USING ADC IN ATMEGA32

CSE 315

Md. Iftekharul Islam Sakib

LEARNING OBJECTIVE

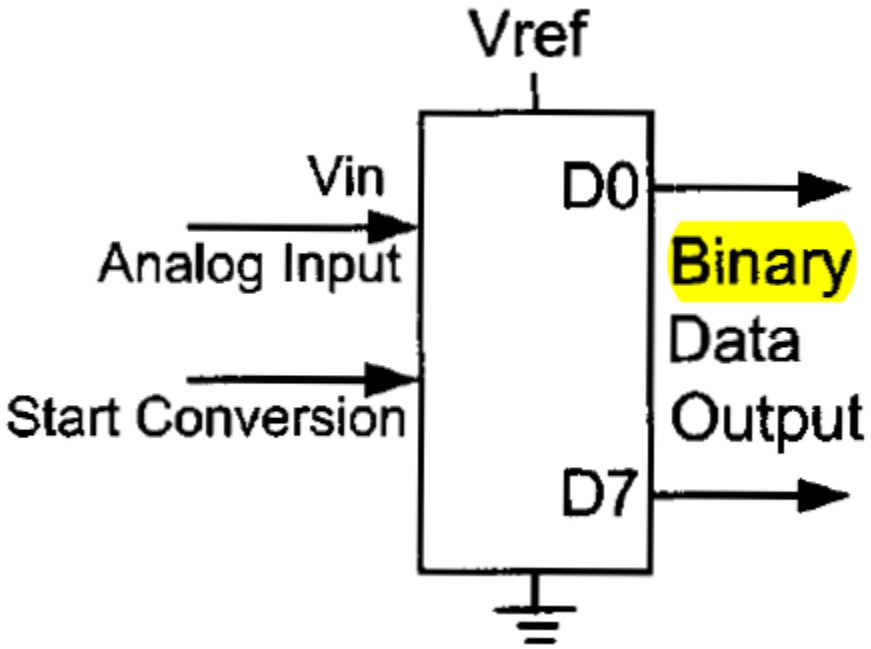
- Why do we need ADC?
- What is ADC?
- ADC in ATmega32
- A simple example

RECAP

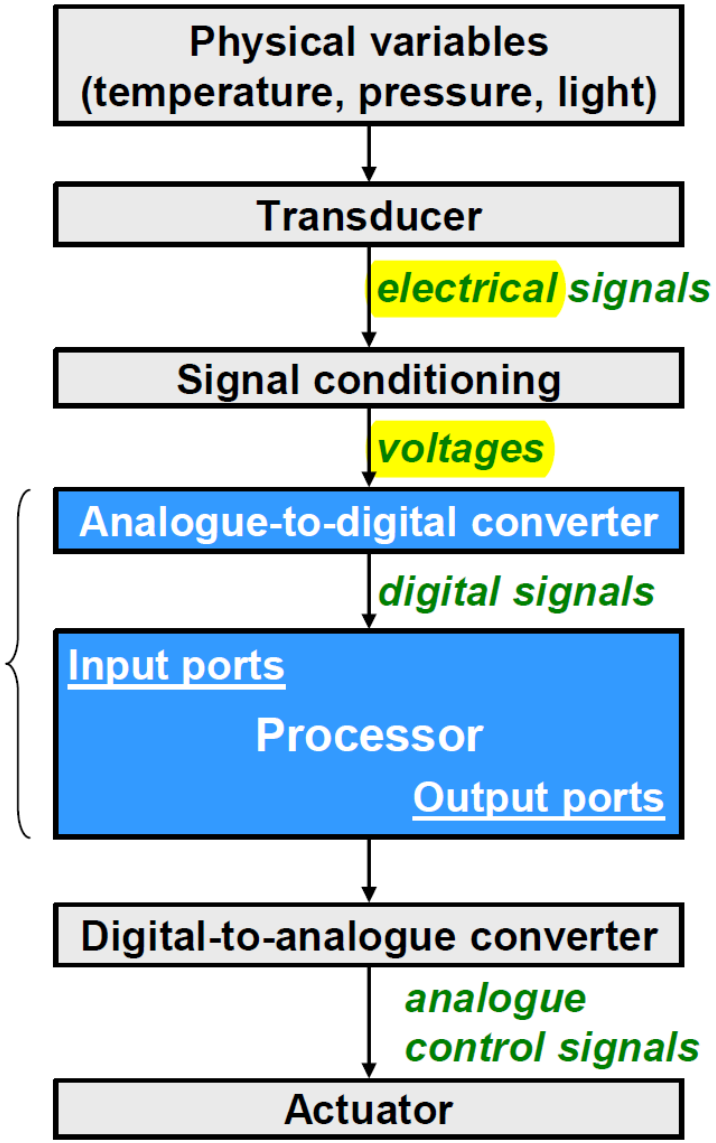
- MCU is a digital processor
  - Works with binary data
    - Just 2 voltage levels
- MCU needs to collect information from the real world
  - Temperature, Pressure, Humidity, Sound …
    - Continuous property
    - Infinite possible values
- Why do we collect information?
  - To analyze the situation
  - To take appropriate actions

RECAP

- But how to collect information?
  - Sensors: Convert physical property into analog signal
    - Voltage, Current, Resistance, …
- How can MCU recognize these signal?
  - Need to convert into digital data
  - How?
    - Approximation
  - Need something special!



TYPICAL EMBEDDED APPLICATION



Because of ADC is commonly needed, most modern microcontrollers include an in-built ADC unit.

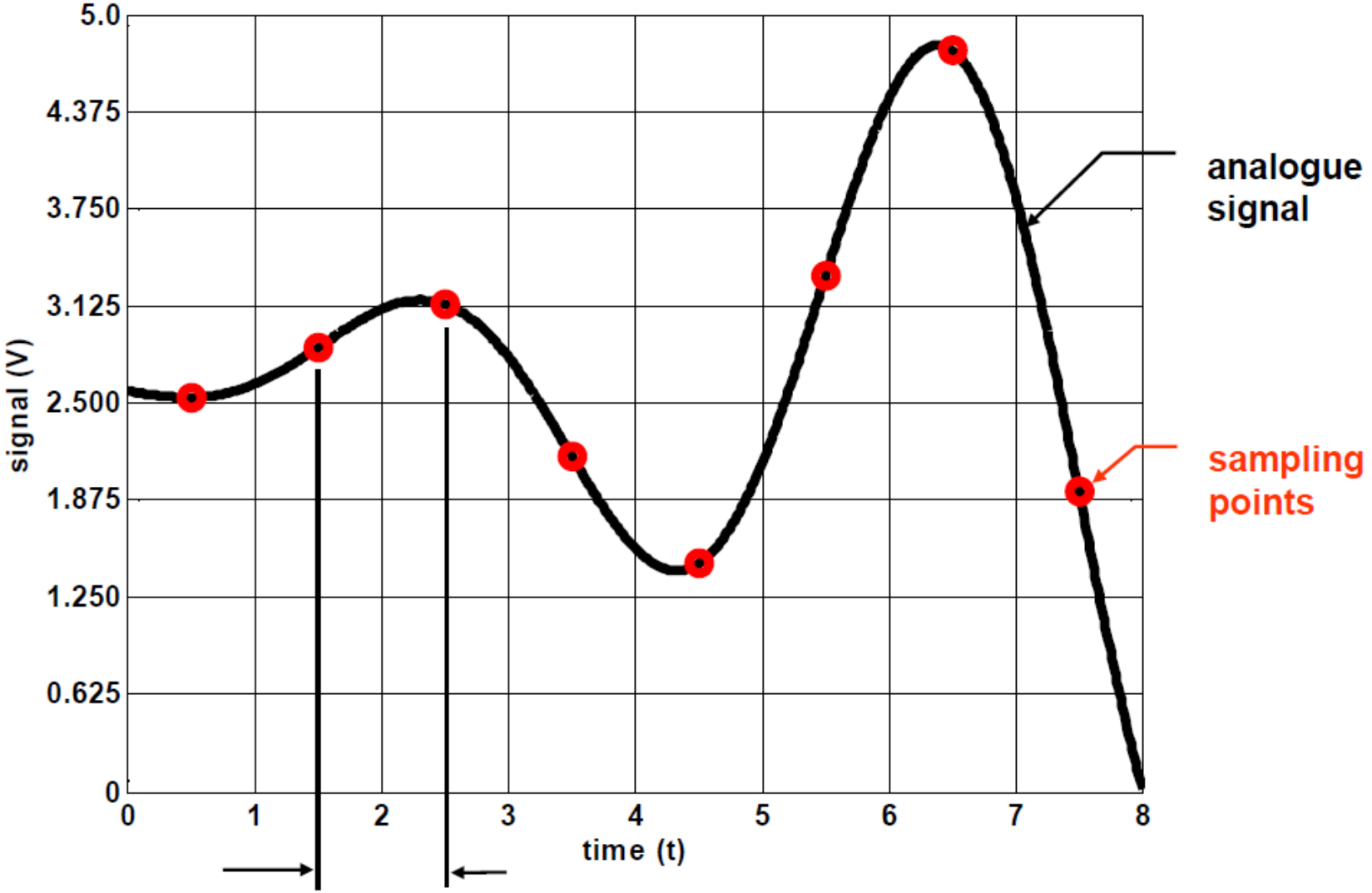
ADC: QUICK REVIEW

A-TO-D CONVERSION

- Two related steps

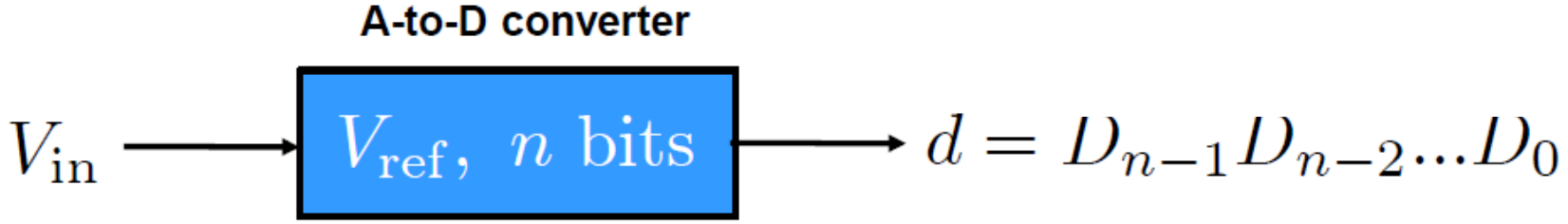
1. Sampling (Approximating the X-axis)
   - the analogue signal is extracted at regularly spaced time instants
   - the samples have real values
2. Quantization (Approximating the Y-axis)
   - the samples are quantized to discrete levels
   - each sample is represented as a binary value

SAMPLING AN ANALOGUE SIGNAL



What is a suitable sampling period for a signal ?

