AVR (ATmega32) Peripherals
(from here for session 26)

Amir Mahdi Hosseini Monazzah

Room 332,
School of Computer Engineering,
Iran University of Science and Technology,
Tehran, Iran.
monazzah@iust.ac.ir

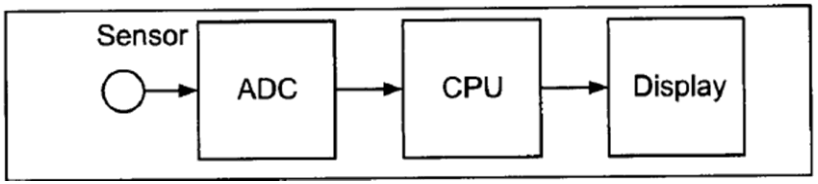
Fall 2020

Overview

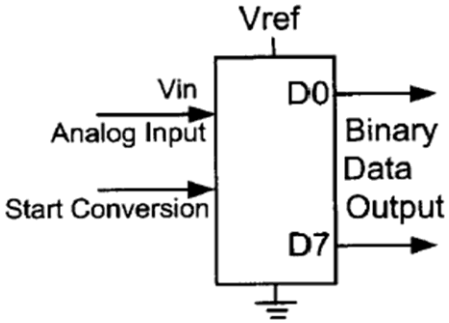
- Analog to digital (ADC)
- Ports
- Interrupts
- Timers
- Counters
- Serial ports
  - USART
- Other peripherals

Analog to Digital Conversion (ADC) in AVR

- Typical usage of ADC (analog to digital convertors)



- A typical ADC
- Vref= the maximum allowable voltage



Analog to Digital Conversion (ADC) in AVR

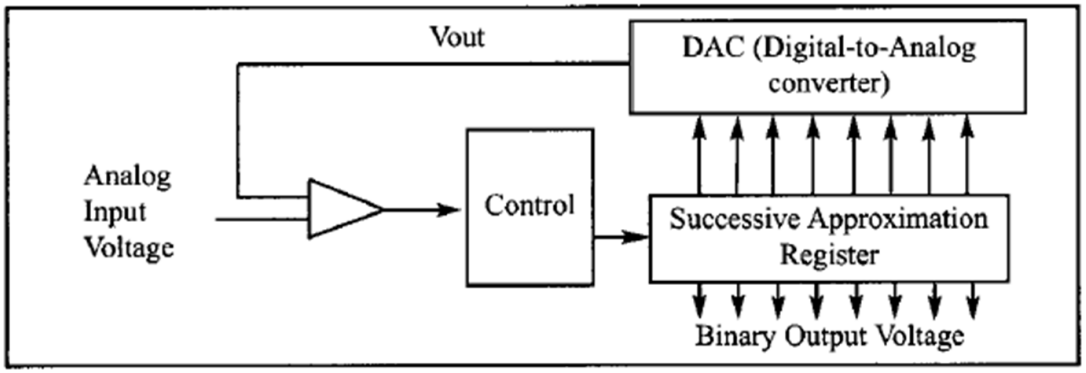
- Step size: the difference between two consecutive output numbers
- For a 8-bit output and Vref=5v,
  - step size= 5/256

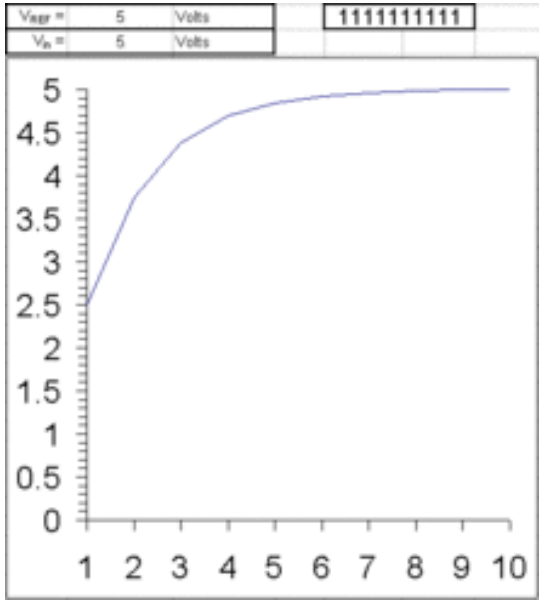
Table 13-1: Resolution versus Step Size for ADC (V_{ref} = 5 V)

n-bit	Number of steps	Step size (mV)
8	256	5/256 = 19.53
10	1024	5/1024 = 4.88
12	4096	5/4096 = 1.2
16	65,536	5/65,536 = 0.076

Analog to Digital Conversion (ADC) in AVR

- ADC internal structure

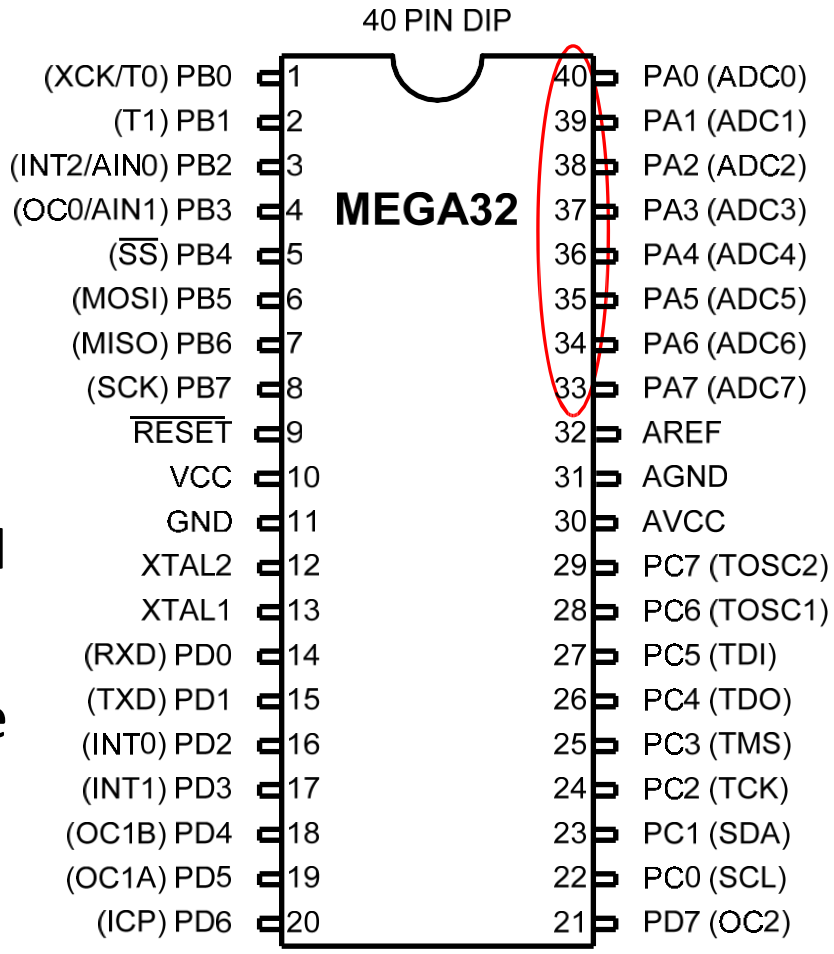




  - Successive approximation method
    - Search the web for algorithm details
  - n steps for an n-bit output ADC
  - Requires n cycles to calculate digital output

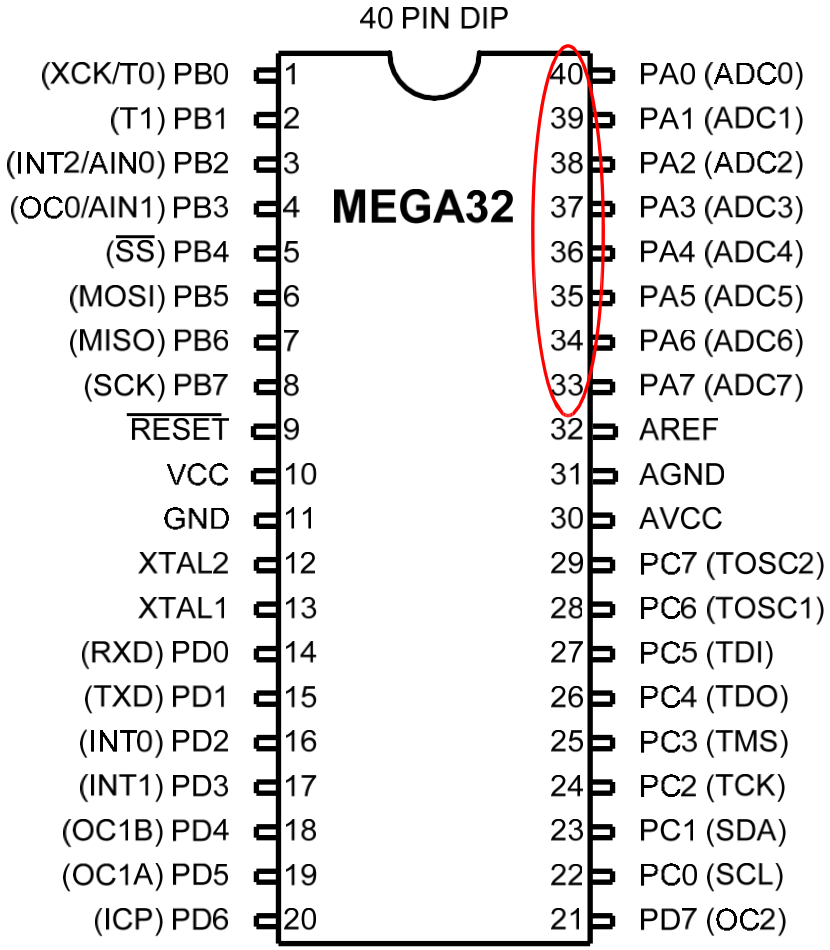
ADC in ATmega32

- 8 analog input channels
  - Each ADC multiplexed by an input channel (Port A)
- 10-bit output
  - Kept in ADCL and ADCH registers
  - 6 unused bits, can be set the upper or lower 6 bits



ADC in ATmega32

- AVR ADC Registers
  - ADCL and ADCH to keep digital data
  - ADCSRA to control ADCs
  - ADMux to select one input channel for conversion
  - SPIO, special function register



ADMUX register

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

REFS1:0 Bit 7:6 Reference Selection Bits
These bits select the reference voltage for the ADC.

ADLAR Bit 5 ADC Left Adjust Results
This bit dictates either the left bits or the right bits of the result registers ADCH:ADCL that are used to store the result. If we write a one to ADLAR, the result will be left adjusted; otherwise, the result is right adjusted.

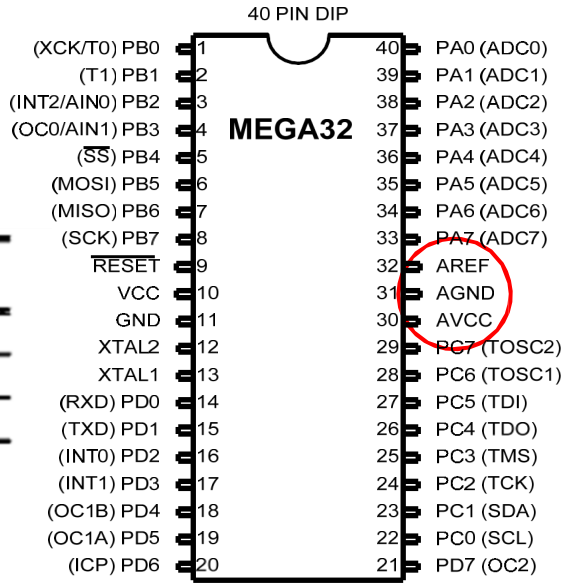
MUX4:0 Bit 4:0 Analog Channel and Gain Selection Bits
The value of these bits selects the gain for the differential channels and also selects which combination of analog inputs are connected to the ADC.

ADMUX register

- Select the reference voltage (the maximum acceptable input channel voltage)

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

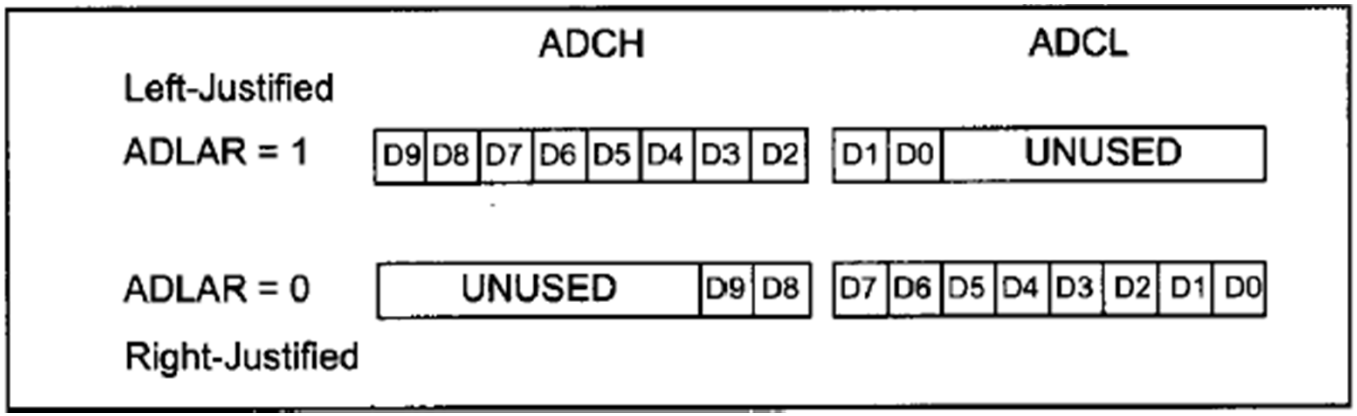
REFS1	REFS0	V_{ref}	
0	0	AREF pin	Set externally
0	1	AVCC pin	Same as VCC
1	0	Reserved	----
1	1	Internal 2.56 V	Fixed regardless of VCC value



ADMUX register

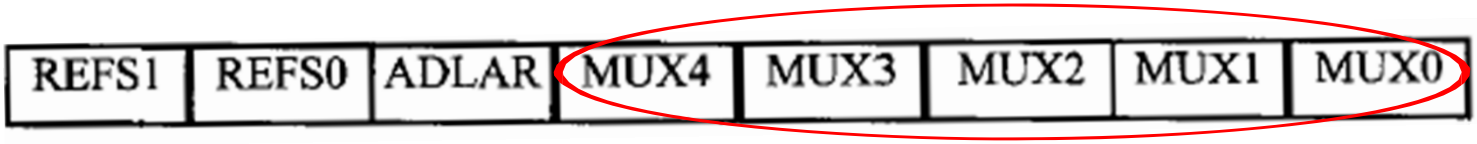
- Which bits of ADCH and ADCL are unused

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0



ADMUX register

- Which channel is selected to the ADC
  - Can also work in differential mode



MUX4...0	Single-ended Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7

ADCSRA register

- ADC control and status register
  - Control and monitor the ADC

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN Bit 7 ADC Enable
This bit enables or disables the ADC. Setting this bit to one will enable the ADC, and clearing this bit to zero will disable it even while a conversion is in progress.

