Topic 11: Detect
Analog to Digital Conversion

CAB202. Topic 11

Feras Dayoub

The slides were prepared by Dr. Luis Mejias

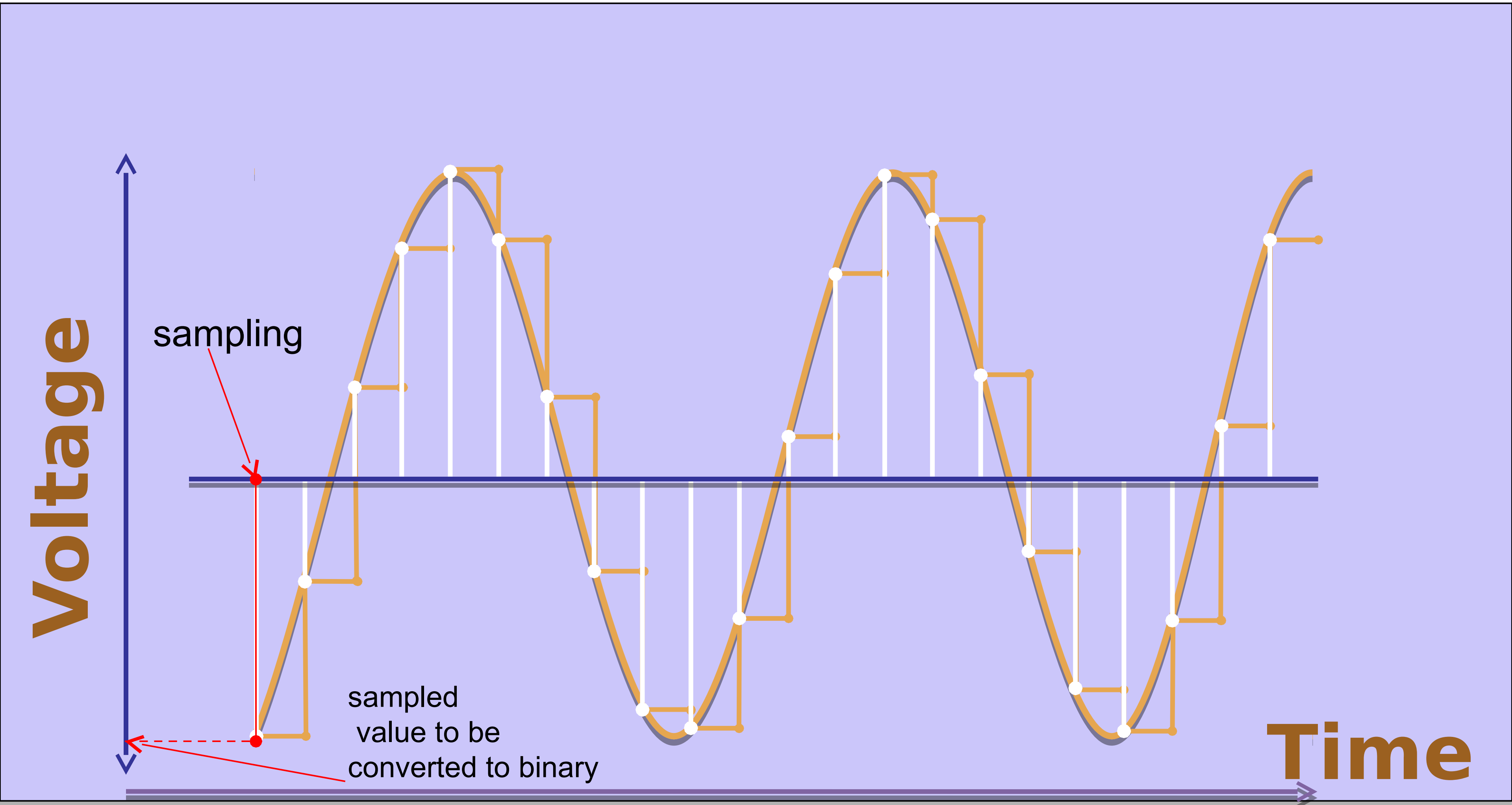
Outline

- Overview
- ADC on the Atmega32U4
- ADC Operation
- Examples

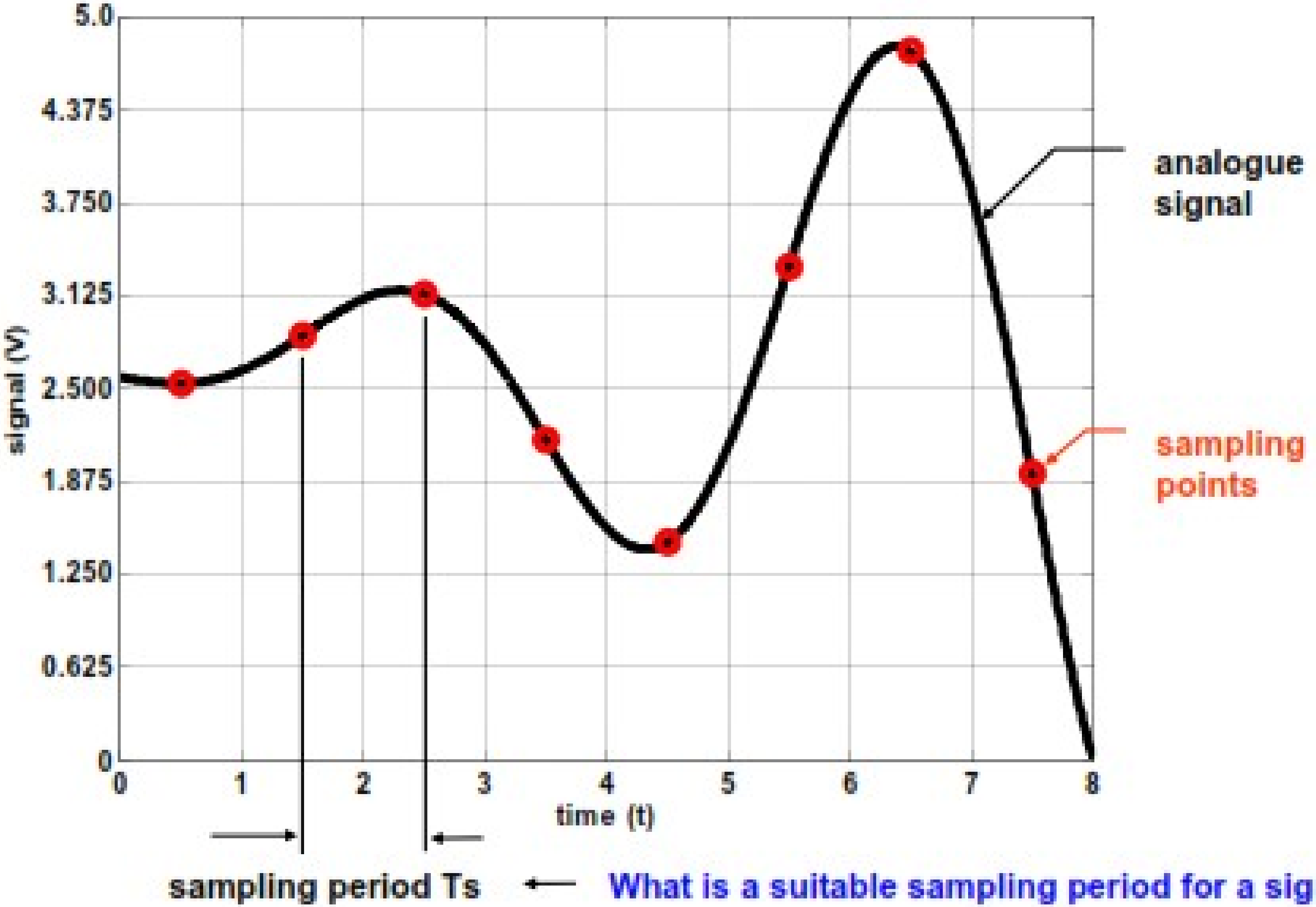
Analog Signals

- Most of the physical quantities around us are continuous.
- By continuous we mean that the quantity can take any value between two extremes.
- For example the atmospheric temperature can take any value (within certain range).
- If an electrical quantity is made to vary directly in proportion to this value (temperature, etc.) then what we have is an analog signal which in most cases is a voltage.
- We have to convert this into digital form if we want to manipulate it with a digital microcontroller.
- For this an ADC or analog to digital converter is needed.

