Technical Workshop

Academic High Altitude Conference

Ethan Harstad Matthew Nelson

Stratospheric Ballooning Association

June 23-24, 2014

aerodyne labs

Outline

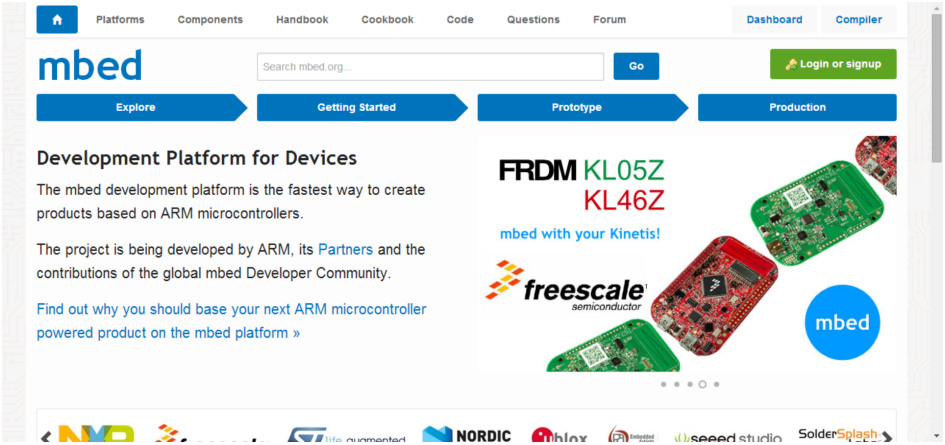
What is mbed?

The mbed platform is a collection of open source hardware and software to allow rapid ARM based prototyping

- Professional online compiler lets you work from any computer
- Integrated version control system lets you easily find and use libraries
- CMSIS based APIs let you work high level or bare metal
- Hardware abstraction layer insulates your application from hardware changes

Essentially a high performance Arduino with highly integrated tools to save you time!

Register on mbed



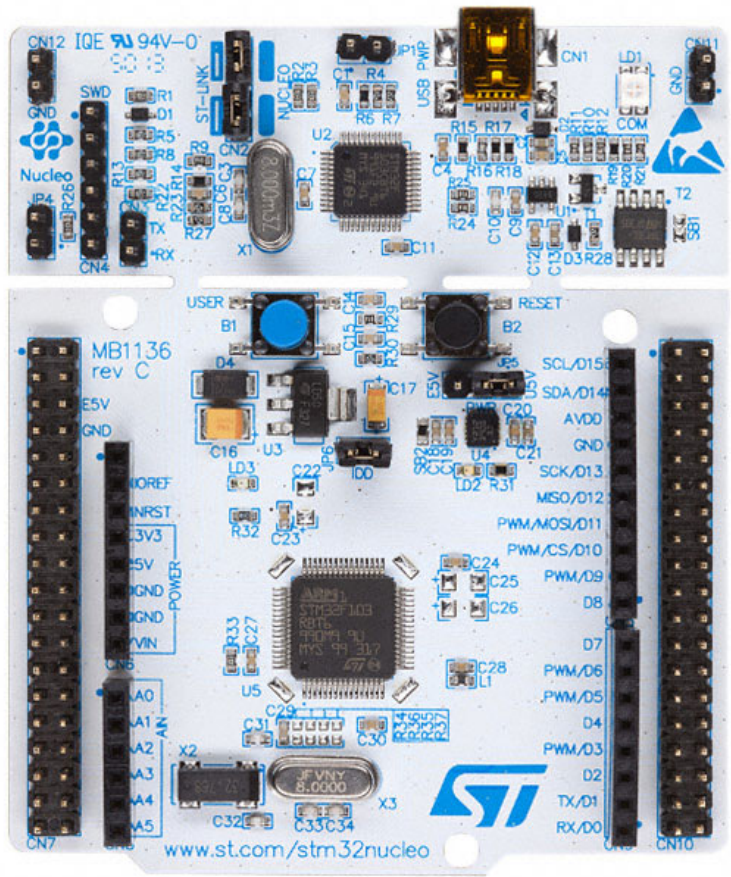
1. Navigate to `http://www.mbed.org`
2. Click the green "Login or signup" button
3. Click the "Signup" button
4. Follow the prompts
5. Confirm your e-mail address

Everyone should have an mbed account. You can create a team to share code between members of your organization.

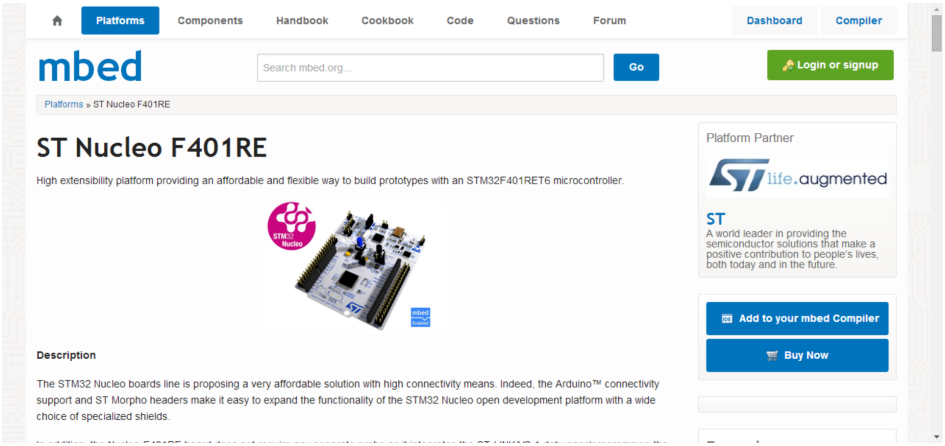
Nucleo Development Board

The Nucleo development board combines a USB programmer with a powerful STM32 processor and Arduino compatible headers

