به نام خدا

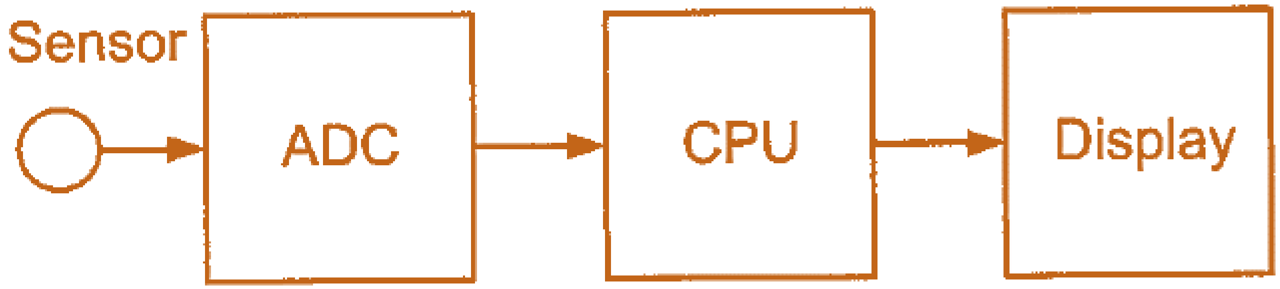
مبدل آنالوگ به دیجیتال

آشنایی و نحوه استفاده

Dr. Aref Karimiafshar
A.karimiafshar@ec.iut.ac.ir

ADC Devices

Analog-to-digital converters are among the most widely used devices for data acquisition. Digital computers use binary (discrete) values, but in the physical world everything is analog (continuous). Temperature, pressure (wind or liquid), humidity, and velocity are a few examples of physical quantities that we deal with every day. A physical quantity is converted to electrical (voltage, current) signals using a device called a *transducer*. Transducers are also referred to as *sensors*. Sensors for temperature, velocity, pressure, light, and many other natural quantities produce an output that is voltage (or current). Therefore, we need an analog-to-digital converter to translate the analog signals to digital numbers so that the microcontroller can read and process them.



Major Characteristic of ADC

Resolution

The ADC has *n*-bit resolution, where *n* can be 8, 10, 12, 16, or even 24 bits. Higher-resolution ADCs provide a smaller step size, where *step size* is the smallest change that can be discerned by an ADC. Some widely used resolutions for ADCs are shown in TablebelowAlthough the resolution of an ADC chip is decided at the time of its design and cannot be changed, we can control the step size with the help of what is called V_{ref}.

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

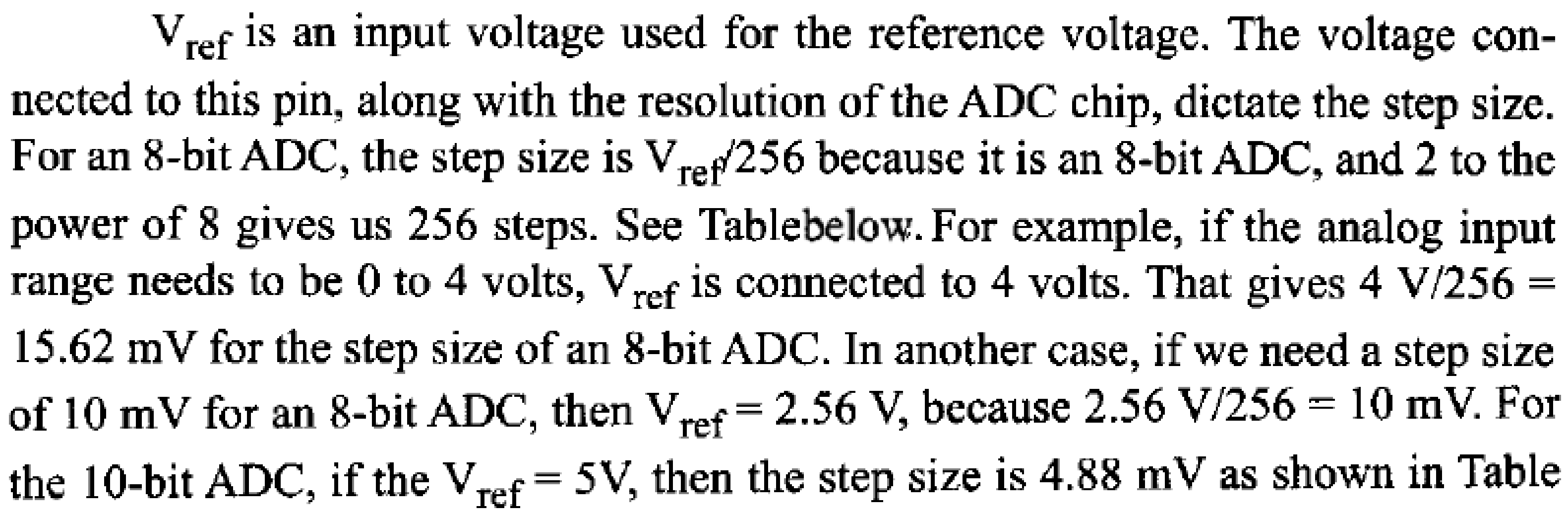
Notes: $V_{CC} = 5$ *V*

Conversion time

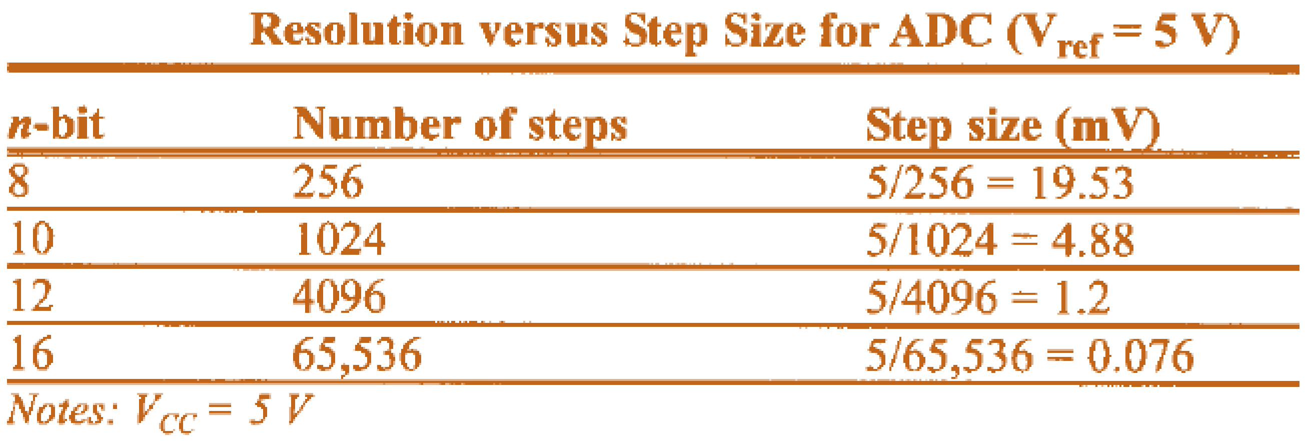
In addition to resolution, conversion time is another major factor in judging an ADC. *Conversion time* is defined as the time it takes the ADC to convert the analog input to a digital (binary) number. The conversion time is dictated by the clock source connected to the ADC in addition to the method used for data conversion and technology used in the fabrication of the ADC chip such as MOS or TTL technology.

