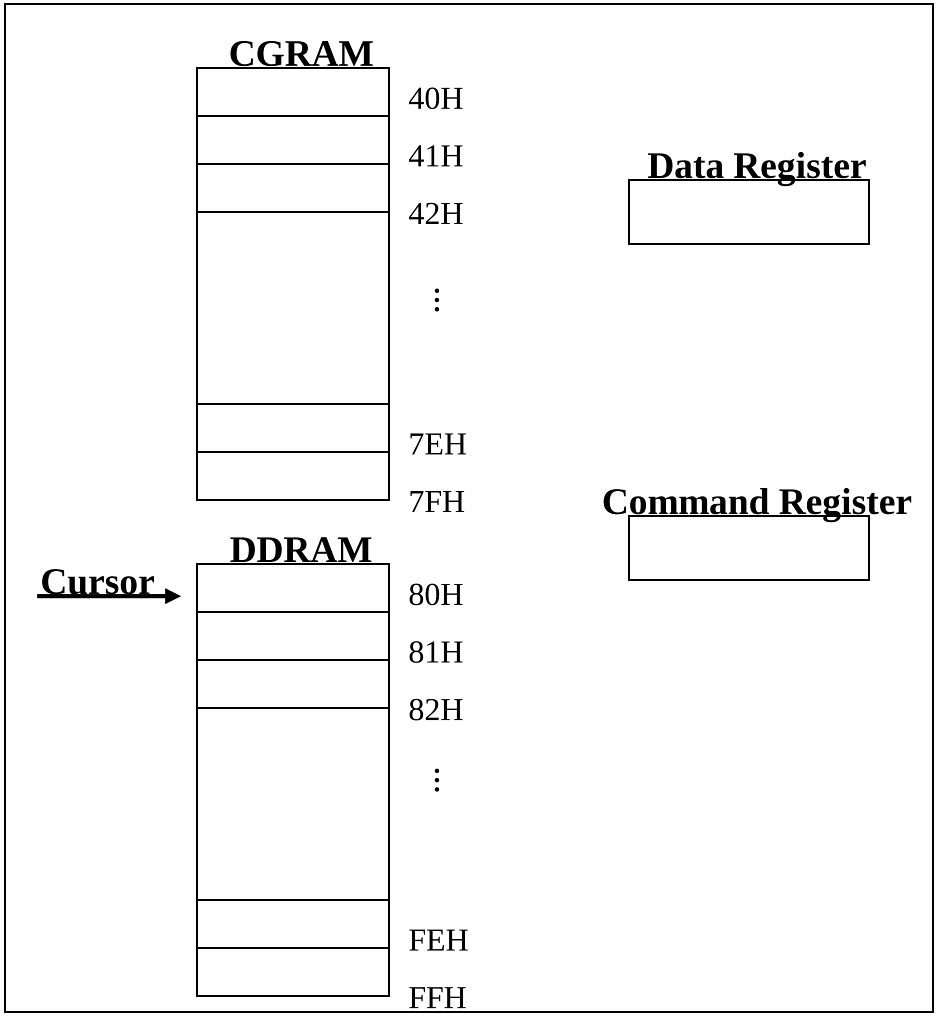
- Sometimes the embedded system needs to inform the user of something. There are different ways to inform the user, such as LEDs, 7segments and LCDs.
- LCD is one of the most powerful ways; as you can display different texts and icons on it.

Topics:

- LCD pin out
- LCD internal components
- How to use LCD
  - Busy
  - LCD commands
  - Changing fonts (case study)
- additional references

LCD internal components

- DDRAM (Data Display RAM)
- CGRAM (Character Generator RAM)
- Cursor (Address Counter)
- Data Register
- Command Register



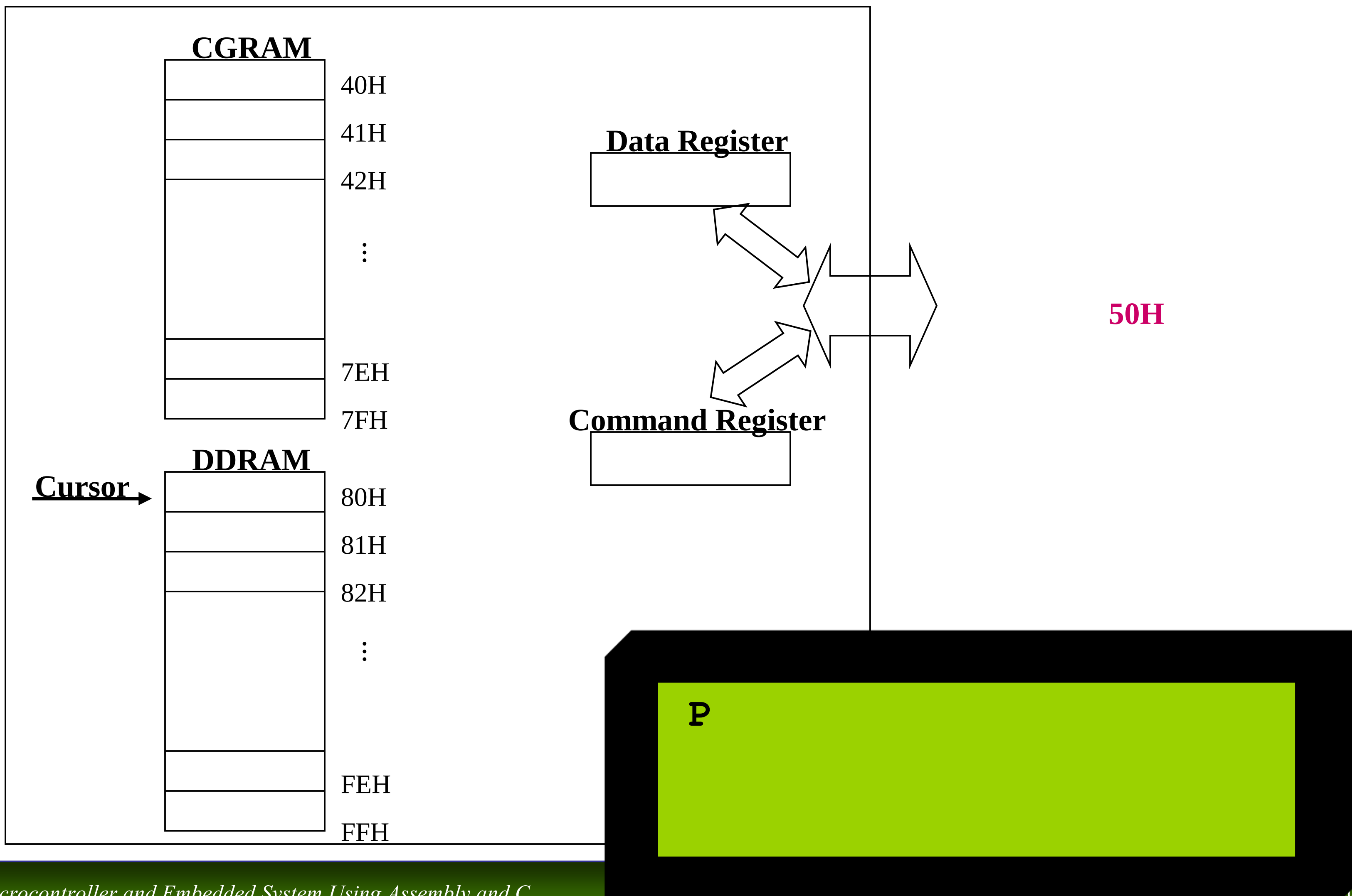
DDRAM (Data Display RAM)

- DDRAM (Data Display RAM)
  - It is a 128x8 RAM (128 bytes of RAM)
  - Contains the data that should be displayed on the LCD.
  - If we write the ASCII code of a character into the RAM the character will be displayed on the LCD.
- CGRAM (Character Generator RAM)
  - It is a 64x8 RAM (64 bytes of RAM).
  - The fonts of characters 00H to 07H are stored in the RAM.
  - We can change the fonts of the 8 characters by writing into the RAM.
- Cursor (Address Counter)
  - Cursor is a register which points to a location of DDRAM or CGRAM.

DDRAM (Data Display RAM)

- Data Register
  - It is an 8 bit register.
  - When we write a byte of data into the data register, the data will be written where the cursor points to.
  - For example, if we write a byte of data into the data register while the cursor points to location 80H of DDRAM, the contents of location 80H will be changed to the data, we have written into the data register.
- Command Register
  - We can command the LCD by writing into the command register.
  - For example, we can ask the LCD, to set cursor location, or clean the screen, by writing into the command Register.

Writing to Data Register (Example)



LCD commands

- We mentioned earlier that we can order the LCD by sending command codes to the command register.
- Some of the command codes are listed in the following table.

Code (Hex)	Instruction	Code (Hex)	Instruction
1	Clear display screen	2	Return home
10	Shift cursor position to left	14	Shift cursor position to right
18	Shift display left	1C	Shift display right
4	After displaying a character on the LCD, shift cursor to left	6	After displaying a character on the LCD, shift cursor to right
80-FF	Set cursor position	40-7F	Set CG RAM address
8	Display off, cursor off	A	Display off, cursor on
C	Display on, cursor off	E	Display on, cursor on
F	Display on, cursor blinking	38	Initializing to 2 lines & 5x7 font

Display and Cursor

- Display on cursor blinking (0FH)

- Display on cursor on (0EH)

- Display on cursor off (0CH)

- Display off cursor off (0AH)

Set cursor position (Set DDRAM address)

- We mentioned earlier that each location of the DDRAM, retains the character that should be displayed in a location of LCD.
- The following figures, represent that if you want to display a character in each of the rooms of the LCD, you should write into which location of the DDRAM. (The numbers are in hex.)
- To move the cursor to any location of the DDRAM, write the address of that location into the command register.

	1	2	3	…	18	19	20
Line 1	**80**	**81**	**82**	…	**91**	**92**	**93**

20x1 LCD

	1	2	3		18	19	20
Line 1	**80**	**81**	**82**	…	**91**	**92**	**93**
Line 2	**C0**	**C1**	**C2**	…	**D1**	**D2**	**D3**
Line 3	**94**	**95**	**96**	…	**A5**	**A6**	**A7**
Line 4	**D4**	**D5**	**D6**	…	**E5**	**E6**	**E7**

20x4 LCD

	1	2	3		18	19	20
Line 1	**80**	**81**	**82**	…	**91**	**92**	**93**
Line 2	**C0**	**C1**	**C2**	…	**D1**	**D2**	**D3**

20x2 LCD

	1	2	3	…	38	39	40
Line 1	**80**	**81**	**82**	…	**A5**	**A6**	**A7**
Line 2	**C0**	**C1**	**C2**	…	**E5**	**E6**	**E7**

40x2 LCD

	1	2	3		14	15	16
Line 1	**80**	**81**	**82**	…	**8D**	**8E**	**8F**
Line 2	**C0**	**C1**	**C2**	…	**CD**	**CE**	**CF**

16x2 LCD

Set cursor position (example)

- We want to display a character in line 4 column 1 of a 20x4 LCD. What should we write to the command register to move the cursor to?