ADSC Bit 6 ADC Start Conversion
To start each conversion you have to set this bit to one.

ADATE Bit 5 ADC Auto Trigger Enable
Auto triggering of the ADC is enabled when you set this bit to one.

ADIF Bit 4 ADC Interrupt Flag
This bit is set when an ADC conversion completes and the data registers are updated.

ADIE Bit 3 ADC Interrupt Enable
Setting this bit to one enables the ADC conversion complete interrupt.

ADPS2:0 Bit 2:0 ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

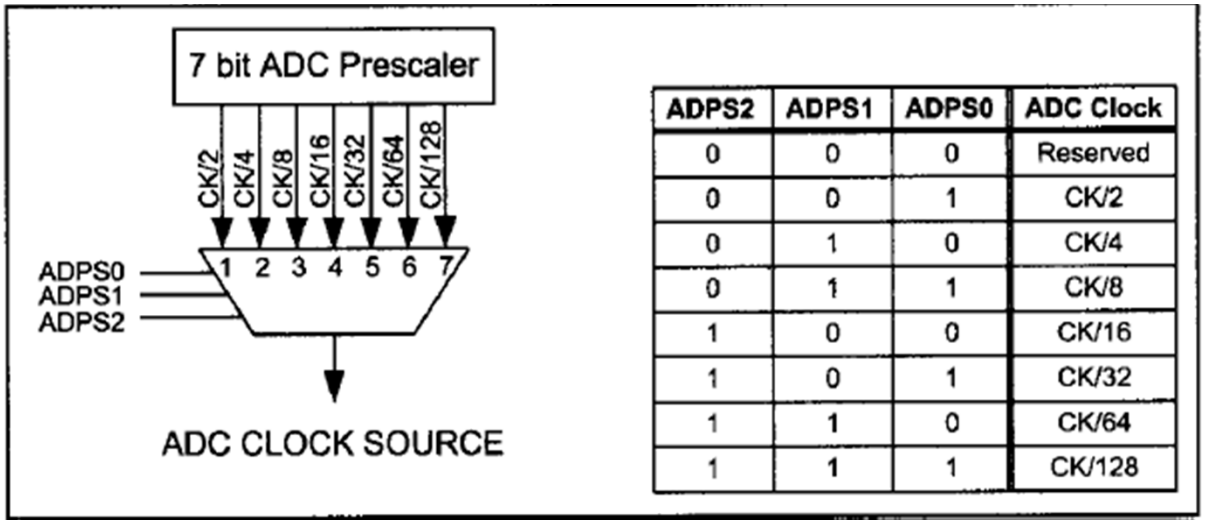
ADCSRA register

- ADATE bit
- Sets the operation mode of ADC
  - 0 = single conversion
    - Converts the input just one time every time ADSC becomes 1
  - 1 = free running
    - Continuously converts the input to digital values with some frequency
- For ADATE = 1 we have more ADC options, Take a look at AVR documents for more details if you are interested!

ADCSRA register

- Selecting the ADC clock

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0



ADPS2	ADPS1	ADPS0	ADC Clock
0	0	0	Reserved
0	0	1	CK/2
0	1	0	CK/4
0	1	1	CK/8
1	0	0	CK/16
1	0	1	CK/32
1	1	0	CK/64
1	1	1	CK/128

ADC clock

- Determines the speed of sampling the input data
  - Each conversion takes around 13 ADC clocks
- Larger frequency has smaller accuracy
- In AVR, frequency have to be less than 200 kHz
- Set it to smaller frequency (ADPS[0..2]=111) if speed is not critical

ADC programming in C

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 ADC input channels. We use the REFS0 and REFS1 bits in the ADMUX register to select voltage reference and the MUX4:0 bits in ADMUX to select the ADC input channel.
5. Activate the start conversion bit by writing a one to the ADSC bit of ADCSRA.
6. Wait for the conversion to be completed by polling the ADIF bit in the ADCSRA register.
7. After the ADIF bit has gone HIGH, read the ADCL and ADCH registers to get the digital data output. Notice that you have to read ADCL before ADCH; otherwise, the result will 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 V_{ref} source or input channel, go back to step 4.

ADC programming in C

- A program that coverts the analog voltage of ADC0 and copies it to PORTD and PORTB
  - Polling method

- The ADSC bit should be 1 for the ADC to start conversion
  - Returns to 0 automatically once the conversion is completed

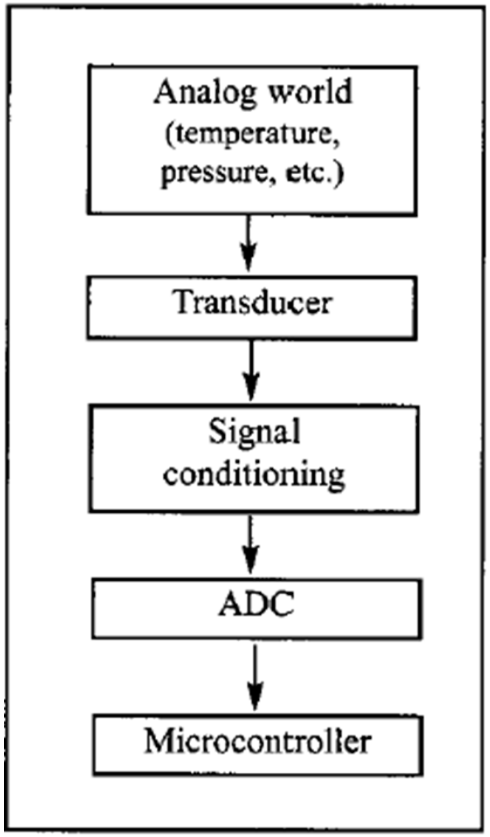
ADC programming in C

```
#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;
}
```

Connecting sensors to AVR

- Transducer (sensor): convert environment parameters (temperature, pressure, velocity,…) to electrical quantities (voltage, current, capacitance, resistance)

- Signal conditioning: converting electrical quantities to voltage

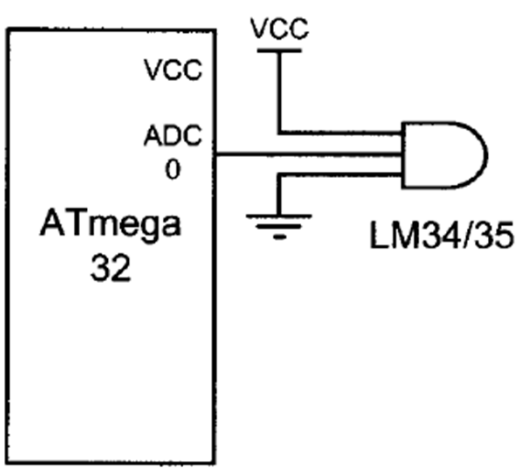


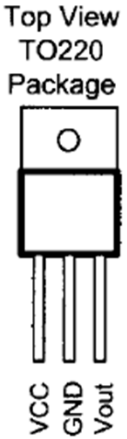
Connecting LM35 to AVR

Table 13-10: LM35 Temperature Sensor Series Selection Guide

Part	Temperature Range	Accuracy	Output Scale
LM35A	–55 C to +150 C	+1.0 C	10 mV/C
LM35	–55 C to +150 C	+1.5 C	10 mV/C
LM35CA	–40 C to +110 C	+1.0 C	10 mV/C
LM35C	–40 C to +110 C	+1.5 C	10 mV/C
LM35D	0 C to +100 C	+2.0 C	10 mV/C

Note: Temperature range is in degrees Celsius.





Connecting LM35 to AVR

- In AVR:
  - Vref=2.56v
  - 10 bit output, 1024 values ⇨ step size=2.56/1024= 2.5 mv
- In LM35 for 1 degree increase in temperature, we have 10mv increase in output voltage
- If the temperature increases by one degree:
  - 10mv increase in sensor output
  - 10/2.5=4 increase in ADC output
- Divide the ADC result by 4 to get actual temperature (shift right 2 times)

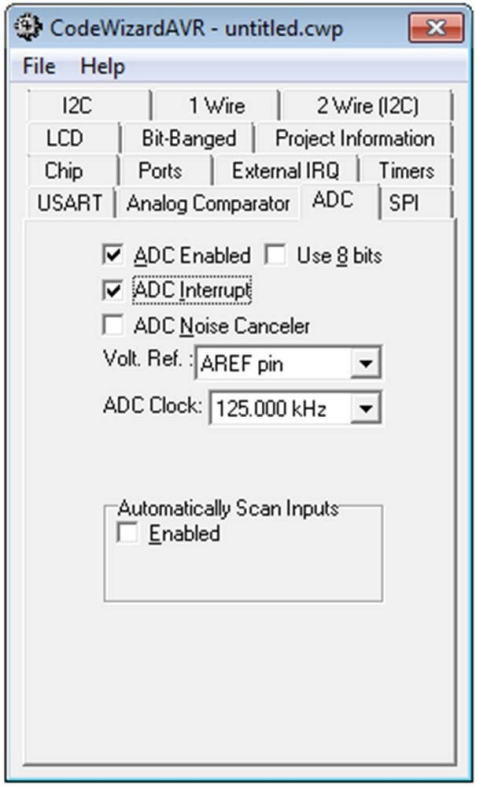
Connecting LM35 to AVR

Temp. (F)	V_{in} (mV)	# of steps	Binary V_{out} (b9–b0)	Temp. in Binary
0	0	0	00 00000000	00000000
1	10	4	00 00000100	00000001
2	20	8	00 00001000	00000010
3	30	12	00 00001100	00000011
10	100	20	00 00101000	00001010
20	200	80	00 01010000	00010100
30	300	120	00 01111000	00011110
40	400	160	00 10100000	00101000
50	500	200	00 11001000	00110010
60	600	240	00 11110000	00111100
70	700	300	01 00011000	01000110
80	800	320	01 01000000	01010000
90	900	360	01 01101000	01011010
100	1000	400	01 10010000	01100100

- Example:
  - Temp = 20 degrees, sensor output = 200mv (10mv for every degree), this voltage is converted to 0x1010000=80 (200mv/2.5mv=80) by the ADC
- Divide by 4 to get actual temperature by ADC

ADC programming in C by CodeVision

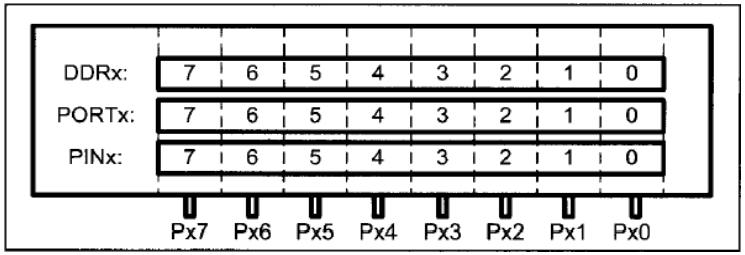
- Set different values in the project setup wizard




```
#include <mega32.h>

#define ADC_VREF_TYPE 0x00
// ADC interrupt service routine
interrupt [ADC_INT] void adc_isr(void)
{
unsigned int adc_data;
// Read the AD conversion result
adc_data=ADCW;
// Place your code here

}

// Declare your global variables here

void main(void)
{
// ADC initialization
// ADC Clock frequency: 125.000 kHz
// ADC Voltage Reference: AREF pin
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x8E;

// Global enable interrupts
#asm("sei")

while (1)
      {
      // Place your code here

      };
}
```

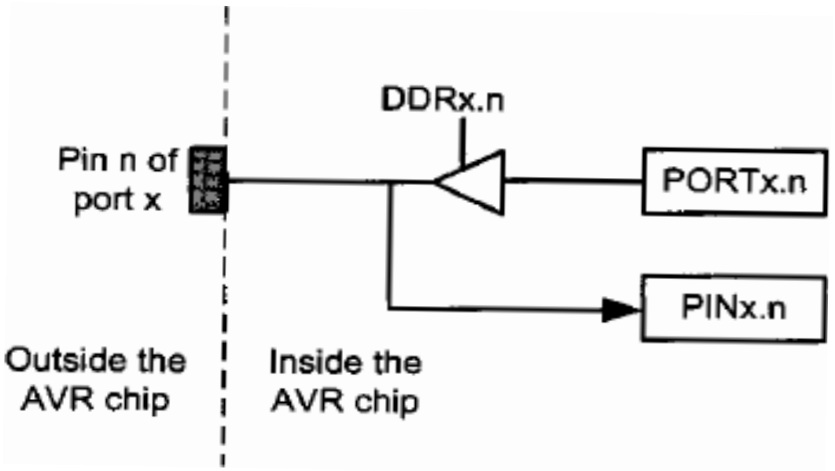
ATmega32 ports (up to/from here for session 26/27)

- All ports have read-modify-write capability
  - You can change pin direction and value, without effecting any other pins in the port
- Control of all ports and pins is done with three registers
  - DDRx (DDRB is data direction register port B)
  - PORTx (PORTB is the output register for port B)
  - PINx (PINB in the input register for port B)

