Topic 9 Pre-processor and Commands in C and Analog Input/Output

9.1 The Pre-processor

#define, #error, #include

9.2 Conditional Compilation

#if, #else, #endif

#elif,

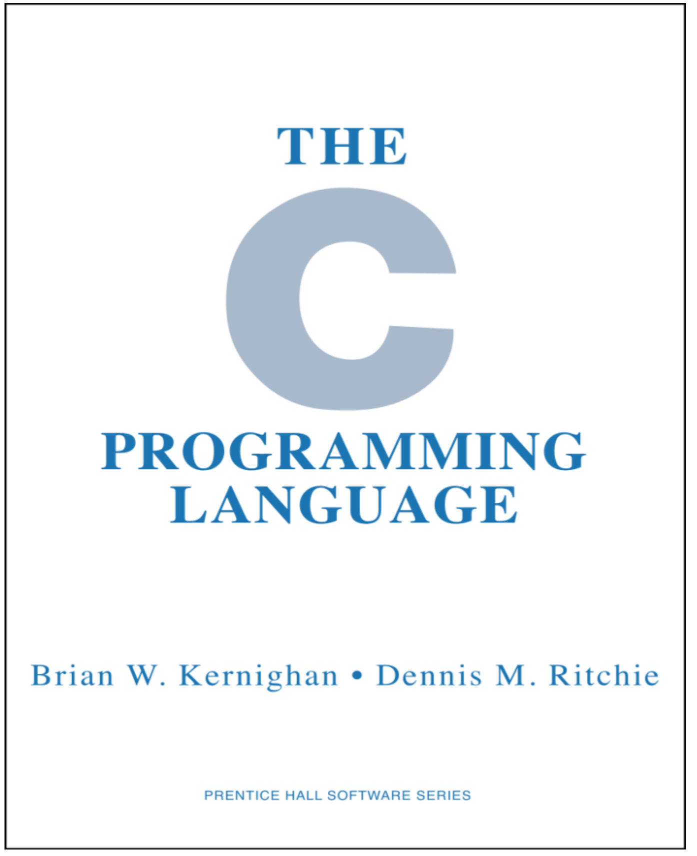
#ifdef, #ifndef, #undef

9.3 Using defined

#line,

9.4 Comments

9.5 Analog Input/Output



9.1 The Pre-processor

- You can include various instructions to the compiler in the source code of a C program, namely *pre-processor.*

#define	*#endif*	*#ifdef*	*#line*
#elif	*#error*	*#ifndef*	*#pragma*
#else	*#if*	*#include*	*#undef*

#define Its general form is

#define macro-name char-sequence

e.g.

```
#define LEFT 1
#define RIGHT 0

printf("%d %d %d", RIGHT, LEFT, LEFT+1);
```

9.1 The Pre-processor

Defining Function-like Macros

- The *#define* directive has another powerful feature: The macro name can have arguments.
- This form of a macro is called a function-like macro, e.g.

```
#include <stdio.h>
#define ABS(a)     (a)<0 ? -(a) : (a)
int main(void)
{
      printf("abs of -1 and 1: %d %d", ABS(-1), ABS(1));
      return 0;
}
```

9.1 The Pre-processor

#error

- *#error* forces the compiler to stop compilation. It is used primarily for debugging. Its general form is

#error error-message

#include

- *#include* tells the compiler to read another source file in addition to the one that contains the #include directive.
- The name of the source file must be enclosed between double quotes or angle brackets. For example,

```
#include "stdio.h"
#include <stdio.h>
```

9.2 Conditional Compilation

#if, #else and #endif

- The general form of *#if* is.

  #if constant-expression

  statement sequence

  #endif

- The program at the right displays the message on the screen because MAX > 99.
- The expression that follows the *#if* is evaluated at compile time.

```
/* Simple #if example. */
#include <stdio.h>
#define MAX 100
int main(void)
{
     #if MAX>99
   printf("Compiled for array
greater than 99.\n");
     #endif
     return 0;
}
```

9.2 Conditional Compilation

- *#else* works much like *else* that is part of the C language. The previous example can be expanded as shown here:

```
/* Simple #if/#else example. */
#include <stdio.h>
#define MAX 10
int main(void)
{
   #if MAX>99
     printf("Compiled for array greater than 
99.\n");
   #else
     printf("Compiled for small array.\n");
   #endif
   return 0;
}
```

9.2 Conditional Compilation

#elif means "else if" and establishes an if-else-if chain for multiple compilation options.

#elif is followed by a constant expression.

