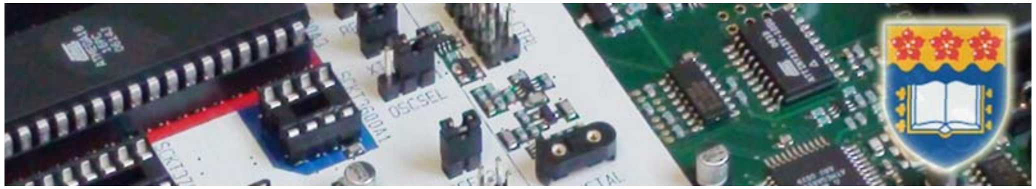


# ECTE333
# Lecture 11 - Analogue-to-Digital Converter

School of Electrical, Computer and Telecommunications Engineering
University of Wollongong
Australia

## ECTE333's schedule

| Week | Lecture (2h) | Tutorial (1h) | Lab (2h) |
|---|---|---|---|
| 1 | L7: C programming for the ATMEL AVR | | |
| 2 | | Tutorial 7 | Lab 7 |
| 3 | L8: Serial communication | | |
| 4 | | Tutorial 8 | Lab 8 |
| 5 | L9: Timers | | |
| 6 | | Tutorial 9 | Lab 9 |
| 7 | L10: Pulse width modulator | | |
| 8 | | Tutorial 10 | Lab 10 |
| 9 | L11: Analogue-to-digital converter | | |
| 10 | | Tutorial 11 | Lab 11 |
| 11 | L12: Revision lecture | | |
| 12 | | | Lab 12 |
| 13 | L13: Self-study guide (no lecture) | | |
| | *Final exam (25%), Practical exam (20%), Labs (5%)* | | |



## Lecture 11's sequence

11.1 Introduction to analogue-to-digital conversion

↓

11.2 Analogue-to-digital converter in ATmega16

↓

11.3 Example application of ADC

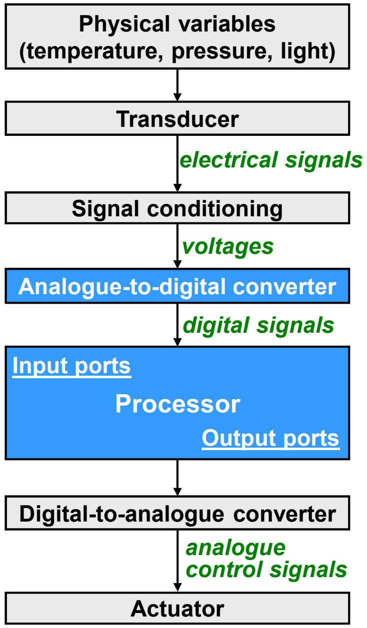
[Lab11: Task 1 | Task 2]



## 11.1 Introduction to A-to-D conversion

- An ADC samples an analogue signal at discrete times, and converts the sampled signal to digital form.
- Used with transducers, ADCs allow us to monitor real-world inputs and perform control operations based on these inputs.
- Many dedicated ICs are made for ADC, e.g.
  - ADC0804: 8-bit, successive approximation.
  - Maxim104: 8-bit, flash type.

## A-to-D conversion: Typical embedded application




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



## A-to-D conversion: Example applications

- **Local positioning sensor for object tracking**
  [ECTE457 Project]
  - Measure the distance between FM transmitter/receiver.
  - The receiver has an RSSI output (Receiver Signal Strength Indicator)
  - The RSSI voltage is inversely proportional to the squared distance.
- **Temperature sensor for shower water**
  [ECTE350 Third-year Group Project, 3rd prize]
  - Measure the temperature of shower water.
  - Control hot/cold water valves.
  - Use a thermistor as sensor.



## A-to-D conversion: Example applications

- **Obstacle sensor & audio cues in the cane for the blind**
  [ECTE250 Second-year Group Project, 1st prize]
  - Measure distance to nearest object with an ultrasonic sensor.
  - The sensor output is digitized using the ADC.
- **Electric fence monitoring**
  [ECTE350 Third-year Group Project]
  - Determine if a electric fence is being tampered.
  - Measure the voltage level of an electric fence.



