EITF40
IR-Ball

Erik Hogeman - ada09eho@student.lu.se
Fredrik Johnsson - dt08fj8@student.lth.se

March 5, 2013

Contents

1 Abstract

The goal of this project was to implement a version of the classic game DX-Ball, controlled with an IR remote control. The game should consist of a ball, a paddle and some square, floating blocks that should disappear when the ball hits them. The hardware used is an ATmega16 microprocessor, an IR decoder, an IR detector and an LCD display using some LCD-controllers. A switch and a couple of resistors were also used, and of course a remote control for actually playing the game.

When the game starts, it immediately starts polling for new commands from the remote while also updating the ball position and writing to the display. Whenever a command is found, the paddle is also updated. The end result turned out to work almost according to the initial requirements, with the exception of the overall flow of the game which we would have wanted to be a bit faster. After doing all kinds of optimizations, we succeeded to make the game playable, but there is still some noticeable delay while playing, and a few minor bugs that show up from time to time. The overall result has to be considered a sucess though.

2 Introduction

The project described in this report was done as a part of the course EITF40, digital and analogue projects. The goal was to build and program a small hardware construction of our own choice. We chose to build a version of the classic game DX Ball controlled by IR light using a remote control.

This report will describe the initial requierements and how the actual result turned out in the end. The different parts used and how they are communicating with each other, and a general description of the code architecture will also be presented.

3 Requirements

We decided to work from the following requirements:

- On the receiver side there should be a microprocessor which is connected to an IR-receiver and a display.
- The IR-receiver should decode the signal from a remote-control and send the signal to the microprocessor.
- The microprocessor should show relevant information, handle user-input and control the game.
- The game should be something similar to a game called DX-ball. The player controls a paddle at the bottom of the screen and bounces a ball between the paddle and different blocks above which disappears when they are hit by the ball. The game ends when all the blocks are gone.
- If time allows it, we would build our own remote controller and make a more complex game.

4 Components

The core of the system is of course the microprocessor, an AVR Atmega16. This 8-bit microcontroller has 16kB of flash memory and an internal clock of 8 MHz. This makes the project more challenging (and more instructive) since performance and resources are much more limited compared to an average PC. The microcontroller handles the received commands form the remote, the game logics and updates the display depending on what happens in the game. All source-code was written in C, and we used AVR Studio 4 for writing the code and for writing to the memory of the microcontroller. To connect to the microcontroller from the PC we used the JTAG-interface of the microcontroller.

The display module contains a two-color LCD-display with a resolution of 128x64 pixels and three IC-controllers which are used for storing and updating the data sent to the display. There are also two IC-modules for the backlight and power supply. This display-module came pre-built by the institution.

To be able to receive commands from the remote-control we have used a remote-control-detector, ELRIM-8608S, which detects IR-signals and amplifies them. To decode the signal, an IC-module called SAA3049A is used. This module translates the raw IR-signal into two binary integers that represents a command and an address. The command represent which button that has been pressed on the remote control and the address is used for keeping track of different remotes for different devices. For example a TV might use a different address than a DVD-player and they will therefor not interfere with eachother since they only listen to their own address. In our system we disregard the adress part of the data.

5 Schematics and Communication

For an overview of the entire construction, see figure 1. A description of the overall data communication when receiving a command from the remote will now

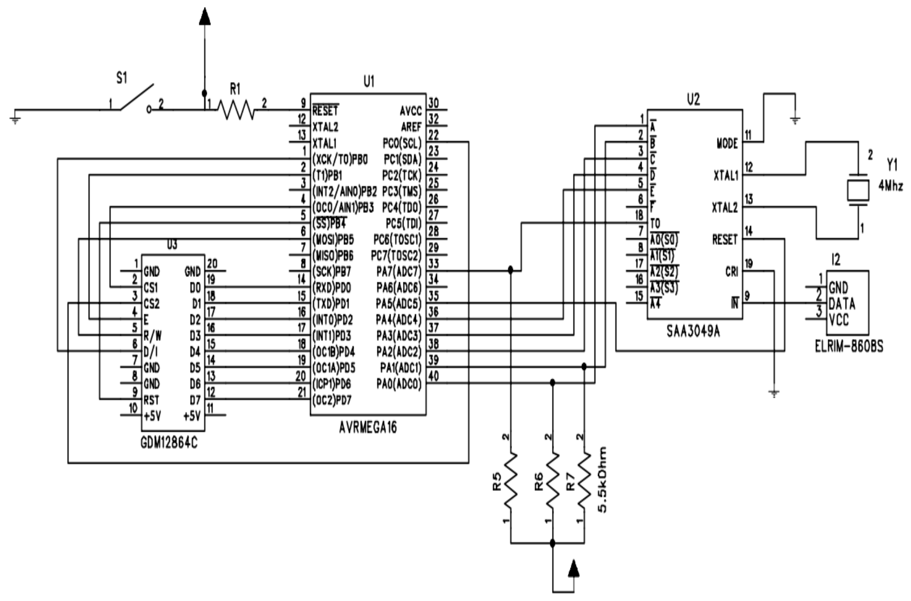

Figure 1: Schematics of the construction

follow below. The first step is when the IR receiver detects the light sequences from the remote controller, and then forwards this sequence to the IR decoder on a special input pin. The IR decoder will depending on the mode selection and controller button pressed interpret this sequence as some command. Different commands will output different values on six output ports. Every time a new complete command is interpreted a special toggle bit will also toggle, and can be read on a special output pin. These output pins are also connected to pullups (resistor to VCC, see the schematics) in order to keep them from floating.