Set cursor position (example)

- We want to display a character in line 4 column 1 of a 20x4 LCD. What should we write to the command register to move the cursor to?

Solution:

We should move cursor to address D4H of the DDRAM. So, we should write D4H, into the command register.

	1	2	3		18	19	20
Line 1	80	81	82	…	91	92	93
Line 2	C0	C1	C2	…	D1	D2	D3
Line 3	94	95	96	…	A5	A6	A7
Line 4	D4	D5	D6	…	E5	E6	E7

Decrease and increase Cursor

- If you write a byte of data into the data register, the data will be written where the cursor points to, and cursor will be incremented, by default.
  - If you want to make the LCD, to decrement the cursor, you should write 4H into the command register.
  - If you want to make the LCD, to reactivate the default (shift cursor to right) you should write 6H into the command register.

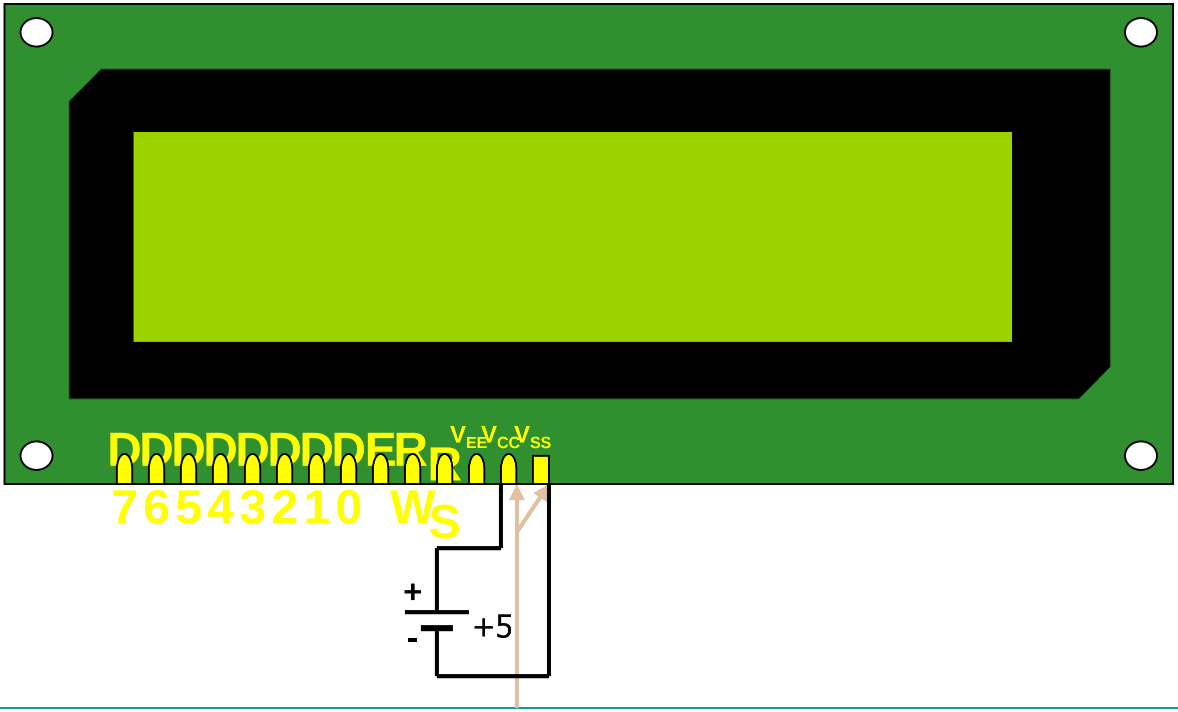
Increment cursor

Decrement cursor

LCD pins

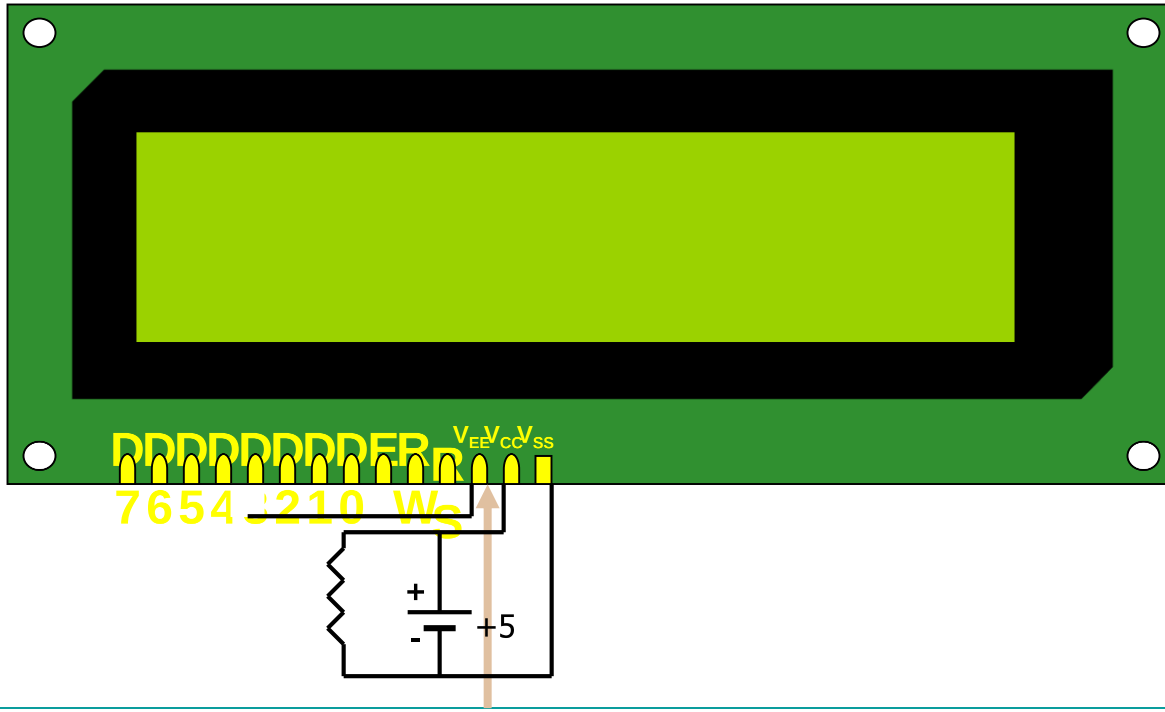
In this section, you learn the functionalities of the LCD pins.

LCD pins



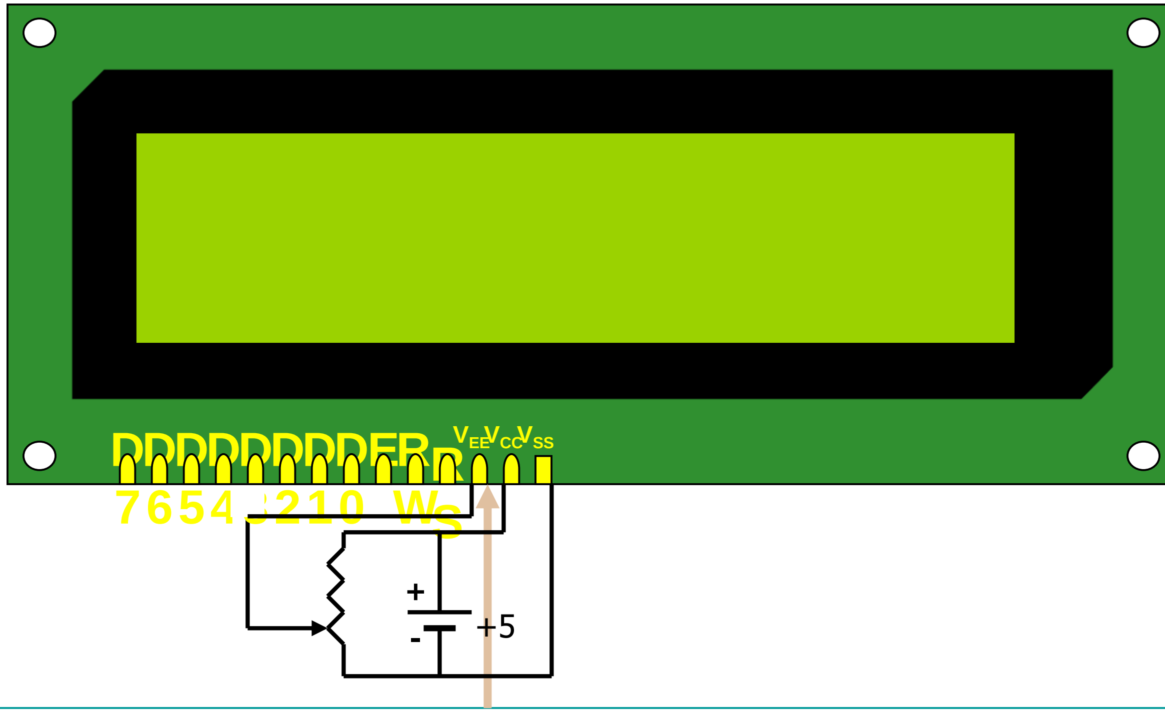
- V_{SS} and V_{CC}: These pins provide the energy to the LCD. We must connect them to $+5^{V}$.

LCD pins



- V_{EE}: We control the contrast of the LCD by giving a voltage between 0^V and $+5^V$ to the pin.

LCD pins



- V_{EE}: We control the contrast of the LCD by giving a voltage between 0^V and $+5^V$ to the pin.

LCD pins

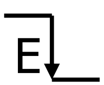
- D0 to D7: LCD sends and receives data, through the 8 pins.

LCD pins

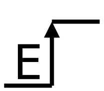
- R/W (Read/Write):
  - When we want to send (write) data to the LCD, we make the pin, low.
  - When we want to receive (read) data from the LCD, we set the pin to high.

