Serial Peripheral Interface (SPI) - LED Matrix

** This version of the lab will be used effective Fall 2020 for CS/EE 120B and will continue until on-campus instruction resumes.

Pre-lab

Wire your board

1) Learn SPI

One of SPI's advantages over USART is scalability with peripheral devices. USART requires 2 wires (Send/Receive) for each peripheral device. SPI requires 4 wires (Send/Receive/Clock/Select) for the first peripheral, and 1 additional wire per additional peripheral. Refer to SPI wiki for details -- specifically the "Typical SPI bus: master and three independent slaves" and the "master and cooperative slaves" figure.

The ATmega1284 datasheet details the SPI capabilities, search for "serial peripheral interface". The first hit will be the section "Serial Peripheral Interface - SPI".

2) Wire your boards -- Incrementally!

Refer to page 2 of the ATmega1284 datasheet for SPI pin locations. Your *Master* microcontroller should have a keypad and LED Matrix that do not interfere with the SPI pins. Your *Servant* microcontroller should have 8 LEDs that do not interfere with the SPI pins.

Wire the boards as you see fit -- ***you may wish to reference the demo videos to see potential board layouts.***

You will be using the Keypad and LED matrix for this lab, the .h folder on the GDrive contains 2 files: bit.h, keypad.h, that can help interface to the keypad. There is also a reference document for these files. The code to run the keypad *can work* ***independently*** *of the task scheduler*.

It is to your benefit to work incrementally. While you have worked with these components before don't be fooled -- wiring everything up at once and expecting it to work will most likely result in bugs that are difficult to isolate. Below are suggested incremental steps to get set up that do not require the timer/task scheduler to be tested.

- Wire the keypad on one port and 8 LEDs on another port, modify the pin define code in the keypad.h accordingly. Confirm the DDR/PORTSs are set properly -- half input, half output.
    - Output the result of `GetKeypadKey()` to the port with your 8 LEDs. If this was done correctly you should see the LEDs display the corresponding ASCII values for each button press.
- Wire up your LED Matrix, similar to Laboratory Exercise #12

- Add the task scheduler and create a simple task that periodically checks the keypad for new input and outputs an acknowledgment signal to the LED Matrix. If all works well, you now can feel confident you are ready to move on to the SPI lab.

Overall System Description:

The SPI based system described below will also be incrementally developed in the following lab exercises.

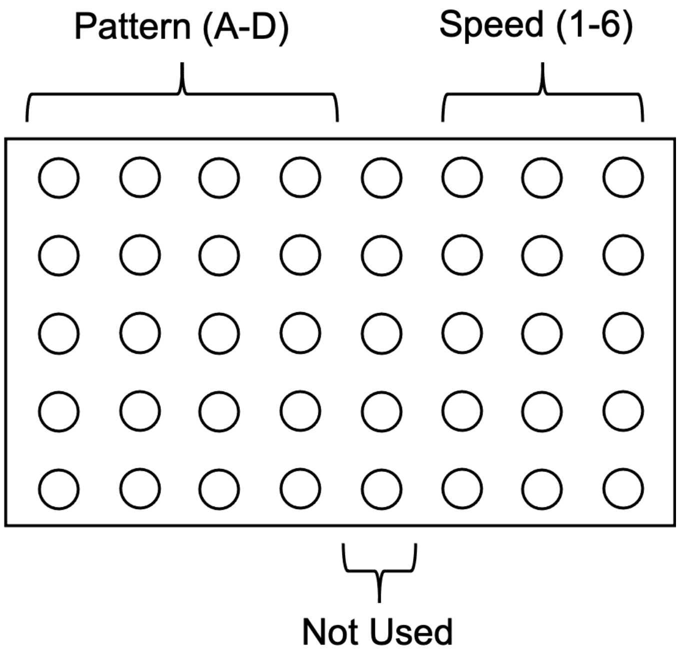
Create a festive lights display using 8 LEDs that has a user-selectable blinking pattern and speed. There should be 4 unique blinking patterns and 6 unique blinking speeds. The user selects the blinking pattern, blinking speed, and *Servant* microcontroller using the keypad. The LED Matrix displays the currently selected blinking pattern and speed.

The 4 blinking patterns:
1) 11110000 -> 00001111
2) 10101010 -> 01010101
3) 10000000 -> 01000000 -> 00100000 -> … -> 00000001 -> 00000010 -> 00000100 -> …
4) Create your own.