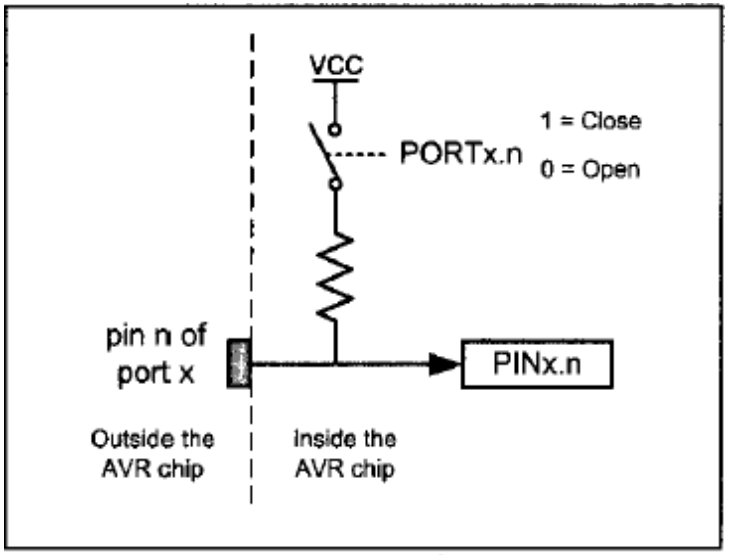


ATmega32 ports

- PORTx internal circuit



- PORTx in input mode enables the pullup for PINx



ATmega32 ports

- Port programming examples
  - Copy PD to PB

```
//configure PORTB as output
  PORTB=0X00;  DDRB=0XFF;
//configure PORTD as input
 PORTD=0XFF;  DDRD=0X00;
 While(1)
 {
   PORTB=PIND;
 }
```

  - An AVR C program that shows the count from 0x0 to 0xFF on LEDs

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
   DDRB = 0xFF;                        //Port B is output
   while (1)
   {
      PORTB = PORTB + 1;
   }
   return 0;
}
```

AVR interrupt

- Interrupt vs. polling
  - Polling: wasting time to check the devices continuously
  - What if we are to generate two delays at the same time?
    - Example: Toggle bit PB.5 every 1s and PB.4 every 0.5s.
  - What if there are some task to be done simultaneously with the timers?
    - Example: (1) read the contents of port A, process the data, and send them to port D continuously, (2) toggle bit PB.5 every 1s, and (3) PB.4 every 0.5s.

AVR interrupt

- Interrupts
  - A mechanism to work with peripheral devices
- No need for the processor to monitor the status of the devices and events
- Let the events notify the processor when they occur
  - By sending an interrupt signal to processor

AVR interrupt

- Example:
  - Copy the contents of port A to port D continuously and toggle bit PB.5 every 1s and PB.4 every 0.5s.
- Solution:
  - Copying the contents of port A to port D as the main program
  - Get timers 0 and 1 to generate the delays
  - Define two interrupts for timers 0 and 1 to notify the processor when they finish counting
  - Upon an interrupt, stop the main program, service the timers and continue the main program

AVR interrupt

- Interrupting mechanism in all microprocessors and microcontrollers is almost the same:
  - Define the set of devices and events that can generate an interrupt
  - Write a function for each interrupt that will be executed when the corresponding interrupt is activated
    - The address of this function must be saved somewhere
  - Set a priority scheme among interrupts
  - A mechanism is needed to disable all or some interrupts

AVR interrupt

- ISR: Interrupt Service Routine
  - The function that is executed when an interrupt is enabled
- IVT: Interrupt Vector Table
  - A table that keeps the address of each ISR in the instruction memory

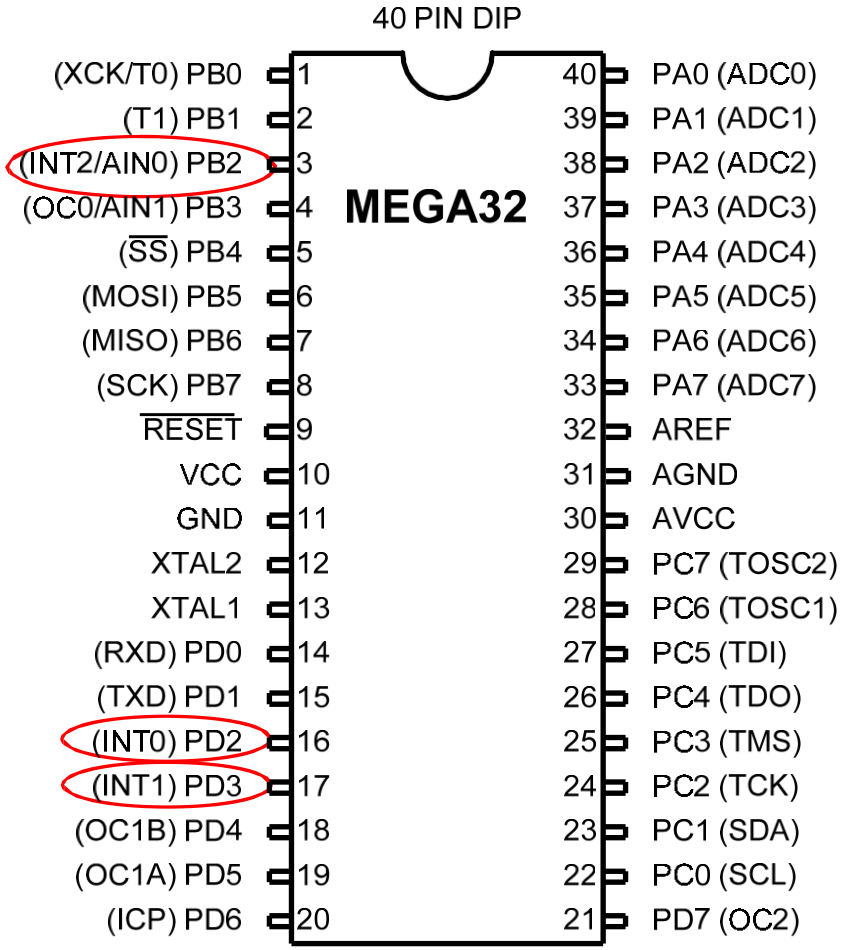
AVR interrupt sources

- Some AVR devices may be a little different!

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

External interrupts in Atmega32

- Three pins



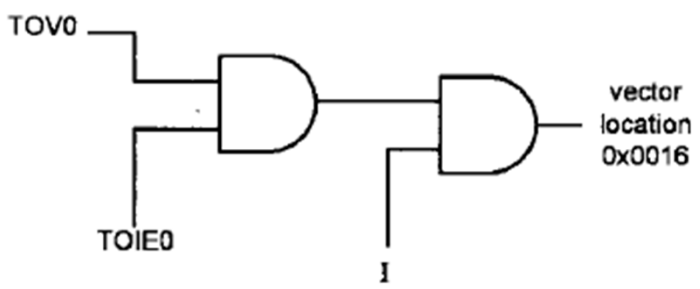
How is an interrupt serviced?

- The interrupt cycle
  - Stop fetching the next instruction and save PC
  - Go to Interrupt Vector Table to find the address of the ISR of the interrupting device
  - Execute the function
  - Resume normal execution by retrieving PC

- Interrupts can be enabled or disabled by programmer
  - Bit7 (I) in SREG (status register)
  - SREG keeps the processor status
  - Disabled on reset (I=0)

Enabling interrupts

- In addition to Bit7 (I) in SREG each interrupt should be enabled independently
- The enable bit of each interrupt is in some register
  - Example: TIMSK register to enable/disable timer interrupts
  - TIMSK= Timer Interrupt Mask
    - Mask



Enabling interrupts

- To enable timer1 overflow interrupt:
  - Bit7 (I) in SREG ⇨ 1
  - Bit0 of TIMSK (TOIE0) ⇨ 1

Interrupt priority

- What if two or more interrupts occur at the same time?
  - The interrupt with lower ISR address is prioritized (external int. 0 has the highest priority)

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Interrupt priority

- When an interrupt is serviced, the I bit becomes automatically 0 to disable interrupts
  - Will be enabled when returning from the ISR

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Interrupt programming in C

- Enable interrupts
- Set the mask register
  - Example: TIMSK=0x01;
- Write the ISR function to specify what operation should be done when the interrupt occurs
- Compiler dependent
  - Varies in different compilers!

Interrupt programming in C

- A way to use assembly instructions in C:
  - #asm (“instruction”)
- SEI: (Set I) an assembly instruction that enables interrupts (bit 7 of SREG=1)
- CLI: Clear I
  - #asm(“sei”);
    - enable interrupts in C

Example 10-8 (C version of Program 10-1)

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

Solution:

```
#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
      #asm("sei");             //enable interrupts

      DDRC = 0x00;             //make PORTC input
      DDRD = 0xFF;             //make PORTD output

      while (1)                //wait here
            PORTD = PINC;
}

interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
      TCNT0 = -32;
      PORTB ^= 0x20;           //toggle PORTB.5
}
```

On entering the timers ISR, the TOV0 bit is automatically cleared, no need to be cleared by software

Interrupt programming in C

- No way to access bit 7 of SREG in C
- Some books uses a different compiler
  - Different instructions:
    - sei()
      - instead of #asm("sei"), different ISR names

Example 10-8 (C version of Program 10-1)

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

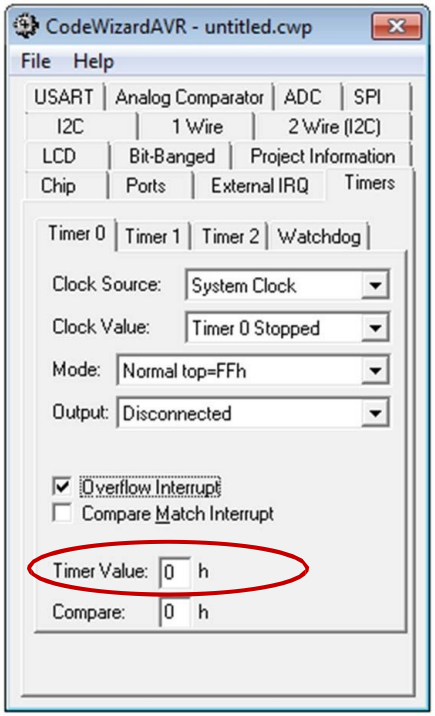
Solution:

```
#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
      #asm("sei");             //enable interrupts

      DDRC = 0x00;             //make PORTC input
      DDRD = 0xFF;             //make PORTD output

      while (1)                //wait here
            PORTD = PINC;
}

interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
      TCNT0 = -32;
      PORTB ^= 0x20;           //toggle PORTB.5
}
```

On entering the timers ISR, the TOV0 bit is automatically cleared, no need to be cleared by software

Interrupt programming in C

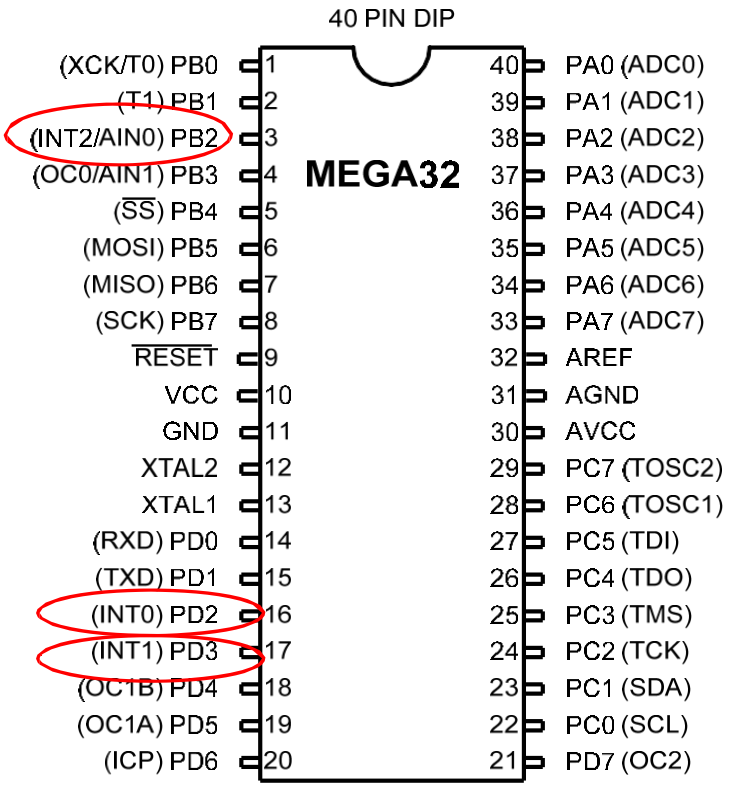
- In CodeVision, ISR is generated during project setup.
  - Just fill the function body!