QUANTIZING THE SAMPLED SIGNAL



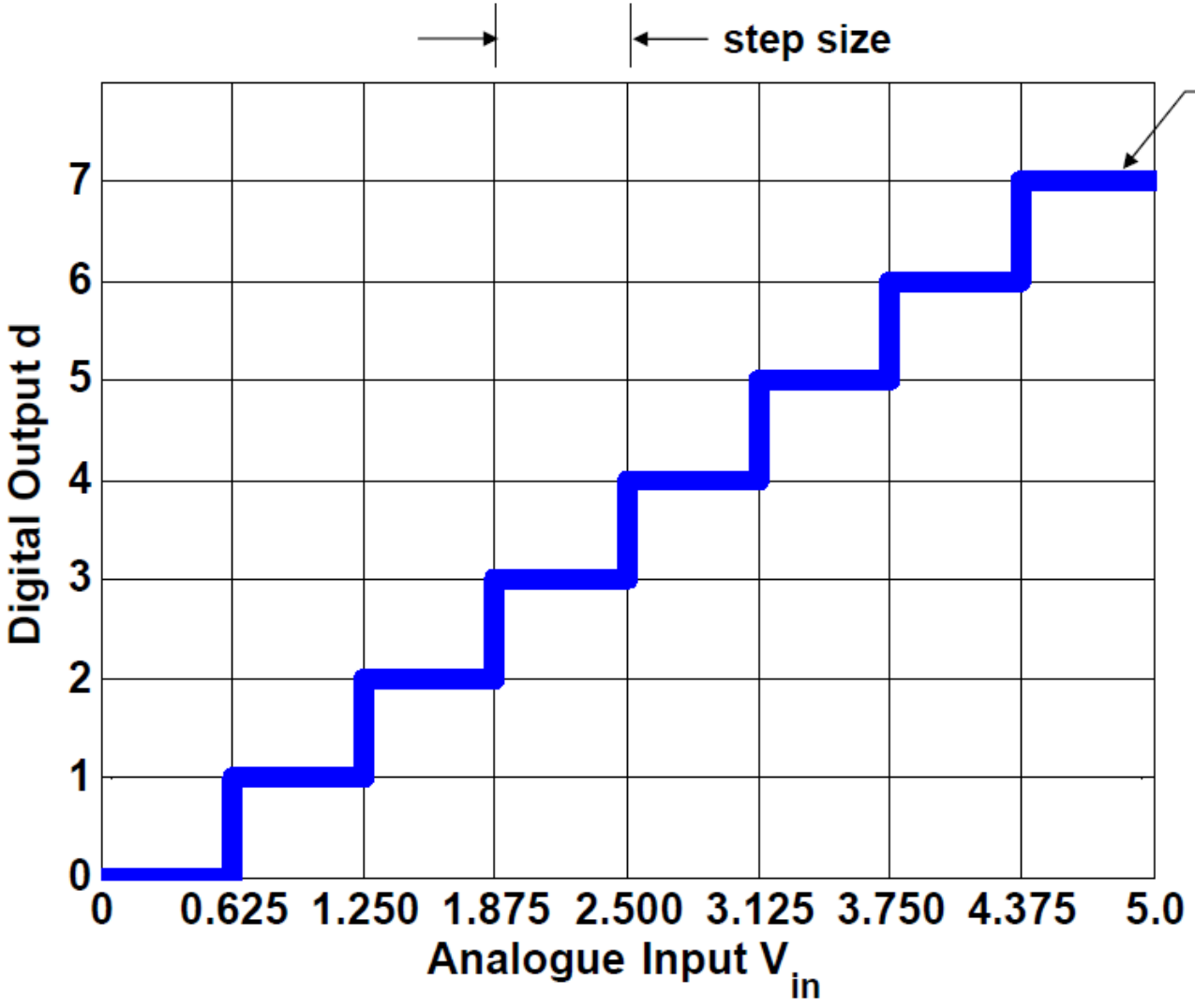
- Let's consider an n-bit ADC.
- Let V_{ref} be the reference voltage.
- Let V_{in} be the analogue input voltage.
- Let V_{min} be the minimum allowable input voltage, usually $V_{min} = 0$.
- The ADC's digital output, $d = D_{n-1}D_{n-2} \ldots D_0$, is given as

$$d = \text{round down}\left[\frac{V_{\text{in}} - V_{\text{min}}}{\text{step size}}\right]$$

- The step size (resolution) is the smallest change in input that can be discerned by the ADC:

$$\text{step size} = \frac{V_{\text{ref}} - V_{\text{min}}}{2^n}$$

QUANTIZING THE SAMPLED SIGNAL



Maximum digital output

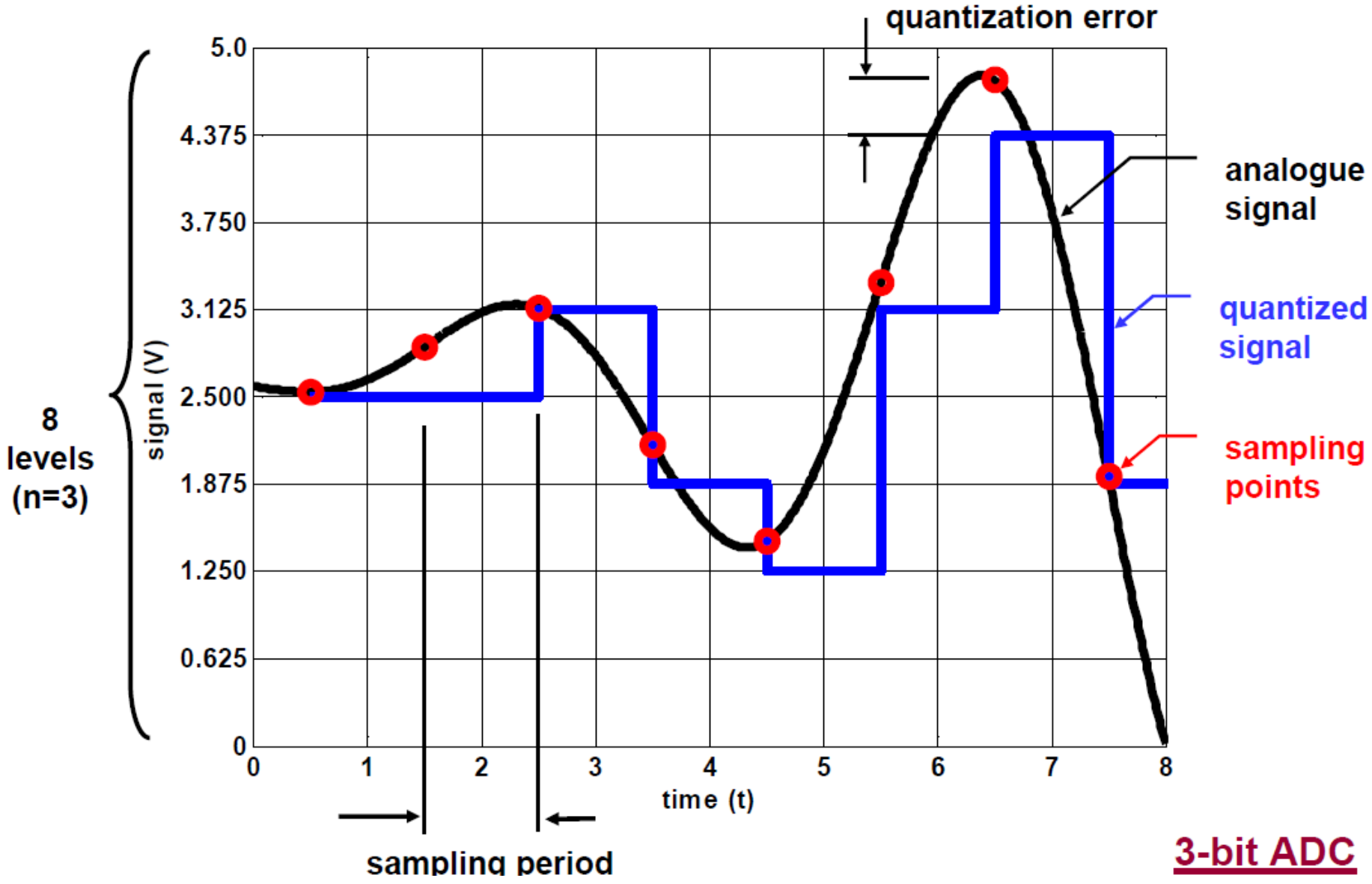
2^3 levels

Reference Voltage V_{ref}

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

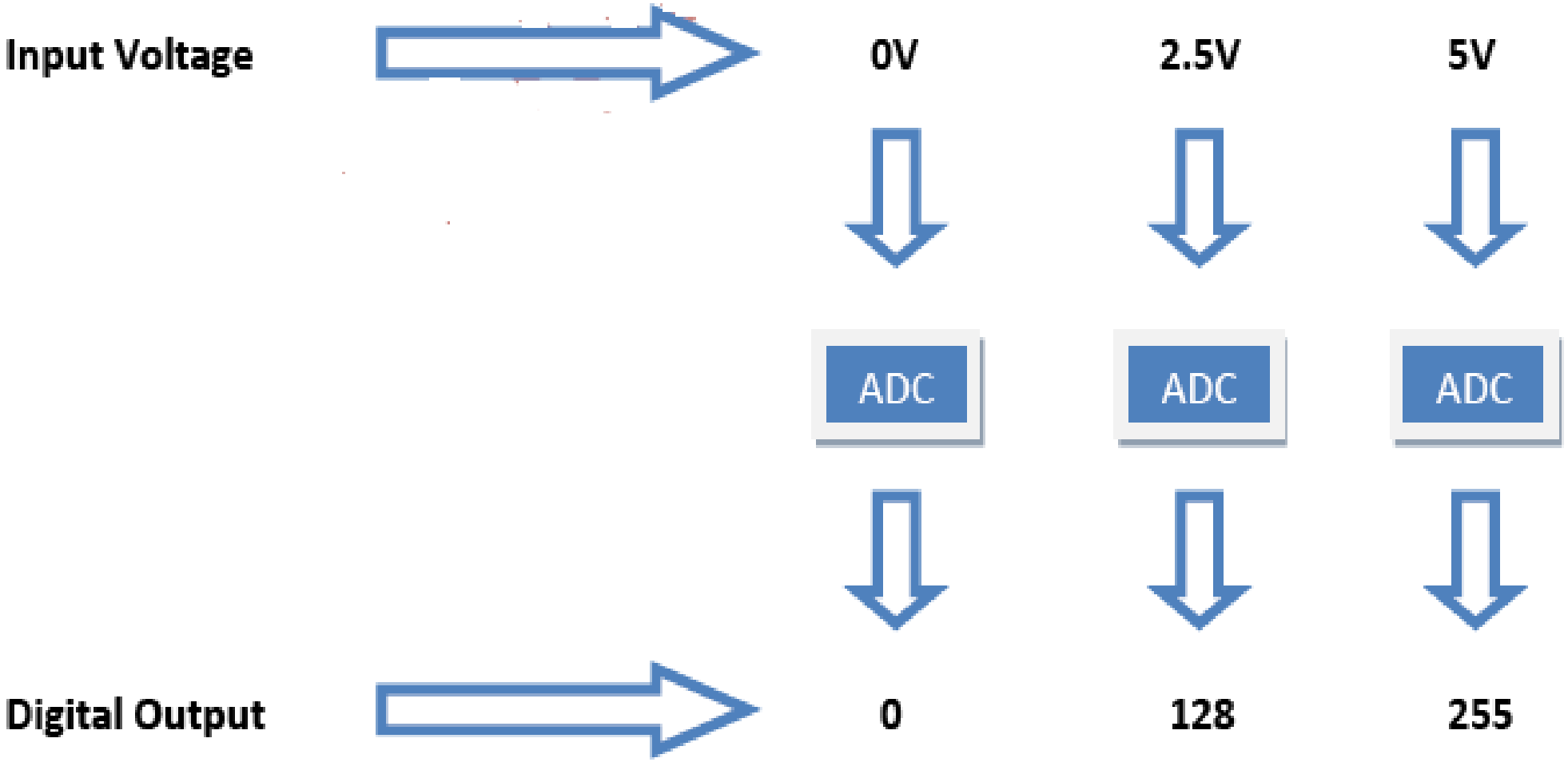
A 3-bit A-to-D converter

QUANTIZING THE SAMPLED SIGNAL



3-bit ADC

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.

Solution:

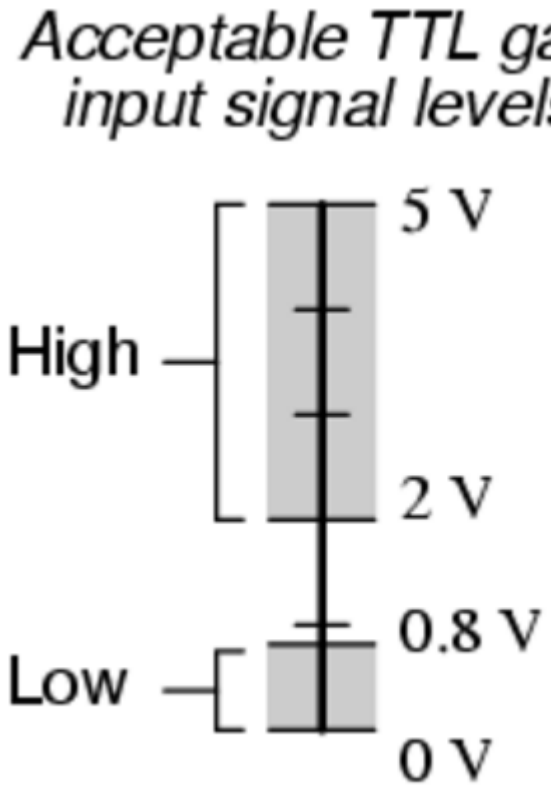
Because the step size is 2.56/256 = 10 mV, we have the following:

(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.

A QUESTION

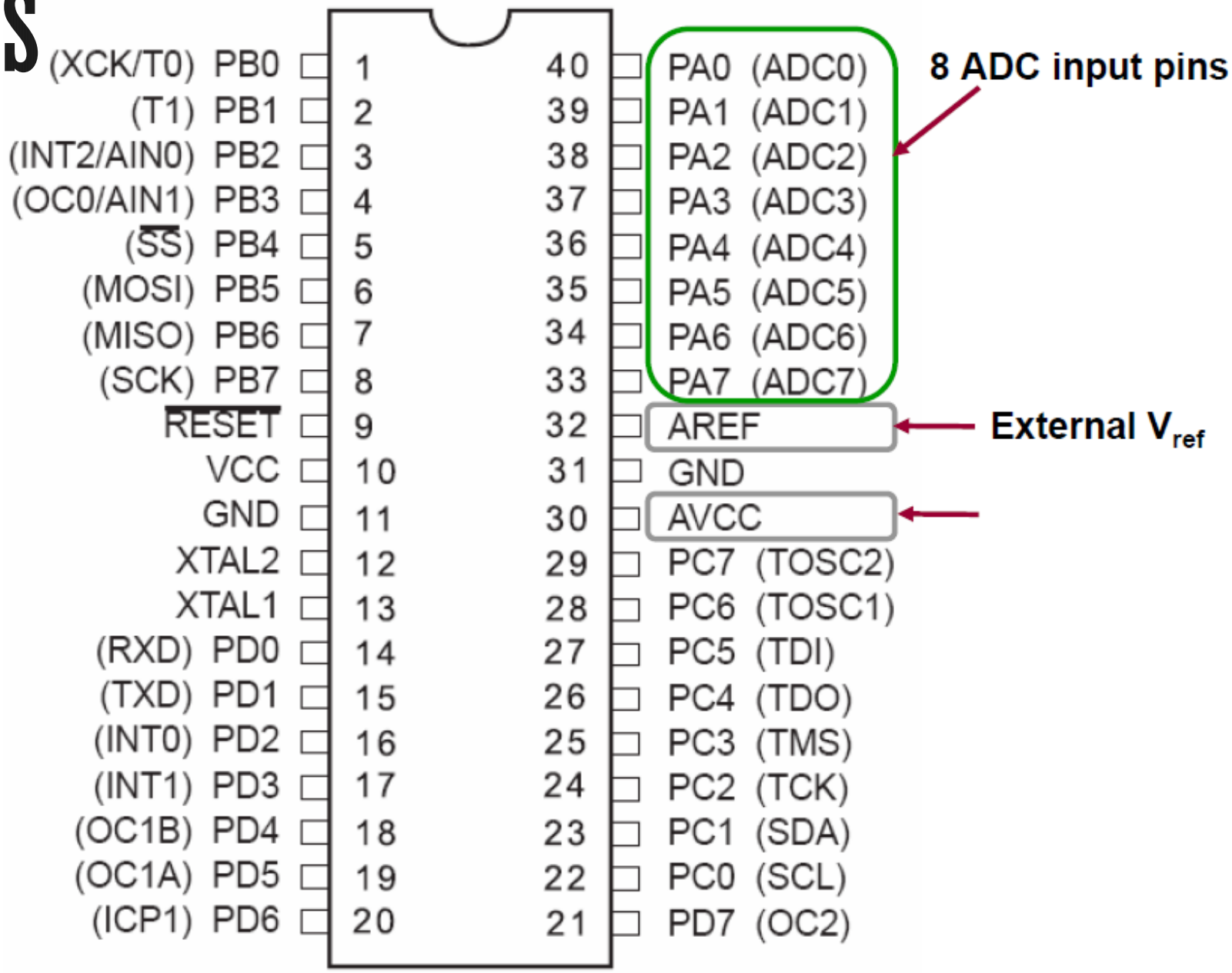
- What is the difference between an ADC and a digital input pin?



