2015

Space Impact

Summary: This report presents the Space Impact game designed for the fulfillment of engineering job in C-course at AGSTU. It describes the SW and hardware architecture used to implement the game.

Dinsefa M. mustefa@kth.se

TEIS AB

1/4/2015

Table of Contents

1 Introduction 3
2 Description of construction 3
2.1 Software 3
2.1.1 SW architecture 3
2.1.2 Files and Function 4
2.1.3 Code size 5
2.2 Hardware 5
2.2.1 Hardware architecture 5
2.2.2 Memory map 6
2.2.3 Port Descriptions 6
3 Test protocol 7
4 Validation 8
5 Conclusion and future improvements 8
Appendix 9
C-code for the entire project 9

1 Introduction

The game Space Impact (see Figure 1) is constructed and is played using three push buttons with each of them have a respective function up, down, and one to launch the weapon.

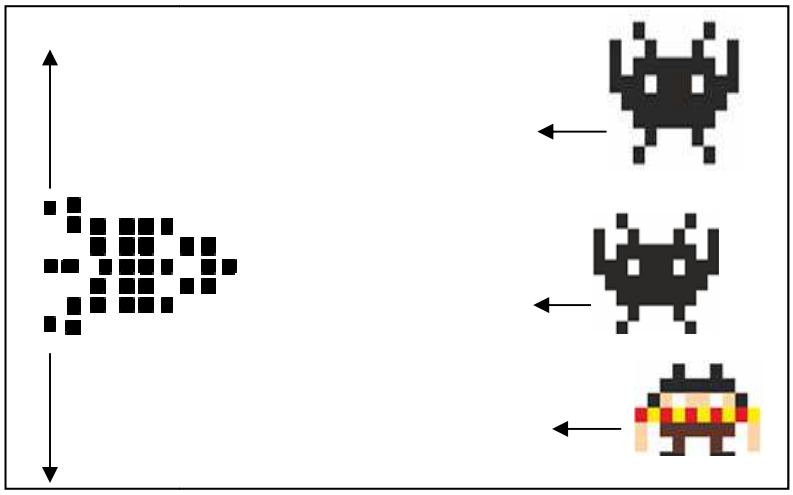

Figure 1: Block diagram of the game that shall be constructed.

The objective of the game is to use the three pushbuttons to eliminate various space objects that comes flying straight toward the left. The launcher that fires the weapon is located in the left part of the screen while space objects are generating from the right and then move to the left. The player need to move up and down the launcher using two of the pushbuttons so that the launcher can be straight with the coming object and then fires the weapon using the third pushbutton to kill or eliminate the space object.

2 Description of construction

2.1 Software

2.1.1 SW architecture

The software constructed based on the state machine depicted in Figure 2 to implement the space impact game.

Upon start-up, it clears the screen and draws the launcher and space objects on the screen before it moves to scanning KEY ports. It leads to; shooting state if a player pressed KEY_0 on Altera board and the launcher fire weapon targeting on a space object. If weapon hits a space object, then the object gets eliminated and a new space object is generated at a random location in far right end of the screen.

The launcher moves up if KEY_1 is pushed and moves down if KEY_2 is pressed. If none of the three KEYs are pressed, then we move to moving space objects state to move the space objects towards left side of the screen.

On the check game status state, it checks if any of the game specification are met, i.e., it checks number of space objects killed by the player and those objects that made it to the left without getting killed. If the player eliminates 15 space objects before 4 space objects make it to the left end, then the first level

of the game is completed and it continues to the second level. On the second level the number of space objects on the screen increases and hence the challenge increases. The player should eliminate 15 objects to complete the second level as well. If he/she does so, then the player wins and the game ends. If 4 space objects make it to the left then the game ends which applies in both first and second level of the game, but it starts counting from zero in both levels.

