This lesson will introduce concurrent synchSMs. To keep things simple, all concurrent tasks will have the same period and inter-task communication will be limited. Use the timer abstraction and setup from Lesson 3.

Again, you will need a couple of important lines of code.

- 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

As well as including the “TimerOne.h” Library.

**Exercises**

**Video Demonstration:** [https://youtu.be/eO3Wz2rCLMI](https://youtu.be/eO3Wz2rCLMI)

1. Connect LEDs to Pin 13, Pin 12, Pin 11, and Pin 10. In one state machine (Three LEDs), the following behavior occurs: set only Pin 13 to 1, then only Pin 12, then only Pin 11 in sequence for 1 second each. In a second state machine (Blinking LED), the following behavior occurs: set Pin 10 to 1 for 1 second, then 0 for 1 second.

2. Modify the above example so the three LEDs light for 300 ms, while PB3’s LED still blinks 1 second on and 1 second off.

**Generating Sound**

Sound can be generated by vibrating a membrane that creates sound waves in the air. A membrane vibrating at 261.62 Hz generates a "middle C" sound. A speaker has a membrane that moves when a voltage is applied (typically using a magnet that is moved by the electromagnetic wave of the changing electric current). Toggling a port from 0 to 1 at a frequency in the range of human hearing (around 20 Hz to 20,000 Hz) should generate sound if a speaker is connected to that port. The sound won’t be pleasant because it’s a square wave rather than a smoother sine wave, so it will sound more like a buzzer than a smooth tone.

For more info, see [Wikipedia: Audio frequency](https://en.wikipedia.org/wiki/Audio_frequency).

Arduino provides us with a library that generates frequency for tones. To use this library, simply call the functions:
3. Connect a speaker to pin 2 or 3. **(Pin 2 or 3 must be used because these pins are the only pins that can generate a square wave fast enough for playing tones on the Intel Galileo)** Generate a tone on the Intel Galileo with a frequency of 261.63.

   **Note:** Be sure to implement this code in the main loop. You will also need a delay of 1 second because running the tone library in an infinite loop causes major problems from the Galileo. Generally, these instructions are not necessary on Arduino boards but they are necessary on the Intel Galileo because not all libraries of Arduino are developed for the Galileo at this moment.

**Challenge Problems**

4. Extend the previous exercise to allow a user to adjust the sound frequency up or down using the Serial Monitor. The Serial Monitor allows us to send serial data via USB to the Galileo using the textbox.

The following functions will help us achieve our goal:

- Serial.available() // Returns true if the Send button is pressed and there is data in the buffer ready to be read.
- Serial.read() // Returns a char of the data from the Serial Monitor

**For Example:**

```java
if(Serial.available()){
    value = Serial.read();
    if(value == 'a')
        // Do something
}
```

For more information and documentation, refer to: [https://www.arduino.cc/en/reference/serial](https://www.arduino.cc/en/reference/serial)
Program the Galileo to play a default tone with a frequency of 261.63. If the user inputs a ‘1’, raise the frequency by 50. If the user inputs a ‘0’, decrease the frequency by 50.

5. Buttons are connected to A0 and A1. Pins 13 – 10 are connected to 4 LEDs. Pressing A0 increments a binary number displayed on the bank of LEDs (stopping at 9). Pressing A1 decrements the binary number (stopping at 0). If both buttons are depressed (even if not initially simultaneously), the display resets to 0. If a button is held, then the display continues to increment (or decrement) at a rate of once per second. However, if the button is held for 3 seconds, the incrementing/decrementing occurs once per 400 ms. Use synchronous state machines captured in C.