Acceleration Based RC Toy Car Controller

Kazumi Malhan, Justen Beffa
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
Emails: kmalhan@oakland.edu, jbbeffa@oakland.edu

***Abstract*—this document will outline the hardware and software design choices implemented in order to create a motion controlled wireless remote controlled toy car. Based on moving a remote forward, back, left or right a user will see these movements mirrored in a toy car.**

I. INTRODUCTION

Many remote controlled cars operate by pressing buttons on a controller. Instead of analog buttons, an acceleration based motion controller is desired. The motion controller will have parity of movement between it and the car it controls. Due to timing limitations the RC car itself was not designed here. This project takes an existing RC car and its original controller and modifies its controller in order to accept this new input and see the desired affect.

To achieve this type of control two microcontrollers are needed. One microcontroller is needed on the motion-controlled remote in order to read, digitize, and send out the data from the accelerometer. Another is used to process the data sent from the controller and generate the appropriate controlling code such that the car is driven in the desired directions. Here the processing of these signals should be fast enough such that the user feels they have real time control of the vehicle, but not so fast that unwanted movements are reflected in the vehicle.

II. METHODOLOGY

A. Hardware Circuit on HCS12 Board

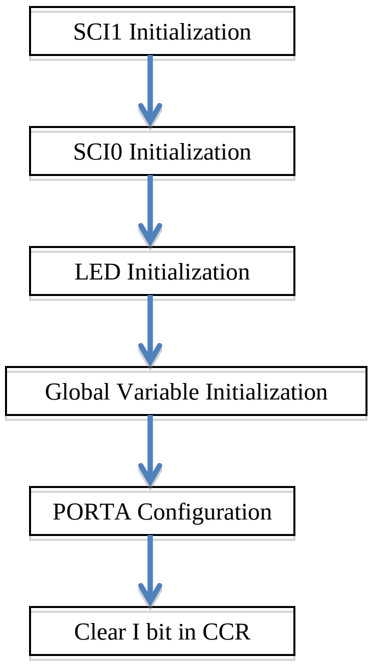
Bluetooth Serial Connection

The SparkFun Bluetooth Mate Silver module is used to replace the otherwise wired connection that would be required to communicate with the Arduino microcontroller. The communication to the Arduino is performed via SCI1. Rx pin (SP2) is connected to Tx pin of Bluetooth module while and Tx pin (PS3) is connected to Rx pin of Bluetooth module. The module was powered with 5V on-board supply and GND pin connected to ground.

CD4007 MOSFET Array

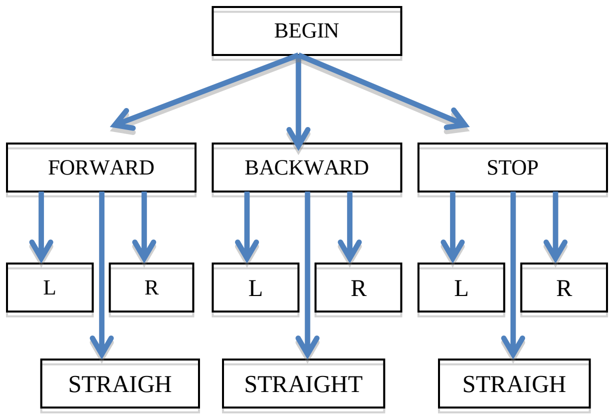
The original design of the RC Car Controller used four buttons to send a signal to a control chip(PT8A977B). Each button was assigned a directional pin on the control chip (forward, back, left, or right). Essentially by grounding the connection to control chip; the chip would then send the appropriate signal through an antennae to the car and move the car in the desired direction. In order to achieve the same affect of the button grounding the pin, N channel MOSFTs configured in Open Collector configuration provide the same affect. This design was chosen because the gate of the NMOS could be controlled by one of the HCS12's GPIO pins. Similar to the affect of pressing a button on the original controller, the HCS12 drives four pins that will either ground the connection to the control chip or keep the connection open. In this way when the HCS12 interprets the signal from the motion controller, the HSCS12 can then send the correct commands to the remote controlled car. Instead of using four individual NMOSs two CD4007 MOSFET Arrays are used here to provide the four open collector MOSFETs.

