

UNIT 2

Peripheral Interfacing-I

Starting to Program Digital Input and Output

Voltages as Logic Values

Introducing Analog output Data Conversion

Digital Output on the mbed

Lab 4: A/D conversion program

Digital Input and Output.

Switching Larger DC Loads

Lab 5: Mini Project: Letter Counter

Another Form of Analog Output: Pulse Width Modulation

Pulse Width Modulation on the mbed

Design of PWM problem

Lab 6: PWM waveform generation

Starting to Program Digital Input and Output

/*Program Example 2.1: A program which flashes mbed LED1 on and off.

Demonstrating

use of digital output and wait functions. Taken from the mbed site. */

`#include "mbed.h"` //include the mbed header file as part of this program

// program variable myled is created, and linked with mbed LED1

DigitalOut myled(LED2);

int main() { //the main function starts here

while(1) { //a continuous loop is created

myled = 1; //switch the led on, by setting the output to logic 1

wait(0.2); //wait 0.2 seconds

myled = 0; //switch the led off

wait(0.2); //wait 0.2 seconds

} //end of while loop

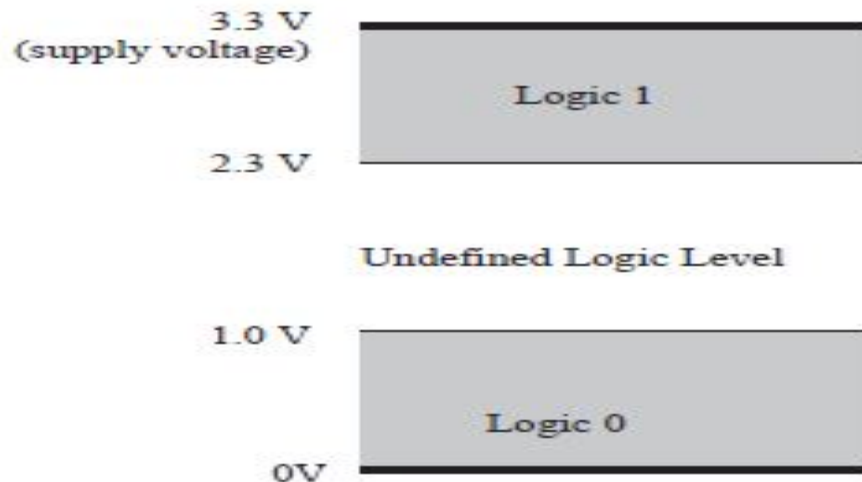
} //end of main function

C/C++ Function	Action
wait	Waits for the number of seconds specified (float)
wait_ms	Waits for the whole number of milliseconds specified (int)
wait_us	Waits for the whole number of microseconds specified (int)

Voltages as Logic Values

This data show that, for most digital inputs, the LPC1678 interprets any input voltage below 1.0 V (specified as 0.3 3.3 V) as Logic 0, and any input voltage above 2.3 V (specified as 0.7 3.3 V) as logic 1

This idea is represented in the diagram of Fig



Introducing Analog output Data Conversion

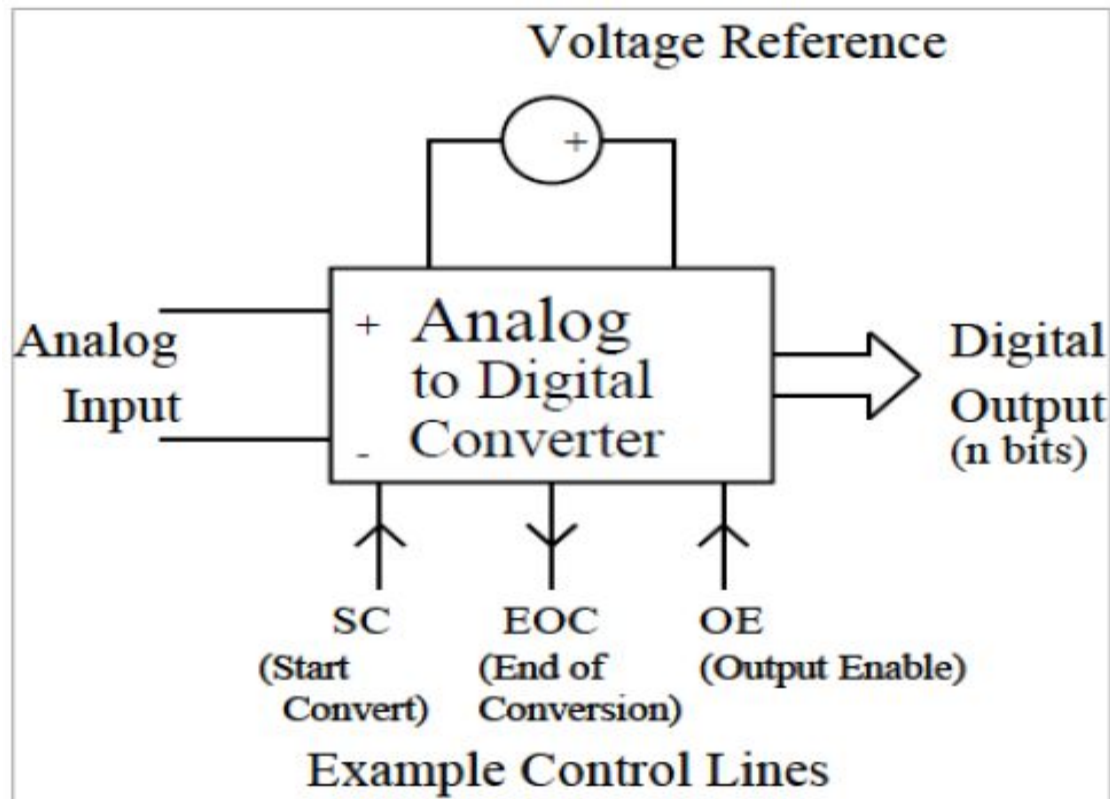
- ❑ Microcontrollers are often required to interface with analog signals
- ❑ They must be able to convert input analog signals, for example from **microphone** or temperature sensor, to digital data
- ❑ They must also be able to convert digital signals to analog form, for example if driving a **loudspeaker** or dc motor
- ❑ An analog-to-digital convertor (ADC) is an electronic circuit whose digital output is proportional to its analog input
- ❑ Effectively it "measures" the input voltage, and gives a binary output number proportional to its size
- ❑ The input range of the ADC is usually determined by the value of a voltage reference

Introducing Analog output Data Conversion

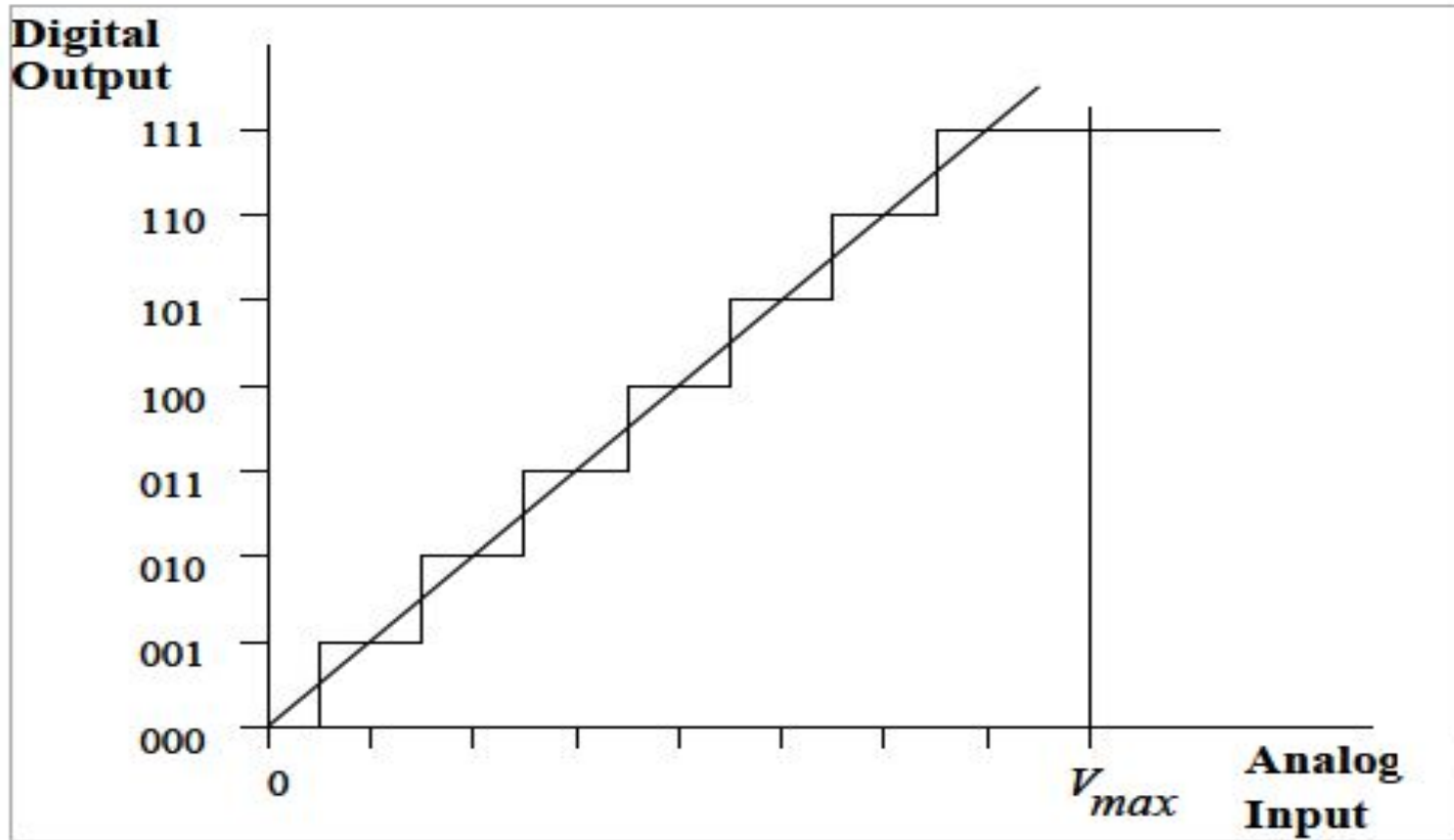
The conversion is started by a digital input, called here **SC**

It takes finite time, and the ADC signals with the **EOC line** when the conversion is complete

The resulting data can then be enabled onto a data bus using **the OE line**



Introducing Analog output Data Conversion



Introducing Analog output Data Conversion

Resolution and quantisation

☐ By converting an analog signal to digital, we are effectively approximating it, as any one digital output value has to represent a very small range of analog input voltages, i.e. the width of any of the steps on the “staircase” n.

☐ If we want to convert an analog signal that has a range 0-3.3 V to an 8-bit digital signal, then there are 256 (i.e. 2^8) distinct output values. Each step has a width of $3.3/256 = 12.89$ mV, and the **worst case quantisation error** is 6.45mV.

☐ The mbed uses a 12-bit ADC. This leads to a step width of $3.3/2^{12}$, or 0.008 V; the worst case quantisation error is therefore 0.004 V.

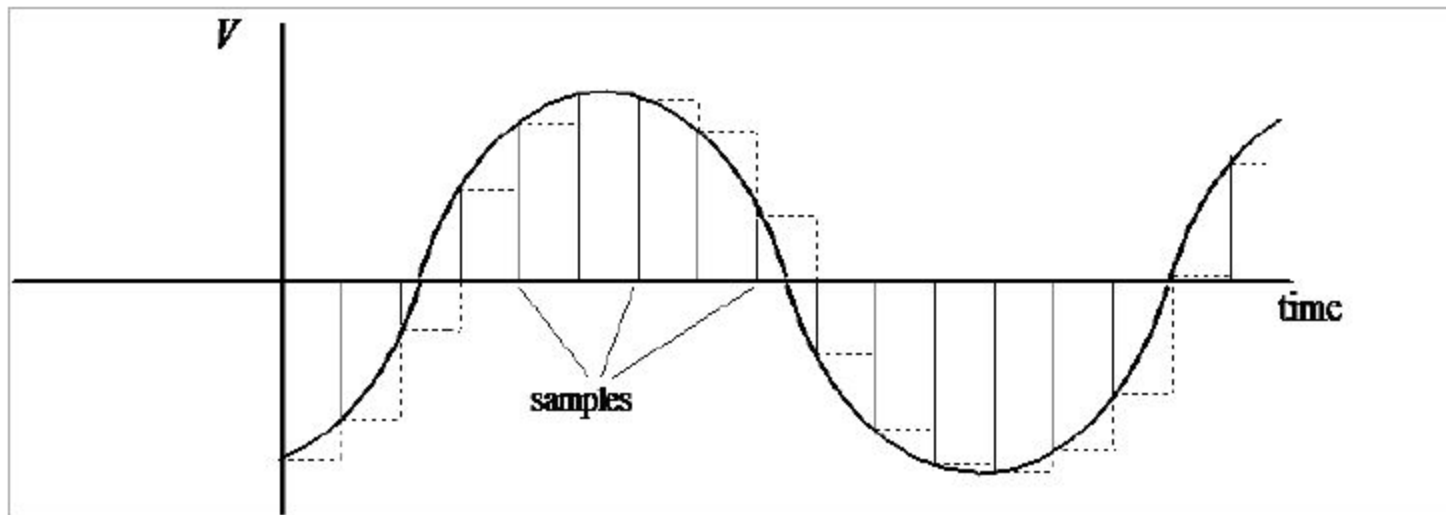
☐ When converting an analog signal to digital, we repeatedly take a „sample“ and quantise this to the accuracy defined by the resolution of our ADC.

