Technical Workshop

Academic High Altitude Conference

Ethan Harstad Matthew Nelson

Stratospheric Ballooning Association

June 23-24, 2014

aerodyne labs

Outline

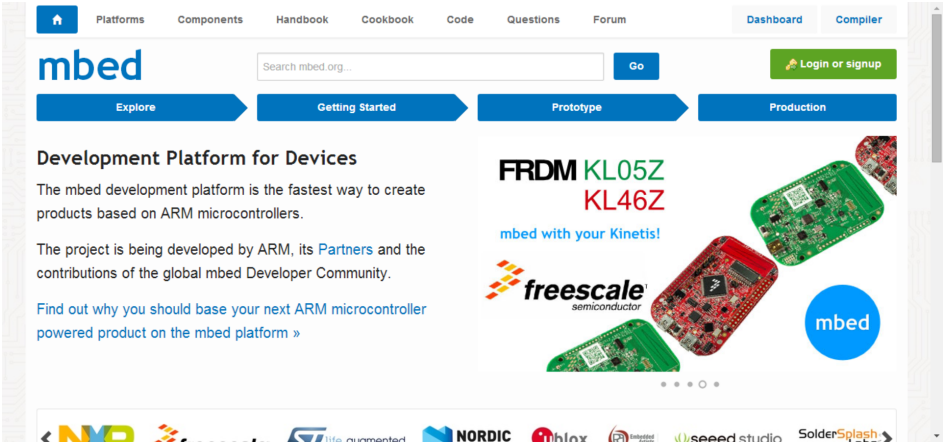
What is mbed?

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



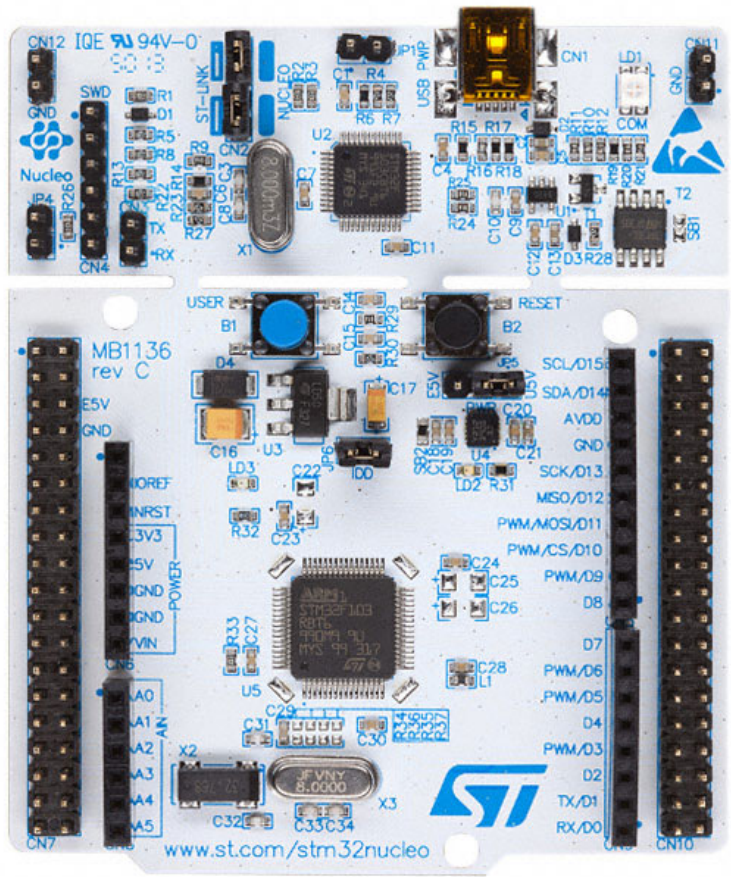
1. Navigate to `http://www.mbed.org`
2. Click the green "Login or signup" button
3. Click the "Signup" button
4. Follow the prompts
5. Confirm your e-mail address