THE ADC IN ATMEGA32

- The ADC has a 10-bit resolution.
  - The binary output has n = 10 bits
- The ADC has 8 input channels
  - Analogue input can come from 8 different sources
  - However, it performs conversion on only one channel at a time
- If default reference voltage V_{ref} = 5V is used
  - step size: 5 V/1024 = 4.88mV
- The clock rate of the ADC can be different from the CPU clock rate
  - One ADC conversion takes 13 ADC cycles
  - An ADC prescaler will decide the ADC clock rate

RELEVANT PINS



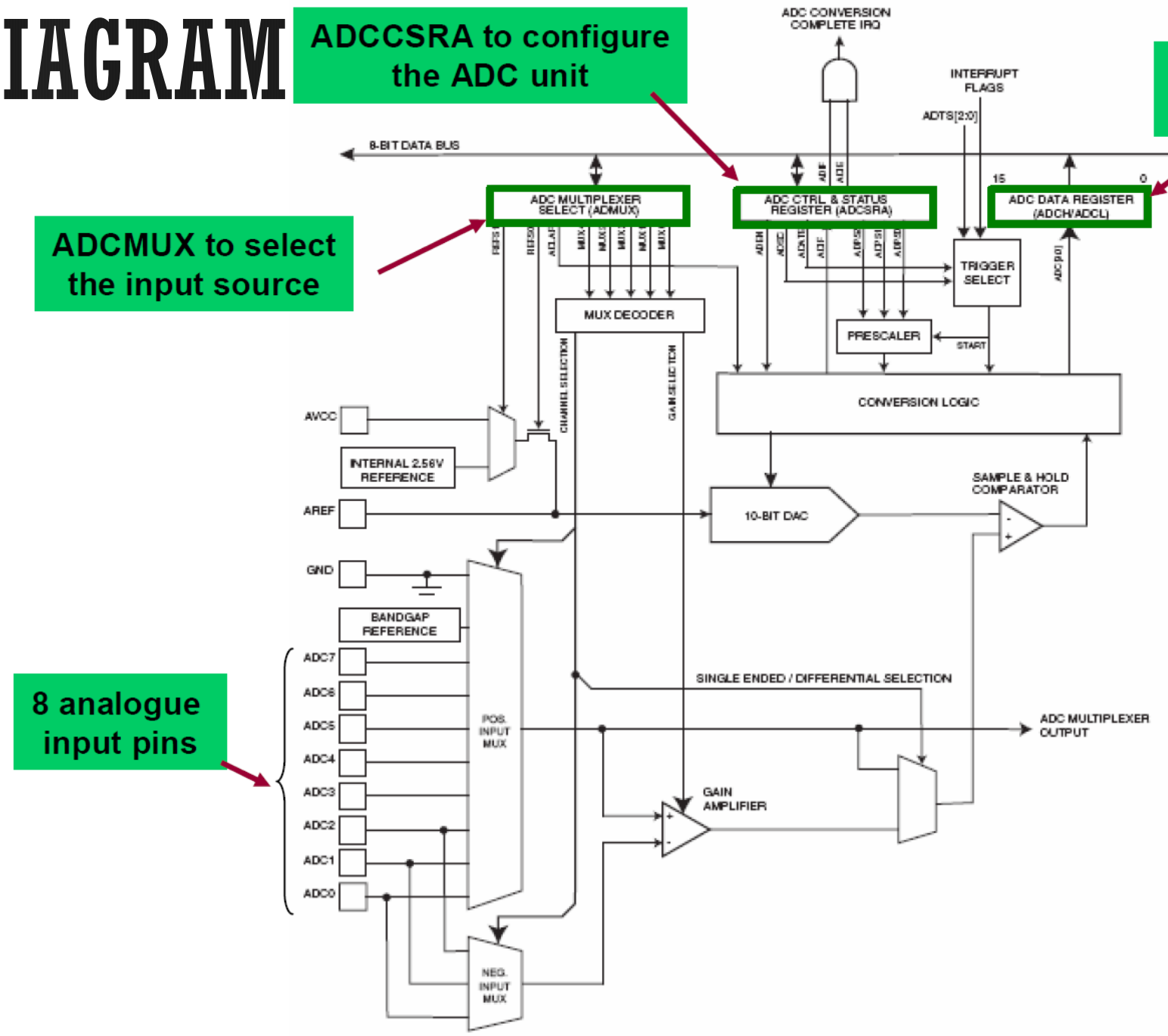
BLOCK DIAGRAM

ADCCSRA to configure the ADC unit

ADCH/ACHL register to store digital output

ADCMUX to select the input source

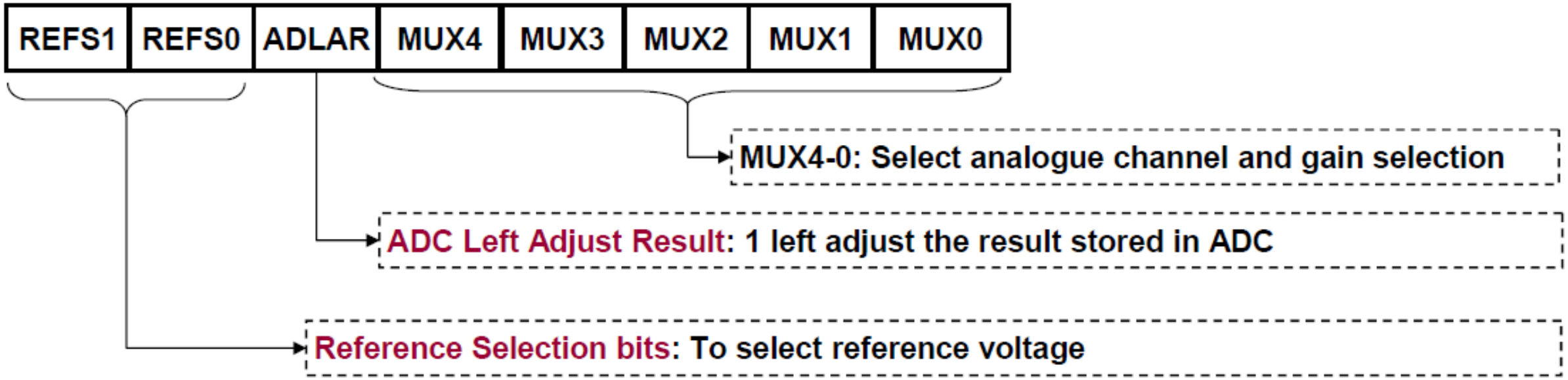
8 analogue input pins



MAIN ASPECTS

- What are the relevant ADC registers?
  - ADCMUX
  - ADCH/ADCL
  - ADCCSRA
  - SFIOR
- What are the steps to use the ADC?
- How to use ADC interrupt?

ADC MULTIPLEXER SELECTION REGISTER (ADCMUX)



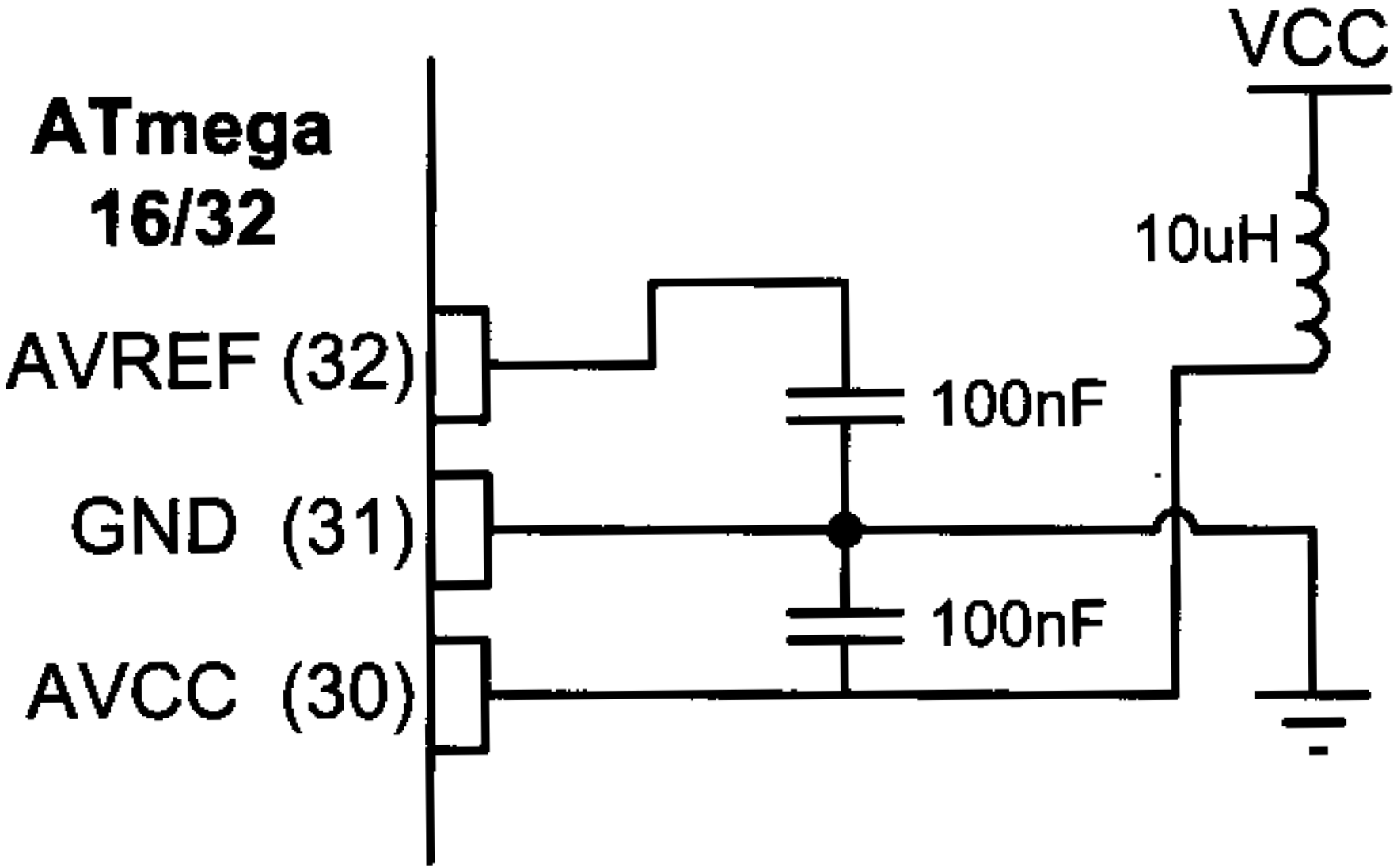
SELECTING REFERENCE VOLTAGE V_{REF}

Table 11.1: ADC reference voltage selection

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

- Usually, mode 01 is used: AVCC = 5V as reference voltage.
- However, if the input voltage has a different dynamic range, we can use mode 00 to select an external reference voltage.

