ECTE333
Lecture 11 - Analogue-to-Digital Converter

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

ECTE333’s schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Lecture (2h)</th>
<th>Tutorial (1h)</th>
<th>Lab (2h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L7: C programming for the ATMEL AVR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Tutorial 7</td>
<td>Lab 7</td>
</tr>
<tr>
<td>3</td>
<td>L8: Serial communication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Tutorial 8</td>
<td>Lab 8</td>
</tr>
<tr>
<td>5</td>
<td>L9: Timers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Tutorial 9</td>
<td>Lab 9</td>
</tr>
<tr>
<td>7</td>
<td>L10: Pulse width modulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Tutorial 10</td>
<td>Lab 10</td>
</tr>
<tr>
<td>9</td>
<td>L11: Analogue-to-digital converter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Tutorial 11</td>
<td>Lab 11</td>
</tr>
<tr>
<td>11</td>
<td>L12: Revision lecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Lab 12</td>
</tr>
<tr>
<td>13</td>
<td>L13: Self-study guide (no lecture)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Final exam (25%), Practical exam (20%), Labs (5%)*

Lecture 11’s sequence

11.1 Introduction to Analogue-to-Digital 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.

11.2 Analogue-to-digital converter in ATmega16

11.3 Example application of ADC

[Lab 11: Task 1 | Task 2]
**A-to-D conversion: Typical embedded application**

- Physical variables (temperature, pressure, light)
- Transducer *electrical signals*
- Signal conditioning *voltages*
- Analogue-to-digital converter *digital signals*
- Input ports
  - Processor
  - Output ports
- Digital-to-analogue converter *analogue control signals*
- Actuator

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 an 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

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

**Significance**

- If maximum frequency of the signal is $F_{\text{max}}$, the sampling rate should be at least:
  $$\text{Nyquist rate} = 2 \times F_{\text{max}}$$

- If the sampling rate is $F_s$, the maximum frequency in the signal must not exceed:
  $$\text{Nyquist frequency} = \frac{1}{2} F_s$$
Quantising the sampled signal

Consider an n-bit ADC.

- Let $V_{\text{ref}}$ be the reference voltage.
- Let $V_{\text{in}}$ be the analog input voltage.
- Let $V_{\text{min}}$ be the minimum allowable input voltage, usually $V_{\text{min}} = 0$.

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:

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

A 3-bit A-to-D converter

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** $T_{\text{sample}}$: A sampling capacitor must be charged for a duration of $T_{\text{sample}}$ before conversion taking place.

- **Conversion time** $T_{\text{conv}}$: 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.

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.

Flash ADC

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

**Successive-approximation ADC**

Binary search for a 3-bit ADC

- \( V_{ref} = 5V \)
- \( V_{appr} \)
- \( V_{in} = ? \)
- Full scale value
- \( D_2 D_1 D_0 \)
- \( D_2 = 1 \)
- \( D_1 = 1 \)
- \( D_0 = 1 \)
- \( [V_{in} > V_{appr}] \)
- \( [V_{in} < V_{appr}] \)
- \( [V_{in} = V_{appr}] \)
### Example IC for ADC

#### ADC0804
(National Semiconductor)

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

### 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 \( V_{ref} = 5\text{V} \) is used.
  - step size: \( 5(\text{V})/1024 \) (steps) = ± 4.88mV.
  - accuracy: \( 2 \times \text{LSB} = ± 9.76\text{mV} \).
- 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 Multiplexer Selection Register (ADCMUX)**
  - 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.  
  - Analog input voltage can be selected among different pins. Differential input and custom gain factor can also be chosen.
  - ADLAR flag determines how the 10-bit digital output is stored in output registers.
Selecting reference voltage $V_{ref}$

Table 11.1: ADC reference voltage selection.