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

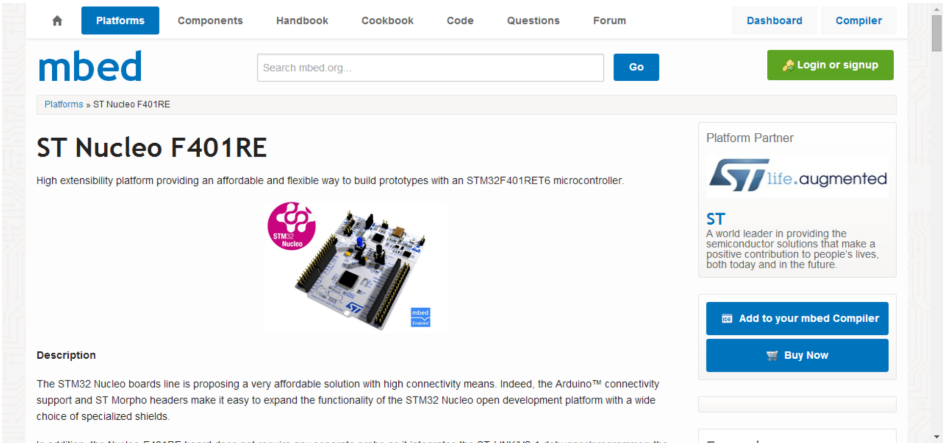
Nucleo Development Board

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"

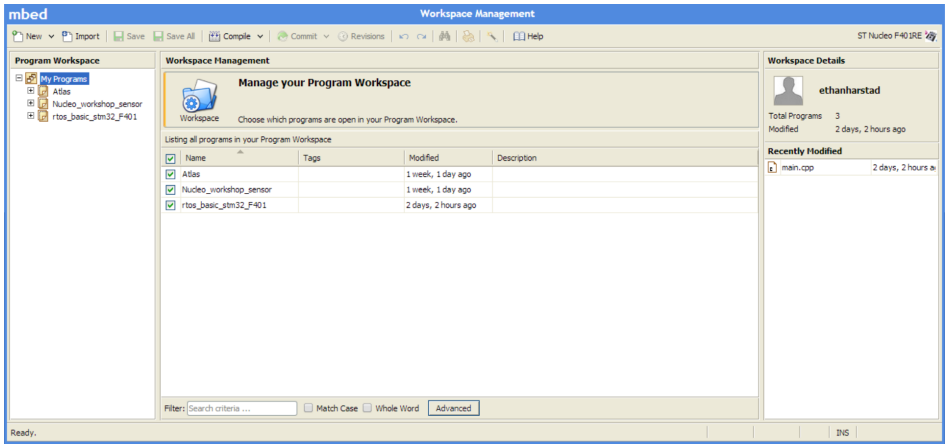
Note

You only need to do this once per account!

