Microprocessors
Boston University
ENG EC450

MSP430 RC Car

Final Report
May 6, 2011

Benjamin Duong	Yevgeniy Kolodenker
U21811421	U48508619

Table of Contents

Objective

The main objective of our project was to create an end-system that is able to be controlled wirelessly from a personal computer. The design of the system involved a graphical user interface on the PC that the end-user was able to enter commands that are wirelessly transmitted to a receiver that then is able control the radio-controlled car.

Application

The application of the complete system would be in either the military or construction sector. The user can use the workstation to direct the radio-controlled car into position and then electrically detonate a pack of explosives from a safe distance away.

Project Design

The entire system can be viewed as three separate subsystems networked together in order to create a complete end-to-end system. These three subsystems are the internal interfaces used:

1. Serial Interface – The first subsystem is the serial interface between the sending target board and graphical user interface on the workstation. The input to this subsystem is the commands from the user that dictate subsequent movement of the car. The commands are then sent through the USCI UART to the sending target board. The commands are encoded on the workstation before being sent to the first target board. The output of this subsystem is to the wireless interface.
2. Wireless Interface – The second subsystem is the wireless interface between the sending target board and the receiving target board. The input to this subsystem is the encoded instructions from the workstation. The instructions are transmitted wirelessly from one target board to another and are stored in a character array on the receiving target board. The output of this subsystem is to the electrical interface.
3. Electrical Interface – The last subsystem is the electrical interface between the receiving target board and physical motor powering the RC car. The input to this system is the character array of instructions in the second target board. The instructions are decoded one by one and a digital output from target board is converted to an analog signal seen by the motor. The output of this subsystem is to the DC motor contained in the car.

Implementation

At startup, both RF2500 target boards run configuration setups. Both stop the watchdog timer, initialize the onboard LEDS and configure the USCI B for SPI protocol. The sender target board also configures the backchannel UART for communication with the workstation. The receiver board configures the PORT 2 output pins to be outputs to the electrical interface.

The system starts with the first interface with the user entering valid commands into the graphical user interface written in MATLAB. As the user enters in commands, the commands are encoded into a sequence of 8-bit binary characters. The specific character used to represent a given command is given by a specific encoding scheme (Appendix A). The encoding scheme

allows for distances of up to 3.1 feet in a single instruction, therefore commands of over 3.1 feet are emulated with several instructions of 3.1 feet or less. The encoded instructions are stored in an array until either the user presses the run button or the "detonate" command is issued. At this point, MATLAB opens creates a serial object and connects it to the sending RF2500 target board through the USB port and writes to the RF2500 target board through its backchannel UART. When the serial connection is opened, first the number of instructions is written to a character variable on the RF2500 and then the script waits for confirmation from the target board. Each instruction is then written to an element in a character array on the RF2500, one instruction at a time with the script waiting for confirmation after each write. When all of the instructions have been written to the sender board, the MATLAB script waits until it receives a completion character from the sender board.

Whenever the MATLAB script attempts to write a character to the sender board, an interrupt on the board triggers to save that character from the buffer into a character array. The interrupts continue to trigger until the board has read in a number of characters equal to the number of instructions. The interrupt of the last character also starts and handles the wireless sending of the instructions.

The second interface is the transmission of data between the two RF2500 target boards. The sender board starts by transferring the packet length, packet address, and the total number on instructions into the first three elements of its buffer. Then the instructions stored in the character array are transferred to the buffer. After all of the data needed to be transferred is saved in the buffer, the sender board calls a function to send the packet to the other board and toggles the on board green led for visual confirmation.

On the other side of the interface with the receiving board, there will be an interrupt from PORT 2 when the packet has been sent. When the interrupt triggers, the packet will be fetched and the data stored in the receiving buffer is transferred to a character array. The interrupt handler then runs through each instruction individually. As each instruction is executed, bit masks are used to separate the instruction's opcode and argument. The code uses a switch statement with the opcode as the argument to determine what direction to move the car. For all instructions other than "detonate", the motion will be accomplished by outputting a logical high to one or more of the output pins for a certain amount of time. For forward and backward motion, the amount of time that the outputs are set high is proportional to the argument of the instruction. For left and right motion, the amount of time is fixed. The direction of motion determines the exact logic for the pin outputs (Appendix B). Each instruction ends with writing a logical low to all pin outputs before the next instruction is executed.

The last interface is the conversion of the digital output signal from the RF2500 board to an analog signal seen by the DC motor. In order to power the motor in both directions, the RF2500 needs to be able to apply a positive and negative voltage over the terminals of the motor. It also needs to provide the motor with enough power to overcome the rotational inertia and friction. The ability to reverse the polarity of the output required an H-bridge. The H-bridge uses two TIP32C PNP power transistors as the sources and two TIP31C NPN power transistors as the sinks for the current going through the motor (Appendix D). Each of the power transistors are switched on and off by a PN2222A NPN BJT. The PN2222A transistors are controlled by the

RF2500 target board. If the top left source and bottom right sink are turned on and the other sink and source are turned off, the current will follow through the motor from left to right, causing the motor to spin forward. If the input conditions are reversed, the current will follow from the top right source to the bottom left sink through the motor from right to left, causing the motor to spin backward. The four clamping diodes are used to prevent a large inductive current that is generated after the instruction is completed the logical outputs all go to zero does not damage the components. The current need by the DC motor to power itself is around 500mA and in order to provide this current, the output pins from the RF2500 are biased with a 1.5V battery. In order to simulate the "detonate" command, the output pin designated for the detonation is connected to an LED.