- ARM Cortex-M4 with FPU at 84 MHz
- 512 KBytes of flash memory
- 12 bit ADC at 2.4 Msps with up to 10 channels
- Up 3xUART, 3xI2C, 4xSPI interfaces



Add Nucleo to Your Account



1. Connect your Nucleo to your computer
2. Open the external drive the connects
3. Double click the mbed.htm file
4. Click "Add to your mbed Compiler"

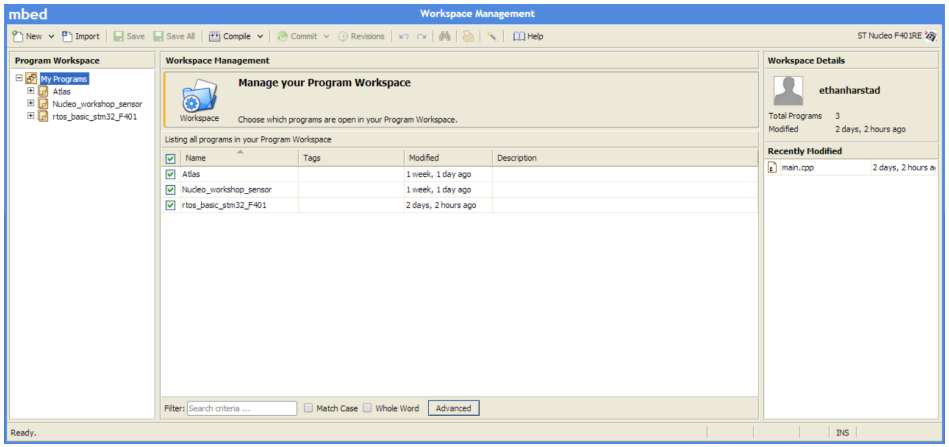
Note

You only need to do this once per account!

Install Drivers

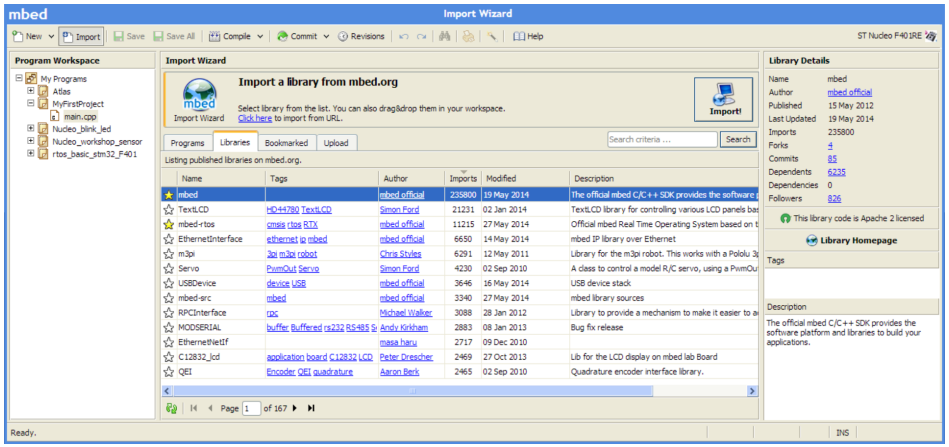
Outline

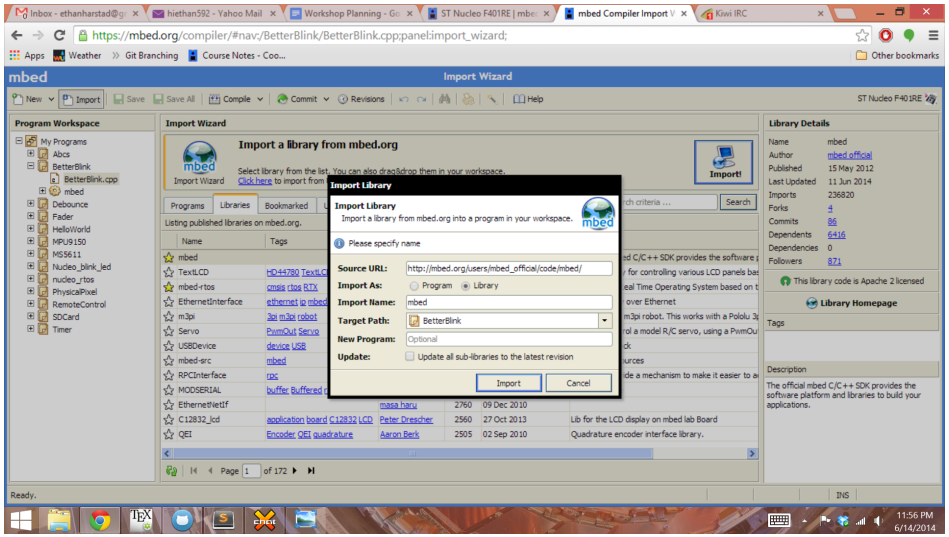
Creating a Program



1. Navigate to the mbed homepage and click the "Compiler" button
2. Click the "New" button and select "New Program"
3. Change the Template field to "Empty Program"
4. Give your program a name and click "OK"

Importing a Library





1. Click the "Import" button
2. Search for the "mbed" library
3. Select the library
4. Click the "Import!" button
5. Make sure the Target Path is your project root
6. Click "Import" one last time

Creating a New File

1. Click the root directory of your project to select it
2. Click the arrow next to "New" and select "New File"
3. Name the file "main.cpp" and click "OK"

Warning

Be sure to select the folder you want your file in before creating it!