<table>
<thead>
<tr>
<th>REF1</th>
<th>REF0</th>
<th>Voltage Reference Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>AREF, Internal Vref turned off</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>AVCC with external capacitor at AREF pin</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Internal 2.56V Voltage Reference with external capacitor at AREF pin</td>
</tr>
</tbody>
</table>

- 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

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

- 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)

- 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.**

<table>
<thead>
<tr>
<th>ADPS2</th>
<th>ADPS1</th>
<th>ADPS0</th>
<th>Division Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>128</td>
</tr>
</tbody>
</table>

- 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**

<table>
<thead>
<tr>
<th>ADTS2</th>
<th>ADTS1</th>
<th>ADTS0</th>
<th>ACME</th>
<th>PUD</th>
<th>PSR2</th>
<th>PSR10</th>
</tr>
</thead>
</table>

**Table 11.4: ADC Auto Trigger Source.**

<table>
<thead>
<tr>
<th>ADTS2</th>
<th>ADTS1</th>
<th>ADTS0</th>
<th>Trigger Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Free Running mode</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Analog Comparator</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>External Interrupt Request</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Timer/Counter0 Compare Match</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Timer/Counter0 Overflow</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Timer/Counter1 Compare Match B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Timer/Counter1 Overflow</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Timer/Counter1 Capture Event</td>
</tr>
</tbody>
</table>

- 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 ADCSRA).

**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
<table>
<thead>
<tr>
<th>REFS1</th>
<th>REFS0</th>
<th>ADLAR</th>
<th>MUX4</th>
<th>MUX3</th>
<th>MUX2</th>
<th>MUX1</th>
<th>MUX0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

ADCMUX
```

```
1 0 0 0 0 0 0 1 0
<table>
<thead>
<tr>
<th>ADEN</th>
<th>ADSC</th>
<th>ADATE</th>
<th>ADIE</th>
<th>ADPS2</th>
<th>ADPS1</th>
<th>ADPS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

ADCSRA
```

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

```
Example 11.1: adc.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
  // ADPS2:0 = 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;
}
```

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 ADCSRA.

- 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){
  DDDB = 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 = 0b1000111; // ADEN = 1: enable ADC,
  // ADSC = 0: don't start conversion yet
  // ADATE = 0: disable 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;
}
```

---

**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.
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

### Working voltage VCC

<table>
<thead>
<tr>
<th>Description</th>
<th>Min</th>
<th>Typical</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working voltage VCC</td>
<td>4.75V</td>
<td>5.00V</td>
<td>5.25V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Min</th>
<th>Typical</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>X coordinate (after digitising)</td>
<td>206</td>
<td>516</td>
<td>798</td>
</tr>
<tr>
<td>Y coordinate (after digitising)</td>
<td>203</td>
<td>507</td>
<td>797</td>
</tr>
</tbody>
</table>

### 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)**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFS1</td>
<td>REFS0</td>
<td>ADLAR</td>
<td>MUX4</td>
<td>MUX3</td>
<td>MUX2</td>
<td>MUX1</td>
<td>MUX0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>ADCMUX</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADEN</td>
<td>ADSC</td>
<td>ADATE</td>
<td>ADIF</td>
<td>ADIE</td>
<td>ADPS2</td>
<td>ADPS1</td>
<td>ADPS0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>ADCSRA</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```c
#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
  // ADPS2,0 = 111: prescaler = 128

  while(1){
    // Read X coordinate...
    ADMUX = 0b01000000; // REFS1:0 = 01 -> AVCC as reference,
    // ADLAR = 0: Right adjust
    // MUX4:0 = 0000 -> 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; // REFS1:0 = 01 -> AVCC as reference,
    // ADLAR = 0: Right adjust
    // MUX4:0 = 0000 -> 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_y = (result_high << 8) + result_low;

    // Serial port code for camera control ...
  }
  return 0;
}
```
**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?**
    - 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_task1.mp4) [avr/ecte333/lab11_task2.mp4](avr/ecte333/lab11_task2.mp4)

---

**Lecture 11’s references**


