Lab 2: USART (Synchronized Warning Lights)

UCR CS122A

In the following exercises, we will be building a system that mimics the behavior of warning lights found on windmills in Palm Springs, CA. Follow the link below for a video of the synchronized windmill warning lights.

Synchronized Windmill Warning Lights

The warning lights are designed to blink synchronously. The reason for the blinking is to warn pilots flying at night that there are windmills to avoid. The reason for blinking synchronously is to avoid distracting nearby drivers with chaotic blinking.

We will be using the ATmega1284's onboard USART to communicate between microcontrollers in order to maintain a steady, synchronized blinking pattern.

IMPORTANT: All exercises should be completed in groups of two. Each group member should program their own microcontroller on their own breadboard.

USART

A Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is special hardware built into the ATmega1284 microcontroller. The USART allows for serial communication between the microcontroller and other external devices, including other microcontrollers. The ATmega1284 has two built-in USARTs (USART0 and USART1).

A USART on the ATmega1284 uses two pins to communicate data (RXDn and TXDn).

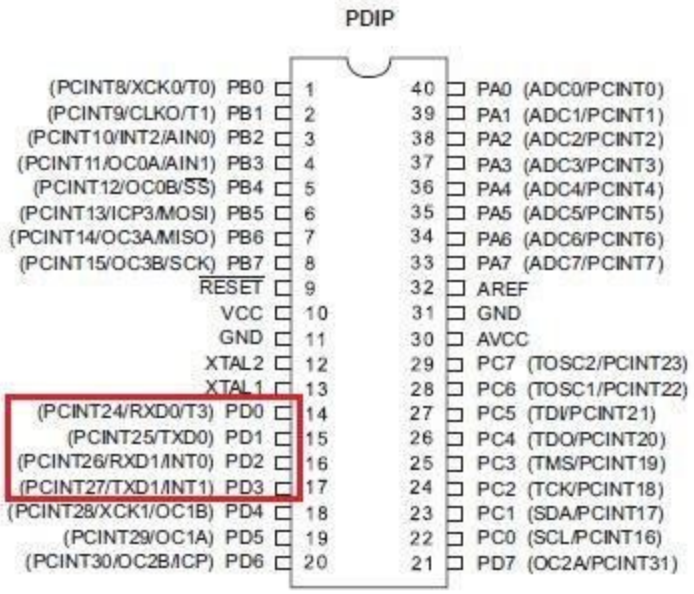
- RXDn receives serial data
- TXDn transmits serial data

RXD0 and TXD0 are the receive and transmit pins for USART0 (this is the same for the ATmega32)

- RXD0 is mapped to PD0
- TXD0 is mapped to PD1

RXD1 and TXD1 are the receive and transmit pins for USART1(the ATmega32 does not have a USART on these pins)

- RXD1 is mapped to PD2
- TXD1 is mapped to PD3



How the USART works?

The USART in the ATmega1284 sets flags to indicate its state.

- UDRE: When this flag is set, the USART is ready to send data
- TXC: When this flag is set, the USART has completed transmitting a byte of data
- RXC: When this flag is set, the USART has received data

The USART in the ATmega1284 uses an 8-bit register, named UDR, to transmit and receive data

- When data has been received by the USART, the data is stored in the Receive buffer. The USART cannot receive more data until the contents of the Receive buffer are emptied. When a program reads from UDR, the contents of the Receive buffer are transferred to UDR to be read.
- When data is ready to be transmitted by the USART, a program can write a byte of data to UDR. When data is written to UDR, the data is transferred to the Transmit buffer which triggers a transmission. The USART then transmits the

data serially and sets TXC once the transmission has completed.

USART functions: In the following functions, the return value is determined by checking the USART flags described above. Download and include usart_ATmega1284.h to access and use the functions below.

IMPORTANT: For all functions, pass in 0 or 1 for "usartNum" to specify which USART is referenced (USART0 or USART1).

```
// Initializes the desired USART.
// Call this function before the while loop in main.
void initUSART(unsigned char usartNum);

Example: initUSART(0); // initializes USART0
          initUSART(1); // initializes USART1


// Empties the UDR register of the desired USART, this will cause USART_HasReceived to return
false.
void USART_Flush(unsigned char usartNum);

Example: USART_Flush(0); // Empties the UDR of USART0


// Returns a non-zero number if the desired USART is ready to send data.
// Returns 0 if the desired USART is NOT ready to send data.
unsigned char USART_IsSendReady(unsigned char usartNum);

Example: if ( USART_IsSendReady(0) ) {
              //...send data...
          }


// Returns a non-zero number if the desired USART has finished sending data.
// Returns 0 if the desired USART is NOT finished sending data.
unsigned char USART_HasTransmitted(unsigned char usartNum);

Example: if ( USART_HasTransmitted(0) ) {
              //...do something...
          }


// Returns a non-zero number if the desired USART has received a byte of data.
// Returns 0 if the desired USART has NOT received a byte of data.
unsigned char USART_HasReceived(unsigned char usartNum);

Example: if ( USART_HasReceived(1) ) {
              //...receive data...
          }
```

```
// Writes a byte of data to the desired USARTs UDR register.
// The data is then sent serially over the TXD pin of the desired USART.
// Call this function after USART_IsSendReady returns 1.
void USART_Send(unsigned char data, unsigned char usartNum);

Example: USART_Send(0xFF, 1); // Sends 0xFF on USART1


// Returns the data received on RXD pin of the desired USART.
// Call this function after USART_HasReceived returns 1.
unsigned char USART_Receive(unsigned char usartNum)

Example: unsigned char temp;     // variable used to store received data
          temp = USART_Receive(1); // write data received by USART1 to temp
```

Pre-lab

No prelab is required for this lab.