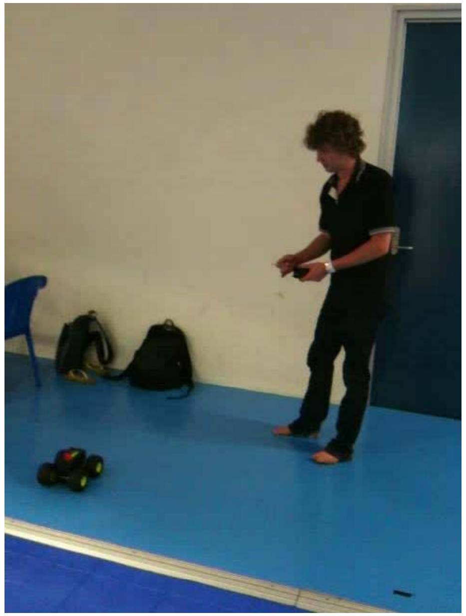
## A-to-D conversion: Example applications

- **Wireless irrigation system**
  [ECTE350 Third-year Group Project, 1st prize]
  - Measure the moisture of the soil with resistor & ADC.
  - Transmit data wirelessly to base station
  - Turn ON or OFF the sprinklers.
- **Intelligent clothesline**
  [ECTE350 Third-year Group Project]
  - Use a set of sensors to measure humidity, temperature, wind speed.
  - Open or close the clothesline cover to protect against rain.

# A-to-D conversion: Example applications

- Car control using 3-D accelerometers

  [ECTE350 Third-year Group Project in 2010, 2nd prize]





# A-to-D conversion: The process

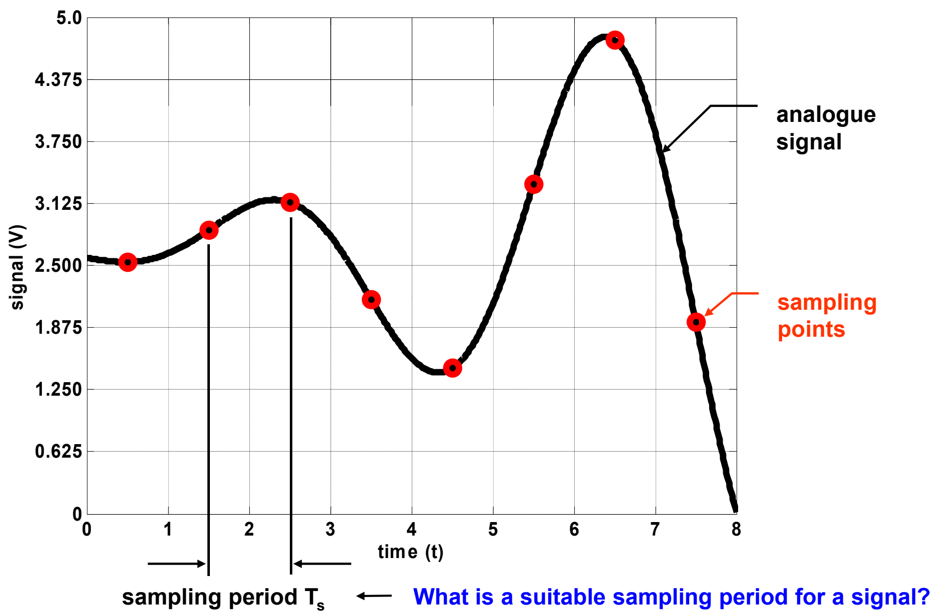
- There are two related steps in A-to-D conversion:
  - sampling,
  - quantisation.

- Sampling:
  - The analogue signal is extracted, usually at regularly-spaced time instants.
  - The samples have real values.

- Quantisation:
  - The samples are quantized to discrete levels.
  - Each sample is represented as a digital value.



# Sampling an analogue signal






# The sampling theorem

An analogue signal $x(t)$ with frequencies of no more than $F_{max}$ can be reconstructed exactly from its samples if the sampling rate satisfies: $F_s \geq 2 \times F_{max}$.

**Significance**

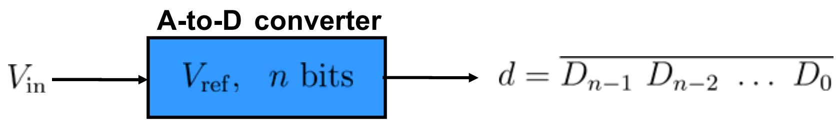
- If maximum frequency of the signal is $F_{max}$, the sampling rate should be at least:

$$\text{Nyquist rate} = 2 \times F_{\max}$$

- If the sampling rate is $F_s$, the maximum frequency in the signal must not exceed:

$$\text{Nyquist frequency} = \tfrac{1}{2} F_s$$

# Quantising the sampled signal




- Consider an n-bit ADC.
- Let $V_{ref}$ be the reference voltage.
- Let $V_{in}$ be the analog 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

