DTU Compute

DTU

Michael Sylvest Christensen s153712

Mustafa Sidiqi s153168

02321 Hardware/Software programmering E16

Contents

Description

This is a game, inspired by the well-known retro game, "Space Invaders". The game itself consists of a spaceship in the bottom of the screen that is only able to move to the left and right. This spaceship then has the ability to shoot a projectile, one at a time, that has the one purpose of destroying alien objects, that fly towards the spaceship. The player obviously controls the spaceship and must complete the given objective: Destroy all the alien objects and survive and by doing so, get a score that could possibly beat the current highscore of the game.

Requirement specification

The requirements of this project is to create and implement a game on the Zybo board, that uses IP-cores that allow the use of buttons, switches and LED's on the Zybo board. The buttons are used to move the spaceship right or left and shoot, the LED's are used as an indicator how much health is left and the switches are to be used as developer uses in the game (possibly immortality, speed of aliens and shots and insta-killing enemies).
The player (spaceship) moves, with the use of buttons, to the right or left and shoots where ever.
There is only one projectile at one time, so if a projectile has been fired, the player has to wait until the projectile either hits an enemy or gets out of bound (beyond screen border)
The enemies move only down and has a (random) start location on top of screen and a (random) square structure shape. These enemies don't shoot, BUT, if the enemies get below the bottom screen edge you lose a life and if an object hit's you, you lose all your remaining health (full or not).
The projectile moves only up and at a fixed rate. If the projectile hits an enemy, the enemy is destroyed and thus vanishing from the screen.

Progression tables

Needed components

	Expected time (days)	Actual time (days)	Importance (1-10)	Priority
Screen (VGA)	4	2	10	1
Game logic	7	8	10	2
Graphics	5	6	7	4
Buttons/switches/leds	1	1	3	3
Lyd	3	0	2	5

Risk analysis

What can go wrong	How likely it is to go wrong (1-5)	How serious are the consequences if it goes wrong	Overall risk
Using vsync on vga as timer interrupt fails.	4	We must use different approach to timing.	Game is still doable
Colors are different from expectations in graphics (RGB).	2	The game will have graphics in different colors, but the overall game will be unaffected (if the figures are still intact)	Minor annoyance.
Automatically moving of aliens.	1	If the aliens to be shot isn't moving, we must take a different approach to level completion.	Minor annoyance.
Automatically moving of projectile.	1	If the shot isn't moving the game will	

Design

Prerequisites

Hardware

- ZYBO board
- Micro-USB cable
- VGA monitor

Software

- Vivado Design Suite & Xilinx SDK 2016.2

Following figure is an overview of the block design we want to create in Vivado Design Suite 2016.2.

Hardware design

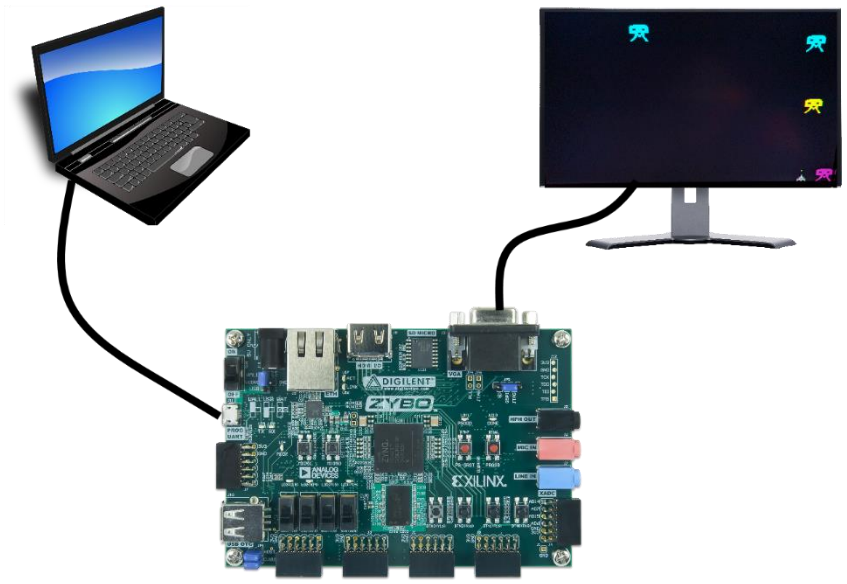
Before we design the hardware of this project, we need to figure out what components are needed. The base of the hardware design will always contain three obligatory hardware: A processing system core, a system reset and an AXI interconnect. (shown/described in implementation section below). After the main hardware is added, we need to think of what needs to be added for our game to function as intended. The game obviously needs to be visualized using a screen. For this to be done we need (in our instance) a VGA cable connected to the Zyboboard where the game is implemented. This functionality needs four additional hardware functionality:

