Welcome to the next Lesson of Programming with the Intel Galileo. We will be learning how to implement task scheduling using an open source library on the Intel Galileo. The open-source community provides us with a Timer library that allows us to use functions provided instead of digging deep into low-level configurations. The Timer1 library has two functions that are essential in order for an Interrupt Service Routine on the Galileo to behave properly. These functions are:

- void Timer1.initialize(M) -- set the timer to tick every M microseconds
- void Timer1.attachInterrupt(function, period) – Calls a function at a specified interval in microseconds

Refer to the Timer1 Documentation for supplemental functions: [http://playground.arduino.cc/Code/Timer1](http://playground.arduino.cc/Code/Timer1)

First, we will need to download and add TimerOne to our Arduino Libraries. Refer to the link above for the download link and instructions.

**Example:**

The code below toggles Pins A0 – A2 every 1 second. Copy, Paste and Upload the sketch to the Intel Galileo. All LED’s should now blink on 1 sec and off 1 sec.
#include "TimerOne.h"

boolean blinkInterval = false;
const int ledPinOne = A0;
const int ledPinTwo = A1;
const int ledPinThree = A2;

void setup() {
  pinMode(ledPinOne, OUTPUT);
  pinMode(ledPinTwo, OUTPUT);
  pinMode(ledPinThree, OUTPUT);
  Timer1.initialize(10000); // the timer period is 100000 useconds, that is 0.1 sec
  Timer1.attachInterrupt(callback, 1000000); // the callback will be called on each 10th timer interrupt, i.e. every second
}

void callback() {
  if (blinkInterval) {
    digitalWrite(ledPinOne, HIGH);
    digitalWrite(ledPinTwo, HIGH);
    digitalWrite(ledPinThree, HIGH);
  } else {
    digitalWrite(ledPinOne, LOW);
    digitalWrite(ledPinTwo, LOW);
    digitalWrite(ledPinThree, LOW);
  }
  blinkInterval = !blinkInterval;
}

void loop() {

}

Once you have understood how to use the TimerOne library, proceed to the following exercises.

**Video Demonstration:** [https://youtu.be/DuQT3ZORLIY](https://youtu.be/DuQT3ZORLIY)

**Exercise 1**

1) Blink an LED at A0 at a rate of 500 ms.
2) Blink an LED at A1 at a rate of 1000 ms.
3) Blink an LED at A2 at a rate of 2500 ms.
Exercise 2

Using the task scheduler, implement a simple morse code to ASCII translator.

I recommend drawing a finite state machine for planning.

For those unfamiliar with morse code, more information can be found on wikipedia at http://en.wikipedia.org/wiki/Morse_code

A button on A0 is used to send morse code signals to your translator. Use a single syncSM task to decode signals and determine if they are dots or dashes.

One method for doing this might be to say that a dot is any '1' on A0 that lasts less than ~300 milliseconds, and a dash lasts greater than 300 milliseconds (choose the task period appropriately). A signal could be considered complete if no '1' on A0 is detected for 1 second. You can ignore the actual morse code timing specifications - use your own timing intervals as desired.

Once you have decoded a complete signal, print the array of morse code in the serial monitor.

If a complete morse signal is invalid, print "Invalid Morse Code".

Hint: There are 36 alphanumeric morse codes (A-Z, 0-9). When receiving the morse signal, create a char array of '1's(dots) and '2's(dashes). Then write a function that creates a unique hash value for the char array. Use the hash value to index into a lookup table that contains the corresponding ASCII code for the given code.

Demonstrate various morse codes and decode S-O-S.

Exercise 3

Expand upon Part 2 by converting each morse code array to its corresponding character. Demonstrate a few characters and decode S-O-S.

Hint:
- Create a 2D array to traverse through when converting from morse code array to character.
- Use ASCII values with the array index number for matching character.