ECE 350 Final Project Report

Aditya Sridhar, Gerry Chen, Faith Rodriguez

December 2018

1 Introduction

In this project, two wireless controlled robot raced each other in an attempt to hit checkpoints that would allow them to gain points and power ups. In order to implement this project, a combination of handmade hardware components and various forms of software were required. This report demonstrates the overall project design, the input and output used, the challenges faced, circuit diagrams, the testing approach, and process and MIPS assembly logic.

2 Overall Project Design

The robots are controlled using wireless controllers that contain a steering wheel and pedal to control steering and throttle, respectively, for each robot. These values, which indicate the robots' state, are read using an ADC0808 chip in conjunction with the FPGA and stored appropriately for the processor to use. This controller FPGA also handles speed/steering power-up duration logic for robots and stores motor effects. The wheel commands are then sent over UART to the radio module, where they are communicated wirelessly to the robots. When a robot connects with a checkpoint both it and that checkpoint send out a message through UART communication that indicated that the object has been touched. The receiver FPGA than stores these values, recognizing that messages sent within a specific time frame correlate to the same touch event. These touch events correlate to point updates to the robot's score. The receiver FPGA then outputs these values to the VGA, which displays the points on a scoreboard. Please note that we had a bug in our logic, which caused the FPGA to not recognize when two messages were a part of the same touch event. Due to this, scores are not updated. In response, the player one score was instead turned into a game time counter to show that input can be received and displayed on the VGA.

Figure 1 show the schematic of the project, including overall underlying features implemented at each phase of the project.

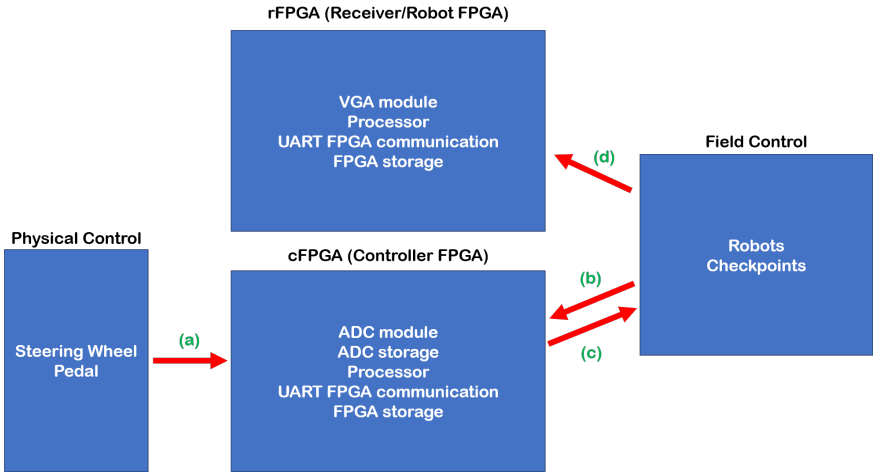


Figure 1: Schematic of the robotic control. (a) The physical control enforced by the combination of the steering wheel and the pedal produce the steering and throttle, respectively. This information is fed into the cFPGA module through the ADC. (b) Using UART, touches between robots and checkpoints enforce the power-up effects on speed and steering for those robots. (c) The processor-converted left and right motor velocities are transmitted as packets via UART to the robots for necessary control. (d) The power-up effects on points are enforced by UART-based touches between the robots and checkpoints, just as in part b.

3 Input and Output

Primary input and output hardware for the FPGAs included the ADC (non-negligible!), radio, and VGA display.

3.1 ADC

The ADC0808 IC was used for its availability. Part of the difficulty in interfacing with the IC arose from switching the addresses at the appropriate times and waiting for EOC to be asserted before saving the data, but also in maintaining a state machine to register the ADC values to be later processed by the processor.

The state control had to be made such that the clock, ALE, START, and ADDR pins were asserted at the proper times according the specifications and EOC. The digital out pins had to, of course, be read solely at the appropriate times between EOC and START of the next conversion. By cycling through the 4 analog input addresses of the 4 controllers, the human control values were updated around 1kHz - far more than adequate. Combined with the stability of the registers during read operations and accessibility by the main processor, the ADC module was a non-negligible task to implement.