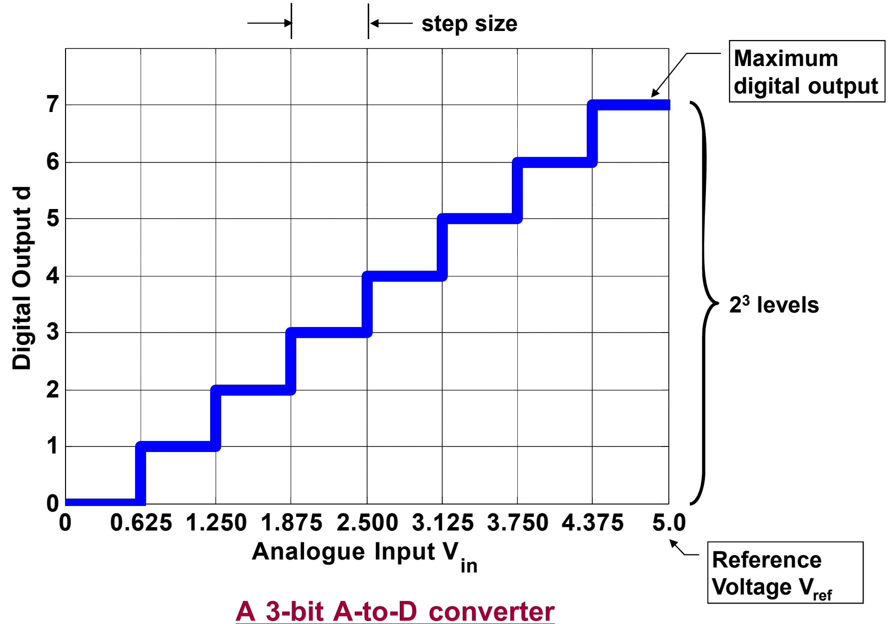
$$d = \text{round down}\left[\frac{V_{in} - V_{min}}{\text{step size}}\right]$$

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

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



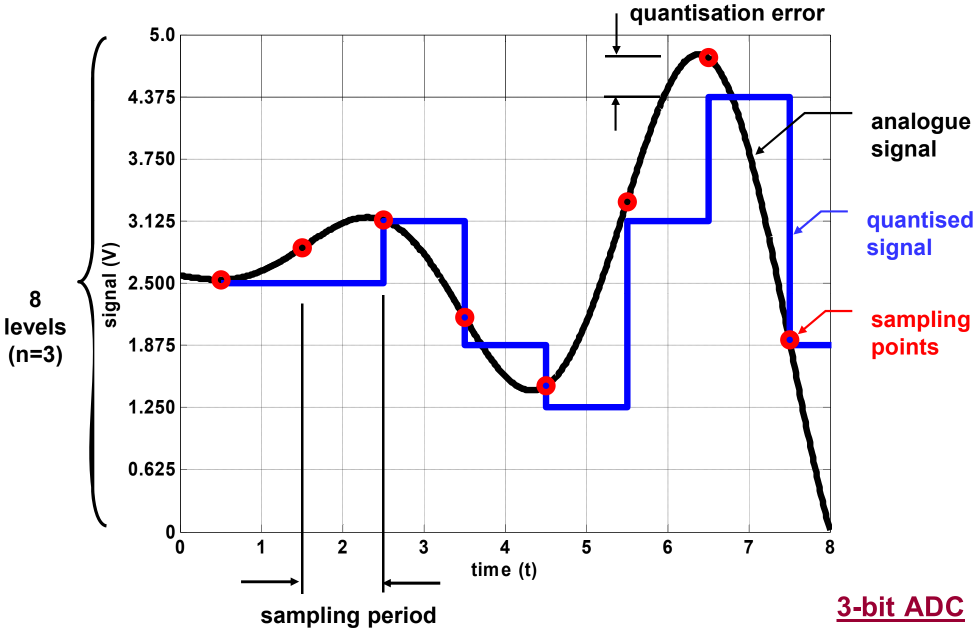
# Quantising the sampled signal




A 3-bit A-to-D converter



# Quantising the sampled signal




3-bit ADC



# A-to-D converter: Parameters

- Number of bits n: The higher is the number of bits, the more precise is the digital output.
- Quantization error $E_q$: The average difference between the analogue input and the quantised value. The quantisation error of an ideal ADC is half of the step size.
- Sample time $T_{sample}$: A sampling capacitor must be charged for a duration of $T_{sample}$ before conversion taking place.
- Conversion time $T_{conv}$: Time taken to convert the voltage on the sampling capacitor to a digital output.

