Materials Required for Lab 2

- Particle Argon
- LED Output Module
- Button Input Module
- Module Connector Cables
- USB Cable
Application: “Smart Light” System

- Control regular and smart lights
- Adjust your smart thermostat
- Activate home automation scenes
- Camera with privacy shutter and motion sensor
- Everyone can DJ with Sonos
- Dim lights or tap to turn lights on/off
- Lock / Unlock your doors with smart locks
- See who’s at the door with Ring
Application: “Smart Light” System

IoT Input Device → IoT Cloud → IoT Output Device

- Button Input Module
- LED Output Module
Particle Argon

WiFi Atenna

Mode Button
Status LED
Reset Button
USB Port
Blue LED on Pin D7
Power Connector for LiPo Battery
Analog Port A0
Digital Port D4
Digital Port D2
I2C Port 1
I2C Port 2
UART Port
Analog Port A2
Analog Port A4

ARGON
**Algorithm: Flowcharts**

(a) Flowchart for IoT Input Device

1. **Setup IoT Device**
2. **Read Button State**
3. **Button Pressed?**
   - **yes** → **Send "on" Msg**
   - **no** → **Wait 1 Second**
4. **Wait 1 Second**

(b) Flowchart for IoT Output Device

1. **Setup IoT Device**
2. **Wait for Msg**
3. **Msg is "on"?**
   - **yes** → **Turn LED On**
   - **no** → **Turn LED Off**
Computer Systems Stack

Application
Algorithm
Programming Language
Operating System
Compiler
Instruction Set Architecture
Microarchitecture
Register-Transfer Level
Gate Level
Circuits
Devices
Technology

Smart Light
Flowchart
C++
Particle OS
Particle Development Environment
ARM Machine Instructions
Ripple Carry Adder
NOT, AND, OR, XOR
Inverter
Resistors, LEDs, Transistors

Computer Engineering

CURIE Lab 1
CURIE Lab 2
Programming Language: C++

1 // function to add two integers
2 int add( int a, int b )
3 {
4   int sum;
5   sum = a + b;
6   return sum;
7 }

(a) 1 int c;
(b) 2 c = add( 2, 3 );