Major Characteristic of ADC

V_{ref}



V_{ref} is an input voltage used for the reference voltage. The voltage connected to this pin, along with the resolution of the ADC chip, dictate the step size. For an 8-bit ADC, the step size is $V_{ref}/256$ because it is an 8-bit ADC, and 2 to the power of 8 gives us 256 steps. See Tablebelow. For example, if the analog input range needs to be 0 to 4 volts, V_{ref} is connected to 4 volts. That gives 4 V/256 = 15.62 mV for the step size of an 8-bit ADC. In another case, if we need a step size of 10 mV for an 8-bit ADC, then V_{ref} = 2.56 V, because 2.56 V/256 = 10 mV. For the 10-bit ADC, if the V_{ref} = 5V, then the step size is 4.88 mV as shown in Table

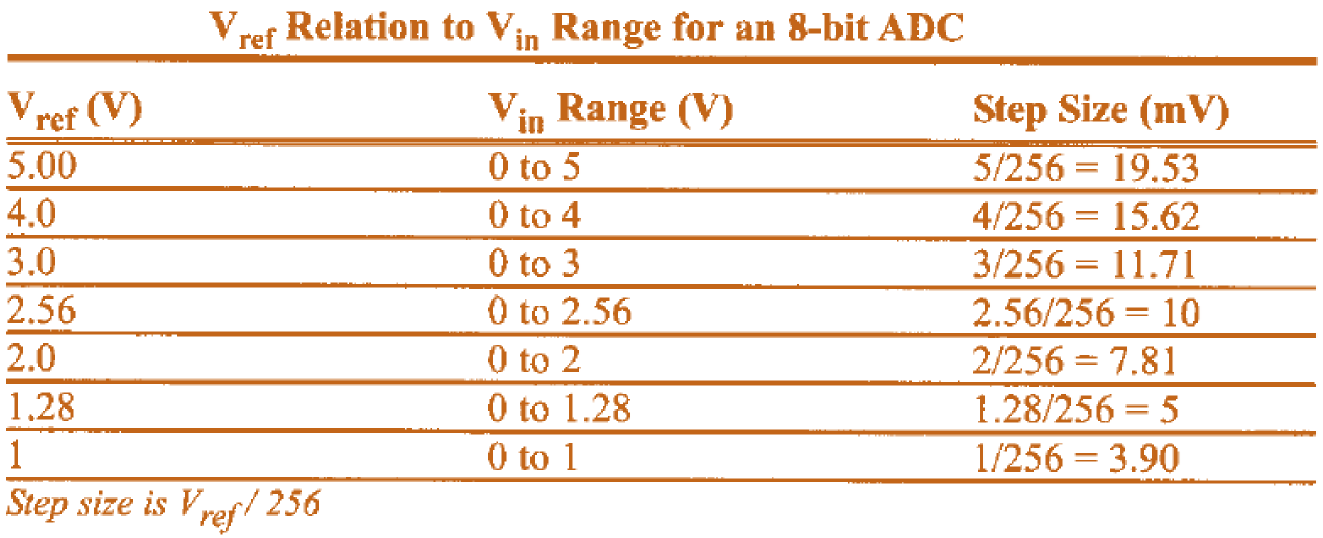


Resolution versus Step Size for ADC (V_{ref} = 5 V)

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

Notes: V_{CC} = 5 V

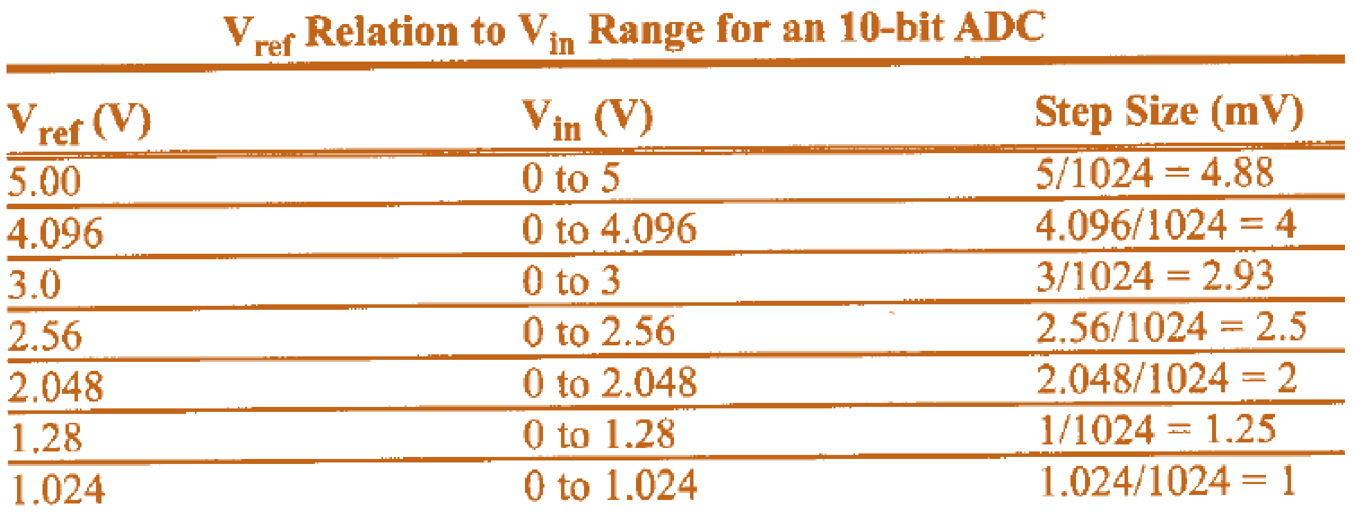
Major Characteristic of ADC



V_{ref} Relation to V_{in} Range for an 8-bit ADC

V_{ref} (V)	V_{in} Range (V)	Step Size (mV)
5.00	0 to 5	5/256 = 19.53
4.0	0 to 4	4/256 = 15.62
3.0	0 to 3	3/256 = 11.71
2.56	0 to 2.56	2.56/256 = 10
2.0	0 to 2	2/256 = 7.81
1.28	0 to 1.28	1.28/256 = 5
1	0 to 1	1/256 = 3.90

Step size is V_{ref} / 256



V_{ref} Relation to V_{in} Range for an 10-bit ADC

V_{ref} (V)	V_{in} (V)	Step Size (mV)
5.00	0 to 5	5/1024 = 4.88
4.096	0 to 4.096	4.096/1024 = 4
3.0	0 to 3	3/1024 = 2.93
2.56	0 to 2.56	2.56/1024 = 2.5
2.048	0 to 2.048	2.048/1024 = 2
1.28	0 to 1.28	1/1024 = 1.25
1.024	0 to 1.024	1.024/1024 = 1

Major Characteristic of ADC

Digital data output

In an 8-bit ADC we have an 8-bit digital data output of D0–D7, while in the 10-bit ADC the data output is D0–D9. To calculate the output voltage, we use the following formula:

$$D_{out} = \frac{V_{in}}{step\ size}$$

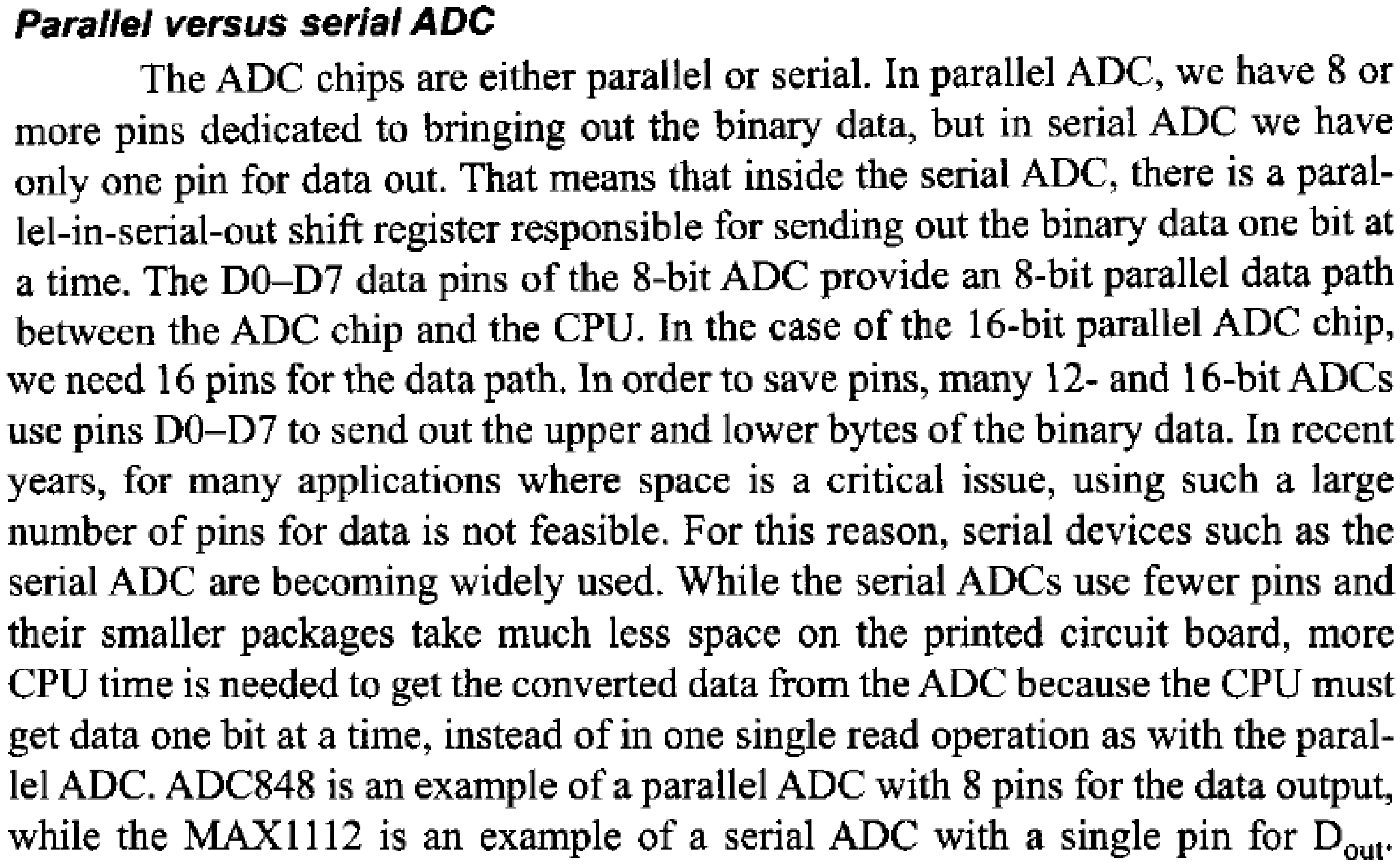
where D_{out} = digital data output (in decimal), V_{in} = analog input voltage, and step size (resolution) is the smallest change, which is $V_{ref}/256$ for an 8-bit ADC.

Example

For an 8-bit ADC, we have $V_{ref} = 2.56$ V. Calculate the D0–D7 output if the analog input is: (a) 1.7 V, and (b) 2.1 V.

Because the step size is 2.56/256 = 10 mV, we have the following:
(a) $D_{out} = 1.7$ V/10 mV=170 in decimal, which gives us 10101010 in binary for D7–D0.
(b) $D_{out} = 2.1$ V/10 mV = 210 in decimal, which gives us 11010010 in binary for D7–D0.

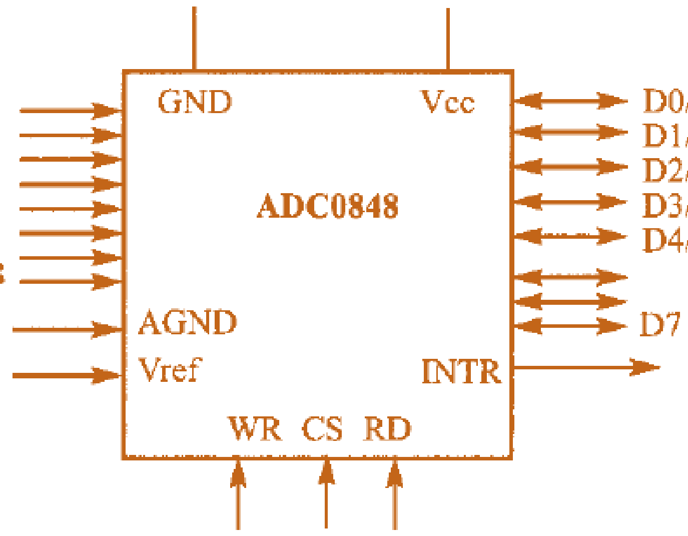
Major Characteristic of ADC

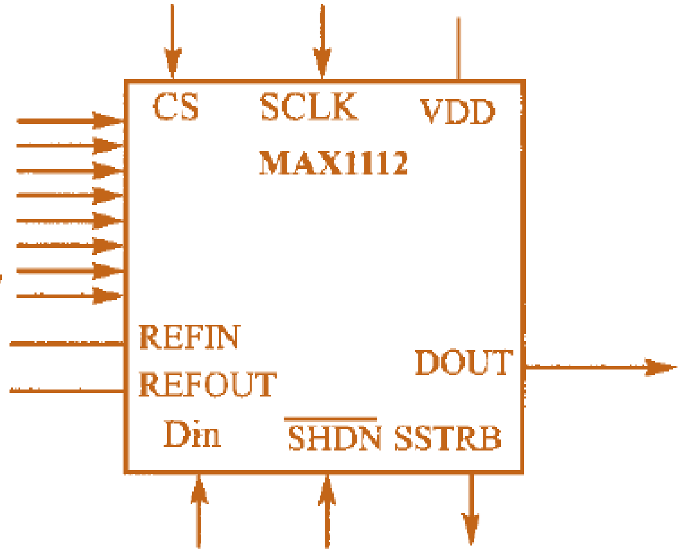


Parallel versus serial ADC

The ADC chips are either parallel or serial. In parallel ADC, we have 8 or more pins dedicated to bringing out the binary data, but in serial ADC we have only one pin for data out. That means that inside the serial ADC, there is a parallel-in-serial-out shift register responsible for sending out the binary data one bit at a time. The D0–D7 data pins of the 8-bit ADC provide an 8-bit parallel data path between the ADC chip and the CPU. In the case of the 16-bit parallel ADC chip, we need 16 pins for the data path. In order to save pins, many 12- and 16-bit ADCs use pins D0–D7 to send out the upper and lower bytes of the binary data. In recent years, for many applications where space is a critical issue, using such a large number of pins for data is not feasible. For this reason, serial devices such as the serial ADC are becoming widely used. While the serial ADCs use fewer pins and their smaller packages take much less space on the printed circuit board, more CPU time is needed to get the converted data from the ADC because the CPU must get data one bit at a time, instead of in one single read operation as with the parallel ADC. ADC848 is an example of a parallel ADC with 8 pins for the data output, while the MAX1112 is an example of a serial ADC with a single pin for D_{out}.

ADC Block Diagram





Major Characteristic of ADC

Analog input channels

Many data acquisition applications need more than one ADC. For this reason, we see ADC chips with 2, 4, 8, or even 16 channels on a single chip. Multiplexing of analog inputs is widely used as shown in the ADC848 and MAX1112. In these chips, we have 8 channels of analog inputs, allowing us to monitor multiple quantities such as temperature, pressure, heat, and so on. AVR microcontroller chips come with up to 16 ADC channels.