#if *expression*
    statement sequence
#elif *expression 1*
    statement sequence
#elif *expression 2*
………

```
#define US 0
#define ENGLAND 1
#define FRANCE 2
#define ACTIVE_COUNTRY US
#if ACTIVE_COUNTRY == US
char currency[] = "dollar";
#elif ACTIVE_COUNTRY == ENGLAND
char currency[] = "pound";
#else
char currency[] = "franc";
#endif
```

9.2 Conditional Compilation

#ifdef* and *#ifndef

- Another conditional compilation uses the directives *#ifdef* and *#ifndef*, which mean "if defined" and "if not defined," respectively.
- The general form of *#ifdef* is

```
#ifdef macro-name
     statement sequence
#endif
```

- The general form of *#ifndef* is

```
#ifndef macro-name
     statement sequence
#endif
```


•

9.2 Conditional Compilation

For example:

- If *TED* were defined, The code at right will print *Hi Ted* and *RALPH not defined*.
- However, if *TED* were not defined, *Hi anyone* would be displayed, followed by *RALPH not defined*.

```
#include <stdio.h>
#define TED 10
int main(void)
{
   #ifdef TED
       printf("Hi Ted\n");
   #else
       printf("Hi anyone\n");
   #endif

   #ifndef RALPH
       printf("RALPH not
defined\n");
   #endif
       return 0;
}
```

9.2 Conditional Compilation

#undef has the general form below:

#undef   macro-name //directive removes the current definition of identifier

For example:

```
#define LEN 100
#define WIDTH 100
char array[LEN][WIDTH];
……
#undef LEN
#undef WIDTH
/* at this point both LEN and WIDTH are 
undefined */
```

9.3 Using defined

The **defined** operator has this general form:

defined *macro-name*

- If *macro-name* is currently defined, the expression is true; otherwise, it is false.
- For example, to determine whether the macro MYFILE is defined, you can use either of the following commands:

```
#if defined   MYFILE
```

or

```
#ifdef   MYFILE
```

- You can precede defined with the ! to reverse the condition.

```
#if ! defined DEBUG
      printf("Final version!\n");
#endif
```

9.3 Using defined

#line has the following general form:

This contains the current line number as a decimal constant

For example*:*

```
#include <stdio.h>
#line 100                        /* reset the line counter */
int main(void)                   /* line 100 */
{                                /* line 101 */
   printf("%d\n", _ _LINE_ _); /* line 102 */
   return 0;
}
```

9.4 Comments

C defines the style of comments as follows:

/ this is a comment */*

or

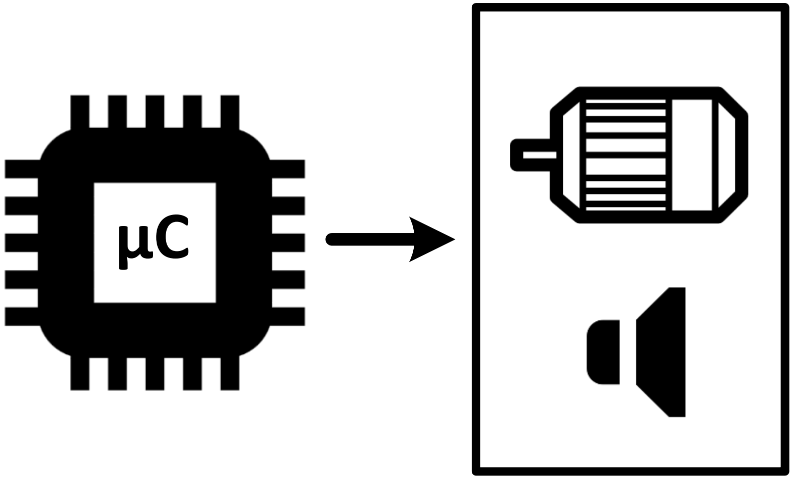
// this is a single-line comment

```c
#include <stdio.h>
int main(void)
{
    printf("hello");     /* print it o screen */
    return 0;
}
```

/* this is a multiline comment */

9.5 Digital-to-analog conversion

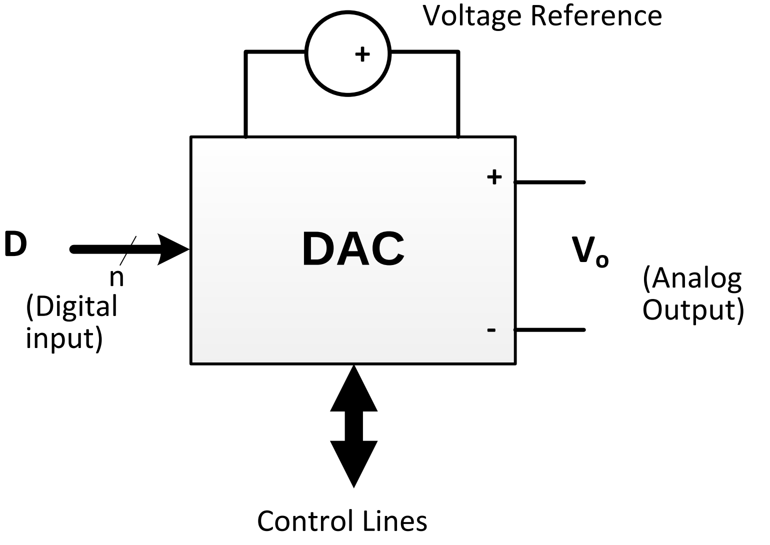
- Microcontrollers must be able to convert digital signals to analog (e.g. driving loudspeaker or DC motor)
  - This process is called ***data conversion***
- A digital-to-analog converter (DAC) is used to perform this operation
  - It is a circuit that converts a binary input number into an analog output