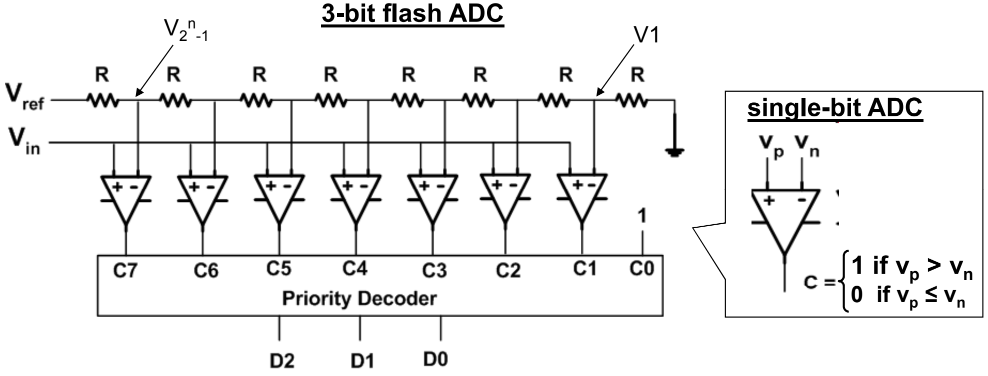
# A-to-D converter: Designs

- There are many designs for analogue-to-digital converters.
- We'll consider briefly two common designs.
  - Flash ADC.
  - Successive-approximation ADC.



# Flash ADC

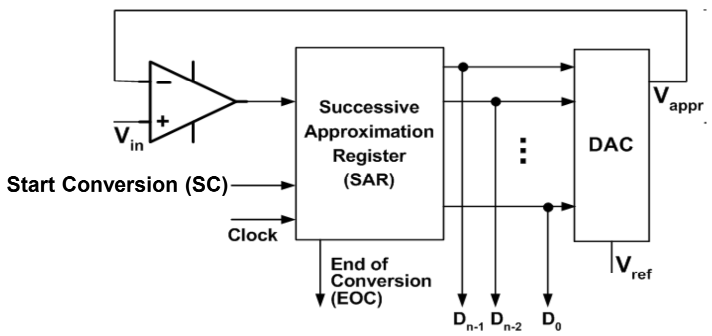
**3-bit flash ADC**




- An $n$-bit flash ADC uses $(2^n\text{-}1)$ comparators, and a priority decoder.
- **Advantage**: the fastest type of ADC.
- **Disadvantages**: limited resolution, expensive, and large power consumption.



# Successive-approximation ADC

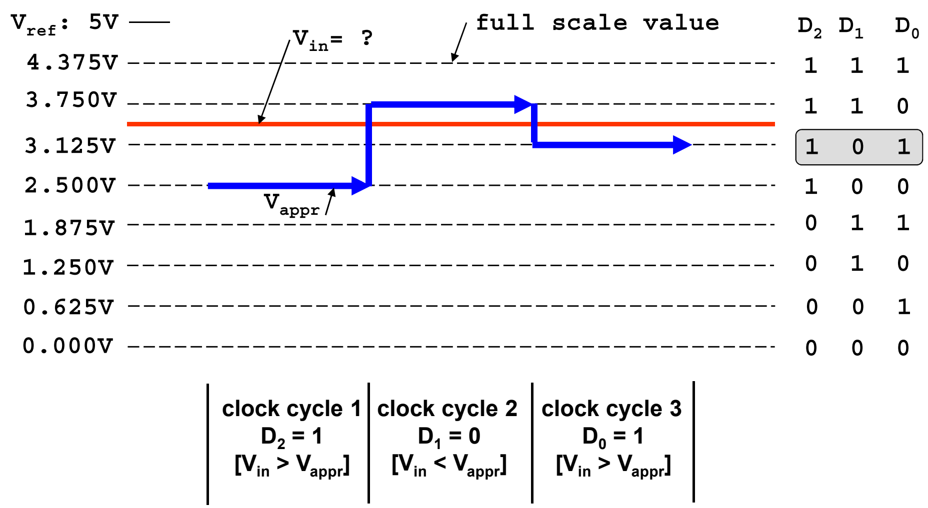



- A DAC is used to generate approximations of the input voltage.
- A comparator is used to compare $V_{in}$ and $V_{appr}$.
- In each cycle, SAR finds one output bit using comparator.
- To start conversion, set SC = 1. When conversion ends, EOC = 1.
- Quite fast, one of the most widely used design for ADCs.