LCD pins

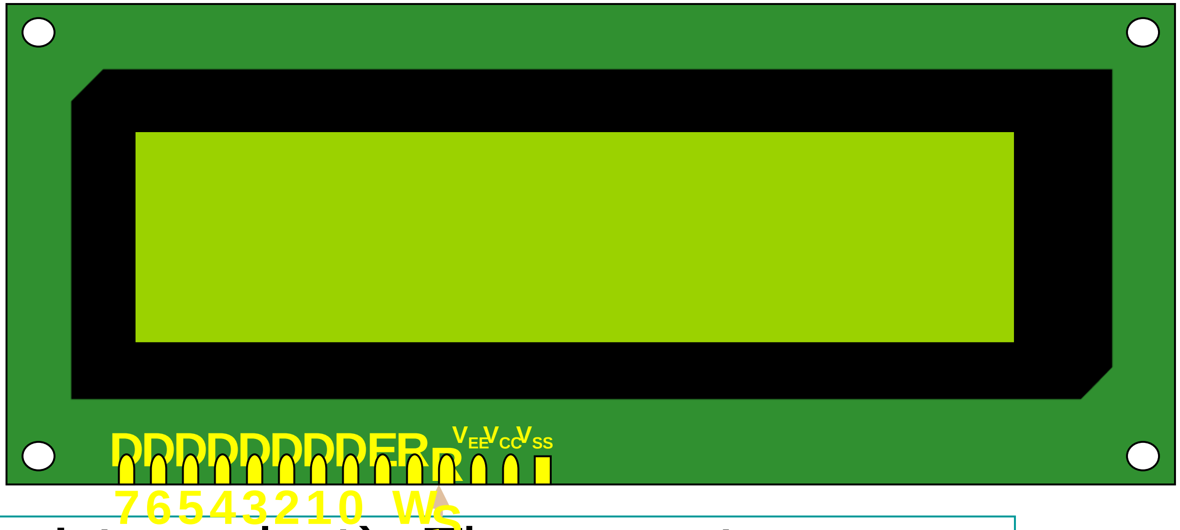
- E (Enable): We activate the pin when we want to send or receive data from the LCD.
  - When we want to send data to the LCD, we make the RW pin, low; and supply the data to data pins (D0 to D7); and then apply a high to low pulse to the **E**nable pin.



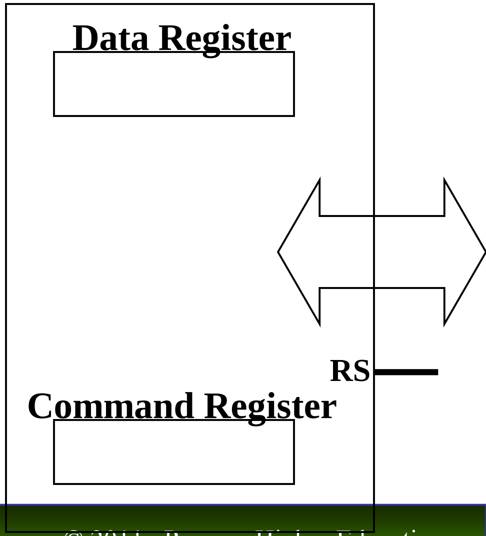
  - When we want to receive data from the LCD, we make the RW pin, high; and then apply a low to high pulse to the **E**nable pin. LCD supplies data to the data pins (D0 to D7).



LCD pins

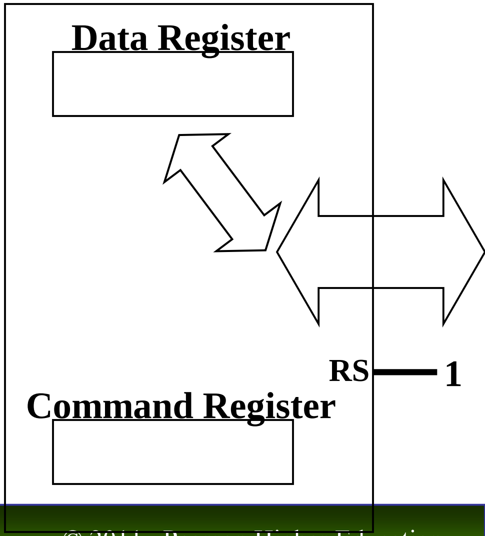


- RS (Register select): There are two registers with names of command register and data register in the LCD.
- If RS = 1, whenever we send data to the LCD, the data will be located in the data register.
- If RS = 0, whenever we send data to the LCD, the data will be located in the command register

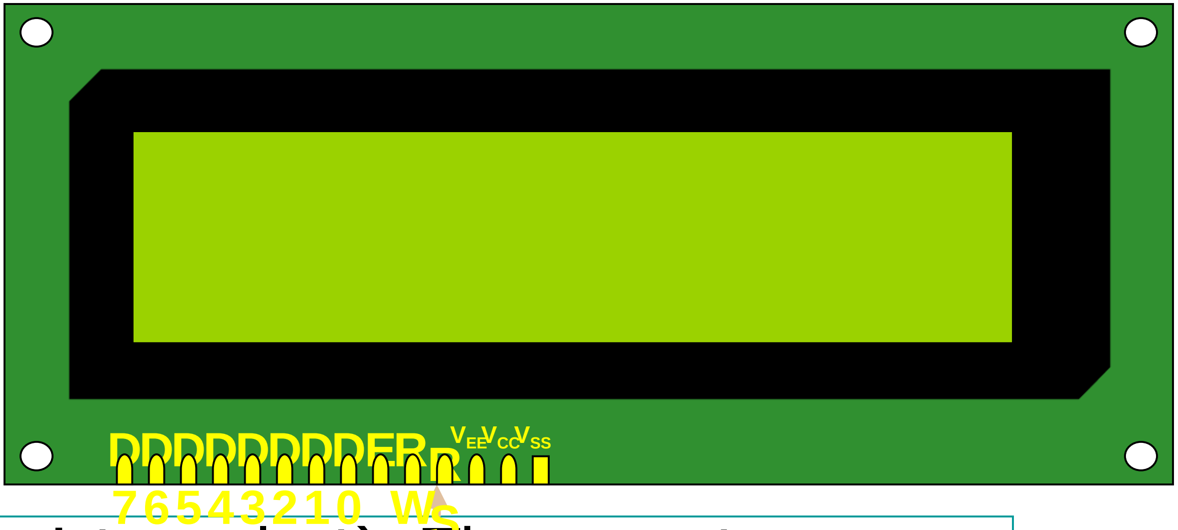


LCD pins

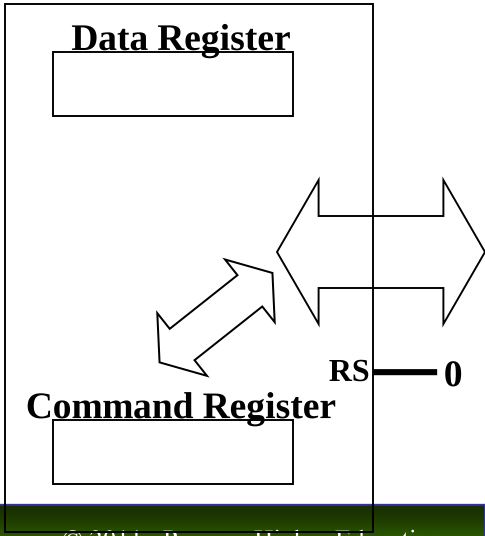
- RS (Register select): There are two registers with names of command register and data register in the LCD.
- If RS = 1, whenever we send data to the LCD, the data will be located in the data register.
- If RS = 0, whenever we send data to the LCD, the data will be located in the command register



LCD pins



- RS (Register select): There are two registers with names of command register and data register in the LCD.
- If RS = 1, whenever we send data to the LCD, the data will be located in the data register.
- If RS = 0, whenever we send data to the LCD, the data will be located in the command register

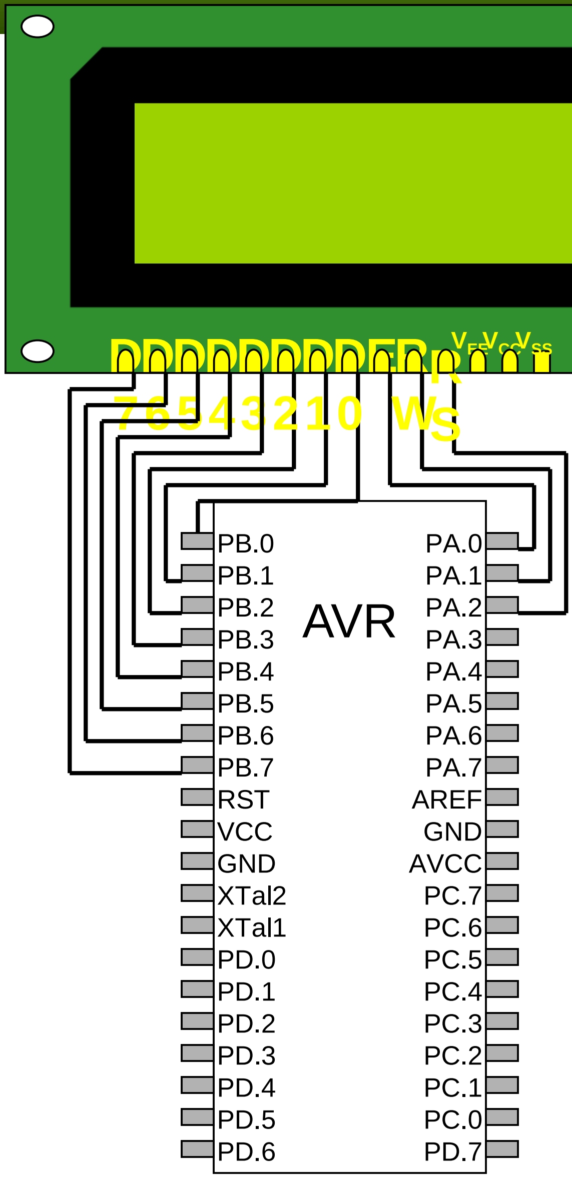


LCD Programming

- Initialization
  - We must initialize the LCD before we use it.
  - To initialize an LCD, for 5×7 matrix and 8-bit operation, 0x38, 0x0E, and 0x01 are send to the command register.
- Sending commands to the LCD
  - Make pins RS and R/W = 0
  - Put the command number on the data pins (D0–D7)
  - Send a high-to-low pulse to the E pin to enable the internal latch of the LCD (wait about 100us after each command)
- Sending data to the LCD
  - make pins RS = 1 and R/W = 0.
  - put the data on the data pins (D0–D7)
  - send a high-to-low pulse to the E pin (wait about 100us)

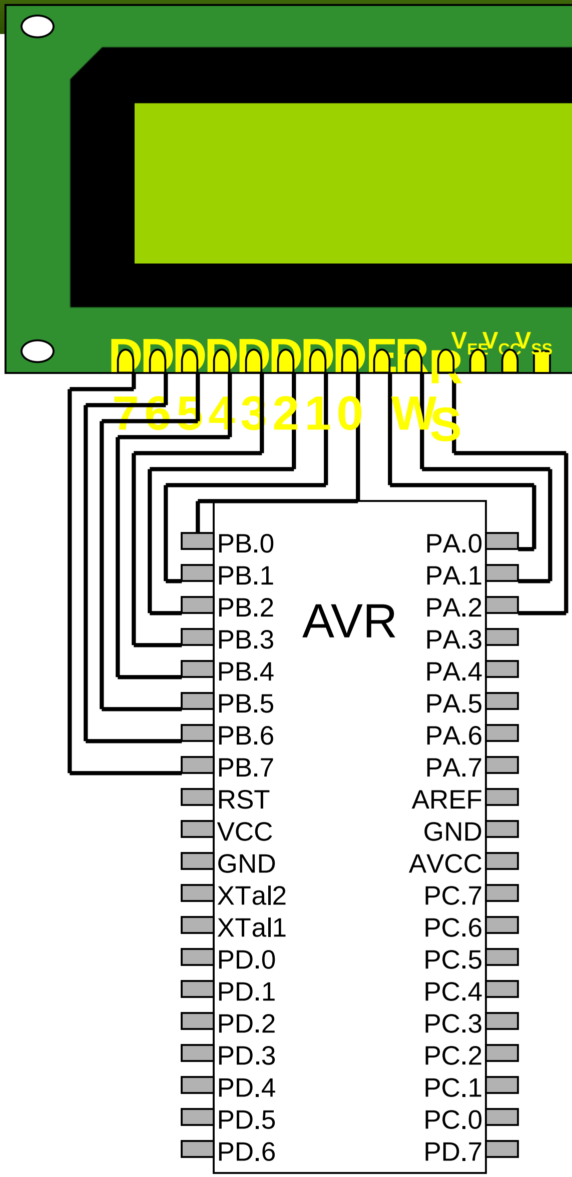
5x10 Font

An example



- Write a program that displays ‘H’ on the LCD, where the cursor is located.