Using these command bits and the toggle bit, the AVR can communicate with the IR decoder and receive commands from the remote control. Depending on the command received, the AVR will update it's internal status of the different objects in play and update the LCD display with the current status of the game. The display uses two different segments, and the AVR has to keep track of when to write to each segment. Different command bits can be used to control the display, for example to change to read or write mode, choose which pixels to write to or read the status register. There is also a special bit called the e-bit, which should be toggled from high to low to write the current input to the display. Whenever the display should be updated, its internal memory is first updated, and then a special command is sent to it to make it display its current memory contents.

6 Code architecture

In the most basic sense, the code consists of a main loop which keeps polling the toggle bit for new commands, checks how the ball and enemies should be updated, and then writes the new status to the display. A more detailed explanation follows below.

6.1 Main loop

The main loop initializes all the ports and different objects (ball, enemies and paddle), and then proceeds to loop forever, polling the toggle bit, updating the different objects by calling different functions and updates the display with the new current status.

6.2 The Ball

The code for the ball is divided into two files, one c-file and one h-file containing some macros and function declarations. The struct representing the ball contains its current coordinates, its current angle and the direction its currently traveling. The ball files also contain a function for checking if a certain pixel is located inside the ball or not, which is used when updating the display. There is also a function which checks if the ball is currently touching an enemy or the paddle, and if that's the case bounces the ball by changing the direction and angle.

To represent the coordinates of the ball, we chose to use 16 bit integers in fixed point representation, since the processor doesn't have an FPU. We also hard coded in a couple of predefined angles instead of using sine and cosine functions. These numbers are then converted to integers before using them to write to the display.

6.3 The Paddle and enemies

The paddles and enemies work very similarly to the ball. They also contain coordinates (though not in fixed point) and functions for finding out if a pixel is located inside them or not. The enemies also contain a variable indicating if this particular enemy has been destroyed or is still alive.

6.4 The display handler

The display handler updates and writes to the display. It has a function for updating the display memory and one for displaying the current memory. In order to optimize for speed, the update memory function only writes to the pixels where something actually has changed recently, instead of writing to the whole 128x64 array every time. To be able to do this, some global variables are used indicating for example if the paddle has moved or if some enemy has been killed and so on. The last position of the ball and paddle are also saved, so that the earlier version can be overwritten.

7 Result

In the end we managed to implement a fully working game that was controlled using an ordinary IR remote control which is what we intended. We did have some issues with the performance of the display which was a large bottleneck. Despite out best efforts we were not able to optimize the performance to the desired degree but atleast we made it to a degree which made the game playable.

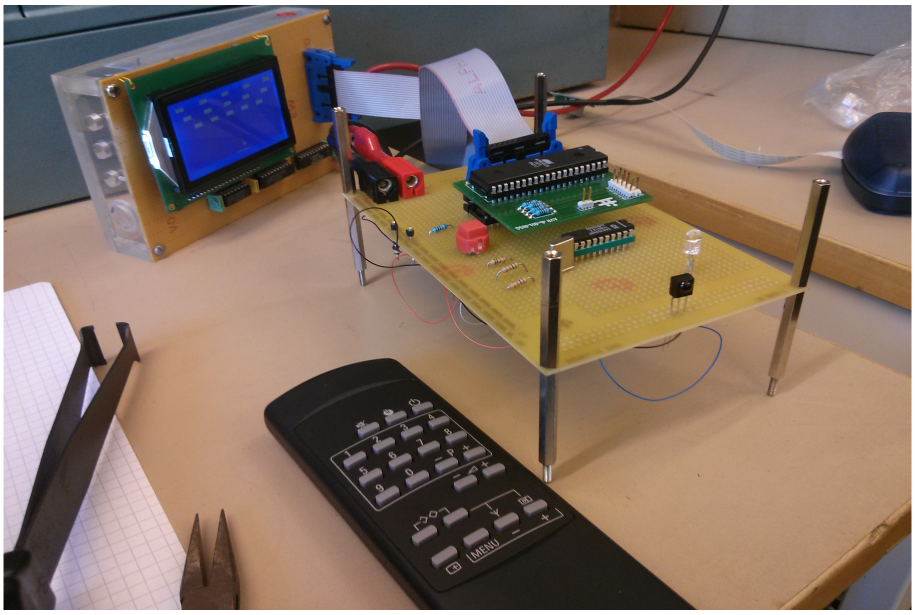

Figure 2: The complete construction

Because of the way we handle the data received by the remote control, the user cannot hold down the button to keep moving the paddle in one direction but instead have to repeatedly press the button. We had a solution for this problem but we didn't have enough time to implement it.

Overall the course was fun and interesting.