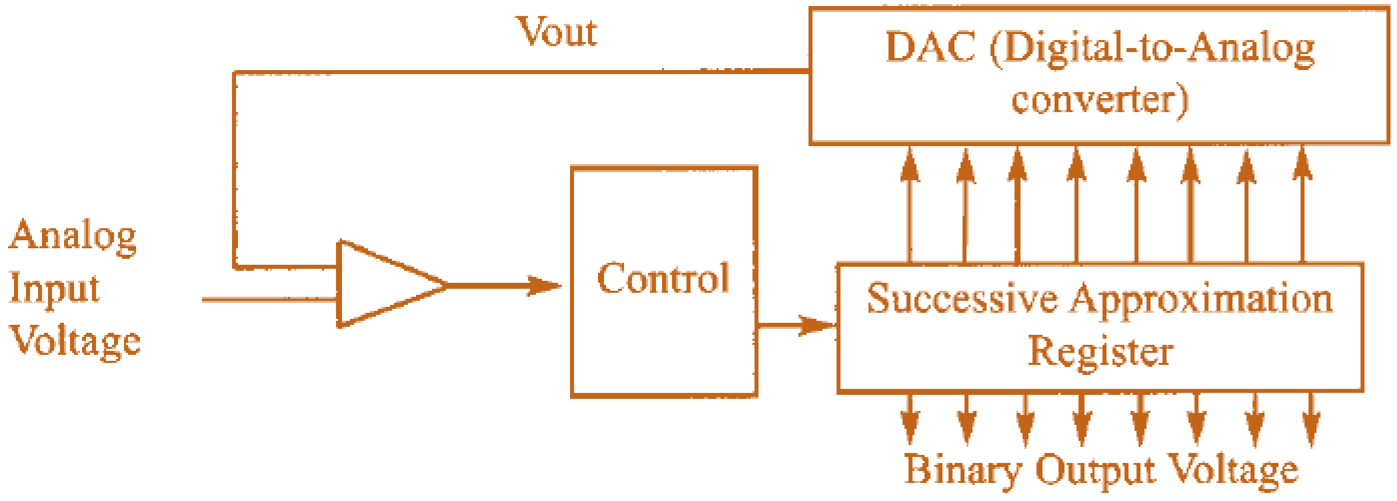
Start conversion and end-of-conversion signals

The fact that we have multiple analog input channels and a single digital output register creats the need for start conversion (SC) and end-of-conversion (EOC) signals. When SC is activated, the ADC starts converting the analog input value of Vin to an *n*-bit digital number. The amount of time it takes to convert varies depending on the conversion method as was explained earlier. When the data conversion is complete, the end-of-conversion signal notifies the CPU that the converted data is ready to be picked up.

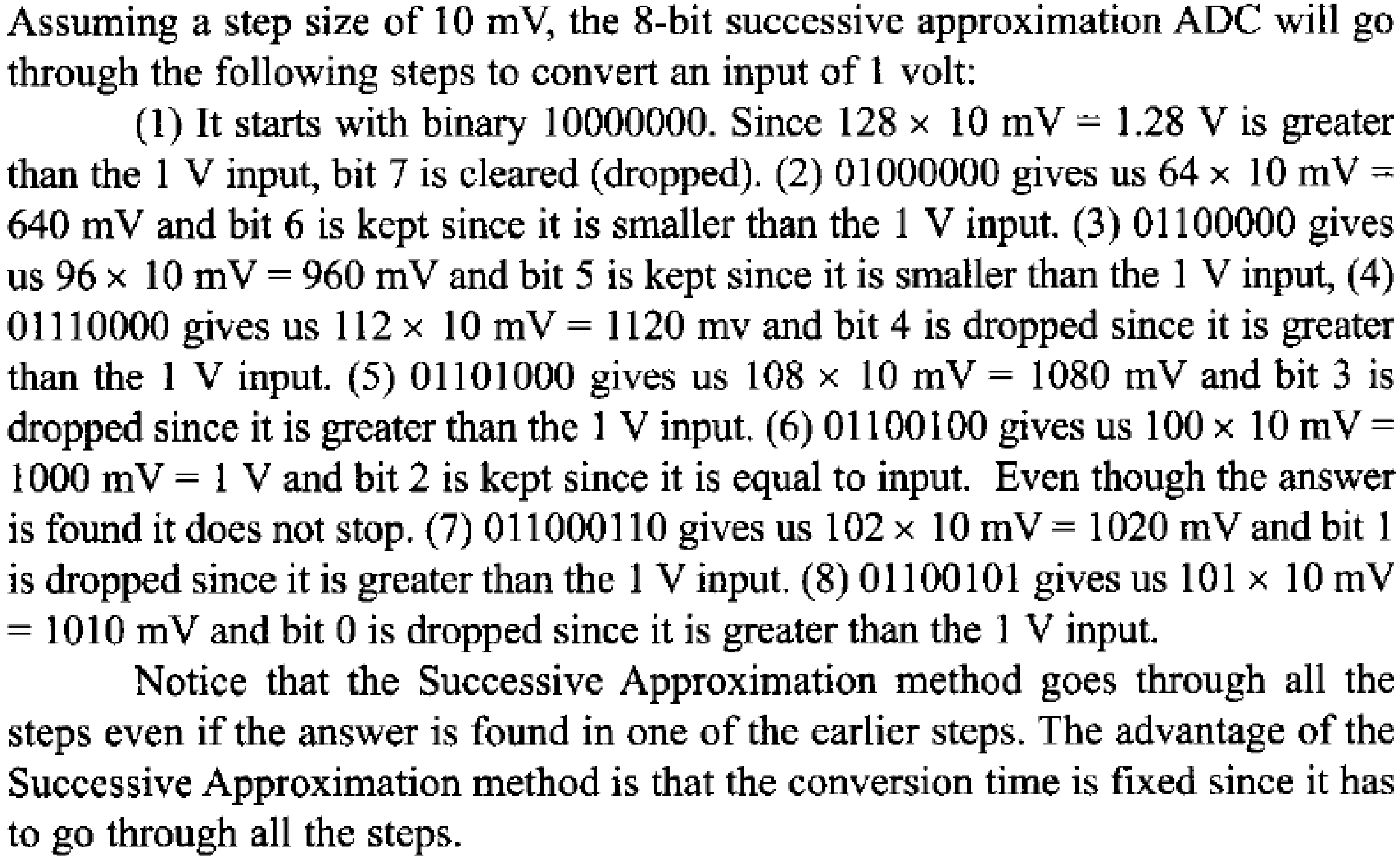
Successive Approximation ADC

Successive Approximation ADC

Successive Approximation is a widely used method of converting an analog input to digital output. It has three main components: (a) successive approximation register (SAR), (b) comparator, and (c) control unit. See the figure below.


Vout
DAC (Digital-to-Analog converter)
Analog Input Voltage
Control
Successive Approximation Register
Binary Output Voltage

Successive Approximation ADC



Assuming a step size of 10 mV, the 8-bit successive approximation ADC will go through the following steps to convert an input of 1 volt:

(1) It starts with binary 10000000. Since 128 × 10 mV = 1.28 V is greater than the 1 V input, bit 7 is cleared (dropped). (2) 01000000 gives us 64 × 10 mV = 640 mV and bit 6 is kept since it is smaller than the 1 V input. (3) 01100000 gives us 96 × 10 mV = 960 mV and bit 5 is kept since it is smaller than the 1 V input, (4) 01110000 gives us 112 × 10 mV = 1120 mv and bit 4 is dropped since it is greater than the 1 V input. (5) 01101000 gives us 108 × 10 mV = 1080 mV and bit 3 is dropped since it is greater than the 1 V input. (6) 01100100 gives us 100 × 10 mV = 1000 mV = 1 V and bit 2 is kept since it is equal to input. Even though the answer is found it does not stop. (7) 011000110 gives us 102 × 10 mV = 1020 mV and bit 1 is dropped since it is greater than the 1 V input. (8) 01100101 gives us 101 × 10 mV = 1010 mV and bit 0 is dropped since it is greater than the 1 V input.