An example



- Write a program that displays 'H' on the LCD, where the cursor is located.

Solution:

```
PORTA |= (1<<2); // (RS = 1) as we want to write to the data register
PORTA&= (1<<1); // (RW = 0) as we want to send data to the LCD.
PORTB = 'H'; // as we want to send 'H' to the LCD.
//To make a High to low pulse on the Enable pin :
PORTA |= 1;
DELAY_ms (1);
PORTA.0 &= ~1;
```

Changing fonts (Changing CGRAM)

- Each character LCD has a CGRAM (Character generator RAM). It stores the fonts of the first 8 characters (character 0H to character 7H). So, you can change the font of the 8 characters and define new characters, by writing into the CGRAM. Each byte of the CGRAM stores a row of a font. The fonts are stored respectively, in the CGRAM. For example, if you change the content of first byte of the CGRAM (whose address is 40H), you have changed the highest row of character 0H.
- Attention: in an LCD with 5x7 font, each font has actually 8 rows. The 8th row is put aside for the cursor. You would better not set the bits of the 8th row.

	D7	D6	D5	D4	D3	D2	D1	D0	
Character 0	0	0	0	1	1	1	1	0	40
	0	0	0	1	0	0	0	1	41
	0	0	0	1	0	0	0	1	42
	0	0	0	1	1	1	1	0	43
	0	0	0	1	0	1	0	0	44
	0	0	0	1	0	0	1	0	45
	0	0	0	1	0	0	0	1	46
	0	0	0	0	0	0	0	0	47
Character 1	0	0	0	1	0	0	0	1	48
	0	0	0	0	1	0	1	0	49
	0	0	0	1	1	1	1	1	4A
	0	0	0	0	0	1	0	0	4B
	0	0	0	1	1	1	1	1	4C
	0	0	0	0	0	1	0	0	4D
	0	0	0	0	0	1	0	0	4E
	0	0	0	0	0	0	0	0	4F
									:
									:

CGRAM (Its first 16 bytes)

Changing fonts

- To change a row of a font, you should follow the following direction:
  1. Set the cursor position to point to the location of the CGRAM that you want to change.
  2. Change the font of the selected row, by writing into data register.
- Attention: LCD has only one cursor. When you want to change the CGRAM you make it point to CGRAM and when you want to display something on the screen you make it point to a location of DDRAM. So, when you finished changing the fonts don't forget to set the cursor position, so that, it points to DDRAM.

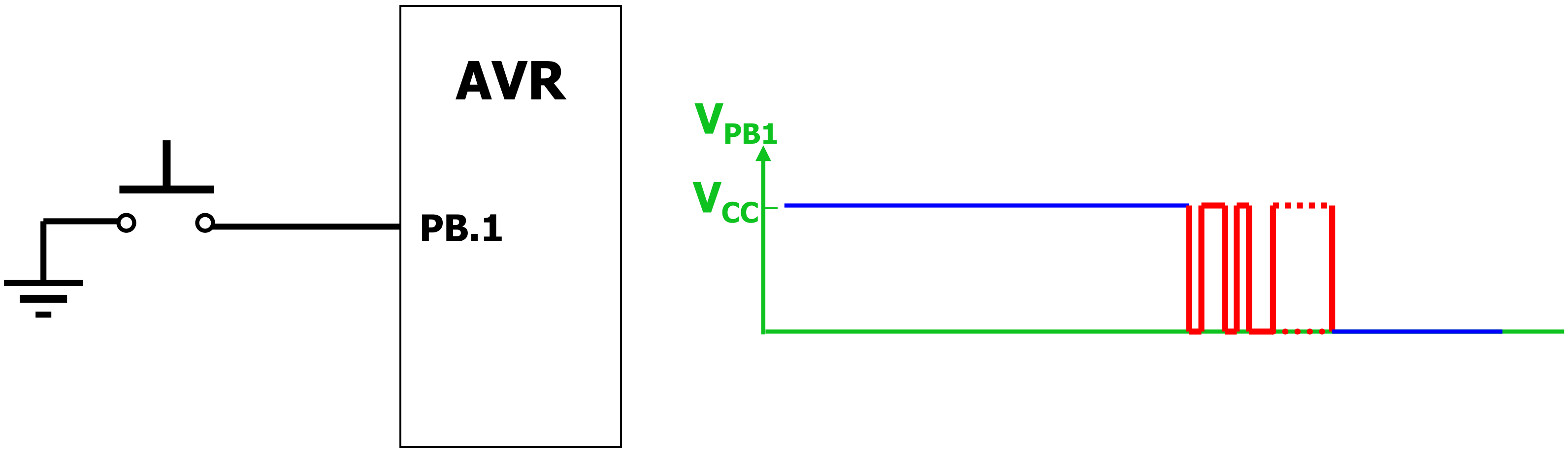
additional references

- You can find useful datasheets and user manuals about different LCDs in http://www.optrex.com/

Keyboard

Bounce

A key press may be considered as more than one click

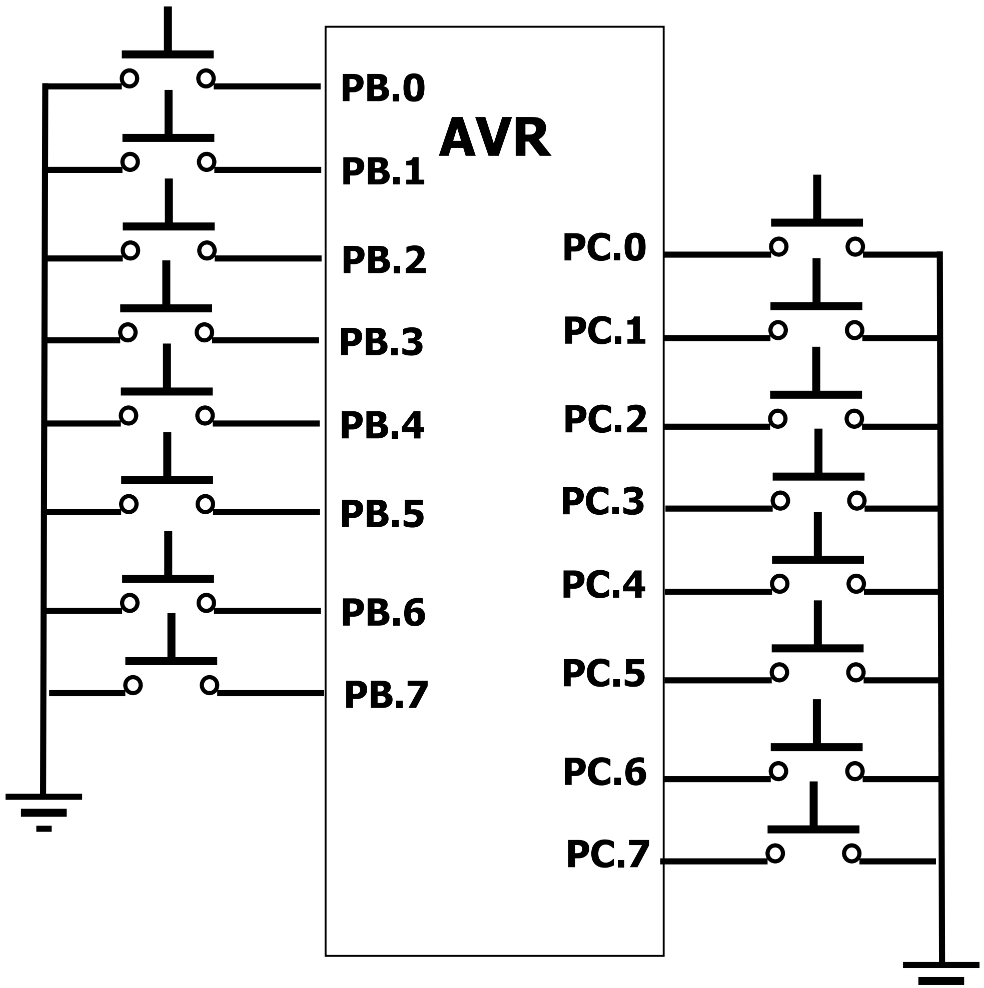


Debouncing (The correct way of reading keys)

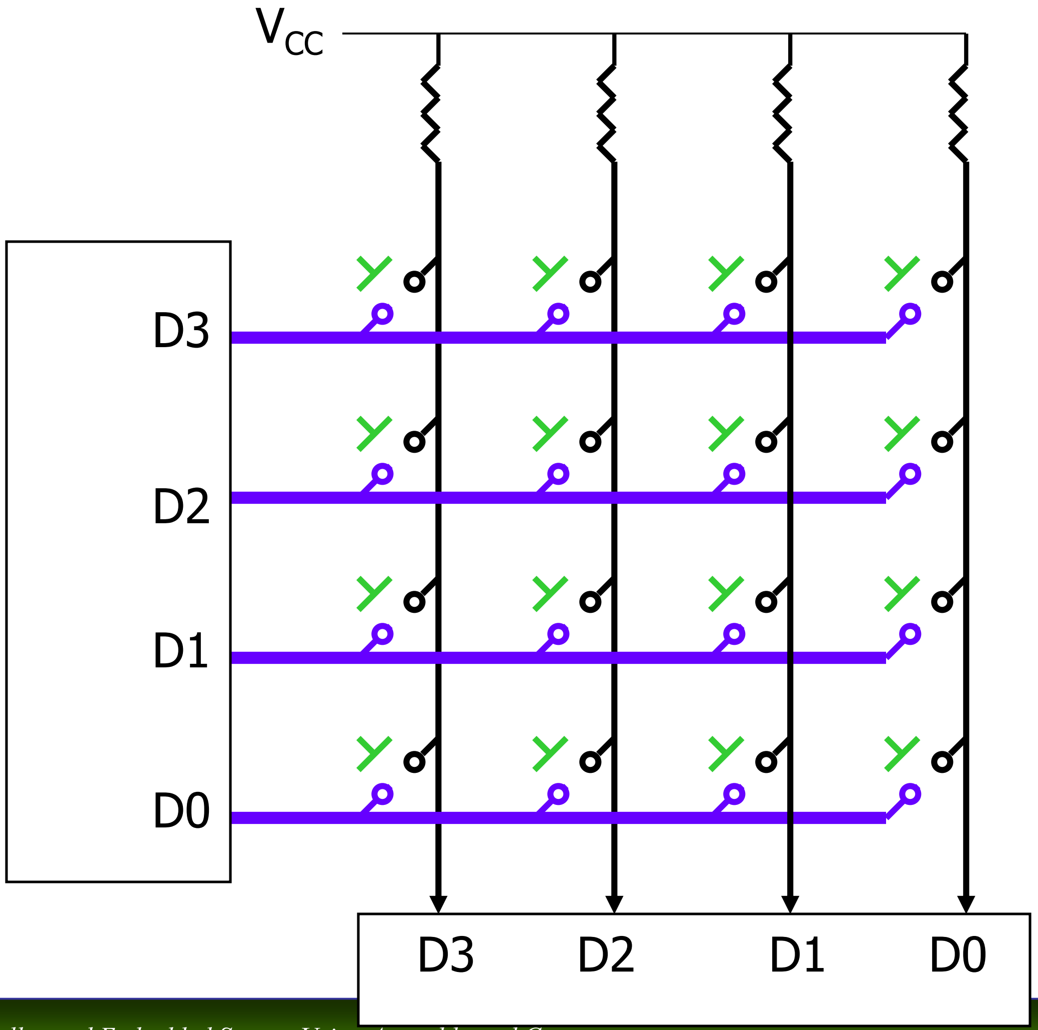
```
do{
        while((PORTB&1) == 0);
        delay_ms (20);
}while((PORTB&1) == 0);
do{
        while((PORTB&1) == 1);
        delay_ms (20);
}while((PORTB&1) == 1);
A++;
```

