Lund University

Digital projects
EITF40

Battleships gaming system

Author:
Niklas Hjern
Jonas Vistrand

Supervisor:
Bertil Lindvall

March 4, 2013

Abstract

The purpose of the project was to construct a gaming system pre-programmed with the game Battleships. The game consists of two parts, singleplayer and multiplayer. The singleplayer game is played against a computer AI with the ability to play the game making logical decisions based on previously made guesses. The multiplayer game is played against another system by connecting the two using the communication protocol SPI. The system itself consists of a microprocessor of type ATmega16, a 16 digit keyboard and a 128x64 pixel LCD display. Every input to the game is read from the keyboard and the game is controlled by entering the coordinates of a guess, consisting of a letter A-F and a number 0-9, on the keyboard. Upon powering up the system the user is presented with the two previously mentioned choices and also a third, highscores, which presents a list of the 5 previously lowest scores and the initials of the corresponding player.

By the end of the project the singlepayer part of the game was finished and fully functional. The multiplayer part however was implemented but never tested.

Contents

1 Introduction **1**
- 1.1 Initial specifications . . . 1

2 Hardware **1**

3 Software **3**
- 3.1 Battleship . . . 3
- 3.2 Game . . . 3
- 3.3 GameUtil . . . 3
- 3.4 Display . . . 3
- 3.5 Draw . . . 3
- 3.6 Keyboard . . . 4
- 3.7 AI . . . 4

4 Theory **4**
- 4.1 SPI -Serial Peripheral Interface . . . 4

5 Execution **5**

6 Results **8**

7 Discussion **8**
- 7.1 Evaluation of project . . . 8
- 7.2 Improvements . . . 8
  - 7.2.1 General . . . 8
  - 7.2.2 Security . . . 9

8 References **9**
- 8.1 Figures . . . 9

1 Introduction

The purpose of this project was to program a version of the game Battleships on an ATmega16 microcontroller and display it on an LCD display. The game was to be played against an opponent with a similar microcontroller as ours using the communication protocol SPI.

1.1 Initial specifications

The playing field should consist of a 6x10 matrix where the user can deploy a specified number of boats with sizes ranging from 2 to 3 squares. To let the player communicate with the microcontroller when e.g. placing a boat on a coordinate in the matrix, a 16 digit keyboard (0-9 and A-F) is to be used. During the course of the game the playing field is at all times displayed on a 128x64 LCD display for the users viewing purpose. Multiplayer is to be implemented using SPI communication, letting two players with different microcontroller play against each other. The game is over when all boats belonging to one of the players are sunk, at which point the winner should be declared and the option to play again is to be displayed on the screen.
Due to outer circumstances the above specifications regarding the game modes was not followed as a sigleplayer option in addition to the multiplayer option was implemented in the final product. Unfortunately the multiplayer part was never properly tested, although there is no reason to believe that presented with an identical chip as the on used in this project it wouldn't work as expected. A bigger emphasis was therefore put on the singleplayer part which allows the player go head to head against an AI with several different difficulty levels. To read more about this you are refered to the Software and Execution sections of the report.

2 Hardware

- ATMEL ATmega16 microprocessor
- GDM12864C 128x64 Graphics Display
- 16 digit keyboard
- 54C922 16 key encoder
- Reset button

The base of the system is the ATmega16 microcontroller. Connected to this is the display, the key encoder and a reset button to be able to reset the system. A 16 digit keyboard is connected to the key encoder which when a button is pressed reads the signals from the keyboard and translates them to a four bit number that equals the value of the button that was pressed. As the key encoder is connected directly to the ATmega16 the output enable pin on the encoder

is grounded. The display is also connected directly to the ATmega16 and is controlled by issuing a set of instructions that sets the internal RAM of the display to a certain value. The display then internally reads from this RAM as it displays content. This simplifies the process by not having to constantly send instructions to the display, instead signals are only sent when changing the content on the display. The display consists of two halves that are written to one at a time. Which half is currently active is chosen with the two signals CS0 and CS1.
The full schematics of the system is seen below.