Install Drivers

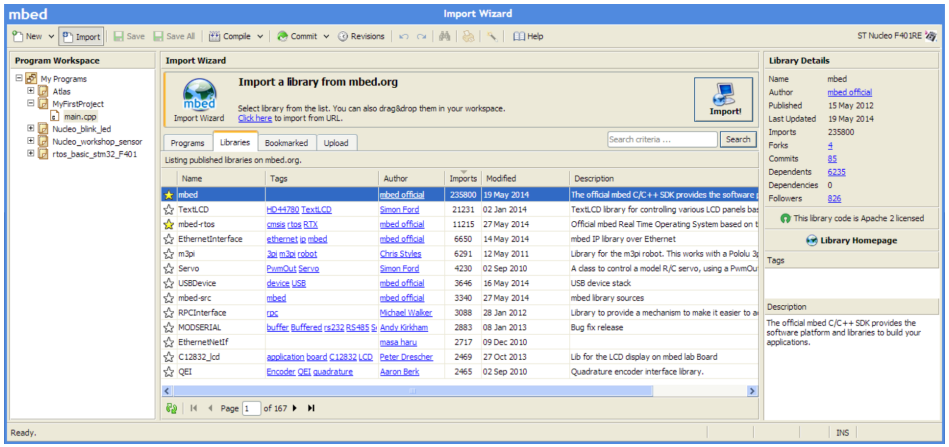
Outline

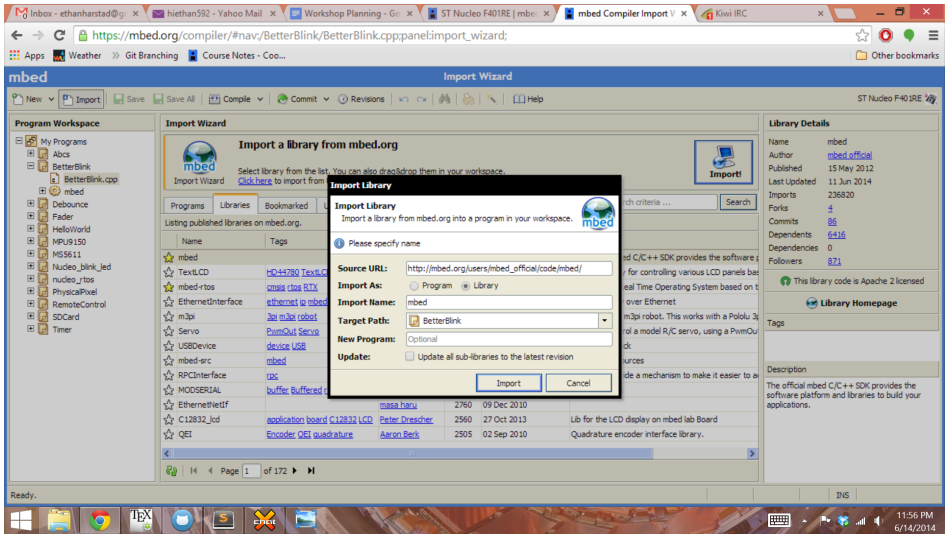
Creating a Program



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).

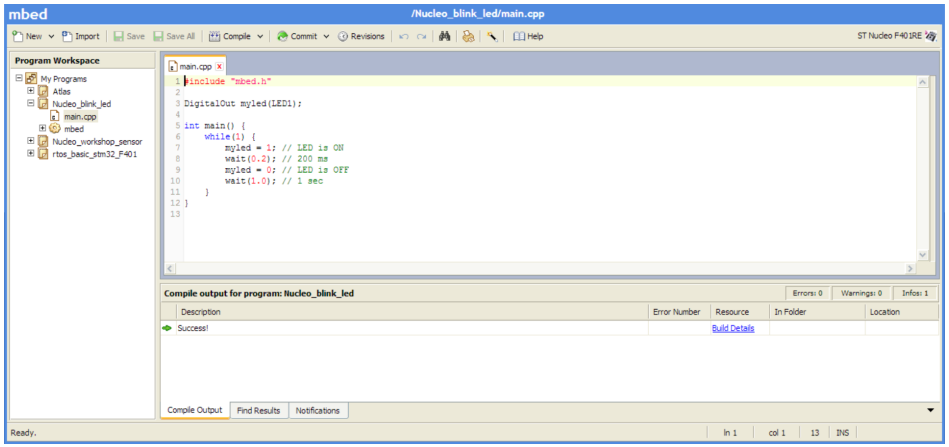
Program Structure

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

Listing 1: 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 }
```

Listing 2: main.cpp

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

Digital Output

DigitalOut constructor:

