UART Battleship

Sharyar Khalid
Katie Neff
Adolfo Pineda

May 2016

We affirm that this report and its contents are solely the work of Sharyar Khalid, Katie Neff and Adolfo Pineda.

Signature	Date
Signature	Date
Signature	Date

Contents

List of Figures

List of Tables

1 INTRODUCTION

The purpose of this project was to use the previously created Rs232 protocol system and create a two player game from it using the NIOS II processor and C programming language. The game chosen was battleship. Players will send coordinates back and forth through the UART and these signals will be interpreted by the C program. This report will cover the hardware design and testing process of the UART system as well as the top level game system. The UART was used simply as a way to transmit characters back and forth between computers using the NIOS II Eclipse IDE. The hardware can be used for any system that needs to send and receive single characters. The Battleship game was a good method to present the functionality of the UART system and represented one of many uses of this kind of hardware.

2 DISCUSSION

Here the hardware design of the UART system as well as the software design of the game will be discussed in detail.

2.1 Hardware Design

The hardware design included three major modules: the transmitter, the receiver and the NIOS II processor. Figure 2.1 shows a top level schematic of the entire system.

2.1.1 Transmitter

Major Functions

The transmitter will take 8 bits of parallel data and output the data in serial at a rate of 9600 Hz. The transmitter is meant to send an ASCII character over a single wire.

Inputs and Outputs

The inputs and outputs to the system are described as follows.

Inputs

- `data` - 8 bit parallel data bus that accepts and ASCII character.
- `load` - High when the 8 bit parallel data bus has been loaded with the correct data.
- `transmitEnable` - Allows to system to transmit data, active low.

