THAPAR UNIVERSITY STUDENT SATELLITE INITIATIVE

REPORT ON

ANALOG TO DIGITAL CONVERTORS

SUBMITTED BY:

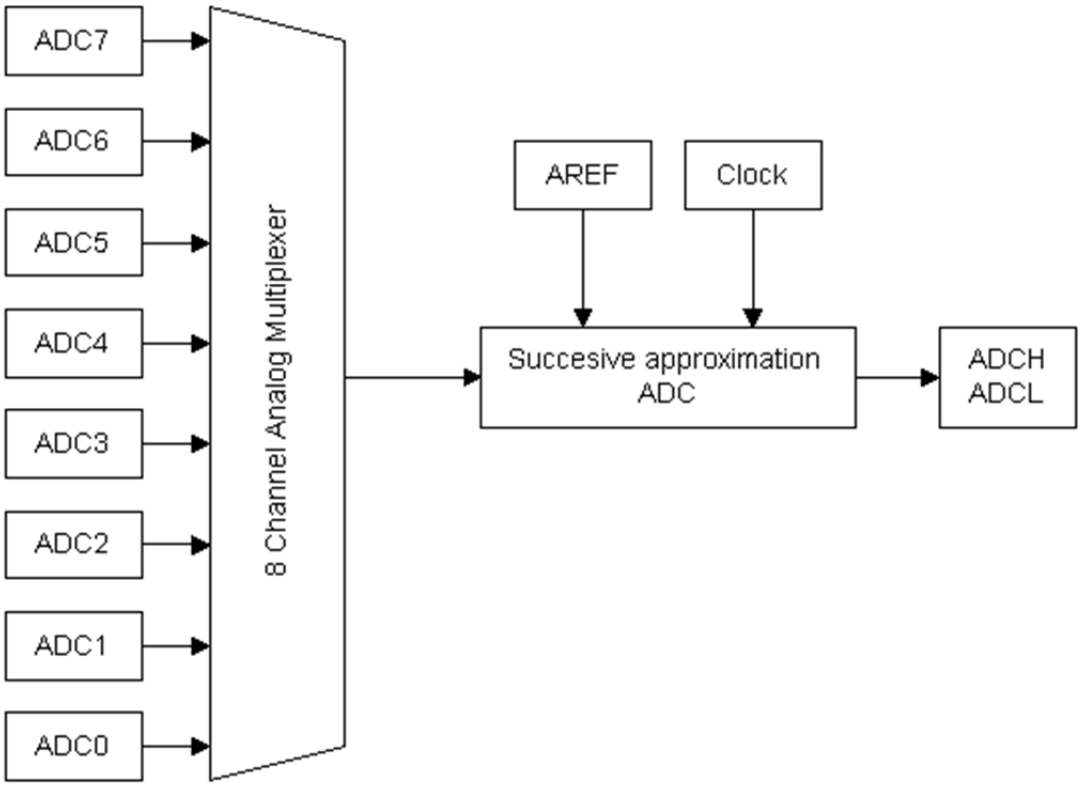
GURSIMRAN SINGH

KESHAV SODHI

ABOUT AVR ADC

CONNECTED TO **PORT A (PA0-PA7) 33-44**

For instance, it can be used to log the output of a sensor (temperature, pressure, etc) at regular intervals(SAMPLING), or to take some action in function of the measured variable value. There are several types of ADCs. The one used by AVR is of the "**succesive approximation ADC**" kind. The following is a simplified scheme of the ADC.



At the input of the ADC itself is an analog multiplexer, which is used to select between eight analog inputs. That means that you can convert up to eight signals (not at the same time of course). At the end of the conversion, the corresponding value is transferred to the registers ADCH and ADCL. As the AVR's registers are 8-bit wide, the 10-bit value can only be held in two registers.

ABOUT THE PROJECT

This ADC takes data from ADC0 pin(PA0,40) and converts the IR sensor's analog data to 10-bit digital on its own, in free conversing mode, which we use in our program in a loop(POLLING) but for the requirement of the team we have converted the 10-bit data to 6 bit data by using the most significant bits and left adjusting the result in ADCH data register.

The output of this 6-bit data is shown in the PORT-C which is connected to 6 LEDs to verify the result in embedded systems. The first two bits (0 n 1) are controlled by the external interrupt 0 of the ADC, pin (PD2 , 16) of the PDIP.