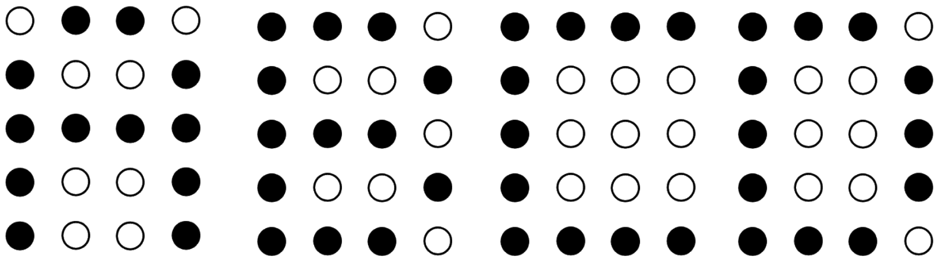
The 6 blinking speeds (a blinking speed is the time spent with a particular LED output):
1) 2 second
2) 1 second
3) 500 ms
4) 250 ms
5) 100 ms
6) 50ms

Hardcode 13 buttons on your keypad:
- A - D for blinking patterns
- 1 - 6 for blinking speeds

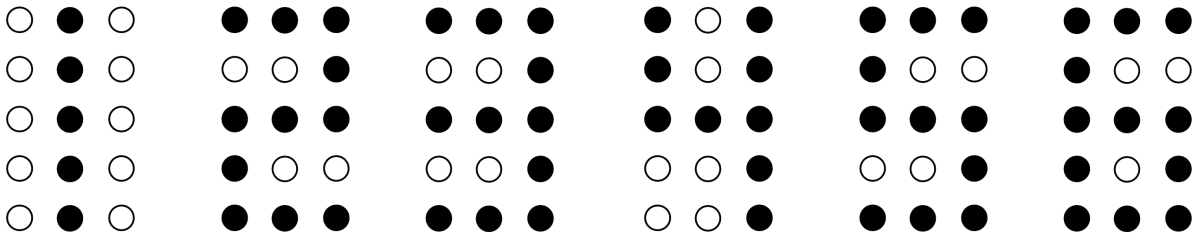
An image of the LED Matrix and how it will be used is shown below. Starting from the left, the Pattern (character A-D) is displayed using Columns 1-4. Column 5 is unused. And the speed (positive integer 1-6) is displayed on columns 6-8. The LED patterns for each character and positive integer are shown below.



Characters A, B, C, and D:



Positive Integers 1-6:



The following video demonstration illustrates the system functionality. It uses an LCD Screen, rather than an LED Matrix. It displays the Pattern (Ptrn) and the Speed (Spd) on the upper-left and upper-right hand side of the screen. It also displays a third value (uC) which will not be used in this laboratory exercise; you can ignore uC.

Video Demonstration: http://youtu.be/i5YEFsAuntQ

-- **Part 1** --

SPI communication

In the ATmega1284 datasheet, search for “serial peripheral interface”. The first hit will be the section “Serial Peripheral Interface - SPI”. Use the SPI section to fill out the code outline below.

```
// Master code
void SPI_MasterInit(void) {
        // Set DDRB to have MOSI, SCK, and SS as output and MISO as input
        // Set SPCR register to enable SPI, enable master, and use SCK frequency
        //   of fosc/16  (pg. 168)
        // Make sure global interrupts are enabled on SREG register (pg. 9)
}

void SPI_MasterTransmit(unsigned char cData) {
        // data in SPDR will be transmitted, e.g. SPDR = cData;
        // set SS low
        while(!(SPSR & (1<<SPIF))) { // wait for transmission to complete
                ;
        }
        // set SS high
}


// Servant code
void SPI_ServantInit(void) {
        // set DDRB to have MISO line as output and MOSI, SCK, and SS as input
        // set SPCR register to enable SPI and enable SPI interrupt (pg. 168)
        // make sure global interrupts are enabled on SREG register (pg. 9)
}

ISR(SPI_STC_vect) { // this is enabled in with the SPCR register’s “SPI
                    // Interrupt Enable”
        // SPDR contains the received data, e.g. unsigned char receivedData =
        // SPDR;
}
```

Important: When programming a microcontroller, disconnect any SPI lines connected to other microcontrollers. Once the microcontroller has been programmed, the SPI lines may be reconnected.