Notice that the Successive Approximation method goes through all the steps even if the answer is found in one of the earlier steps. The advantage of the Successive Approximation method is that the conversion time is fixed since it has to go through all the steps.

ADC Programming in the AVR

Because the ADC is widely used in data acquisition, in recent years an increasing number of microcontrollers have had an on-chip ADC peripheral, just like timers and USART. An on-chip ADC eliminates the need for an external ADC connection, which leaves more pins for other I/O activities. The vast majority of the AVR chips come with ADC.

ATmega32 ADC features

The ADC peripheral of the ATmega32 has the following characteristics:

(a) It is a 10-bit ADC.

(b) It has 8 analog input channels, 7 differential input channels, and 2 differential input channels with optional gain of 10x and 200x.

(c) The converted output binary data is held by two special function registers called ADCL (A/D Result Low) and ADCH (A/D Result High).

(d) Because the ADCH:ADCL registers give us 16 bits and the ADC data out is only 10 bits wide, 6 bits of the 16 are unused. We have the option of making either the upper 6 bits or the lower 6 bits unused.

(e) We have three options for V_{ref}. V_{ref} can be connected to AVCC (Analog V_{cc}), internal 2.56 V reference, or external AREF pin.

(f) The conversion time is dictated by the crystal frequency connected to the XTAL pins (Fosc) and ADPS0:2 bits.

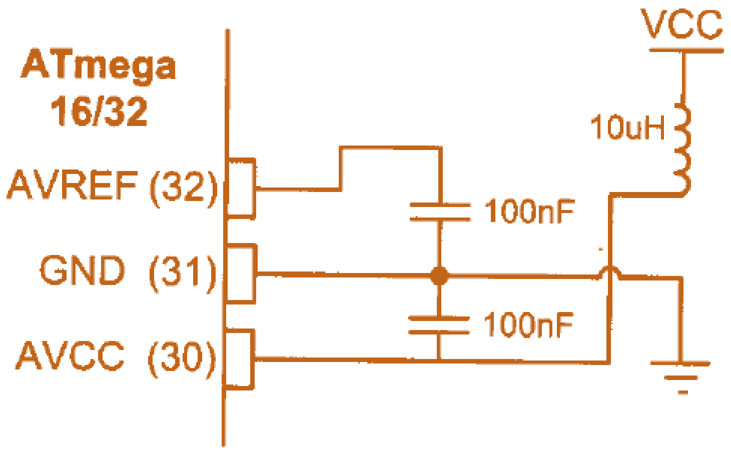
ADC Recommended Connection

Decoupling AVCC from VCC

the AVCC pin provides the supply for analog ADC circuitry. To get a better accuracy of AVR ADC we must provide a sta-bie voitage source to the AVCC pin.

Connecting a capacitor between V_{ref} and GND

By connecting a capacitor between the AVREF pin and GND you can make the V_{ref} voltage more stable and increase the precision of ADC.



ADC Programming in the AVR

In the AVR microcontroller five major registers are associated with the ADC that we deal with in this course . They are ADCH (high data), ADCL (low data), ADCSRA (ADC Control and Status Register), ADMUX (ADC multiplexer selection register), and SPIOR (Special Function I/O Register). We examine each of them in this section.

ADMUX register

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

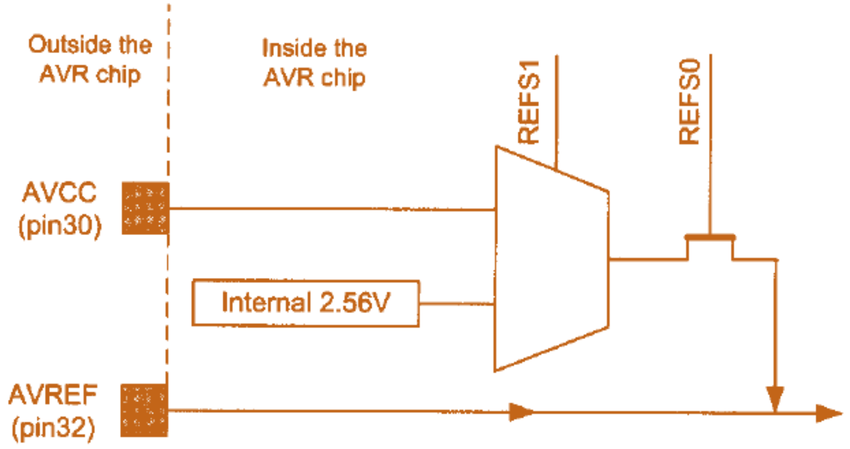
REFS1:0 Bit 7:6 Reference Selection Bits
These bits select the reference voltage for the ADC.

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

Figure below shows the block diagram of internal circuitry of V_{ref} selection. As you can see we have three options: (a) AREF pin, (b) AVCC pin, or (c) internal 2.56 V. The Table shows how the REFS1 and REFS0 bits of the ADMUX register can be used to select the V_{ref} source.