- A core to get access a direct memory access between memory and AXI4-Stream video.
- A core that accepts a Xilinx vid_io input and outputs an independently customizable color depth, properly blanked RGB pixel bus to connect to a VGA DAC.
- A core that provides a bridge between video processing cores with AXI4-Stream interfaces and a video output.
- A Video Timing Controller core allowing control of timing generation parameters for the screen VGA output.

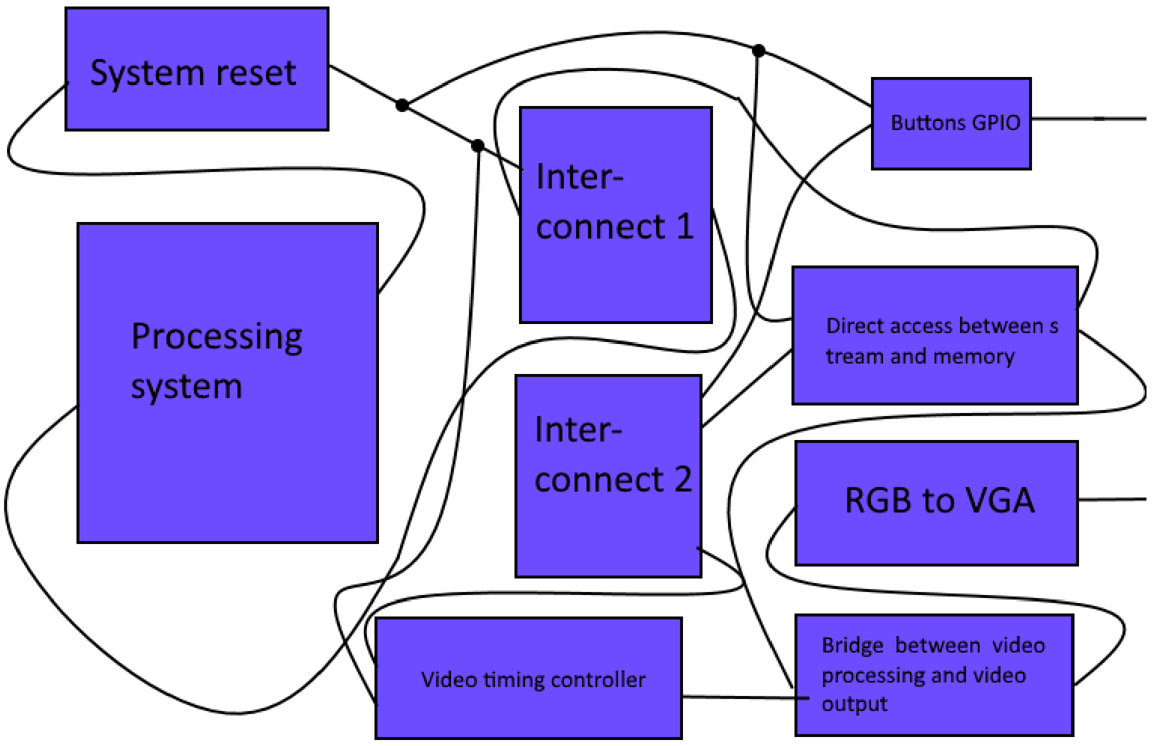
Now that we can get output to a screen to visualize our game, the next step is to design the components for our game to work as intended. The idea is to have the player in the bottom of the screen, that is able to move right or left along the bottom. For this functionality to work, we need some kind of movement controlling hardware. We agreed to implement buttons on the Zybo board, and to make this work, we need a AXI-GPIO hardware that accepts an input from the buttons on the board.