```
#include <mega32.h>


// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here
}

// Declare your global variables here
 void main(void)
{

TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;

// Global enable interrupts
#asm("sei")

while (1)
      {
      // Place your code here

      };
}
```

External interrupts

- To allow external sources interrupt the microcontroller
- Can be masked by GICR register
- GICR: General Interrupt Control Register



External interrupts

- GICR: General Interrupt Control Register

D7							D0
INT1	INT0	INT2	-	-	-	IVSEL	IVCE

INT0 External Interrupt Request 0 Enable
= 0 Disables external interrupt 0
= 1 Enables external interrupt 0

INT1 External Interrupt Request 1 Enable
= 0 Disables external interrupt 1
= 1 Enables external interrupt 1

INT2 External Interrupt Request 2 Enable
= 0 Disables external interrupt 2
= 1 Enables external interrupt 2

These bits, along with the I bit, must be set high for an interrupt to be responded to.

- **Bit 1 – IVSEL: Interrupt Vector Select**

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash.

- **Bit 0 – IVCE: Interrupt Vector Change Enable**

The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts.

External interrupts

- Interrupts can be edge triggered or level triggered
  - Edge trigger: activated when a change in signal level occurs
  - Level trigger: activated when a signal has a specific value
- INT0 and INT1 can be programmed to be edge or level triggered
  - Low-level active by default
- INT 2 is only edge triggered

External interrupts

- ISC (Interrupt Sense Control) register
  - Can set the interrupt type of INT0 and INT1

ISC Register

SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

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.

ISC10 and ISC11 set the same setting for INT1

External interrupts

- C programming!

Example 10-12 (C version of Example 10-5)

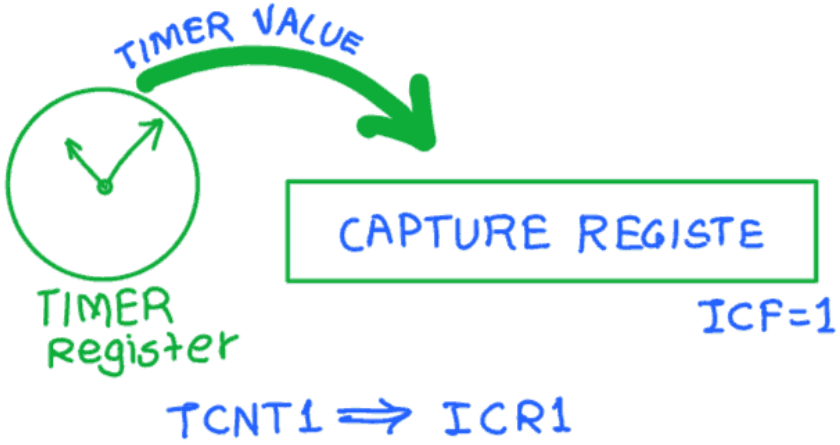
Assume that the INT0 pin is connected to a switch that is normally high. Write a program that toggles PORTC.3, whenever INT0 pin goes low. Use the external interrupt in level-triggered mode.

Solution:

```
#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
      #asm("sei");              //enable interrupts
      while (1);                //wait here
}

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

Timers in AVR (up to/from here for session 27/28)

- ATmega32: 3 timers
  - Timer0 (8-bit)
  - Timer1 (16-bit)
  - Timer2 (8-bit)



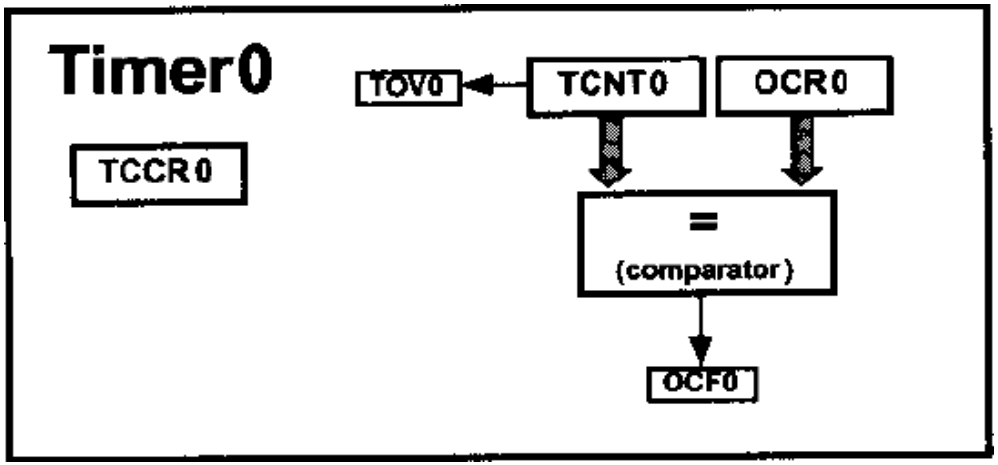
Timers in AVR

- Basic registers:
  - TCNTx (x=0,1,2) = Timer/counter register
    - Keeps the timer/counter value
    - On reset, contains 0
    - Counts up with each pulse
  - TOVx(x=0,1,2) = Timer/counter overflow flag
    - TOVx Becomes 1 when TCNTx overflows
      - Switches from 0xFF to 0x00
    - Should be reset by software

	D7	D6	D5	D4	D3	D2	D1	D0
TCNT0								

Timers in AVR

- Basic registers:
  - OCRx (x=0,1,2) = Output compare register
    - Another way to count
    - The contents of OCRx are compared to TCNTx
    - OCFx is set if they are equal



Timers in AVR

- Basic registers:
  - TCCRx (x=0,1,2) = Timer/counter control register
    - Setting modes of operation
    - Bit 7 - FOC0: Force compare match
      - After setting this bit, timer forced to match occur. i.e. setting output compare flag.

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

Timers in AVR

- Basic registers:
  - TCCRx (x=0,1,2) = Timer/counter control register

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

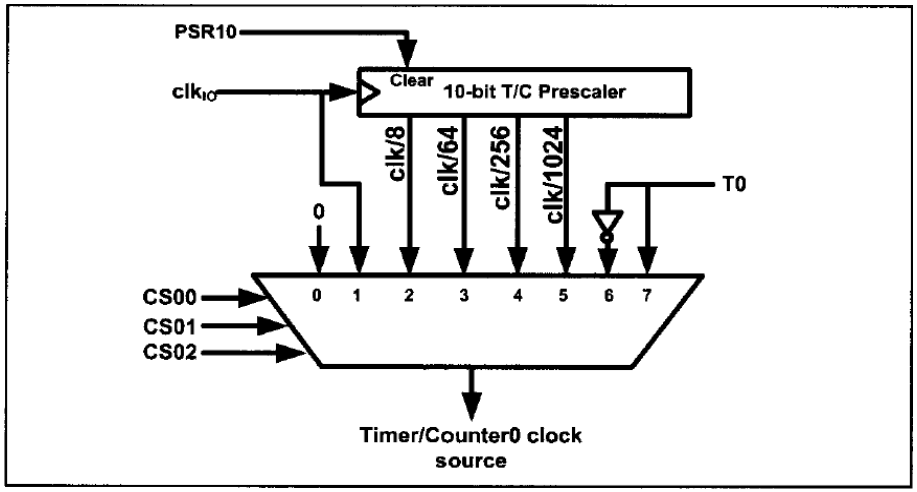
COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

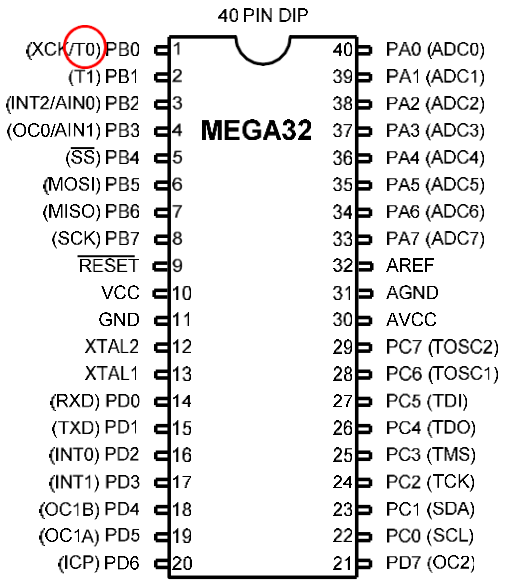
CS02:00	D2	D1	D0	Timer0 clock selector
	0	0	0	No clock source (Timer/Counter stopped)
	0	0	1	clk (No Prescaling)
	0	1	0	clk / 8
	0	1	1	clk / 64
	1	0	0	clk / 256
	1	0	1	clk / 1024
	1	1	0	External clock source on T0 pin. Clock on falling edge.
	1	1	1	External clock source on T0 pin. Clock on rising edge.

Timers in AVR

- TCCR0

CS02:00	D2	D1	D0	Timer0 clock selector
	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.





Timers in AVR

- TCCR0 in AVR

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

WGM00, WGM01

D6	D3	Timer0 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

Timers in AVR

- TIFR (timer/counter interrupt flag register)
- To keep the state of the counters
- One register for all counter/timers

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
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

TOV0 D0 Timer0 overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (going from $FF to $00).

OCF0 D1 Timer0 output compare flag bit
0 = compare match did not occur.
1 = compare match occurred.

TOV1 D2 Timer1 overflow flag bit

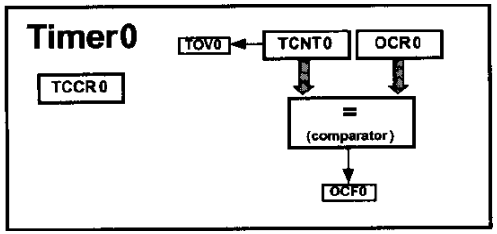
OCF1B D3 Timer1 output compare B match flag

OCF1A D4 Timer1 output compare A match flag

ICF1 D5 Input Capture flag

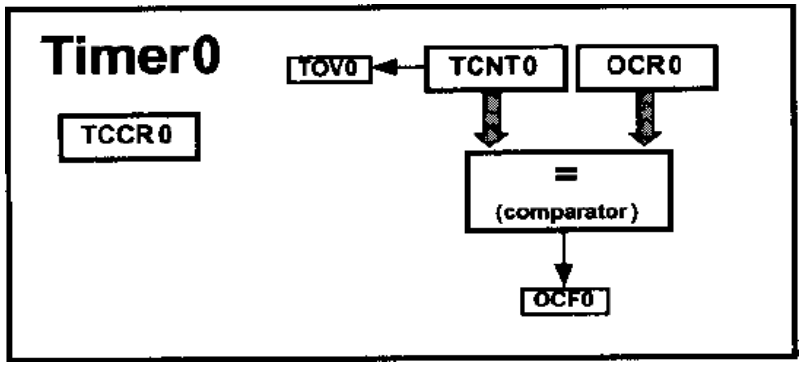
TOV2 D6 Timer2 overflow flag

OCF2 D7 Timer2 output compare match flag



Timers in AVR

- Timer0 in normal mode
  - Set TCNT0 with proper value
  - Set TCCR0: which clock source? Which prescalar?
    - When is set, the timer starts
  - Keep monitoring TOV0
  - Stop timer Set TCCR0
  - Clear TOV0



Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

Timers in AVR

Example 9-11

Find the value for TCCR0 if we want to program Timer0 in Normal mode with a prescaler of 64 using internal clock for the clock source.

Solution:

From Figure 9-5 we have TCCR0 = 0000 0011; XTAL clock source, prescaler of 64.

TCCR0 =

0	0	0	0	0	0	1	1
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

D2	D1	D0	Timer0 clock selector
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.

D6	D3	Timer0 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

Timers in AVR

Example 9-1

Find the value for TCCR0 if we want to program Timer0 in Normal mode, no prescaler. Use AVR's crystal oscillator for the clock source.

Solution:

TCCR0 =

0	0	0	0	0	0	0	1
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

D2	D1	D0	Timer0 clock selector
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.

D6	D3	Timer0 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

Timers in AVR

Example 9-7

Assuming that XTAL = 8 MHz, write a program to generate a square wave with a period of 12.5 µs on pin PORTB.3.

Solution:

For a square wave with T = 12.5 µs we must have a time delay of 6.25 µs. Because XTAL = 8 MHz, the counter counts up every 0.125 µs. This means that we need 6.25 µs / 0.125 µs = 50 clocks. 256 – 50 = 206 = 0xCE. Therefore, we have TCNT0 = 0xCE.

```
TCCR0=0x01      //normal mode, no prescaling
TCNT0=0xCE
```

Timers in AVR

Example 9-8

Assuming that XTAL = 8 MHz, modify the program in Example 9-7 to generate a square wave of 16 kHz frequency on pin PORTB.3.

Solution:

Look at the following steps.
(a) T = 1 / F = 1 / 16 kHz = 62.5 μs the period of the square wave.
(b) 1/2 of it for the high and low portions of the pulse is 31.25 μs.
(c) 31.25 μs / 0.125 μs = 250 and 256 - 250 = 6, which in hex is 0x06.
(d) TCNT0 = 0x06.

Timers in AVR

- Compare mode (clear timer on compare) = CTC mode
  - Another way to count
  - Increment TCNT at each clock cycle
  - When OCRn=TCNTn
    - OCFn=1

Example 9-20

Assuming XTAL = 8 MHz, write a program to generate a delay of 25.6 ms. Use Timer0, CTC mode, with prescaler = 1024.

Solution:

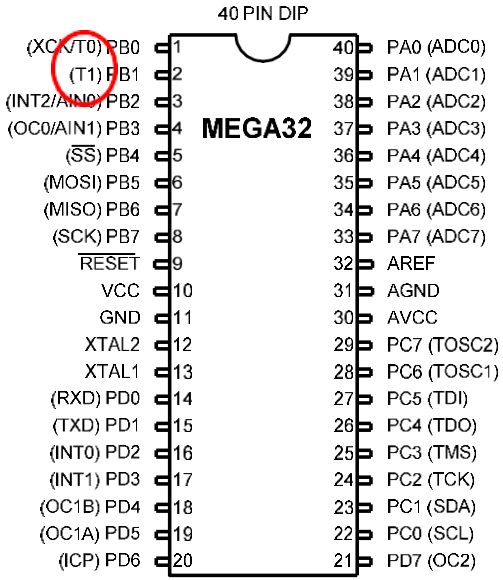
Due to prescaler = 1024 each timer clock lasts 1024 × 0.125 µs = 128 µs. Thus, in order to generate a delay of 25.6 ms we should wait 25.6 ms / 128 µs = 200 clocks. Therefore the OCR0 register should be loaded with 200 – 1 = 199.

Timers in AVR

- Timer 2 in ATmega32
  - Just like timer 0, but no external clock
    - Timer only
  - TCCR2:

Bit	7	6	5	4	3	2	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

CS22:20	D2	D1	D0	Timer2 clock selector
	0	0	0	No clock source (Timer/Counter stopped)
	0	0	1	clk (No Prescaling)
	0	1	0	clk / 8
	0	1	1	clk / 32
	1	0	0	clk / 64
	1	0	1	clk / 128
	1	1	0	clk / 256
	1	1	1	clk / 1024



Timers in AVR

- Timer programming in C
  - We can use the register names in C codes:
    - TCNT0, TCNT1, TCNT2
    - TIFR0, TCCR0,…
  - TOV0 is cleared by writing a logic one to the flag

Example 9-40

Write a C program to toggle only the PORTB.4 bit continuously every 70 µs. Use Timer0, Normal mode, and 1:8 prescaler to create the delay. Assume XTAL = 8 MHz.