Using Keyboard

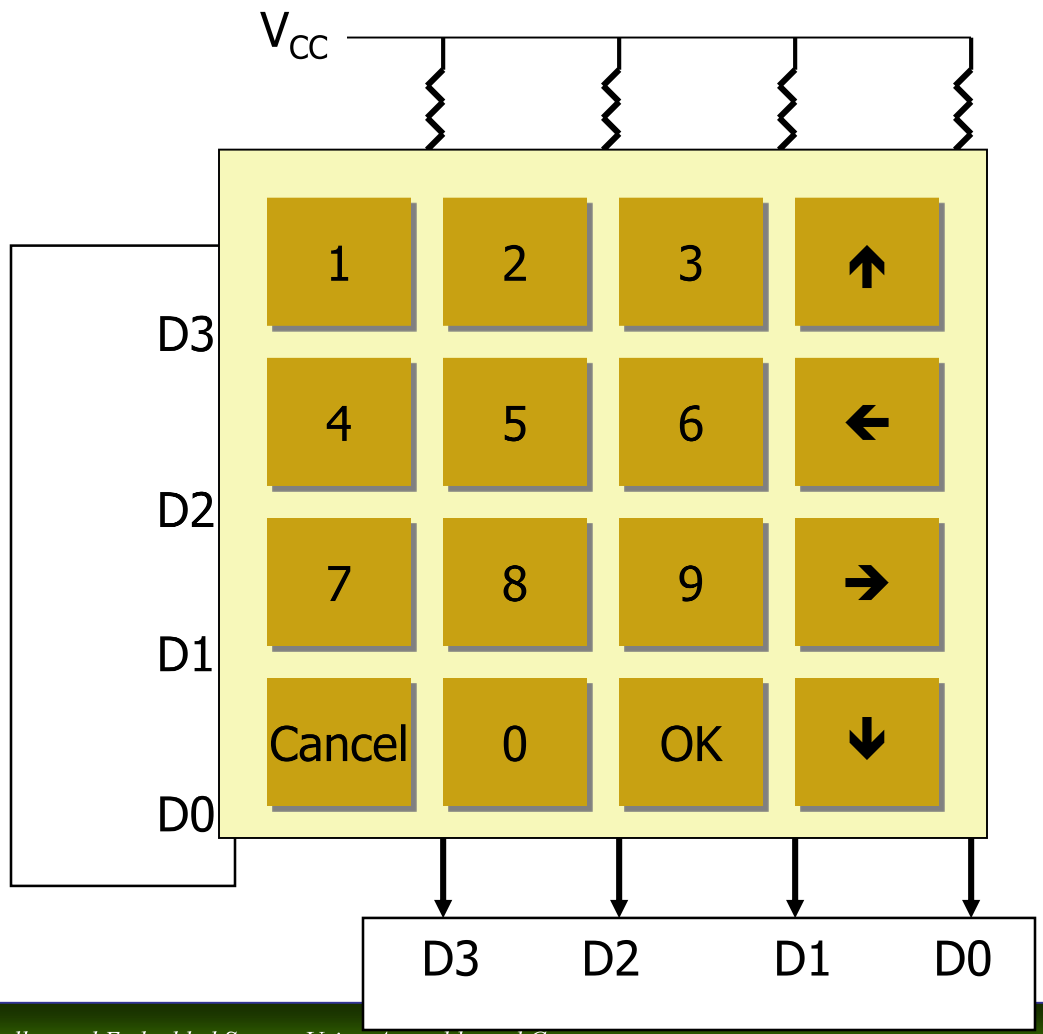
If we connect each key to a pin of the AVR, we waist many pins. So we use scanning as shown in the next slide