We also want the player to be able to shoot, but since the zybo-board has four integrated buttons, this functionality is solved with the AXI-GPIO for the buttons, same as movement.

At a larger overview, we just need a Zybo board, a VGA screen and our PC to upload:



And inside the zybo board we have the deeper hardware design:



(If there is a black dot on a wire, the wire is split).

Software design

When it comes to our game-logic, we write in C-code language. We use the Xillinx Vivado SDK 2016.2 to write and run the .c files on the Zybo board. First of we need to include all the necessary header files to get the c code to connected to the hardware implementation. Then we need to initialize the different hardware. When that is done we need to implement the game logic and use the initialized hardware to create the game.
For the game, we need an infinite while-loop to keep the program from ending. In between this this while loop the code block will be the game itself. We need functions that draw the different objects that are to be visualized on the screen, functions to dictate what happens during specific situations that will occur in the game. To start with we need an object to represent the player. Here we want to start with just producing a square with the colour green. To do this drawing of objects, we need to specify some coordinates for each object. For this to work, a struct needs to be created that will represent stats for each object to be created. This struct will contain:

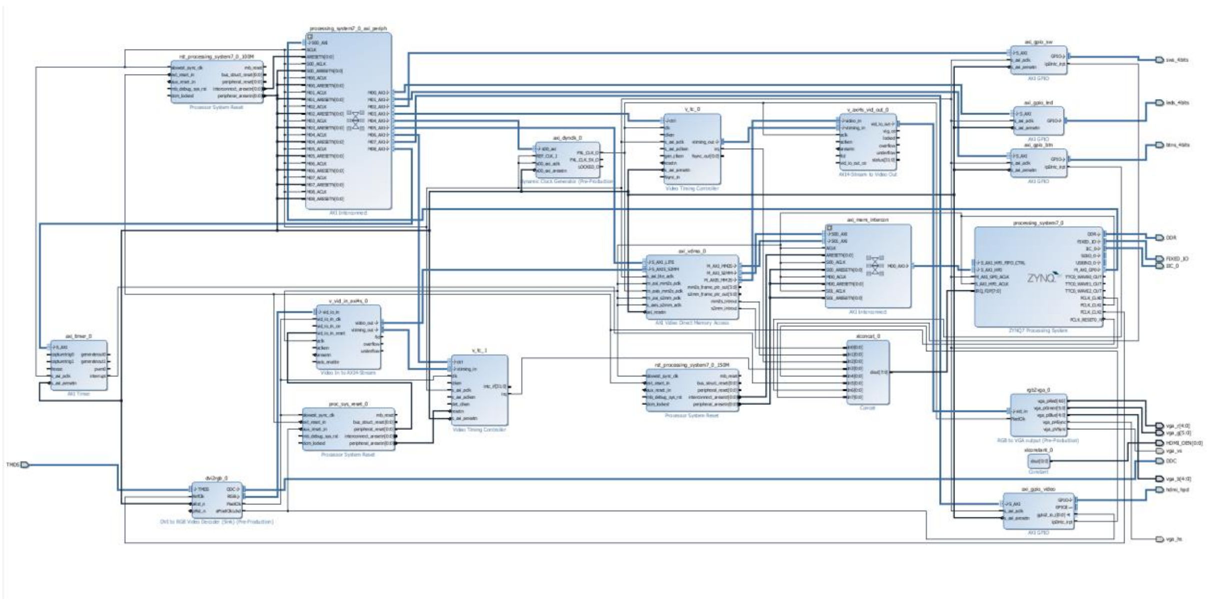
- X- and Y-axis position
- X- and Y-axis size
- X- and Y-axis speed
- Lives
-

The defining of coordinates for each object makes it possible to move the different objects just by changes this coordinates. If the player hits an enemy, the enemy must "die" and disappear from the screen to visualize the death of the shot enemy alien. Same goes with the player; when the player gets down to 0 lives we want to visualize that the game is over. This is why the struct contains lives. The different objects will be initialized by a function going through the created objects, giving them values. An array of this struct will be created for the enemies.
To move player and shoot, a switch statement will be made with btn_value as parameter. This switch statement will have a case for each button, that will control the actions made by each button.