3.2 Radio

The UART communication used to interface with the radio module required the use of a non-memory mapped UART module and the accompanying control to manage the packet protocol (packet protocol provided in Appendix A.2). The byte packet protocol was custom developed to maintain robust, bidirectional, low-latency communication among devices. The protocol supported unambiguous communication from any device to any other set of devices. The transmission must also execute exactly once on a control signal edge and complete a full transmission in the presence of interrupting requests. The receiving must hold on to the data in an appropriate buffer before saving the data to a specified space upon a control signal.

Although the physical layer- decoding/encoding of the UART signals from bytes were imported from a Quartus IP generator, the real challenge was managing the receiving and transmission of arbitrary bytes to unambiguously define the exact stream of data across multiple bytes and devices. Due to the inherently unreliable nature of all communications (especially wireless!), error detection and handling was very challenging and required significant original thought/design.

3.3 VGA

The VGA is used to display the game state information. The VGA requires the use of custom design to selectively stitch together many images (i.e. background, text, numbers) into one screen in real time.

4 Challenges and Solutions

4.1 ADC

Analog to Digital Conversion proved to be a much more difficult task than initially expected. While we did have a starting basis provided due to a lab activity, the process of storing the values into registers turned out to add a large complication. Ensuring that the conversion of the signal specified by ADDR pins had completed and indeed reflected the signal desired (as opposed to the previous signal, for instance), required careful timing and trial/error. The storing of the read values into registers accessible by the processor also posed challenges pertaining to difficulty of verification. The general approach we took was to output one single register's data to LEDs and confirm the LED value matched expected behavior of pin voltage even in the presence of many other signals.

4.2 UART Implementation

Overall, UART turned out to be a much more difficult task than originally anticipated. It took us a really long time to even understand the use of the UART

module as provided by a Quartus async component. Once making sense of the inputs and outputs that were required of us. While getting the transmission and reception of a single packet did not prove to be a hugely difficult task, we had a lot of trouble implementing the transmission and reception of an entire message. Initially the FPGA was constantly receiving packages but never registering a message. When we attempted to look at the data of the packages being received they appeared to have no correlation to each other. However, once we realized that we needed to add a delay to the robot's communication code, we were able to begin to once again make progress in figuring out UART. Once we figured out basic UART communication we experienced additionally difficulty obtaining integration between that module and the FPGAs. It was difficult to determine the validity of data being sent and received and the conversion of message formatting required to reach the radio module added an extra layer of difficulty. Finally, testing this module and its associated submodules was extremely difficult due to the fixed speed requirement and inability to perceive information that fast. A signal analyzer was key to tracking down bugs and ensuring error-free communication, while a UART-USB converter was critical to ensuring matching of the protocol.

4.3 VGA

5 Circuit Diagrams

4 custom PCBs were ordered for this project:

1. controller board - houses ADC0808, 2 potentiometers, joystick footprint, 3 switches, buzzer + driving MOSFETs, pull-up resistors, and a 40-pin IDC header.

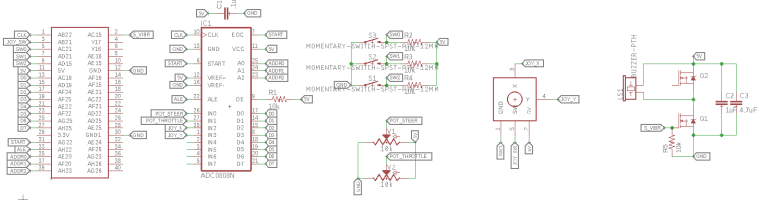
2. radio board - held an 8-pin Nordic nRF24 module, an Arduino nano for packet transcieving, and a 40-pin header to connect to the FpGA. These electrical connections significantly improve reliability and robustness of the system.

3. robot board - the main board that makes electrical connections in the robot. This involves motor control, touch sensing, wireless communication, and visual identification.

4. checkpoint board - the main board that makes the checkpoint electrical connections including many simlar functionalities as the robot board but without motor control and with more refined reading.

6 Testing

When testing modules that connected software to the hardware that had been created, an incremental approach was used. This means that we would focus on



getting one aspect of a component completed and then tested its functionality on the FPGA through a combination of LED displays and switches. For example, when testing UART communication. We first implemented and tested single packet receiving, then single pack transmitting. We then approached message receiving then message transmitting. Starting small allowed us to be confident in our code base and helped isolate bugs in order to make the development process more smooth.