9.5 Digital-to-analog conversion (Cont'd)

- The DAC has a digital input D and an analog output V_o
- It uses a *voltage reference* (precise and known voltage) to calculate its output voltage
- Most DACs, including the one inside the chip, apply the following equation to calculate the analog output

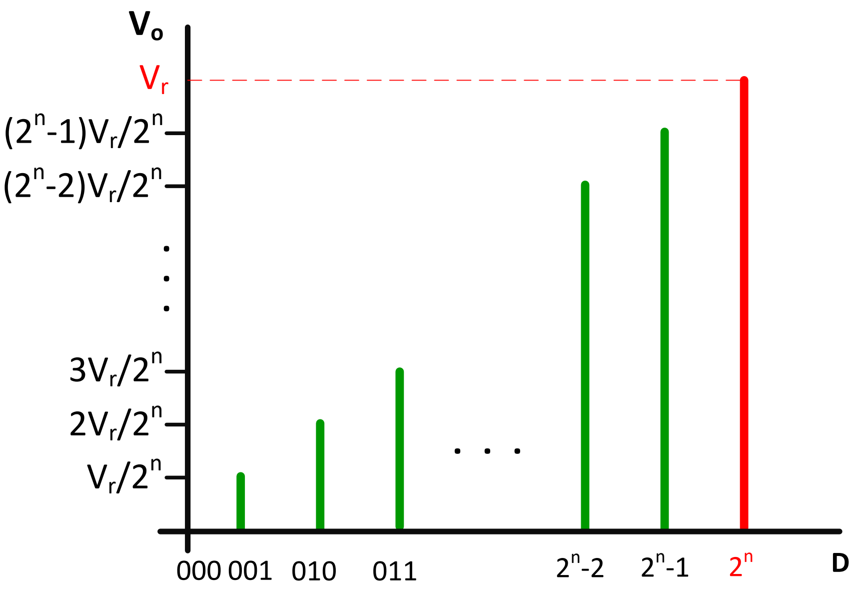
$$V_o = \frac{D}{2^n} \times V_r$$



- V_r: the reference voltage
- D: the value of the binary input
- V_o: the output voltage
- n: the number of bits in D

9.5 Digital-to-analog conversion (Cont'd)

- For each digital input value there is a corresponding analog output
- The number of possible output values is $\mathbf{2^n}$ and a step size (also called ***resolution***) of $\boldsymbol{V_r/2^n}$
- The maximum possible output occurs when $\boldsymbol{D = (2^n - 1)}$
- The range of a DAC is the difference between its maximum and minimum output values



9.5 Digital-to-analog conversion (Cont'd)

Example

- A 6-bit DAC will have $\mathbf{2^6 = 64}$ possible output values.
- If it has a 3.2 V reference, it will have a resolution (step size) of $\mathbf{3.2/2^6 = 50\ mV}$

Exercise

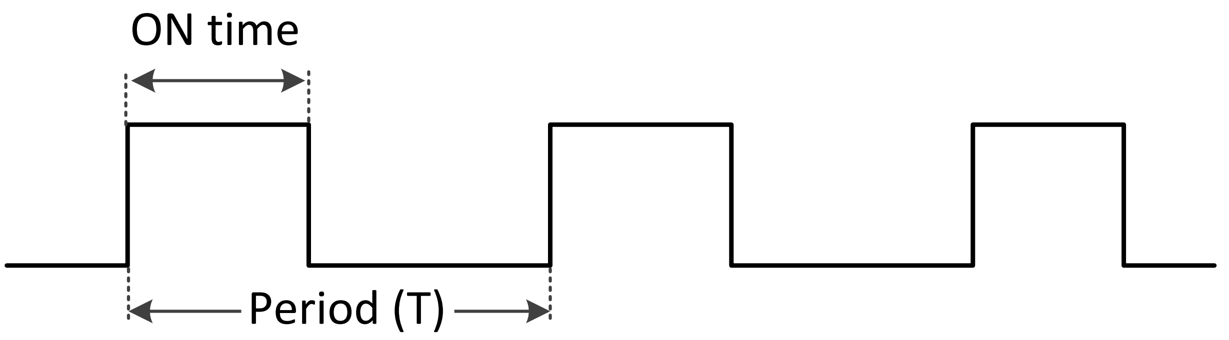
- The microcontroller has a 10-bit DAC and a reference voltage 3.3 V
  - How many output values we can get from it?
  - What is the step size?