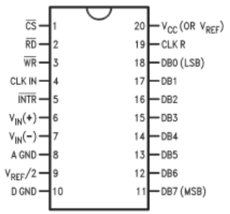
# Successive-approximation ADC

**Binary search for a 3-bit ADC**



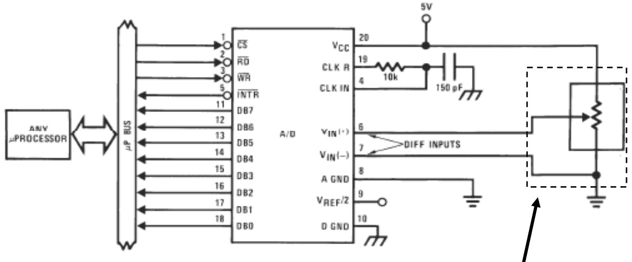
# Example IC for ADC

**ADC0804**
(National Semiconductor)




- $V_{CC}$: reference voltage
- RD: to read digital output
- WR: to start a new conversion
- INTR: when conversion completes
- $V_{IN}(+)$, $V_{IN}(-)$: analogue input
- DB0-DB7: 8-bit output
- CLK IN, CLK R: clock signal

**A typical application of ADC0804**




transducer



# AVR Demo: ATmega128L and Color LCD

ATmega128L, color LCD, temperature sensor
www.youtube.com/watch?v=m6mtCak3krE



# Lecture ~~11's~~ sequence

11.1 Introduction to analogue-to-digital conversion

11.2 Analogue-to-digital converter in ATmega16

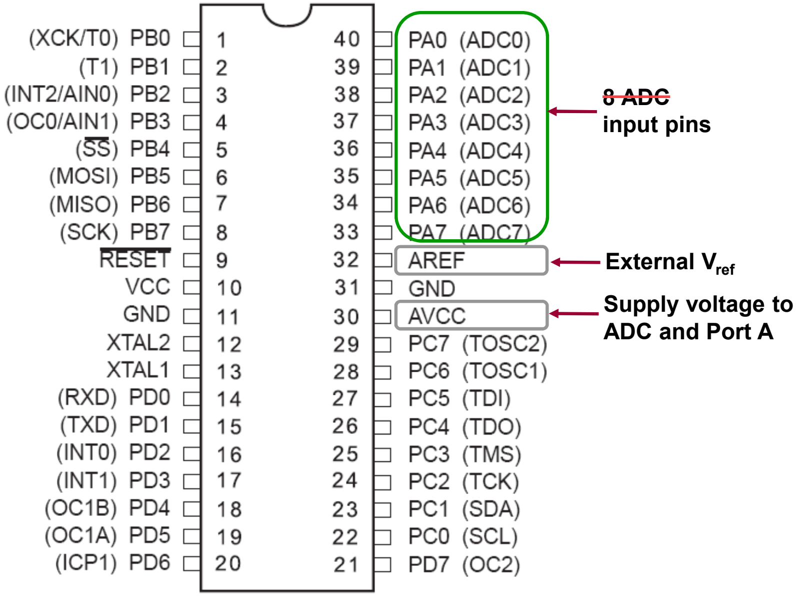
11.3 Example application of ADC



# 11.2 The ADC in ATmega16

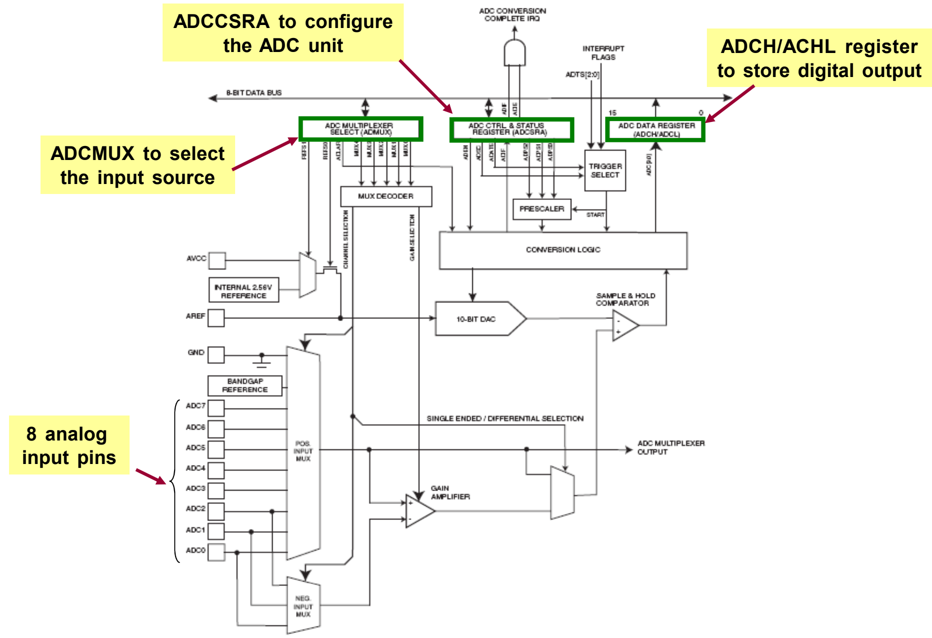
- The ADC in ATmega16 has a 10-bit resolution.
  - The digital output has n = 10 bits.
- The ADC has 8 input channels.
  - Analog input can come from 8 different sources.
  - However, it performs conversion on only one channel at a time.
- If default reference voltage $V_{ref}$ = 5V is used.
  - step size: 5(V)/1024 (steps) = ± 4.88mV.
  - accuracy: 2 × LSB = ± 9.76mV.
- 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 actual ADC clock rate.