ADC RECOMMENDED CONNECTION



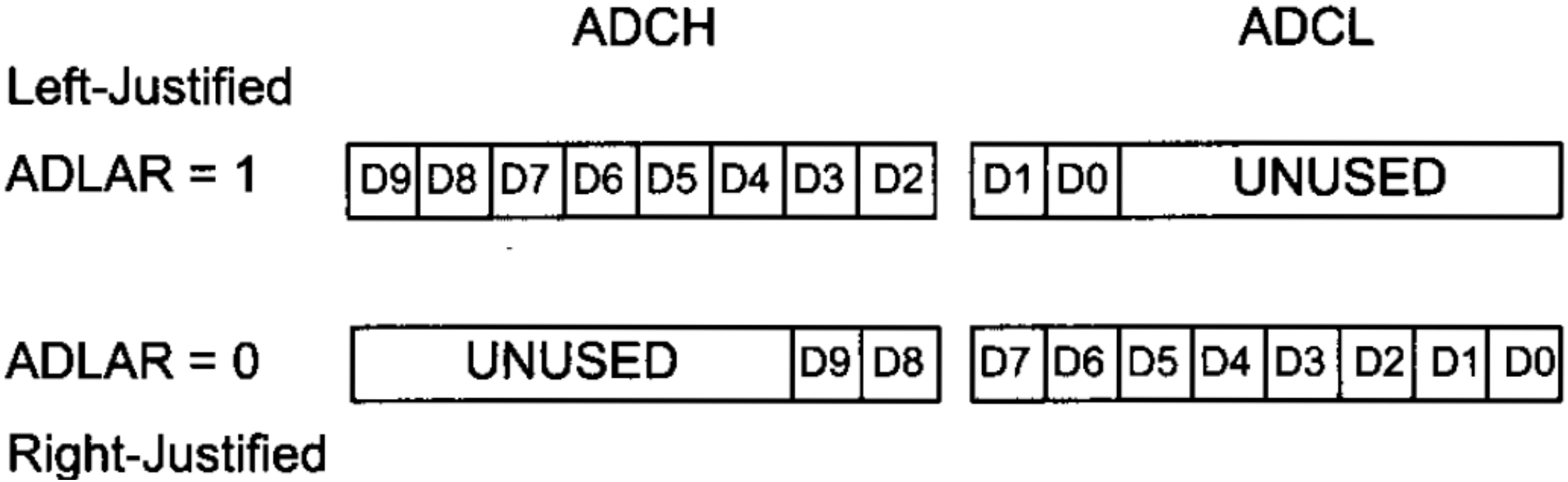
SELECTING INPUT SOURCE

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			
01000	N/A	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

SELECTING INPUT SOURCE

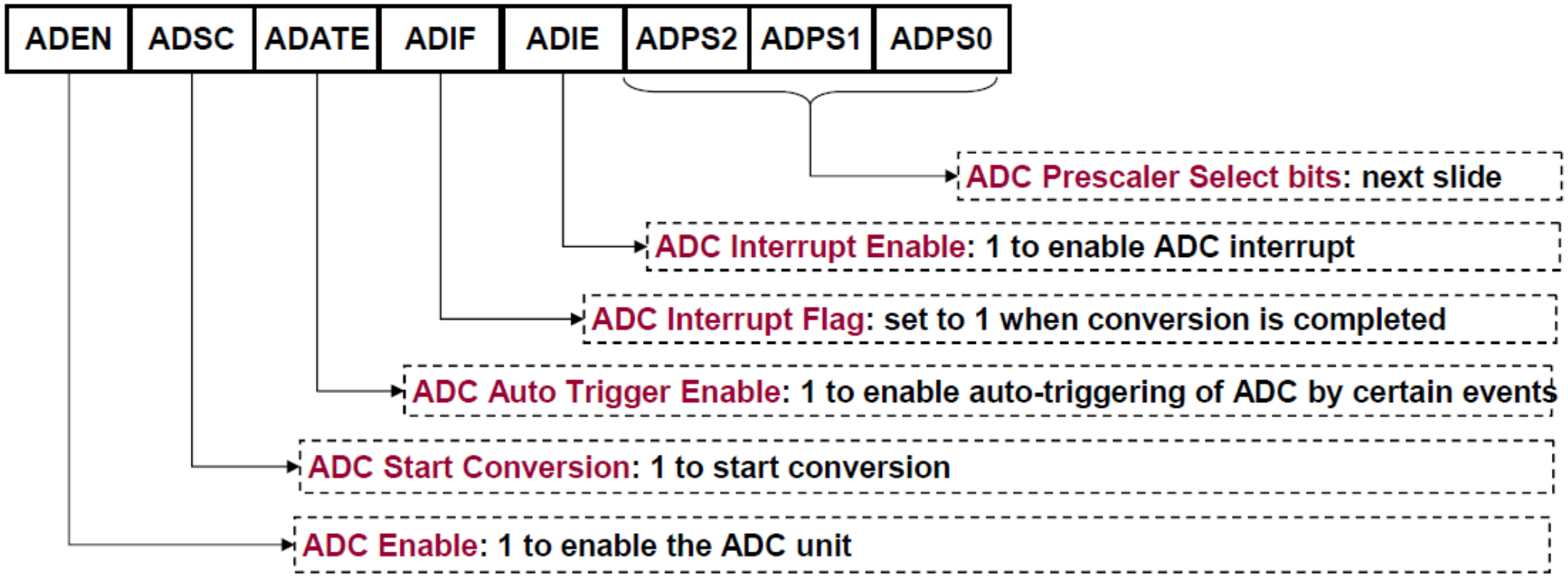
MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			

ADCH & ADCL REGISTERS



- **Important**
  - When retrieving digital output, register ADCL must be read first, before register ADCH.

ADC CONTROL AND STATUS REGISTER (ADCCSRA)



- ADC unit can operation in two modes: manual or auto-trigger.
- In manual mode, set bit ADSC will start conversion.
- In auto-trigger mode, an predefined event will start conversion.

ADC CLOCK

Table 11.3: ADC Prescaler Selection

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

- The clock of the ADC is obtained by dividing the CPU clock and a division factor.
- There are 8 possible division factors, decided by the three bits

  {ADPS2, ADPS1, ADPS0}

- Example: Using internal clock of 1Mz and a ADC prescaler bits of '010', the clock rate of ADC is: 1MHz/4 = 250Hz.

SPECIAL FUNCTION IO REGISTER (SFIOR)

ADC Auto Trigger Source bits

ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10

Table 11.4: ADC Auto Trigger Source

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

- Three bits in register SFIOR specify the event that will auto-trigger an A-to-D conversion.

STEPS TO USE THE ADC

- Step 1: Configure the ADC using registers ADMUX, ADCSRA, and SFIOR.
  - What is the ADC source?
  - What reference voltage to use?
  - Align left or right the result in ADCH, ADCL?
  - Enable or disable ADC auto-trigger?
  - Enable or disable ADC interrupt?
  - What is the prescaler?
- Step 2: Start ADC operation
  - Write 1 to flag ADSC of register ADCCSRA.
- Step 3: Extract ADC result
  - Wait until flag ADSC becomes 0.
  - Read result from registers ADCL and then ADCH.

SIMPLE EXAMPLE

Write C program that repeatedly performs ADC on a sinusoidal signal and displays the result on LEDs.

- Step 1: Configure the ADC
  - What is the ADC source? ADC0
  - What reference voltage to use? AVCC = 5V
  - Align left or right? Left, top 8-bit in ADCH
  - Enable or disable ADC auto-trigger? Disable
  - Enable or disable ADC interrupt? Disable
  - What is the prescaler? 2 (fastest conversion)

SIMPLE EXAMPLE

- Step 1: Configure the ADC
  - What is the ADC source? ADC0 (pin A.0)
  - What reference voltage to use? AVCC = 5V
  - Align left or right? Left, top 8-bit in ADCH
  - Enable or disable ADC auto-trigger? Disable
  - Enable or disable ADC interrupt? Disable
  - What is the prescaler? 4 (010)

0	1	1	0	0	0	0	0	
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADCMUX

1	0	0	0	0	0	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCCSRA

- Steps 2 and 3: Show next in C program.

```
#include<avr/io.h>
int main (void){
   unsigned char result;

   DDRB = 0xFF;  // set port B for output

   // Configure the ADC module of the ATmega16
   ADMUX = 0b01100000;  // REFS1:0 = 01    -> AVCC as reference,
                        // ADLAR   =  1    -> Left adjust
                        // MUX4:0  = 00000 -> ADC0 as input

   ADCSRA = 0b10000001; // ADEN = 1: enable ADC,
                        // ADSC = 0: don't start conversion yet
                        // ADATE = 0: disable auto trigger,
                        // ADIE  = 0: disable ADC interrupt
                        // ASPS2:0 = 001: prescaler = 2

   while(1){            // main loop
        // Start conversion by setting flag ADSC
        ADCSRA |= (1 << ADSC);

        // Wait until conversion is completed
        while (ADCSRA & (1 << ADSC)){;}

        // Read the top 8 bits, output to PORTB
        result = ADCH;

        PORTB = ~result;
   }
   return 0;
}
```

1

2

3

SIMPLE EXAMPLE: USING ADC INTERRUPT

Write interrupt-driven program to digitise a sinusoidal signal and display the result on LEDs.