You can name this file anything but it must have the .cpp extension. It is suggested that the main file be named "main" or the same as your project name (important later on).

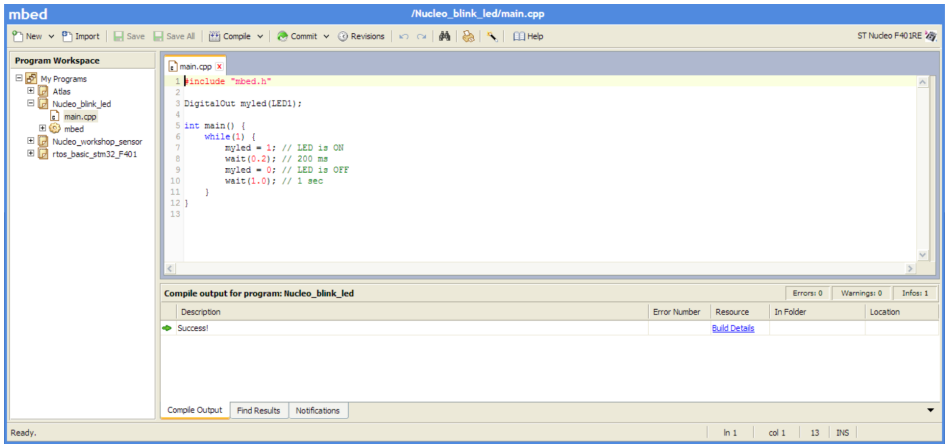
Program Structure

```
1  /* Includes */
2  #include "mbed.h"
3
4  /* Global Variable
5     Declarations     */
6
7  /* Main Function */
8  int main() {
9    // Program code
10 }
```

main.cpp

- Line 2 includes the mbed library, **every** mbed program needs this.
- Line 8 is the main function, this is the entry point into your program. **Every** program needs a main function.
- Line 9 is a comment, everything after // is ignored.
- Line 4-5 is a multi line comment, everything between /* */ is also ignored.

Compiling



1. Click "Compile" (Ctrl D) to compile your program
2. A *.bin file will be downloaded
3. Move the downloaded file to the Nucleo drive
4. The Nucleo will flash red and green while programming
5. When the lights stop, your program has started successfully!

Tip

Set your browsers download location to the Nucleo to save time

You can also click "Build Only" (Ctrl B) to simply test your code

Outline

While Loops

```
1 while(conditional) {
2   // Code to execute when true
3 }
```

While loops executes the code contained within them while their conditional statement is true.

Example

A main function should (almost) never exit:

```
1 int main() {
2   while(true) {
3     // Main loop code, runs forever
4   }
5 }
```

Your Program

```
1  /* Includes */
2  #include "mbed.h"
3
4  /* Global Variable
5     Declarations     */
6
7  /* Main Function */
8  int main() {
9    while(true) {
10     // Program code
11   }
12 }
```

main.cpp

- Add an infinite while loop to your main function to prevent your program from ending.

Digital Output

DigitalOut constructor:

```
DigitalOut(PinName pin)
```

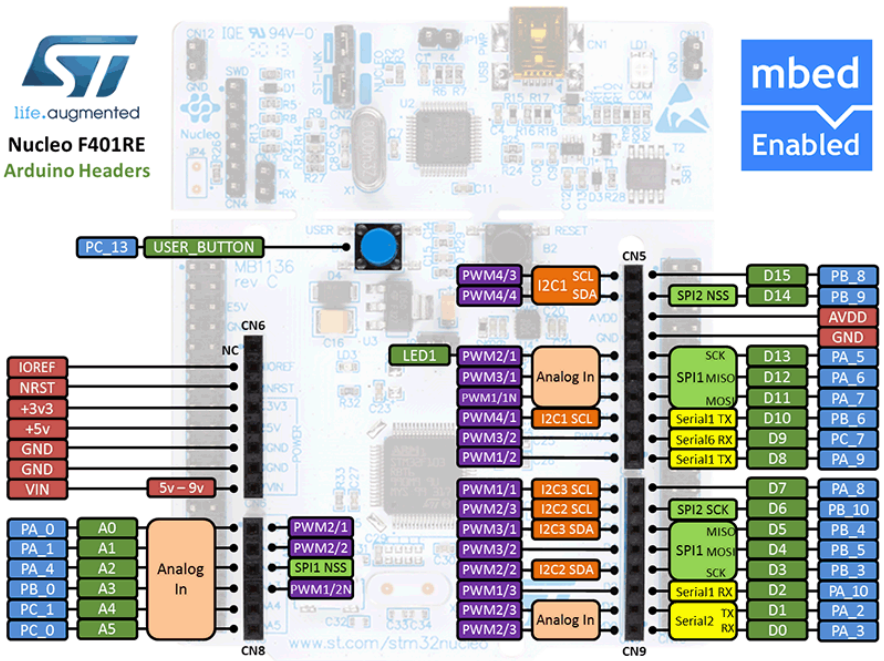
It creates an object attached to the given pin. Anytime you see PinName, use a name from the images on the next slide.
You can assign a value to the object using the equals sign. 1 turns the pin on while a 0 turns the pin off.