Figure 6: Example C++ Code Snippets
Programming Language: C++

```
1 int a;  // declaration
2 a = 2;  // assignment
3 int b = 3;  // initialization
4 int c;
5 c = a + b;

// function to add two integers
1 int add( int a, int b )
2 {
3     int sum;
4     sum = a + b;
5     return sum;
6 }
7

(a)

Figure 6: Example C++ Code Snippets

1 int button_state;
2 button_state = read_button_state( button_pin );
3
4 if ( button_state == 1 ) {
5     // send "on" msg
6 }
7 else {
8     // send "off" msg
9 }
10
11 // wait 1 second

(a) Sketch of IoT Input Device Program

1 void receive_msg( msg )
2 {
3     if ( msg == "on" ) {
4         // turn light on
5     }
6     else {
7         // turn light off
8     }
9 }

(b) Sketch of IoT Output Device Program

(b) Sketch of IoT Output Device Program

Figure 7: Sketch of C++ Programs for Smart Light
Computer Systems Stack

- Application
- Algorithm
- Programming Language
- Operating System
- Compiler
- Instruction Set Architecture
- Microarchitecture
- Register-Transfer Level
- Gate Level
- Circuits
- Devices
- Technology

- Smart Light
- Flowchart
- C++
- Particle OS
- Particle Development Environment
- ARM Machine Instructions
- Ripple Carry Adder
- NOT, AND, OR, XOR
- Inverter
- Resistors, LEDs, Transistors
// Global constants for pin assignments and global variables

int led_pin = D7;

int x = 2;
int y = 3;
int z = 0;

// Helper functions

int add( int a, int b )
{
    int sum;
    sum = a + b;
    return sum;
}

// The setup routine runs once when you press reset

void setup()
{
    // Configure led_pin as digital output
    pinMode( led_pin, OUTPUT );
}

// The loop routine runs over and over again

void loop()
{
    // Do the addition
    z = add( x, y );

    // Blink LED z times
    for ( int i = 0; i < z; i++ ) {
        digitalWrite( led_pin, HIGH ); // Turn on the LED
delay(500); // Wait 0.5 seconds
digitalWrite( led_pin, LOW ); // Turn off the LED
delay(500); // Wait 0.5 seconds
    }

    // Wait four seconds
delay(4000);
}
int button_pin = D4;
int button_state = -1;

void setup()
{
    pinMode( button_pin, INPUT );
}

void loop()
{
    button_state = digitalRead( button_pin );
    if ( button_state == 1 ) {
        Particle.publish( "button_state", "on" );
    }
    else {
        Particle.publish( "button_state", "off" );
    }
    delay(1000);
}

(a) IoT Input Device Program

(b) IoT Output Device Program

Figure 9: Complete C++ Programs for Smart Light

5. Compiler: Particle Development Environment

Now that we have refined our algorithm into a programming language with operating system support, we can use a compiler to translate the high-level program statements into the low-level instructions that the machine can actually execute. In this lab, we will be using the online Particle development environment to compile our programs. You can start the Particle development environment by going to this URL:

• https://build.particle.io

You should not create a new account. You should login with your group username and password.

Figure 10 labels the key icons on the left-hand side of the Particle development environment:

• Flash: Compiles and flashes the current code to the selected device
• Verify: Compiles without flashing the current code
• Save: Saves the current code
• Code: Shows a list of available programs
• Library: Explore libraries
• Help: Does not work for our Particle Argon
• Docs: Brings you to the Particle documentation site
• Devices: Shows a list of all devices
• Console: Brings you to the Particle console for monitoring the IoT cloud
• Settings: Log out

The most important icon is the flash (lightning) icon in the upper left-hand corner. Clicking this icon will cause the Particle IDE to compile your program into machine instructions and then to upload the resulting machine instructions to the Particle Argon for execution. Figure 10 illustrates what happens

Setup IoT Device

Read Button State

Button Pressed?

yes

Send "on" Msg

Wait 1 Second

no

Send "off" Msg

Wait for Msg

Msg is "on"?

yes

Turn LED On

no

no

Turn LED Off

yes

Figure 10: Complete C++ Programs for Smart Light

(a) Flowchart for IoT Input Device

(b) Flowchart for IoT Output Device
Now that we have refined our algorithm into a programming language with operating system support, we can use a compiler to translate the high-level program statements into the low-level instructions that the machine can actually execute. In this lab, we will be using the online Particle development environment to compile our programs. You can start the Particle development environment by going to this URL: https://build.particle.io. You should not create a new account. You should login with your group username and password.

Figure 10 labels the key icons on the left-hand side of the Particle development environment:

- **Flash**: Compiles and flashes the current code to the selected device
- **Verify**: Compiles without flashing the current code
- **Save**: Saves the current code
- **Code**: Shows a list of available programs
- **Library**: Explore libraries
- **Help**: Does not work for our Particle Argon
- **Docs**: Brings you to the Particle documentation site
- **Devices**: Shows a list of all devices
- **Console**: Brings you to the Particle console for monitoring the IoT cloud
- **Settings**: Log out

The most important icon is the flash (lightning) icon in the upper left-hand corner. Clicking this icon will cause the Particle IDE to compile your program into machine instructions and then to upload the resulting machine instructions to the Particle Argon for execution. Figure 10 illustrates what happens.

---

(a) IoT Input Device Program

```c
int button_pin = D4;

void setup()
{
  pinMode( button_pin, INPUT );
}

void loop()
{
  button_state = digitalRead( button_pin );
  if ( button_state == 1 ) {
    Particle.publish( "button_state", "on" );
  }
  else {
    Particle.publish( "button_state", "off" );
  }
  delay(1000);
}
```

(b) IoT Output Device Program

```c
int led_pin = D4;

void receive_msg( const char* event, const char* msg )
{
  if ( strcmp( msg, "on" ) == 0 ) {
    digitalWrite( led_pin, HIGH );
  }
  else {
    digitalWrite( led_pin, LOW );
  }
}

void setup()
{
  pinMode( led_pin, OUTPUT );
  Particle.subscribe( "button_state", receive_msg );
}