# ADC unit — Relevant pins






# ADC unit — Block diagram






# ADC unit

## We focus on the major aspects of the ADC unit.

11.2.1 What are the relevant ADC registers?

a) ADCMUX

b) ADCH/ADCL

c) ADCCSRA

d) SFIOR

11.2.2 What are the steps to use the ADC?

11.2.3 How to use the ADC interrupt?



# 11.2.1a ADC Multiplexer Selection Register (ADCMUX)

| REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 |
|---|---|---|---|---|---|---|---|

MUX4-0: Select analog channel and gain

ADC Left Adjust Result: 1 to left adjust the ADC result

Reference Selection bits: To select reference voltage

- Reference voltage $V_{ref}$ can be selected among 3 choices. [Slide 29]
- Analog input voltage can be selected among different pins. Differential input and custom gain factor can also be chosen. [Slide 30]
- ADLAR flag determines how the 10-bit digital output is stored in output registers. [Slide 31]

## 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.



## Selecting input source and gain factor

| MUX4..0 | Single Ended Input | Positive Differential Input | Negative Differential Input | Gain |
|---|---|---|---|---|
| 00000 | ADC0 | | | |
| 00001 | ADC1 | | | |
| 00010 | ADC2 | | | |
| 00011 | ADC3 | N/A | | |
| 00100 | ADC4 | | | |
| 00101 | ADC5 | | | |
| 00110 | ADC6 | | | |
| 00111 | ADC7 | | | |
| 01000 | | ADC0 | ADC0 | 10x |
| 01001 | | ADC1 | ADC0 | 10x |
| 01010(1) | | ADC0 | ADC0 | 200x |
| 01011(1) | | ADC1 | ADC0 | 200x |
| 01100 | | ADC2 | ADC2 | 10x |
| 01101 | | ADC3 | ADC2 | 10x |
| 01110(1) | | ADC2 | ADC2 | 200x |
| 01111(1) | | ADC3 | ADC2 | 200x |
| 10000 | | ADC0 | ADC1 | 1x |
| 10001 | | ADC1 | ADC1 | 1x |
| 10010 | N/A | 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 |

Table 11.2: ADC input source.

- Analog input voltage can be selected as
  - 8 ADC pins ADC7 to ADC0,
  - the differential input between two of ADC pins.
- A gain factor of 1, 10 or 200 can be selected for differential input.



## 11.2.1b ADC Left Adjust flag and ADCH/L registers

When bit ADLAR = 0 (Right Aligned)

| ADCH | | | | | | | | ADCL | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | # | # | # | # | # | ADC9 | ADC8 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 |

When bit ADLAR = 1 (Left Aligned)

| ADCH | | | | | | | | ADCL | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADC9 | ADC8 | ADC7 | ADC6 | ADC5 | ADC4 | ADC4 | ADC2 | ADC1 | ADC0 | # | # | # | # | # | # |

- Digital output is stored in two 8-bit registers ADCH and ADCL.
- The format of ADCH and ADCL are interpreted differently depending on flag ADLAR.
- Important: When retrieving digital output, register ADCL must be read first, before register ADCH.



## 11.2.1c ADC Control and Status Register (ADCSRA)

| ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 |
|---|---|---|---|---|---|---|---|

- ADPS2, ADPS1, ADPS0 → ADC Prescaler Select bits: next slide
- ADIE → ADC Interrupt Enable: 1 to enable ADC interrupt
- ADIF → ADC Interrupt Flag: set to 1 when conversion is completed
- ADATE → ADC Auto Trigger Enable: 1 to enable auto-triggering of ADC by certain events
- ADSC → ADC Start Conversion: 1 to start conversion
- ADEN → ADC Enable: 1 to enable the ADC unit

- ADC unit can operate in two modes: manual or auto-trigger.
- In manual mode, setting flag ADSC = 1 will start conversion.
- In auto-trigger mode, a 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.