Interrupts, as the name suggests, interrupt the normal program flow. When an interrupt occurs, the ALU calls the **corresponding interrupt vector** and executes the code at that address. As the interrupt vectors each are only one word long (classics AVR, two words for some megas), you'd usually put a jump instruction there which goes to an **Interrupt Service Routine**.

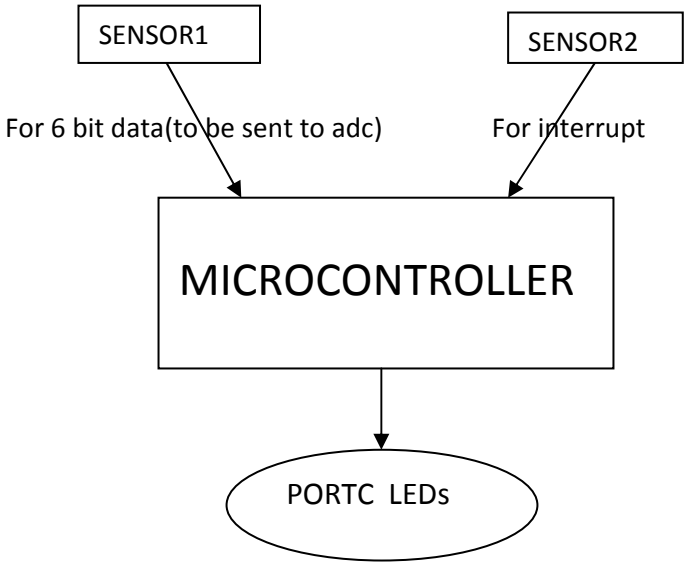
The choice of ADC0 is immaterial any channel of the multiplexer can be used with no major change in coding. Only the corresponding change has to made in ADMUX register of the ADC.

Different interrupts vectors that could be used for the achievement of the goal

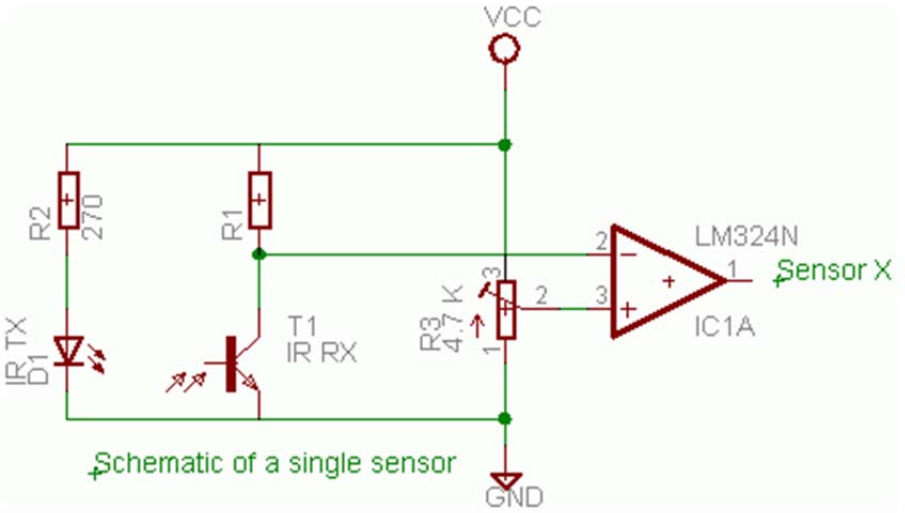
VECTOR NO	PROGRAM ADDRESS	SOURCE	INTERRUPT DEFINITION
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
19	$024	INT2	External Interrupt Request 2
15	$01C	ADC	ADC Conversion Complete

We have chosen INT0 external interrupt in our program and in our report. Other possibilities can be set as a task for further study in this topic. This program takes data from the sensor and changes it into 6 bit data.

PROJECT OVERVIEW



CIRCUIT DIAGRAM for an IR Sensor


VCC
R2
270
R1
LM324N
Sensor X
T1
IR RX
R3
4.7 K
IR TX
D1
IC1A
Schematic of a single sensor
GND