Solution:

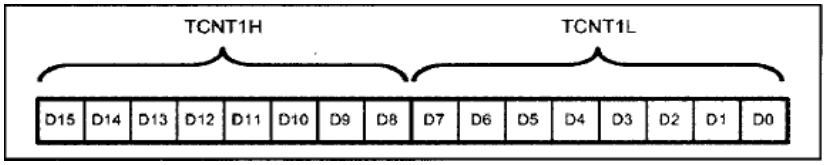
XTAL = 8MHz ➔ $T_{machine\ cycle}$ = 1/8 MHz
Prescaler = 1:8 ➔ T_{clock} = 8 × 1/8 MHz = 1 µs
70 µs/1 µs = 70 clocks ➔ 1 + 0xFF – 70 = 0x100 – 0x46 = 0xBA = **186**

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

void T0Delay ( );

int main ( )
{
      DDRB = 0xFF;       //PORTB output port

      while (1)
      {
            T0Delay ( );            //Timer0, Normal mode
            PORTB = PORTB ^ 0x10;   //toggle PORTB.4
      }
}

void T0Delay ( )
{
      TCNT0 = 186;       //load TCNT0
      TCCR0 = 0x02;      //Timer0, Normal mode, 1:8 prescaler
      while ((TIFR&(1<<TOV0))==0);  //wait for TOV0 to roll over

      TCCR0 = 0;         //turn off Timer0
      TIFR = 0x1;        //clear TOV0
}
```

Timers in AVR

- Timer 1
  - 16-bit counter/timer
  - TCNT1L and TCNT1H
  - 2×8-bit registers to control timer 1
    - TCCR1L and TCCR1H
  - 2 registers in compare mode
    - OCR1A and OCR1B

TCNT1

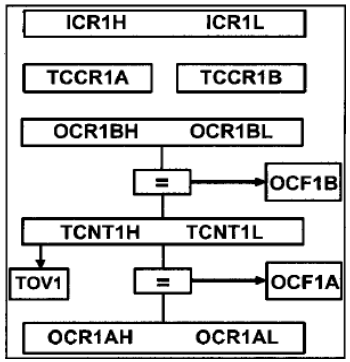


3 flags in TIFR:

TOV1

and

OCF1A- OCF1B



Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
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

TOV0	D0	Timer0 overflow flag bit 0 = Timer0 did not overflow. 1 = Timer0 has overflowed (going from $FF to $00).
OCF0	D1	Timer0 output compare flag bit 0 = compare match did not occur. 1 = compare match occurred.
TOV1	D2	Timer1 overflow flag bit
OCF1B	D3	Timer1 output compare B match flag
OCF1A	D4	Timer1 output compare A match flag
ICF1	D5	Input Capture flag
TOV2	D6	Timer2 overflow flag
OCF2	D7	Timer2 output compare match flag

Timers in AVR

- Timer 1 control registers
  - 2 registers
  - Plenty of operation modes

TCCR1A

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TCCR1B

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Timers in AVR

- Timer 1 control registers

In this course, we focus on modes 0 and 4

Mode	WGM13	WGM12	WGM11	WGM10	Timer/Counter Mode of Operation	Top	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

TCCR1A

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TCCR1B

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Timers in AVR

- Accessing 16-bit registers in AVR
  - TCNT1=0x05ff, we want to save the content of TCNT1 in R20 and R21
  - Cannot read TCNT in one cycle
    - AVR is a 8-bit machine
    - Read TCNT1L (0xff) at t0, at the same cycle occurs TCNT=0x0600
    - Read TCNT1H (0x06)
    - The content is detected as 0x06ff instead of the correct value 0x05ff

Timers in AVR

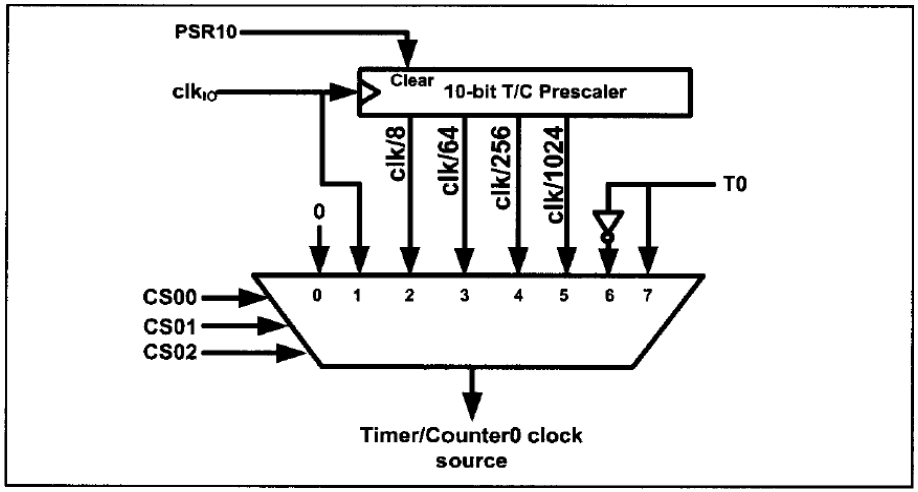
- Accessing 16-bit registers in AVR
  - Solution:
    - AVR buffers the high byte when the lower byte is read
    - When the higher byte is read, the buffered value is used
      - first read the lowest byte and then the higher byte

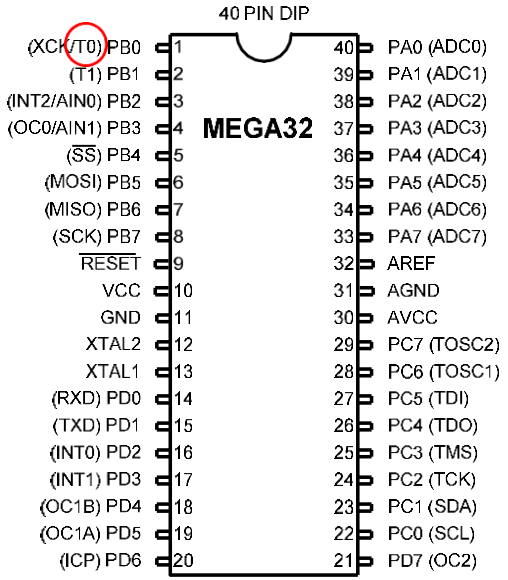
```
IN    R20,TCNT1L        ;R20 = TCNT1L, TEMP = TCNT1H
IN    R21,TCNT1H        ;R21 = TEMP of Timer1
```

Counters in AVR

- To count external events

CS02:00	D2	D1	D0	Timer0 clock selector
	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.

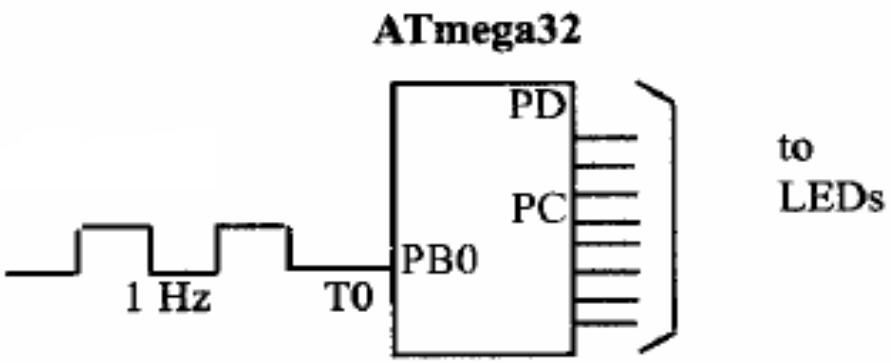




Counter programming in AVR

- Configure T0 (PB0) or T1 (PB1) as input
- Set the other registers as in timers!
- Example:

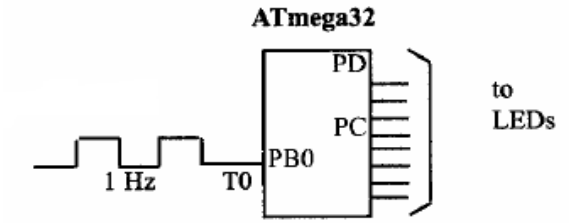
Assuming that a 1 Hz clock pulse is fed into pin T0, use the TOV0 flag to extend Timer0 to a 16-bit counter and display the counter on PORTC and PORTD.



Counter programming in AVR

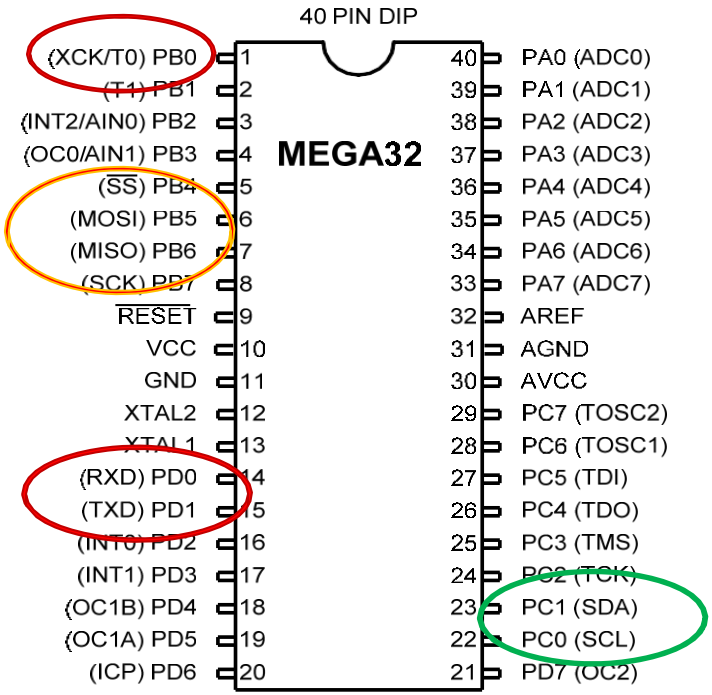
- Solution:

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

int main ( )
{
      PORTB = 0x01;            //activate pull-up of PB0
      DDRC = 0xFF;             //PORTC as output
      DDRD = 0xFF;             //PORTD as output

      TCCR0 = 0x06;            //output clock source
      TCNT0 = 0x00;

      while (1)
      {
            do
            {
                  PORTC = TCNT0;
            }while((TIFR&(0x1<<TOV0))==0);//wait for TOV0 to roll over

            TIFR = 0x1<<TOV0;        //clear TOV0
            PORTD ++;                //increment PORTD
      }
}
```



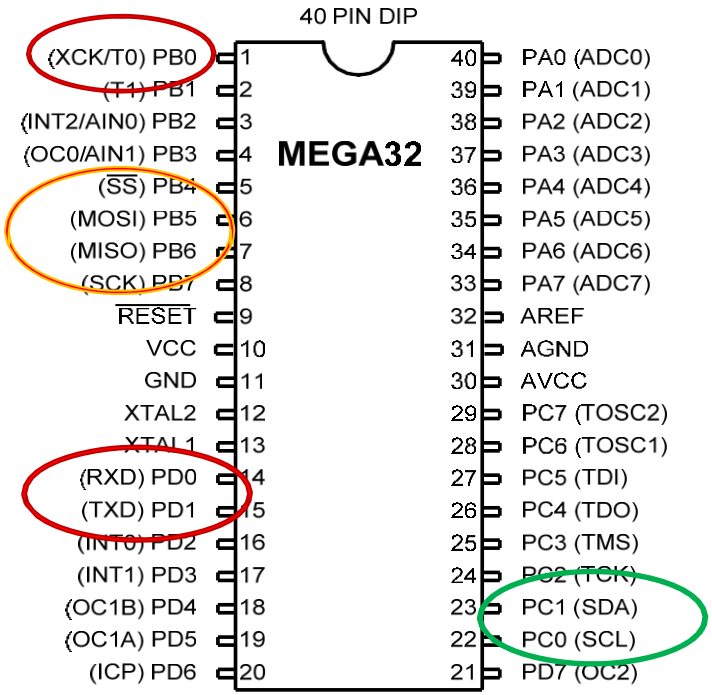
Serial ports in AVR

- USART
  - An old and widely used standard (pins 1, 14 and 15 of ATmega32)
- SPI
  - Very high-speed
    - Almost two orders of magnitude faster than USART
  - Synchronous serial port



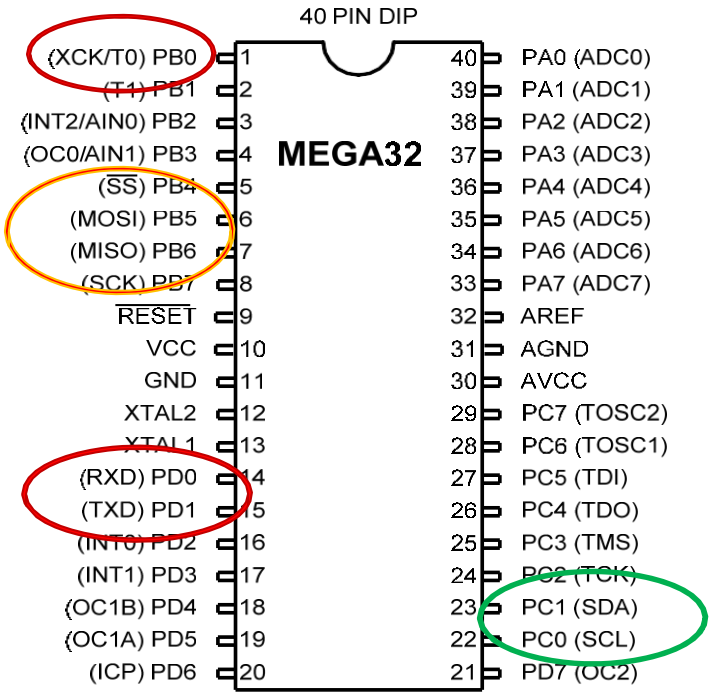
Serial ports in AVR

- SPI
  - Usage example
    - High-speed data transfer to EPROM
  - Pins 5 to 8 of Atmega32
- I^2C (Inter IC)
  - Connection using a shared bus connecting up to 128 devices