The Analog to Digital Converter (ADC)



Sampling

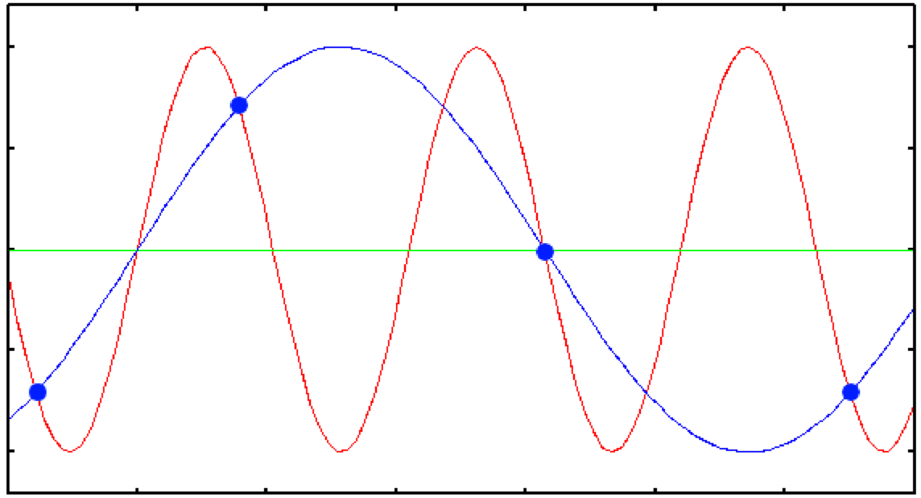


What is a suitable sampling period for a signal ?