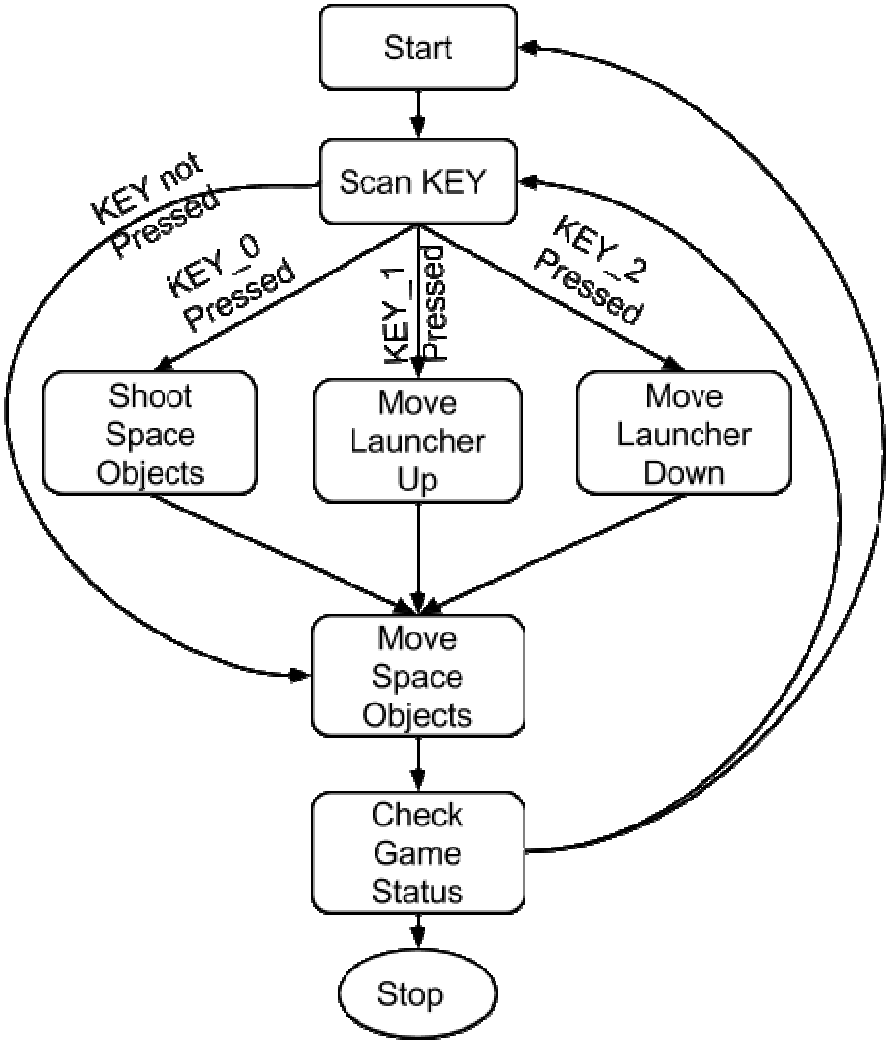

Figure 2: Game state machine.

2.1.2 Program files

The following header and source files are used to implement the application.

Header files

- altera_avalon_timer_regs.h: Declares and defines driver functions to access the timer hardware.
- altera_avalon_ip_vga_regs.h: Declare constants and function prototypes that are used and defined in the corresponding source file.
- SpaceObjects.h: Contains declaration of function prototypes and constants which are related with generating and controlling of the space objects.

- Launcher.h: Declaration of functions to control the launcher.

Source files

- Main_SpaceImpact.c: Implements the SW state machine that integrates all application files.
- altera_avalon_ip_vga_regs.h: Defines all driver function needed to interface the hardware with the application program.
- Launcher.c: Defines all function used to control the launcher on the screen.
- SpaceObjects.c: Defines and uses all function and constants that are declared in the corresponding header file.

2.1.3 Code size

The following code minimization steps are implemented on the code which enables reducing the code footprint from about 100k bytes to about 18k bytes as shown in Table 1.

- Reduced device driver is enabled.
- Small ANSI C library is enabled.
- Lightweight device driver api is enabled
- c_plus_plus, clean_exit and exit BSP properties are disabled
- alt_printf () is used instead of printf()
- Optimization Level is set to “Size”.

Table 1: Code size overview.

Text	Data	Bss	Dec	Hex	Filename
16000	1772	92	17864	45c8	Enjineering_Job.elf

2.2 Hardware

2.2.1 Hardware architecture

The hardware architecture at Figure 3 consists of CPU that executes instructions, communication bus for interfacing the devices in the system, internal RAM and external SRAM to store the software programs, jtag_uart is responsible for serial communication between a PC and FPGA board. MY_TIMER and IP_VGA_0 components are used for timing and displaying purpose respectively. PIO_in_key/PIO_out_led interfaces the I/O pins with the bus to scan input ports and to write to output ports. The character_lcd_0 facilitates communication with LCD on Altera’s board and pio_0 is used to avoid flickering on the screen.

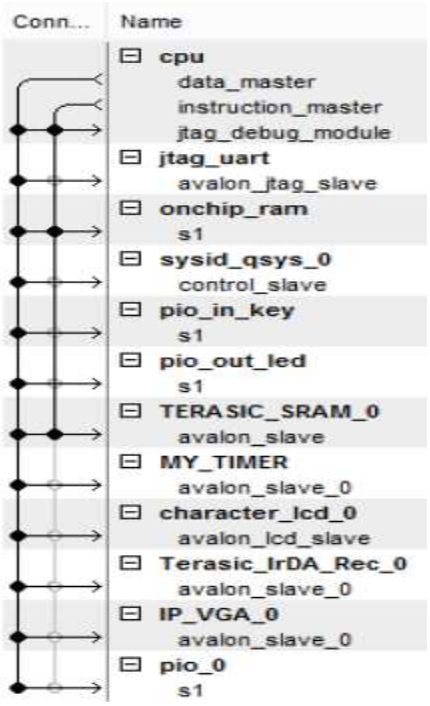

Figure 3: Microcomputer architecture.

2.2.2 Memory map

Among the main hardware components used in this game project are **pi0_in_key** is mapped 0x00101030 - 0x0010103F, **MY_VGA** is mapped 0x00000000 - 0x0007FFFF, **onchip_ram** is mapped 0000C0000 - 0x000EFFFF and MY_TIMER is mapped 0x00101010 - 0x0010101F.