PROGRAM IN AVR STUDIO

```
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include<compat/deprecated.h>

ISR(INT0_vect )          //This ISR called when for external interrupt  request0 called in IVT.
{
        sbi(PORTC,PIN1);
        sbi(PORTC,PIN0);

        _delay_ms(100);

        cbi(PORTC,PIN1);
        cbi(PORTC,PIN0);
}

void main(void)
{
        char i;                 //int could also be used but char takes one byte data

        //DATA DIRECTION REGISTER FOR PORT C.
        DDRC=0xFF;              //PortC is output port(ALL BITS SET IN DDR)

        //GENERAL INTERRUPT CONTROL REGISTER.
        GICR=0X01000000;    //Internal interrupt 0 request enable

        //MCU CONTROL REGISTER
        MCUCR=0X01;         //Any logical cange in value at int0 generates an interrupt

        sei();    //Enable all global interrupts.SETS GLOBAL INTERRUPT ENABLE BIT IN SREG

        //ADC MULTIPLEXER SELECT
        ADMUX=0b00100000;
                /*Setting result to left adjust
                Reference selected to AREF that we provide 4V
                Selecting Pin0 as input for sensor1 using mux pins*/

        //SPECIAL FUNCTION IO REGISTER
        SFIOR=0x00;             //Setting auto trigger source to free running mode

        //ADC CONTROL AND STATUS REGISTER
        ADCSRA=0b11100101;
                /*ADC enabled
                ADC conversion started
                ADTS enabled*/

        while(1)
        {
                i=ADCH;         //ADCH stores value of converted data

                i=i>>2;         //DUMP THE LAST TWO VALUES OF ADC
                i=i<<2;         //LEFT ADJUST THE VALUES

                PORTC=i;
                //whatever data we get is displayed in the six led. Last 2(0,1) will be always off.
        }
}
```

DETAILED DESCRIPTION OF REGISTERS INVOLVED

Code
DDRC=0xFF; //PortC is output port(ALL BITS SET IN DDR)

Port C Data Direction Register – DDRC

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
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	

Code
GICR=0X01000000; //Internal interrupt 0 request enable

General Interrupt Control Register – GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – INT1: External Interrupt Request 1 Enable**

When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the MCU General Control Register (MCUCR) define whether the External Interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 interrupt Vector.

- **Bit 6 – INT0: External Interrupt Request 0 Enable**

When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the MCU General Control Register (MCUCR) define whether the External Interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 interrupt vector.

- **Bit 5 – INT2: External Interrupt Request 2 Enable**

When the INT2 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control2 bit (ISC2) in the MCU Control and Status Register (MCUCSR) defines whether the External Interrupt is activated on rising or falling edge of the INT2 pin. Activity on the pin will cause an interrupt request even if INT2 is configured as an output. The corresponding interrupt of External Interrupt Request 2 is executed from the INT2 Interrupt Vector.

Code

```
MCUCR=0X01;        //Any logical cange in value at int0 generates an interrupt
```

MCU Control Register – MCUCR

The MCU Control Register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	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	

- **Bits 7, 5, 4 – SM2..0: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the six available sleep modes as shown in Table 13.

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.

- **Bit 6 – SE: Sleep Enable**

The SE bit must be written to logic one to make the MCU enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering the sleep mode unless it is the programmers purpose, it is recommended to write the Sleep Enable (SE) bit to one just before the execution of the SLEEP instruction and to clear it immediately after waking up.

Code

```
ADMUX=0b00100000;
```

ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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 7:6 – REFS1:0: Reference Selection Bits

These bits select the voltage reference for the ADC. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 84 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

Table 84. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			

Code

```
SFIOR=0x00;              //Setting auto trigger source to free running mode
```

SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
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	

- **Bit 7:5 – ADTS2:0: ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS2:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected Interrupt Flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

Table 86. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Code
ADCSRA=0b11100101;

ADCSRA

Bit	7	6	5	4	3	2	1	0	
	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**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running Mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR.

ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

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

ADCH and ADCL data registers

The ADC Data Register – ADCL and ADCH

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Code

```
i=i>>2;          //DUMP THE LAST TWO VALUES OF ADC
i=i<<2;          //LEFT ADJUST THE VALUES
```

eg
if char i is 11111111
i>>2 is 00111111
i<<2 is 11111100

We therefore get 8 bit data by reading ADCH and then we convert it into 6 bit.
And the output is displayed to last 6 LED's (PIN2-PIN7) of PORTC.

Program part for sensor 2

This program is used to interrupt the normal running of the program and is used to glow the last 2 LED's.

OUTPUT EXPLAINED

The reference voltage is the maximum value that the ADC can convert. Our example 8-bit ADC can convert values from 0V to the reference voltage.