☐ The more samples taken, the more accurate the digital data will be. Samples are normally taken at fixed periods (i.e., every 0.2ms) and define the rate of sampling by the sampling frequency (the number of samples taken per second).

Introducing Analog output Data Conversion

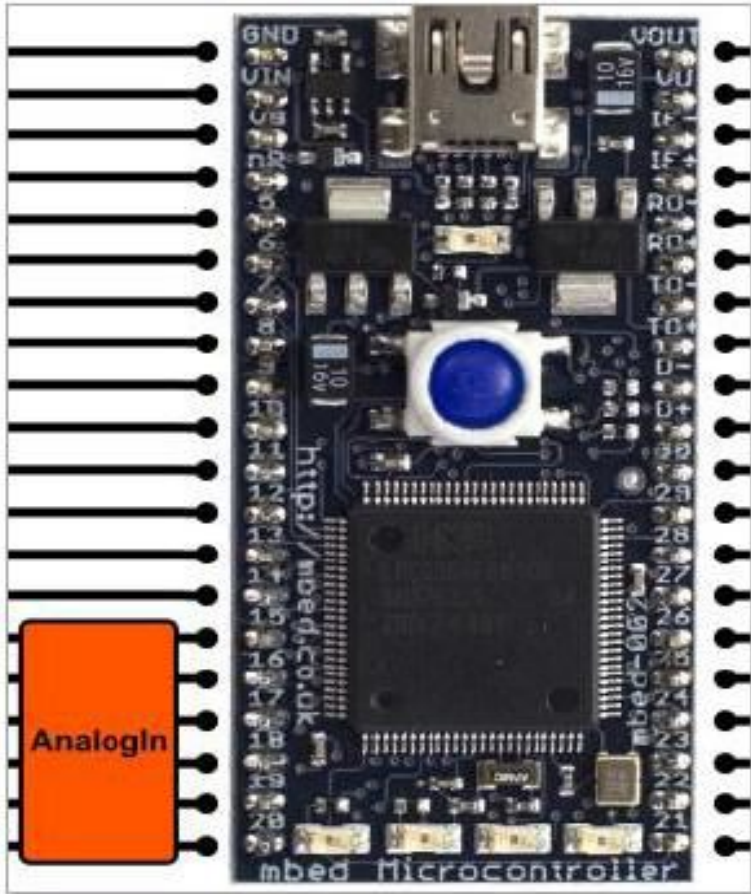
The sample frequency needs to be chosen with respect to the rate of which the sampled data is changing. If the sample frequency is too low then rapid changes in the analog signal may not be obvious in the resulting digital data.

☐ For this reason the Nyquist sampling criterion states that the sampling frequency must be at least double that of the highest frequency of interest.



Introducing Analog output Data Conversion

The mbed has up to six analog inputs, on pins 15 to 20



AnalogIn	An analog input, used for reading the voltage on a pin
Functions	Usage
AnalogIn	Create an AnalogIn, connected to the specified pin
read	Read the input voltage, represented as a float in the range [0.0, 1.0]
read_u16	Read the input voltage, represented as an unsigned short in the range [0x0, 0xFFFF]
operator float	operator float An operator shorthand for read()

Introducing Analog output Data Conversion

Exercise 1: Attach a potentiometer output to mbed pin 20. (**Note: pin 20 is connected to potentiometer Pot 2 of the application board**)

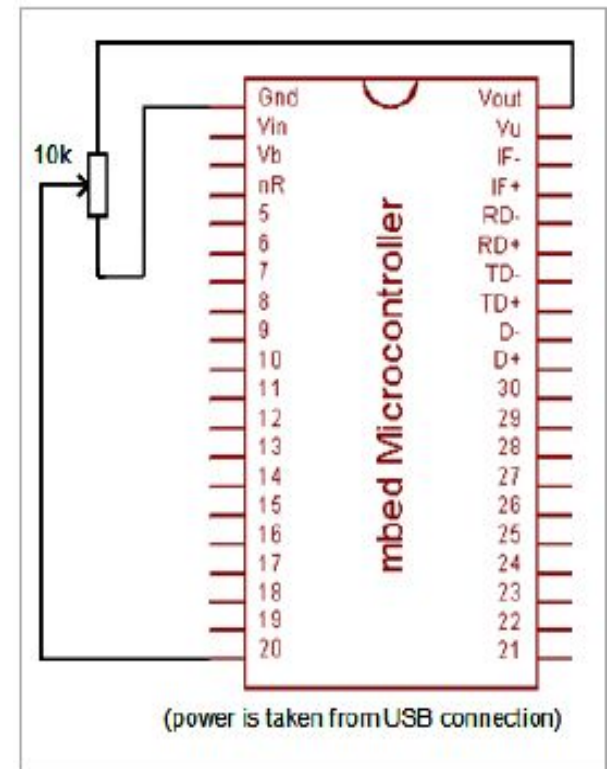
☐ Start a new mbed project and enter the code below.

☐ This code will continuously display the analog input value when used with a host PC terminal application.

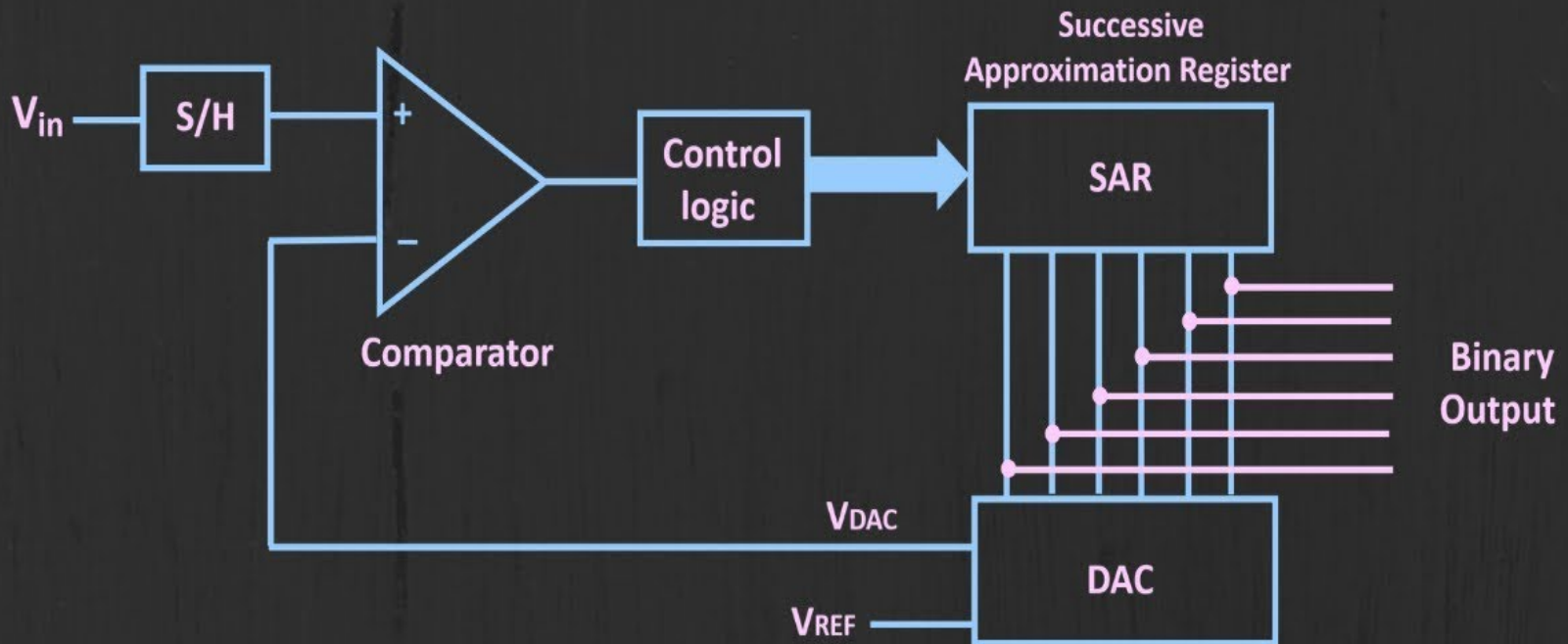
```
//Reads input through the ADC, and transfers to PC terminal
#include "mbed.h"

Serial pc(USBTX, USBRX);
AnalogIn Ain(p20);
float ADCdata;

int main() {
    pc.printf("ADC Data Values... \n\r");
    while (1) {
        ADCdata=Ain;
        pc.printf("%f \n\r",ADCdata);
        wait (0.5);
    }
}
```



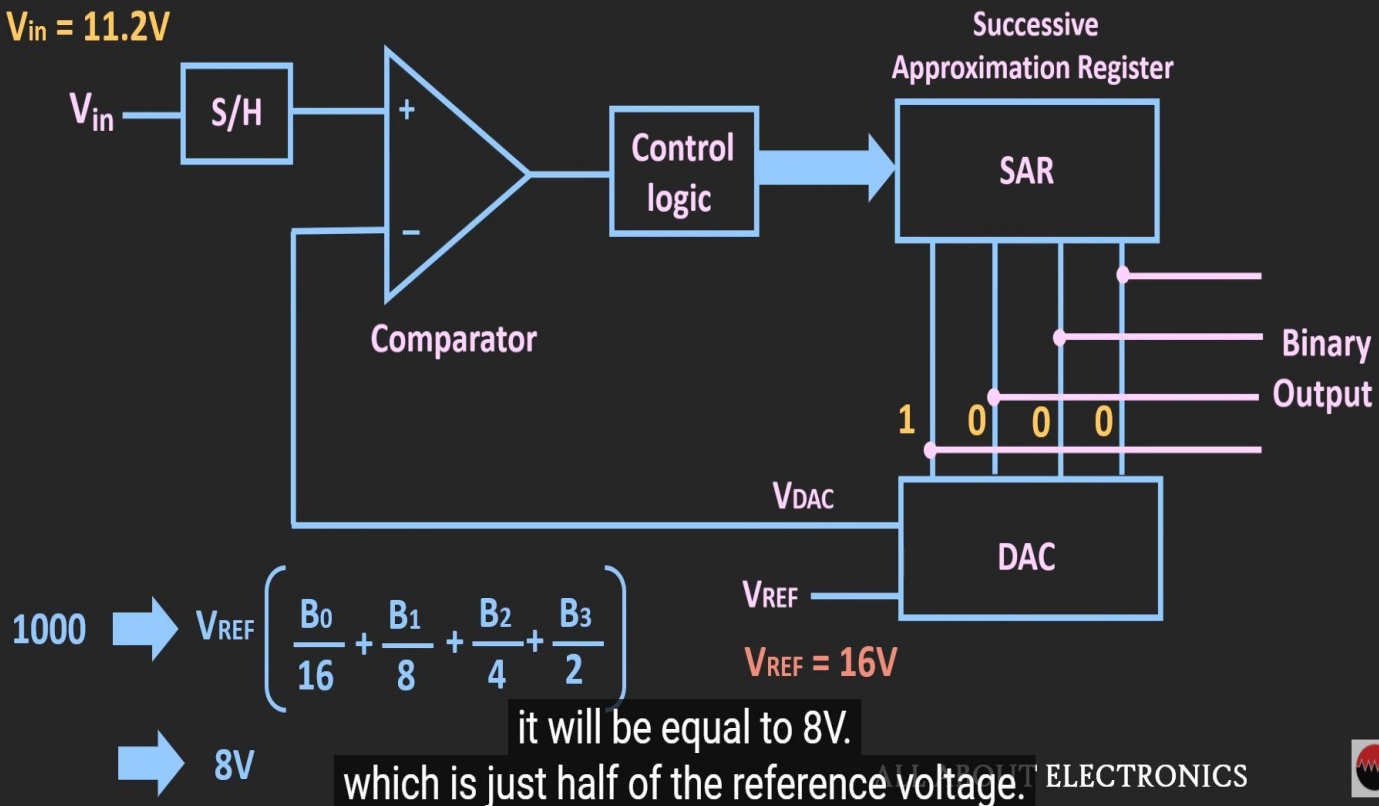
Successive Approximation ADC Explained



<https://www.youtube.com/watch?v=h0CGtr4SC9s>

Successive Approximation ADC

$V_{in} = 11.2V$





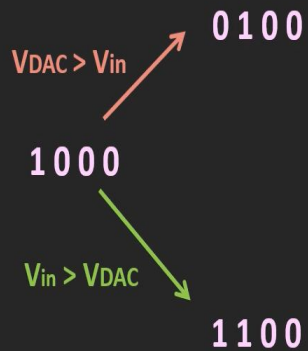
Successive Approximation ADC

Press Esc to exit full screen

$V_{REF} = 16V$

4-bit ADC

$V_{in} = 11.2V$



Now, here for the code of 1000, the output of the DAC is equal to 8V.