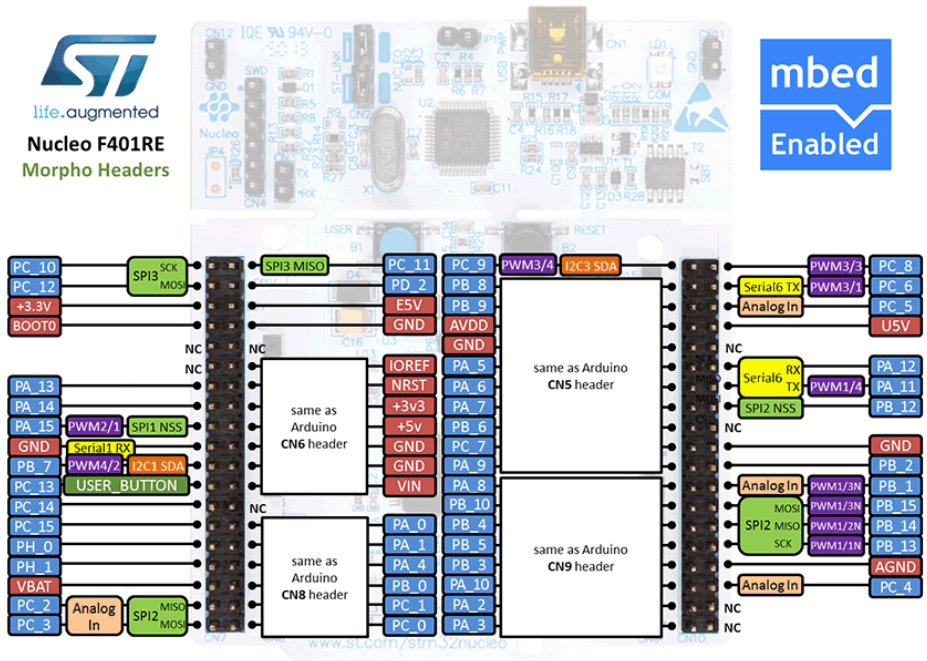
Example

Attach a DigitalOut to the LED1 pin on the Nucleo and turn it on:

```
DigitalOut led(LED1);
led = 1;
```

Nucleo Pin Names





Warning

You can only use the labels in blue and green!

Full size versions are available at

`https://mbed.org/platforms/ST-Nucleo-F401RE/`

aerodyne labs

Your Program

```
1  /* Includes */
2  #include "mbed.h"
3
4  /* Global Variable
5     Declarations     */
6  DigitalOut led(LED1);
7
8  /* Main Function */
9  int main() {
10   while(true) {
11     led = 1;   // Turn LED on
12     led = 0;   // Turn LED off
13   }
14 }
```

main.cpp

- Declare a global DigitalOut object
- Turn the output on and off in your main loop

Note

The LED won't seem to be flashing, but it actually is at about 42 MHz, much faster than your eye.

Waiting

There are three statements that can slow down execution:

```
void wait(float s);
void wait_ms(int ms);
void wait_us(int us);
```

All three will pause execution for the amount of time specified. Use these statements any time you need a controlled delay.

Notice

Wait and other block statements can have some unintended side effects. This will be demonstrated later.

Your Program

```
1  /* Includes */
2  #include "mbed.h"
3
4  /* Global Variable
5     Declarations     */
6  DigitalOut led(LED1);
7
8  /* Main Function */
9  int main() {
10   while(true) {
11     led = 1;      // Turn LED on
12     wait(0.2);    // Wait a bit
13     led = 0;      // Turn LED off
14     wait(0.8);    // Wait longer
15   }
16 }
```

main.cpp

- Add a wait statement after each write to your output
- You should now be able to see your LED flashing
- Try making your own patterns!

Outline

Variables

A variable is a container that has:

- **Type** What data it can hold
- **Identifier** The name you access it with
- **Value** The data it holds
- **Scope** Where you can access it from

You must declare a variable before you can use it:

```
type identifier = value;
```

The value is optional if you do not need a starting value.

Variable Types

Format	Type	Bits	Range
Integer	char	8	0 – 255
	signed char	8	-128 – 127
	unsigned short	16	0 – 65,535
	short	16	-32,768 – 32,767
	unsigned int	32	0 – 4,294,967,295
	int	32	2.1×10^9 – 2.1×10^9
	unsigned long long	64	0 – 1.8×10^{19}
	long long	64	-9×10^{18} – 9×10^{18}
Floating	float	32	-3.4×10^{38} – 3.4×10^{38}
	double	64	-1.7×10^{308} – 1.7×10^{308}

Note

The mbed natively works with 32 bit types, a huge advantage over 8 and 16 bit processors

Variable Scope

Variable scope is set by where a variable is declared:

```
int a = 0;

int main() {
  float b = 0.0f;
  while(true) {
    double c = 0.0;
  }
}
```

- a is outside any functions and can be seen anywhere in the file
- b can be seen anywhere inside main()
- c can only be seen inside the while loop

A simple rule of thumb is a variable can only be seen inside its enclosing braces.

Best Practice

Use the narrowest scope possible

Using Variables

Arithmetic Operators	
=	assignment
+	addition
-	subtraction
*	multiplication
/	division
%	modulus

Bitwise Operators	
&	bitwise and
|	bitwise or
^	bitwise xor
~	bitwise not
«	bitshift left
»	bitshift right

Compound Operators	
++	increment
--	decrement
+=	compound addition
-=	compound subtraction
*=	compound multiplication
/=	compound division
&=	compound bitwise and
|=	compound bitwise or

Digital Input

```
DigitalIn(PinName pin, PinMode mode);
```

DigitalIn operates nearly identically to DigitalOut, except you can optionally specify a pin mode:

- **PullNone** Standard mode, default
- **PullUp** Weak pull up resistor enabled
- **PullDown** Weak pull down resistor enabled

Access the value of the pin by treating it like any other variable.