B. Software on HCS12



The flow chart above shows the setup sequence of HCS12 software. First, both Serial Communication Interface (SCI) 1 and SCI0 are initialized at rate of 9600 bps. The SCI1 is for communication between Arduino, and the SCI0 is for debugging and could be connected to a PC. PORTB is set as an output and then the lEDs are configured. PORTA is configured as output to control the signals to the MOSFET. Lastly, I bit in CCR register is cleared to allow interrupts. Coding for the HCS12 is done using the concept of modularity, SCI1, SCI0 and LED related codes are placed in separated header and implementation files, and then included to main file. Inside main for loop, CPU checks for performTask flag that triggers the processing of received data.

In order to continuously receive the accelerometer readings via SCI1 during operation, a receive serial interrupt has been setup. To enable this feature, SCI1CR2 RIE bit (bit 2) must be set to 1. One noticeable point is that SCI interface only has single interrupt flag for both transmitter and receiver. It is necessary to check which caused the flag to raise inside the SCI interrupt service routine. The vector number for SCI1 is 21. Inside the interrupt service routine (ISR), RDRF flag is checked at the beginning to confirm the source of flag, and copies the received byte into array of size 4. Since it is agreement between HCS12 and Arduino that one sample data consists of 4 bytes, array size is defined to be 4 via macro. When index of array reaches last element and doneTask flag is on, IRS copies the content of array into separate array for further data processing by other function that run in main. The doneFlag is an indication from data process function that process has completed. By having this feature, SCI1 continuously updates the accelerometer data, and provides the latest to only when data process function is ready to take. Lastly, IRS sets the performTask flag and exit the function.



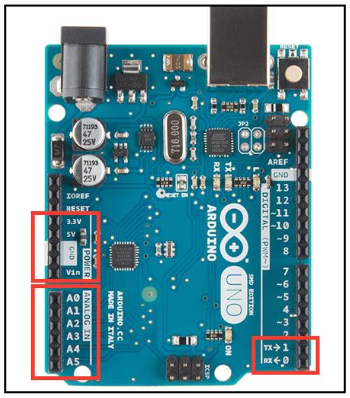
Process_Data is the function that performs data manipulation and outputs appropriate signal to PORTA to control the MOSFITS. First, received data are formatted back from four numbers of unsigned char to two numbers whose data type is unsigned int. Next, based on the threshold value calibrated during the testing phase, the function goes through flow control and decide what combination of output be sent to PORTA. The flow chart above shows the decision flow. First, forward, backward, stop is determined using x-axis acceleration. For each option, it checks the z-axis acceleration to determine left, right, or straight. Finally, the output combination defined in macro is outputted. Same output is also displayed on 8 LEDs to visualize the current command. The output pin assignment is defined as follow.

7	6	5	4	3	2	1	0
-	Right	-	Left	-	Backward	-	Forward

Software on HCS12 is designed to avoid using magic numbers. All parameters including threshold value, size of array, output commands, are defined in macro and macro are used inside the program. This allows programmer to fix the code with minimum effort in case of logic correction.

C. *Hardware Circuit on Arduino Board*

Arduino Uno



Due to its simpler architecture, fewer GPIO pins, and overall smaller size than the HS12, an Arduino Uno was chosen to be the microcontroller that converts the acceleration readings to a digital signal and then passes that signal via serial communication to an HCS12 microcontroller. The image below shows the image of an Arduino Uno, and highlighted are the pins that were used in this project.

Analogue to Digital Conversion

The Arduino Uno comes equipped with an ADC with six channels. For the purposes of reading acceleration on the x and z axis of the accelerometer, only the channels A0 and A4 are needed. The ADC will give an 10bit number as its output, so an integer is required to store its value