void loop()
{
  // empty
}
```

---

**Setup IoT Device**

1. **Wait for Msg**
2. **Msg is "on"?**
3. **Turn LED On**
4. **Turn LED Off**

---

**Flowchart for IoT Input Device**

1. Button Pressed?
2. Send "on" Msg
3. Send "off" Msg
4. Wait 1 Second

---

**Flowchart for IoT Output Device**

1. Wait for Msg
2. Msg is "on"?
3. Turn LED On
4. Turn LED Off
Computer Systems Stack

- Application
- Algorithm
- Programming Language
- Operating System
- Compiler
- Instruction Set Architecture
- Microarchitecture
- Register-Transfer Level
- Gate Level
- Circuits
- Devices
- Technology

- Smart Light
- Flowchart
- C++
- Particle OS
- Particle Development Environment
- ARM Machine Instructions
- Ripple Carry Adder
- NOT, AND, OR, XOR
- Inverter
- Resistors, LEDs, Transistors
Compiler: Particle Devel Environment

Particle Web IDE

Flash
Verify
Save
Code
Library
Help
Docs
Devices
Console
Settings

device OS target: Default (2.1.0)
On the device: 2.1.0

void setup()
{
  pinMode( button_pin, INPUT );
}

void loop()
{
  button_state =digitalRead( button_pin );
  if ( button_state == 1 )
  {
    Particle.publish( "button_state", "on" );
  }
  else
  {
    Particle.publish( "button_state", "off" );
  }
  delay(1000);
Compiler: Particle Devel Environment

- Always confirm your Particle Argon status LED is breathing cyan
- Always confirm that your Particle Argon is selected in the device list as indicated by the yellow star
- Always confirm that your Particle Argon is selected as indicated by the device name in the lower left-hand corner
- Always prefix the names your Particle Argon C++ programs with your first name (e.g., jane-blink-led)
Computer Systems Stack

Application
Algorithm
Programming Language
Operating System
Compiler
Instruction Set Architecture
Microarchitecture
Register-Transfer Level
Gate Level
Circuits
Devices
Technology

Smart Light
Flowchart
C++
Particle OS
Particle Development Environment
ARM Machine Instructions
Ripple Carry Adder
NOT, AND, OR, XOR
Inverter
Resistors, LEDs, Transistors

Computer Engineering
CURIE Lab 1
CURIE Lab 2
at the instruction set architecture, where hardware meets software and software meets hardware!

In Lab 2, we explored from applications to machine instructions. The computer systems stack is complete! In Lab 1, we explored technology to machine instructions, and you developed in Lab 1 to implement the add machine instruction which means our tour of the machine instructions: the first two machine instructions load values from main memory into registers, the third machine instruction adds these two values together, and the final machine instruction stores the sum back out to main memory. Note that the processor can use a ripple-carry adder similar to the one in Figure 6(b).

The compiler translates the high-level statements used in the programming language into very simple architecture with a processor to compute on data and two kinds of memory to store data: main memory which is fast but can only store a small amount of data, while is slow but can store a large amount of data. This simple architecture supports three kinds of machine instructions: arithmetic instructions that perform simple arithmetic on values stored in registers, load/store instructions that move values from memory into registers, and instructions that do the actual addition.

Figure 13: Simple Architecture

![Simple Architecture Diagram](image)

Figure 14: Machine Instructions for Line 5 in Figure 6(b)

```c
int add( int a, int b )
{
    int sum;
    sum = a + b;
    return sum;
}
```

```c
# load two values from main memory into two registers
ldr r2, [r7, #4]
ldr r3, [r7]

# do the actual addition
add r3, r3, r2

# store the sum from a register
# back into main memory
str r3, [r7, #12]
```

CURIE Academy, Summer 2021 Lab 2 Notes: Computer Engineering – Software Perspective
ISA: ARM Machine Language

Write your C++ code here ...

Select ARM gcc trunk (linux) here

Compiler will generate the corresponding machine instructions here ...

Try entering -O3 here
Lab 2 Overview

• Part 1.A Test Simple Addition Program
• Part 1.B Examine Machine Instructions
• Part 2.A Experiment with LED Output
• Part 2.B Experiment with Button Input
• Part 3.A Experiment with Particle Variables
• Part 3.B Experiment with Sending Particle Events
• Part 3.C Experiment with Receiving Particle Events
• Part 4.A Develop a “Smart Light” System
• Part 4.B Share Photo or Video of IoT System
• Experiment with IoT Geolocation System

Let’s write a simple blinking LED program
In lab 2 we explored from applications to machine instructions. One you developed in Lab 1 to implement the add machine instruction, which means our tour of the machine adds these two values together, and the final machine instruction stores the sum back out to main memory. Note that the processor can use a ripple-carry adder similar to the one-bit numbers together when both of its inputs are one.

As mentioned in the previous section, the compiler transforms the sequence of high-level statements into memory; and

The compiler translates the high-level statements used in the programming language into very simple instructions: the first two machine instructions load values from main memory into registers, one of which is a constant.

We meet in the middle of the stack. We meet in the middle of the stack. We meet in the middle of the stack. We meet in the middle of the stack.

OS & Compiler

PROG LANG

Methods

void loop()
    
    pinMode( led_pin, OUTPUT );

    digitalWrite( led_pin, HIGH ); // Turn on the LED

    delay(4000); // Wait 0.5 seconds

    // Global constants for pin assignments and global variables

    int add( int a, int b )
    {
        int sum;
        sum = a + b;
        return sum;
    }

    void setup()
    {
        // Code
        // Library
        // Help
        // Docs
        // Devices
        // Console
        // Settings
    }