Master: Design a system that uses SPI to send an incremented 8-bit value to the *Servant* microcontroller every second.

```
SPI_MasterTransmit(1) -> SPI_MasterTransmit(2)-> SPI_MasterTransmit(3)...
```

Servant: Design a system that outputs any data received from the *Master* to a bank of 8 LEDs.

```
PORTB = receivedData;
```

--

--------------------------------------   Part 2   --------------------------------------

--

Master: Sending pattern and speed numbers to a Servant

Design a system where a pattern number and speed number are sent as one 8-bit value to the *Servant* whenever a new key is pressed on the keypad.

Criteria

- Use shared variables to hold the current pattern and speed numbers.
- Use another shared variable named "data" that holds the 8-bit value to be sent to the *Servant*.
- The upper 4 bits of "data" should hold the pattern number.
- The lower 4 bits of "data" should hold the speed number.
- "data" is updated and sent whenever a button is pressed on the keypad.

Examples of "data" values

- data = 0x11; // Pattern 1, Speed 1
- data = 0x34; // Pattern 3, Speed 4

Video Demonstration: http://youtu.be/RltGj0ijagg

Servant: Display one of four patterns

The *Servant* microcontroller displays one of four available patterns on a bank of 8 LEDs.

A shared unsigned char variable "receivedData" stores data received from a *Master* microcontroller. The upper 4 bits of "receivedData" represent the pattern number and the lower 4 bits of "receivedData" represent the speed number.

Design a system where the value read from the "receivedData" variable determines which pattern is displayed on the LED bank. For example, if "receivedData" is 0x24, then pattern number 2 should be displayed on the LED bank.

SynchSMs

- Write a "pattern" synchSM for each pattern described in the "Overall System Description" part of the lab. Each "pattern" synchSM writes its pattern to a unique shared variable.

    - "pattern" synchSM 1 writes to shared variable output1
    - "pattern" synchSM 2 writes to shared variable output2
    - "pattern" synchSM 3 writes to shared variable output3
    - "pattern" synchSM 4 writes to shared variable output4

- Write an "output" synchSM that writes a pattern to the bank of 8 LEDs. The pattern written is determined by the upper 4 bits of "receivedData".

    - if data is 0x24, then the "output" synchSM writes output2 to the LEDs
    - if data is 0x36, then the "output" synchSM writes output3 to the LEDs

Hints

- Try and design this system in anticipation of the functionality added in part 3 of this lab.
- *Servant's* functionality can be tested by hardcoding values to "receivedData".

Video Demonstration: http://youtu.be/Wq7AsbC8VyQ

-----------------------------------   Part 3   -----------------------------------
---

After the *Master* and *Servant* portions of part 3 are completed, the functionality of the system should match the description found in the Overall System Description section.

Master: Keypad input with LED Matrix Display

The LED Matrix display is updated to display the currently selected pattern number and speed.

Expand upon part 2 by designing a system where a button press on the keypad updates the LED Matrix display. The following video demonstration shows correct system functionality, but uses an LCD screen rather than the LED Matrix. Once again, the uC value is not used in this project.

Video Demonstration: http://youtu.be/UTF0ajiBk78

Servant: Adjust speed of blinking patterns

Expand upon part 2 of the lab by adjusting the rate at which the blinking patterns updates. The lower 4 bits of "receivedData" represent the desired speed number. Refer to the given speeds at the top of the lab.

Examples

- data = 0x24;
  - Pattern 2 is displayed
  - Pattern updates every 250 ms
- data = 0x41;
  - Pattern 4 is displayed
  - Pattern updates every 2 seconds

Hints

- One solution is to set the period of each "pattern" synchSM to the GCD of the speeds, and count up to a shared variable which governs how often the displayed pattern is updated. For example, if the GCD is 100, and the current speed is 500 ms, then each "pattern" synchSM will count up to 5 before updating its output variable. (500 / 100) = 5.

Video Demonstration: http://youtu.be/9jX_r4qtNLI

Submission

Each student must submit their source files (`.c`) and test files (`.gdb`) according to instructions in the lab submission guidelines.

```
$ tar -czvf [cslogin]_lab2.tgz turnin/
```

Don’t forget to commit and push to Github before you log out!

** Not used for online CS/EE 120B during COVID-19 **

Serial Peripheral Interface (SPI) - LCD Screen

Pre-lab

Wire your board

1) Division of labor

The following lab exercises should be completed in groups of two. Each lab exercise is divided into two sub-exercises: *Master* and *Servant*. One group member should complete the *Master* sub-exercises, while the other group member completes the *Servant* sub-exercises. Each group member should program their own microcontroller on their own breadboard.