9.5 Pulse Width Modulation (PWM)

- Pulse Width Modulation is another form of analog output
- PWM represents a neat and simple way of getting a rectangular digital waveform to control an analog variable, usually voltage or current
- PWM control is used in a variety of applications (e.g. telecommunications, robotic)

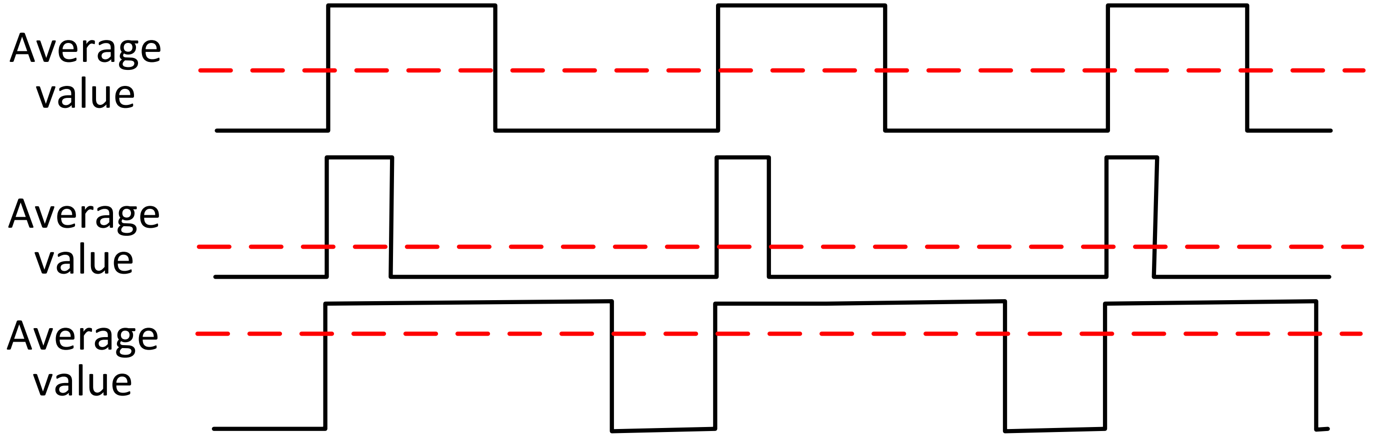
9.5 PWM signal

- PWM signal:



9.5 PWM signal (Cont'd)

- Whatever duty cycle a PWM has, there is an ***average value***
  - The average value is the point of interest when using PWM
  - If the ON time is small, the average value is low
  - If the ON time is large, the average value is high