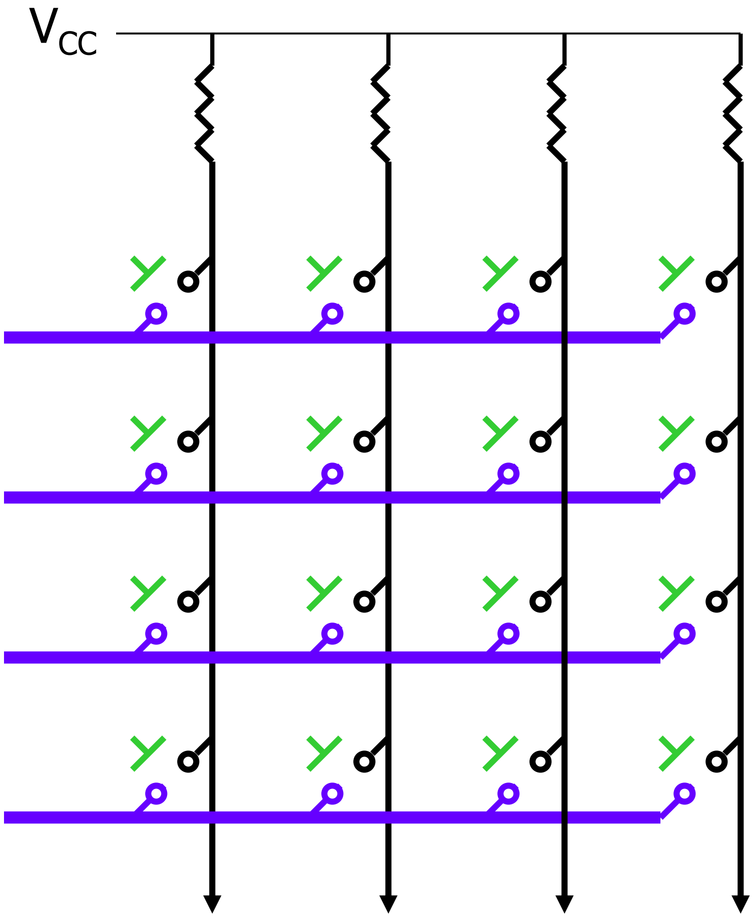
Keyboard



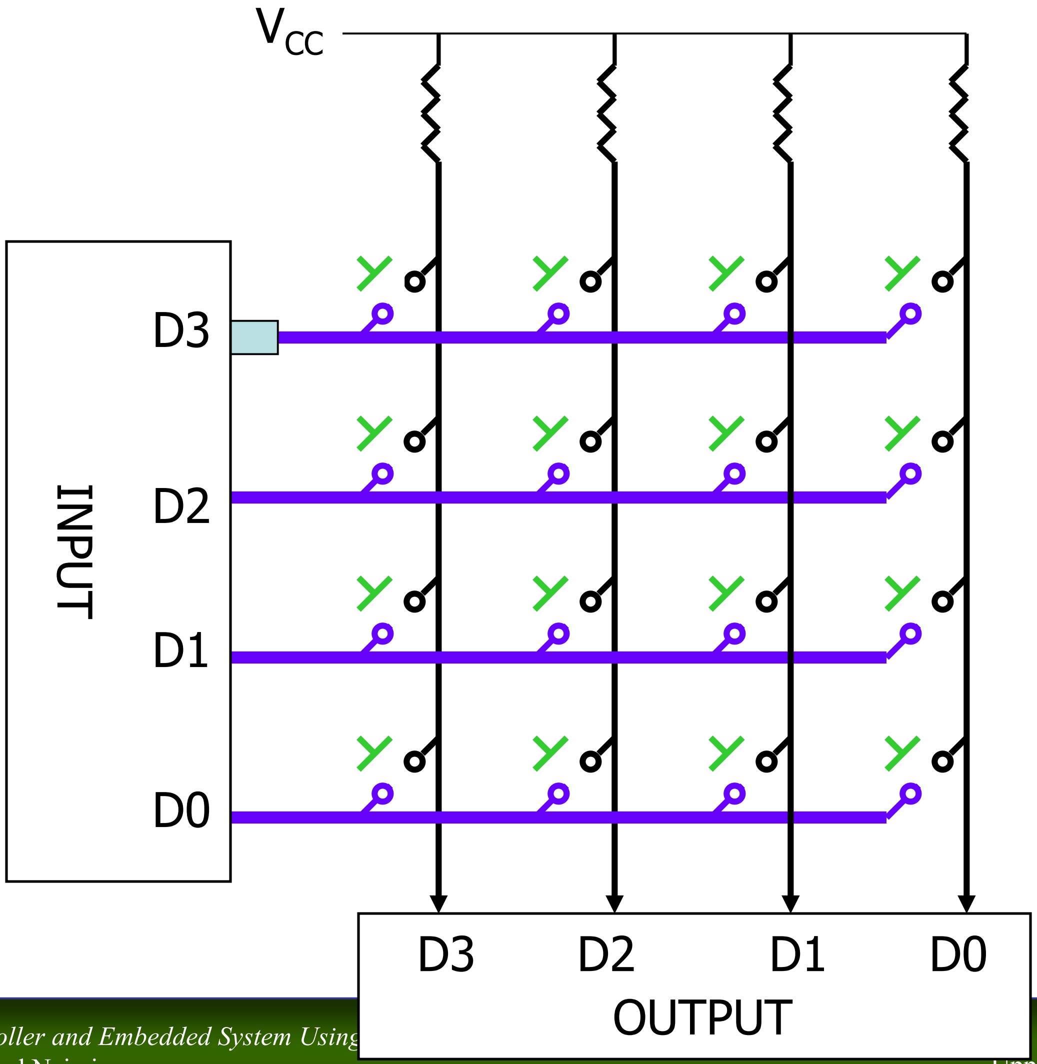
Keyboard



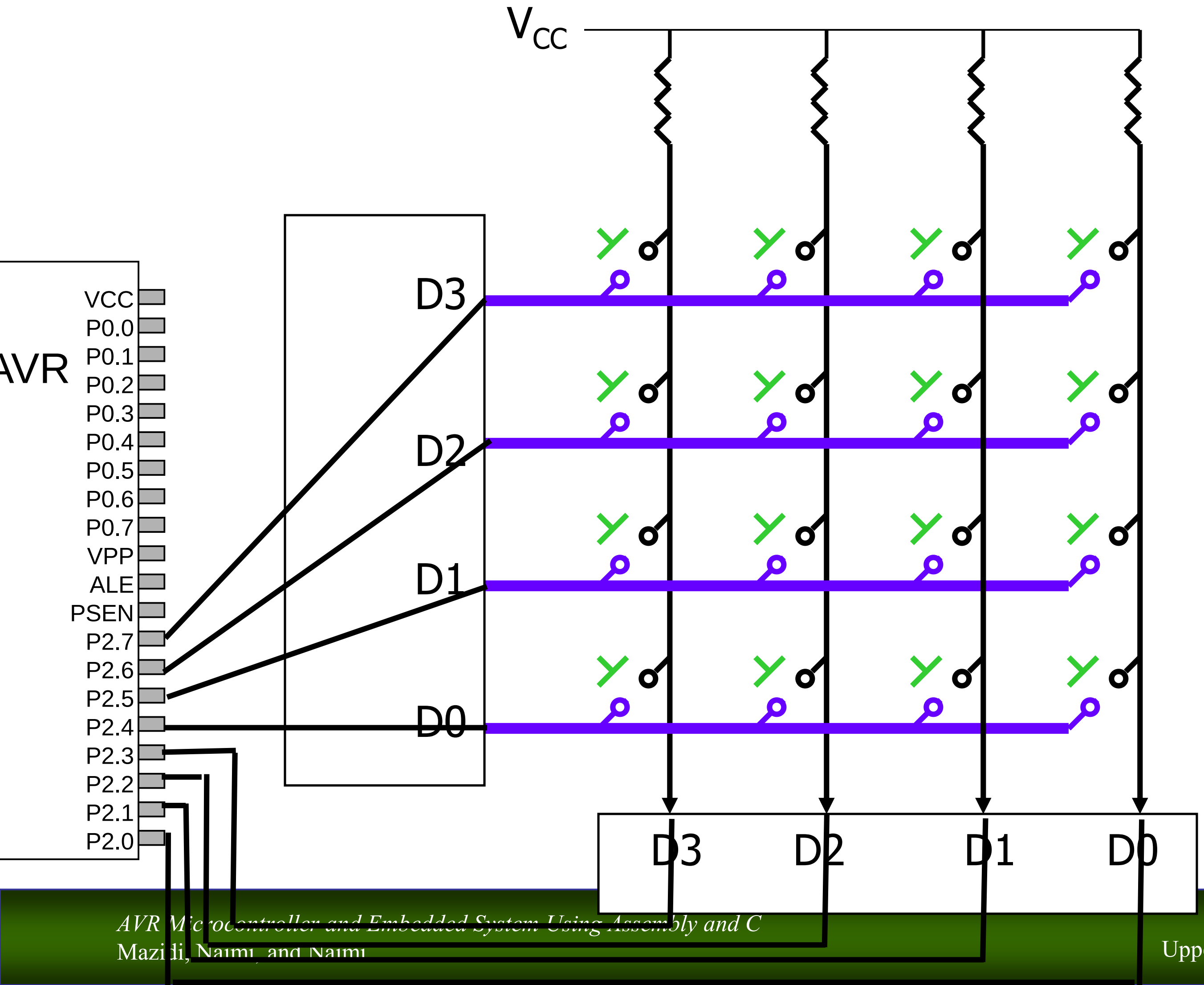
Creating a Matrix keyboard



Creating a Matrix keyboard



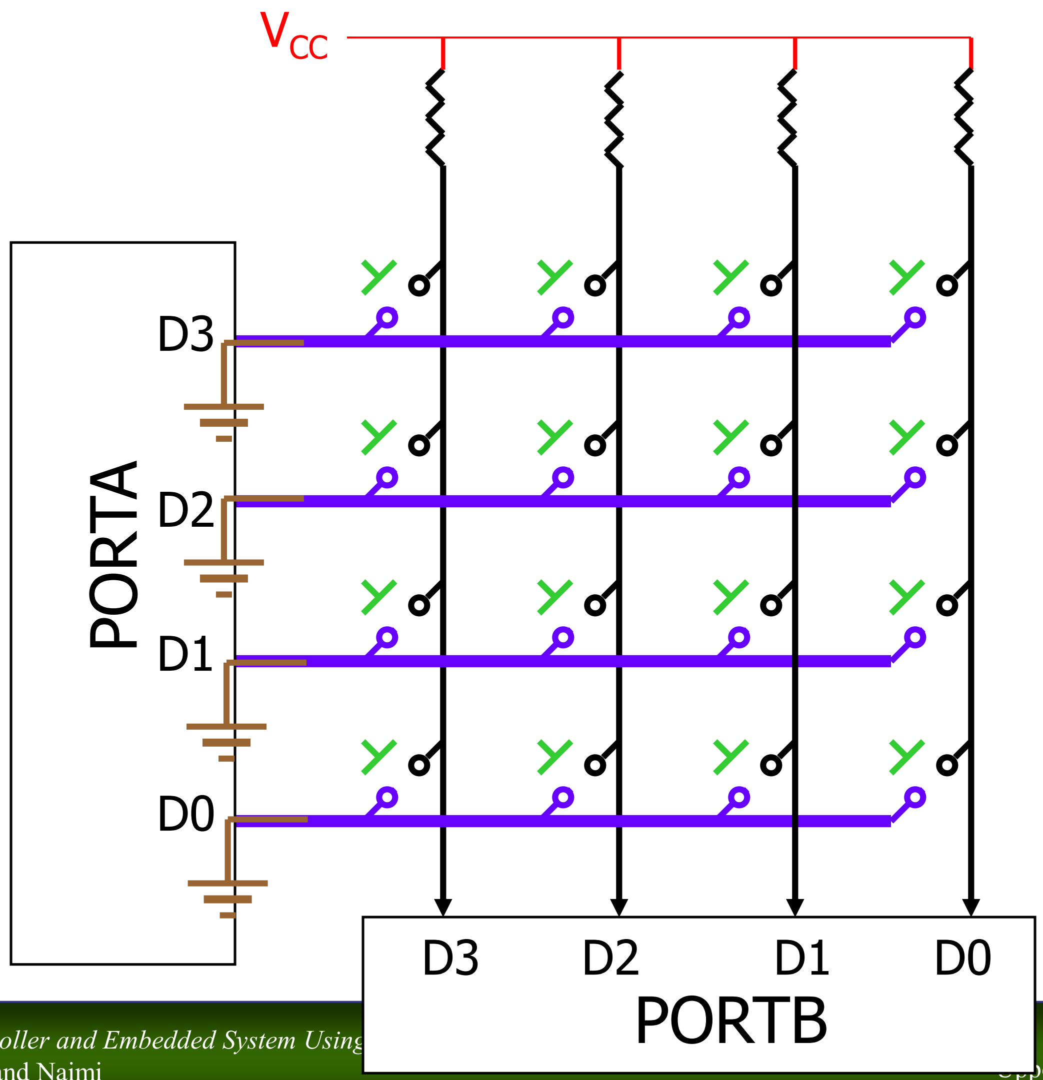
Connecting to AVR



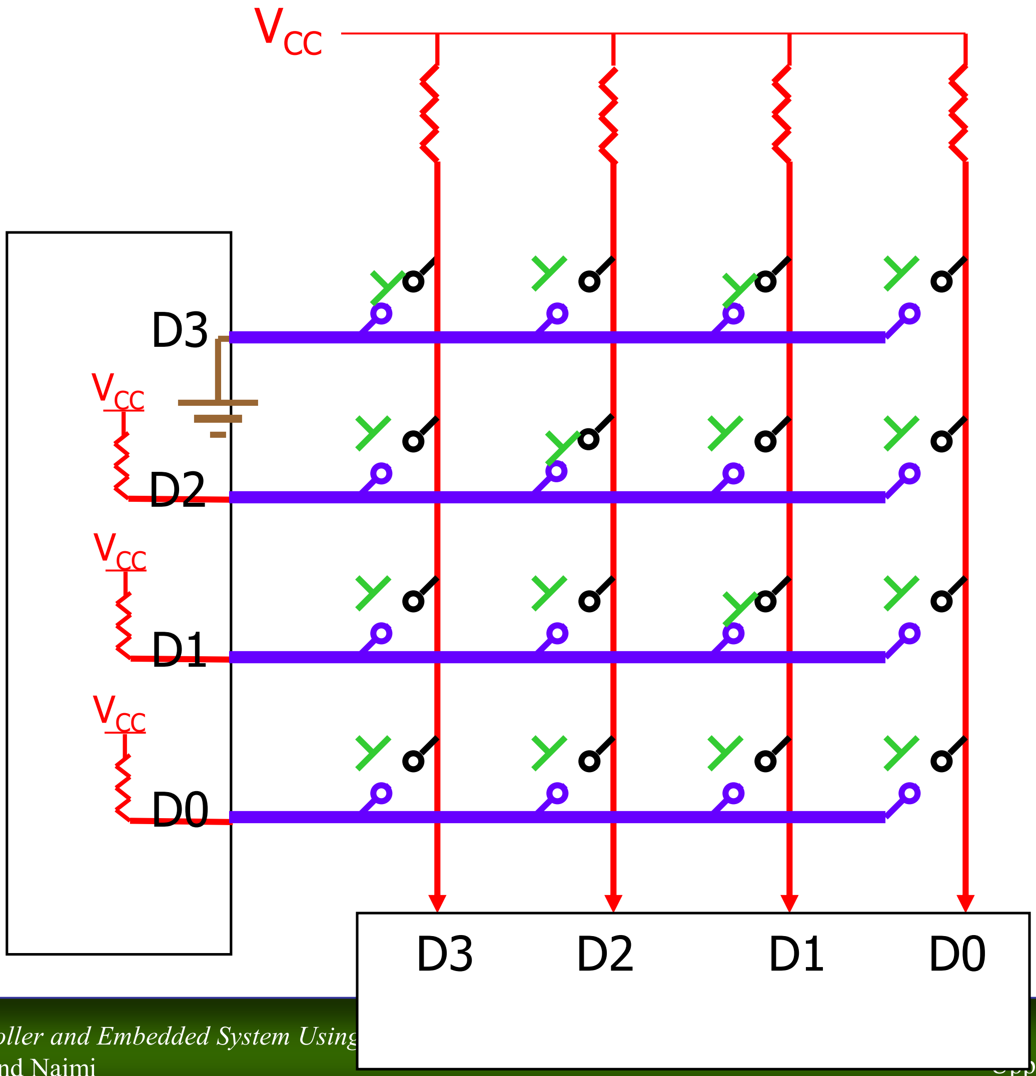
Keyboard Programming

- Writing programs for Matrix Keyboard
  - Key press detection
    - Aim: detecting if any of the keys is pressed
  - Key identification (scanning the keyboard)
    - Aim: identifying that which of the keys is pressed