2) Learn SPI

One of SPI's advantages over USART is scalability with peripheral devices. USART requires 2 wires (Send/Receive) for each peripheral device. SPI requires 4 wires (Send/Receive/Clock/Select) for the first peripheral, and 1 additional wire per additional peripheral. Refer to SPI wiki for details -- specifically the "Typical SPI bus: master and three independent slaves" and the "master and cooperative slaves" figure.

The ATmega1284 datasheet details the SPI capabilities, search for "serial peripheral interface". The first hit will be the section "Serial Peripheral Interface - SPI".

3) Wire your boards -- Incrementally!

Refer to page 2 of the ATmega1284 datasheet for SPI pin locations. Your *Master* microcontroller should have a keypad and LCD that do not interfere with the SPI pins. Your *Servant* microcontroller should have 8 LEDs that do not interfere with the SPI pins.

Wire the boards as you see fit -- ***you may wish to reference the demo videos to see potential board layouts.***

You will be using the Keypad and LCD for this lab, the .h folder on the GDrive contains 3 files: bit.h, keypad.h, lcd.h that can help interface to the keypad/lcd. There is also a reference document for these files. The code to run the keypad and lcd *can work* ***independently*** *of the task scheduler*.

It is to your benefit to work incrementally. While you have worked with these components before don't be fooled -- wiring everything up at once and expecting it to work will most likely result in bugs that are difficult to isolate. Below are suggested incremental steps to get set up that do not require the timer/task scheduler to be tested.

- Wire the keypad on one port and 8 LEDs on another port, modify the pin define code in the keypad.h accordingly. Confirm the DDR/PORTSs are set properly -- half input, half output.
    - Output the result of `GetKeypadKey()` to the port with your 8 LEDs. If this was done correctly you should see the LEDs display the corresponding ASCII values for each button press.
- Wire up your LCD, modify the pin define code in the lcd.h, and set the DDR/PORTs accordingly.
    - Call the `LCD_init()` function *once* before the while(1) loop, if this was done properly you should see a cursor flashing on your LCD.
- Add the task scheduler and create a simple task that periodically checks the keypad for new input and outputs this to the LCD. If all works well, you now can feel confident you are ready to move on to the SPI lab.

Overall System Description:

The SPI based system described below will also be incrementally developed in the following lab exercises.

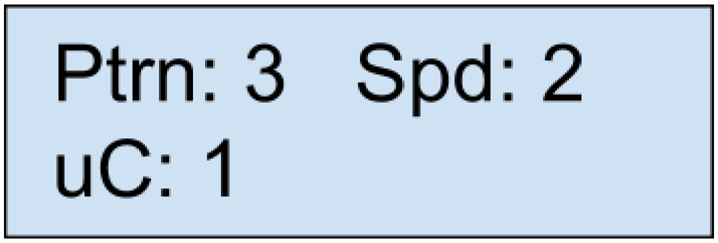
Create a festive lights display using 8 LEDs that has a user-selectable blinking pattern and speed. There should be 4 unique blinking patterns and 6 unique blinking speeds. The user selects the blinking pattern, blinking speed, and *Servant* microcontroller using the keypad. The LCD displays the currently selected blinking pattern, speed, and microcontroller.

The 4 blinking patterns:
1) 11110000 -> 00001111
2) 10101010 -> 01010101
3) 10000000 -> 01000000 -> 00100000 -> … -> 00000001 -> 00000010 -> 00000100 -> …
4) Create your own.

The 6 blinking speeds (a blinking speed is the time spent with a particular LED output):
1) 2 second
2) 1 second
3) 500 ms
4) 250 ms
5) 100 ms
6) 50ms

Hardcode 13 buttons on your keypad:
- A - D for blinking patterns
- 1 - 6 for blinking speeds
- 7 - 9 for *Servant* microcontroller

An image of how the LCD display should display the above information is given below. **NOTE:** Ptrn is assigned a number between 1 and 4, NOT a letter between 'A' and 'D'. Also, uC is assigned a number between 1 and 3, NOT a number between 7 and 9.