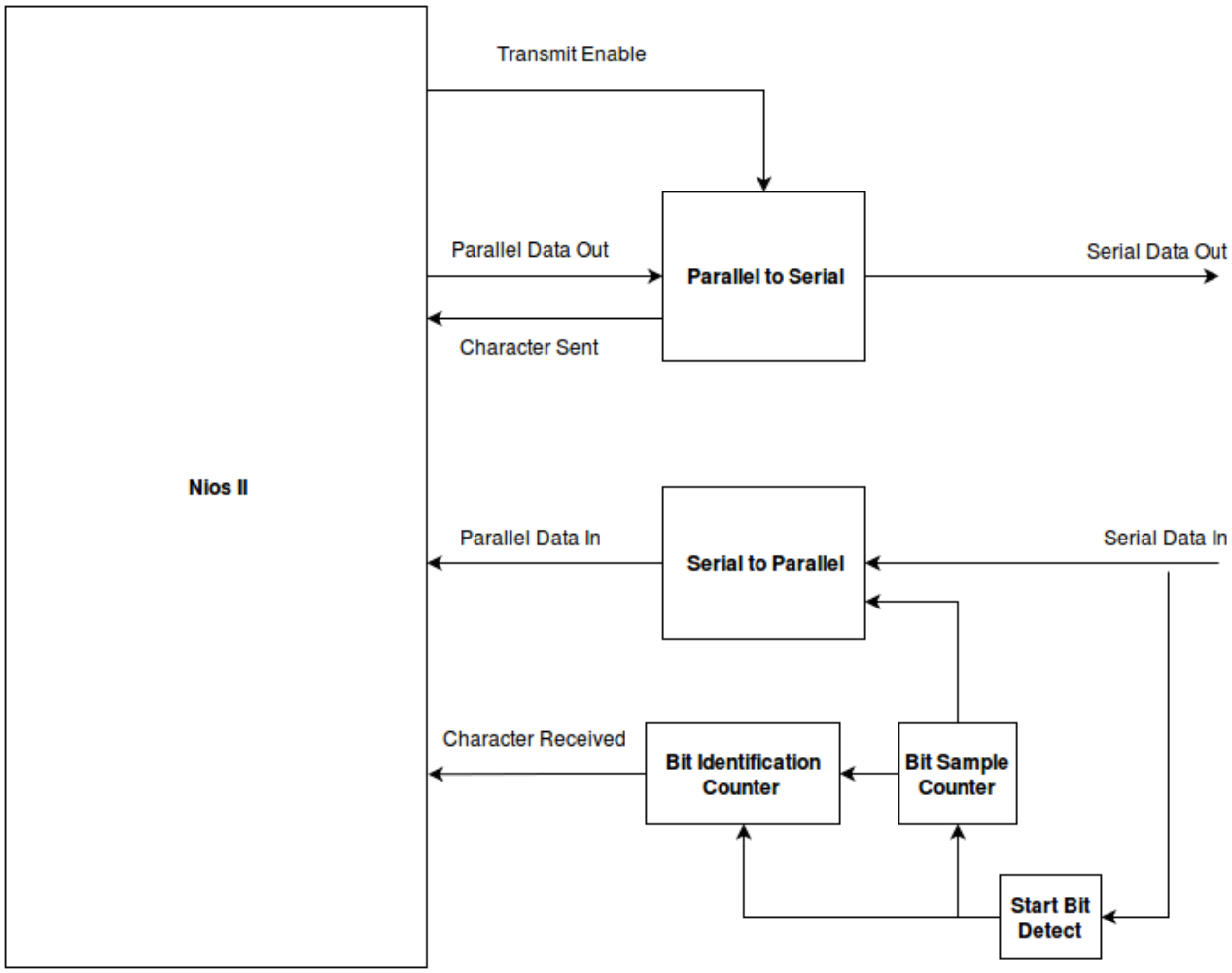

Figure 2.1: Top level schematic of communication system.

Outputs

- `dataOut` - 1 bit serial data out. If transmitEnable is low, outputs high Z. When transmit enable is high, outputs most significant bit on the buffer of the transmitter
- `characterSent` - High for one minor clock cycle when 10 bits worth of data is sent by the transmitter.

2.1.2 Receiver

Major Functions

The receiver will take serial data and covert it to 8 bit parallel data to be interpreted as an ASCII character. The receiver expects incoming serial data to have a start bit of 1, eight middle bits representing the character, and an end bit of 0. After the start bit is detected, the receiver will accept the next eight bits and convert them to parallel.

Inputs and Outputs

The inputs and outputs to the system are described as follows.

Inputs

- `data` - 1 bit serial data port

Outputs

- `dataOut` - 8-bit converted parallel ASCII character.
- `characterReceived` - High for one minor clock cycle when 10 bits worth of data has been received by the system (i. e. the start bit, the end bit, the middle character bits).

2.2 Software Design

2.2.1 NIOS II Processor

The software portion was designed using a NOIS II microprocessor. The microprocessor was integrated with the hardware design and has inputs and outputs as described.

Inputs

- `data_bus_in` - 8 bit data received by system
- `character_received` - High when a character is received by the communication system
- `character_transmitted` - High when a character was transmitted through the communication system

Outputs

- `data_bus_out` - 8 bit data to be transmitted
- `transmit_enable` - Allows the system to transmit data on the `data_bus_out` port, active low.
- `load` - High when data is ready to be loaded onto the buffer in the transmitter

2.2.2 Battleship Game Design

The top level interface of the communication system was implemented as a Battleship game between two players. The UART design was used to send data that would be interpreted as coordinates for a missile.

Game Initialization

The game begins by setting up a "board" for the user which is output to the console as an ASCII picture. The output to the console is represented in Figure 2.2.

```
  0 1 2 3 4 5 6 7 8 9
0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1 ~ d ~ ~ ~ ~ ~ ~ ~ ~
2 ~ d ~ ~ ~ ~ ~ ~ ~ ~
3 ~ d ~ ~ ~ ~ ~ d ~ ~
4 ~ d ~ ~ ~ ~ ~ d ~ ~
5 ~ d ~ ~ ~ ~ ~ d ~ ~
6 ~ ~ ~ ~ ~ ~ ~ d ~ ~
7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9 ~ ~ ~ ~ ~ ~ ~ ~ d d
```

Figure 2.2: ASCII image of the game board.

The tildes represent water while the "d" represents part of the boat.

The game then prompts the user for input, asking if they are player one or player two. Both players cannot choose the same number. Based on the user's selection, the game continues into one of two game modes, `playerOnePlay()` and `playerTwoPlay`, where player one one has the first turn and player two starts off waiting for the attack of player one. Figure 2.3 shows the game flow at initialization.