7 Processor Logic and Assembly

To design the logic for the controller and receiver/robot FPGAs, we used the five-stage single-issue 32-bit processor and MIPS assembly code. In this section, we describe the modifications made to the processor, a description of how we tested the processor, and a run-down of the assembly integration linked to the processor.

7.1 Processor Modifications

We used key features of the original processor to facilitate the control flow. Some existing features of the processor include multiplication/division (Mult/Div) modules with Modified Booth's Algorithm, hazard handling, and full bypassing.

To build upon this version of the processor for our project, we first motivate the use of the processor. Based on the current modularization of the code-base/logic, we notice that many processes, including the ADC conversion and the power-up effects on the robots, require the use of the processor, thereby encouraging the need for an appropriate design model. We choose to store the inputs to processor-based operations in storage units, defined by registers. In this way, the processor acts independently from other processes and does not directly rely on the outputs of the other processes. We may also encounter delays and unstable inputs that may adversely affect the processor itself; therefore, the introduction an intermediate module serves as an apt choice and reinforces the need for efficient modularization.

For such implementation, we modify the processor to include additional custom functions, which are defined as follows:

- ***time $rd***: Returns the current time/cycle of the game and stores it in register ***$rd***.

- ***imova $rd, $rs, N***: Moves data into the processor's register file (register ***$rd***) from the intermediate module's "ADC storage" (register ***$rs***) for the cFPGA module; can add an immediate value (***N***).
- ***imovp $rd, $rs, N***: Moves data into the processor's register file (register ***$rd***) from the intermediate module's "communication/packets" (register ***$rs***) for each FPGA module; can add an immediate value (***N***).
- ***imovt $rd, $rs, N***: Moves data into the processor's register file (register ***$rd***) from the intermediate module's "calculation/timings" (register ***$rs***) for each FPGA module; can add an immediate value (***N***).
- ***emovt $rd, $rs, N***: Moves data from the processor's register file (register ***$rd***) to the intermediate module's "calculation/timings" (register ***$rs***) for each FPGA module; can add an immediate value (***N***).

7.2 Assembly

We tested the above processor using assembly code, which defines the list of operations for both FPGAs and is imported as *.MIF* files into the Quartus megafunction for instruction memory. One MIPS code file was generated for each FPGA module.

In the cFPGA MIPS file, we begin by reading in the ADC digital values from ADC. Thereafter, for each possible robot-checkpoint touch, we check if the advertising times for both the robot and the checkpoint match (or differ by δ, where δ is about 500 ms). We also request the same of their touch durations, or the length of time before the robot and checkpoint no longer touch. If both conditions are satisfied, then the "match" triggers a power-up effect on the robot, where the effect can be applied as follows:

- Increase/decrease the speed of the current robot using some operation such as *ADD* and *MUL*.
- Increase/decrease the steering of the current robot.
- Increase/decrease the speed of the opposing robot using some operation.
- Increase/decrease the steering of the opposing robot.

All possible robot-checkpoint combinations are traversed through and checked for similar times and touch durations, and for each match, we incorporate its respective effect using processor computation. Finally, We then loop back to the beginning and restart the for matched robot-checkpoint combinations. In this way, we are "actively watching" for touches at all iterations.

Subsequently, we notice the same patterns for the rFPGA as for the cFPGA, except the effect is applied using only points rather than both speed and steering. An additional difference of the rFPGA is that it does not read from the ADC; therefore, we do not implement the ***imova*** custom function here.

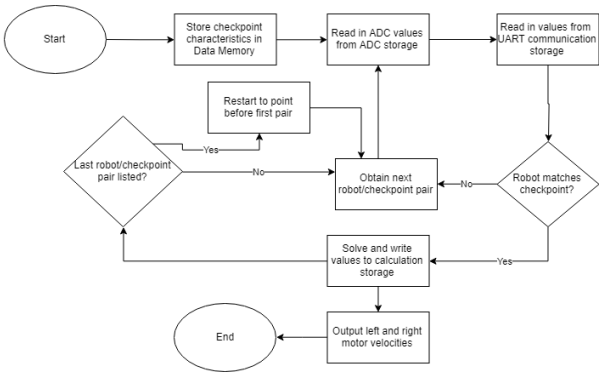


Figure 2: Flowchart of MIPS assembly code for the cFPGA

The flowchart in Figure 2 demonstrates the overall flow of the assembly code and how it is used in the context of the controller FPGA. NOte that the flowchart for the receiver/robot FPGA is very similar, except we do not read from the ADC.