Video Demonstration: http://youtu.be/i5YEFsAuntQ

--
--------------------------------------- Part 1 ------------------------------------
--

SPI communication

In the ATmega1284 datasheet, search for “serial peripheral interface”. The first hit will be the section “Serial Peripheral Interface - SPI”. Use the SPI section to fill out the code outline below.

```
// Master code
void SPI_MasterInit(void) {
	// Set DDRB to have MOSI, SCK, and SS as output and MISO as input
	// Set SPCR register to enable SPI, enable master, and use SCK frequency
	//   of fosc/16  (pg. 168)
	// Make sure global interrupts are enabled on SREG register (pg. 9)
}

void SPI_MasterTransmit(unsigned char cData) {
	// data in SPDR will be transmitted, e.g. SPDR = cData;
	// set SS low
	while(!(SPSR & (1<<SPIF))) { // wait for transmission to complete
		;
	}
	// set SS high
}
```

```
// Servant code
void SPI_ServantInit(void) {
      // set DDRB to have MISO line as output and MOSI, SCK, and SS as input
      // set SPCR register to enable SPI and enable SPI interrupt (pg. 168)
      // make sure global interrupts are enabled on SREG register (pg. 9)
}

ISR(SPI_STC_vect) { // this is enabled in with the SPCR register's "SPI
                    // Interrupt Enable"
      // SPDR contains the received data, e.g. unsigned char receivedData =
      // SPDR;
}
```

Important: When programming a microcontroller, disconnect any SPI lines connected to other microcontrollers. Once the microcontroller has been programmed, the SPI lines may be reconnected.

Master: Design a system that uses SPI to send an incremented 8-bit value to the *Servant* microcontroller every second.

```
SPI_MasterTransmit(1) -> SPI_MasterTransmit(2)-> SPI_MasterTransmit(3)...
```

Servant: Design a system that outputs any data received from the *Master* to a bank of 8 LEDs.

```
PORTB = receivedData;
```

--

-------------------------------------- Part 2 --------------------------------------

--

Master: Sending pattern and speed numbers to a Servant

Design a system where a pattern number and speed number are sent as one 8-bit value to the *Servant* whenever a new key is pressed on the keypad.

Criteria

- Use shared variables to hold the current pattern and speed numbers.
- Use another shared variable named "data" that holds the 8-bit value to be sent to the *Servant*.
- The upper 4 bits of "data" should hold the pattern number.

- The lower 4 bits of "data" should hold the speed number.
- "data" is updated and sent whenever a button is pressed on the keypad.

Examples of "data" values

- data = 0x11; // Pattern 1, Speed 1
- data = 0x34; // Pattern 3, Speed 4

Video Demonstration: http://youtu.be/RltGj0ijagg

Servant: Display one of four patterns

The *Servant* microcontroller displays one of four available patterns on a bank of 8 LEDs.

A shared unsigned char variable "receivedData" stores data received from a *Master* microcontroller. The upper 4 bits of "receivedData" represent the pattern number and the lower 4 bits of "receivedData" represent the speed number.

Design a system where the value read from the "receivedData" variable determines which pattern is displayed on the LED bank. For example, if "receivedData" is 0x24, then pattern number 2 should be displayed on the LED bank.

SynchSMs

- Write a "pattern" synchSM for each pattern described in the "Overall System Description" part of the lab. Each "pattern" synchSM writes its pattern to a unique shared variable.

    - "pattern" synchSM 1 writes to shared variable output1
    - "pattern" synchSM 2 writes to shared variable output2
    - "pattern" synchSM 3 writes to shared variable output3
    - "pattern" synchSM 4 writes to shared variable output4

- Write an "output" synchSM that writes a pattern to the bank of 8 LEDs. The pattern written is determined by the upper 4 bits of "receivedData".

    - if data is 0x24, then the "output" synchSM writes output2 to the LEDs
    - if data is 0x36, then the "output" synchSM writes output3 to the LEDs

Hints

- Try and design this system in anticipation of the functionality added in part 3 of this lab.
- While waiting for the group member working with the *Master*, the *Servant's* functionality

can be tested by hardcoding values to "receivedData".

Video Demonstration: http://youtu.be/Wq7AsbC8VyQ

--
----------------------------------- Part 3 ------------------------------------
--

After the *Master* and *Servant* portions of part 3 are completed, the functionality of the system should match the description found in the Overall System Description section.

Master: Keypad input with LCD Display

The LCD display is constantly displaying the currently selected pattern number, speed, and microcontroller. Expand upon part 2 by designing a system where a button press on the keypad updates the LCD display.

Notes

- Refer to the "Overall System Description" section of the lab for keypad button mappings and LCD display format.
- Files to drive the LCD display and keypad can be found in Student Materials -> labs -> .h files. The files for the LCD display are "io.c" and "io.h". The file for the keypad is "keypad.h".
- In "io.c", make sure the variables: DATA_BUS, CONTROL, RS, and E are set depending on how the LCD display is connected to the microcontroller.
- In "keypad.h", make sure the variables: PORT and PIN are set depending on how the keypad is connected to the microcontroller.