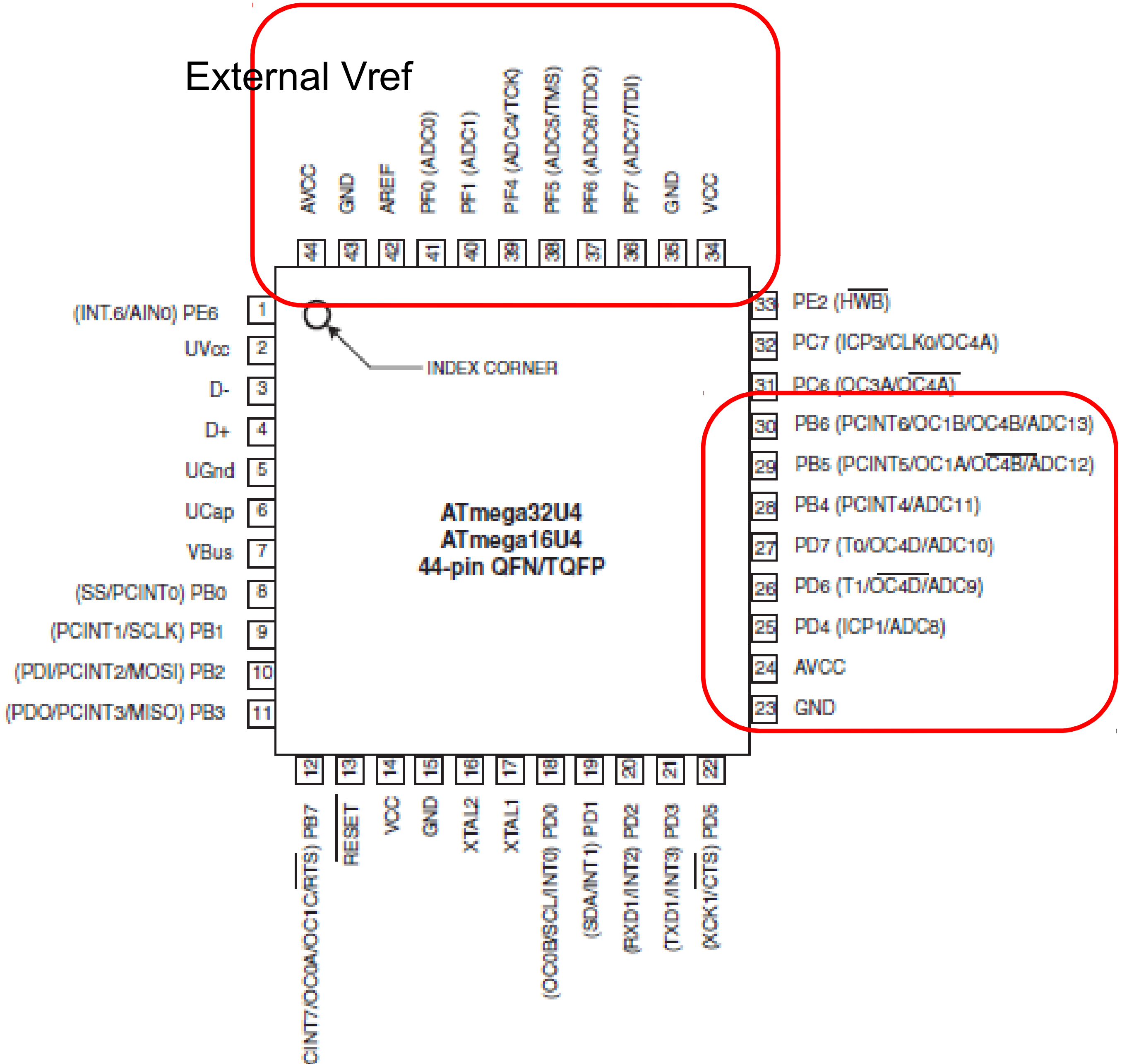
Nyquist Sampling Rate

- The Nyquist rate is the minimum sampling rate required to avoid aliasing, equal to twice the highest frequency contained within the signal.