ALL ABOUT ELECTRONICS

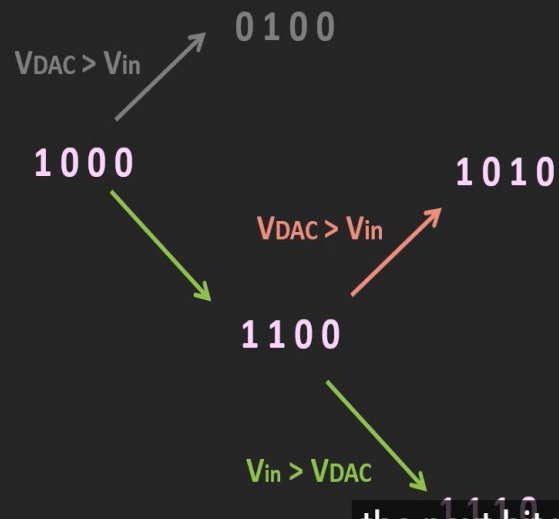


Successive Approximation ADC

$V_{REF} = 16V$

4-bit ADC

$V_{in} = 11.2V$



the next bit will be set to 1 for the new comparison.

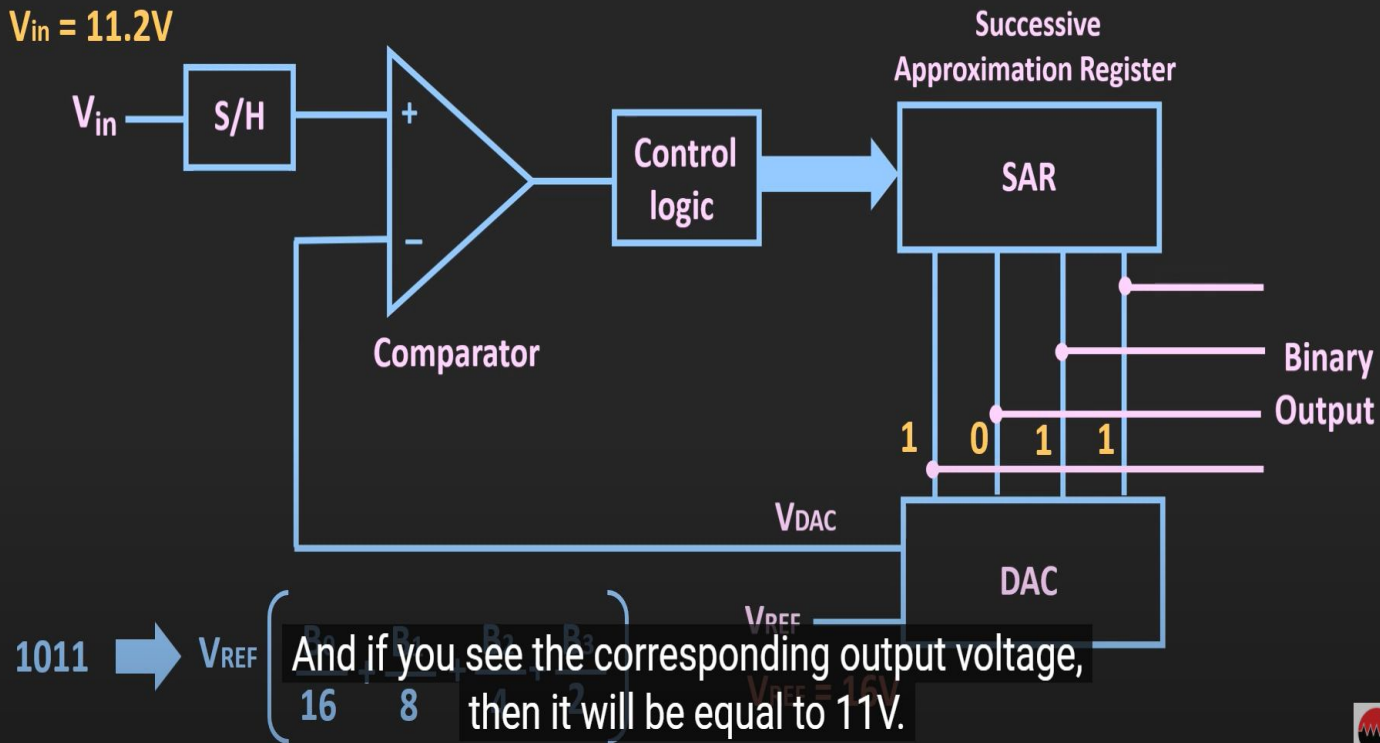
ALL ABOUT ELECTRONICS



Successive Approximation ADC Explained

Successive Approximation ADC

$V_{in} = 11.2V$



1011 → V_{REF} 16 8
11V
4:18 / 8:54 • Working of SAR ADC >

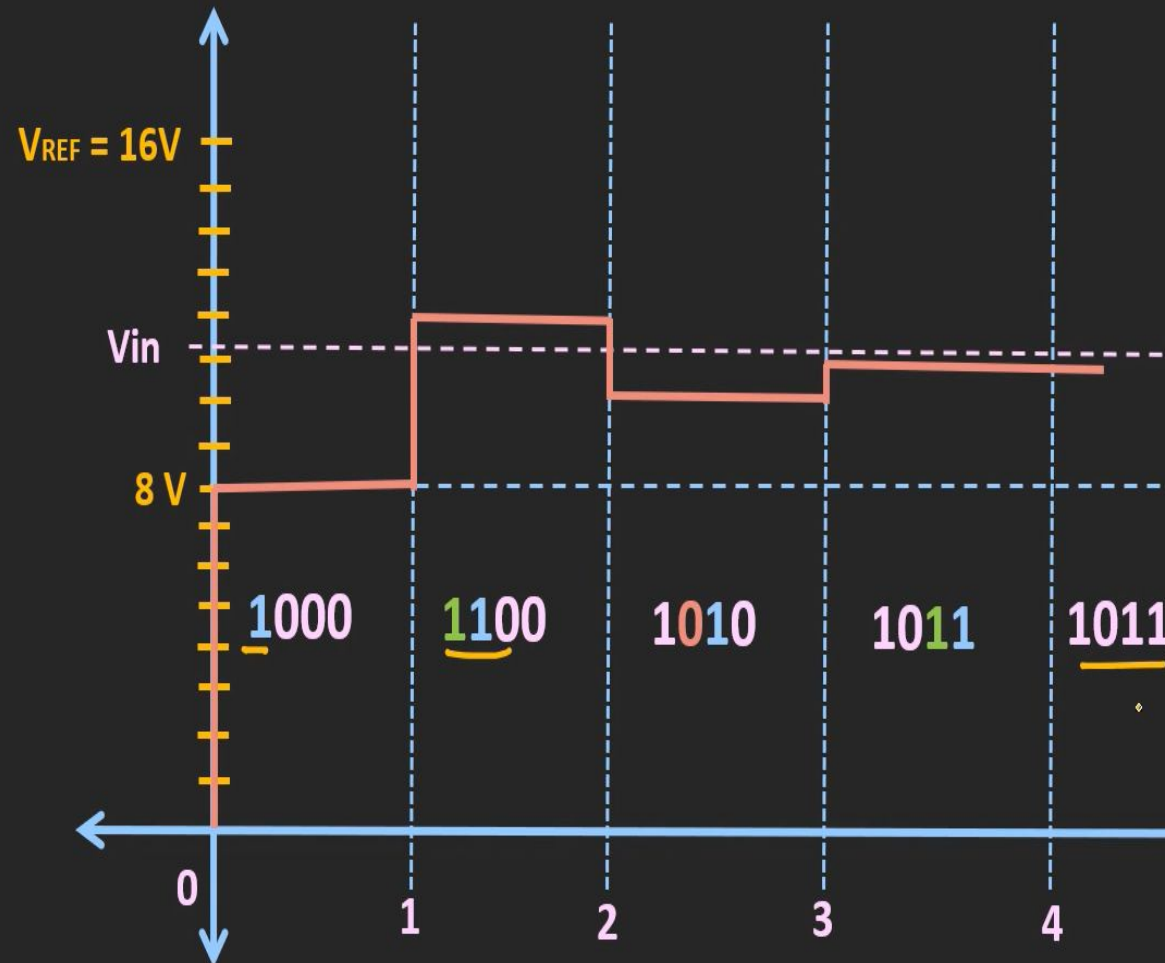
ALL ABOUT ELECTRONICS



$V_{REF} = 16V$

4-bit ADC

$V_{in} = 11.2V$



Successive Approximation ADC

Conversion Time : (Independent of the Input Voltage)

$$T_c = N \times T_{CLK}$$

Resolution: up to 20 bits (e.g. ADS8900B)

Conversion Speed: up to 10 Mega Samples per Second (MSPS)

(e.g.

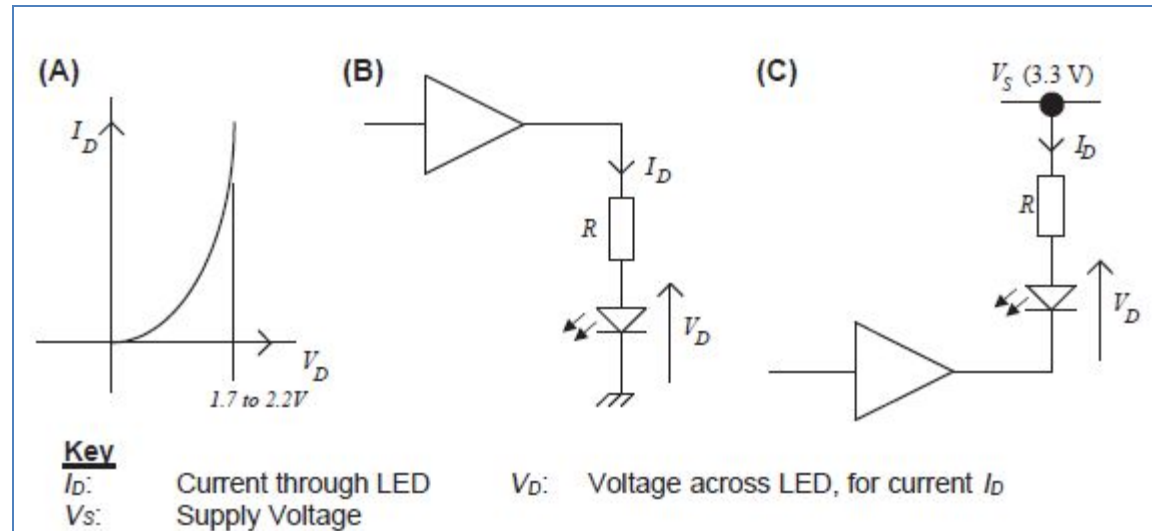
But some ADCs, for example, LTC2368 supports
up to 10 Mega Samples per Second.

