Hot Potato, Hot Potato

CSE321 Project 3 Writeup

Bryan Moy, Carmen Tam, Dee Gao, Rafsan Chowdhury

PARTICIPATION

Overall, each group member worked on the project equally throughout the project through team meetings.

Group Member	Contribution
Bryan Moy	Building the circuit, foundation of master & slaves microcontroller code, logic implementation of software and hardware, connecting microcontrollers via I2C, timer functions, toggling LED to next player, writeup
Carmen Tam	Revision of master & slave code, implementation of left buttons, logic implementation of software and hardware, code to restrict button presses to only selected player, video demonstration, writeup
Dee Gao	Random generator (time and starting player), display (using Serial), reset function, revision of timer LED, logic implementation of software and hardware, filming/editing video, writeup
Rafsan Chowdhury	Building the circuit, foundation of master & slaves microcontroller code, logic implementation of software and hardware, connecting microcontrollers via I2C, timer functions, toggling LED to next player, writeup

PROJECT GOALS

The goal of our project was to simulate a classic hot potato game using embedded systems. We used four Adafruit Metro M0 Express microcontrollers to allow a maximum of three players to participate in the game, each coded using Arduino. One microcontroller is set aside as the master, which is responsible for sending data to the other three microcontrollers, the slaves. Each player has an LED and two buttons that allow them to “pass” the potato to the player before or after them. Each round will be indicated by a blue LED, the timer, and once the timer ends, the player who’s LED remains lit loses the round. Originally, we wanted more players for the game, but we had only purchased enough hardware in the beginning stages for a maximum of three players. Hence, we had to make a compromise. In addition, we hoped to add informative features to the game by displaying relevant game information on the computer screen after each round. We were able to successfully display the starting player for each round, as well as the player who has lost the round at the end. Overall, we accomplished our goal of creating a unique version of the hot potato game that does not require any physical tossing of an object, but nonetheless retains the core rules of the game.

IMPLEMENTATION TIMELINE

1. Purchasing the required hardware (microcontrollers & buttons)
2. Connecting the microcontrollers together via I2C
3. Learning Arduino basics for coding the program
4. Building the hardware for Hot Potato
5. First round of coding the game
6. Debugging and editing code (essentially, we had to experiment and figure out what works and what doesn't in Arduino & how the hardware reacts)
7. Final revision of the code
8. Record the video demonstration
9. Complete the writeup portion
10. Submit the project!

CHALLENGES

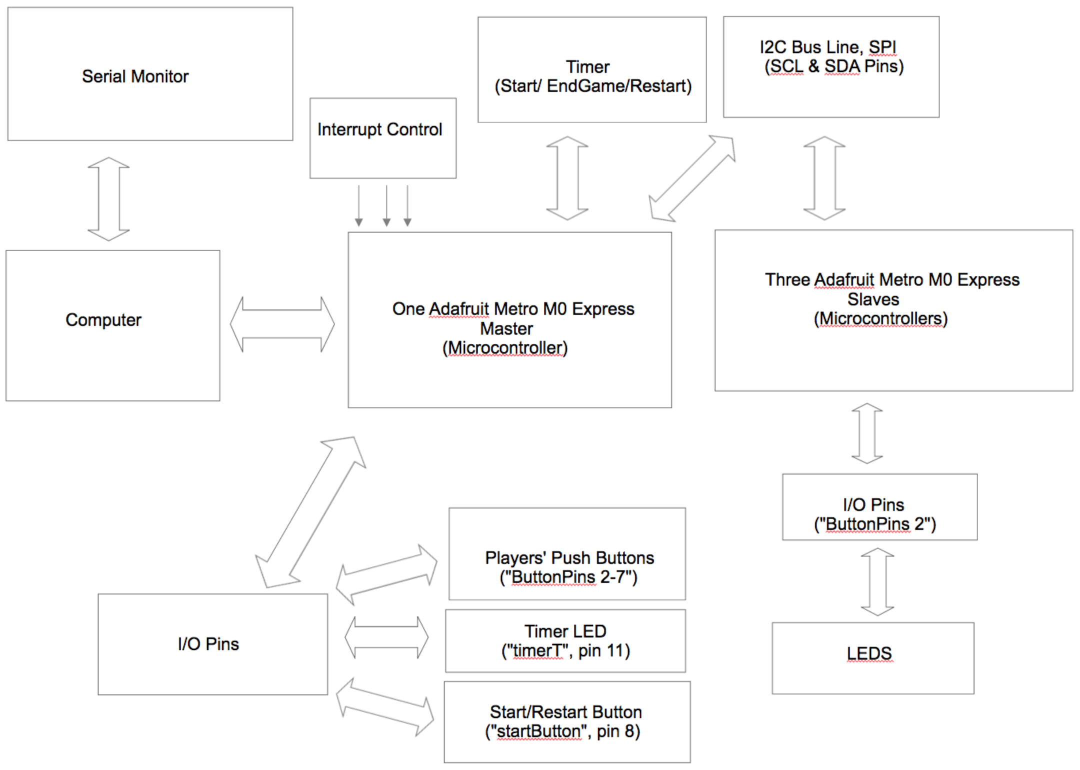
One of the challenges that we ran into during the project was creating a timer. Since we were using four microcontrollers, one being the master, we found difficulty when it came to sending information between the controllers while maintaining the rest of the functionality. Our problem was that we weren't able to run the timer and button presses simultaneously; it was either the timer or button function being ran first. So instead, we inputted a timer function into the first microcontroller and used that to send an indication of when the timer ended. We also thought of implementing the timer function into each controller that wasn't the master in order to obtain information on the LED that was lit when the timer stopped. However, having three of the controllers running the same code seemed very redundant when we could simply grab information from the master regarding which LED was still lit after receiving an indication of the timer ending from the first microcontroller.