8 Conclusion

We designed and implemented a robot game that compete to earn the largest number of points possible. In addition to the design overview, we further presented individual components and key features that contributed to the larger project objectives.

9 Pictures of the Project

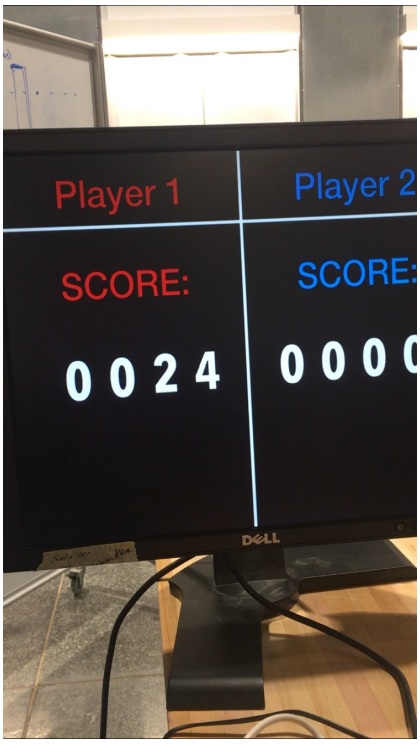


Figure 3: VGA Scoreboard

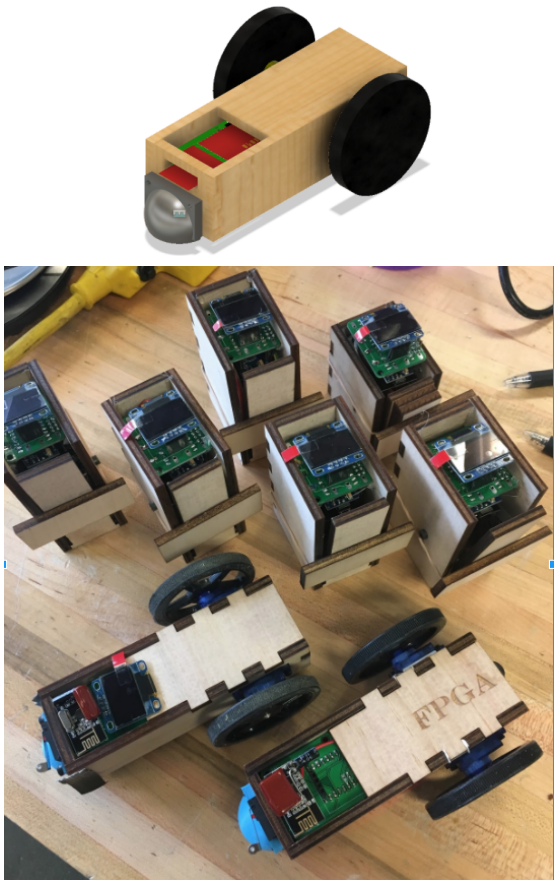


Figure 4: Completed robots and checkpoints

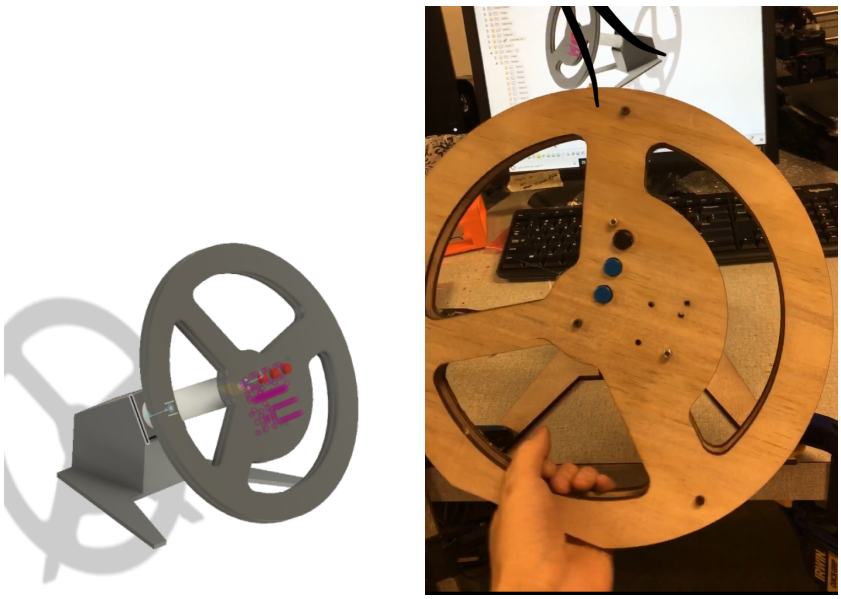


Figure 5: CAD model and completed steering wheel

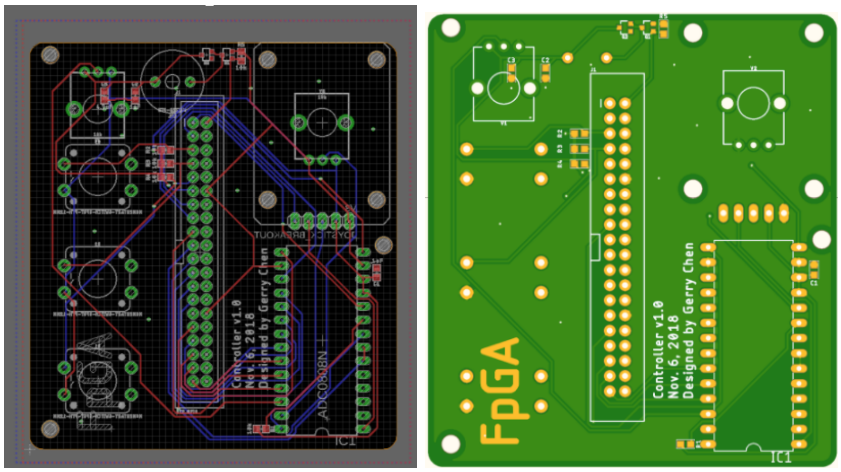


Figure 6: Controller board rouding and PCB

A Communication Packet Protocol

A.1 Communication Protocol

All packets shall be sent as an 8-bit, no-parity, 1 stop bit message at 115200 baud. In other words, a 115200 8N1 UART packet.

Furthermore, each packet's first bit will be a 1 if there will be another byte after. The last packet of each set will have a 0 in the first bit.

A.2 Transmission Structure

Each transmission shall consist of the following 7-byte structure:

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
time[27-21]	time[20-14]	time[13-7]	time[6-0]	command	payload[13:7]	payload[6:0]

This is invariant of whether the transmission is from the FPGA or radio.

A.3 Command Specifications