ABOUT ELECTRONICS



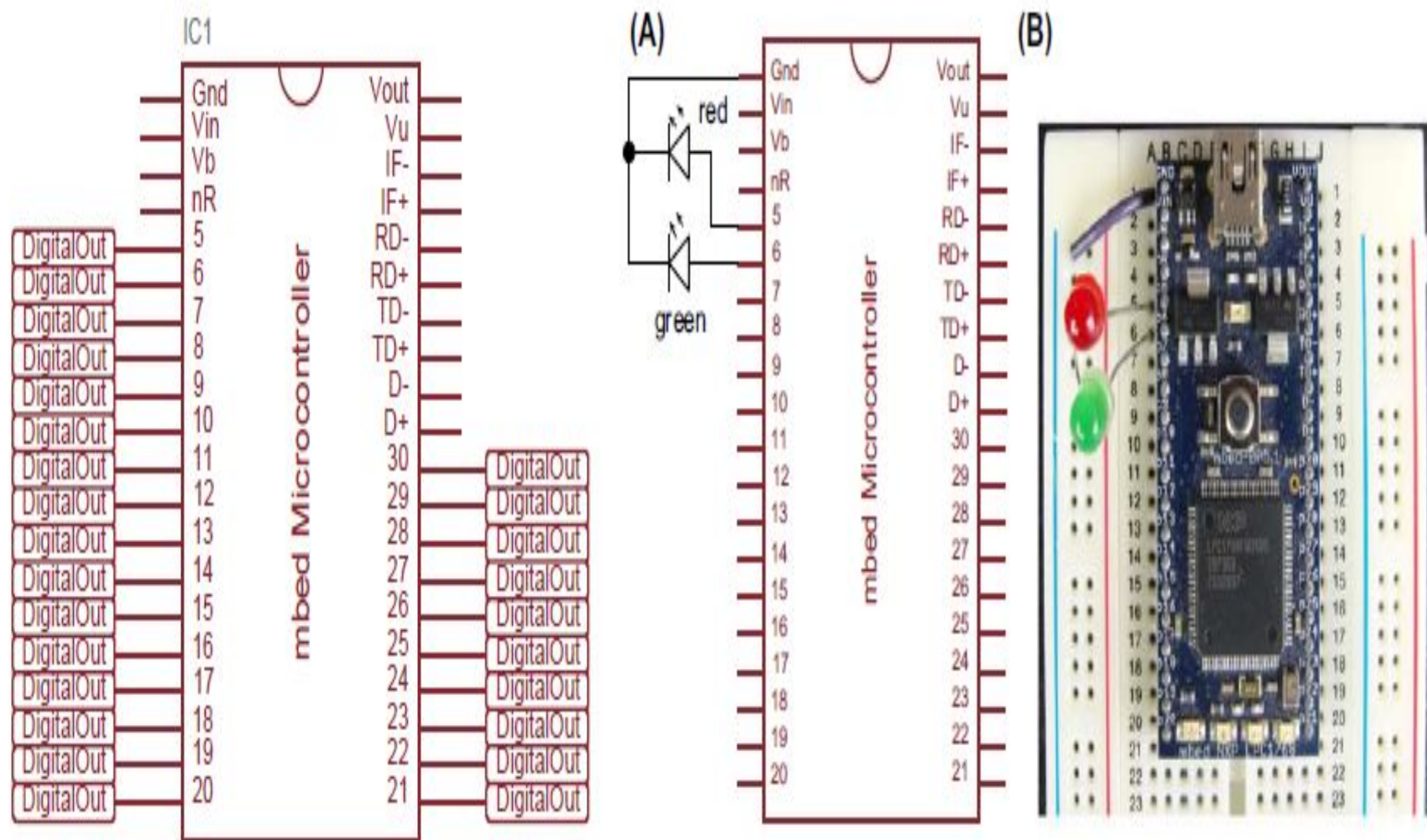
Digital Output on the mbed-Using Light Emitting Diodes

LEDs now appear in all manner of shapes and sizes, the most familiar perhaps being the single LED



Digital Output on the mbed-Using Light Emitting Diodes

The mbed, however, has 26 digital input/output (I/O) pins (pins 5-30) which can be configured either as digital inputs or outputs



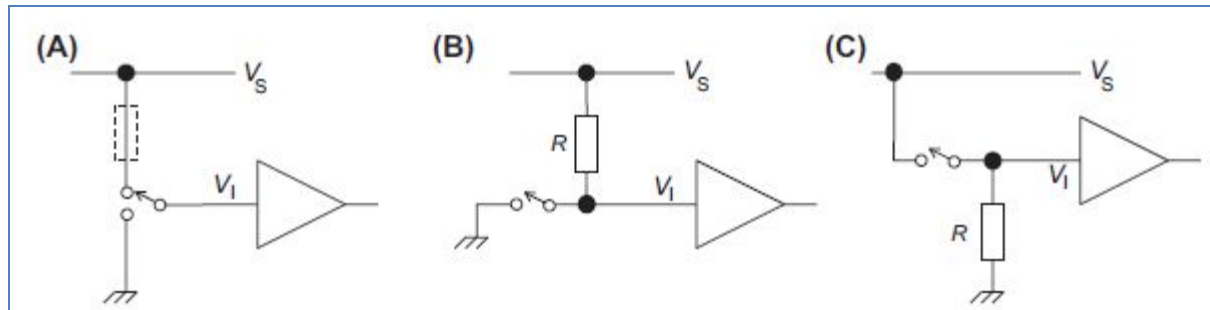
Digital Output on the mbed' Using mbed External Pins

The digital I/O pins are named and configured to output using DigitalOut

```
#include "mbed.h"
DigitalOut redled(p5); //define and name a
digital output on pin 5
DigitalOut greenled(p6); //define and name a
digital output on pin 6
int main() {
while(1) {
redled = 1;
greenled = 0;
wait(0.2);
redled = 0;
greenled = 1;
wait(0.2);
}
}
```

Functions	Usage
DigitalOut	Create a DigitalOut object, connected to the specified pin
write	Set the output, specified as 0 or 1 (int)
read	Return the output setting, represented as 0 or 1 (int)
mbed-defined operator: =	A shorthand for write
mbed-defined operator: int()	A shorthand for read

Using Digital Inputs-Connecting Switches to a Digital System



Using Digital Inputs-The DigitalIn API

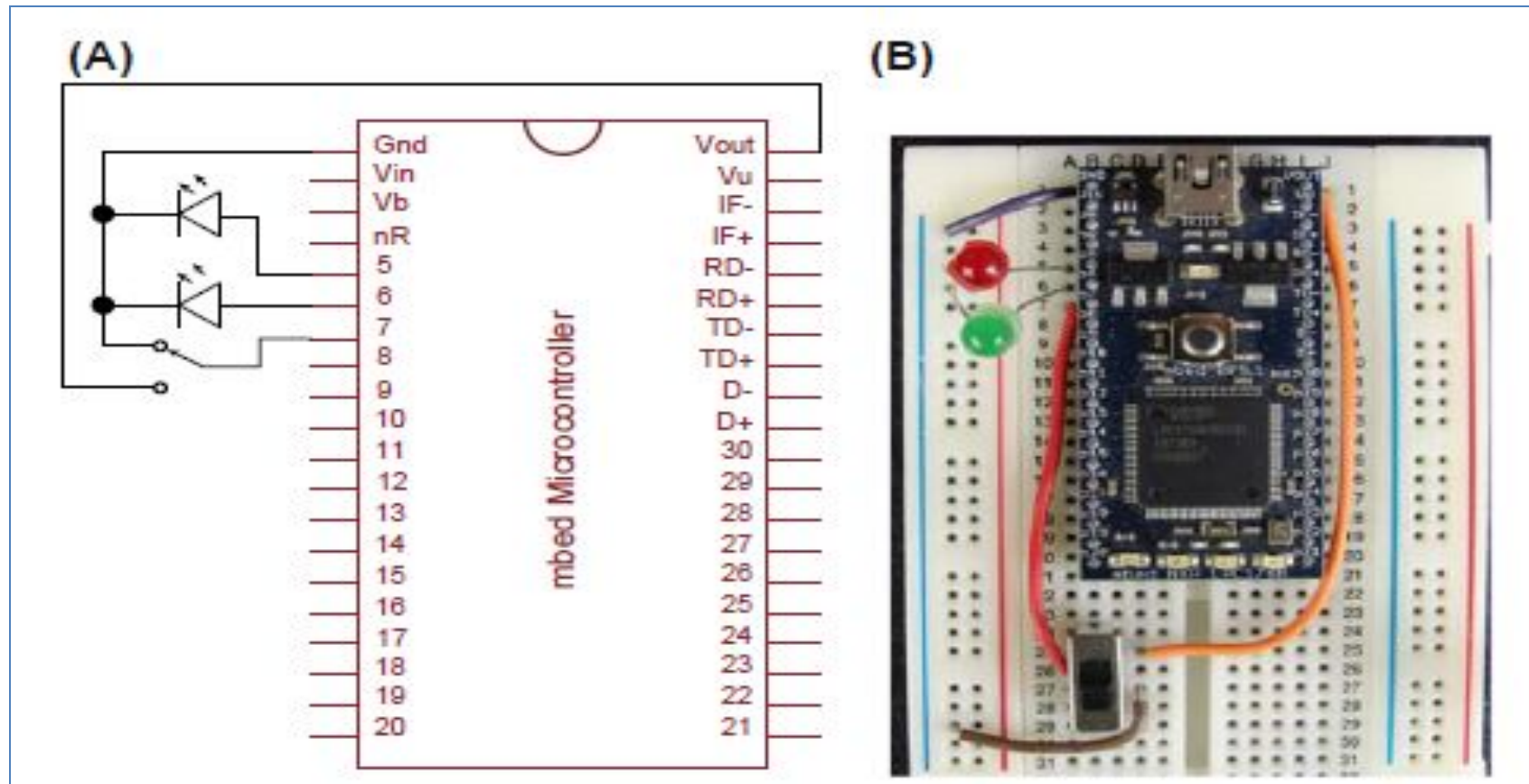
Functions	Usage
<code>DigitalIn</code>	Create a <code>DigitalIn</code> object, connected to the specified pin
<code>read</code>	Read the input, represented as 0 or 1 (int)
<code>mode</code>	Set the input pin mode, with parameter chosen from: <code>PullUp</code> , <code>PullDown</code> , <code>PullNone</code> , <code>OpenDrain</code>
mbed-defined operator: <code>int()</code>	A shorthand for <code>read()</code>

Using Digital Inputs- Using if to Respond to a Switch Input

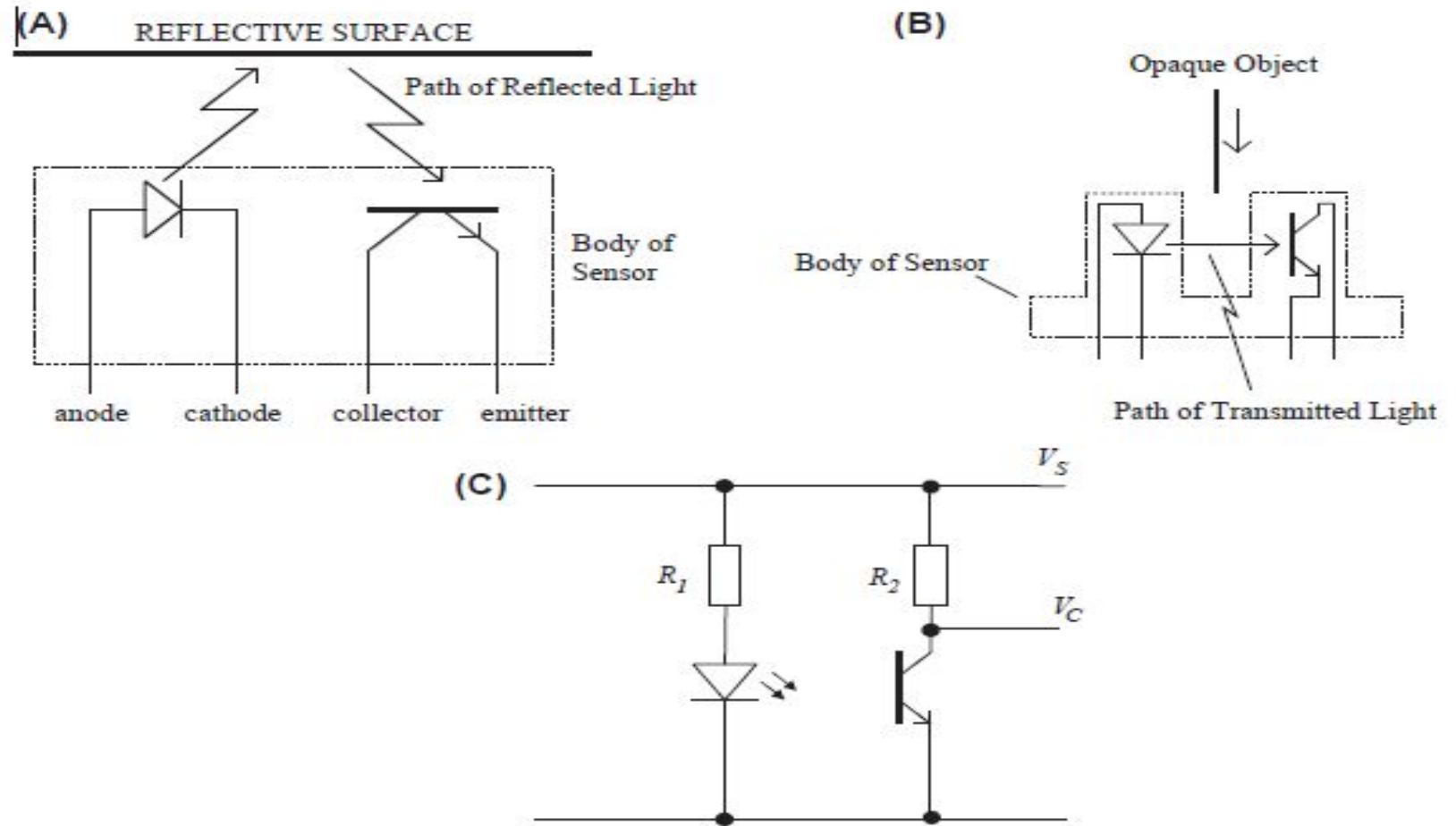
```
#include "mbed.h"
DigitalOut redled(p5);
DigitalOut greenled(p6);
DigitalIn switchinput(p7);
int main() {
while(1) {
if (switchinput==1) { //test value of
switchinput
//execute following block if switchinput
is 1
greenled = 0; //green led is off
redled = 1; // flash red led
wait(0.2);
redled = 0;
wait(0.2); }
```

```
else { //here if switchinput is 0
redled = 0; //red led is off
greenled = 1; // flash green led
wait(0.2);
greenled = 0;
wait(0.2);
} //end of else
} //end of while(1)
} //end of main
```