```
DigitalOut(PinName pin)
```

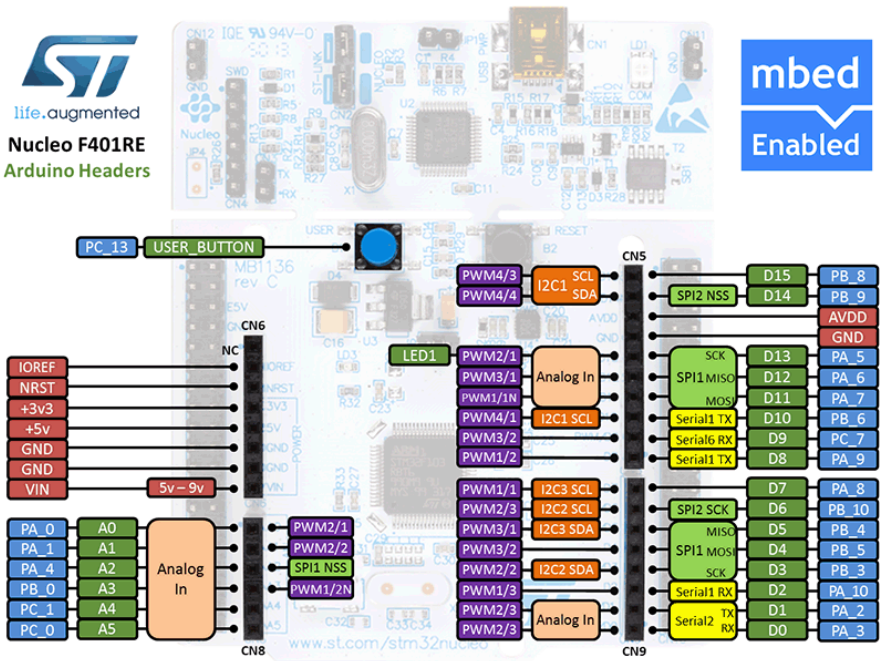
It creates an object attached to the given pin. Anytime you see PinName, use a name from the images on the next slide.
You can assign a value to the object using the equals sign. 1 turns the pin on while a 0 turns the pin off.

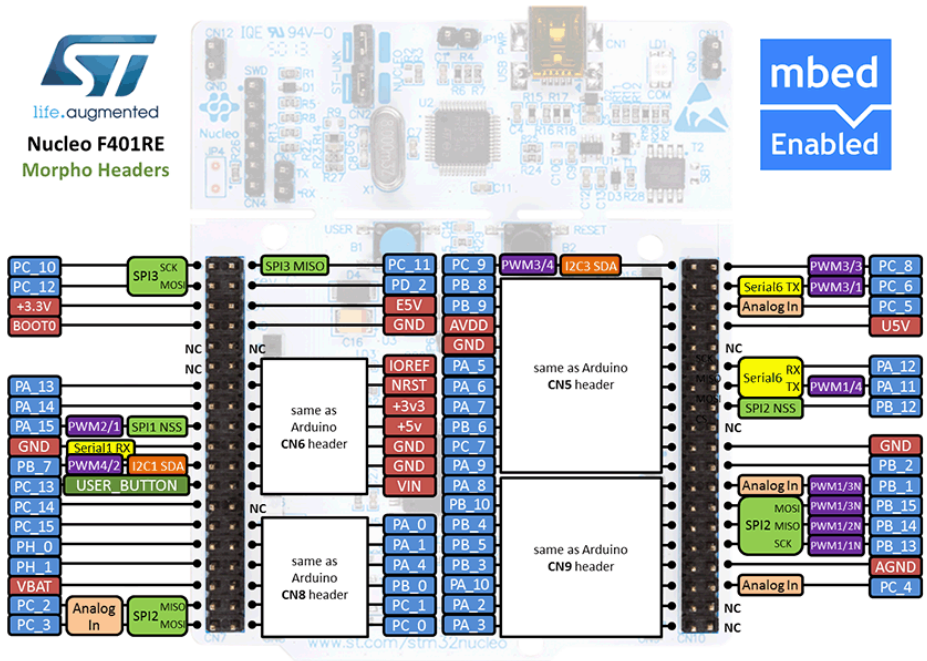
Example

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/`

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 }
```

Listing 3: 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 }
```

Listing 4: 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 }
```

Listing 6: 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 }
```

Listing 8: 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

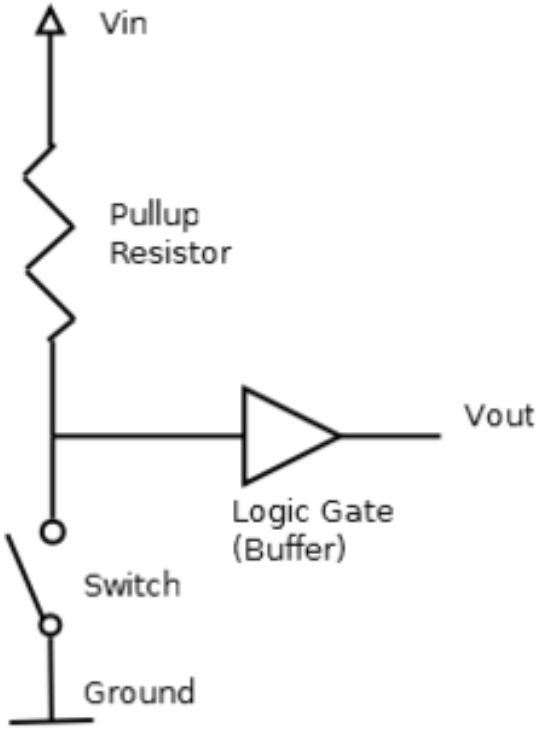


Image from Wikipedia

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

Listing 9: BetterBlink.cpp

```
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 }
```

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 }
```

Listing 10: BetterBlink.cpp

Outline

Outline

Outline

What is mbed?

Outline

What is mbed?