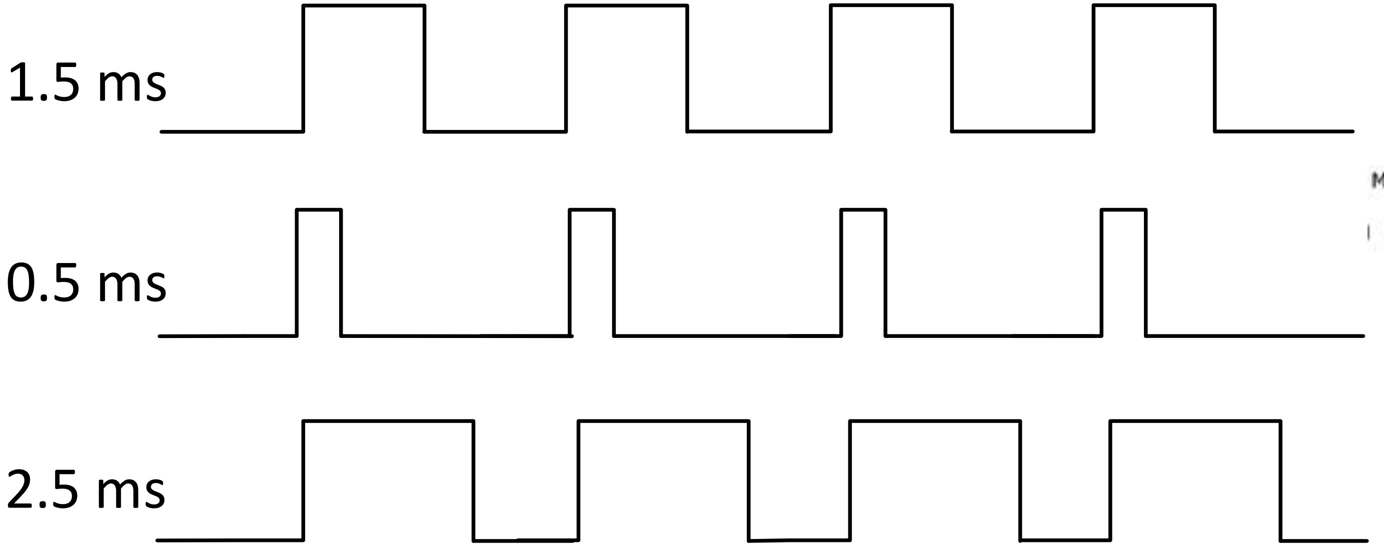


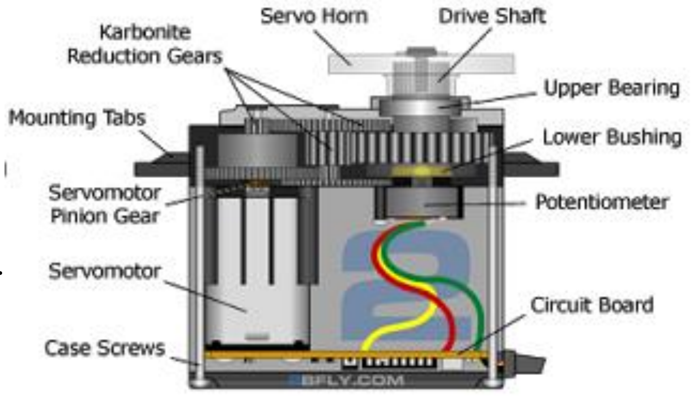
Example

- The control of DC motor is a very common task in robotics and its speed is proportional to the applied DC voltage
- A PWM signal can be used to control the speed

9.5 PWM signal (Cont'd)

- Continuous Rotation Servo Motor Timing
  - A pulse width of 1.5 ms will cause the servo shaft stop spinning.
  - A pulse width of 0.5 ms will cause the servo shaft to spin at full speed counter-clockwise..
  - A pulse width of 2.5 ms will cause the servo shaft to spin at full speed clockwise.





9.5 PWM signal (Cont'd)

Use Emoro lib:

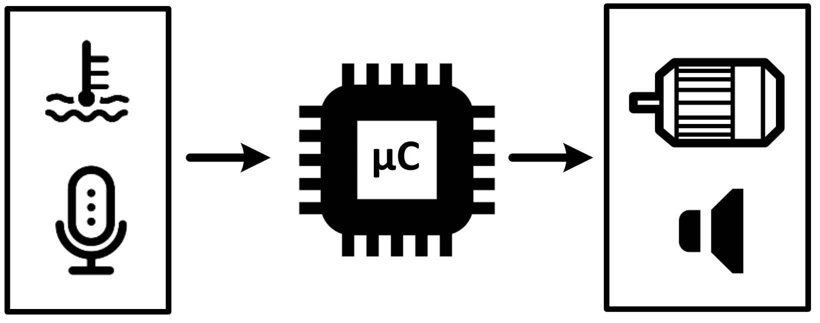
```
EmoroServo.write(SERVO_0, 1500); //stop
EmoroServo.write(SERVO_0, 500); //full speed clockwise
EmoroServo.write(SERVO_0, 2500); //full speed ant-clockwise
```

Use Arduino servo lib:

```
#include <Servo.h>
int servoPort1 = SERVO_0;
Servo myservo1;  // create servo object to control a servo
void setup() {
        myservo1.attach(servoPort1,500,2500);  // attaches the servo on
        myservo1.write(0);
        delay(490);
        myservo2.write(180);
        delay(490);
        myservo1.write(90);
}
```

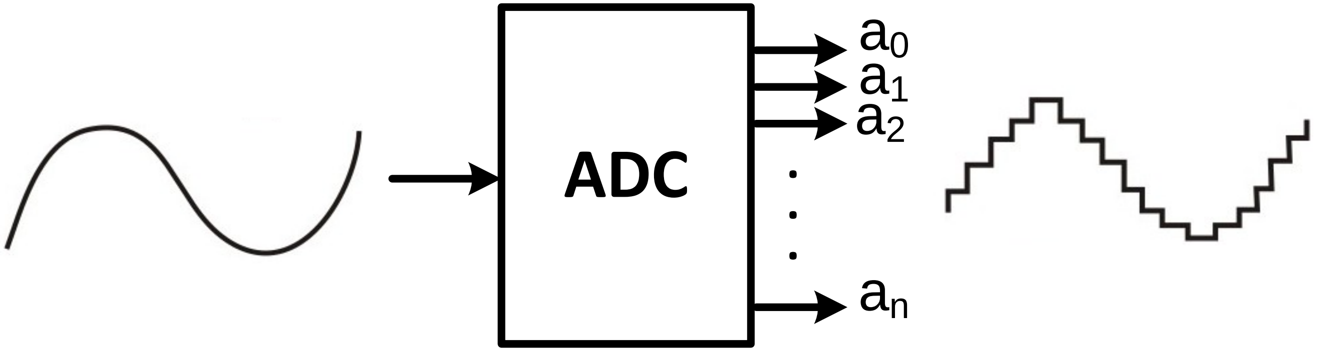
9.5 Introduction to analog data

- Human world signals are usually based upon continuous analog signals that vary in time and space
  - e.g. temperature variation in the room over a day
- Microcontrollers are often required to process analog signals (e.g. from microphone or temperature sensor) and must be able to convert them first to digital data
- They must also be able to convert digital signals to analog (e.g. driving loudspeaker or DC motor)