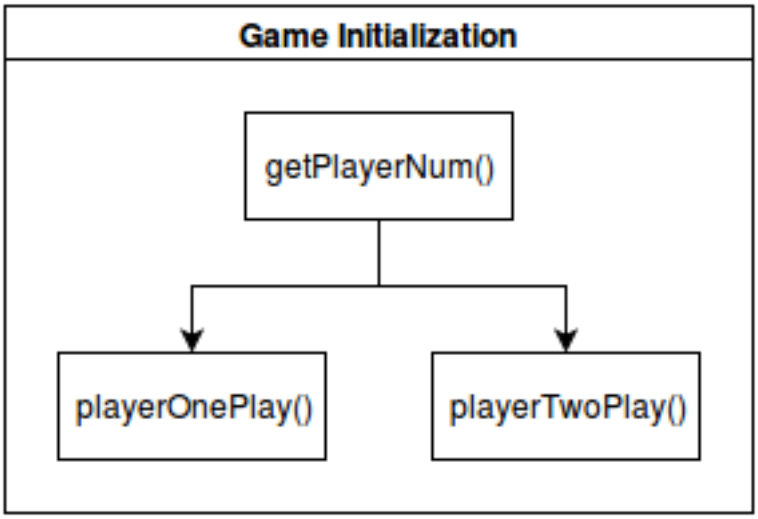

Figure 2.3: Game initialization prompts the user for their player number and then continues into the appropriate function.

Game Play

Each player follows the same game flow. When it is the players turn, they are prompted for coordinates. The coordinates are numbers 0 - 9 and represent predicted x and y positions of the enemies boats. The player sends the coordinates over the UART, and the are received by the enemy. Next, the player waits for the response. The response is sent by the enemy over the UART and tells the player if their shot hit a boat or missed and hit water. These characters are received as characters "h" for hit and "m" for miss. The characters are interpreted by the program and a message is printed to the console ("Hit!" if the user receives and "h" and "Miss!" if the user receives an "m"). If the user hit an enemy boat, their score increments by one.

Next, the enemy takes a turn. The players program waits to receive coordinates from the enemy. The coordinates are received and interpreted as integers. The program looks at the players 2D game board at the indexes received by the UART and checks for the presence of a boat at these x and y values. A tilde in that position indicates water and thus a miss. The player's program sends an "m" character over the UART to be interpreted by the enemies program. If there is a "d" in the array position, this is a hit, and the program send an "h" over the UART to be interpreted by the enemy. The player then gets another turn, and the game proceeds. A flow chart and state machine diagram describing game play is shown in Figure 2.4.