- Step 1: Configure the ADC
  - What is the ADC source? ADC0
  - What reference voltage to use? AVCC = 5V
  - Align left or right? Left, top 8-bit in ADCH
  - Enable or disable ADC auto-trigger? Disable
  - Enable or disable ADC interrupt? Enable
  - What is the prescaler? 2 (fastest conversion)
- Step 2: Start ADC operation
- Step 3: In ISR, read and store ADC result.

```
#include<avr/io.h>
#include<avr/interrupt.h>

volatile unsigned char result;

ISR(ADC_vect){
    result = ADCH; // Read the top 8 bits, and store in variable result
}

int main (void){
    DDRB = 0xFF;  // set port B for output

    // Configure the ADC module of the ATmega16
    ADMUX = 0b01100000;  // REFS1:0 = 01    -> AVCC as reference,
                         // ADLAR   =  1    -> Left adjust
                         // MUX4:0  = 00000 -> ADC0 as input

    ADCSRA = 0b10001111; // ADEN = 1: enable ADC,
                         // ADSC = 0: don't start conversion yet
                         // ADATE = 0: diable auto trigger,
                         // ADIE  = 1: enable ADC interrupt
                         // ASPS2:0 = 002: prescaler = 2

    sei();               // enable interrupt system globally
    while(1){            // main loop
        ADCSRA |= (1 << ADSC); // start a conversion

        PORTB = ~result;       // display on port B
    }
    return 0;
}
```

3

1

2

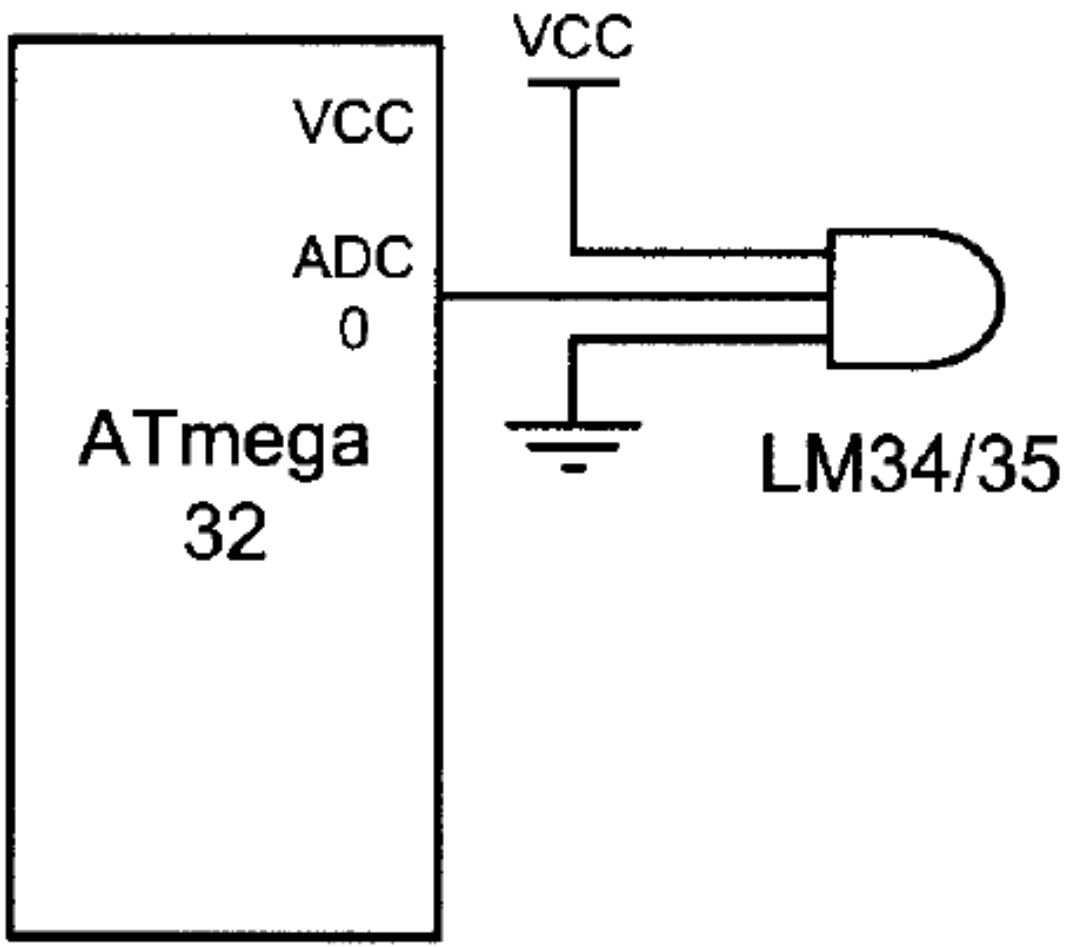
TEMPERATURE SENSOR

Part	Temperature Range	Accuracy	Output Scale
LM34A	–50 F to +300 F	+2.0 F	10 mV/F
LM34	–50 F to +300 F	+3.0 F	10 mV/F
LM34CA	–40 F to +230 F	+2.0 F	10 mV/F
LM34C	–40 F to +230 F	+3.0 F	10 mV/F
LM34D	–32 F to +212 F	+4.0 F	10 mV/F

TEMPERATURE SENSOR

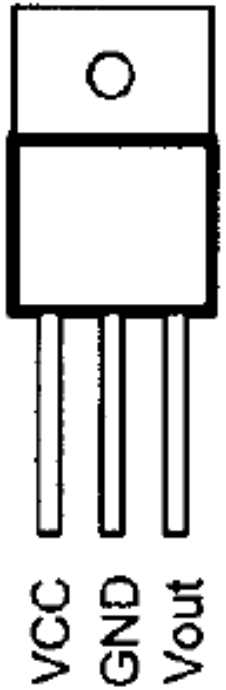
Part	Temperature Range	Accuracy	Output Scale
LM35A	–55 C to +150 C	+1.0 C	10 mV/C
LM35	–55 C to +150 C	+1.5 C	10 mV/C
LM35CA	–40 C to +110 C	+1.0 C	10 mV/C
LM35C	–40 C to +110 C	+1.5 C	10 mV/C
LM35D	0 C to +100 C	+2.0 C	10 mV/C

INTERFACING LM34 TO AVR



Bottom View
TO92
Package

Top View
TO220
Package



INTERFACING LM34 TO AVR

- If we use the internal 2.56 V reference voltage
  - step size would be 2.56 V/1024 = 2.5 mV
  - This makes the binary output of the ADC four times the real temperature
  - Divide the binary output by 4 to get the real temperature