9.5 Analog-to-digital conversion

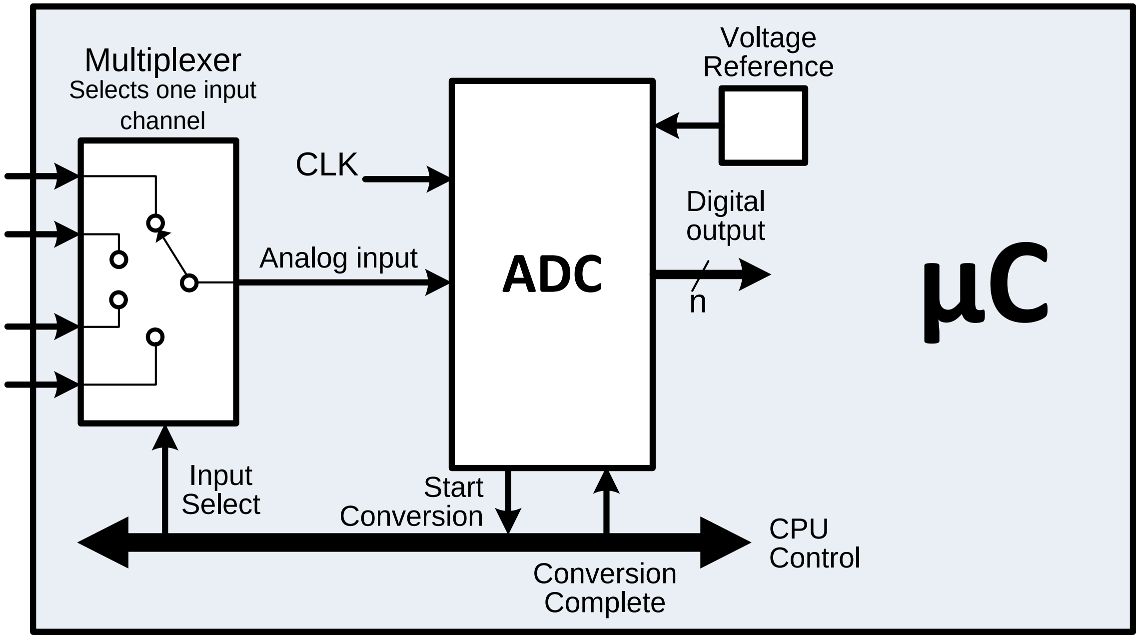
- An analog-to-digital convertor (ADC) is an electronic circuit whose digital output is proportional to its analog input
- The ADC measures the input voltage and gives a binary output number proportional to its size
- Analog signals can be repeatedly converted into digital representations with a **resolution** and at a **rate** determined by the ADC



9.5 Analog-to-digital conversion (Cont'd)

- Usually we want to work with more than one signal
  - More than one ADC could be used- **Costly and consumes semiconductor space**

- An analog multiplexer can be put in front of the ADC



9.5Range and Resolution

- Many ADCs obey the following equation:

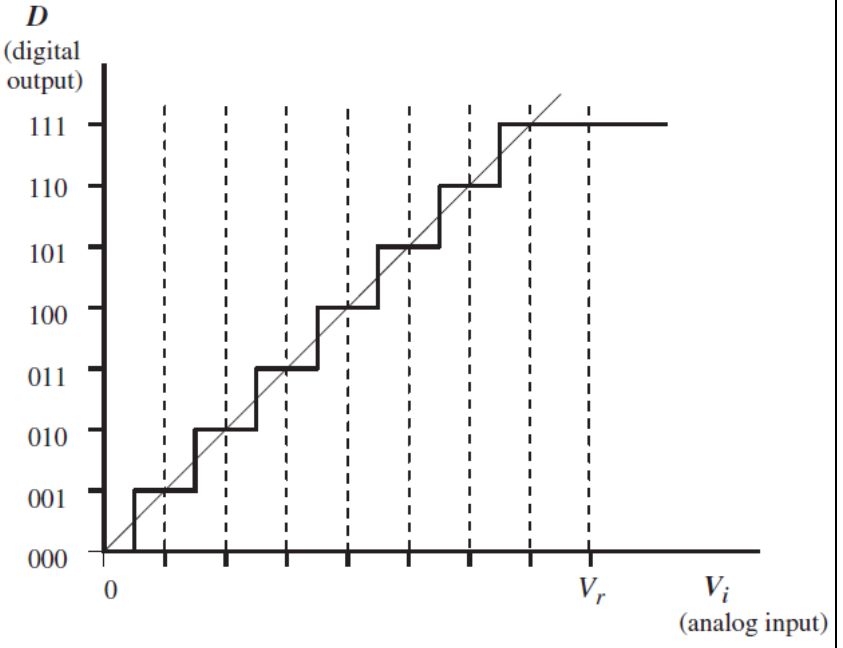
$$D = \frac{V_i}{V_r} \times 2^n$$

  - **D**: the digital output value (integer value)
  - $\mathbf{V_i}$: the input voltage
  - $\mathbf{V_r}$: the reference voltage
  - **n**: the number of bits
- ADC has minimum and maximum permissible input values
  - The difference between min and max values is called the ***range***
  - Often the minimum value is 0 V

9.5 Range and Resolution (Cont'd)

Example 3-bit ADC

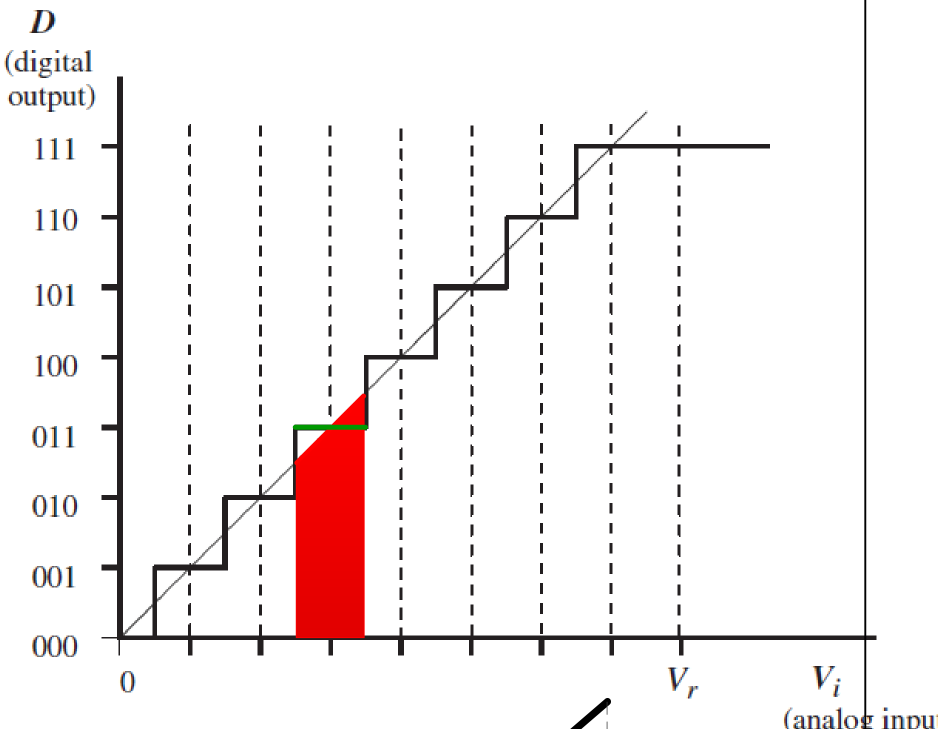
- The input voltage is gradually increased starting from 0 V (Output 000)
- If the analog input slowly increases, there comes a point when the digital output changes to 001
- At some points the output will reach 111 (the max value $\mathbf{2^3 - 1}$)

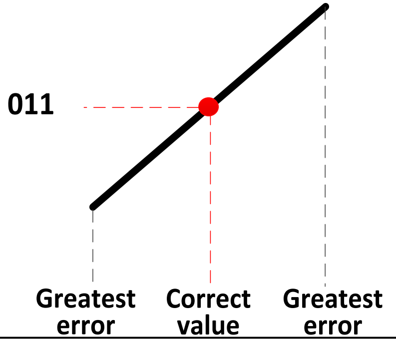


- The input may increase further but cannot force any increase in output value

9.5 Quantization error

- By converting an analog signal to digital there is a risk of approximation
- Any one digit output value has to represent a small range of analog input voltages
- If the output value of 011 is correct for the input voltage at the middle of the step then the greatest error occurs at either end of the step
- This is called ***Quantization Error***





9.5 Quantization error (Cont'd)

- The more steps the lower is the quantization error
- More steps are obtained by increasing the number of bits in the ADC
  - This increases the complexity, cost of the ADC and the conversion time