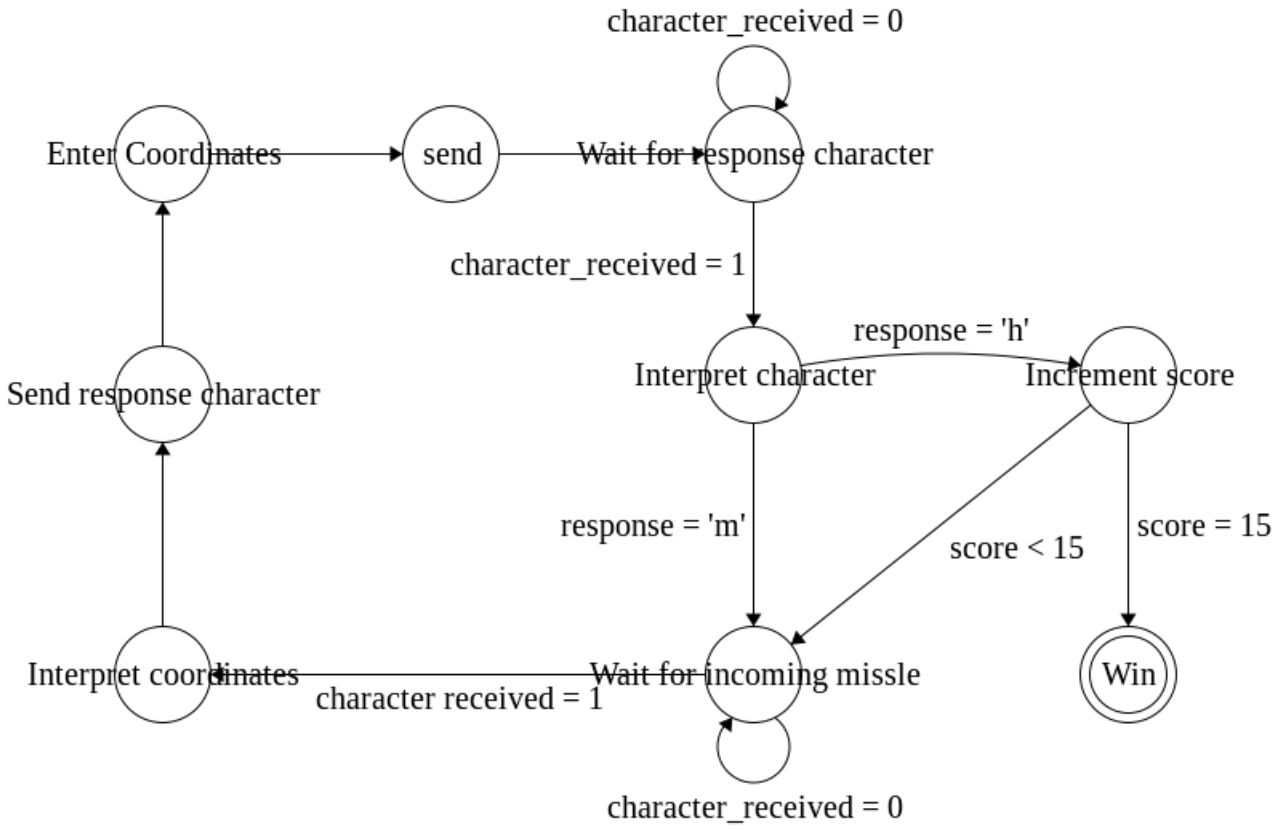

Figure 2.4: State machine of game play.

After one player has a score of 15 (a hard coded value based on how many boats there are on the board), the player wins and the game exits.

3 TEST PLAN

To fully test the battleship system, both hardware and software must be tested for correctness. To test hardware, we will write testbenches in iVerilog for both receiving and transmitting functions along with a top level testing module for synthesis. The results will be examined in GTKWave during simulation and with Quartus Signal Tap Logic Analyzer during synthesis. Specific tests will be written for both the modules. The transmitter but receive parallel input and the receiver must receive serial input. After the hardware tests are approved, the hardware can be integrated with the Nios II processor.

To test the software, we will use console output and `printf` statements to make sure the program is receiving the correct signals. If the software consistently receives corrupt data, correcting functions will have to be implemented.

4 TEST CASES

4.1 Receiving

Test Set Up

To test the receiving module in simulation, a testbench was written in iVerilog that sends the ASCII binary value of the word "test" once bit at a time to the input. A similar method will be followed in synthesis. Using a 9600 MHz clock, the bits from the binary value of the word "test" will be sent to the input of the receiver, one bit every clock cycle.

Inputs and Outputs

The inputs to the system will be the bit string of the ASCII word "test", one bit per clock cycle. All outputs to the system will be evaluated for correctness, i. e. the parallel data bus and the character received ports.

Expected Results

When examining data bus output on the Signal Tap and GTKWave, we should see the ASCII letters of the word "test" appear after the "character received" signal is high for one clock cycle. They should appear in order. The "character received" signal should go high for one clock cycle after 10 bits have been fed to the serial-to-parallel receiver.

Pass/Fail Criterion

The test fails if the expected results are not met.

4.2 Transmission

Test Set Up

To test transmission in simulation, a testbench was written in iVerilog that sent the ASCII values of the letters in the word "test" to the parallel data input bus of the transmitter. A similar test was designed in synthesis, where the input data bus was directly ported to the 8 bits in the ASCII value of the letter "t".

Inputs and Outputs

The inputs for simulation and synthesis will be an 8 bit ASCII value. The examined outputs will be the serial data out port and the character sent port.

Expected Results

The serial data out should output one bit per major clock cycle and the pattern should follow the byte value of the letter that was given on the input. The character sent port should be high for one minor clock cycle after 10 bits have been sent by the transmitter.

Pass/Fail Criterion

The test fails if the above expected results are not met.

4.3 Top Level Game

The software portion of the system will be tested using console outputs and `printf` statements for debugging. The end result should resemble a game that correctly receives and interprets data from the serial in port. Testing fails if the program outputs a corrupt or unexpected ASCII character.

5 RESULTS

5.1 Transmission

The results from the simulation testbench are shown in Figure 7.1. The simulation shows the correct results, with the bits contained in each ASCII code along with the start and stop bits output on `data_out` once every clock cycle. The output `character_sent` goes high after the entire character has been sent as expected.

The results for synthesis are found in Figure 7.3. This results shows one transmission of the ASCII character "t". The `data_out` bus matches the one in simulation, showing that the ASCII character "t" was correctly sent. Additionally, `character_sent` goes high for one clock cycle after the data has been sent as expected.