  Nyquist Rate = 2 x f_{max}

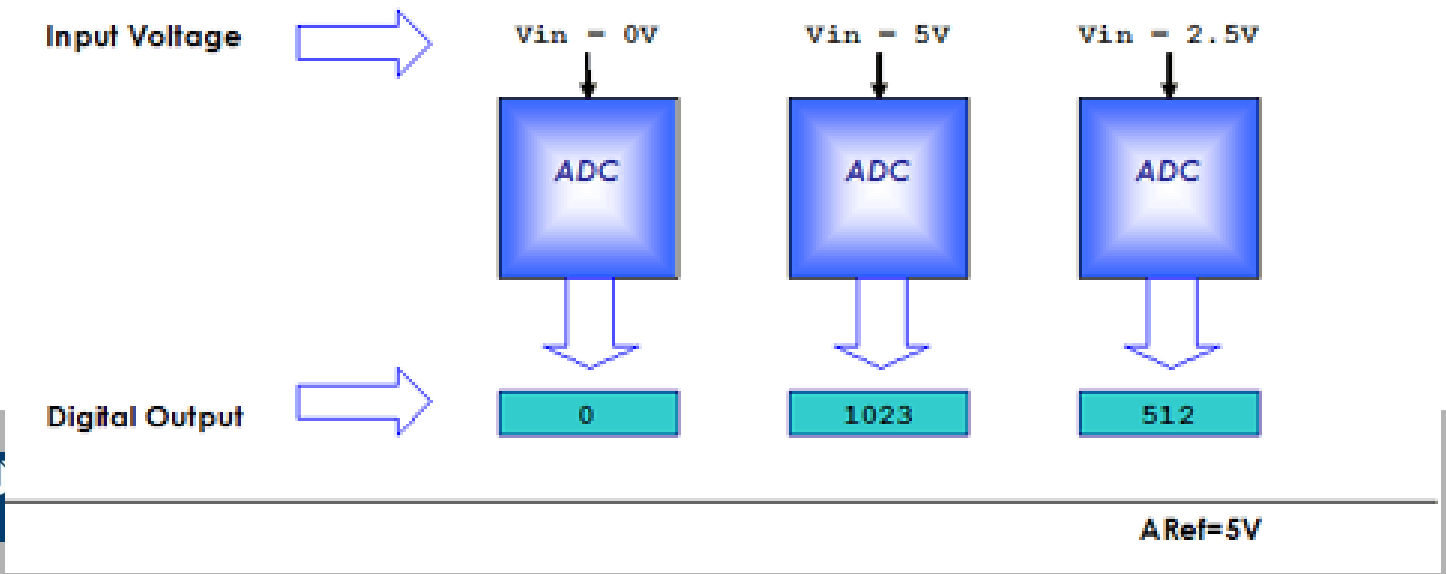


ADC Pins in ATMega32U4

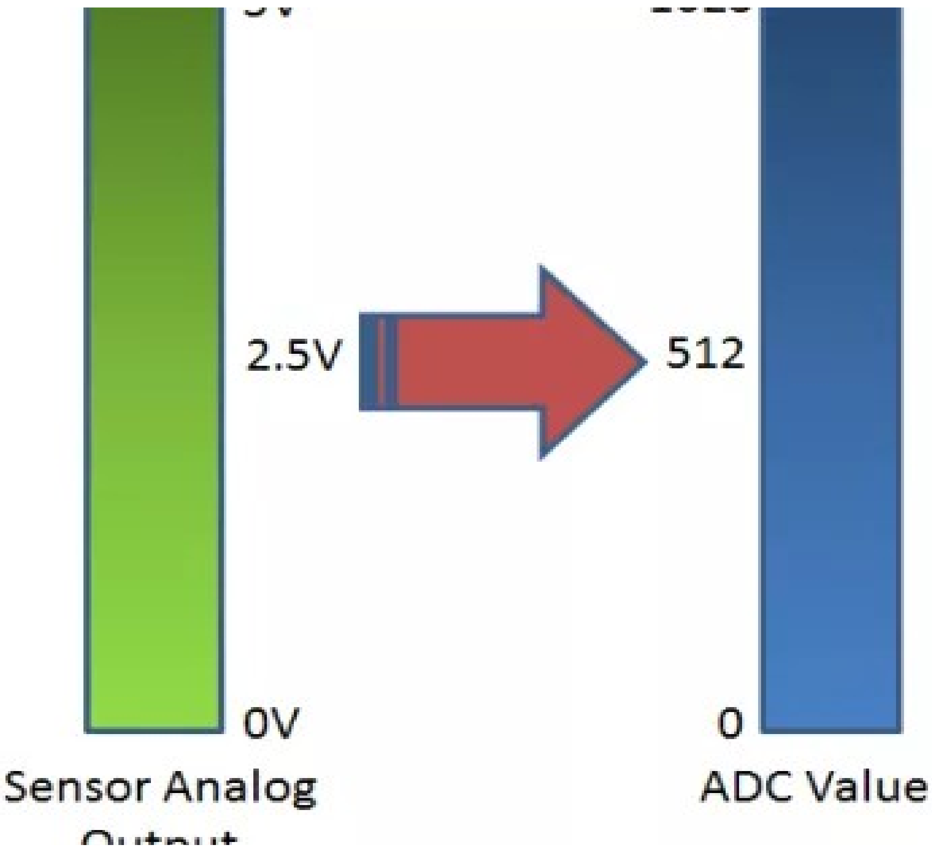


Resolution

- An ADC converts an input voltage into an integer number and therefore has a limited resolution.
- A 10 Bit ADC has a range from 0-1023. (2^10=1024)
- A reference voltage determines the max voltage which can be digitally converted.



Resolution



ATMEGA16/32

- 8 channels » 8 pins
- 10 bit resolution
- $2^{10} = 1024$ steps

ADC operation

- The ADC converts an analog input voltage to a 10-bit digital value through successive approximation.
- The ADC is enabled by setting the ADC Enable bit, in a register.
- Voltage reference and input channel selections will not go into effect until the correct bit is set.
- The ADC generates a 10-bit result which is presented in the ADC Data Registers (ADCH and ADCL).
- Conversion can be single or continuous.

ADC operation

- ADC prescaler
  - Analog signal is converted into digital signal at some regular interval.
  - This interval is determined by the clock frequency.
  - ADC operates within a frequency range of 50kHz to 200kHz.
  - CPU clock frequency is much higher (in the order of MHz).
  - So to achieve it, frequency division must take place. The prescaler acts as this division factor.

ADC operation

- ADC Registers – ADMUX, ADCSRA, ADCH, ADCL and SFIOR
  - Register are used to
    - Enable the ADC
    - Start the conversion
    - Select the channel and gain
    - Set Reference voltage
    - Trigger mode (single conversion or continuous)
    - Enable interrupt
    - Set prescaler
    - Etc..
  - So, by setting values in the registers we make possible the ADC process.