V_{ref} Source Selection Table for AVR

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

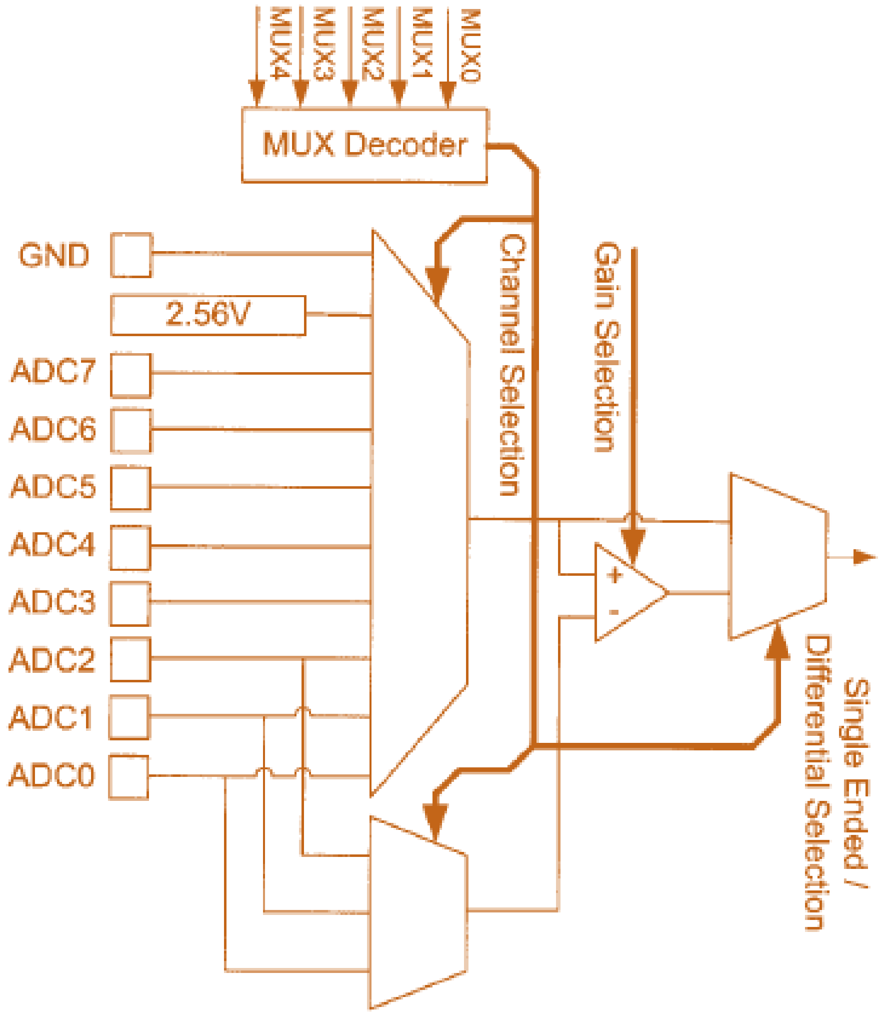
ADMUX

ADC input channel source

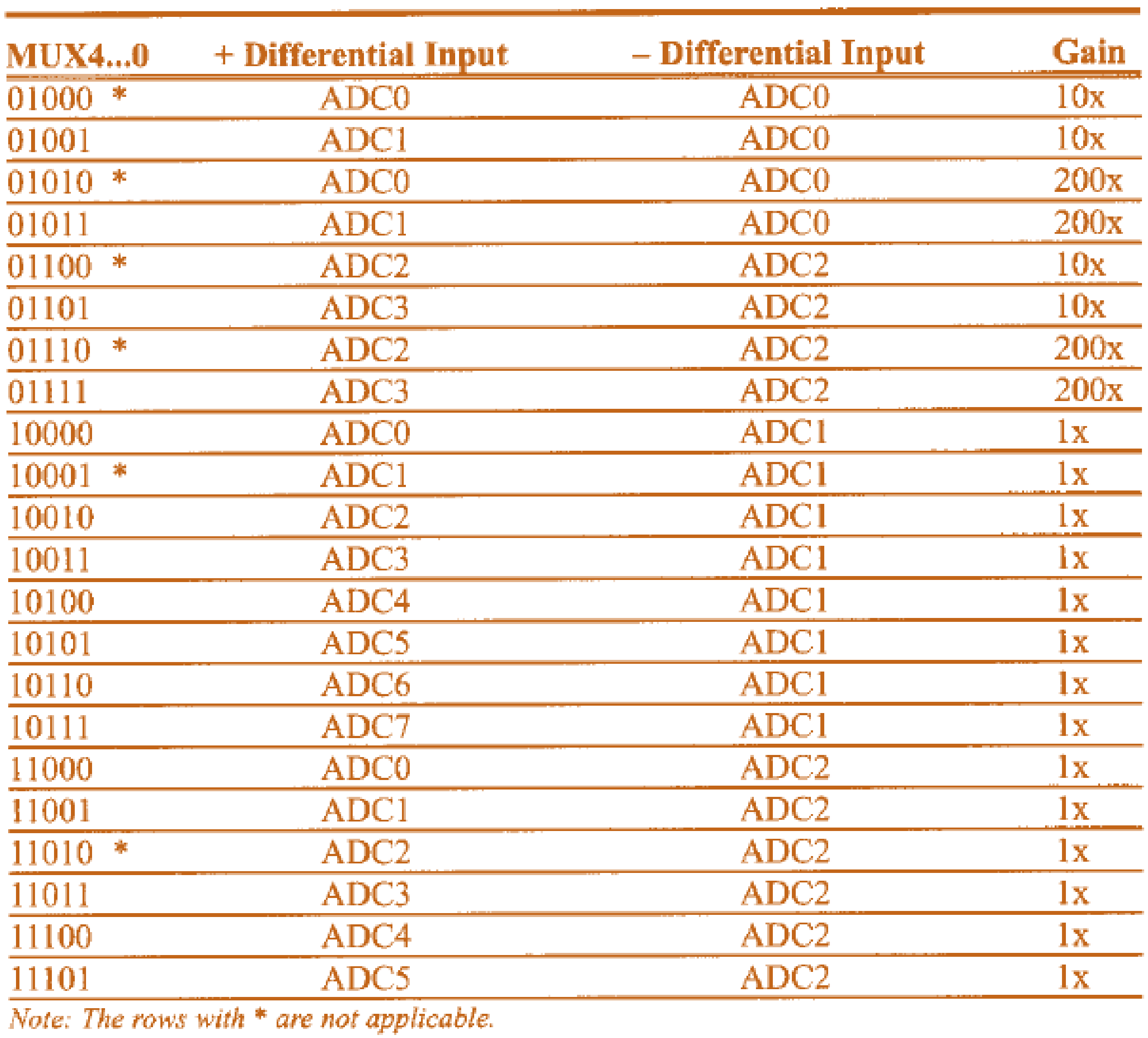
The Figure shows the schematic of the internal circuitry of input channel selection. As you can see in the figure, either single-ended or the differential input can be selected to be converted to digital data. If you select single-ended input, you can choose the input channel among ADC0 to ACD7. In this case a single pin is used as the analog line, and GND of the AVR chip is used as common ground.

Single-ended Channels

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



V_{ref} Source Selection



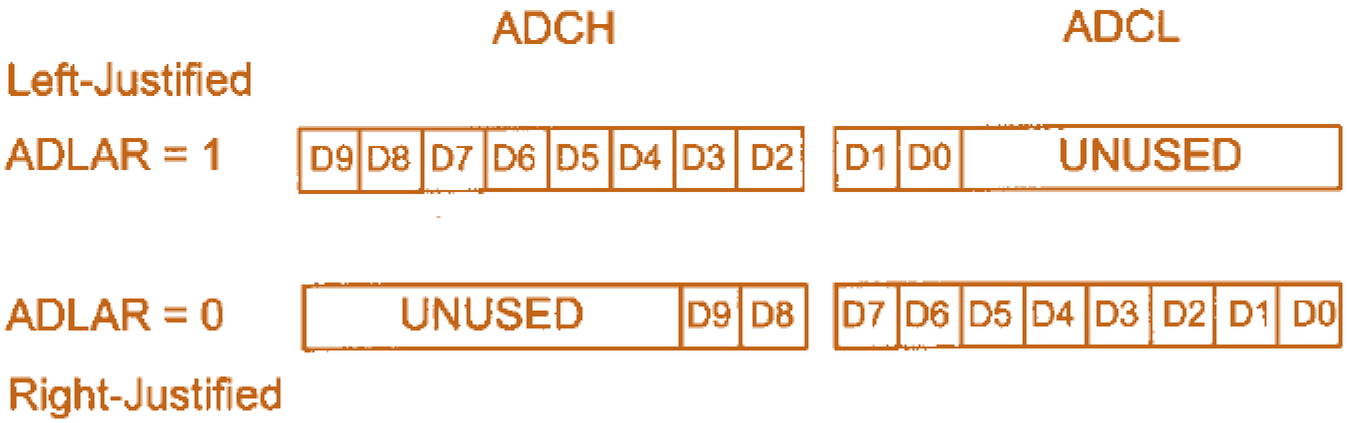
MUX4...0	+ Differential Input	– Differential Input	Gain
01000 *	ADC0	ADC0	10x
01001	ADC1	ADC0	10x
01010 *	ADC0	ADC0	200x
01011	ADC1	ADC0	200x
01100 *	ADC2	ADC2	10x
01101	ADC3	ADC2	10x
01110 *	ADC2	ADC2	200x
01111	ADC3	ADC2	200x
10000	ADC0	ADC1	1x
10001 *	ADC1	ADC1	1x
10010	ADC2	ADC1	1x
10011	ADC3	ADC1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

*Note: The rows with * are not applicable.*

ADMUX

ADLAR bit operation

The AVRs have a 10-bit ADC, which means that the result is 10 bits long and cannot be stored in a single byte. In AVR two 8-bit registers are dedicated to the ADC result, but only 10 of the 16 bits are used and 6 bits are unused. You can select the position of used bits in the bytes. If you set the ADLAR bit in ADMUX register, the result bits will be left-justified; otherwise, the result bits will be right-justified.