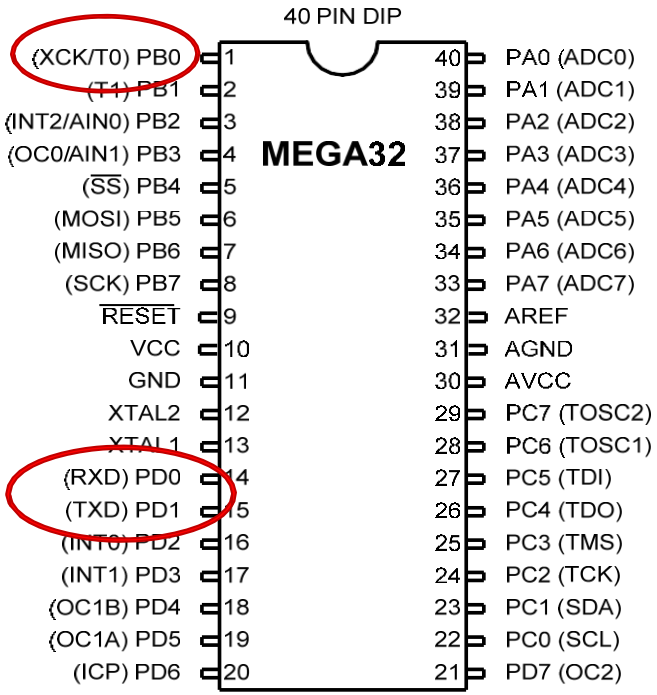
Serial ports in AVR

- I^2C (Inter IC)
  - Pins 22 and 23 of ATmega32

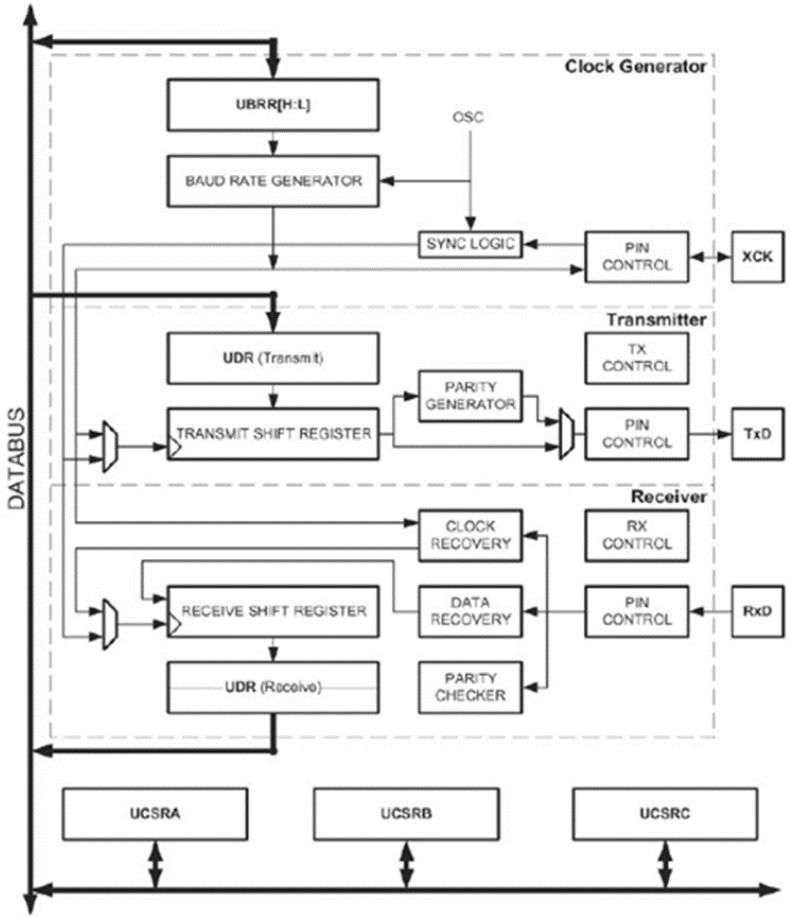


USART in AVR

- USART: Universal Synchronous/Asynchronous Receiver Transmitter
- An standard IC that can provide both synchronous and asynchronous communication
- It is controlled by some AVR registers

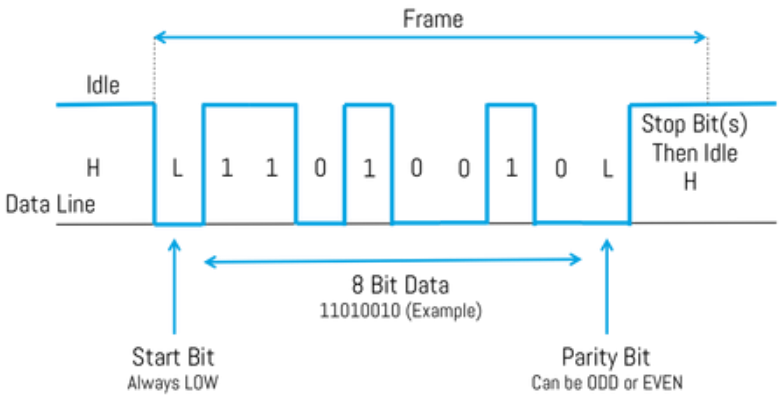


USART block diagram



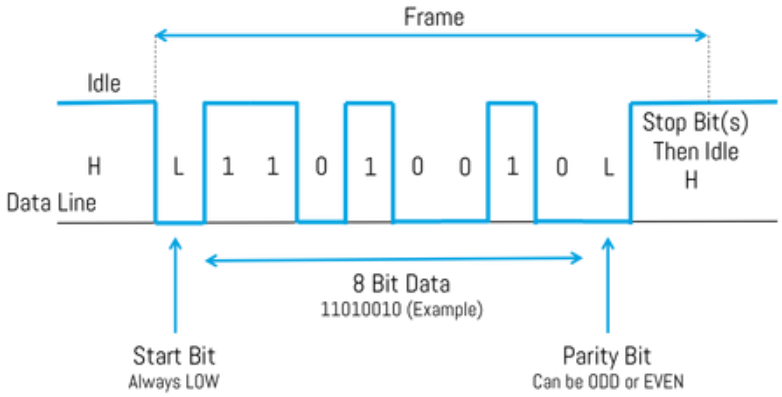
How to use USART?

- The control registers specify
  - The mode of operation
    - Synchronous or asynchronous
  - Parity bit: odd or even
  - Information unit: 5,6,7,8, or 9 bits
  - Information unit separation
    - how to specify the transmission of a word starts and stops?
  - Transmission rate



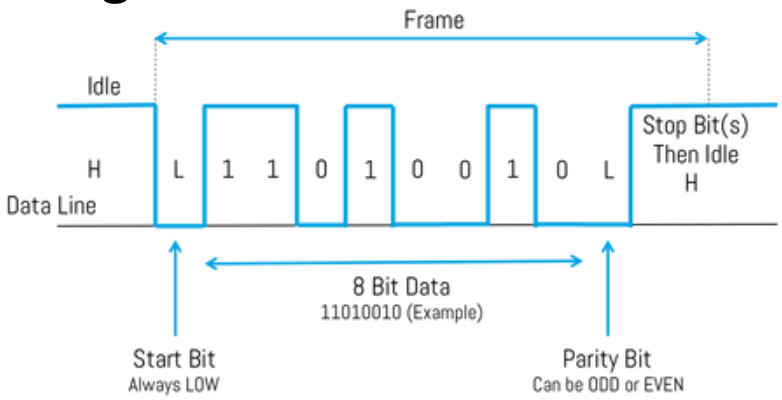
Parity bit

- A way to detect error during data transmission
  - Due to external noises
- Even parity
  - The number of 1s must be an even number
- Add an extra bit to the 8-bit data, called parity bit



Parity bit

- How does it work?
  - If the number of 1s is already even, set it to 0, otherwise to 1
  - Send the parity bit with data
  - If the other side detects odd number of 1s, there is something wrong

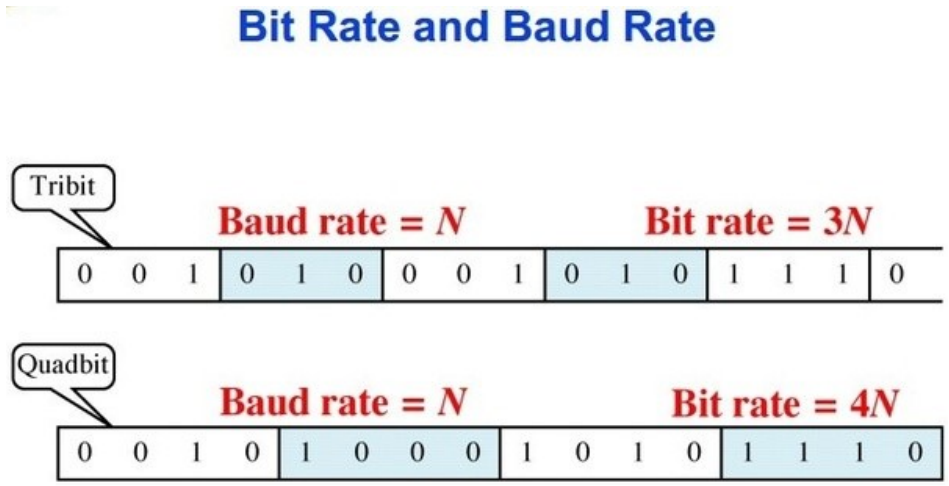


Transmission rate

- Baud rate vs bit rate
  - Baud rate: the number signal changes in a second
  - Bit rate: the number of bits transmitted per second
- Baud rate is not necessarily equal to bit rate
  - Each signal may carry several bits!
- A signal with 8 levels can carry 3 bits
  - If baud rate=x, bit rate=3x

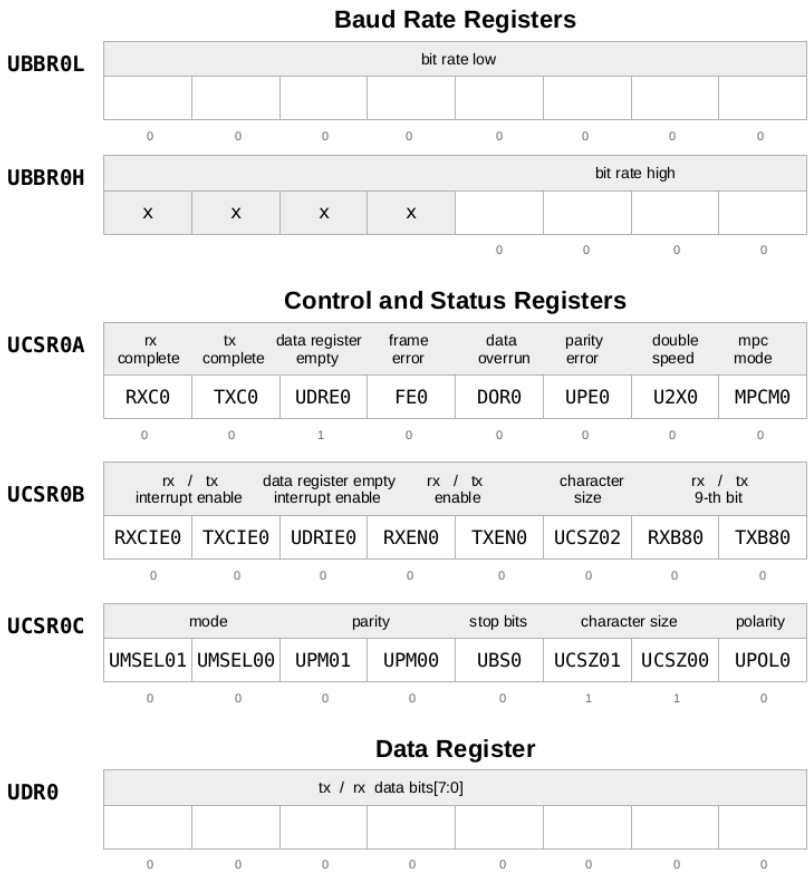
Transmission rate (up to/from here for session 28/29)

- In USART, baud rate= bit rate
- The bit rate of both devices connected to the same serial port must be the same



USART registers

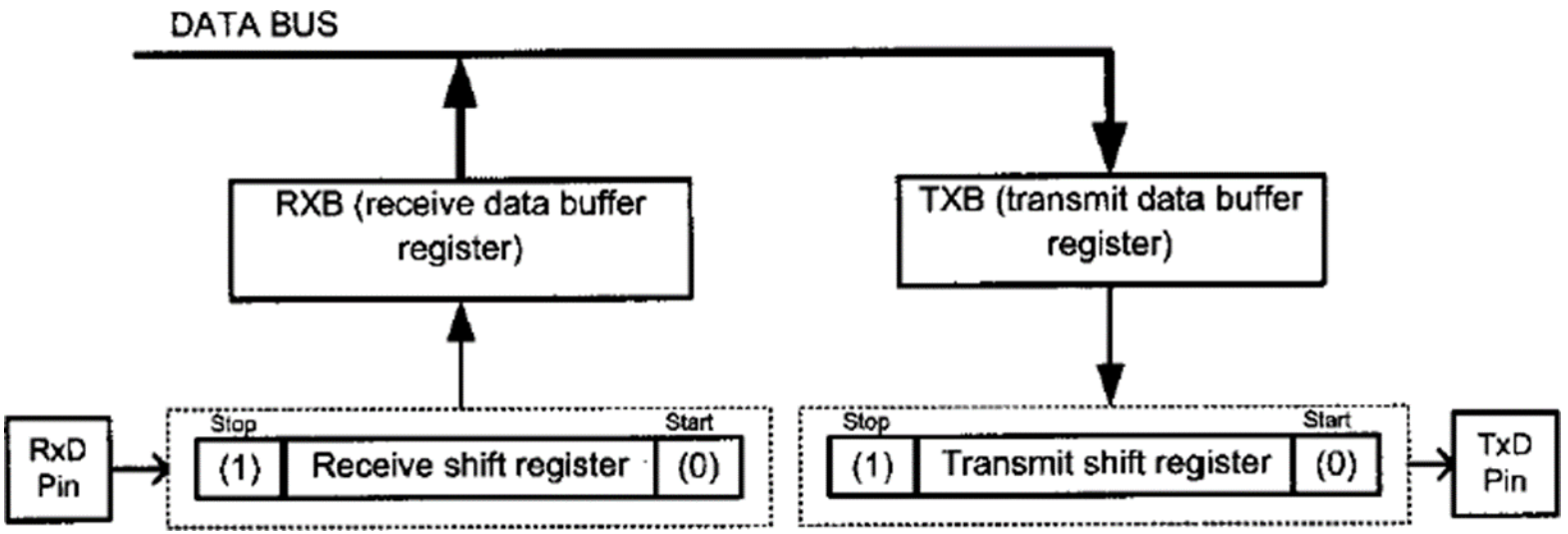
- Five registers are associated with USART port
  - UDR
    - USART data register
  - UCSRA, UCSRB, UCSRC
    - USART control and status register
  - UBRR
    - USART baud rate register



USART data register (UDR)

- Actually two registers
  - One for the transmit direction, the other for receive direction
- Share the same address and name
  - When UDR is read, the data received from the serial line is returned
  - When a data is written to UDR, it is directed to the transmit line

USART data register (UDR)



USART control and status register (UCSR)

- Three 8-bit registers to control the USART operation
- UCSRA

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

RXC (Bit 7): USART Receive Complete
This flag bit is set when there are new data in the receive buffer that are not read yet. It is cleared when the receive buffer is empty. It also can be used to generate a receive complete interrupt.