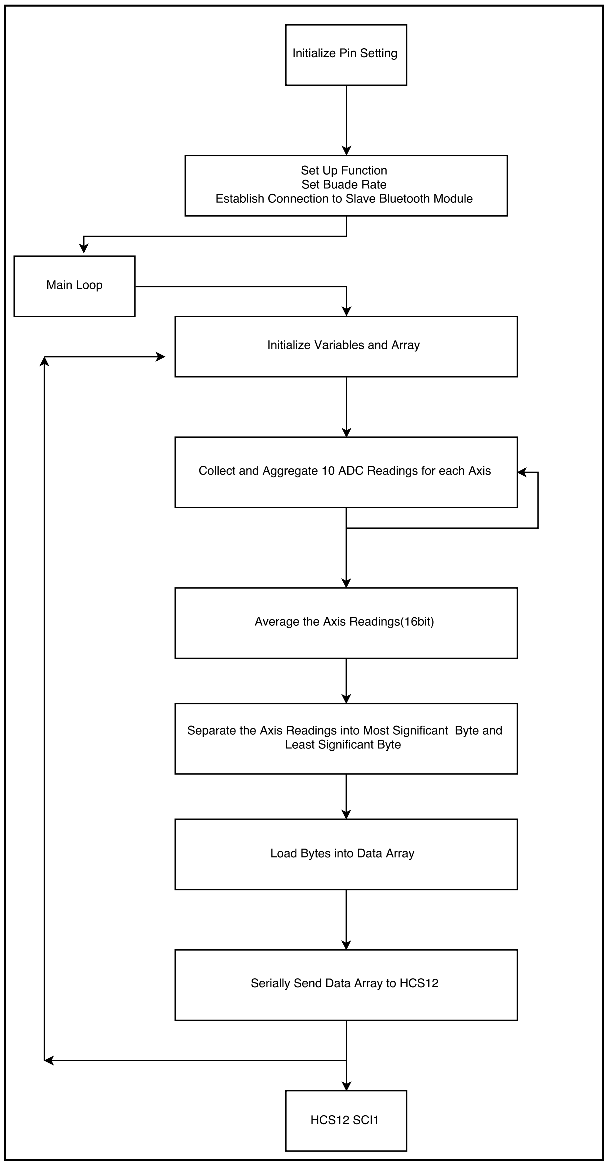
The ADXL335 is a three-axis accelerometer. Due to its placement on the board movement along its x-axis correlates to user movements for car forward or backwards commands. Movement along the z-axis correlates to user movements for car left or car right commands. The accelerometer outputs a continuous analogue voltage, which is related to the gravitational force it is experiencing on that axis. Reasonable movements that the motion controller could expect at each axis will produce readings between -1 g and 1 g. This roughly relates to 1.9 Volts and 1.3 Volts, or 380 and 300 digital reading. The schematic below shows how the ADXL335 was wired. VCC is the onboard 3.3 V supply, and the x channel is connected to ADC A0 and the z channel connected to ADC A4.

Bluetooth Serial Connection

Additionally a SparkFun Bluetooth Mate Silver module is used to replace the otherwise wired connection that would be required to communicate with the HCS12. This module was configured to communicate at 9600bps and was also set a the master of the two Bluetooth modules used. The VCC pin is connected to the Arduino's 5V power supply, its Rx pin connected to the board's Tx pin and vise versa.

D. *Software on Arduino*

The flow chart below shows the progression of the program that is running on the Arduino Uno. After the Analogue to


Initialize Pin Setting
Set Up Function
Set Buade Rate
Establish Connection to Slave Bluetooth Module
Main Loop
Initialize Variables and Array
Collect and Aggregate 10 ADC Readings for each Axis
Average the Axis Readings(16bit)
Separate the Axis Readings into Most Significant Byte and Least Significant Byte
Load Bytes into Data Array
Serially Send Data Array to HCS12
HCS12 SCI1

Digital pins are configured a setup function runs and establishes serial communication to occur at 9600bps. Additionally the set up functions sends a command to the Bluetooth module to connect to the slave module on the HCS12. After this the program enters the main loop, this loop will continue indefinitely.

Inside of the main loop temporary storage variable and a counter are initialized as is an array that will be used to transmit the x and z axis readings to the HCS12. The main loop then sums ten digital readings from each axis and takes the average of the readings. This process takes about 10ms. This approach was chosen to prevent erratic movements or noise from causing unwanted commands being sent to the car. Once the average is stored in an int variable, but the serial communication is set to send out and read a byte at a time. To accommodate this the int average from each axis is split into its most and least significant byte. These bytes are stored in a data array and the array is then sent to HCS12. Then the process of reading, averaging, storing and sending the accelerometer readings begins again.