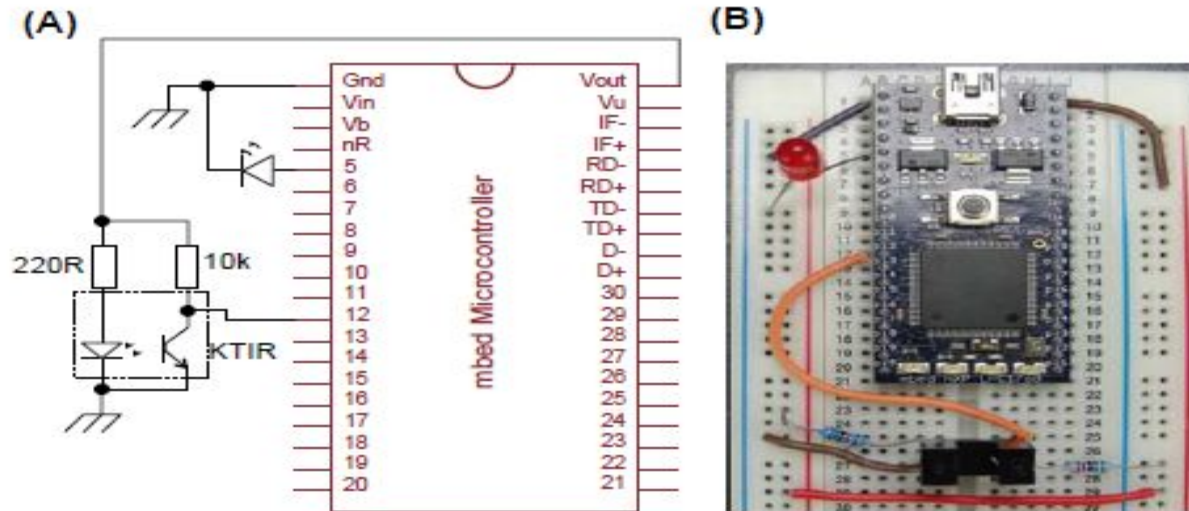
Using Digital Inputs- Using if to Respond to a Switch Input



Optoreflexive and Transmissive Sensors



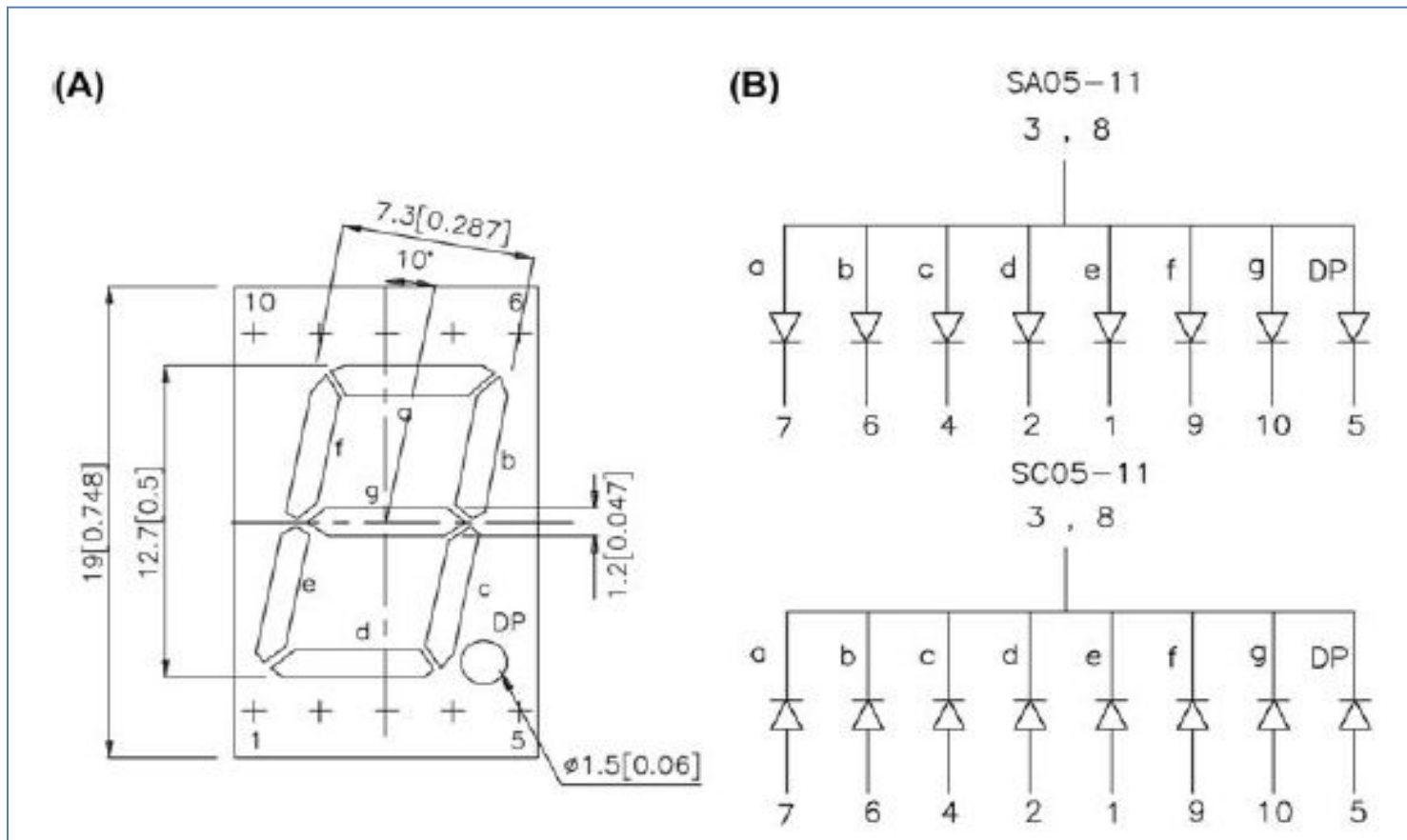
Optoreflective and Transmissive Sensors













```
#include "mbed.h"
DigitalOut redled(p5);
DigitalIn opto_switch(p12);
int main() {
while(1) {
if (opto_switch==1) //input = 1 if beam interrupted
redled = 1; //switch led on if beam interrupted
else
redled = 0; //led off if no interruption
} //end of while
}
```

Seven-Segment Displays

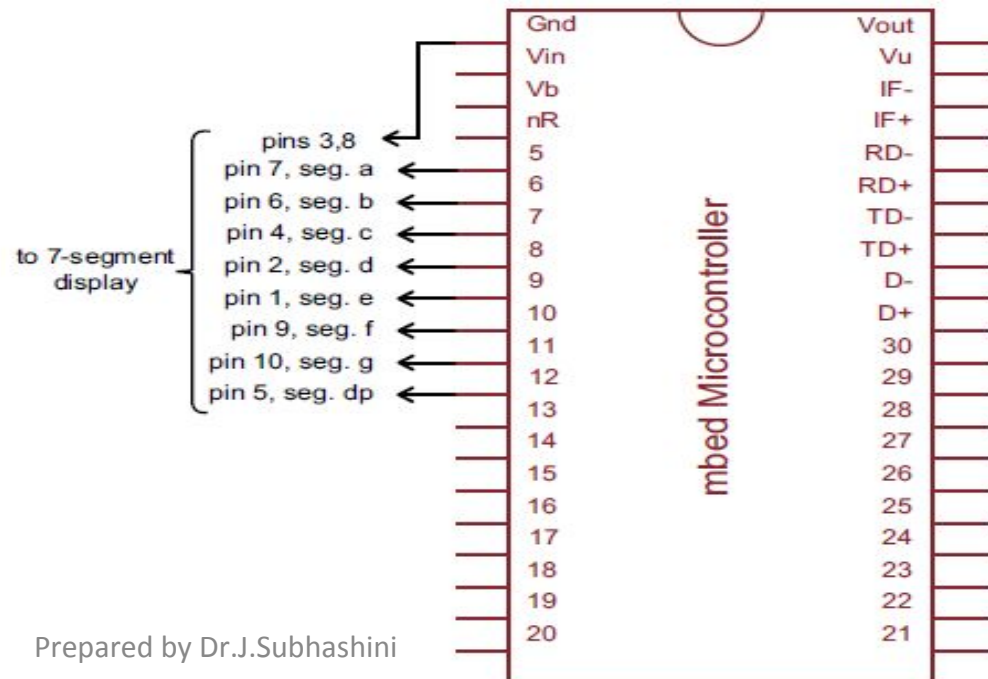
(MSB) 0 0 0 0 0 1 1 1 (LSB)



Seven-Segment Displays

Display Value	0	1	2	3	4	5	6	7	8	9
Segment Drive (B) (MSB) (LSB)	0011 1111	0000 0110	0101 1011	0100 1111	0110 0110	0110 1101	0111 1101	0000 0111	0111 1111	0110 1111
Segment Drive (B) (hex)	0x3F	0x06	0x5B	0x4F	0x66	0x6D	0x7D	0x07	0x7F	0x6F
Actual Display										

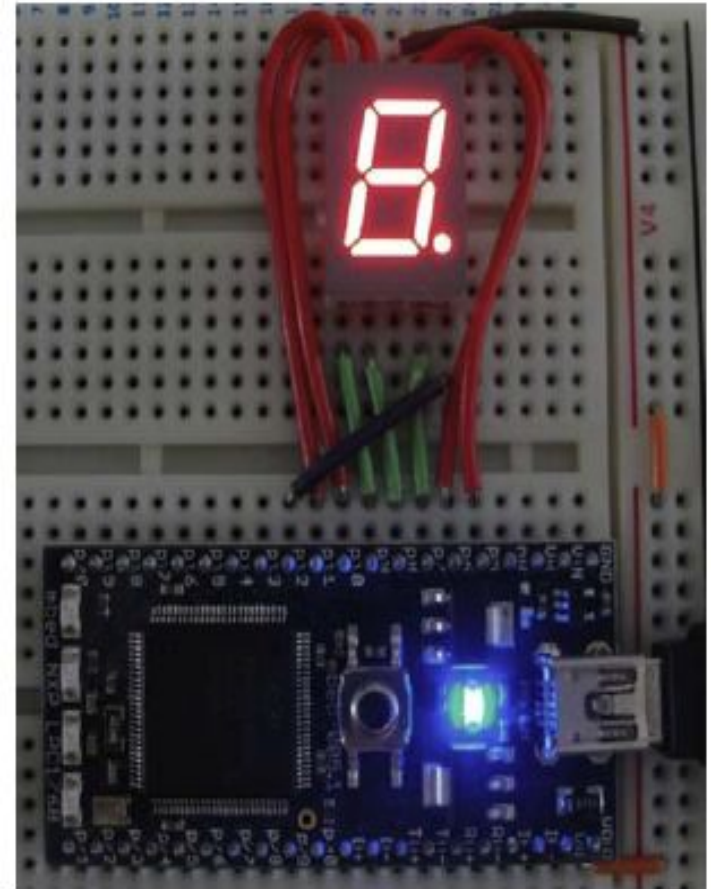
(MSB) DP g f e d c b a (LSB)