A.4 7-bit command codes and payloads

The 7-bit command and accompanying payload bytes are defined as follows

23emsender	24emreceiver	command		payload	
		6 5 4 3 2 1 0		13 12 11 10 9 8 7	6 5 4 3 2 1 0
cFPGA	robot	000	robot ID	left motor vel	right motor vel
robot	FPGAs	001	robot ID	touch duration	
checkpoint	FPGAs	010	checkpoint ID	touch duration	
rFPGA	cFPGA	100	game state	auxilliary info	
cFPGA	rFPGA	101	powerup state	auxilliary info	

A.4.1 4-bit game state command codes

The 4-bit game state commands sent from the rFPGA to the cFPGA are as follows

Description	game state command	auxilliary info
Start game	0000	–
End game	0001	–
Reset game	0010	–
Set time	0011	–
Disable player	0100	robotID (bits 13-10)
Enable player	0101	robotID (bits 13-10)
Disable powerups	0110	–
Enable powerups	0111	–
Clear all powerups	1000	–
Acknowledge receipt	1111	powerup command acknowledged (13-10) robotID (9-6)

A.4.2 4-bit powerup state command codes

The 4-bit powerup state commands sent from the cFPGA to the rFPGA are as follows

Description	powerup state command	auxilliary info
Powerup deactivated	0000	robotID (bits 13-10) - powerupID (bits 9-2)
Powerup activated	0001	robotID (bits 13-10) - powerupID (bits 9-2)
Acknowledge receipt	1111	game command acknowledged (13-10)