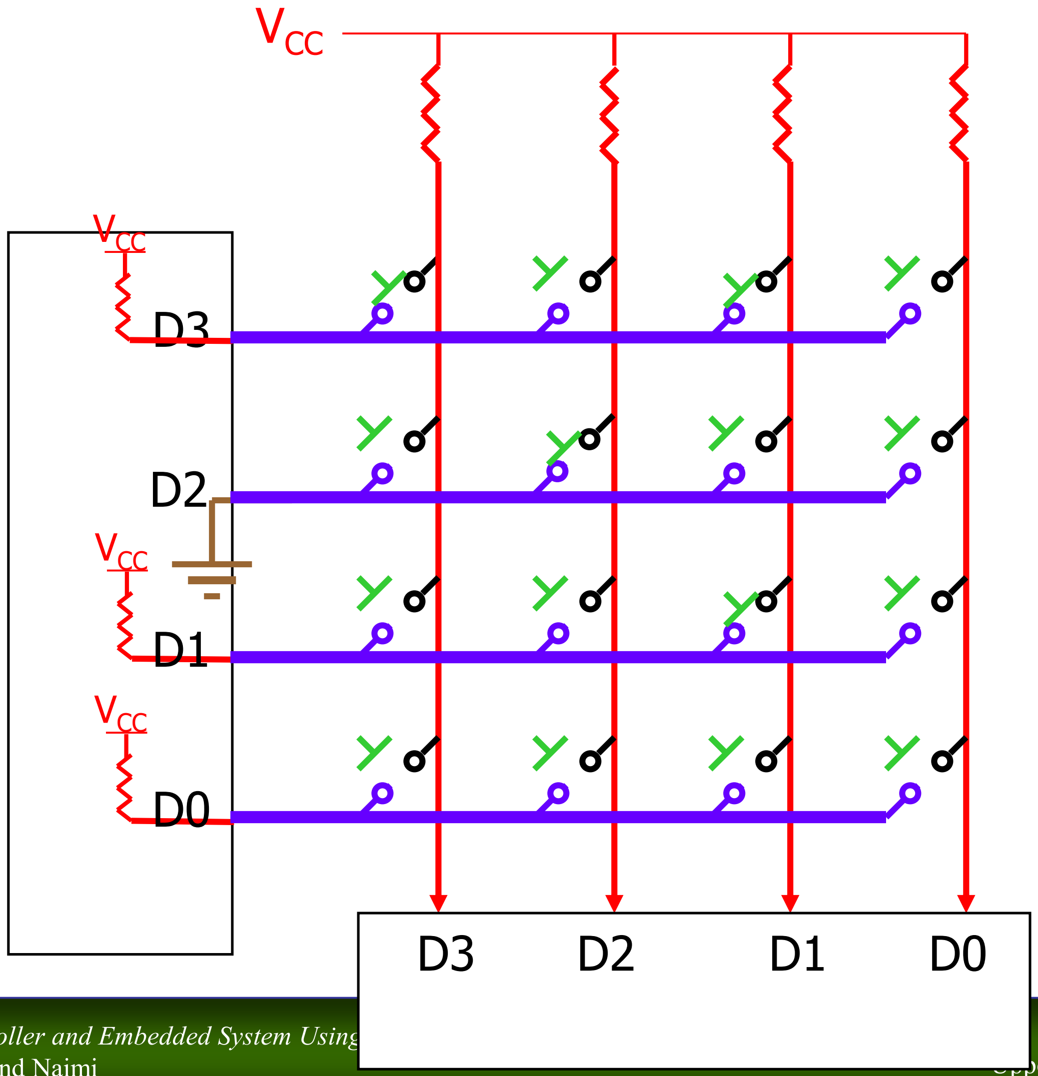
Press detection (is any of the keys pressed)