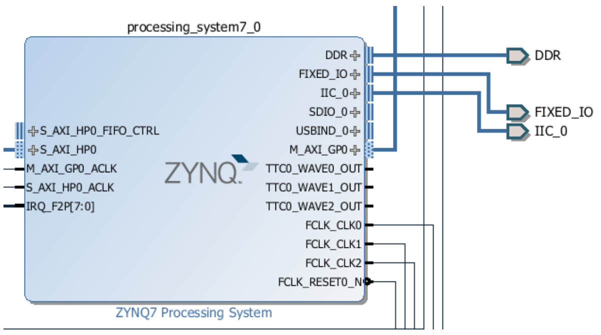
Implementation

The implementation is done by building further on a project found on Digilent Inc. website called VGA DEMO. This project contains the necessary IP-cores and software instantiations of hardware to be able to send data to a VGA screen. ONLY these mentioned functionalities is used by the project provided, rest is made completely by group members.

Hardware implementation:

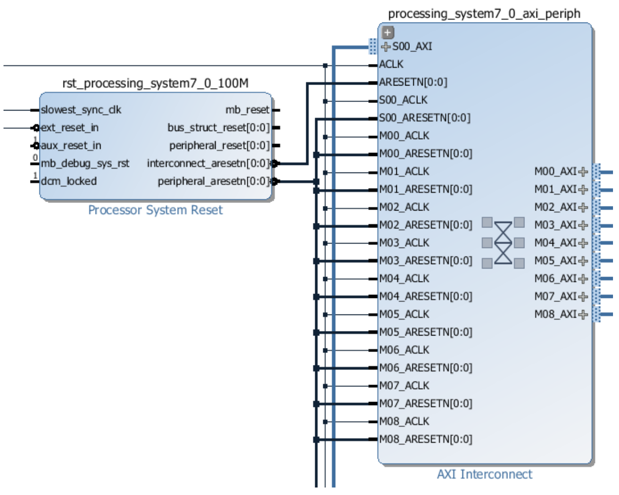


We have the obligatory IP-cores ZYNQ7 Processing System, AXI Interconnect and a System Reset. Interconnect connects the different in- and outputs with the processing system, which is the core of the design that keeps everything going. The reset is, as the name suggests, a reset to the systems different IP-cores.



This is, as said, the core of the design. It connects to the RAM (DDR), connects a clock and a reset to the rest of the system design takes an interrupt source from the concat IP-core (shown later).

Here we have the reset and the interconnect IP-core. This interconnect, connects a clock and a reset to the AXI_GPIOs that is described below.

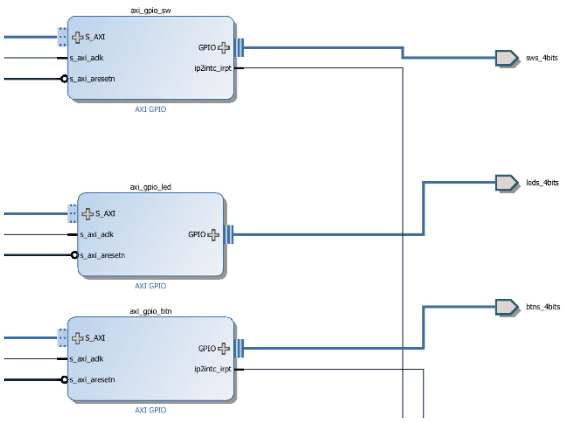


"The Xilinx® LogiCORE™ IP AXI Interconnect core connects one or more AXI memory-mapped master devices to one or more memory-mapped slave devices."
- Pg059-axi-interconnect.pdf

"The Xilinx LogiCORE™ IP Processor System Reset Module core provides customized resets for an entire processor system, including the processor, the interconnect and peripherals. The core allows customers to tailor their designs to suit their application by setting

certain parameters to enable/disable features. The features of designs are discussed in the Design Parameters section."
- pg164-proc-sys-reset.pdf

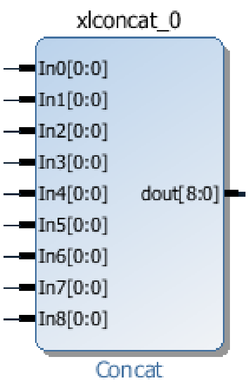
The added cores after this, is what makes this design to be able to do what was intended with the design. Here we implement three AXI_GPIOs; he buttons, LEDs and the switches.