ADCH:ADCL

ADCH: ADCL registers

After the A/D conversion is complete, the result sits in registers ADCL (A/D Result Low Byte) and ACDH (A/D Result High Byte). As we mentioned before, the ADLAR bit of the ADMUX is used for making it right-justified or left-justified because we need only 10 of the 16 bits.

ADCSRA register

The ADCSRA register is the status and control register of ADC. Bits of this register control or monitor the operation of the ADC.

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADCSRA

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.

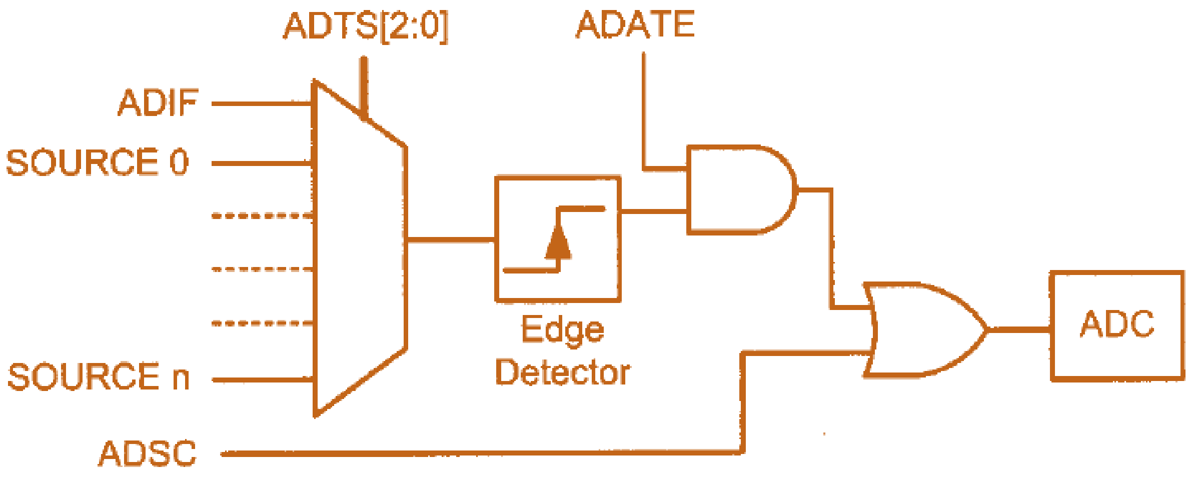
ADIE Bit 3 ADC Interrupt Enable
Setting this bit to one enables the ADC conversion complete interrupt.

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

ADC Start Conversion bit

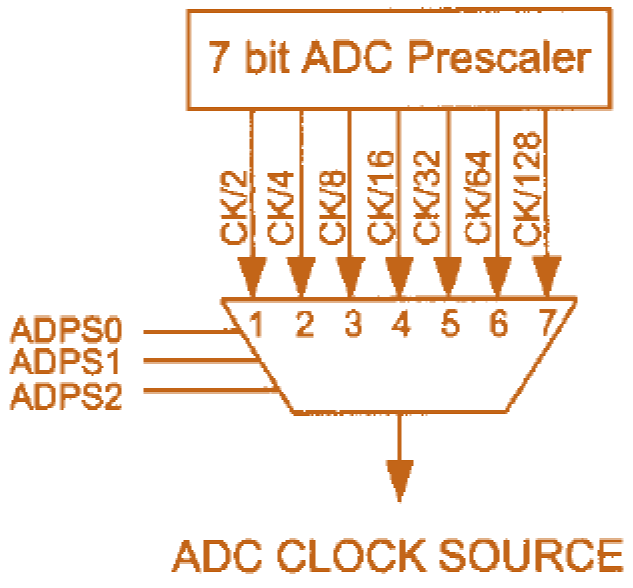
As we stated before, an ADC has a Start Conversion input. The AVR chip has a special circuit to trigger start conversion. As you see in the Figure , in addition to the ADCSC bit of ADCSRA there are other sources to trigger start of conversion. If you set the ADATE bit of ADCSRA to high, you can select auto trigger source by updating ADTS2:0 in the SFIOR register. If ADATE is cleared, the ADTS2:0 settings will have no effect. Notice that there are many considerations if you want to use auto trigger mode. We will not cover auto trigger mode in this course If you want to use auto trigger mode we strongly recommend you to refer to the datasheet of the device that you want to use at www.atmel.com.



ADCSRA

A/D conversion time

As you see in the Figure , by using the ADPS2:0 bits of the ADCSRA register we can set the A/D conversion time. To select the conversion time, we can select any of Fosc/2, Fosc/4, Fosc/8, Fosc/16, Fosc/32, Fosc/64, or Fosc/128 for ADC clock, where Fosc is the speed of the crystal frequency connected to the AVR chip. Notice that the multiplexer has 7 inputs since the option ADPS2:0 = 000 is reserved. For the AVR, the ADC requires an input clock frequency less than 200 kHz for the maximum accuracy.



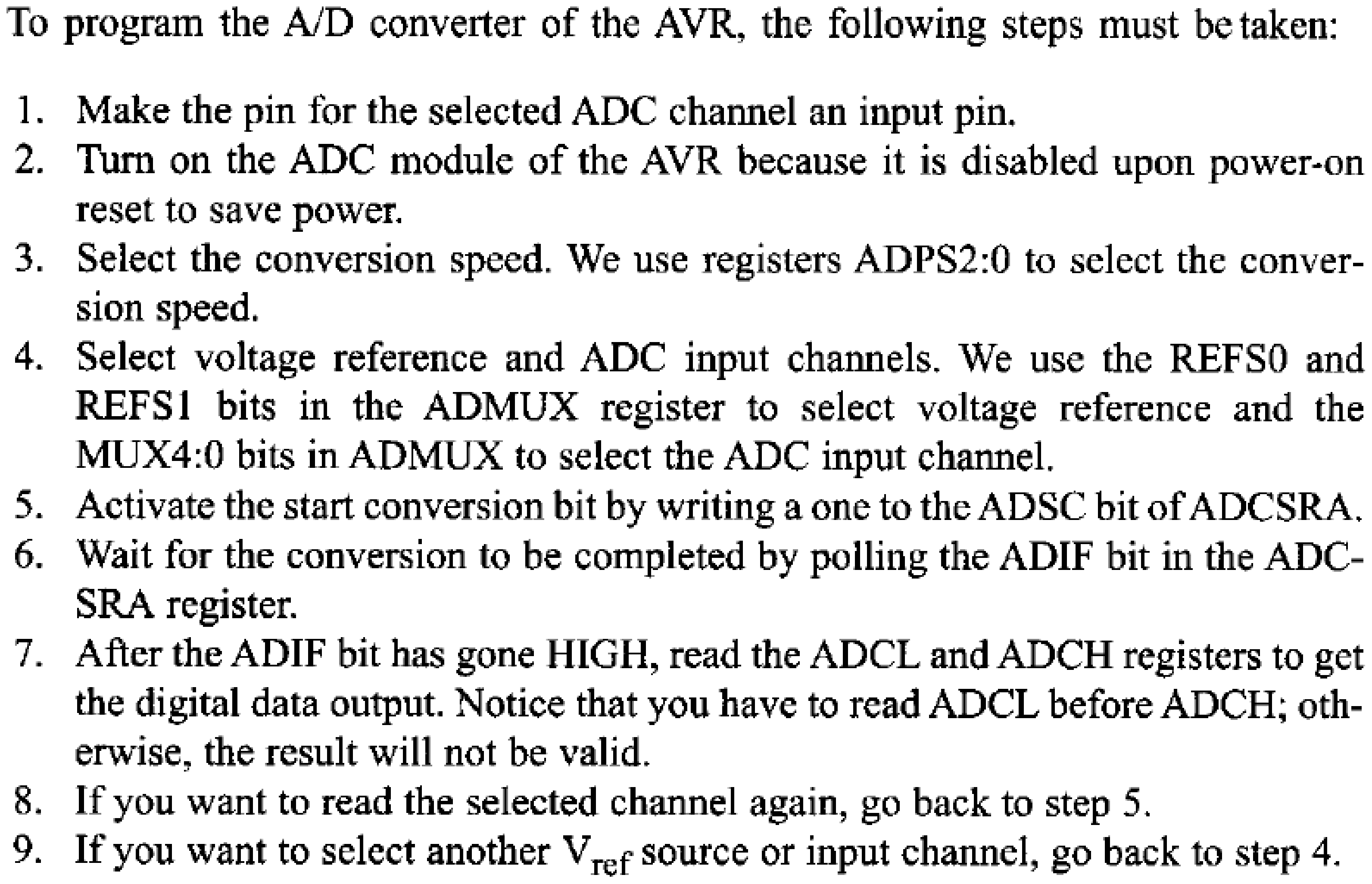
ADPS2	ADPS1	ADPS0	ADC Clock
0	0	0	Reserved
0	0	1	CK/2
0	1	0	CK/4
0	1	1	CK/8
1	0	0	CK/16
1	0	1	CK/32
1	1	0	CK/64
1	1	1	CK/128