In our project we have chosen reference voltage as 5V. *This is an assumption so the IR sensor must give voltage from 0-5V as its output for our requirement .* This voltage range is divided into 1024 (2 pow 10) values, or steps. The size of the step is given by:

V_{ref}/1024

Where V_{ref} is the reference voltage. The step size of the converter defines the converter's resolution. For a 5V reference, the step size is:

5V/1024 = 0.00488V or 4.88mV (our ADC)

But for the requirement of the team we have truncated 10-bit data to 6-bit data so the max resolution or max change in voltage produced by sensor to make a change in LEDs connected to PORT-C is

5V/1024 = 0.078125V or 78mV (our team requirement)

VOLTAGE RANGE	DECIMAL EQUIVALENT	LED 7 PC7,29 PIN	LED6 PC6,28 PIN	LED5	LED4	LED3	LED2
0-0.078	0	0	0	0	0	0	0
0.078-0.156	1	0	0	0	0	0	1
0.156-0.234	2	0	0	0	0	1	0
0.234-0.312	3	0	0	0	0	1	1
0.312-0.39	4	0	0	0	1	0	0
0.39-0.468	5	0	0	0	1	0	1
0.468-0.546	6	0	0	0	1	1	0
So on 64 items….	7,8…63						

TRUNCATION EXPLAINED

		2.5v	1.25v	625	312.5	156	78.12	39	19.5	9.76	4.88mv
	1024	512	256	128	64	32	16	8	4	2	1
VOL	DECIMAL	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	1
	2	0	0	0	0	0	0	0	0	1	0
	3	0	0	0	0	0	0	0	0	1	1
	4	0	0	0	0	0	0	0	1	0	0
	5	0	0	0	0	0	0	0	1	0	1
	6	0	0	0	0	0	0	0	1	1	0
	7	0	0	0	0	0	0	0	1	1	1
	1023	1	1	1	1	1	1	1	1	1	1

Now we have truncated our result form 10-bit to 8-bit for simplicity. We observe that there is no change in the bit3 from decimal 0-3. So we consider this range as one and so neglect the last two bits and count this decimal one in 8 bit system and same result is produced for 0 to 19.5mv ie decimal 0 and again from 4 to 7 bit3 is unchanged with bit0 and bit1 changing values so neglect them and count result from 4-7 as one ie equivalent to decimal 2 in our new truncated system.

So our new truncated system counts only change of 19.5mv so treats 0 and 4.88mv as same giving the same bit word of 0b00000000(truncated result of actual) while the original 10-bit system distinguishes between 0 and 4.88 mv.

EXAMPLE WITH FUTHER EXPLANATION OF HASH FUNCTION(say) AS ABOVE

Take an example of 8-bit converter for simplicity represents the analog input as a digital word. The most significant bit of this word indicates whether the input voltage is greater than half the reference (2.5V, with a 5V reference). Each succeeding bit represents half the range of the previous bit.

Table 1 Example conversion, on an 8-bit ADC ,say

Bit:	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Volts:	2.5	1.25	0.625	0.3125	0.156	0.078	0.039	0.0195
Output Value:	0	0	1	0	1	1	0	0

Table 1 illustrates this point. Adding the voltages corresponding to each set bit in 0010 1100, we get:

.625 + .156 + .078 = .859 volts

The resolution of an ADC is determined by the reference input and by the word width. The resolution defines the smallest voltage change that can be measured by the ADC. As mentioned earlier, the resolution is the same as the smallest step size, and can be calculated by dividing the reference voltage by the number of possible conversion values.

For the example we've been using so far, the resolution is 19.5mV in 8 bit example. This means that any input voltage below 19.5mV will result in an output of 0. Input voltages between 19.5mV and 39mV will result in an output of 1. Between 39mV and 58.6mV, the output will be 2.

Resolution can be improved by reducing the reference input. Changing that from 5V to 2.5V gives a resolution of 2.5/256, or 9.7mV. However, the maximum voltage that can be measured is now 2.5V instead of 5V.

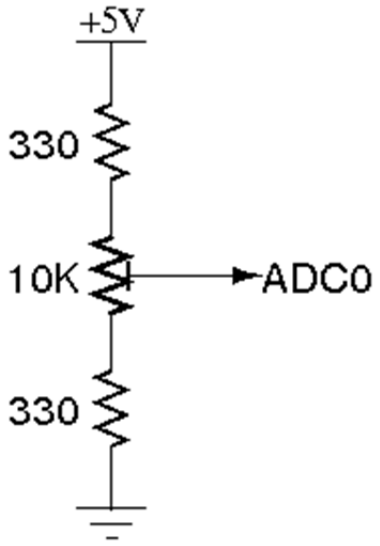
The only way to increase resolution without reducing the range is to use an ADC with more bits. A 10-bit ADC has 2^{10}, or 1,024 possible output codes. So the resolution is 5V/1,024, or 4.88mV; a 12-bit ADC has a 1.22mV resolution for this same reference.