Note: ***You will have to make some accommodations for the implementation of the exercises if you are using the ATmega32 as there is only 1 USART per micro-controller.***

Important: ***To prevent communication issues, ensure that you have common grounds between both micro-controllers and that their fuses are set the same so the clock cycles are identical (8MHz).***

Exercises

1. Leader/Follower Blinking LED. Two microcontrollers each have an LED connected to PA0. Both LEDs are synchronously toggled on/off in one-second intervals using USART.

Criteria

- One team member programs their microcontroller to be a *Leader*.
- The other team member programs their microcontroller to be a *Follower.*
- The *Leader* toggles the value of a local variable between 0 and 1, displays the value of the local variable on the LED connected to PA0, and then transmits a packet containing the value of the local variable to the *Follower*.
- The *Follower* receives a packet from the *Leader* and displays the value received on the LED connected to PA0.

- The LEDs on the *Leader* and *Follower* should appear to blink together.

Hint: Your follower's period will need to be faster than your leaders.

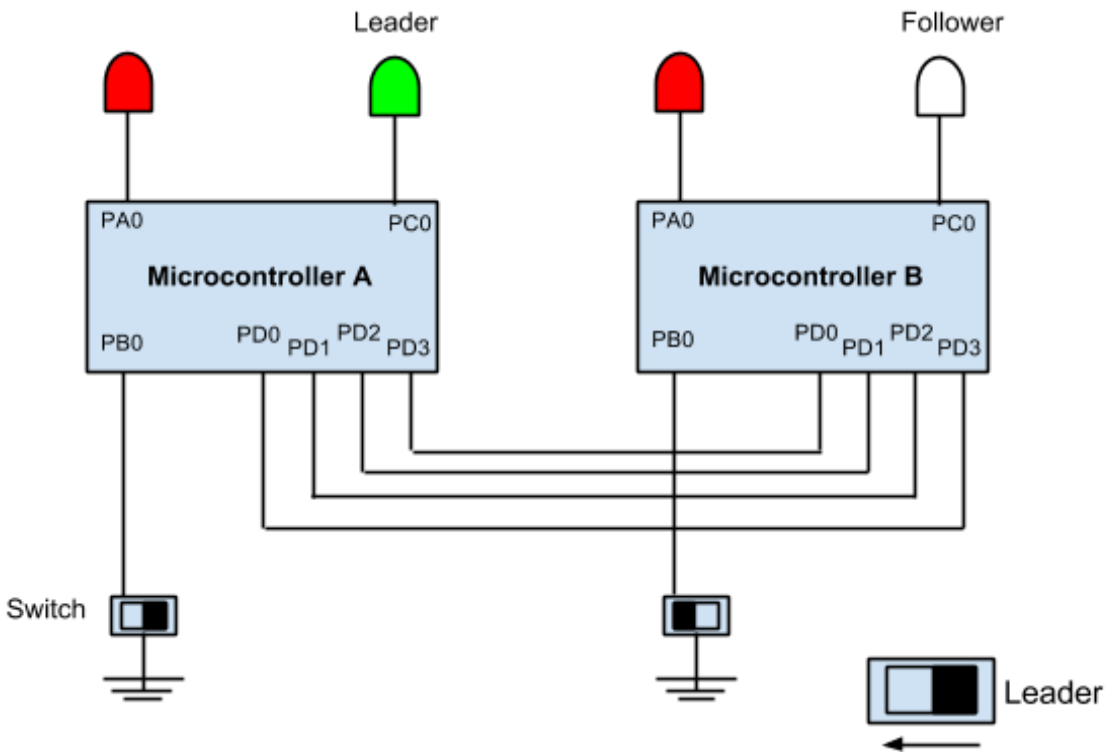
Video Demonstration: http://youtu.be/MooD6KqWa38

2. Switching Leader/Follower behavior

Expand upon Exercise 1 by adding a switch that determines the mode (*Leader* or *Follower*) of the microcontroller.

