

WormCraft

CSEE 4840 Embedded System Design

Tianyi Zhang tz2210

Ziwei Zhang zz2282

Yuxuan Zhang yz2580

Ning Li nl2443

March 2014

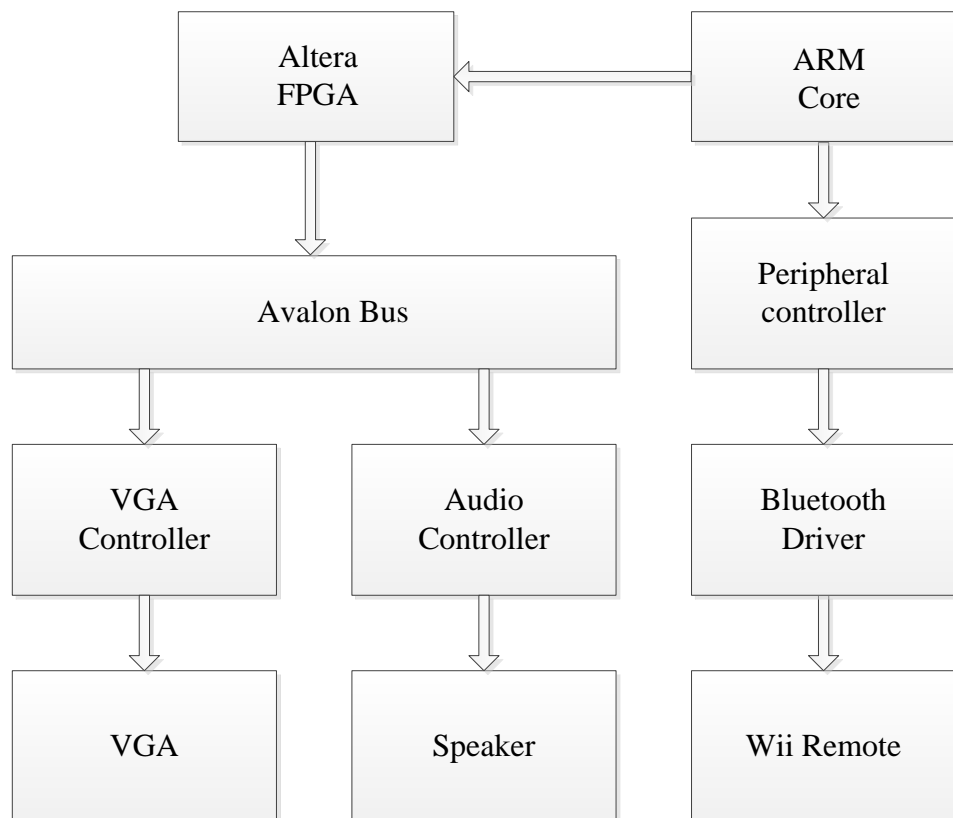
Project Introduction

Our project will implement a 2D Worms® like video game with the sprite graphics technology for two players following the physics rule of gravity and bouncing.

The player will throw two kinds of bombs toward each other until one of the player runs out of life. The game will be implemented with System Verilog and C language. The game will accept Wii remote controller input from user, which will transmit the information of movement, angle and acceleration. The battle scene will be displayed on a VGA output. Multiple types of landforms will be created.

A high-level hardware Architecture

The following is the block structure of our system



As shown in the block diagram, we have Altera SOC FPGA with Arm embedded cores as key part and 4 peripheral components: SRAM controller, VGA controller, Audio Controller and Peripheral controller for Wii mote. The Avalon bus is the communication bus between different components.

VGA Block and Implementation Details

VGA displays the game scenes images and characters with movement which are stored as fixed figures in the memory of the broad. Since the VGA we will use is based on SXGA standard 1280*480 pixels bandwidth of 100MHz, we can refresh the pixel of the screen at 50MHz using for the displays of static and dynamic figures. In order to display the pixels periodically, we can store pixels of 1280*480 in given colors in the framebuffer and display them in the screen in the location we want. Users will interact with the VGA display with Wii Remote™ controller and all of functions in movements, targeting, launching bombs and related settings can be implemented as the corresponding signals are transmitted interrupts to ARM processor via Wii and Bluetooth drivers.

In our design, we are going to use four types of elements that will be displayed on our screen.