Example

```
DigitalIn btn(USER_BUTTON);
int state = btn;
```

A New Program

Create a new program with:

- DigitalIn on the button
- DigitalOut on the LED
- Infinite main loop
- Button controlling the LED

Tip

Look at the pin diagrams to determine what PinName to use

A New Program

Create a new program with:

- DigitalIn on the button
- DigitalOut on the LED
- Infinite main loop
- Button controlling the LED

Tip

Look at the pin diagrams to determine what PinName to use

```
1  #include "mbed.h"
2
3  DigitalOut out(LED1);
4  DigitalIn in(USER_BUTTON);
5
6  int main() {
7    while(true) {
8      out = in;
9    }
10 }
```

TakingControl.cpp

Conditional Statements

Conditional statements are constructed from comparison and boolean operators:

Comparison Operators	
==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Boolean Operators	
&&	and
||	or
!	not

If Statements

An if statement executes different code based on the result of a conditional statement:

```
if(x > 0) {
  // Run if x is positive
} else if(x < 0) {
  // Run if x is negative
} else {
  // Run otherwise
}
```

The else if and else clauses are optional. For simple cases, so are the braces:

```
if(x < 0) x = 0;
```

Revisiting Our Last Program

Try to rewrite the last program using an if statement instead of simple assignment

Revisiting Our Last Program

Try to rewrite the last program using an if statement instead of simple assignment

```
1  #include "mbed.h"
2
3  DigitalOut out(LED1);
4  DigitalIn in(USER_BUTTON);
5
6  int main() {
7    while(true) {
8      if(in == 0) { // Button is pushed
9        out = 1;
10     } else {      // Button is not pushed
11       out = 0;
12     }
13   }
14 }
```

TakingControl.cpp

A Note on Logic Levels

A common practice is to use a pull-up resistor on buttons:

- Forces a known logic state
- **Inverts logic levels!**
- You can do this yourself using the PinMode parameter of DigitalIn objects

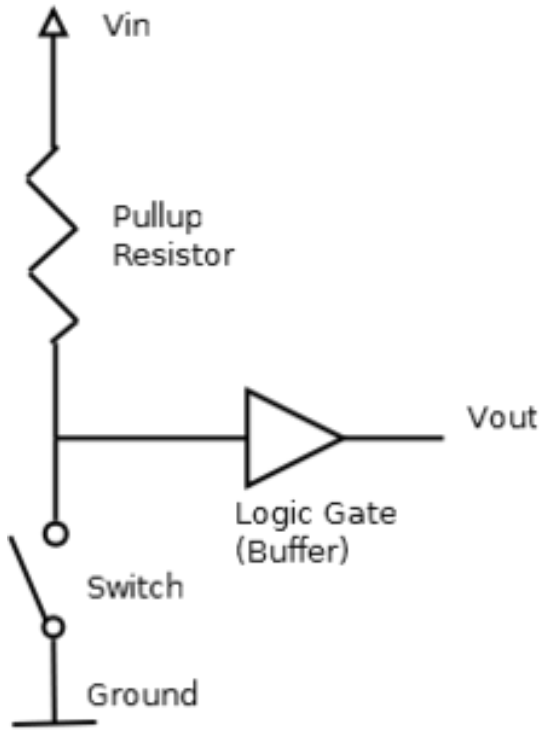


Image by: C4VC3 / CC-BY-SA

A Better Blinker

Lets try rewriting the blinky LED program to be a little more useful: use the button to change the delay time

A Better Blinker

Lets try rewriting the blinky LED program to be a little more useful: use the button to change the delay time

- Use a global variable to count button presses
- Remember the button has inverted logic levels
- Check the bounds of your count
- Try using math instead of discrete states

One Solution

```
1  #include "mbed.h"
2
3  DigitalOut out(LED1);
4  DigitalIn in(USER_BUTTON);
5  int count = 0;
6
7  int main() {
8    while(true) {
9      if(in == 0) count++;
10     if(count == 0) {
11       out = 1;
12       wait_ms(500);
13       out = 0;
14       wait_ms(500);
15     } else if(count == 1) {
16       out = 1;
17       wait_ms(250);
18       out = 0;
19       wait_ms(250);
20     } else {
21       count = 0;
22     }
23   }
24 }
```

BetterBlink.cpp

A More Elegant Solution

```
1  #include "mbed.h"
2
3  DigitalOut out(LED1);
4  DigitalIn in(USER_BUTTON);
5  int divisor = 1;
6
7  int main() {
8    while(true) {
9      if(in == 0) { // Button is pushed
10       divisor *= 2;
11       if(divisor >= 16) divisor = 1;
12     }
13     out = 1;  wait_ms(500 / divisor);
14     out = 0;  wait_ms(500 / divisor);
15   }
16 }
```

BetterBlink.cpp

Outline

Serial Ports

The Nucleo has one serial port that connects to the computer over USB:

```
Serial pc(USBTX, USBRX);
```

Every serial port has an associated baud rate (9600, 19200, 115200):

```
pc.baud(int baudrate);
```

There are methods to test if the port is ready to read or write:

```
pc.readable();
```

```
pc.writeable();
```

These methods return true if the port is ready.