Table 2: Memory map

Slave Descriptor	Address Range	Size	Attributes
TERASIC_SRAM	0x00200000 - 0x002FFFFF	1048576	memory
character_lcd	0x00101054 - 0x00101055	2	
Terasic_IrDA_Rec_0	0x00101050 - 0x00101053	4	
jtag_uart	0x00101048 - 0x0010104F	8	printable
sysid	0x00101040 - 0x00101047	8	
pio_in_key	0x00101030 - 0x0010103F	16	
pio_out_led	0x00101020 - 0x0010102F	16	
MY_TIMER	0x00101010 - 0x0010101F	16	
pio_0	0x00101000 - 0x0010100F	16	
onchip_ram	0x000C0000 - 0x000EFFFF	196608	memory
MY_VGA	0x00000000 - 0x0007FFFF	524288	

2.2.3 Port Descriptions

The microcomputer has the following input/outputs:

- **CLOCK_50** is 50MHz system clock input. All other clocks are generated from this basic clock.
- **SW(17)**: is a reset signal that is connected to slide switch (SW17) on the board
- **KEY** [3..0]: are connected to four push-button switches on the DE2-115 board.
- **LEDG** [3..0] are connected to the four green LEDs on the board.
- **SRAM_DQ**[15..0]: Data signal from Memory
- **SRAM_ADDR**[19..0]: SRAM addresses

- **SRAM_UB_N:** Higher byte strobe
- **SRAM_LB_N:** Lower byte strobe
- **SRAM_WE_N:** Write enable
- **SRAM_CE_N:** Chip select
- **SRAM_OE_N:** Output Enable
- **LCD_DATA**[7..0] Signals to the LCD of the card
- **LCD_ON:** LCD Power ON/OFF
- **LCD_BLON:** LCD Back Light ON/OFF
- **LCD_EN:** LCD Back Light ON/OFF
- **LCD_RS:** LCD Command/Data Select, 0 = Command, 1 = Data
- **LCD_RW**: LCD Read/Write Select, 0 = Write, 1 = Read
- **IRDA_RXD:** is IR receiver and is coupled to the IR detector on the board
- **VGA_CLK:** 25 MHz clock generated in PLL
- **VGA_HS:** Horizontal sync. Make electron beam restart at next screen's scan line
- **VGA_VS**: Vertical sync. Make electron beam restart at first screen's scan line
- **VGA_BLANK_N** : VGA horizontal holding time
- **VGA_SYNC_N**: VGA vertical holding time
- **VGA_B**: Blue intensity
- **VGA_G**: Green intensity
- **VGA_R**: Red intensity
- **Pio_0** is a bit connected to VGA_SYNC_N to write to the VGA memory when the VGA controller is not reading from memory in order to avoid flickering on the screen. Since it is a prototype, a certain flashing accepted upon delivery.

3 Test protocol

The test protocol drawn from the project specification document is shown in Table 3.

Table 3: Test protocol

Case	Description	Ok	Validation (DE2-115)
Case 1	The launcher shall be displayed on the left side of the screen and waits for order from the player	Launcher located on the left	OK/not OK
Case 2	The space objects shall be seen emerging from the right side of the screen at a random time gape.	Space objects are emerging from right side	OK/not OK
Case 3	Push_button 2 pressed	The launcher moves down	OK/not OK
Case 4	Push_button 1 pressed	The launcher moves up	OK/not OK
Case 5	Push_button 0 pressed	The launcher fires weapon	OK/not OK
Case 6	Fired weapon meets the space object	Space object dies and disappear from the screen	OK/not OK
Case 7	15 space objects killed by the player	Level one completes	OK/not OK
Case 8	15 space objects are killed on 2nd level	Game completes	OK/not OK
Case 9	Four space object reached to left before eliminated by the player	The player lose the game	OK/not OK

4 Validation

See video at https://www.youtube.com/watch?v=QsJewEUFqm0

Table 4: Test protocol filled after validation.

Case	Description	Ok	Validation (DE2-115)
Case 1	The launcher shall be displayed on the left side of the screen and waits for order from the player	Launcher located on the left	OK
Case 2	The space objects shall be seen emerging from the right side of the screen at a random time gape.	Space objects are emerging from right side	OK
Case 3	Push_button 2 pressed	The launcher moves down	OK
Case 4	Push_button 1 pressed	The launcher moves up	OK
Case 5	Push_button 0 pressed	The launcher fires weapon	OK
Case 6	Fired weapon meets the space object	Space object dies and disappear from the screen	OK
Case 7	15 space objects killed by the player	Level one completes	OK
Case 8	15 space objects are killed on 2nd level	Game completes	OK
Case 9	Four space object reached to left before eliminated by the player	The player lose the game	OK

5 Conclusion and future improvement

Nearly all the specifications mentioned in the start report are met. The game has two level challenges and is functional. It is possible to improve this game by increasing some more challenging levels. To do that it also requires optimizing the code for achieving best performance.

Appendix

C-code for the entire project

Code is included in the zip file that contains this report.