Another issue that we came across was figuring out a way to enable the user to choose which player, left or right, received the 'hot potato.' Using buttons, we had to find a way to implement the system to allow the LED to pass through each of the three microcontrollers since each microcontroller represented a player which was equipped with two buttons and an individual LED. We solved this by setting each button to a designated microcontroller and indicating on the master code, which controller it would refer to, allowing the LED of that same controller to be lit. So for example, when pressing the left button on the first microcontroller, the signal from the master would be sent to the third controller (left of the first), to light up their LED.

However, from the solved problem above, we ran into a separate dilemma. Whenever the LED was lit for a player, indicating their turn, any player was able to choose which direction it would move to rather than only giving priority to the designated player. This means that at one point, there could be two LEDs on at the same time. To fix this, we had to add restrictions to our code allowing only the player with the LED on, to make a choice in which direction to choose. We also eliminated the choice that other players press by turning off the LED they chose. For example, originally, if the chosen player pressed right and another player chose right as well, two LEDs would light up at the same time (the chosen player's choice as well as the other player's choice). So after adding restrictions, only the chosen player's choice was lit up by disabling the other player's choice of direction, thus only allowing one player to use their buttons per turn.

BLOCK DIAGRAM WITH FUNCTIONAL DESCRIPTION

For this project, our hardware consisted of four Adafruit Metro M0 Express boards with one as the Master and the other three as Slaves. Between the computer and the master, the power and data were sent through a micro USB cable. Between the master and the slaves, data was sent through the I2C busline (SPI) or SCL & SDA ports of each board. The master contained the interrupt handler for the game, which worked by having the timer for the game. The interrupt handler worked by debouncing push button presses to determine the direction of passing "potato" (the player's LED is high) and looping to check the boolean state of the game (ongoing or end game). One feature the master controlled was the timer. It handled the timer LED by turning it high for the entirety of the round until the last few seconds, which then would blink the LED and finally turning it low on round end. In addition, as it calculates for the round ending through decrementing seconds, of the random time until zero, it would stop the debouncing for button presses, therefore rendering them inoperative. The flow of the LEDs for each player is demonstrated from the master board to the slaves' boards. This was through first the debouncing of the current player's push buttons (left/right) and then sending the "high" data to the respective player's LED to turn on depending on the direction of the press. Subsequently, the master would also send a "low" signal to the player's led that just passed the potato. The flow between the master and the computer processes the display of the game countdown and loser of the round. The flow of the slaves in the diagram demonstrates the board to the I/O pins which were connected to the LEDs. The slaves continuously checked for high/low data signals from the I2C line to determine whether their LED should stay on or off.


Serial Monitor
Interrupt Control
Timer (Start/ EndGame/Restart)
I2C Bus Line, SPI (SCL & SDA Pins)
Computer
One Adafruit Metro M0 Express Master (Microcontroller)
Three Adafruit Metro M0 Express Slaves (Microcontrollers)
I/O Pins ("ButtonPins 2")
Players' Push Buttons ("ButtonPins 2-7")
I/O Pins
Timer LED ("timerT", pin 11)
LEDS
Start/Restart Button ("startButton", pin 8)

INTERESTING FUNCTIONAL COMPONENT

One of the interesting components that we would like to highlight is the ability to restart the game, as well as the implementation of the gameplay data on the Serial. Although the concept may seem simple, it definitely required a lot of logical strategies for it to be implemented properly. In order to display the loser of the round in respect to the LED that remains on, we had to keep track of who's turn it was throughout the game. In fact, we had to use a rather non straightforward method of determining the player that would start the game because we had trouble turning on the LED for the first player. The method we used simulated a button press for the respective LED, which tells the hardware to turn on the LED that is connected to that button, even though in reality, no button has been pressed. In addition, as previously mentioned, we created finite states that were responsible for determining the current state of the game.

Essentially, once the round ends, the game time would be equal to zero and the state of the game would be set to false. However, once the start button is pressed again, all necessary variables will be reset through a separate reset function and the game would start again from the beginning. Ideally, the game can be reset any number of times unless the power has been unplugged.

VIDEO LINK

https://youtu.be/W6LIV8Iwul4