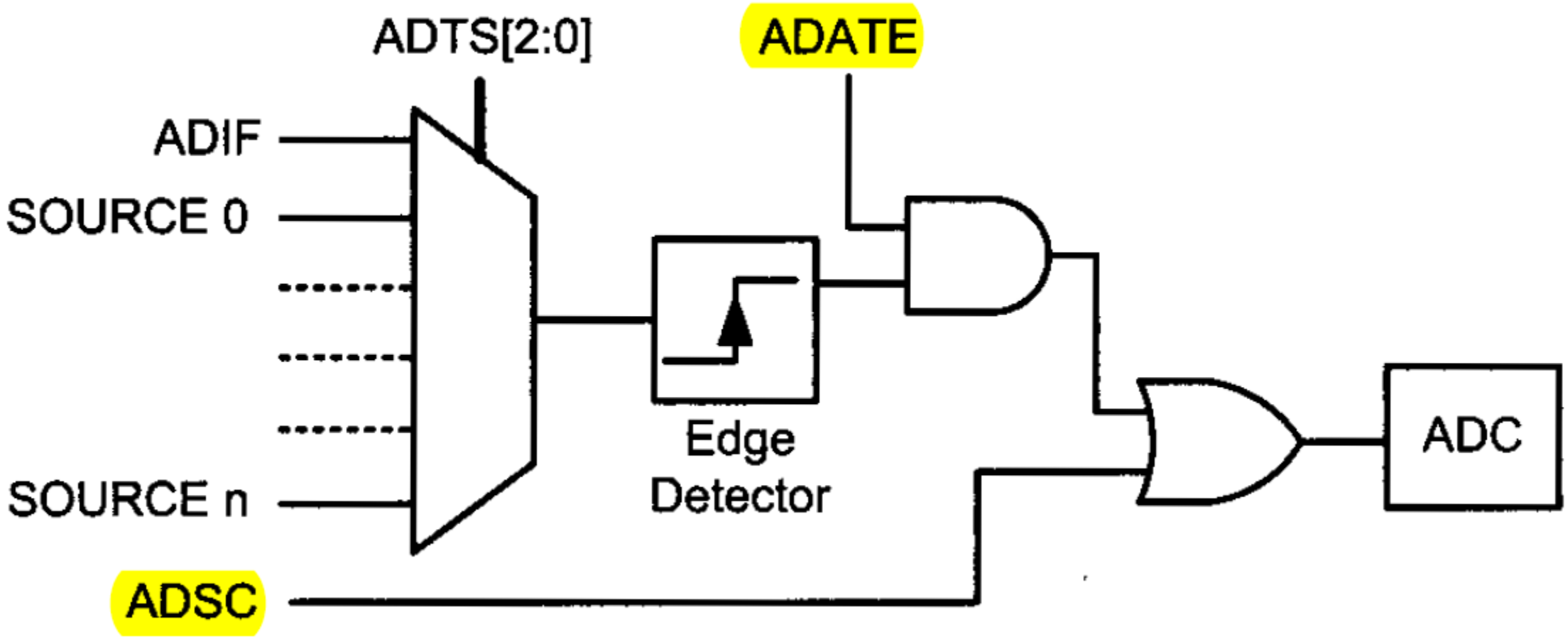
INTERFACING LM34 TO AVR

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

INTERFACING LM34 TO AVR

```
#include <avr/io.h> //standard AVR header
int main (void)
{
    DDRB = 0xFF;              //make Port B an output
    ADCSRA = 0x87;            //make ADC enable and select ck/128
    ADMUX = 0xE0;             //2.56 V  Vref and ADCO single-ended
                              //data will be left-justified
    while (1 ){
        ADCSRA |= (1<<ADSC);                  //start conversion
        while((ADCSRA & (1<<ADIF))==0);       //wait for end of conversion
        PORTB = ADCH;                         //give the high byte to PORTB
    }
    return 0;
}
```

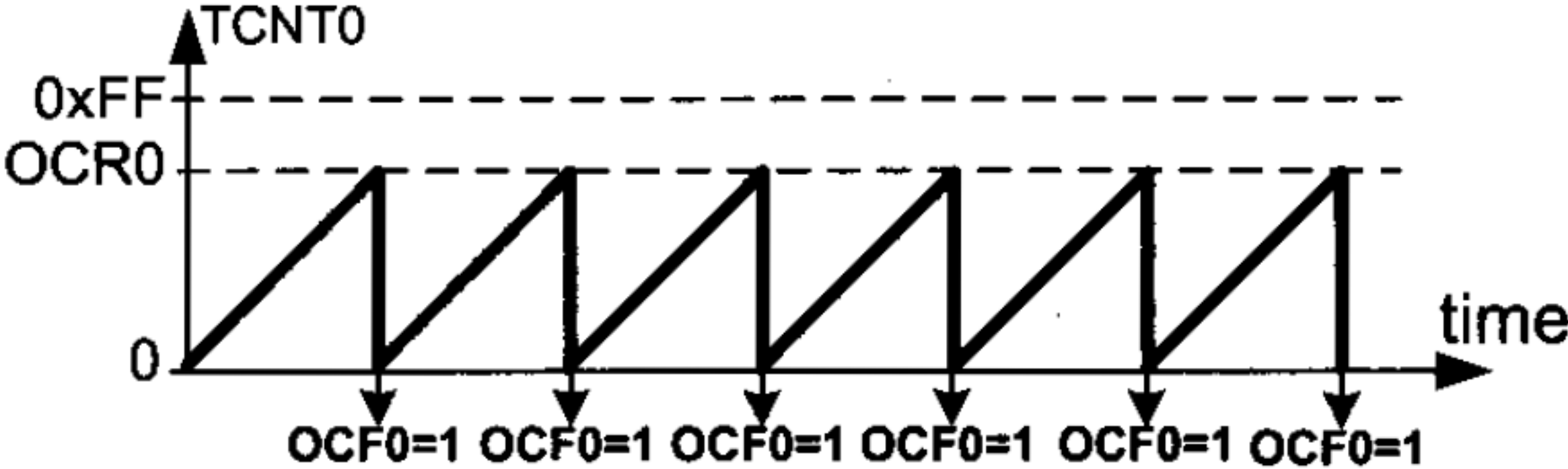
AUTO TRIGGER LOGIC



AUTO TRIGGER SOURCE

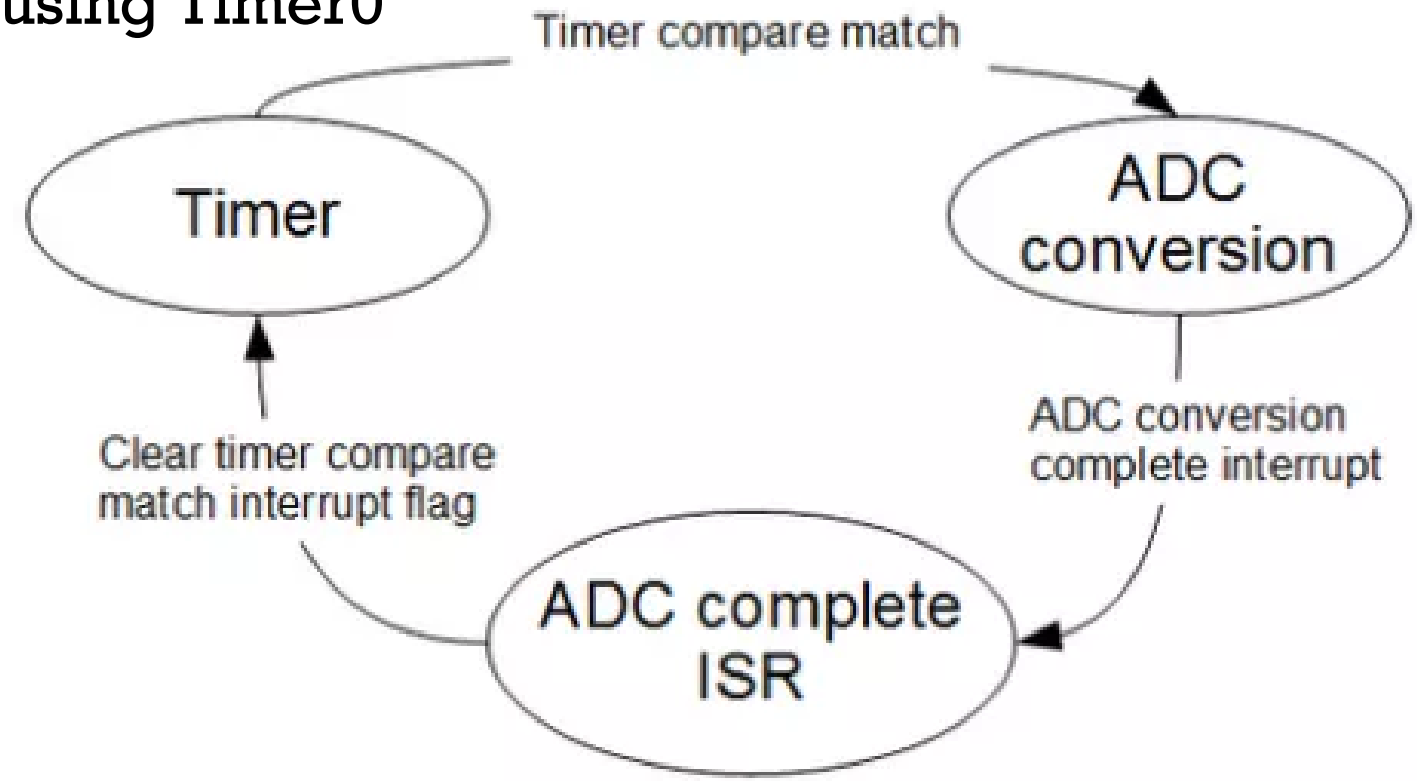
ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

TIMER: NORMAL VS. CTC MODE



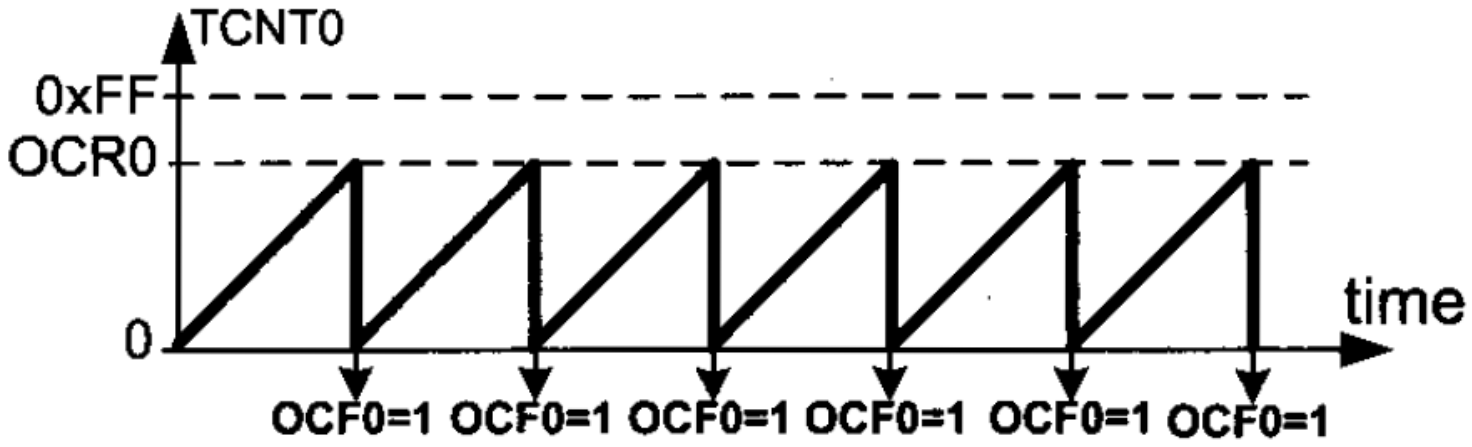
A PRACTICAL PROBLEM

- We need to sample an analog signal @ 20 kHz sampling frequency
- The only way to do this correctly
  - use auto-triggering with exact time intervals
- Let's see how can we do this using Timer0