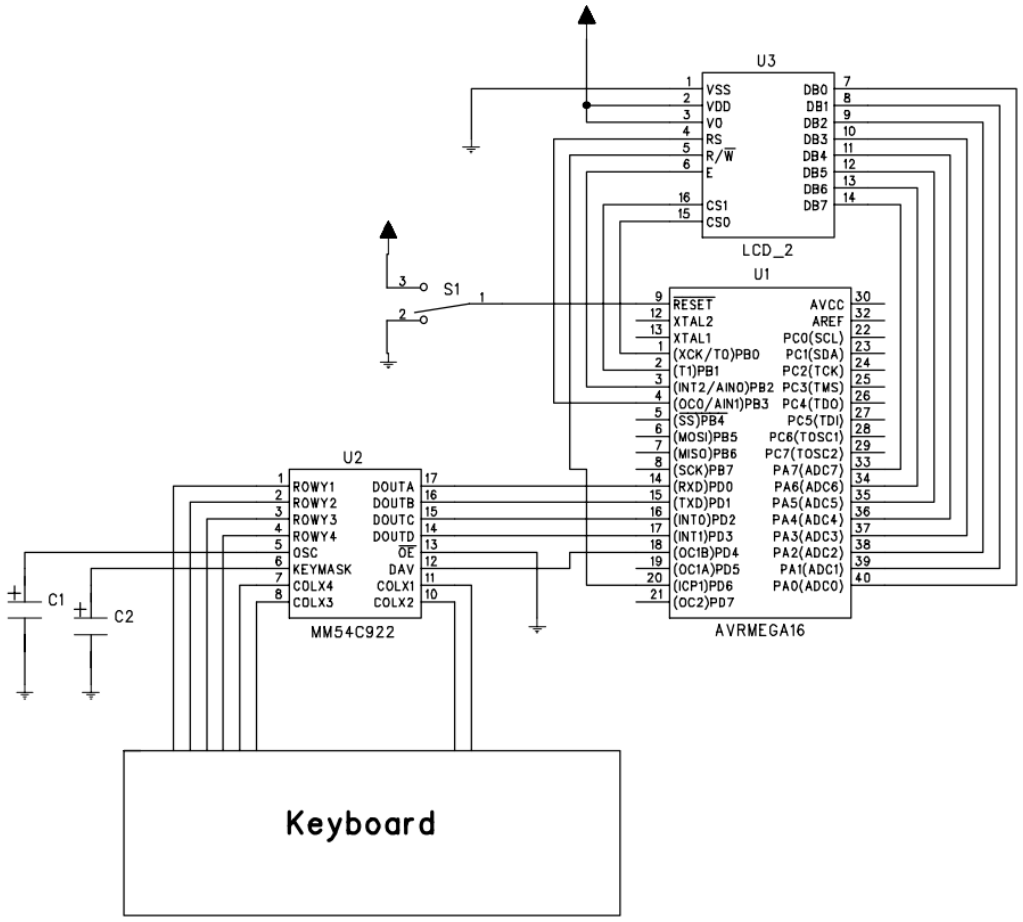

Figure 2.0.1: Full schematics

3 Software

The game code is written in C and can be split up into several different main blocks that perform different tasks. The functionalities of these blocks are briefly described below and to view them in their entirety the reader is referred to the source code.

3.1 Battleship

Contains the main loop that runs through the entire program. When the program runs the player is greeted with a welcome message and 3 different alternatives (1. singleplayer, 2. multiplayer, 3. highscores). The program then waits for user input before it executes the users choice. When the game is over the winner of the match is displayed on the display and if a highscore is achieved the player are asked to fill in their initials. This highscore is then saved to the EEPROM so that if the power to the circuit is broken the highscores remain in the memory. Finally the greeting screen is displayed and the player can choose to continue playing.

3.2 Game

Controls how the actual game is played. It lets the player place their boats on the playing field and then depending on if it's single- or multiplayer executes the propriate actions, e.g if it's in singleplayer mode the AI randomly places the specified number of boats on the playing field. The game can then start and the player that gets to start guessing is randomized. After a move is made the turn goes over to the other player and so it goes until one players boats are all sunk and the game is over.

3.3 GameUtil

Contains a couple of useful functions that e.g. reads and writes to the EEPROM and creates random variables.

3.4 Display

Contains all the instructions that are used to communicate with the LCD display.

3.5 Draw

Contains all the data instructions that needs to be sent to the display for all the drawing and writing that is done during the game. Contains e.g. a text function that, given a string of maximum size 16 characters and a row number, writes that text to the screen on that row.

3.6 Keyboard

This is where all the communication with the 16 digit keyboard is done. When e.g. coordinates are given the program has to wait for two inputs, one digit that is A-F and one that is 0-9, if these conditions are not met the program will wait for other inputs until they are. These coordinates are then compressed into one byte using concatenation and returned from the function.

3.7 AI

When the game is in singleplayer mode the AI is activated and makes the opposing players guesses. At the hardest difficulty the AI is able to guess around a hit appropriately and randomly guesses in a grid fashion making sure that unnecessery guesses are never made.