Reading and Writing Characters

The simplest way to talk is a character at a time:

```
char in = pc.getc();
```

```
pc.putc(char out);
```

To write formatted output:

```
printf(string format (%[flags][width].[precision][key]), ...);
```

Flag	Description
-	Left justify
+	Force sign character
(space)	Leave a space for sign
0	Zero padding

Key	Output
d or i	Signed decimal integer
u	Unsigned decimal integer
f	Decimal floating point
e	Scientific notation
c	Character
s	String of characters

printf Examples

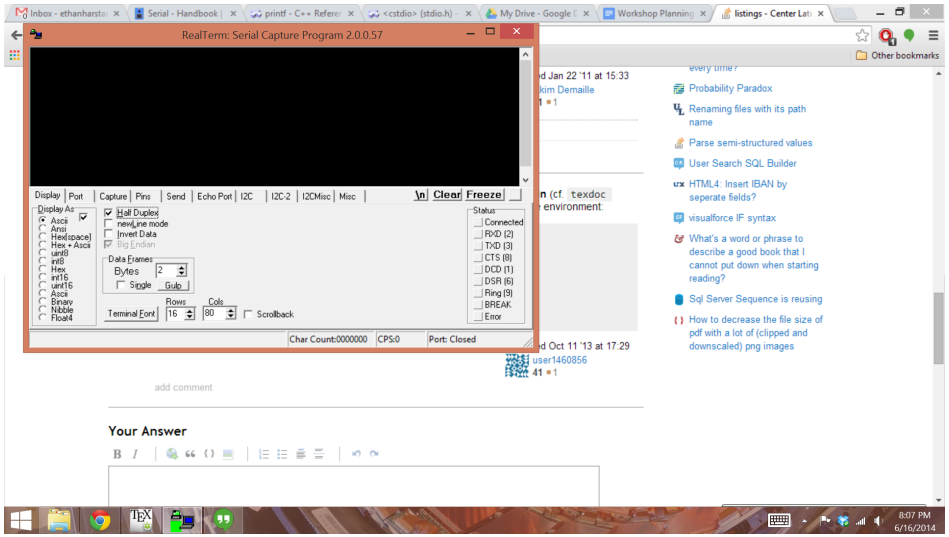
```
1 printf("Characters: %c %c \n", 'a', 65);
2 printf("Decimals: %d %ld\n", 1977, 650000L);
3 printf("Preceding with blanks: %10d \n", 1977);
4 printf("Preceding with zeros: %010d \n", 1977);
5 printf("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
6 printf("Width trick: %*d \n", 5, 10);
7 printf("%s \n", "A string");
```

Code

```
1 Characters: a A
2 Decimals: 1977 650000
3 Preceding with blanks:       1977
4 Preceding with zeros: 0000001977
5 floats: 3.14 +3e+000 3.141600E+000
6 Width trick:    10
7 A string
```

Output

Viewing the Serial Port



1. Open RealTerm
2. Enable "Half Duplex"
3. Click the "Port" tab
4. Baud: 9600 and the correct port
5. Click "Change"

Try running a simple sample program:

```
Serial pc(USBTX, USBRX);
int main() {
  pc.printf("Hello world!");
  while(true) {
    pc.putc(pc.getc() + 1);
  }
}
```

Switch/Case Statements

Switch statements can replace some types of if/else statements:

```
switch(variable) {
  case value1:
    // Code for value1
    break;
  case value2:
  case value3:
   // Code for value2 and 3
    break;
  default:
    // Code
}
```

Warning

A break statement is required between cases. Forgetting can cause many bugs!

Switch statements can only be used with integral type variables. You can group multiple cases together like 2 and 3 above.

A New Program

Create a new program with:

- A serial port connected to USB
- A DigitalOut connected to the LED
- A switch statement
  - One character turns the LED on
  - Another character turns the LED off

A New Program

Create a new program with:

- A serial port connected to USB
- A DigitalOut connected to the LED
- A switch statement
  - One character turns the LED on
  - Another character turns the LED off

```
1  #include "mbed.h"
2
3  DigitalOut led(LED1);
4  Serial pc(USBTX, USBRX);
5
6  int main() {
7    while(true) {
8      switch(pc.getc()) {
9        case 'H':
10         led = 1;
11         break;
12       case 'L':
13         led = 0;
14         break;
15     }
16   }
17 }
```

RemoteControl.cpp

Functions

Functions repackage code into reusable groups:

```
type name(parameter1, parameter2, ...) {statements}
```

type	The type of data returned by the function
name	The name the function can be called by
parameters	Variables used as inputs/outputs for the function
statements	Code statements that define what the function does

Parameters are essentially local variable declarations that are passed to the function, use as many (or as few) as needed:

```
int uselessFunction(int a, float b, double c) {...}
```

Using Functions

Call a function by using its name and your parameters in parenthesis:

```
int add(int x, int y) {
  return x + y;
}

int a = add(2, 3)

a = 5
```

Some functions do not require a return type:

```
void hello(char * name) {
  printf("Hello %s\n!", name);
}

hello("everyone");

>Hello everyone!
```

Revisiting Your Earlier Program

Lets rewrite the earlier remote control program to use a function to handle the input. This helps us stay organized.

Revisiting Your Earlier Program

Lets rewrite the earlier remote control program to use a function to handle the input. This helps us stay organized.

Only changed lines are shown to save space

```
6  void handler(char in) {
7    switch(in) {
8      case 'H':
9        led = 1;
10       break;
11     case 'L':
12       led = 0;
13       break;
14   }
15 }
16
17 int main() {
18   while(true) {
19     handler(pc.getc());
20   }
21 }
```

RemoteControl.cpp

Function Prototypes

Like variables, functions must be declared before they can be used. This is why handler was above main in the last example.