TXC (Bit 6): USART Transmit Complete
This flag bit is set when the entire frame in the transmit shift register has been transmitted and there are no new data available in the transmit data buffer register (TXB). It can be cleared by writing a one to its bit location. Also it is automatically cleared when a transmit complete interrupt is executed. It can be used to generate a transmit complete interrupt.

UDRE (Bit 5): USART Data Register Empty
This flag is set when the transmit data buffer is empty and it is ready to receive new data. If this bit is cleared you should not write to UDR because it overrides your last data. The UDRE flag can generate a data register empty interrupt.

USART control and status register (UCSR)

- UCSRA

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

FE (Bit 4): Frame Error
This bit is set if a frame error has occurred in receiving the next character in the receive buffer. A frame error is detected when the first stop bit of the next character in the receive buffer is zero.

DOR (Bit 3): Data OverRun
This bit is set if a data overrun is detected. A data overrun occurs when the receive data buffer and receive shift register are full, and a new start bit is detected.

PE (Bit 2): Parity Error
This bit is set if parity checking was enabled (UPM1 = 1) and the next character in the receive buffer had a parity error when received.

USART control and status register (UCSR)

- UCSRA

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

U2X (Bit 1): Double the USART Transmission Speed
Setting this bit will double the transfer rate for asynchronous communication.

MPCM (Bit 0): Multi-processor Communication Mode
This bit enables the multi-processor communication mode. The MPCM feature is not discussed in this book.

USART control and status register (UCSR)

- UCSRB

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

RXCIE (Bit 7): Receive Complete Interrupt Enable
To enable the interrupt on the RXC flag in UCSRA you should set this bit to one.

TXCIE (Bit 6): Transmit Complete Interrupt Enable
To enable the interrupt on the TXC flag in UCSRA you should set this bit to one.

UDRIE (Bit 5): USART Data Register Empty Interrupt Enable
To enable the interrupt on the UDRE flag in UCSRA you should set this bit to one.

RXEN (Bit 4): Receive Enable
To enable the USART receiver you should set this bit to one.

USART control and status register (UCSR)

- UCSRB

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

TXEN (Bit 3): Transmit Enable
To enable the USART transmitter you should set this bit to one.

UCSZ2 (Bit 2): Character Size
This bit combined with the UCSZ1:0 bits in UCSRC sets the number of data bits (character size) in a frame.

RXB8 (Bit 1): Receive data bit 8
This is the ninth data bit of the received character when using serial frames with nine data bits. This bit is not used in this book.

TXB8 (Bit 0): Transmit data bit 8
This is the ninth data bit of the transmitted character when using serial frames with nine data bits. This bit is not used in this book.

USART control and status register (UCSR)

- UCSRC

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

URSEL (Bit 7): Register Select
This bit selects to access either the UCSRC or the UBRRH register and will be discussed more in this section.

UMSEL (Bit 6): USART Mode Select
This bit selects to operate in either the asynchronous or synchronous mode of operation
0 = Asynchronous operation
1 = Synchronous operation

UPM1:0 (Bit 5:4): Parity Mode
These bits disable or enable and set the type of parity generation and check.
00 = Disabled
01 = Reserved
10 = Even Parity
11 = Odd Parity

Note on bit7: UCSRC and UBBR share the same address due to some technical issue. Set URSEL=1 when you want the data to be written to UCSRC, otherwise set URSEL=0 to write to UBBR

USART control and status register (UCSR)

- UCSRC

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

USBS (Bit 3): Stop Bit Select
This bit selects the number of stop bits to be transmitted.
0 = 1 bit
1 = 2 bits

UCSZ1:0 (Bit 2:1): Character Size
These bits combined with the UCSZ2 bit in UCSRB set the character size in a frame and will be discussed more in this section.

UCPOL (Bit 2): Clock Polarity
This bit is used for synchronous mode only and will not be covered in this section.

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

USART control and status register (UCSR)

- UCSRC

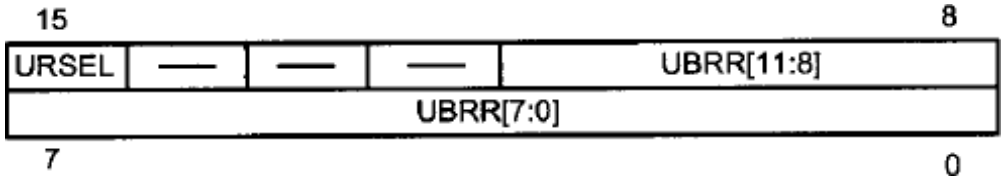
URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

Table 11-5: Values of UCSZ2:0 for Different Character Sizes

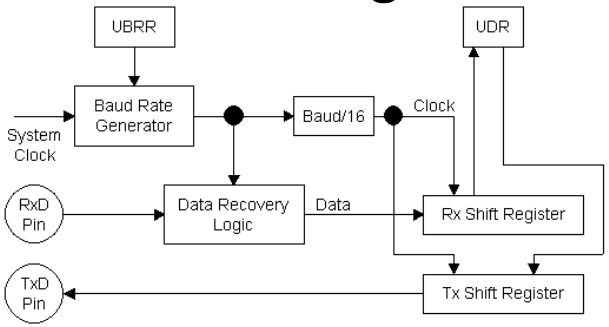
UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	1	1	9

Note: Other values are reserved. Also notice that UCSZ0 and UCSZ1 belong to UCSRC and UCSZ2 belongs to UCSRB

USART baud rate register (UBRR)



- UBRR
- 12 bits
- The most significant byte has a shared address with UCSRC!
  - The clock generated by the UART baud rate generator is 16 times higher than the baud rate we want to use for transferring data



Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

USART baud rate

Example 11-1

With Fosc = 8 MHz, find the UBRR value needed to have the following baud rates:
(a) 9600 (b) 4800 (c) 2400 (d) 1200

Solution:

Fosc = 8 MHz => X = (8 MHz/16(Desired Baud Rate)) – 1
=> X = (500 kHz/(Desired Baud Rate)) – 1

(a) (500 kHz/ 9600) – 1 = 52.08 – 1 = 51.08 = 51 = 33 (hex) is loaded into UBRR
(b) (500 kHz/ 4800) – 1 = 104.16 – 1 = 103.16 = 103 = 67 (hex) is loaded into UBRR
(c) (500 kHz/ 2400) – 1 = 208.33 – 1 = 207.33 = 207 = CF (hex) is loaded into UBRR
(d) (500 kHz/ 1200) – 1 = 416.66 – 1 = 415.66 = 415 = 19F (hex) is loaded into UBRR

USART usage algorithm sample in Tx mode

1. The UCSRB register is loaded with the value 08H, enabling the USART transmitter. The transmitter will override normal port operation for the TxD pin when enabled.
2. The UCSRC register is loaded with the value 06H, indicating asynchronous mode with 8-bit data frame, no parity, and one stop bit.
3. The UBRR is loaded with one of the values in Table 11-4 (if Fosc = 8 MHz) to set the baud rate for serial data transfer.
4. The character byte to be transmitted serially is written into the UDR register.
5. Monitor the UDRE bit of the UCSRA register to make sure UDR is ready for the next byte.
6. To transmit the next character, go to Step 4.

UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

USART usage algorithm sample in Rx mode

1. The UCSRB register is loaded with the value 10H, enabling the USART receiver. The receiver will override normal port operation for the RxD pin when enabled.
2. The UCSRC register is loaded with the value 06H, indicating asynchronous mode with 8-bit data frame, no parity, and one stop bit.
3. The UBRR is loaded with one of the values in Table 11-4 (if Fosc = 8 MHz) to set the baud rate for serial data transfer.
4. The RXC flag bit of the UCSRA register is monitored for a HIGH to see if an entire character has been received yet.
5. When RXC is raised, the UDR register has the byte. Its contents are moved into a safe place.
6. To receive the next character, go to Step 5.

Example 11-6 shows the coding of the above steps.

UCSRA

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

UCSRB

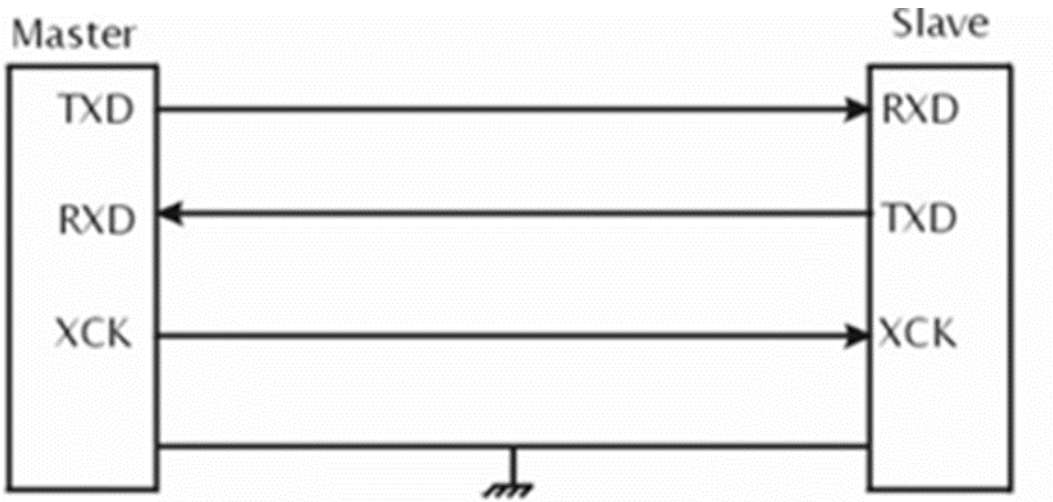
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

UCSRC

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

Synchronous mode

- Work in master-slave mode
- The master send clock signal to slave



USART programming in C

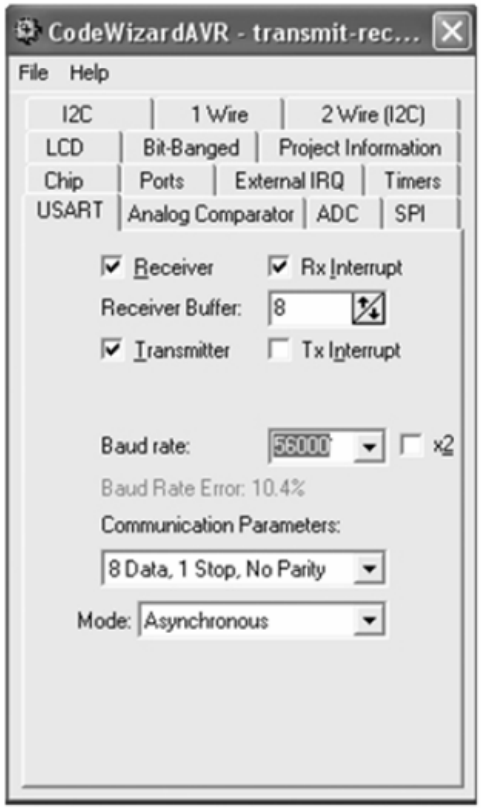
- Send a sequence of numbers started from 0 every 350ms to TXD pin
- Check RXD pin and if the received number is
  - 0x55 set PD.6 (bit 6 of port D) to 1
  - 0x66 set PD.6 to 0
- 1 stop bit, no parity, 8 bit, asynchronous

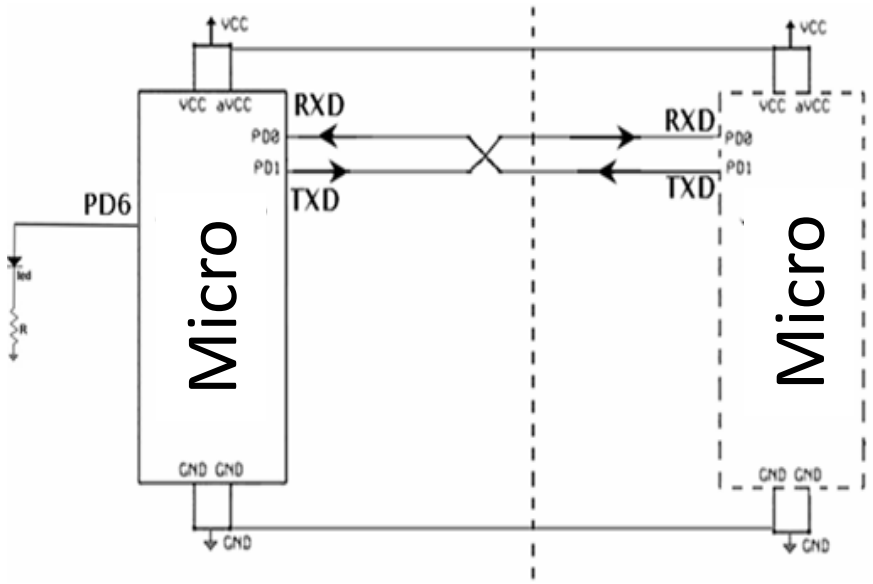
UCSZ2	UCSZ1	UCSZ0	Size
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	1	1	9

UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

USART programming in C

- Code vision configuration





USART programming in C

```
main()
{
     int a=0;
    DDRD.6=1;
    UCSRA=0x0;
    UCSRB=0x98; //10011000 (RXIE=1, RXEN=1, TXEN=1)
    UCSRC=0x86;// 10000110 (URSEL=1,asynch, no parity, one stop bit, 8 bit)
    UBRRH=0;// just set a rate that guarantees the data transfer can be completed before 350ms
    UBRRL=0x08;
    #asm("sei");
    while(1)
    {
            UDR= a++;
            delay_ms(350);
    }
}
```

```
Interrupt [USART_RXC] usart_rx_isr()
{
            char data;
            data=UDR;
            if(data==0x55)
                        PORTD.6=1;
             if(data==0x66)
                        PORTD.6=0;
}
```

Other peripherals in AVR microcontrollers

- PWM generation
- Sleep mode
- Watchdog timer
- EEPROM
- Analog comparator

PWM generation

- Control the power of DC (Direct Current) motors

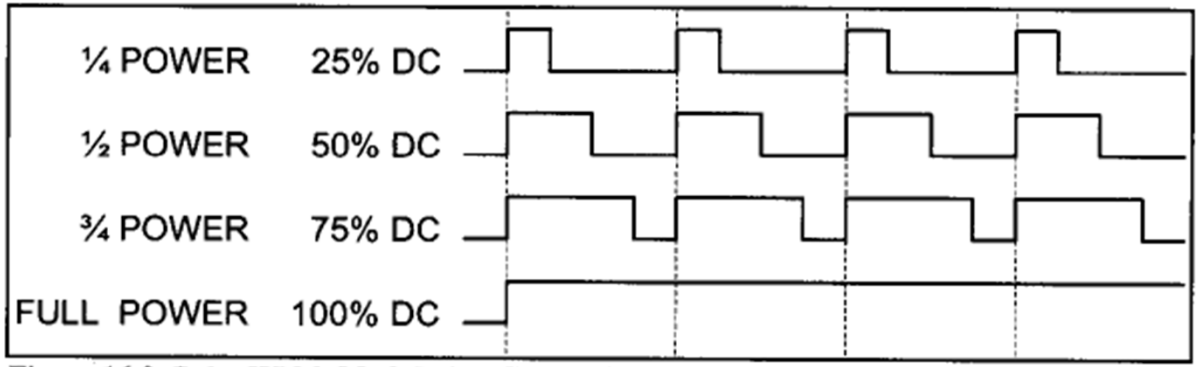


Figure 16-9. Pulse Width Modulation Comparisons

Write a program to generate ¾ power PWM signal

Any volunteer?