III. HARDWARE SCHEMATICS

See Appendix A for HCS12 sensors schematics.
See Appendix B for Arduino sensors schematics.
See Appendix C for wiring on toy controller.

IV. EXPERIMENTAL SETUP

The incremental testing approach is a method to test each component as they are being developed. For this project, this method is critical as the project contains two microcontroller with different types of sensors.

Phase 1 development consists of reading an accelerometer using the ADC on Arduino. The serial monitor on Arduino was used to read the values from an accelerometer, and converted to gravity (g) reading to confirm the functionality of the accelerometer and code. Also, threshold values were defined in the process. Phase 2 consists of HCS12 side of code development. Testing HCS12 code was done by receiving fixed combination of numbers from Arduino board using SCI1 while displaying processed information to terminal on PC using SCI0 for debugging. At this stage, communication between Arduino and HCS12 was done using wired connection.

Once functionality of each microcontroller was tested, system level testing (Phase 3) was performed using wired connection. In order to debug, output command is displayed on LEDs using PORTB, and received valued are checked on PC via SCI0. Phase 4 was paring of two Bluetooth modules. The configuration of master module and slave modules were performed on Arduino microcontroller via serial and software serial communication. Final stage of testing (Phase

5) was to combine phase 3 and 4. Similar debugging method used in phase 3 was utilized. At the end, incremental testing approach paid off and system performed as expected.

V. Challenges

Bluetooth Serial Connection

One challenge that was faced was achieving the wireless serial connection between two modules. In class examples were given on how to achieve a connection between one module and a PC. Originally the components that were purchased could only operate in slave mode, meaning a connection could not be established between the two modules. After the SparkFun Bluetooth Mate Silver modules were purchased the major challenge was in accessing the modules AT commands and configuring them to operate at 9600bps and to connect on a restart of the boards they would be connected to.

Arduino IDE Serial Library

Unlike the C language code that is used to achieve serial communication on the HCS12, the serial communication libraries provided in the Arduino IDE are not clear about how the communication is achieved. Another challenge that was faced to figuring out how the serial libraries of the Ardunio IDE function in order to provide easy to read data to the HCS12. Eventually it was discovered that these standard function provide ASCII encoding to anything its sends out through its Serial.print() function. In this case since the user will not see the data being sent this is not useful at all, in fact it's the opposite. Later it was discovered that a single non-encoded byte could be sent, which is what lead to the need to split the digital readings into two bytes. One could also send an array of bytes.

This led to the HCS12 code being modified to wait for four bytes, recombine the bytes, and then make the correct decision on which instruction to send to vehicle.

SCI Interrupt Service Routine

Inside the interrupt service routine of SCI1, there is a necessity of checking which functionality has triggered the flag as both transmitter and receiver shares the same flag. When receiver gets a byte, RDRF bit (bit 5) in SCI1SR1 becomes high. Originally, (SCI1SR1 & (1<<5) == 1) code was used to mask the RDRF bit specifically. However, this method did not work during the testing. The solution to this problem is to use the pre-defined macro “SCI1SR1_RDRF_MASK” that masks the RDRF bit. After the code is replaced with (SCI1SR1 & SCI1SR1_RDRF_MASK), interrupt service routine correctly processes the received byte.

Conclusions

Throughout the designing stage, constructing stage, and the testing stage of the toy car controller, the group learned how to work together effectively. Upon completion of project, toy car controller using accelerometer was fully functional. The Bluetooth module successfully installed to achieve the extended project goal. The project demonstrated the importance of working simultaneously on different components by creating the agreement on how two components interface. This method increased the efficiency of development. During the development, team repeatedly realized the importance of documentation. Incremental development and testing approach eliminated bugs and problems at early stage, making the integration of sensors and microcontroller much efficient at later stage.

Overall, this project helped deepened the group's understanding of programming a microprocessor, interfacing with peripherals and sensors, and finally testing it. It was satisfying to see how individual pieces of knowledge throughout the semester were combined and used to complete one project.