# 11.2.1d Special Function IO Register (SFIOR)

ADC Auto Trigger Source

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



# 11.2.2 Steps to use the ADC

- Step 1: Configure the ADC using registers ADMUX, ADCSRA, 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 ADC prescaler?
- Step 2: Start ADC operation
  - Write 1 to flag ADSC (register ADCCSRA).
- Step 3: Extract ADC result
  - Wait until flag ADSC becomes 0.
  - Read result from registers ADCL and then ADCH.



# Example 11.1: Performing ADC

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? ADC1
  - 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 ADC pre-scaler? 2 (fastest conversion)

# Example 11.1: Performing ADC

- Step 1: Configure the ADC
  - What is the ADC source? ADC1 (pin A.1)
  - 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 (010)

| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | |
|---|---|---|---|---|---|---|---|---|
| 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 | ADCSRA |

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

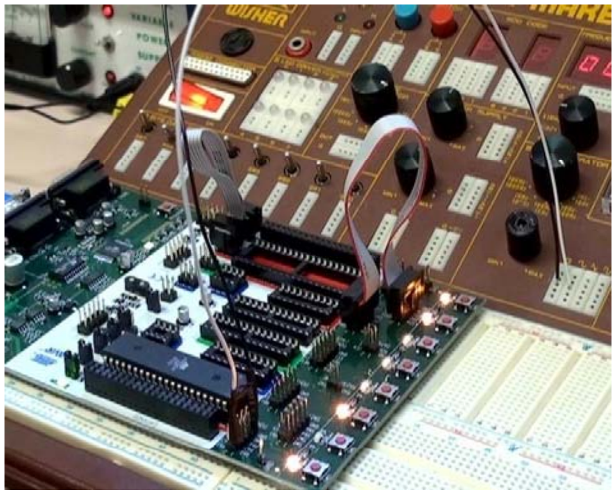


# Example 11.1: adc.c

```c
#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;
}
```



# Example 11.1: Testing



Video demo: [avr]/ecte333/adc.mp4



# 11.2.3 Using ADC interrupt

- In the polling approach shown previously, we must check ADSC flag to know when an ADC operation is completed.
- Alternatively, the ADC unit can trigger an interrupt when ADC is done.
- We enable ADC interrupt through ADIE flag in register ADCCSRA.
- In the ISR, we can write code to read registers ADCL and ADCH.
- ADC interrupt is usually combined with auto-trigger mode [~~Tutorial 11~~].

# Example 11.2: 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.



# Example 11.2: adc_int.c

```c
#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



# Lecture 11's sequence

11.1 Introduction to analog-to-digital conversion

↓

11.2 Analog-to-digital converter in ATmega16

↓

11.3 Example application of ADC



# 11.3 Example application of the ADC

- This section presents an application of the ADC in the ATmega16.

- A joystick is used to move a camera up/down/left/right, as in a security console.
- The pan-tilt camera is mCAM100x (Lecture 8).

- It can be moved by sending a character '4', '6', '8', or '2' via a serial connection: 9600bps, 8 data bits, 1 stop bit, no parity.

# Joystick

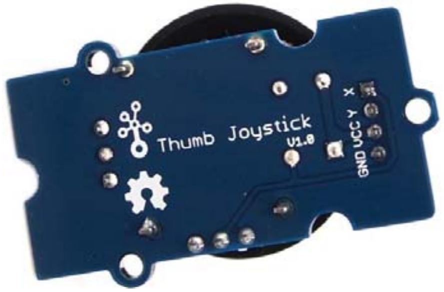
- The joystick we use is Grove SS-COM90133P, by Seed Studio.

- The joystick has 4 pins
  - GRD: ground
  - VCC: 5V-supply
  - X: x coordinate, analogue voltage between [0, 5V]
  - Y: y coordinate, analogue voltage between [0, 5V]

- URLs: littlebirdelectronics.com/products/grove-thumb-joystick
  www.seeedstudio.com/wiki/Grove_-_Thumb_Joystick



# Joystick




| Description | Min | Typical | Max |
|---|---|---|---|
| Working voltage VCC | 4.75V | 5.00V | 5.25V |
| X coordinate (after digitising) | 206 | 516 | 798 |
| Y coordinate (after digitising) | 203 | 507 | 797 |



# Joystick

- Step 1: Configure the ADC
  - What is the ADC source? ADC0 or ADC1
  - What reference voltage to use? AVCC = 5V
  - Align left or right? Right, bottom 8-bit in ADCL
  - Enable or disable ADC auto-trigger? Disable
  - Enable or disable ADC interrupt? Disable
  - What is the ADC pre-scaler? 128 (slowest conversion)

| | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0/1 |
|---|---|---|---|---|---|---|---|---|
| ADCMUX | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 |

| | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| ADCSRA | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 |

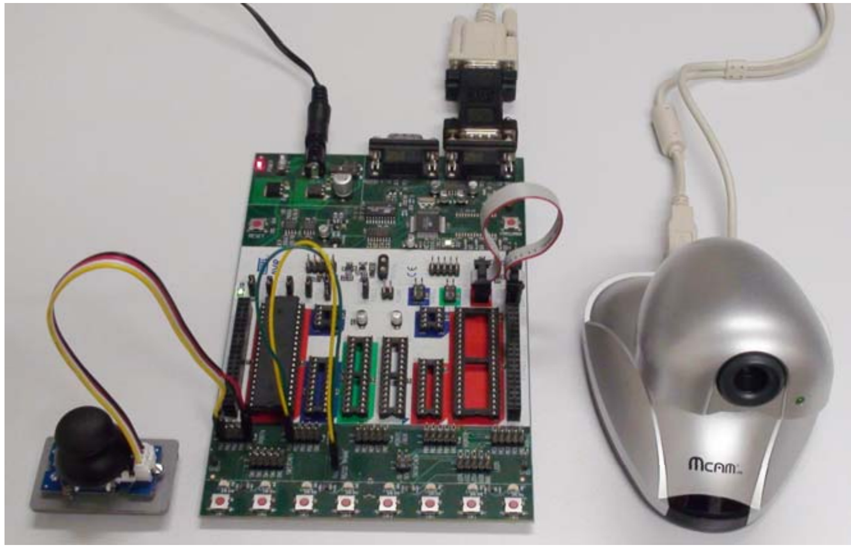
- Steps 2 and 3: Show next in C program.



# Joystick and Pan-Tilt Camera: Code