PWM generation

- Control the power of DC (Direct Current) motors

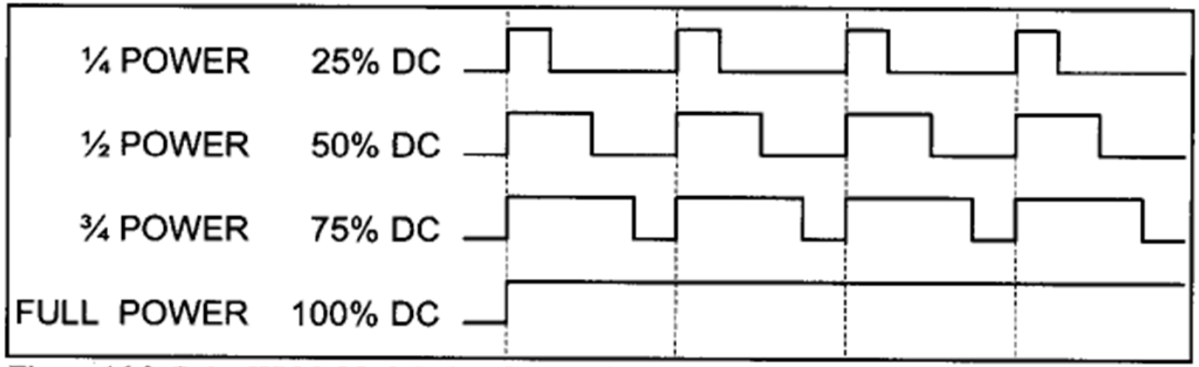


Figure 16-9. Pulse Width Modulation Comparisons

```
While(1)
{
  PORTB.1=1;
  Delay_ms(75);
  PORTB.1=0;
  Delay_ms(25);
}
```

Sleep mode

- To save power when the microcontroller is idle
- Controlled by MCUCR (previously visited at interrupt setting section!)
- The clock is shut-down, CPU is inactive
- Registers and SRAM keep their value
- Wakeup on reset and interrupts

Sleep mode

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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	

- SE: enables sleep mode
- SM0 to SM2 determine the sleep type
- After setting SE to 1, the “sleep” assembly command should be executed
  - To make microcontroller enter the sleep mode
  - Use sleep.h functions in C to enter sleep mode
  - CPU exits the sleep mode by an external interrupt

Sleep mode

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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	

Table 13. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	Extended Standby(1)

Note: 1. Standby mode and Extended Standby mode are only available with external crystals or resonators.

Table 14. Active Clock Domains and Wake Up Sources in the Different Sleep Modes

	Active Clock domains					Oscillators		Wake-up Sources					
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT2 INT1 INT0	TWI Address Match	Timer 2	SPM / EEPROM Ready	ADC	Other I/O
Idle			X	X	X	X	X(2)	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	
Power-down								X(3)	X				
Power-save					X(2)		X(2)	X(3)	X	X(2)			
Standby(1)						X		X(3)	X				
Extended Standby(1)					X(2)	X	X(2)	X(3)	X	X(2)			

Notes: 1. External Crystal or resonator selected as clock source.
2. If AS2 bit in ASSR is set.
3. Only INT2 or level interrupt INT1 and INT0.

Sleep Mode

- Two important sleep states
  - Idle state
    - The clock of CPU and Flash (instruction memory) is disconnected
      - Other parts have their clock active
- Power down state
  - All clocks are deactivated
  - Registers and PCs are stored
- Exit by interrupts

Table 14. Active Clock Domains and Wake Up Sources in the Different Sleep Modes

	Active Clock domains					Oscillators		Wake-up Sources					
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT2 INT1 INT0	TWI Address Match	Timer 2	SPM / EEPROM Ready	ADC	Other I/O
Idle			X	X	X	X	X(2)	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	
Power-down								X(3)	X				
Power-save					X(2)		X(2)	X(3)	X	X(2)			
Standby(1)						X		X(3)	X				
Extended Standby(1)					X(2)	X	X(2)	X(3)	X	X(2)			

Notes: 1. External Crystal or resonator selected as clock source.
2. If AS2 bit in ASSR is set.
3. Only INT2 or level interrupt INT1 and INT0.

Watchdog timer (up to/from here for session 29/30)

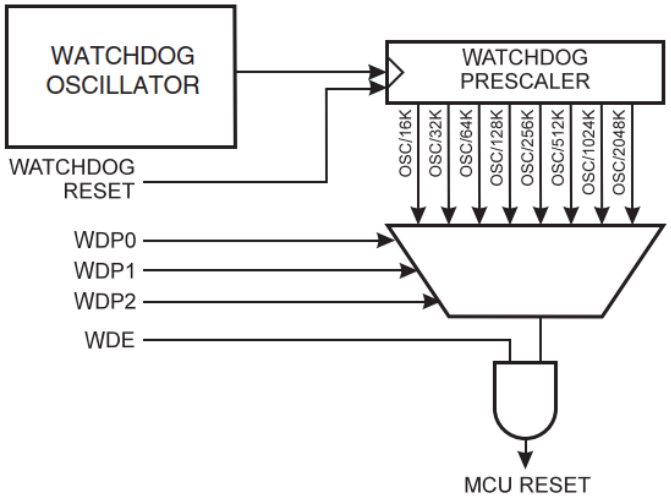
- A fault-tolerant approach
- Resets the microcontroller if detects a crash!
- To ensure that the microcontroller is functional and doesn't hang!
  - Start watchdog timer and set its registers to reset the micro after a time
  - In program disable the watchdog timer before that time
  - If the micro is frozen, the code that disables the watchdog is not executed and it resets the micro
  - Otherwise the watchdog resets the micro

Watchdog timer

- Watchdog uses an independent clock (1 MHz)
- WDTCR register controls the watchdog

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- WDTOE
  - Watchdog turn-off enable
    - Must be set when WDE bit is written 0



Watchdog timer

- WDE
  - Enable timer
- WPD[0-2]
  - The reset time

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17. Watchdog Timer Prescale Select

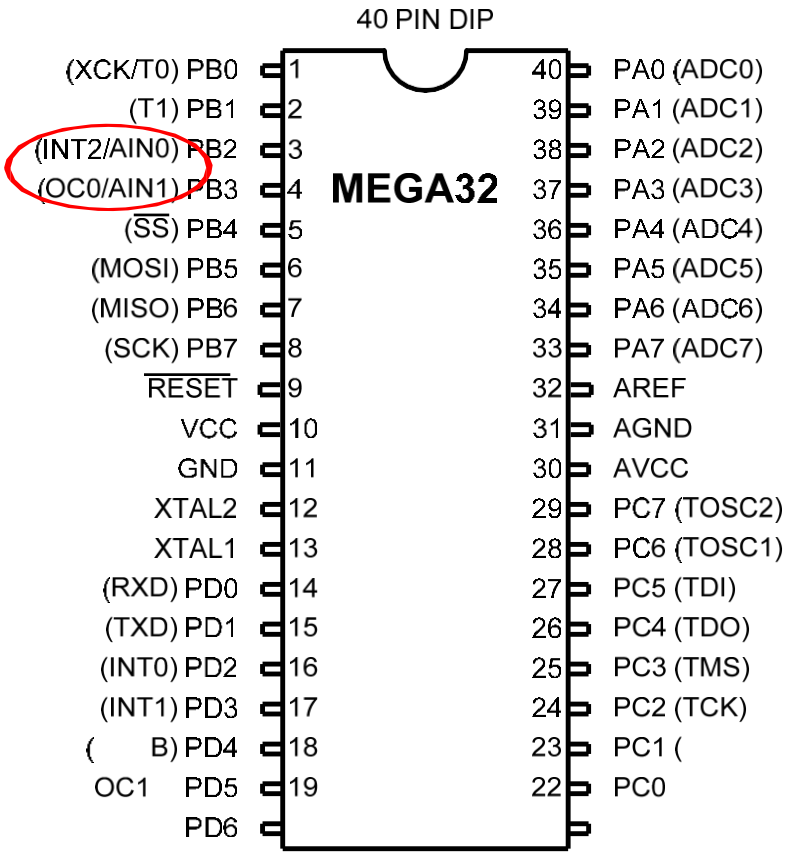
WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1ms	16.3ms
0	0	1	32K (32,768)	34.3ms	32.5ms
0	1	0	64K (65,536)	68.5ms	65ms
0	1	1	128K (131,072)	0.14s	0.13 s
1	0	0	256K (262,144)	0.27s	0.26s
1	0	1	512K (524,288)	0.55s	0.52s
1	1	0	1,024K (1,048,576)	1.1s	1.0s
1	1	1	2,048K (2,097,152)	2.2s	2.1s

EEPROM

- Kind of data memory
- To keep permanent data not changed during time
- Using some registers to communicate with EEPROM
- Two registers to send and receive data
- A register to control read and write

Analog comparator

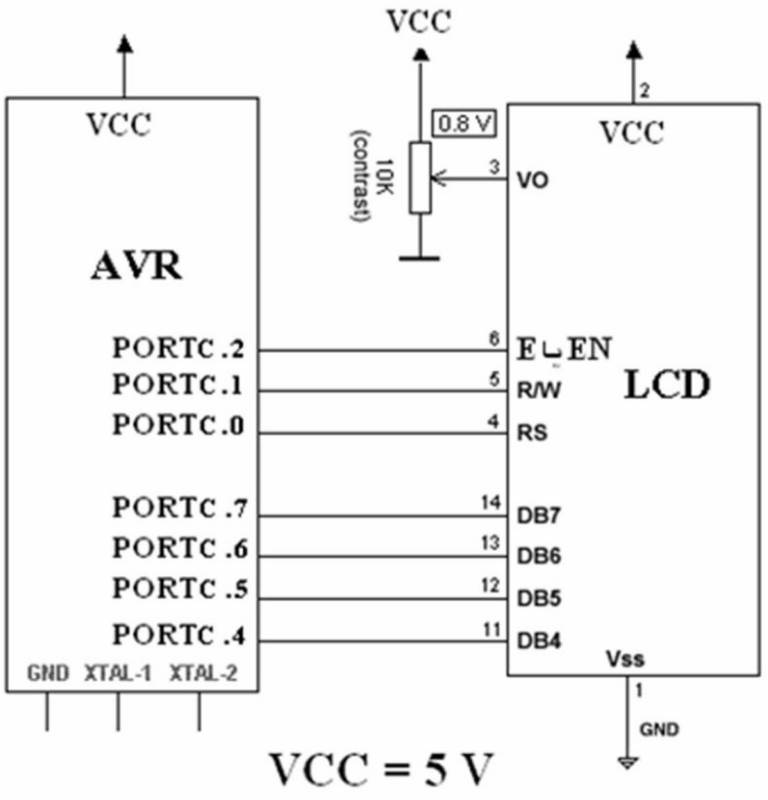
- Compares two analog inputs on AIN0 and AIN1 pins
- Sets the bits of a register based on the results to be used by the micro



LCD

- A device that displays characters
- Many variations
- We study a very common 14-pin LCD

LCD to AVR connections



LCD to AVR connections

- LCD has two important registers
  - Data register
    - Saves the data to be shown
  - Instruction register
    - Commands received to do something
    - Example
      - Clear display

Pin no.	Symbol	External connection	Function
1	Vss	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL

LCD commands

- Example:
  - Initialize lcd for 8 bit and 5x7
  - Commands 38, 0E, and 01 should be sent
- To send a command:
  - RS and W/R set to 0
  - Send appropriate command on data pins

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
28	2 lines and 5 × 7 matrix (D4–D7, 4-bit)
38	2 lines and 5 × 7 matrix (D0–D7, 8-bit)

Character addresses

LCD Type	Line	Address Range				
16 × 2 LCD	Line 1:	80	81	82	83	through 8F
	Line 2:	C0	C1	C2	C3	through CF
20 × 1 LCD	Line 1:	80	81	82	83	through 93
20 × 2 LCD	Line 1:	80	81	82	83	through 93
	Line 2:	C0	C1	C2	C3	through D3
20 × 4 LCD	Line 1:	80	81	82	83	through 93
	Line 2:	C0	C1	C2	C3	through D3
	Line 3:	94	95	96	97	through A7
	Line 4:	D4	D5	D6	D7	through E7
40 × 2 LCD	Line 1:	80	81	82	83	through A7
	Line 2:	C0	C1	C2	C3	through E7

LCD programming in CodeVision

```
int main(void)
{
        lcd_init();
        lcd_gotoxy(1,1);
        lcd_print("The world is but");
        lcd_gotoxy(1,2);
        lcd_print("one country");

        while(1);                    //stay here forever
        return 0;
}
```


The End