Future goals/tasks regarding ADC

- Other possibilities of design with different type of interrupts could be implemented to get more comfortable with programming.
- ADC could be further studied in Assembly language programming which is crucial in limited memory environment as TUSSAT and providing fast real-time environment.
- Making our system as real time by knowing precisely the no of clock cycles required for each command and ADC conversions beforehand by changing the frequency acc to us by using the prescaler of ADC.
- Implementing both the free running mode and single conversion mode
- Understanding of noise and other material left with the ADC.

POSSIBLE TESTING THE PROGRAM BEFORE USING IR SENSORS

This example uses the simplest variable voltage source we could think of a potentiometer. This is a 10k potentiometer on a breadboard.



The two outside terminals were attached to 5 volts and ground, with the center terminal attached to the first ADC channel. The two 330 ohm resistors protect the microcontroller pin from being shorted to ground or to 5 volts at the edges of the potentiometer's travel.

With this setup, turning the potentiometer will give you a range of .15 volts to 4.85 volts between ground and ADC0. In order to read the voltage of this circuit, its ground and the ground of the microcontroller need to be connected.

As we donn't know the voltage output of the IR sensors with the light it receives like it might be linear or non linear. So like this we could never actually test the program. So we must use a potentiometer to test the LEDs glowing according to the analog voltage given to the ADC0 (0-5v).

TERMINOLOGY

RESOLUTION OF ADC

The resolution of the converter indicates the number of discrete values it can produce over the range of analog values. The values are usually stored electronically in binary form, so the resolution is usually expressed in bits. In consequence, the number of discrete values available, or "levels", is usually a power of two. For example, an ADC with a resolution of 8 bits can encode an analog input to one in 256 different levels, since $2^8 = 256$. The values can represent the ranges from 0 to 255 (i.e. unsigned integer) or from -128 to 127 (i.e. signed integer), depending on the application.

Resolution can also be defined electrically, and expressed in volts. The voltage resolution of an ADC is equal to its overall voltage measurement range divided by the number of discrete intervals as in the formula:

$$Q = \frac{E_{FSR}}{2^M} = \frac{E_{FSR}}{N}$$

Where:

Q is resolution in volts per step (volts per output code),

E_{FSR} is the full scale voltage range = $V_{RefHi} - V_{RefLow}$,

M is the ADC's resolution in bits.

N is the number of intervals, given by the number of available levels (output codes), which is: $N = 2^M$

Some examples may help:

Example 1

Full scale measurement range = 0 to 10 volts

ADC resolution is 12 bits: $2^{12} = 4096$ quantization levels (codes)

ADC voltage resolution is: (10V - 0V) / 4096 codes = 10V / 4096 codes ≈0.00244 V/code ≈2.44 mV/code

LINEAR ADCs

Non-linear ADCs

If the probability density function of a signal being digitized is uniform, then the signal-to-noise ratio relative to the quantization noise is the best possible. Because this is often not the case, it is usual to pass the signal through its cumulative distribution function (CDF) before the quantization. This is good because the regions that are more important get quantized with a better

resolution. In the dequantization process, the inverse CDF is needed.

SAMPLING RATE – no of samples taken in one sec.

POLLING

Polling is what you have to do if your microcontroller does not have interrupts or if what you want to do is not time critical. It is a software technique whereby the controller continually asks a peripheral if it needs servicing. The peripheral sets a flag when it has data ready for transferring to the controller, which the controller notices on its next poll. Several peripherals can be polled in succession, with the controller jumping to different software routines, depending on which flags have been set.

INTERRUPT

Rather than have the microcontroller continually polling - that is, asking peripherals (timers / UARTS / A/Ds / external components) whether they have any data available (and finding most of the time they do not), a more efficient method is to have the peripherals tell the controller when they have data ready. The controller can be carrying out its normal function, only responding to peripherals when there is data to respond to. On receipt of an interrupt, the controller suspends its current operation, identifies the interrupting peripheral, then jumps to the appropriate interrupt service routine. The advantage of interrupts, compared with polling, is the speed of response to external events and reduced software overhead (of continually asking peripherals if they have any data ready).

Figure 14. Polling versus Interrupt