Example

- To convert an analog signal that has a range $\mathbf{0-3.3}$ V to an 8-bit digital signal then:
  - There are $\mathbf{2^8 = 256}$ output values
  - The step width is: $\mathbf{3.3/256 = 12.89}\ \boldsymbol{mV}$
  - The worst case quantization error is $\mathbf{6.45}\ \boldsymbol{mV}$

9.5 Quantization error (Cont'd)

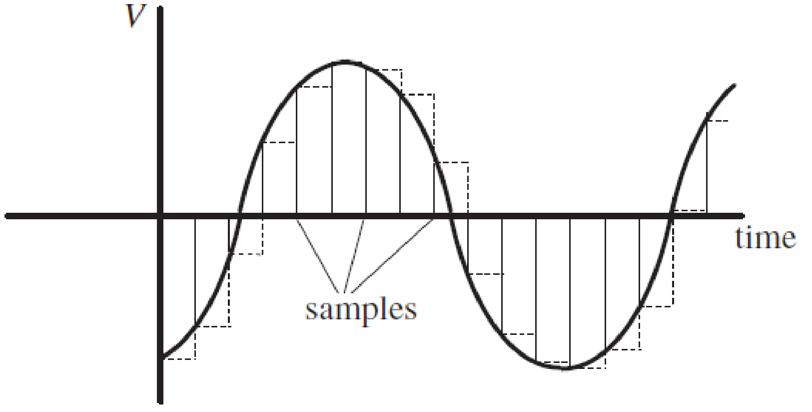
Exercise

- An ADC is 12 bit
  - How many output values we can get from it?
  - What is the step width?
  - What is the worst case quantization error?

9.5 Sampling frequency

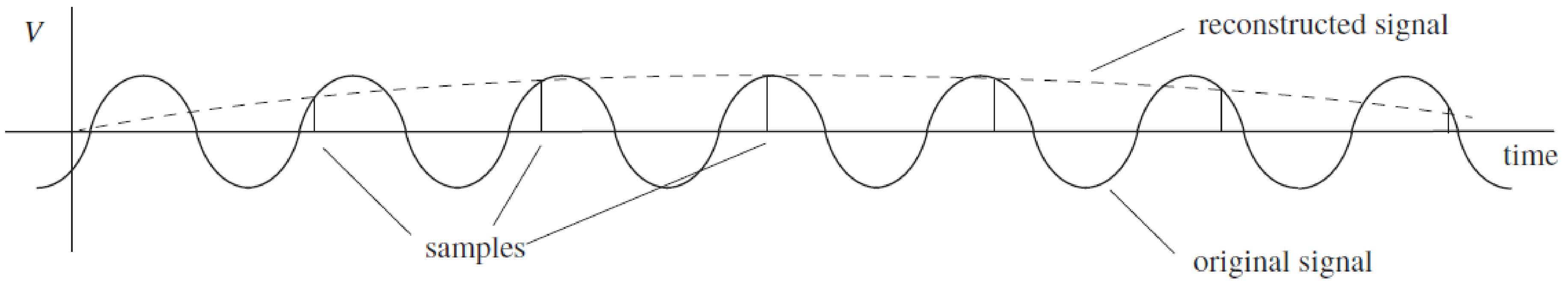
- During the conversion process, a sample is taken repeatedly and quantized to the accuracy defined by the resolution of the ADC
  - The more samples taken the more accurate the digital output will be
- Sampling is done at a fixed frequency called the **sampling frequency**

- Sampling frequency depends on the maximum frequency of the input signal



9.5 Sampling frequency (Cont'd)

- If the sampling frequency is too low then rapid changes in the analog signal may not be represented
- Nyquist sampling criterion: the sampling frequency must be at least double that of the highest signal frequency
  - If it is not satisfied then the ***aliasing*** phenomenon occurs



9.5 Coding example for Analog out

```
// These constants won't change.  They're used to give names to the pins used:
const int analogInPin = ADC_0;                    // analog input pin that the potentiometer is
    attached to
const int analogOutPin = LED_BUILTIN;             // analog output pin that the LED is attached to

int sensorValue = 0;                              // value read from the potentiometer
int outputValue = 0;                              // value output to the PWM (analog out)

void setup() {
  Serial.begin(9600);                             // initialize serial communications at 9600 bps
}

void loop() {
  sensorValue = analogRead(analogInPin);          // read the analog in value:
  outputValue = map(sensorValue, 0, 1023, 0, 255);// map it to the range of the analog out
  analogWrite(analogOutPin, outputValue);         // change the analog out value

  // print the results to the serial monitor:
  Serial.print("sensor = " );                     // print string "sensor ="
  Serial.print(sensorValue);                      // print sensor value (0 - 1023)
  Serial.print("\t output = ");                   // append tabulator and string "output"
  Serial.println(outputValue);                    // print the calculated output value and append
    newline
  // wait 2 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(2);
}
```