Criteria

- If a microcontroller is in *Leader mode,* the microcontroller behaves like the *Leader* described in part 1.
- If a microcontroller is in *Follower* mode, the microcontroller behaves like the *Follower* described in part 1.
- On each microcontroller, a switch is connected to PB0. The direction of the switch determines whether the microcontroller is in *Leader* mode or a *Follower* mode.
- On each microcontroller, an LED is connected to PC0. The LED is illuminated when the microcontroller is in *Leader* mode. The LED is turned off when the microcontroller is in *Follower* mode.
- Each microcontroller receives data on USART0 and sends data on USART1



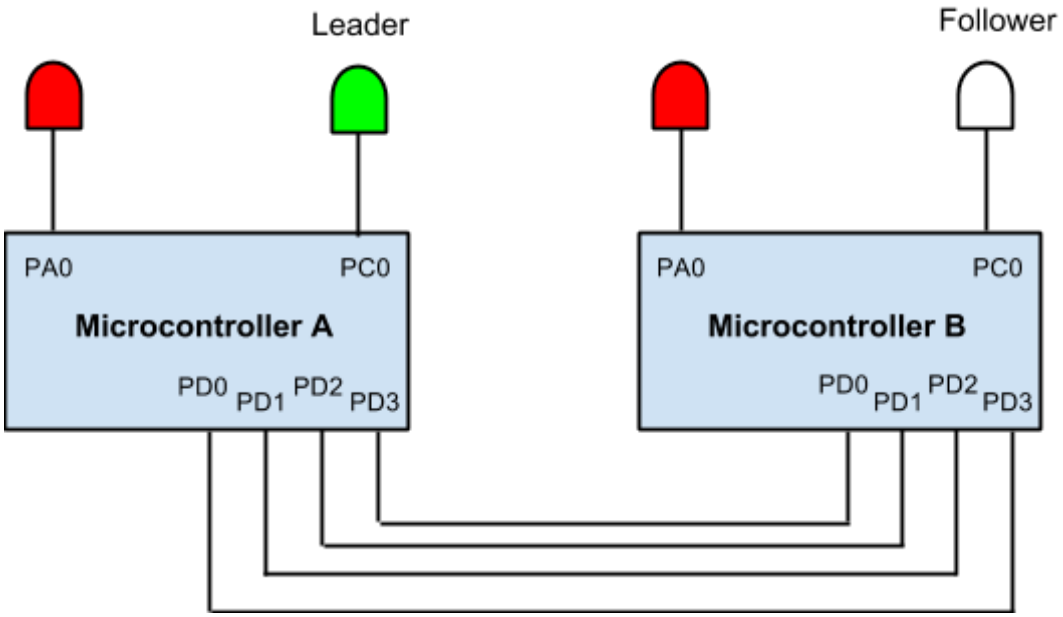
Video Demonstration: http://youtu.be/guwttK9W2Ts

3. Automatic Leader/Follower switching

Expand on exercise 2 by replacing "manual" mode switching with "automatic" mode switching.

Criteria:

- Both microcontrollers start in *Follower* mode.
- If a microcontroller is in *Follower* mode and does not receive a packet for 3 seconds, the microcontroller switches to *Leader* mode.
- If a microcontroller is in *Leader* mode and receives a packet at any time, the microcontroller switches to *Follower* mode.
- Each microcontroller receives data on USART0 and sends data on USART1



Video Demonstration: http://youtu.be/GwwUCJ8PK-M

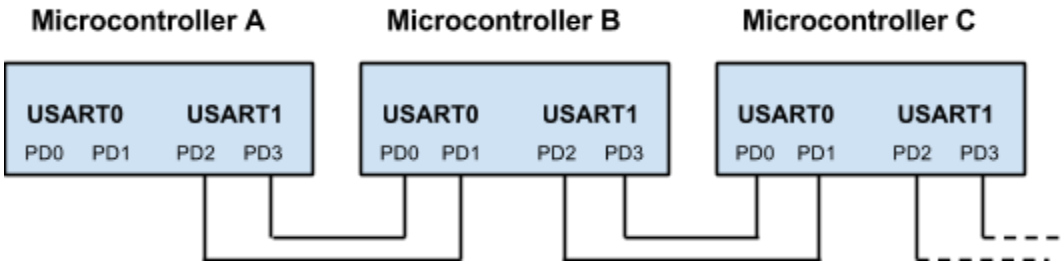
4. (Challenge) Synchronized Warning Lights

This exercise will be implemented at the end of the lab.

First, expand on exercise 3 by adding the following functionality to the *Follower* mode:

- After a microcontroller in *Follower* mode receives a packet on USART0 and displays that value on the LED connected to PA0, the microcontroller will then transmit that same value to another connected *Follower*.

All students who have successfully completed exercise 3, and who have also added the extra functionality described above, will have the option to connect their microcontrollers in series. Refer to the diagram below



If the expansion to exercise 3 was done correctly, then the microcontroller at the front of the series will become the *Leader* and the remaining microcontrollers will become *Followers*.

If a connection is severed in the series of microcontrollers, the microcontrollers still connected to the original *Leader* should continue uninterrupted. The microcontrollers that have been disconnected should remain unchanged for three seconds before the microcontroller at the front of the severed series assumes the *Leader* behavior.

Within each separate series of microcontrollers, there should be only one leader and all LEDs connected to PA0 should appear to blink together.

Video Demonstration: http://youtu.be/7We0k12SBQs

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!