You can declare a function prototype separate from the implementation to avoid this:

```
void handler(char in);

int main() { ... }

void handler(char in) { ... }
```

Put the prototype with the rest of your global declarations and the function where ever it fits best.

For Loops

For loops execute a block of code a defined number of times:

```
for(initialization; condition; increment) {statements}
```

- Like a while loop, it iterates as long as the condition as true
- The initialization statements happens before the first iteration
- The increment statement happens after each iteration

Example

Iterate 10 times, printing the current count:

```
for(int i = 0; i < 10; i++) printf("%d\n", i);
```

Revisiting Your Earlier Program

Edit the remote control program to add a new feature:

- A function to print the abc's using a for loop
- A new character in your handler to call your new function

Revisiting Your Earlier Program

Edit the remote control program to add a new feature:

- A function to print the abc's using a for loop
- A new character in your handler to call your new function

```
15 void handler(char in) {
16   switch(in) {
17     case 'H':
18       led = 1;
19       break;
20     case 'L':
21       led = 0;
22       break;
23     case 'S':
24       sing();
25       break;
26   }
27 }
28
29 void sing() {
30   for(int i = 'a'; i <= 'z'; i++) {
31     pc.printf("%c", i);
32     wait(0.1);
33   }
34 }
```

Outline

Analog Input

AnalogIn operates almost identically to DigitalIn:

```
AnalogIn in(PinName pin);
```

You can access the value by treating it like a variable or with a function:

```
1 float value = in;
2 float value = in.read();
3 unsigned short value = in.read_u16();
```

1 and 2 are identical and return a float in the range [0.0 – 1.0].
3 returns an unsigned short in the range [0x0000 – 0xFFFF].

Using Potentiometers

A potentiometer functions as an adjustable voltage divider:

- One leg connects to 3.3v
- One leg connects to Gnd
- Wiper connects to an analog pin

The wiper varies between 0 and 3.3v as you turn the knob.

Analog Input Demo

Create a new program:

- AnalogIn connected to potentiometer
- Serial port connected to USB
- Print the analog value to serial
- Try using format specifiers with printf

Analog Input Demo

Create a new program:

- AnalogIn connected to potentiometer
- Serial port connected to USB
- Print the analog value to serial
- Try using format specifiers with printf

```
1  #include "mbed.h"
2
3  DigitalIn in(A0);
4  Serial pc(USBTX, USBRX);
5
6  int main() {
7    while(true) {
8      pc.printf("%1.4f\n", in);
9      wait(0.1);
10   }
11 }
```

Analog.cpp

PWM Output

Pulse Width Modulation encodes a signal by varying the width of a series of pulses.

Used for:

- Analog output (filtered)
- Servo position control
- Load control

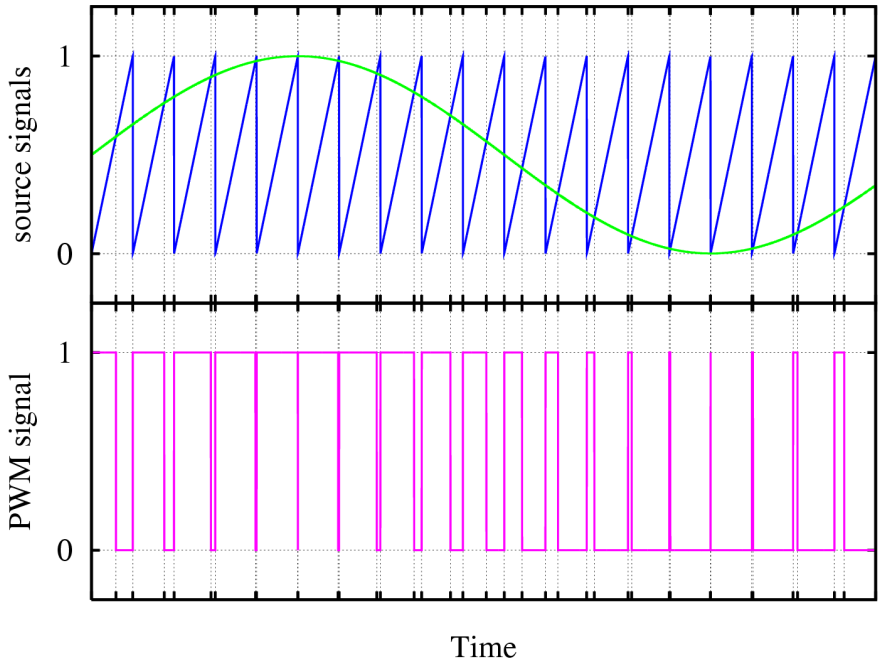


Image by: CyrilB / CC-BY-SA

PWM Output

You can use PwmOut on any purple pin marked PWM:

```
PwmOut out(PinName pin);
```

The period is controlled with:

```
out.period(float seconds);
out.period_ms(int ms);
out.period_us(int us);
```

and defaults to 0.02 seconds.

The pulse width is controlled with:

```
out = 0.5;
out.write(float value);
out.pulsewidth(float seconds);
out.pulsewidth_ms(int ms);
out.pulsewidth_us(int us);
```

The first two methods take a float in the range [0.0 – 1.0].

Use whichever methods suit your application best.

PWM Demo

Create a new program:

- PwmOut connected to the LED
- A loop changing the brightness of the LED

PWM Demo

Create a new program:

- PwmOut connected to the LED
- A loop changing the brightness of the LED
- Brightness is proportional to duty cycle
- Don't forget to slow your program down!