4 Theory

4.1 SPI -Serial Peripheral Interface

SPI is a communication protocol that operates in master/slave mode where all the communication is full duplex and needs to be initiated by the master. The protocol need 4 logical signals ot work:

- SS – slave select that is active low.
- SCLK – the clock signal that is controlled by the master.
- MOSI – master output and slave input.
- MISO – master input and slave output.

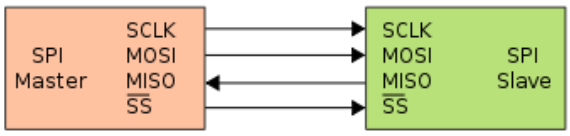

Figure 4.1.1: SPI with one Master and one Slave

The communication is initiated when the master loads a designated shit register with information. The master then shifts out all the bits stored in the register at the same time as it shifts in the bits located in the slaves designated register.

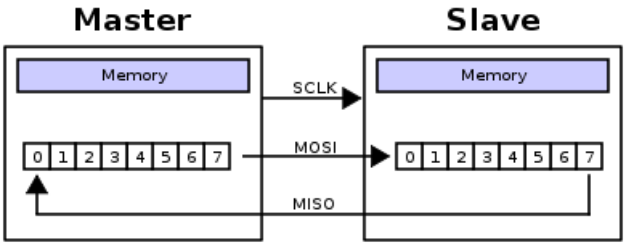

Figure 4.1.2: SPI transfer example

5 Execution

As described in earlier parts of this report the initial idea was to put emphasis on the multiplayer component of the game, i.e. being able to communicate with a similar system through SPI and this way allow two human players to play against each other. It was also initially believed that the difficulties of the project would lie in the hardware part of the project, i.e. the assembly of the different components. The original program to be run on the system was therefore quite restricted and contained only the essential functionalities. It was quickly discovered however that the assembly of the components was not as difficult as expected as this part of the project was finished in just a few days. The decision was then made to enhance the software part of the project to involve a singleplayer component as well in the form of an AI that a human player could play against on a single machine. Before this could be implemented however, software had to be written that could interpret signals from the keyboard and actually display content on the display. The keyboard software was the first to be implemented as this was imagined to be the easiest, something that later turned out to be true. Once this was in place focus could be shifted to the display. This would turn out to be a bit of a harder problem to solve. The documentation provided all instructions needed to control the display RAM, so implementing this part was done fairly easily. The difficulty lay in the synchronisation of the signals so that the correct information was displayed on the intended position of the screen. This was not as thoroughly described in the documentation and had to be figured out by trial and error and was not fully understood and implemented until the very end of the project.

Once the display was working, be it with a few glitches, functions for drawing actual symbols, e.g. boats, numbers and squares, had to be implemented. The playing field consisted of 6 x 10 squares with letters A-F on the left hand side and numbers 0-9 below, used in the game as coordinates. Boats were realized as circles, a miss as an X and when a boat was hit it turned into a circle with an X in it. Numbers were implemented by sending a number to a function that then drew this number on the display and letters were implemented in a similar manner. These functions were sufficient for drawing an entire playing field

spanning both halves of the display. To display text however a separate function had to be implemented, as it would have been highly inefficient to manually call upon the function for drawing individual letters for every letter in the text. A function was designed that could take a string and the row where the text should be displayed as input and display this string on the display, centred over the two halves. With this completed the next thing to implement would be the AI of the game.

Figure 5.0.3: Start menu

Figure 5.0.4: The playing field

The AI would have to be able to do a number of things in order to play Battleships. It would at first have to be able to place out boats at random for the player to find. It would then have to be able to guess the location of the player's boats and if a boat is found place the next guess according to its previous findings. The first thing to be built was the random placement of boats. The difficulties here lay in that all parts of the boat has to be in a line and that no part of the boat should be placed outside the playing field. This was implemented fairly quickly, but upon testing quite a big flaw was discovered. It turned out that the pseudorandom generator of the ATmega16 is not random

at all and always generates the same numbers upon every start up. This meant that the boats of the AI would always be placed in the same position. This was solved by writing a value to the EEPROM of the ATmega16, a register that saves values even when the power is removed, that was read and incremented at every start up. Using this value made it possible to build a random generator that gave new values every time the system was turned on.