Seven-Segment Displays

(MSB) DP g f e d c b a (LSB)

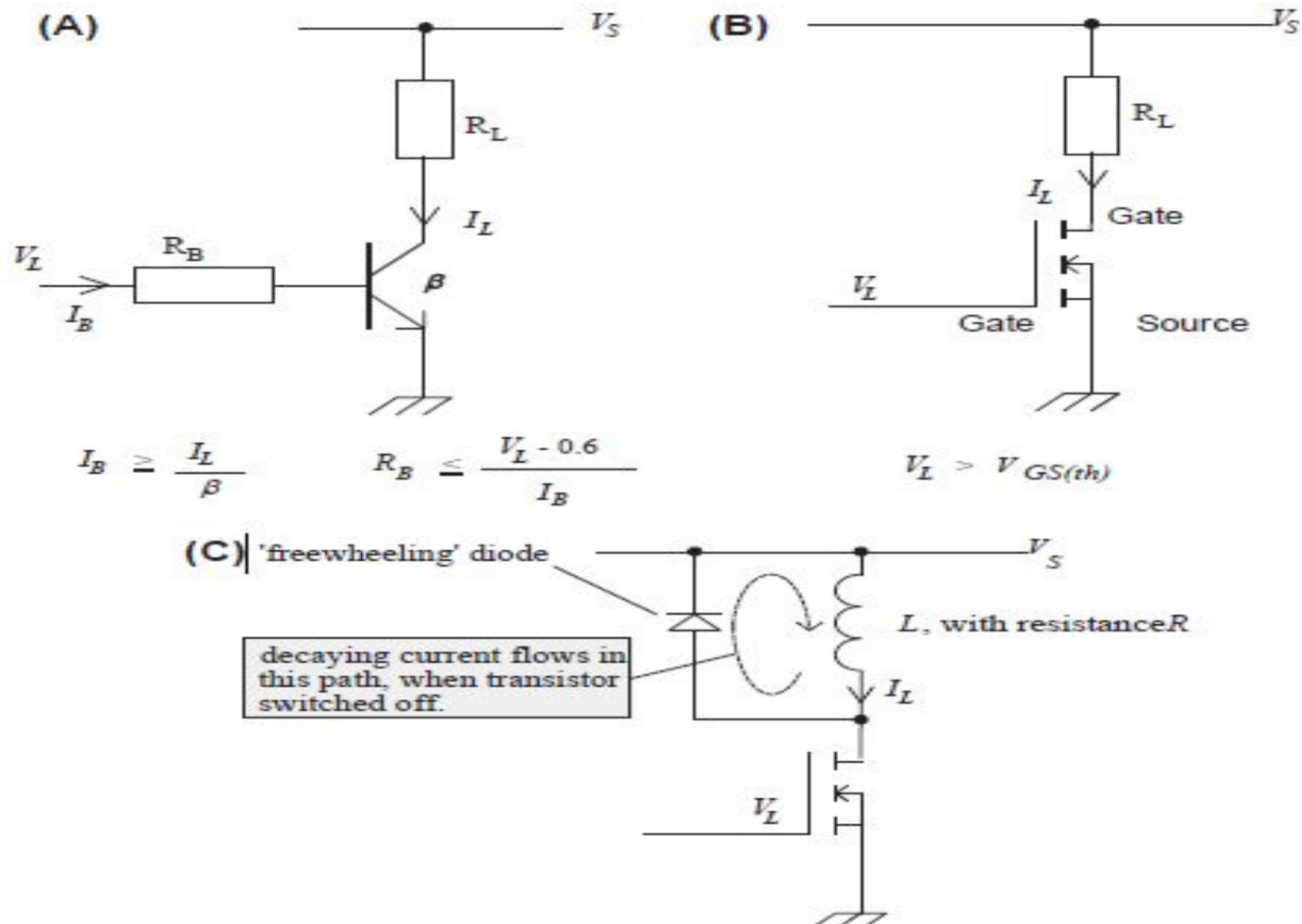
```
#include "mbed.h"
BusOut display(p5,p6,p7,p8,p9,p10,p11,p12); //
segments dp,a,b,c,d,e,f,g
int main() {
while(1) {
for(int i=0; i<4; i++) {
switch (i){
case 0: display = 0x3F; break; //display 0
case 1: display = 0x06; break; //display 1
case 2: display = 0x5B; break;
case 3: display = 0x4F; break;
} //end of switch
wait(0.2);
} //end of for
} //end of while
}
```



Switching Larger DC Loads

a port pin can source up to around 40 mA.

DC motor needs to run from a higher voltage, then an interface circuit will be needed.

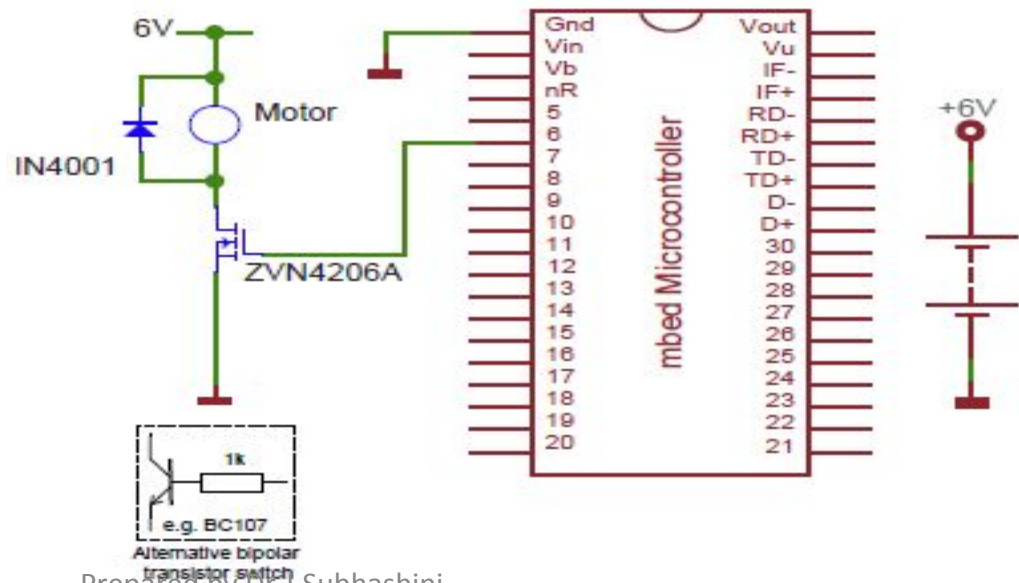


Switching Larger DC Loads

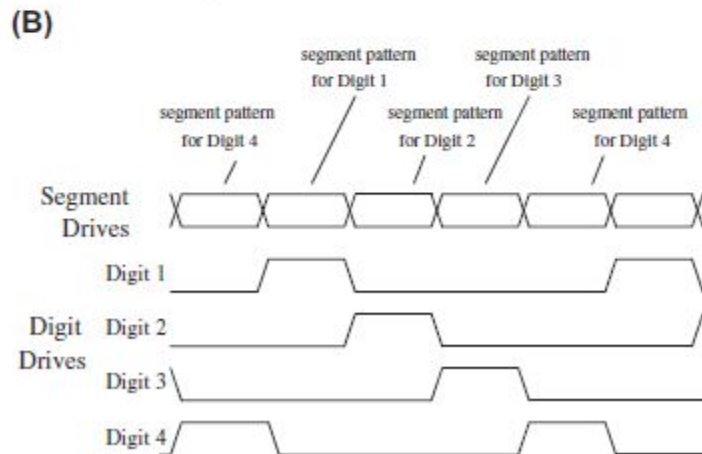
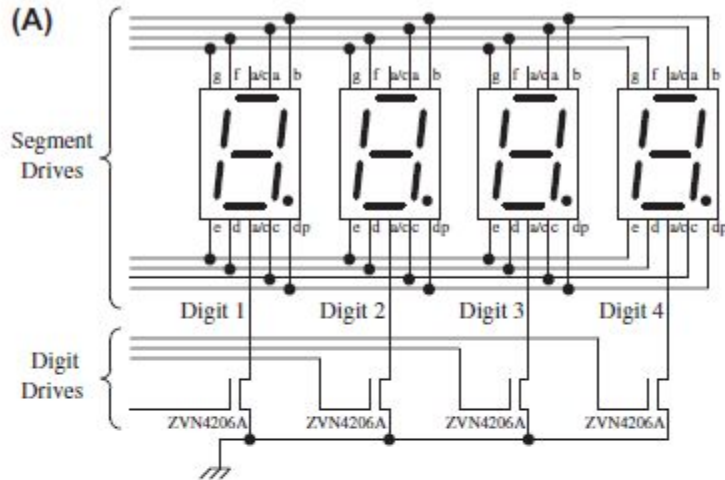
- A resistive load, like a motor or heater using a bipolar transistor requires load current, and the value of the current gain (β) of the transistor
- Devices in today's electronics using MOSFET requires threshold gate-to-source voltage, above which the transistor switches on.
- Inductive load, like a solenoid or DC requires freewheeling diode.
- This is needed because any inductance with current flowing in it stores energy in the magnetic field which surrounds it.
- When that current is interrupted, in this case, by the transistor being switched off, the energy has to be returned to the circuit.
- This happens through the diode, which allows decaying current to continue to circulate.
- If the diode is not included then a high voltage transient occurs, which can/will destroy the FET.

Switching Larger DC Loads

Characteristic	ZVN4206A
Maximum drain-source voltage, V_{DS}	60 V
Maximum gate-source threshold, $V_{GS(th)}$	3 V
Maximum drain-source resistance when 'On,' $R_{DS(on)}$	1.5 Ω
Maximum continuous drain current, I_D	600 mA
Maximum power dissipation	0.7 W
Input capacitance	100 pF



Switching Multiple Seven-Segment Displays



Mini Project: Letter Counter

Use the slotted optosensor, push-button switch and one seven-segment LED display to create a simple letter counter.

Increment the number on the display by one every time a letter passes the sensor.

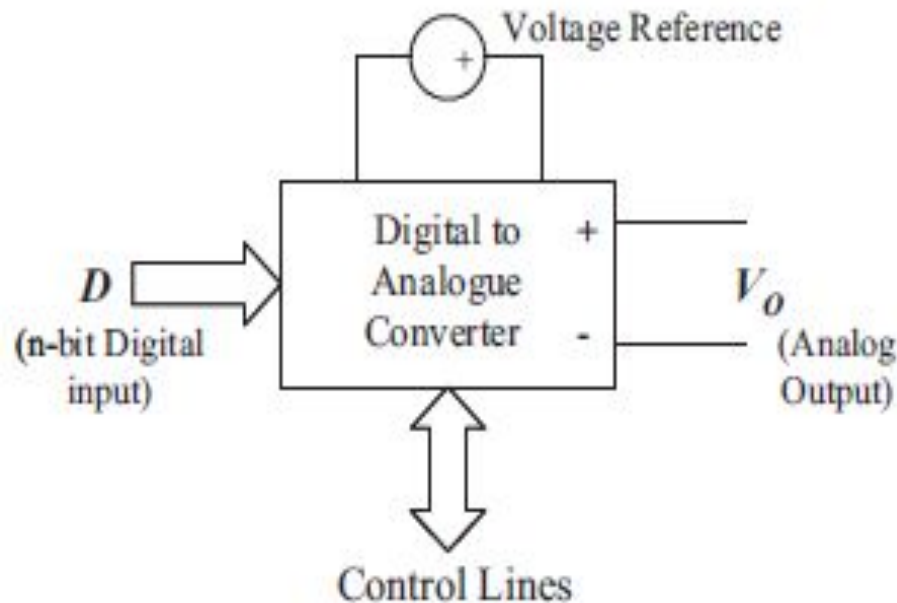
Clear the display when the push button is pressed.

Analog Output-DAC

After processing the data, they may then need to convert digital data back to analog form, for example, to drive a loudspeaker or DC motor

A digital-to-analog converter is a circuit which converts a binary input number into an analog output.