5.2 Receiving

The results from the simulation testbench for the receiver are shown in Figure 7.2. This waveform shows the receiver receiving the character "t" , correctly displaying a "t" after receiving all the data. The `character_received` output goes high for one minor clock cycle just before the "t" is received on the parallel data bus. This was a minor discrepancy corrected in synthesis.

The results from the synthesis testing module are found in Figure 7.4. This waveform shows the receiver correctly interpreting the serial data and outputting all the letters in "test" in order. The `character_received` port goes high right when the character is correctly received, as expected.

5.3 Top Level Game

After much tweaking and implementing error correcting in the software, the console output of the final result of the game appeared as pictured in Figure 7.5. All functionality of the game was tested and produced expected results. When testing the score keeping functionality of the game, the console output correct results shown in Figure 7.6.

6 SUMMARY AND CONCLUSION

Overall, the lab covered many things that have previously been covered and new things that were just introduced. The most important thing we learned was how to incorporate asynchronous serial communication in NIOS II microprocessor onto our DE1-SoC FPGA board to communicate with other FPGA boards. We did this by starting at a base design, which we implemented for Lab 4. We first got different components (shift registers, counters, parallel to serial registers, etc) of our hardware design ready by following the diagrams that were given to us in the Lab 4 specification. Then, we introduced the microprocessor component for serial communication using transmit and receive signals. We also learned the importance of design process when creating a specification all by ourselves as we were solely responsible for the implementation design for the game of Battleship. It was a trial and error process, we also learned during the trial and error that the RAM on the board was very limited and that making global variables fills its up really fast so we decided to make the maximum use of our SRAM to store the board for the game correctly. We kept making revisions of the initial implementation plans we had for our game to make it compatible for both boards. As a result, we successfully implemented our game of Battleship using the microprocessor and the FPGA through the tasks mentioned in the paragraph above. We were also able to add support for general purpose I/O to enable information exchange between the processor and functionality both within and external to the FPGA.

The biggest takeaway of this lab was learning how to how to incorporate a NIOS II microprocessor onto our DE1-SoC FPGA board and develop a successful asynchronous network communication system that could be used to make a multi-player game of Battleship. Another takeaway was to learn how to reiterate and improve the design and usability of an existing model based on new requirements. In the process of working with the microprocessor, we got introduced to a variety of new tools such as eclipse and Qsys, as well as the console interface of the eclipse IDE. Learning how to adapt to different tools in a short period of time would be essential in the industry since most industry jobs will not have a default set of tools that we can learn once and always use. It will be a more dynamic environment, therefore being able to adapt to different tools or even to new technology quickly will be an essential skill in the industry. In conclusion, we used the new tools we were introduced to make a successful applications using the NIOS II microprocessor. In this process, we also learned how to make a working model for not only our board, but also design meaningful software for our microprocessor that is compatible with other FPGA system and hardware designs.