PWM Demo

Create a new program:

- PwmOut connected to the LED
- A loop changing the brightness of the LED
- Brightness is proportional to duty cycle
- Don't forget to slow your program down!
- Try playing with the period

```
1  #include "mbed.h"
2
3  PwmOut out(LED1);
4  float step = 0.01;
5
6  int main() {
7    while(true) {
8      out += step;
9      if(out >= 1.0)
10       step = -0.01;
11     if(out <= 0.0)
12       step = 0.01;
13     wait(0.01);
14   }
15 }
```

Pwm.cpp

PWM Notes

Be aware that the same PWM signal may be exposed on several pins, you can only use one at a time.

PWMx/y is the y-th channel of the x-th timer. All PWM's on a single timer must have the same frequency/period but can have different duty cycles.

PWMx/yN has the same period and duty cycle as PWMx/y but is inverted.

Classes

A class is an object that can contain both variables and functions. DigitalIn, DigitalOut and all the other objects used so far are examples of classes.

Using classes helps you:

- Stay organized - functions are attached to the data they use
- Reuse code - you can make as many copies as needed without extra code
- Control scope - you don't have to worry about name clashes

Class Structure

```
1  class Rectangle {
2    private:
3      int width, height;
4    public:
5      void setDimensions(int x, int y) {
6        width = x;
7        height = y;
8      }
9      int area() {
10       return x * y;
11     }
12 }
13
14 int main() {
15   Rectangle rect;
16   rect.set_values(3, 4);
17   printf("%d\n", rect.area());
18 }
```

Class Structure

The `class` keyword defines this block as a class.

The `public` and `private` keywords are access specifiers. Private members are only available from other members of the same class. Public members are available anywhere the object is.

Your class becomes a new data type in your program. You can create instances of it and use them like any other object.

Class members are accessed with dot notation: `name.member`

Class scope can be accessed with the :: scope operator

Class Scope

```
1  class Rectangle {
2    private:
3      int width, height;
4    public:
5      void setDimensions(int x, int y);
6      int area();
7  }
8
9  void Rectangle::setDimensions(int x, int y) {
10   width = x;
11   height = y;
12 }
13
14 int Rectangle::area() {
15   return x * y;
16 }
```

Header Files

Header files allow you to separate interface from implementation:

- Interface - Class and member definition in header file (.h)
- Implementation - Function bodies go in source file (.cpp)

Always use include guards in your header files:

```
1 #ifndef SAMPLE_H_
2 #define SAMPLE_H_
3 ...
4 #endif
```

sample.h

Include your headers with the `#include` directive:

```
1 #include "sample.h"
```

Header Files

```
1 class Rectangle {
2   private:
3     int width, height;
4   public:
5     void setDimensions(int x, int y);
6     int area();
7 }
```

Rectangle.h

```
1 #include "Rectangle.h"
2
3 void Rectangle::setDimensions(int x, int y) {
4   width = x;
5   height = y;
6 }
7
8 int Rectangle::area() {
9   return x * y;
10 }
```

Rectangle.cpp

Writing a Class

Lets convert this simple code into a class by following these steps:

1. Refactor into functions
2. Encapsulate into a class
3. Expose class with a header

```
1  #include "mbed.h"
2
3  AnalogIn in(A0);
4  PwmOut out(LED1);
5
6  int main() {
7    out.period(0.01);
8
9    while(true) {
10     out = in;
11   }
12 }
```

Refactoring

Pull out as much common code as you can into functions:

```
1  #include "mbed.h"
2
3  AnalogIn in(A0);
4  PwmOut out(LED1);
5
6  void update() {
7    out = in;
8  }
9
10 int main() {
11   out.period(0.01);
12
13   while(true) {
14     update();
15   }
16 }
```

Encapsulating

Encapsulate your code in a class by moving any relevant variables and functions into the class body:

- Variables should be private unless you have a good reason to make them public!
- Provide get/set functions to access needed private variables
- Most functions should be public
- Use private functions to avoid repeating yourself

Encapsulating

```
1  #include "mbed.h"
2
3  class Fader {
4  public:
5    Fader(PinName in, PinName
         out) : _in(in), _out(
         out) {
6      _out.period(0.01);
7    }
8
9    void update() {
10     _out = _in;
11   }
12
13 private:
14   AnalogIn _in;
15   PwmOut _out;
16 };
17
18 Fader fader(A0, LED1);
19
20 int main() {
21   while(true) {
22     fader.update();
23   }
24 }
```

Exposure

Separate the interface and implementation of your class into separate files:

- Interface (definition) into header (classname.h)
- Implementation (functions) into source (classname.cpp)
- Include the header file in your source file
- Don't forget to use the scope operator to put the functions into the class scope

Exposure

```
1  #ifndef FADER_H_
2  #define FADER_H_
3
4  #include "mbed.h"
5
6  class Fader {
7  public:
8    Fader(PinName in, PinName
         out);
9    void update();
10
11 private:
12   AnalogIn _in;
13   PwmOut _out;
14 };
15
16 #endif
```

Fader.h

```
1  #include "Fader.h"
2
3  Fader::Fader(PinName in,
       PinName out) : _in(in),
       _out(out) {
4    _out.period(0.01);
5  }
6
7  void Fader::update() {
8    _out = _in;
9  }
```

Fader.cpp

Using Your Class

- Create an instance of your class
- Invoke your class methods in your main function
- You can easily add more by creating more instances

```
1 #include "Fader.h"
2
3 Fader fader(A0, LED1);
4
5 int main() {
6   while(true) {
7     fader.update();
8   }
9 }
```

FaderDemo.cpp