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



V_r is the value of the voltage reference,
 D is the value of the binary input word,
 n is the number of bits in that word, and
 V_o is the output voltage

Analog Output-DAC

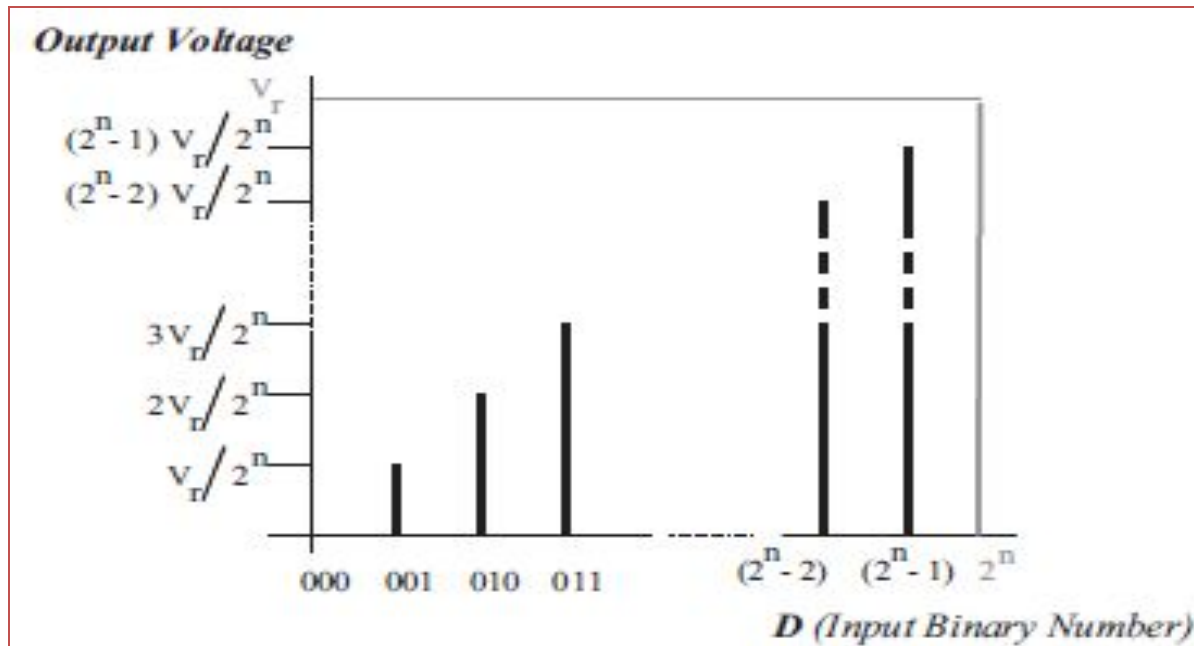
The number of possible output values is given by 2^n , and the step size by $V_r / 2^n$; this is called the resolution.

The maximum possible output value occurs when $D = (2^n - 1)$

The range of the DAC is the difference between its maximum and minimum output values.

For example, a 6-bit DAC will have 64 possible output values; if it has a 3.2 V reference, it will have a resolution (step size) of 50 mV.

Analog Output-DAC



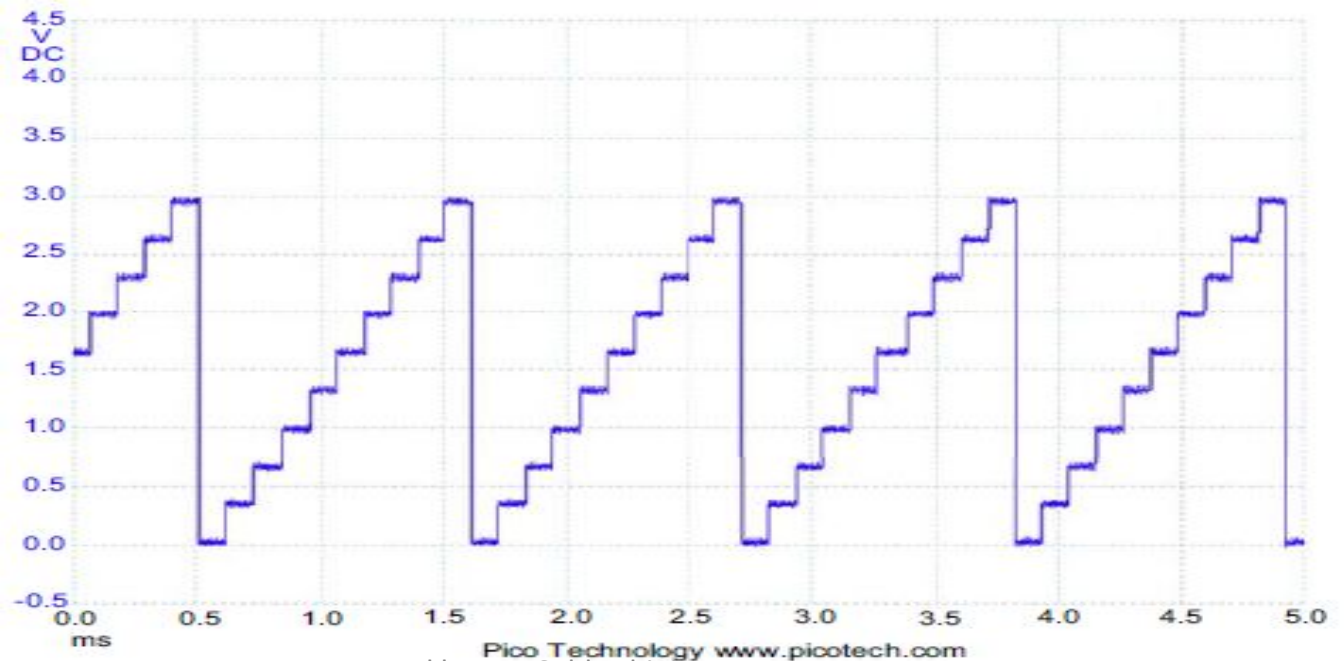
Functions	Usage
AnalogOut	Create an AnalogOut object connected to the specified pin
write	Set the output voltage, specified as a percentage (float)
write_u16	Set the output voltage, represented as an unsigned short in the range [0x0, 0xFFFF]
read	Return the current output voltage setting, measured as a percentage (float)
mbed-defined operator: =	An operator shorthand for write()

Analog Output-DAC

```
/*Program Example 4.1: Three values of DAC are output in turn
on Pin 18. Read the output on a DVM. */
#include "mbed.h"
AnalogOut Aout(p18); //create an analog output on pin 18
int main() {
while(1) {
Aout=0.25; //  $0.25 \times 3.3V = 0.825V$ 
wait(2);
Aout=0.5; //  $0.5 \times 3.3V = 1.65V$ 
wait(2);
Aout=0.75; //  $0.75 \times 3.3V = 2.475V$ 
wait(2);
}
}
```

Analog Output-DAC

```
/*Program Example 4.2: Saw tooth waveform on AC output. View on oscilloscope*/  
#include "mbed.h"  
AnalogOut Aout(p18);  
float i;  
int main() {  
    while(1){  
        for (i=0;i<1;i=i+0.1){ // i is incremented in steps of 0.1  
            Aout=i;  
            wait(0.001); // wait 1 millisecond}}}
```



Another Form of Analog Output: Pulse Width Modulation

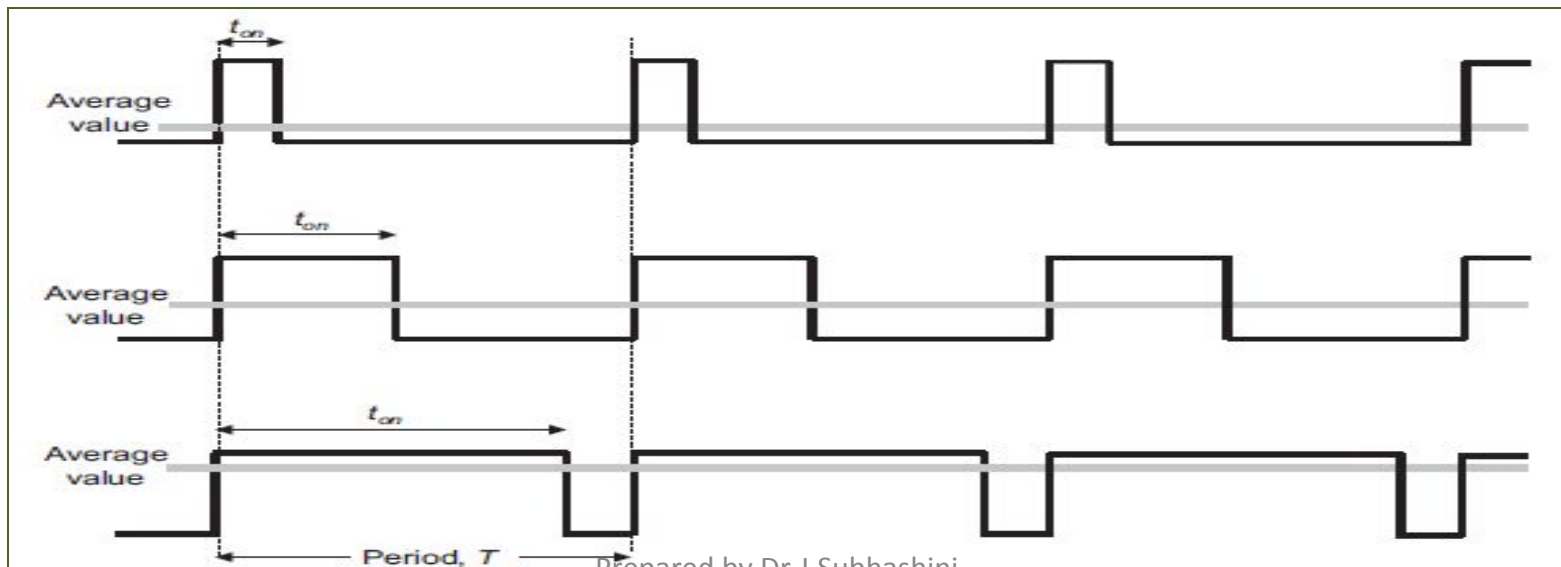
Pulse Width Modulation on the mbed

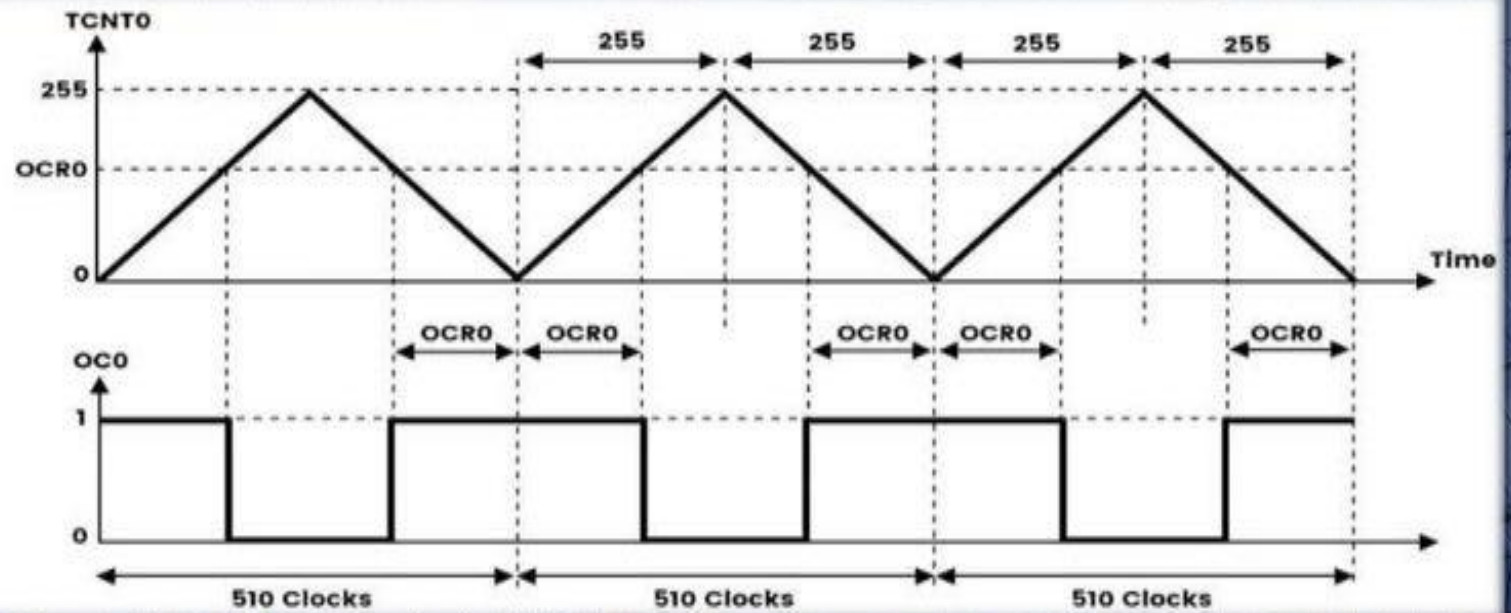
Pulse width modulation (PWM), an alternative, represents a neat and remarkably simple way of getting a rectangular digital waveform to control an analog variable, usually voltage or current.

PWM control is used in a variety of applications, ranging from telecommunications to robotic control.