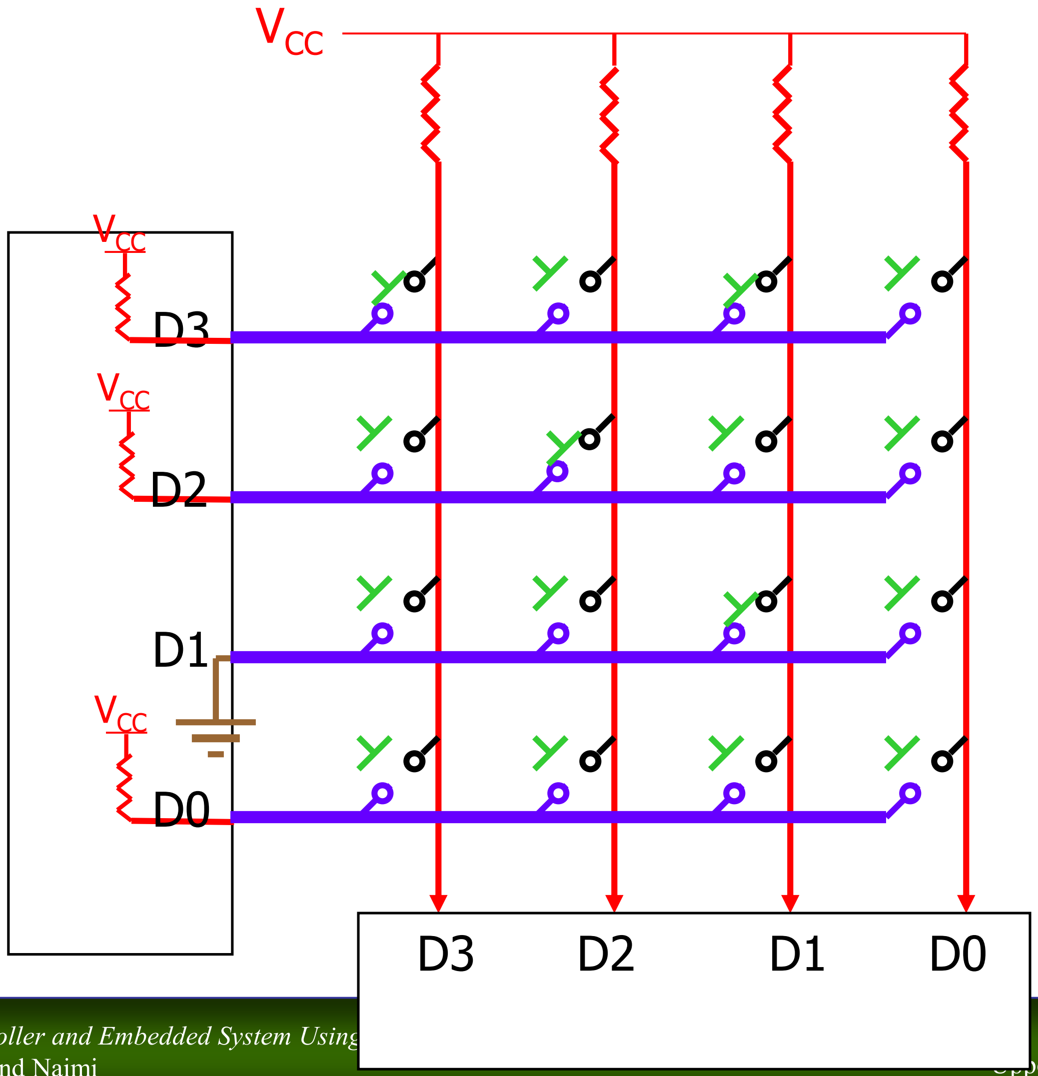
Key identification



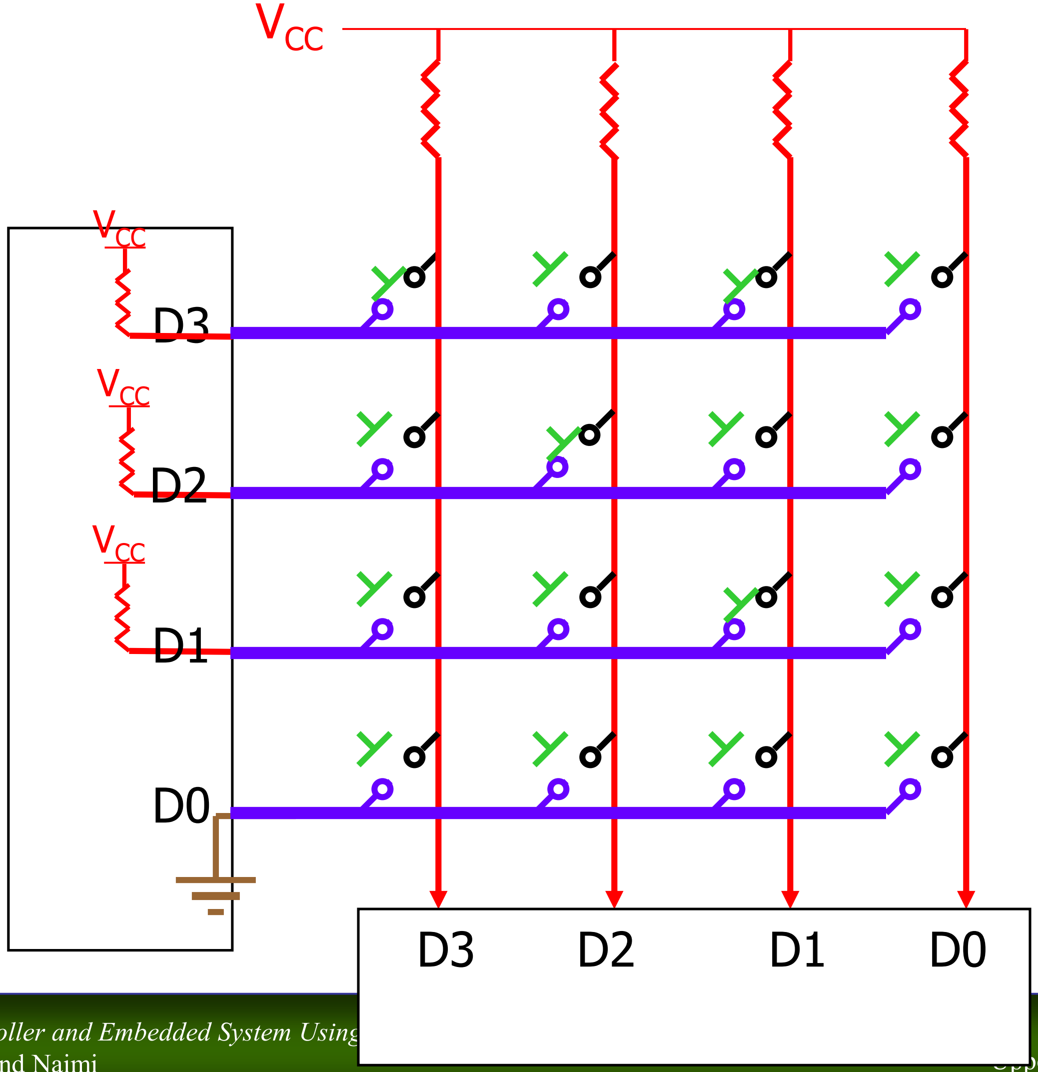
Key identification



Key identification

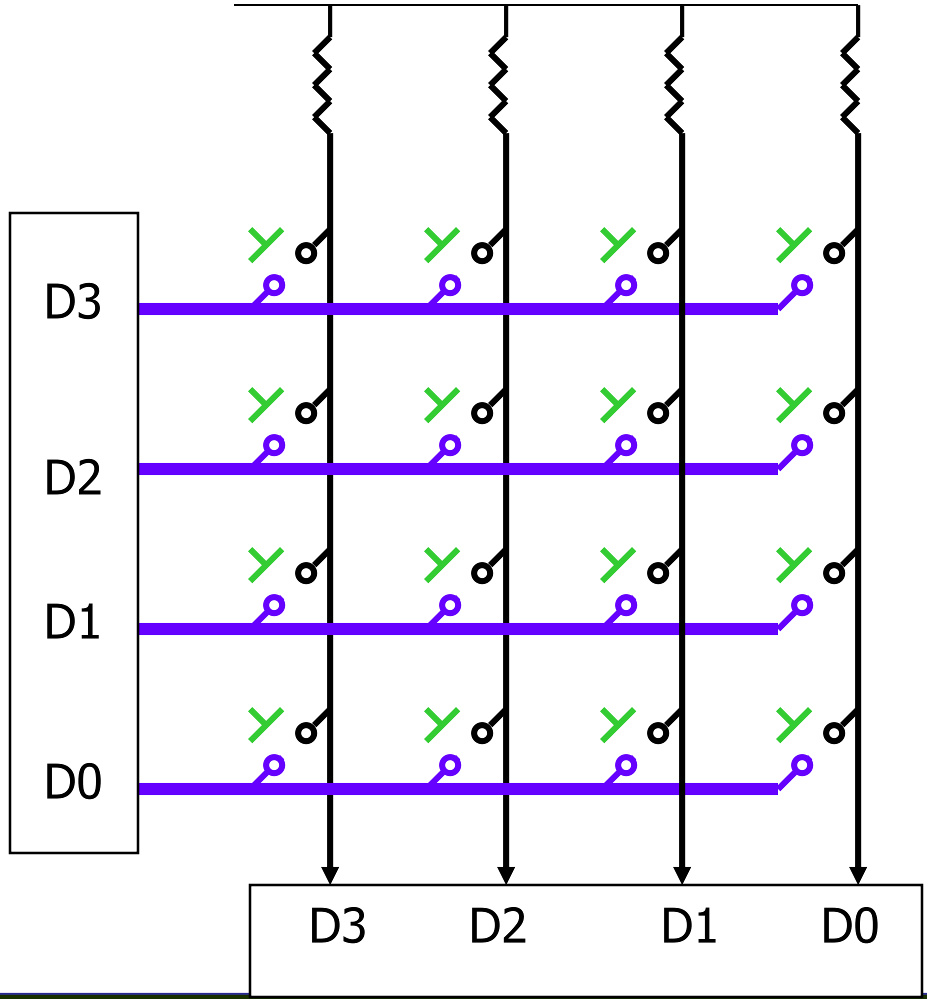


Key identification



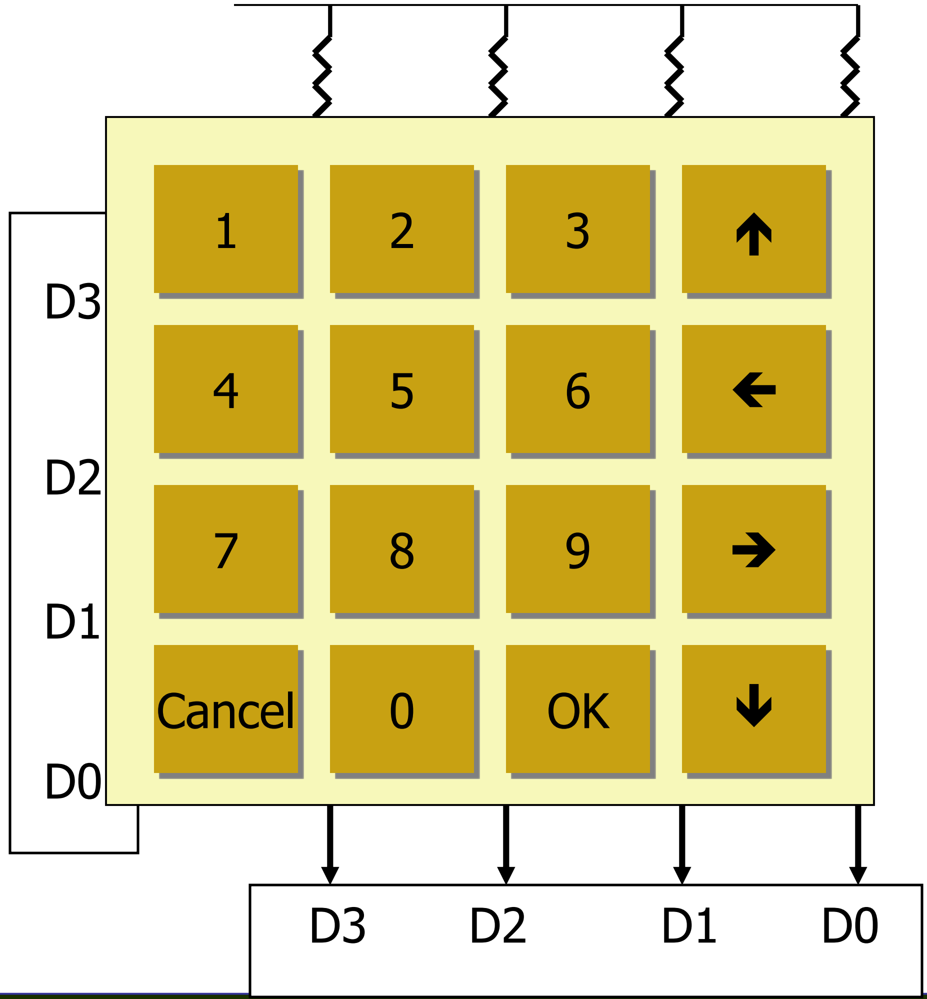
Example

- Write a function, that waits for a key to be pressed and then returns the code of the pressed key.

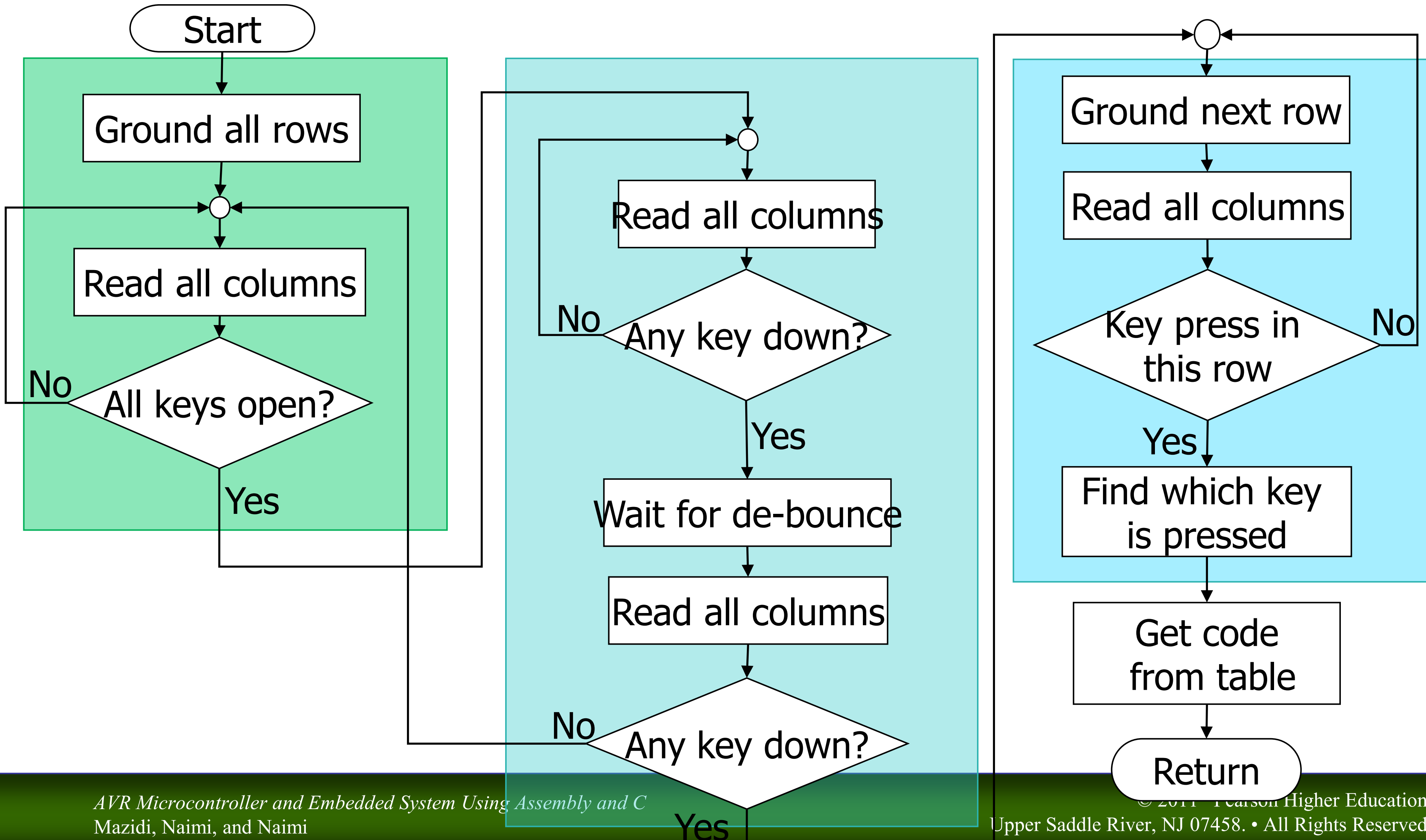


Example

- Write a function, that waits for a key to be pressed and then returns the code of the pressed key.



Solution


Start
Ground all rows
Read all columns
No
All keys open?
Yes
Read all columns
No
Any key down?
Yes
Wait for de-bounce
Read all columns
No
Any key down?
Yes
Ground next row
Read all columns
Key press in this row
No
Yes
Find which key is pressed
Get code from table
Return