The next issue to deal with was how to make the AI guess as a human player would, namely if a boat is hit the next guess is placed around this hit and if two hits are made and the boat is not sunk the next guess is placed in line with the other two hits. Several versions of the AI had to be implemented before this feature could be realized. The first one just placed random guesses and only checked if a previous guess had been made on a coordinate before placing a guess there. The next version knew if a boat was hit and could guess near this hit, but it did not know if the boat in question was already sunk and it could not recognize a boat consisting of three dots. The version after this knew when a boat was sunk and did not guess near this one again and yet another version later the ability to guess correctly on a boat of size three was finally completed. A final addition to the AI was made before completion, the ability to never place two guesses next to eachother, something that greatly reduces the number of guesses but not restricting the ability to find all boats.

As time was not running out as the AI was completed the decision was made to implement a function that made it possible to save and display highscores. Score is measured in the total number of guesses before winning and to not have these scores disappear every time the system is shut down they have to be written to the EEPROM. A function to read and write an array from/to the EEPROM was constructed and later functions that added scores and initials to this array were also built. Initials are entered by the player after winning a game with a score that puts him/her on the highscore list. Initials consists of three letters that are read from the keyboard. Highscores can be displayed by choosing this alternative in the start up menu.

As the singleplayer component of the system was now fully implemented and functional, work was started on multiplayer and communication with SPI. The game itself was not hard to implement as most of the components already existed as part of the singleplayer game. Instead of letting the AI place a guess the system would instead wait for input from the SPI registers and when the player was to guess his guesses were sent over SPI instead of being put on the AI playing field. The challenge here lay on setting up the SPI protocol correctly and writing the necessary functions to use this protocol. The protocol turned out to be very simple and functions that used this were constructed, but as the second system that was to be the opponent was not completed at this time none of the SPI functions could be tested and this part of the project was put on ice.

6 Results

The final product is a Battleships game with a fully functional singleplayer mode with 3 difficulty levels on the opponent. The multiplayer part is implemented but never truly tested. The communication between the ATmega16 and the display works flawlessly and the game is easy to understand for the player since it's easy to write text on the display. In singleplayer mode there are 3 difficulties on the AI, with the easiest being the computer guessing only randomly and the hardest, with the same amount of information as a human player would have, not making any unnecessery guesses at all. After the game the number of guesses it took for the winner is checked against a highscore table located on the controllers EEPROM. If the result belong in the highscores the table is updated accordingly and the player is asked to enter their initials. This table can then be viewed at the starting screen.

7 Discussion

7.1 Evaluation of project

In general the project was a success. We achieved most of the goals we initially set up, with the exception of the multiplayer part. In the beginning our idea of the difficulties were a bit off, as we thought that the assembly of the hardware would be the most time-consuming. Before long however we understood the scale of the project and could modify our initial specifications to match this. We managed to spread out the workload rather well which resulted in the project not being a burden but rather something very fun and educational. We also managed to include small milestones along the way which gave us a steady feeling of accomplishment, something that made it a lot easier to work with.

7.2 Improvements

7.2.1 General

The most obvious improvement to be done is to actually test the multiplayer part, making sure that it's functional thereby adding another dimension to the game. Another improvement to be made would be to have more printouts on the screen to aid the player in their playing, telling them e.g. how many boats they should place and the general rules of the game. The reason this hasn't been done is that so far the only ones playing the game has been the creators of the project and thus such information felt redundant at the time. However if the game were to be launched side by side with the new PS4 the addition of such informational screens is highly recommended.

7.2.2 Security

Although the multiplayer part of the game isn't functional one can't keep from wondering how this could be implemented in the best way and most important of all keep malicious players from cheating. First of all it's important not to share the correct boat placement to your opponent by sending them your boat matrix. If this is done then don't be sursprised if your opponent magically always finds your boat on their first try. The solution to this is that instead of sending the correct matrix, you send your guess coordinate and ask them if it's a hit or a miss after each guess. This is how the multiplayer aspect of the game in the project is implemented. However this will lead to the problem that the opponent can always lie about the guesses being misses when actually a boat is hit. This is where there is room for improvements. A solution to this would be to have both players exchange their playing field matrices with each other after the boats is placed, not in plaintext but hashed so that you cannot retrieve the correct boat placement from the hashvalue. When the game is over the two players exchange their matrices in plaintext with each other, these matrices are then hashed and compared to the original hashvalue to see if the final matrix is the same as the one they started with and thus making sure no cheating has been done.

8 References

ATMEL ATmega16 Datasheet
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf

GDM12864C Graphic Display Datasheet
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Display/GDM12864H.pdf

54C922 Key encoder Datasheet
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Periphery/Other/MM54C922.pdf

8.1 Figures

Figure 4.1.1
http://en.wikipedia.org/wiki/File:SPI_single_slave.svg

Figure 4.1.2
http://en.wikipedia.org/wiki/File:SPI_8-bit_circular_transfer.svg