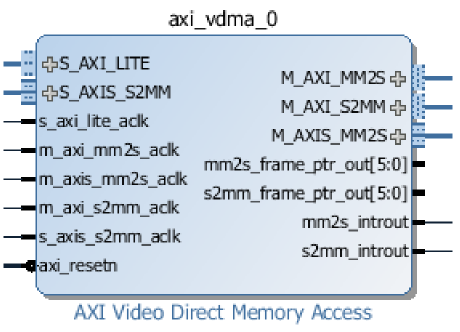
"The Xilinx® LogiCORE™ IP AXI General Purpose Input/Output (GPIO) core provides a general purpose input/output interface to the AXI interface. This 32-bit soft Intellectual Property (IP) core is designed to interface with the AXI4-Lite interface."

- Pg144-axi-gpio.pdf

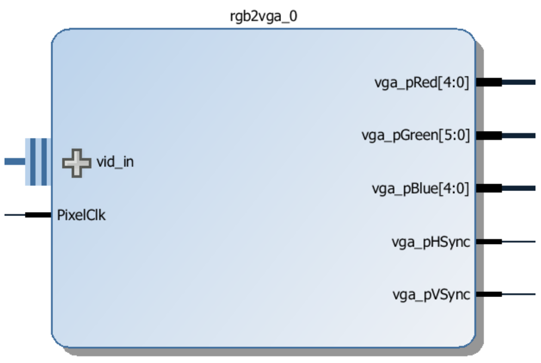
As you can see on the picture to the right, we have a connected interrupt on both the switches and the buttons. These are NOT implemented at the current moment of hand-in, but can be implemented from software without changes to hardware.



Here's the xlconcat_0 IP-core. This the what connects interrupt sources from other IP-cores and sends this information as output to the processing system.

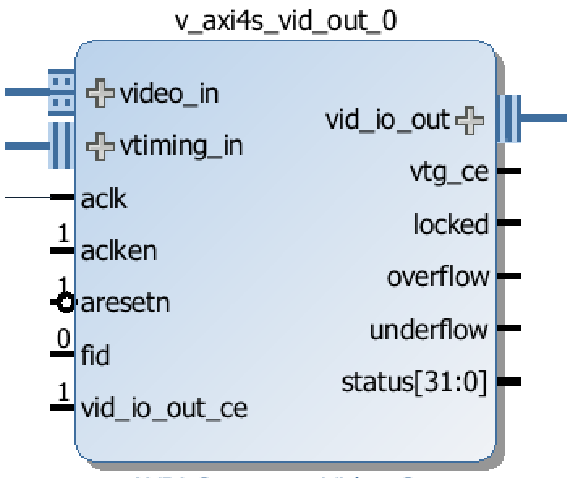


"The Xilinx® LogiCORE™ IP AXI VDMA core is a soft IP core. It provides high-bandwidth direct memory access between memory and AXI4-Stream video type target peripherals including peripherals"
- pg020_axi_vdma.pdf



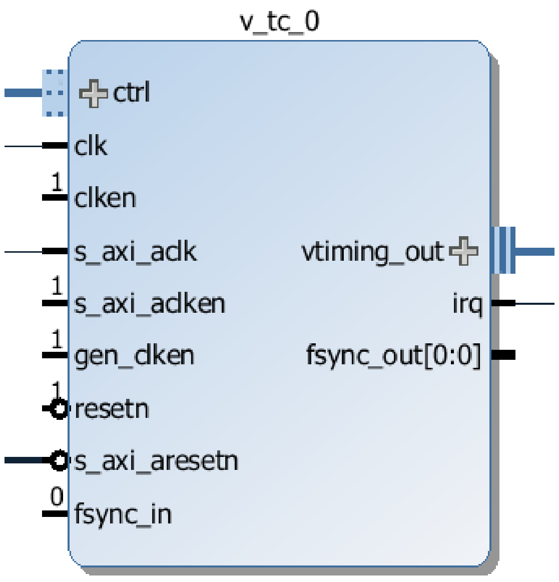
RGB to VGA output (Pre-Production)