ADC operation: Setting the registers

- ADMUX – ADC Multiplexer Selection Register
  - Used to choose the reference voltage, left adjust the result and select channel

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	

Bits of interest are in yellow

ADC operation: Setting the registers

- ADMUX

Left adjust: see slide 18

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	

ADMUX = (1<<REFS0);
or
ADMUX = 0b 0 1 0 0 0 0 0 0;

Will select Vref=Vcc and channel 0 (ADC0)

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

MUX4..0	Single Ended Input	Positive Differential Input
00000	ADC0	N/A
00001	ADC1	
00010	ADC2	
00011	ADC3	
00100	ADC4	
00101	ADC5	
00110	ADC6	
00111	ADC7	
01000		ADC0
01001		ADC1
01010		ADC0
01011		ADC1
01100		ADC2
01101		ADC3
01110		ADC2
01111		ADC3
10000		ADC0
10001		ADC1

ADC operation: Setting the registers

- ADCSRA – ADC Control and Status Register A
  - Used to enable, start the conversion, select trigger mode, prescaler, etc.

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	

Bits of interest are in yellow

ADC operation: Setting the registers

- 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: it enables the ADC feature.
Bit 6 – ADSC – ADC Start Conversion – Write this to ‘1’ before starting any conversion.
Bit 5 – ADATE – ADC Auto Trigger Enable – Setting it to ‘1’ enables auto-triggering of ADC
Bit 4 – ADIF – ADC Interrupt Flag – Whenever a conversion is finished and the registers are updated, this bit is set to ‘1’ automatically. .
Bit 3 – ADIE – ADC Interrupt Enable – When this bit is set to ‘1’, the ADC interrupt is enabled. This is used in the case of interrupt-driven ADC.
Bits 2:0 – ADPS2:0 – ADC Prescaler Select Bits – The prescaler is determined by selecting the proper combination from the following.

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

```
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
// prescaler = 128
or
ADCSRA = 0b 1 0 0 0 0 1 1 1;
// prescaler = 128
```

So f_ADC=8 mhz / 128 = 62.5 khz (sampling frequency)

ADC operation: Setting the registers

- ADCL and ADCH – ADC Data Registers
  - The result of the ADC conversion is stored here. Since the ADC has a resolution of 10 bits, it requires 10 bits to store the result. Hence one single 8 bit register is not sufficient. We need two registers – ADCL and ADCH (ADC Low byte and ADC High byte) as follows. The two can be called together as ADC.

```
uint16_t adc_read(uint8_t ch)
{
   // select the corresponding channel 0~7

   // start single conversion
   // write '1' to ADSC

   // wait for conversion to complete
   // ADSC becomes '0' again


    //return conversion
   return (ADC);
}
```

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 = 0

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	

ADLAR = 1

Dive into ADC in C

```
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include "lcd.h"
#include "cpu_speed.h"
#include "sprite.h"
#include "graphics.h"

int main (void){
        long result;   //store conversion result
        char disp_buffer[32];   // store value to display on LCD
        char int_buffer[10];    // store ascii conversion

        set_clock_speed(CPU_8MHz); //set clock speed to 8MhZ
        lcd_init(LCD_DEFAULT_CONTRAST);   // initialize lcd
        // Configure the ADC module of the ATmega32U4
        ADMUX = 0b01000000;               // REFS1:0 = 01   -> AVCC as reference,
                                          // ADLAR   =  0   -> Left adjust
                                          // MUX4:0  = 00000 -> ADC0 as input

        ADCSRA = 0b10000111;              // ADEN = 1: enable ADC,
                                          // ADSC = 0: don't start conversion yet
                                          // ADATE = 0: disable auto trigger,
                                          // ADIE  = 0: disable ADC interrupt
                                          // ASPS2:0 = 111: prescaler = 128
        while(1){          // main loop
                // Start conversion by setting flag ADSC
                ADCSRA |= (1 << ADSC);

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

                // Read conversion outcome and copy value in results
                result = ADC;
                //clear display pixels
                clear_screen();
                // convert integer to ascii
                itoa(result, int_buffer, 10);
                // copy value to display variable
                sprintf(disp_buffer,"ADC = %s",int_buffer);
                // copy values to lcd
                 draw_string(0,0,disp_buffer);
                 //display pixels
                 show_screen();
        }
        return 0;
}
```

Dive into ADC in C: DEMOS

- Examples
  - simple_adc.c
    - reads ADC value and show result on the LCD
  - sprite_adc.c
    - Reads ADC value and move a character on the LCD screen

Summary

- ADC is a key operation when using microcontrollers and sensors
- Variables used to store ADC conversion values should be properly declared (10bits)
- Example code Topic_11.zip (in Blackboard)