When the set of instructions is completed, the system runs through a cleanup; the receiver board wirelessly sends a character back to the sender board. It then resets its interrupt flags and waits for the next set of instructions. When the sender board receives the confirmation character, an interrupt triggers and saves the character in the UART buffer. The MATLAB script sees that a byte is available in the UART buffer and reads it to confirm that the previous instruction set is completed and then unlocks the GUI for the user. If the set of commands ended with a "detonate" command, the GUI locks indefinitely as it makes no sense to send more commands to an object that no longer exists.

Self-Assessment

Our project was ultimately successful; we achieved wireless communication which was a large focus of our initial design goal. Our final GUI is overall bug free. One bug found in the GUI and serial communication is that instructions sent too rapidly after the completion of the previous instruction set had a small chance of throwing the GUI off sync and required restarting the application.

The only issue found in our wireless communication was that we offered no digital signature in our transmitted signals. The wireless receiver would pick up any signals at 2.4GHz and this could potentially cause an issue in the security of the communication channel and the validity of the data.

The largest problem area of our project was the electronic circuit that operated our RC car. We had a lot of problems in providing enough power to our 1.5W motor. The microprocessor pins could only provide us with 20mA, which did not amplify to the 500mA of current needed by the motor. We also needed to be able to provide negative potential, which the microprocessor cannot provide by itself; this meant we need to build an h-bridge which allows us to reverse polarity through the motor. At first we tried using an LMD18200 H-bridge chip by National Semiconductors that did not work for us. The chip did not provide any current through it and was deemed faulty. After this setback we decided to build out own H-bridge. The H-bridge required moderately high power components that could handle 500mA, so we needed to use high power components. The H-bridge was difficult to modify to provide us with proper power output. In order to overcome this, we eventually biased the outputs with a constant current. This provided us with a solution but made our circuit require a large amount of batteries and also weigh more.

In the end, we had a fully functional microprocessor controlled by a GUI; however we could not provide enough power to the car for it to overcome friction.

Future Improvements

Our project could have used several major improvements. The largest improvement we could have done was to lower the weight of our electronics. By using an integrated chip or smaller discrete components on a PCB instead of a breadboard, we could lower the weight of our circuit drastically and then implement it inside the toy car chassis. The next largest source of weight is from the amount of batteries we used. The RC car originally bought ran from 2 AA batteries, whereas our circuit ended up using a 6V battery pack, 2 AA bias batteries and 2 AAA batteries. By using better components that provide more amplification and a better circuit design we could try to reduce our battery usage. With a lower weight of the electronics and more amplification, the wheels would be able to overcome the weight of the car easier and move on the floor.

The two second largest improvements we would have liked to add were left/right turning and range detection. The RF2500 is implemented with the ability to turn left and right; however, we were not about to implement because it required a second H-bridge which time and material constraints did not allow for. The issue of range detection is a big one. Our preliminary idea, without involving any extra peripherals was to just to use time as a measure of length. This method has multiple issues in that other variables account for the distance travel like the friction coefficient and amount of current being pulled through the circuit. An improvement would be to use a measure of absolute distance traveled provided from an external sensor instead of an estimated value.

Appendix A: Instruction Encoding Scheme

Bit	7	6	5	4	3	2	1	0
Meaning	Start	Opcode		Argument				

Start:

Indicates a new character – set to 0

Opcode:

00 – Detonate/Stop
01 – Forward
10 – Backward
11 – Turn

Argument:

The meaning of the argument depends on the opcode of the instruction.

Forward/Backward:

Determines the distance traveled

Turn:

00000 – Turn Left
11111 – Turn Right

Detonate/Stop

00000 – Stop
11111 – Detonate

Appendix B: Pin Output Layout

Command	P2.4	P2.3	P2.2	P2.1	P2.0
Forward	0	0	0	0	1
Backward	0	0	1	0	0
Turn Left	0	1	0	0	1
Turn Right	1	0	0	0	1
Detonate	0	0	0	1	0
Stop	0	0	0	0	0

Appendix C: Code and Listings

Component Listing:

1	Maisto R/C 1:24 Lamborghini Reventon
1	eZ430 RF2500 Development Kit
1	Breadboard
1	1.5W DC Motor
1	6V Battery Pack
2	AA Batteries
2	AAA Batteries
12	1kΩ ¼W Resistors
4	10kΩ ¼W Resistors
4	100kΩ 1/4 W Resistors
2	TIP31C NPN BJT Power Transistors
2	TIP32C PNP BJT Power Transistors
4	1N4007 Diodes
4	PN2222A NPN BJT Transistors

Code Included:

GUI	CarGui.m
Sender Board	Sender.c
Receiver Board	Receiver.c
Header Files	CC2500.c CC2500.h include.h TI_CC_CC2500.h TI_CC_hardware_board.h TI_CC_msp430.h TI_CC_spi.c TI_CC_spi.h

Appendix D: H-Bridge Schematic