```
#include<avr/io.h>
int main (void){
   unsigned int result_low, result_high, result_x, result_y;
   // … Initialise serial port ...

   // Configure the ADC module of the ATmega16
   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){
        // Read X coordinate...
        ADMUX = 0b01000000;  // REFS1:0 = 01    -> AVCC as reference,
                             // ADLAR   =  0    -> Right adjust
                             // MUX4:0  = 00000 -> ADC0 as input
        ADCSRA |= (1 << ADSC);           // Start conversion by setting flag ADSC
        while (ADCSRA & (1 << ADSC)){;} // Wait until conversion is completed
        // Read digital output
        result_low  = ADCL;              // low 8 bits in ADCL
        result_high = ADCH & 0b00000011; // bit 8 and 9 in ADCH
        result_x = (result_high << 8) + result_low;


        // Read Y coordinate...
        ADMUX = 0b01000001; // MUX4:0  = 00001 -> ADC1 as input
        ...

        // Serial port code for camera control ...
   }
   return 0;
}
```

## Joystick and Pan-Tilt Camera: Demo




[A live demo will be shown in the lecture]



## Lecture 11's summary

- What we learnt in this lecture:
  - Analogue-to-digital conversion process.
  - Sampling and quantization steps.
  - Using the ADC in the ATmega16 microcontroller.
  - An example application of ADC.
- What are the next activities?
  - Tutorial 11: 'Analogue-to-Digital Converter'.
  - Lab 11: 'Analogue-to-Digital Converter'.
    - Complete the online Pre-lab Quiz for Lab 11.
    - Write programs for Tasks 1 and 2 of Lab 11.
    - See video demos of Lab 11: [avr]/ecte333/lab11_task1.mp4
      [avr]/ecte333/lab11_task2.mp4



## Lecture 11's references


- Atmel Corp., 8-bit AVR microcontroller with 16K Bytes In-System Programmable Flash ATmega16/ATmega16L, 2007, [Analog to Digital Converter]. Manual
- S. F. Barrett and D. J. Pack, Atmel AVR Microcontroller Primer: Programming and Interfacing, 2008, Morgan & Claypool Publishers, [Chapter 3: Analog-to-Digital Conversion].




## Lecture 11's references


- M. Mazidi, J. Mazidi, R. McKinlay, "The 8051 microcontroller and embedded systems using assembly and C," 2nd ed., Pearson Prentice Hall, 2006, [Chapters 13.1].
- P. Spasov, "Microcontroller technology the 68HC11," 3rd ed., Prentice Hall, 1999, [Chapters 12].
- H. Huang, "MC68HC12 an introduction: software and hardware interfacing," Thomson Delmar Learning, 2003, [Chapter 10].