"This user guide describes the Digilent RGB-to-VGA Intellectual Property. This IP accepts a Xilinx vid_io input and outputs an independently customizable color depth, properly blanked RGB pixel bus to connect to a VGA DAC. On Digilent boards the outputs directly connect to the on-board resistor ladder that serves as VGA DAC. The supported color depth varies between boards and the IP should be configured to match it."
- RGB2VGA_v1_0.pdf



AXI4-Stream to Video Out

"The Xilinx LogiCORE™ IP AXI4-Stream to Video Out core is designed to interface from the AXI4-Stream interface implementing a Video Protocol to a video source (parallel video data, video syncs, and blanks).This core works with the Xilinx Video Timing Controller (VTC) core. This core provides a bridge between video processing cores with AXI4-Stream interfaces and a video output."
- pg044_v_axis_vid_out.pdf



Video Timing Controller

"The Xilinx LogiCORE™ IP Video Timing Controller core is a general purpose video timing generator and detector. The core is highly programmable through a comprehensive register set allowing control of various timing generation parameters. This programmability is coupled with a comprehensive set of interrupt bits which provides easy integration into a processor system for in-system control of the block in real-time. The Video Timing Controller is provided with an optional AXI4-Lite compliant interface."
- pg016_v_tc.pdf

The documentation of these cores are listed in the appendix which is quoted is listed in the appendix.

Software implementation:

The following is a quick overview of our software implementation.

The main component in our project is the graphics, and to be able to display anything we have used the VGA output on the Zybo board. The functionality is based on the use of framebuffers, where we draw on one frame at a time.

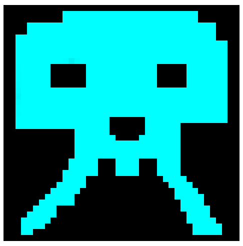
To draw on the screen via VGA cable, we use a framebuffer that contains the three main colours that can produce any other colour; red, green, blue, hence the use of 'RGB'. But using only one framebuffer kawill create a flickering frame on the screen, as the code draws everything on the same frame and then refresh it. That is why make use of two framebuffers instead where the entire frame is printed on one of the framebuffers and changes will be drawn on the other. Then the printed frame is shifted, then changes made on the other and so on. This gets rid of the flickering and creates relatively fluid visuals on the screen.
To use two framebuffers, we make an array of pointers the array of framebuffers. And then shift every time one of the framebuffer has been shown.

We have used PPM file format to draw our main elements of the game. Where we firstly start by drawing the character or element in a jpg format, with the wanted pixel ratio. Thereon the JPG file is converted to PPM file, so we can extract the RBG values with a simple C program. The c program is pasted in appendix.

The way we are drawing our main elements is by going through the screen pixel one by one, and assign a value for each pixel. We assign a RBG value there is an element to be drawn in that pixel, otherwise we don't assign anything there. Since our background for our elements is black, and is black if we don't write any RBG value there.

The RBG values are stored in an array, which we access with our c-function. The function runs through the array, and assign a value for each of the RBG colours.

Drawing of enemies:



We have one standard drawing of enemy, which is 40 x 40. We have saved the values in an array, and we access the values with a for loop. With this method we are able to draw the enemy. Since we have just one enemy with one colour, we decided to draw it differently to be able to change the colour. So by drawing the figure with just 2 colour out of the 3 RBG colours, we can have a combination of 3 different colours.

Statistics

We have chosen statistics as the subject in the CDIO part of this project, and have chosen to display stats about the gameplay, how the play is doing, number of enemies killed, accuracy and efficiently.

Statistics	
Shots	An integer that increments every time the player shoots (pressing button S (8)).
Point gained	An integer that increments with a constant every time an alien is killed.
Aliens killed	An integer that increments for every alien killed.
Lives	An integer that decreases for every time an enemy passes the player, or set to zero if the player gets hit by and enemy.
Accuracy	A float value that changes in correlation to how many aliens that are hit per Projectile (laser).
Enemies spawned	An integer that increments every time an enemy spawns.

