AVR Interfacing

ADC

Agenda

- Introduction to ADC.
- AVR ADC.
- AVR ADC Registers.
- AVR ADC Programming.

Introduction to ADC

- In the Real World, a **sensor** senses any physical parameter and **converts** into an equivalent analog electrical signal this analog signal is converted into a digital signal using an **Analog to Digital Converter** (ADC).

- ADC can be defined by
  - The ADC **precision = No. of levels** is the number of distinguishable ADC inputs that ADC can measure (e.g. 1024 alternatives for 10 bits ADC).
  - The ADC **range** is the maximum and minimum ADC input (e.g. 0 to Vref).
  - he ADC **resolution** is the smallest distinguishable change in input voltage that can be sensed by ADC. The resolution is the change in input that causes the digital output to change by 1.

    ex:

$$\text{Resolution(volts)} = \frac{\text{Range(volts)}}{\text{Precision(alternatives)}} = \frac{5V-0}{1024} = 4.88\text{mv}$$

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

Input Value Read by Arduino (0-1023)

Introduction to ADC

- **ADC Prescaler**
- The ADC of the AVR **converts analog signal** into digital signal at some regular interval. This interval is determined by **the clock frequency.**
- The prescaler acts as frequency **division factor.**
- There are some predefined division factors $\mathbf{2^1}$ to $\mathbf{2^7}(\boldsymbol{default})$
- For example, a prescaler of 128 implies
  F_ADC = F_CPU/8.
  Thus, for F_CPU = 16MHz, F_ADC = 16M/128 = 125kHz.

So time required to convert the analog signal(sample*) to digital is 1/125 ms.

//Above is what was in the original slides, in practice its known that the
//**10-bit SAR ADC** of AVR takes **13 clock cycles** to do one conversion.

AVR ADC Registers

- **ADMUX - ADC Multiplexer Selection Register**

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

➢ **Bits 7:6 – REFS1:0 – ADC V_{ref}** Reference Selection Bits (last two bits)

These bits select the voltage reference for the ADC. The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin (Otherwise it's 5V)

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal V_{REF} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

AVR ADC Registers

- **Bit 5 – ADLAR – ADC Left Adjust Result** [MSB First or LSB first]
  The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted.

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL

- **Bits 3:0 – MUX4:0 – Analog Channel Bits**
  There are 6 ADC channels (PC0…PC5). They can work simultaneously*.

AVR ADC Registers

- **ADCSRA – ADC Control and Status Register A**

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

- **Bit 7 – ADEN – ADC Enable**
  This is enabled, ADC operations. Otherwise the pins behave as GPIO ports.
- **Bit 6 – ADSC – ADC Start Conversion**
  Write 1 to start conversion it stays at 1 and when the conversion is complete, it returns to zero.
- **Bit 5 – ADATE – ADC Auto Trigger Enable**
  1 enables auto trigger where the ADC will start a conversion on a positive edge of the selected trigger signal.
- **Bit 4 – ADIF – ADC Interrupt Flag**
  This bit is set when an **ADC conversion completes** and the **Data Registers are updated.** **(Instead of checking ADSC)**

AVR ADC Registers

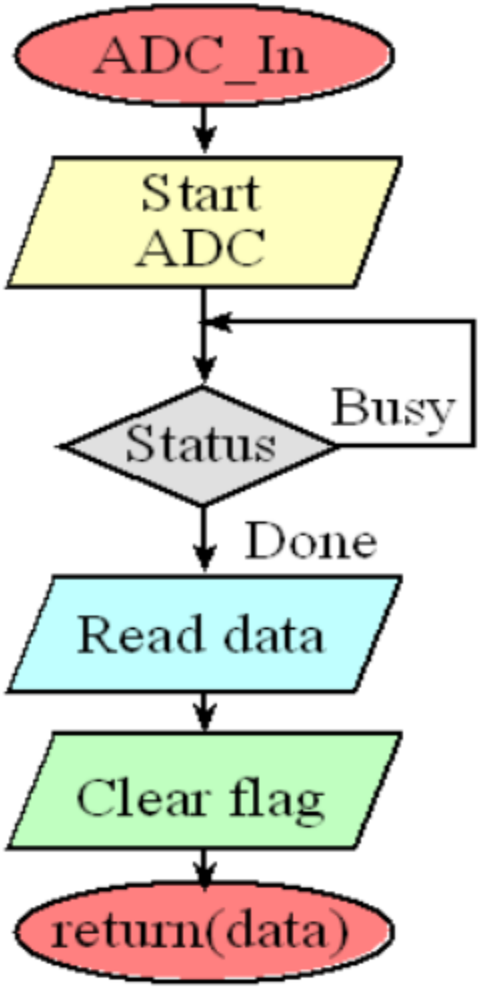
- **Bit 3 – ADIE – ADC Interrupt Enable**
- **Bits 2:0 – ADPS2:0 – ADC Pre-scaler Select Bits**

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

- **ADCL and ADCH – ADC Data Registers**

  The result of the ADC conversion is stored here (data registers).