VI. Refrences

1.Analog Devices . "Small, Low Power, 3-Axis ±3 g Accelerometer." *Sparkfun.com.* 2009. https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf (accessed April 20, 2016).

2.Mazidi, and Causey. *HCS12 Microcontrollers and Embedded Systems using Assembly and C with CodeWarrior.* Edited by 1st. Prentice Hall,, 2009.

3.PERICOM. "PT8A977B 5-Function Remote Controller." *Pericom.com.* July 6, 2012. https://www.pericom.com/products/home-appliance/remote-controller/part/PT8A977B (accessed April 20, 2016).

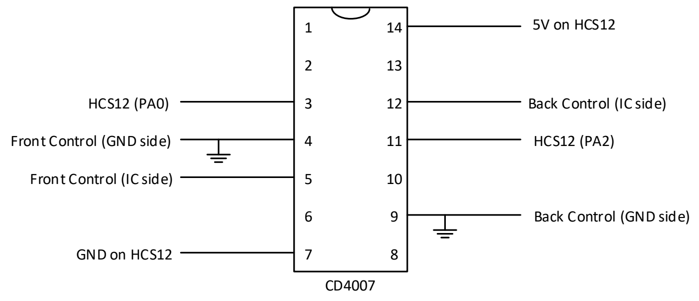
4.Roving Networks . "Bluetooth Data Module Command Reference & Advanced Information User's Guide." *Sparkfun.com.* March 26, 2013. https://cdn.sparkfun.com/assets/1/e/e/5/d/5217b297757b7fd3748b4567.pdf (accessed April 20, 2016).

5.Sadeh, Waseem. *ECE-470-11322 / ECE-570-11323 / CSE-470-11324.201610.* Jan 2016. https://moodle.oakland.edu/course/view.php?id=151039 (accessed April 20, 2016).

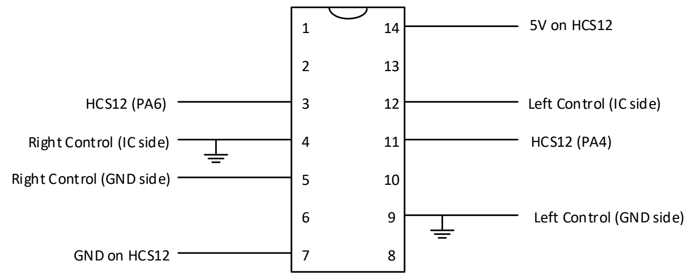
6.Sparkfun.com. "Using the BlueSMiRF." *Sparkfun.com.* https://learn.sparkfun.com/tutorials/using-the-bluesmirf?_ga=1.84799071.79649871.1460297461 (accessed April 20, 2016).

7.Texas Instruments . "TI.com." *CMOS Dual Complementary Pair*. September 2003. http://www.ti.com/lit/ds/symlink/cd4007ub.pdf (accessed April 20, 2016).

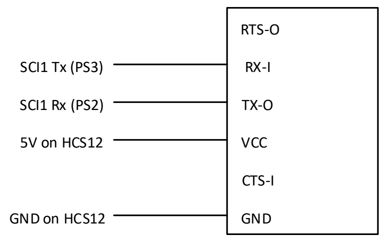
Appendix A (HCS12 Sensors Schematics)



CD4007

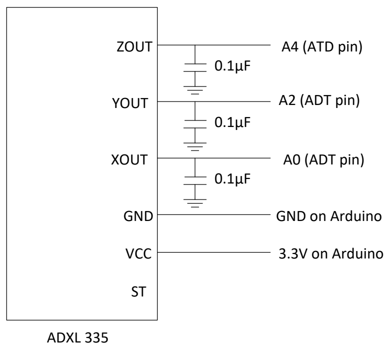


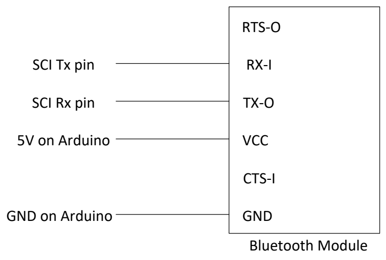
CD4007



Bluetooth Module

Appendix B (Arduino Sensors Schematics)





Appendix C (Wiring on Toy Controller)