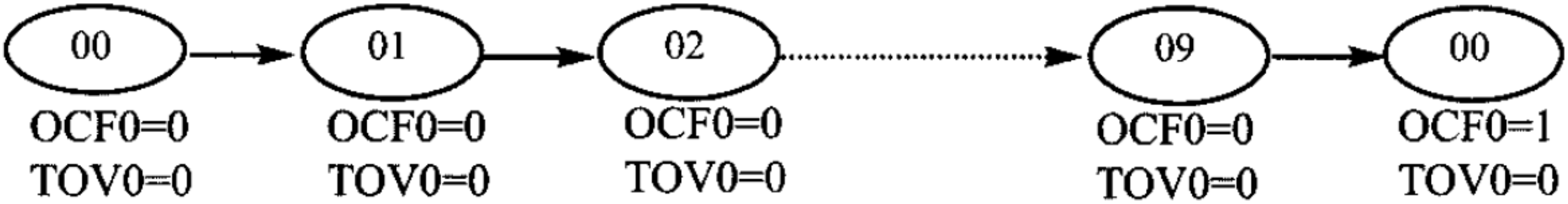
SOLUTION OVERVIEW

- Assume
  - system clock frequency = 16 MHz
  - Timer0 prescaler = 8
- To ensure 20 kHz sampling frequency
  - Time between successive ADC conversions?
    - 50us
  - Use Timer0 to trigger ADC every 50us
  - How to trigger Timer0 compare match every 50us?
    - Need to count?
      - 50us/(8/16000000) = 100
      - From 0 to 99
      - Set OCR0 = 99



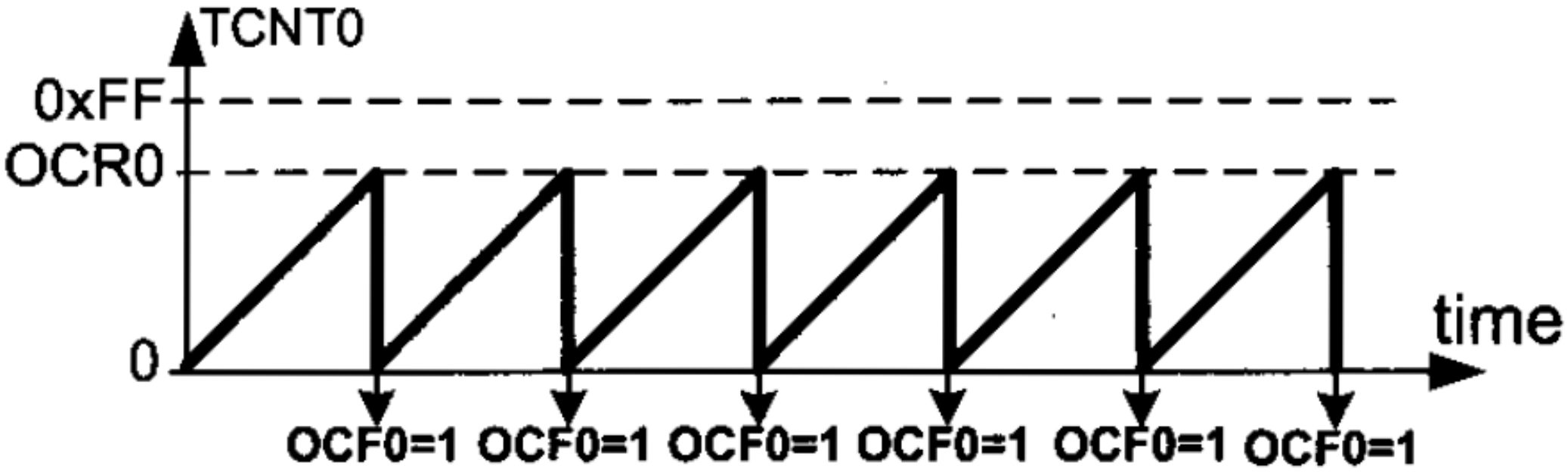
SOLUTION OVERVIEW

- Timer0 compare match trigger every 50us
  - Need to count?
  - 50us/(8/16000000) = 100
  - From 0 to 99
  - Set OCR0 = 99
  - Why 99 instead of 100 ?



SOLUTION OVERVIEW

- Now let's select ADC prescaler properly
  - ADC conversion must be completed before next Timer0 compare match occur
- One auto triggered ADC conversion time takes 13.5 ADC cycles



SOLUTION OVERVIEW

- Now let's select ADC prescaler properly
  - ADC conversion must be completed before next Timer0 compare match occur
- One auto triggered ADC conversion time takes 13.5 ADC cycles

Prescaller	ADC clock, kHz	ADC clock period, us	Total conversion time, us
4	4000	0.25	3.375
8	2000	0.5	6.75
16	1000	1	13.5
32	500	2	27
64	250	4	54
128	125	8	108

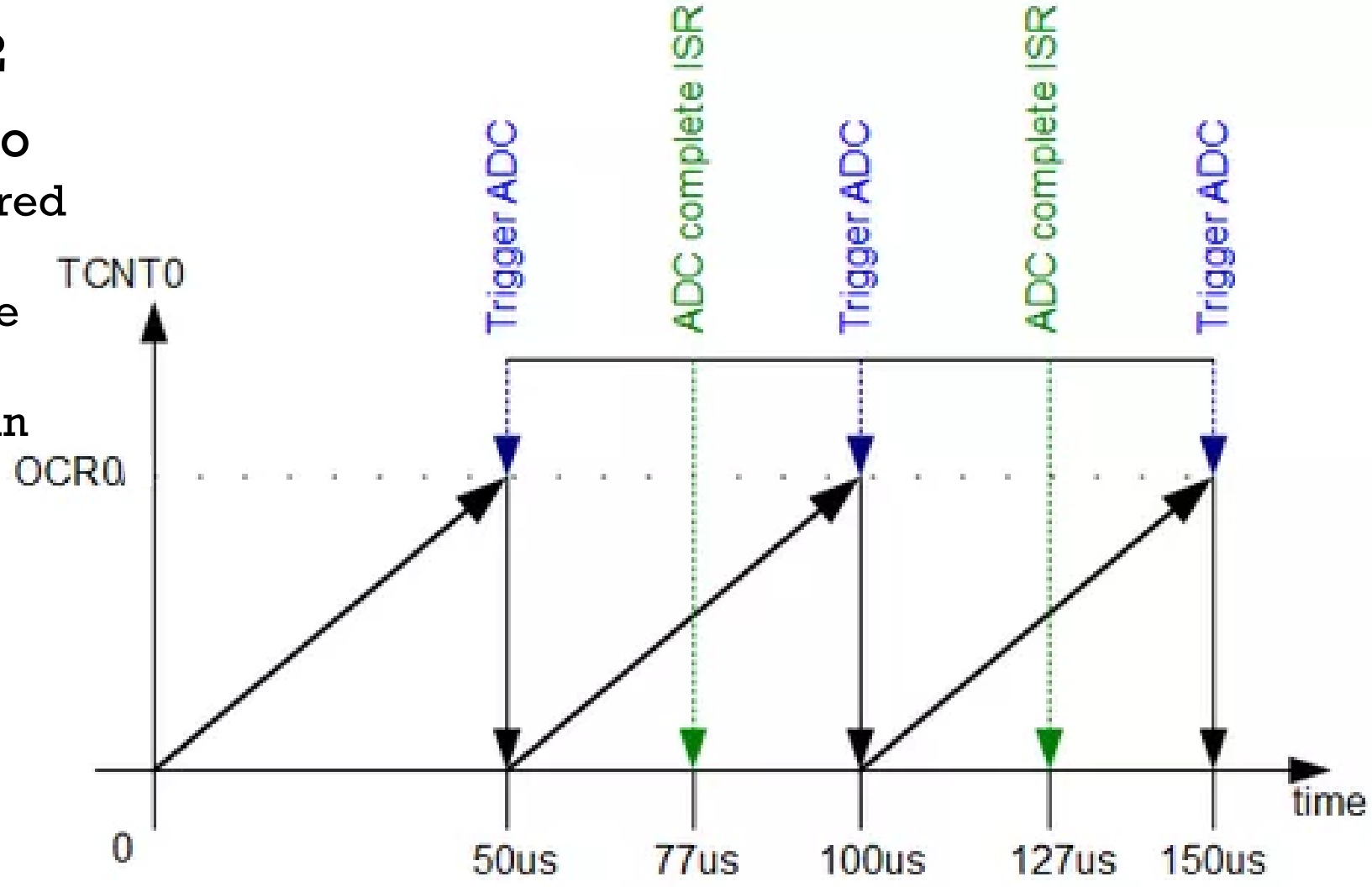
SOLUTION OVERVIEW

- To fit in to 50us window we can select prescaller up to 32
- Gives enough time to
  - Complete auto triggered ADC cycle
  - Clear Timer0 compare match flag
  - Store ADC value within one sample period
  - …

Prescaller	ADC clock, kHz	ADC clock period, us	Total conversion time, us
4	4000	0.25	3.375
8	2000	0.5	6.75
16	1000	1	13.5
32	500	2	27
64	250	4	54
128	125	8	108

SOLUTION OVERVIEW

- ADC prescaller = 32
- Gives enough time to
  - Complete auto triggered ADC cycle
  - Clear Timer0 compare match flag
  - Store ADC value within one sample period
  - …



COMPLETE CODE FOR ATMEGA328

- http://www.embedds.com/adc-on-atmega328-part-2/