Pulse width modulation (PWM), an alternative, represents a neat and remarkably 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, ranging from telecommunications to robotic control.





Another Form of Analog Output: Pulse Width Modulation

The duty cycle is the proportion of time that the pulse is “on” or “high” and is expressed as a percentage,

$$\text{duty cycle} = \frac{\text{pulse on time}}{\text{pulse period}} * 100\% = \frac{t_{on}}{T} * 100\%$$

A 100% duty cycle therefore means “continuously on” and a 0% duty cycle means “continuously off.”

PWM streams are easily generated by digital counters and comparators, which can readily be designed into a microcontroller.

They can also be produced simply by program loops and a standard digital output, with no dedicated hardware at all

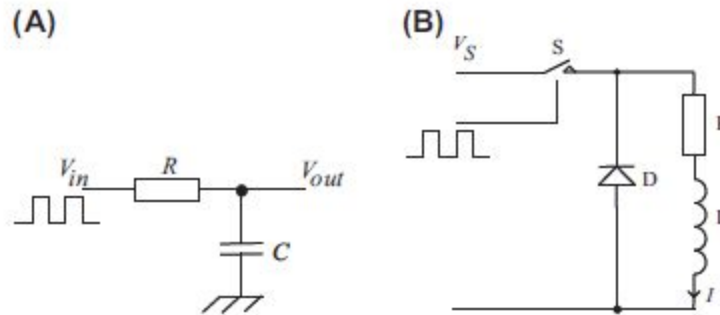
Another Form of Analog Output: Pulse Width Modulation

Pulse Width Modulation on the mbed

Functions	Usage
<code>PwmOut</code>	Create a <code>PwmOut</code> object connected to the specified pin
<code>write</code>	Set the output duty cycle, specified as a normalized float (0.0–1.0)
<code>read</code>	Return the current output duty cycle setting, measured as a normalized float (0.0–1.0)
<code>period</code>	Set the PWM period, specified in seconds (float), keeping the duty cycle the same.
<code>period_ms</code>	Set the PWM period, specified in milliseconds (int), keeping the duty cycle the same.
<code>period_us</code>	Set the PWM period, specified in microseconds (int), keeping the duty cycle the same.
<code>pulsewidth</code>	Set the PWM pulse width, specified in seconds (float), keeping the period the same.
<code>pulsewidth_ms</code>	Set the PWM pulse width, specified in milliseconds (int), keeping the period the same.
<code>pulsewidth_us</code>	Set the PWM pulse width, specified microseconds (int), keeping the period the same.
<code>mbed-defined operator: =</code>	An operator shorthand for <code>write()</code>

Another Form of Analog Output: Pulse Width Modulation

Electrically, we can use a low-pass filter, e.g., the resistor-capacitor combination



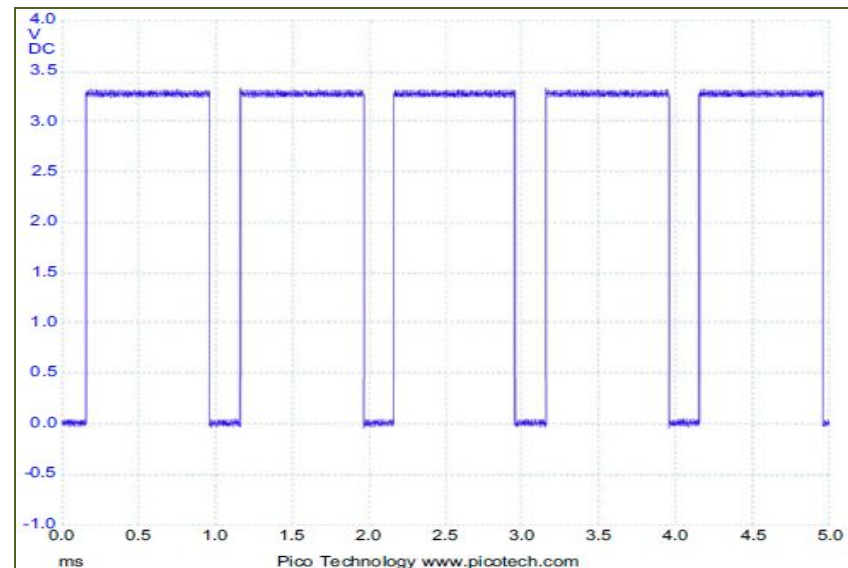
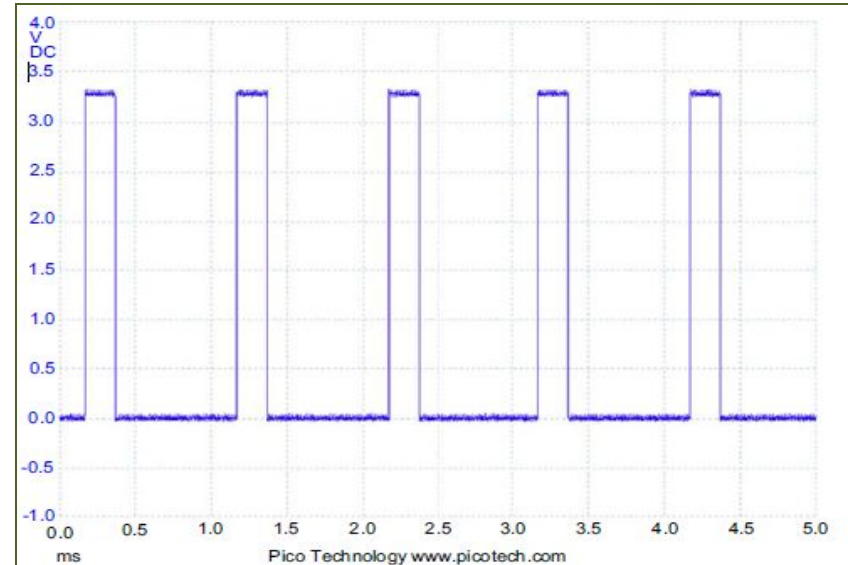
Pulse Width Modulation on the mbed

```
/*Sets PWM source to fixed frequency and duty cycle. Observe output
on oscilloscope.*/
#include "mbed.h"
PwmOut PWM1(p21); //create a PWM output called PWM1 on pin 21
int main() {
    PWM1.period(0.010); // set PWM period to 10 ms
    PWM1=0.5; // set duty cycle to 50%
}
```

Another Form of Analog Output: Pulse Width Modulation

Speed Control of a Small Motor

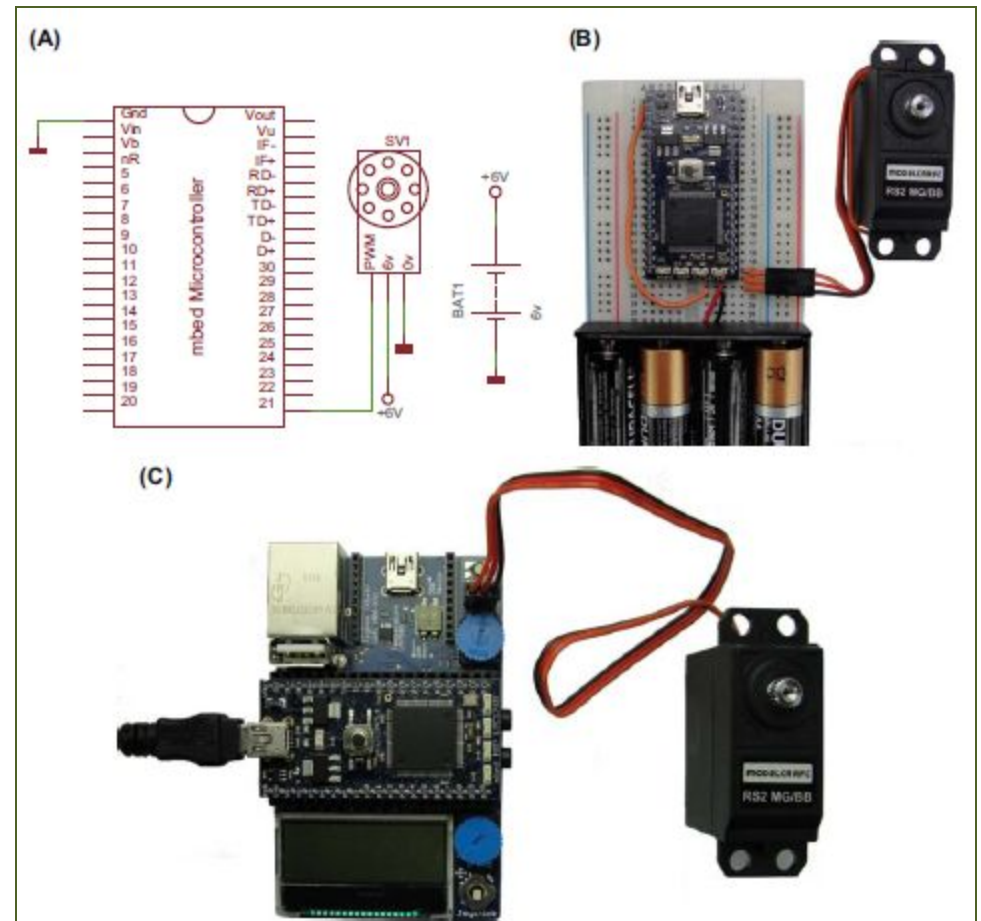
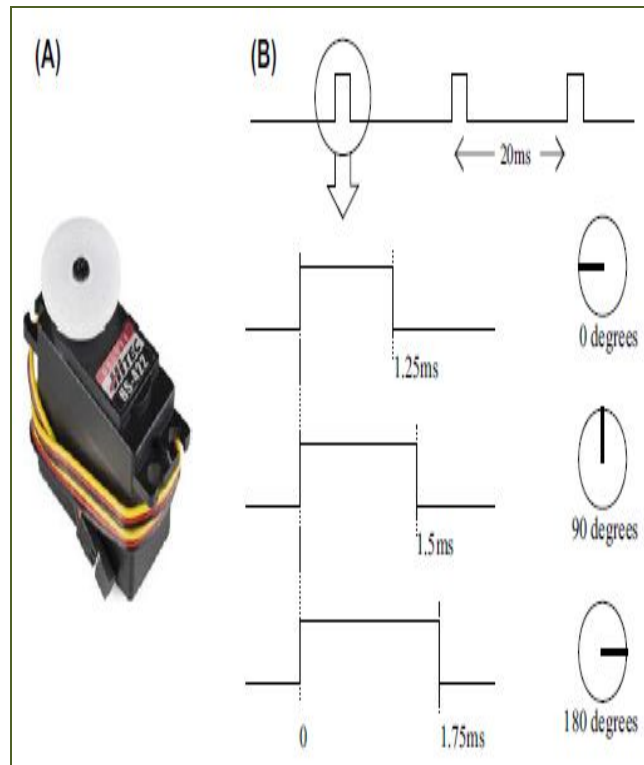
```
/*Program Example 4.5: PWM control  
to DC motor is repeatedly ramped */  
#include "mbed.h"  
PwmOut PWM1(p21);  
float i;  
int main() {  
    PWM1.period(0.010); //set PWM  
    period to 10 ms  
    while(1) {  
        for (i=0;i<1;i=i+0.01) {  
            PWM1=i; // update PWM duty cycle  
            wait(0.2);  
        }  
    }  
}
```



Another Form of Analog Output: Pulse Width Modulation

Servo Control

A servo is a small, lightweight, rotary position control device, often used in radio controlled cars and aircraft to control angular position of variables such as steering, elevators, and rudders



Another Form of Analog Output: Pulse Width Modulation

Producing Audio Output

A servo is a small, lightweight, rotary position control device, often used in radio controlled cars and aircraft to control angular position of variables such as steering, elevators, and rudders

```
#include "mbed.h" PwmOut speaker(p21);
void play_tone(float frequency, float volume, int interval, int rest) {
    speaker.period(1.0 / frequency);
    speaker = volume;
    wait(interval);
    speaker = 0.0;
    wait(rest);}
int main(){
    while(1) {
        play_tone(659.0, 0.5, 1, 1);    play_tone(554.0, 0.5, 1, 1);
        play_tone(659.0, 0.5, 1, 1);    play_tone(554.0, 0.5, 1, 1);
        play_tone(440.0, 0.5, 1, 1);    play_tone(494.0, 0.5, 1, 0.5);
        play_tone(554.0, 0.5, 1, 0.5);    play_tone(587.0, 0.5, 1, 1);
        play_tone(494.0, 0.5, 1, 1);    play_tone(659.0, 0.5, 1, 1);
        play_tone(554.0, 0.5, 1, 1);    play_tone(440.0, 0.5, 1, 2);    }}
```