Hints

- The LCD display uses ASCII values
    - `LCD_WriteData(2);          // Displays garbage on the LCD display`
    - `LCD_WriteData(2 + '0');    // Displays 2 on the LCD display`

Video Demonstration: http://youtu.be/UTF0ajiBk78

Servant: Adjust speed of blinking patterns

Expand upon part 2 of the lab by adjusting the rate at which the blinking patterns updates. The lower 4 bits of "receivedData" represent the desired speed number. Refer to the given speeds at the top of the lab.

Examples

- data = 0x24;
    - Pattern 2 is displayed
    - Pattern updates every 250 ms
- data = 0x41;
    - Pattern 4 is displayed
    - Pattern updates every 2 seconds

Hints

- One solution is to set the period of each "pattern" synchSM to the GCD of the speeds, and count up to a shared variable which governs how often the displayed pattern is updated. For example, if the GCD is 100, and the current speed is 500 ms, then each "pattern" synchSM will count up to 5 before updating its output variable. (500 / 100) = 5.

Video Demonstration: http://youtu.be/9jX_r4qtNLI

-----------------------------------   Part 4 (Challenge)   ----------------------------

---

Since no edits are required on the *Servant* microcontroller, both group members should contribute to the edits needed on the *Master* microcontroller.

Three Servants

Expand upon part 3 of the lab by allowing the *Master* microcontroller to communicate with three *Servant* microcontrollers.

When using SPI, each *Servant* microcontroller has an independent enable line (SS). In SPI_MasterTransmit, SS is set to 0 allowing the *Servant* microcontroller to receive data. SS remains 0 while the data is transmitted from the *Master* to the *Servant*. When the transmission is complete, SS is set to 1, preventing the *Servant* microcontroller from receiving anymore data.

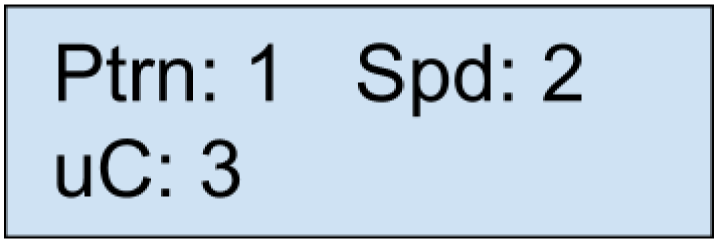
Select two free pins on the *Master* microcontroller to act as enable lines for two additional *Servant* microcontrollers. Add another parameter to SPI_MasterTransmit called "uC". This parameter is used to determine which of the three enable lines is manipulated so the selected microcontroller receives the data.

Example: uC is 2

- *Servant* 2's enable line is set to 0
- Data is transmitted between the *Master* and *Servant* 2
- *Servant* 2's enable line is set to 1

Make sure the data observed on the LCD display properly describes the behavior seen on the *Servant* microcontrollers. Keep in mind that when the selected *Servant* microcontroller changes, the other two microcontrollers continue displaying their own patterns at their own speeds.

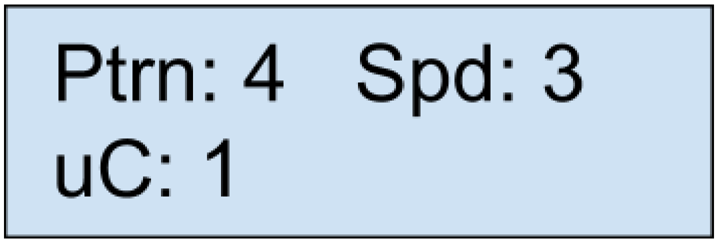
Example 1
- The *Master* microcontroller displays the following on the LCD display



```
Ptrn: 1   Spd: 2
uC: 3
```

- *Servant* 3 displays pattern 1 at speed 2. **Note:** *Servants* 1 and 2 retain their previous pattern number and speed.

Example 2
- The *Master* microcontroller displays the following on the LCD display



```
Ptrn: 4   Spd: 3
uC: 1
```

- *Servant* 1 displays pattern 4 at speed 3. **Note:** *Servants* 2 and 3 retain their previous pattern number and speed.

Video Demonstration: http://youtu.be/wZLCo-KR9wM

Submission

Each student must submit their source files (`.c`) and test files (`.gdb`) according to instructions in the lab submission guidelines.

```
$ tar -czvf [cslogin]_lab2.tgz turnin/
```

Don’t forget to commit and push to Github before you log out!