7 APPENDICES

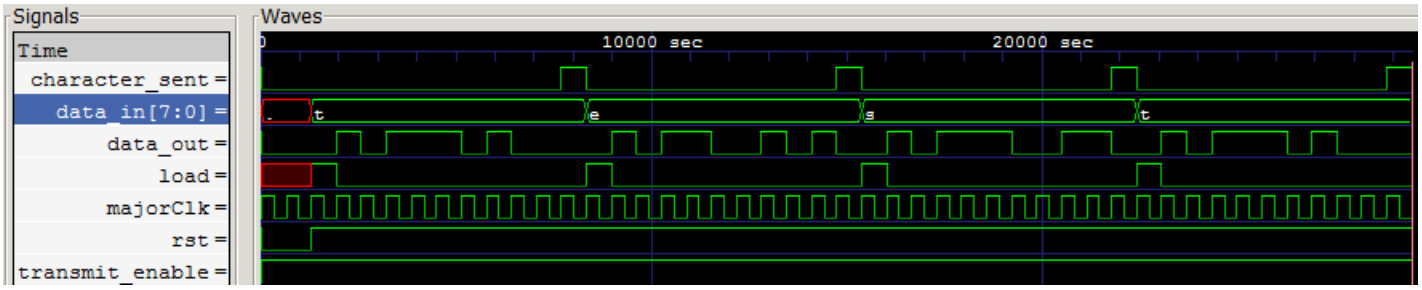

Figure 7.1: GTKWave of transmitter testbench

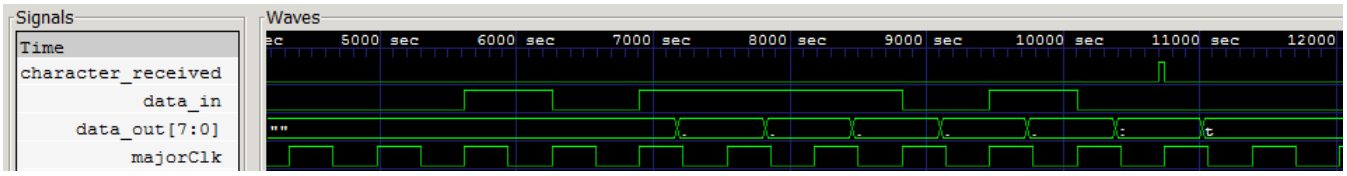

Figure 7.2: GTKWave of receiver testbench

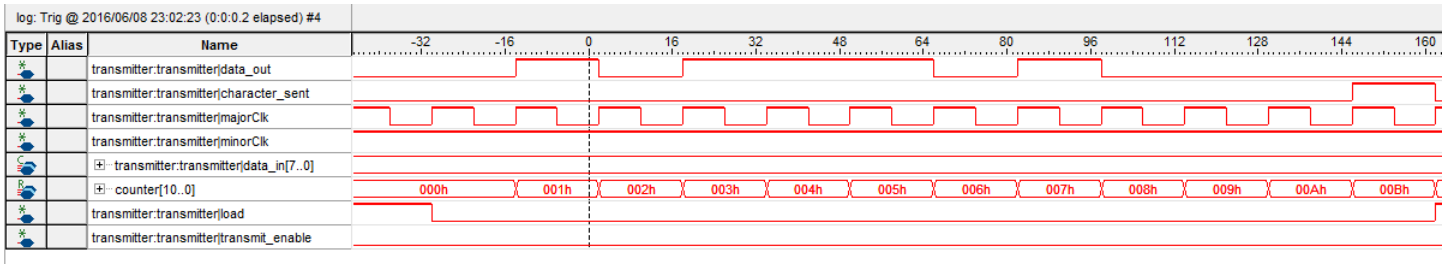

Figure 7.3: Signal Tap of transmitter testing module

Figure 7.4: Signal Tap of receiver testing module

Figure 7.5: Console output of the battleship game.

```
battleship_with_sram Nios II Hardware configuration - cable: DE-SoC on localhost [USB-1] device ID: 2 instance ID: 0 name: jtaguart_0
  0 1 2 3 4 5 6 7 8 9
0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1 ~ d ~ ~ ~ ~ ~ ~ ~ ~
2 ~ d ~ ~ ~ ~ ~ ~ ~ ~
3 ~ d ~ ~ ~ ~ ~ d ~ ~
4 ~ d ~ ~ ~ ~ ~ d ~ ~
5 ~ d ~ ~ ~ ~ ~ d ~ ~
6 ~ ~ ~ ~ ~ ~ ~ d ~ ~
7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9 ~ ~ ~ ~ ~ ~ ~ ~ d d
Enter latitude:
5
Enter longitude:
7
You hit!
Your score: a
Player 2's turn... Miss!
  0 1 2 3 4 5 6 7 8 9
0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1 ~ d ~ ~ ~ ~ ~ ~ ~ ~
2 ~ d ~ ~ ~ ~ ~ ~ ~ ~
3 ~ d ~ ~ ~ ~ ~ d ~ ~
4 ~ d ~ ~ ~ ~ ~ d ~ ~
5 ~ d ~ ~ ~ ~ ~ d ~ ~
6 ~ ~ ~ ~ ~ ~ ~ d ~ ~
7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9 ~ ~ ~ ~ ~ ~ ~ ~ d d
Enter latitude:
6
Enter longitude:
7
You hit!
Your score: b
Player 2's turn... Miss!
  0 1 2 3 4 5 6 7 8 9
0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
1 ~ d ~ ~ ~ ~ ~ ~ ~ ~
2 ~ d ~ ~ ~ ~ ~ ~ ~ ~
3 ~ d ~ ~ ~ ~ ~ d ~ ~
4 ~ d ~ ~ ~ ~ ~ d ~ ~
5 ~ d ~ ~ ~ ~ ~ d ~ ~
6 ~ ~ ~ ~ ~ ~ ~ d ~ ~
7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
9 ~ ~ ~ ~ ~ ~ ~ ~ d d
You win!
```

Figure 7.6: Console output when a player wins the battle ship game.