AVR ADC Programming



AVR ADC Programming

ADC Initialization

The following code segment initializes the ADC

```
void adc_init()
{
    // AREF = AVcc
    ADMUX = (1<<REFS0);

    // ADC Enable and prescaler of 128
    // 16000000/128 = 125000
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```

AVR ADC Programming

Reading ADC Value

The following code segment reads the value of the ADC

```
uint16_t adc_read(uint8_t ch)
{
  // select the corresponding channel 0~5
  // ANDing with '7' will always keep the value
  // of 'ch' between 0 and 5
  ch &= 0b00000111;  // AND operation with 7
  ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing

  // start single conversion
  // write '1' to ADSC
  ADCSRA |= (1<<ADSC);

  // wait for conversion to complete
  // ADSC becomes '0' again
  // till then, run loop continuously
  while(ADCSRA & (1<<ADSC));

  return (ADC);
}
```

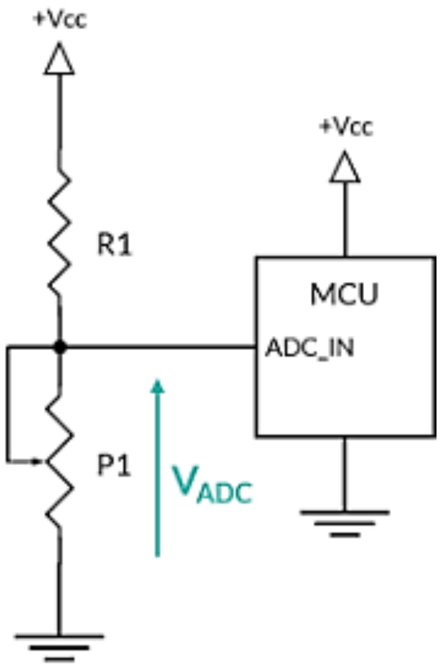
AVR ADC Programming

The following circuit describes the connection of potentiometer.
The following code reads ADC value and checks if the reading is above 500, the led will be turned on. Otherwise, the led will be turned off.

```
void main()
{
    uint16_t adc_result0;
    DDRB = 0x20;            // to connect led to PB5

    // initialize adc
    adc_init();
    while(1)
    {
        adc_result0 = adc_read(0);  // read adc value at PC0

        // condition for led to turn on or off
        if (adc_result0 > 500)
            PORTB = 0x20;
        else
            PORTB = 0x00;
    }
}
```



AVR ADC Programming

Using ADC interrupt:

- Replacing conversion waiting to be completed with firing a flag or signal that called “ADC interrupt”.
- The main difference that will happen in the previous code to be adopted with the ADC interrupt, will be as following;
  - **In ADC Initialization, global interrupt and ADC interrupt enable have to be set.**
  - **In Reading ADC Value, while loop which waiting for the conversion ending will be remove and global flag will checked if it comes high or not**

    In case of the flag comes high the ADC read will be ready to be returned from the function.
  - This flag is controlled in the interrupt function (ISR).

AVR ADC Programming

ADC Initialization

The following code segment initializes the ADC

```
void adc_init()
{
    // AREF = AVcc
    ADMUX = (1<<REFS0);

    // ADC Enable and prescaler of 128
    // 16000000/128 = 125000
    // (1<<ADIE)=1 → set ADC interrupt enable
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
             |(1<<ADIE);
    // Set global interrupt
    sei();
}
```

AVR ADC Programming

Reading ADC Value

```
uint8_t ADC_endconversion_Flag=0;
int16_t adc_read(uint8_t ch)
{
  // select the corresponding channel 0~5
  // ANDing with '7' will always keep the value
  // of 'ch' between 0 and 5
  ch &= 0b00000111;  // AND operation with 7
  ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing
  // start single conversion
  // write '1' to ADSC
  ADCSRA |= (1<<ADSC);

  // wait for conversion to complete
  // ADSC becomes '0' again
  // till then, run loop continuously
  if(ADC_endconversion_Flag==1){
        ADC_endconversion_Flag=0;
         return (ADC);
  }
  else
        return(-1);
}
```

```
ISR(ADC_vect)
{
   ADC_endconversion_Flag=1;

}
```

AVR ADC Programming

Main function:

```
void main()
{
    uint16_t adc_result0;
    DDRB = 0x20;              // to connect led to PB5

    // initialize adc
    adc_init();
    while(1)
    {
        adc_result0 = adc_read(0);  // read adc value at PA0

        // condition for led to turn on or off
        if(adc_result0!=-1){
            if (adc_result0 > 500)
                PORTB = 0x20;
            else if ()
                PORTB = 0x00;
        }
    }
}
```