Example

An AVR is connected to the 8 MHz crystal oscillator. Calculate the ADC frequency for (a) ADPS2:0 = 001 (b) ADPS2:0 = 100 (c) ADPS2:0 = 111

(a) Because ADPS2:0 = 001 (1 decimal), the ck/2 input will be activated; we have
8 MHz / 2 = 4 MHz (greater than 200 kHz and not valid)
(b) Because ADPS2:0 = 100 (4 decimal), the ck/8 input will be activated; we have
8 MHz / 16 = 500 kHz (greater than 200 kHz and not valid)
(c) Because ADPS2:0 = 111 (7 decimal), the ck/128 input will be activated; we have
8 MHz / = 62 kHz (a valid option since it is less than 200 kHz)

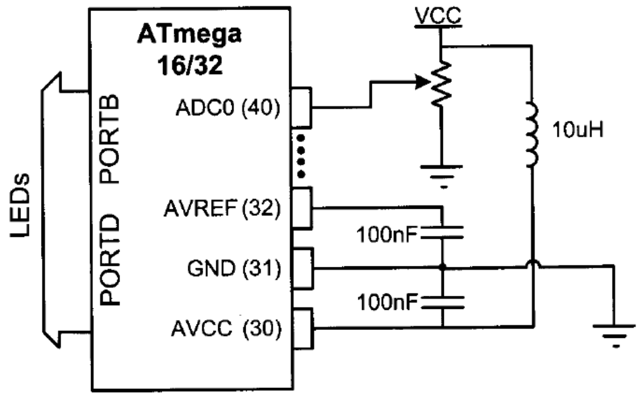
A/D Using Polling



To program the A/D converter of the AVR, the following steps must be taken:

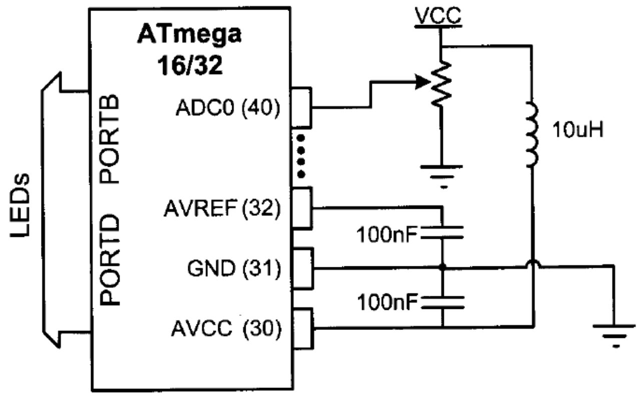
1. Make the pin for the selected ADC channel an input pin.
2. Turn on the ADC module of the AVR because it is disabled upon power-on reset to save power.
3. Select the conversion speed. We use registers ADPS2:0 to select the conversion speed.
4. Select voltage reference and ADC input channels. We use 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.

Example (Assembly)



```
.INCLUDE "M32DEF.INC"
      LDI   R16,0xFF
      OUT   DDRB, R16          ;make Port B an output
      OUT   DDRD, R16          ;make Port D an output
      LDI   R16,0
      OUT   DDRA, R16          ;make Port A an input for ADC
      LDI   R16,0x87           ;enable ADC and select ck/128
      OUT   ADCSRA, R16
      LDI   R16,0xC0           ;2.56V Vref, ADC0 single ended
      OUT   ADMUX, R16         ;input, right-justified data
READ_ADC:
      SBI   ADCSRA,ADSC        ;start conversion
KEEP_POLING:                   ;wait for end of conversion
      SBIS  ADCSRA,ADIF        ;is it end of conversion yet?
      RJMP  KEEP_POLING        ;keep polling end of conversion
      SBI   ADCSRA,ADIF        ;write 1 to clear ADIF flag
      IN    R16,ADCL           ;YOU HAVE TO READ ADCL FIRST
      OUT   PORTD,R16          ;give the low byte to PORTD
      IN    R16,ADCH           ;READ ADCH AFTER ADCL
      OUT   PORTB,R16          ;give the high byte to PORTB
      RJMP  READ_ADC           ;keep repeating it
```

Example (AVR 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;
}
```

A/D Using Interrupts

we showed how to use interrupts instead of polling to avoid tying down the microcontroller. To program the A/D using the interrupt method, we need to set HIGH the ADIE (A/D interrupt enable) flag. Upon completion of conversion, the ADIF (A/D interrupt flag) changes to HIGH; if ADIE = 1, it will force the CPU to jump to the ADC interrupt handler.

```
.INCLUDE "M32DEF.INC"
.CSEG
      RJMP  MAIN
.ORG  ADCCaddr
      RJMP  ADC_INT_HANDLER
.ORG  40
;*****************************
MAIN: LDI   R16, HIGH(RAMEND)
      OUT   SPH, R16
      LDI   R16, LOW(RAMEND)
      OUT   SPL,R16
      SEI
      LDI   R16,0xFF
      OUT   DDRB, R16        ;make Port B an output
      OUT   DDRD, R16        ;make Port D an output
      LDI   R16,0
      OUT   DDRA, R16        ;make Port A an input for ADC
      LDI   R16,0x8F         ;enable ADC and select ck/128
      OUT   ADCSRA, R16
      LDI   R16,0xC0         ;2.56V Vref, ADC0 single ended
      OUT   ADMUX, R16       ;input right-justified data
      SBI   ADCSRA,ADSC      ;start conversion
WAIT_HERE:
      RJMP  WAIT_HERE       ;keep repeating it
;*****************************
ADC_INT_HANDLER:
      IN    R16,ADCL        ;YOU HAVE TO READ ADCL FIRST
      OUT   PORTD,R16       ;give the low byte to PORTD
      IN    R16,ADCH        ;READ ADCH AFTER ADCL
      OUT   PORTB,R16       ;give the high byte to PORTB
      SBI   ADCSRA,ADSC     ;start conversion again
      RETI
```

A/D Using Interrupts

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

پایان

موفق و پیروز باشید