Test Summary

Test case	Expected result	Actual result	Remarks
Enemies movement	Enemies move either straight down or in a zigzag patern.	The enemies xPos (x-axis position) and xSpeed (the speed of with the xPos changes) gets a random value (within the screen borders). This means the enemies move from side to side by pixels between 0 and 3. (0 means straight down movement).	The random generator function rand() needs to be seeded somehow. But for now it works as intended.
Moving player	The player can move from right to left using buttons.	The player can move from right to left using buttons. NO interrupt is used.	The code that allows moving using buttons is in a while loop with nested switch statement. Because there is no interrupt, the use of buttons become

			quite an annoyance, but still doable.
Killing enemies	If the enemies are hit by the fired projectile from the player, the enemy hit, "dies" and vanishes from screen. (maybe hit more if lives is implemented).	The enemies hit disappear from screen as intended.	For future work, a picture of an explotion could be used as visualisation instead of the alien just vanishing. But this works too.
Graphics	We can use a picture, either drawn by ourself or found on google pictures, as our enemies and our player.	We were able to paint our own pictures, save them as a picture format and convert this picture format to a .ppm file that could be introduced to the code as an array of RGB.	This made the game feel much more like a game instead of just squares flying around.
Statistics	The statistics of the game is visualized in the side of the screen during gameplay.	Unfortunately we didn't have time to implement the visualization of statistics on the screen. Instead the statistics is shown in the console window on the uploading PC with the use of PutTy.	This is not a hard implementation, but still a very time consuming feature to implement.

Discussion

There where several considerations along the way of getting this project together.
To begin with the idea of the game was to make a text based, ASCII graphics adventure game. Within two days we decided to change our game to a space-invaders like game instead. This was quite a transformation to make, but it was agreed that the text based game we intended was too boring and low level.

The first version of alien enemies were square shapes that had random sizes. This was before we had graphics implemented. The squared alien enemies were boring and ugly to look at and overall just didn't feel right. We got a self-drawn alien monster image instead of the squares and suddenly it looked like a game worth playing.

The first version of the player was also NOT a .ppm picture file but instead a triangle. This was in itself okay and didn't discourage you to play the game, but an upgrade to an actual space-ship looking figure, did look very good.

The laser projectile never got it's own .ppm picture as visualisation. Instead we agreed in keeping just a small green rectangle as projectile. It COULD'VE been upgraded to a laser looking projectile or a green space fireball, but as it is so small this won't have as big an effect as the player and enemies.

It was discussed whether to give the player a rate of fire or a single shot per firing. It was however agreed very quickly to just give the player single shot per firing, as giving a rate of fire might've made the game too easy. The single shot works with the gameplay, so no regrets on that matter from the group members.

The amount of lives that the enemies has was discussed, again in correlation with the difficulty of the game. It was decided in the end that the enemies were to have three hit tolerance before dying and vanishing from the screen. But we are still in doubt whether this is a wise choice for the gameplay or not.

Conclusion:

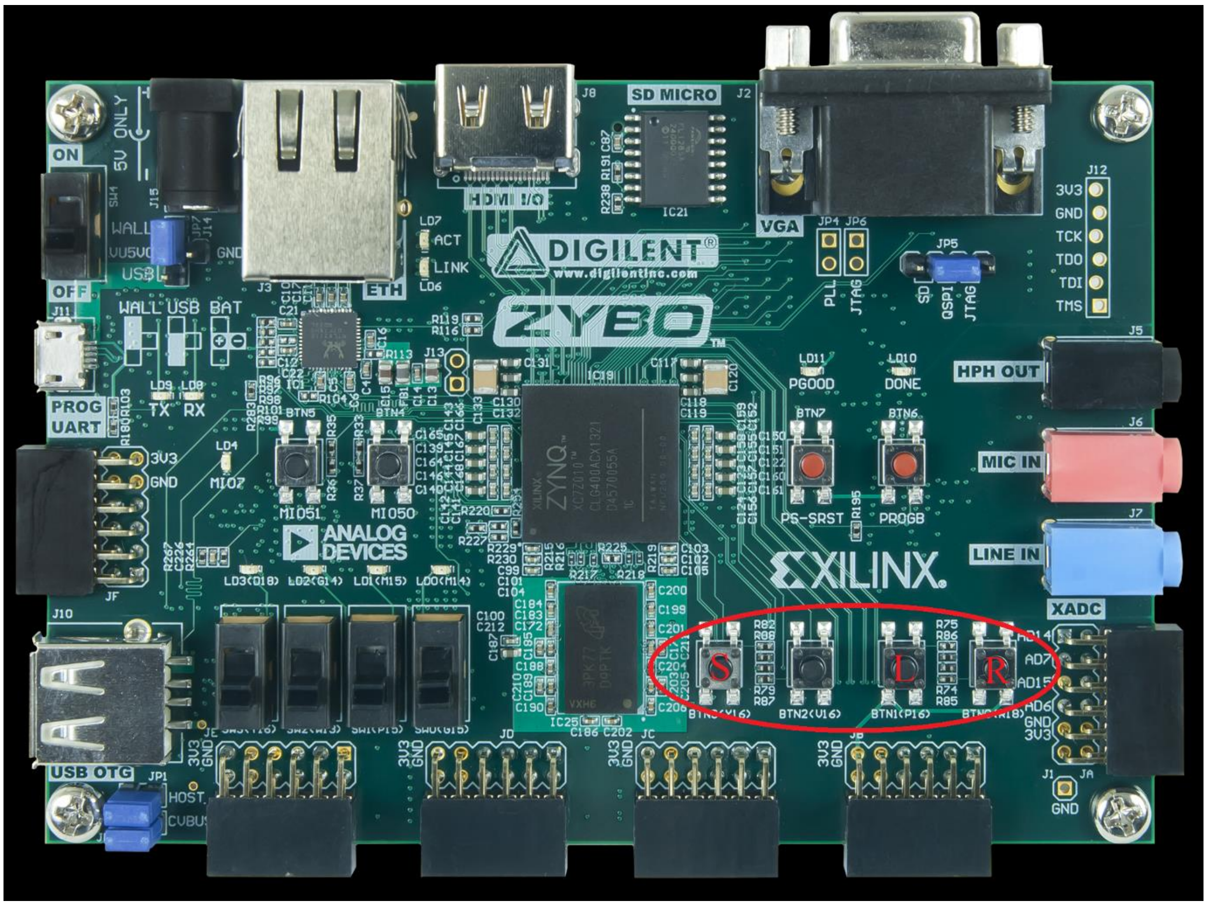
In this 3-weeks period we have worked in a group of 2 persons, where our goal has been to design and implement a hardware/software project. We have we have made the hardware in Xilinx Vivado, while the software part is done in SDK. Through this project we have design our hardware and written code to be able to use it. Our project is based on developing a, space invaders inspired, game, where the objective is to survive the invasion of scary monsters.

We have managed a functional game where we have fulfilled our minimum implementation goals, and even achieved more than our aim. Our game is coded in c and runs on the Zybo board with the Vivado created hardware. We are happy with our results in this project, all though there was a few hick ups. But at the end we have managed to develop a game with double framebuffer, smooth graphics output on VGA which is controlled by input from user. We have mixed in statistic as the CDIO-subject. Overall we are happy and satisfied with our end results.

Appendix

User manual

To play this game, use the buttons on the zybo board as follows:



The buttons has the following functions:
S, Shoot
L, move left
R, move right

Bug report

- There is a very small chance that an alien is spawning beyond screen borders and crashes the game. Current cause of crash is unknown.
- The ranbow flickering alien enemies are not intentionally, but we consider it as a slightly annoying feature of the game.

Extra diagrams

Contribution to project work

- https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-hdmi-demo/start
- kuno Heltborg – A Big Thanks

Code to convert .PMM to RBG values:

```
#include <stdio.h>
#include <stdlib.h>
#include "PPM.h"


int main(void) {
   PPMImage *image;
   image = readPPM("game_over.ppm");
   int i=0;
   for(i=0;i < 4800;i++){
      printf("%u,",image->data[i].red);
      printf("%u,",image->data[i].blue);
      printf("%u,",image->data[i].green);
   }


   return (0);
}
```