1. ground block
2. players
3. bomb
4. explosion
5. background scene

Ground Block

Ground blocks can be used to form the ground in the game. There are three kinds of ground block: *grass*, *dirt* and *stone*. When a bomb is exploded, the *grass* and *dirt* blocks nearby will be destroyed, leaving a crater there. The *stone* blocks cannot be destroyed by the bomb.

Character

Character can be controlled by player to move, jump and throw bombs. Each character may get injured when an explosion happened around. There will be two *character* in the game *player1* and *palyer2* controlled by different player.

Bombs

Bombs are a weapon throw by the *players*, the trajectory of bomb is parabola. There are two kinds of *bombs*, *bomb1* and *bomb2*. *Bomb1* will explode whenever it hit the *ground block* or *player*. *Bomb2* will explode within 5 second after being throws out.

Explosion




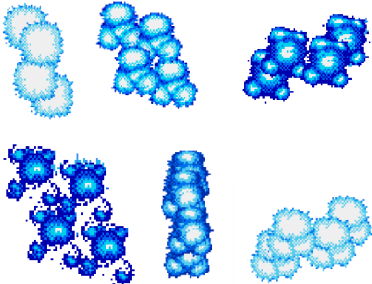
When an explosion happened, there shows explosion effect. Explosions will last a few microseconds shown the explosion effect image from *explosion1* to *explosion6*. Explosions will destroy ground blocks and hurt *players*.

Background Scene

The background image shows us different scenes where battles happen. It cannot be destroyed as by the explosion of bombs like ground blocks. We will implement the scenes using a second level of frame.

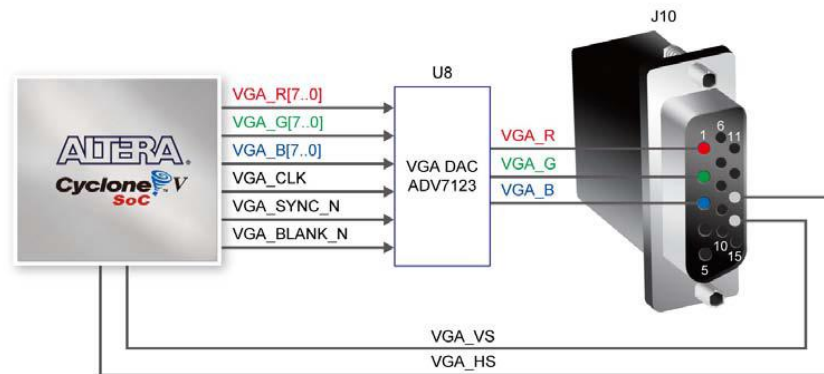
In order to display each kind of basic elements on our screen, we can update each element in the form of sprites. There are two levels of sprites shown on our screen, top level and bottom level. The figures on the top level are ground blocks, character,

bombs and explosion. Because they are probably change at periodically. The background scene on the bottom level will remain the same or dynamic background that changes regularly. So in each cycle we will update the background on the bottom level first and then update the top level right after the bottom one. The size of our screen is 640*480, each sprite in different amount of pixels is shown below.

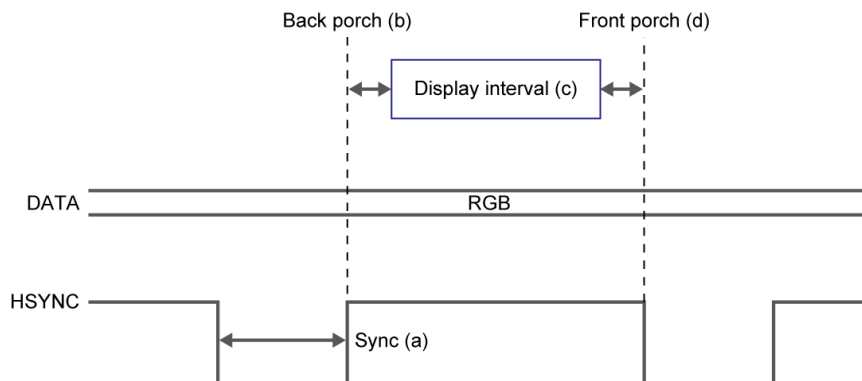
Element	Image	Size (pixels)
Gound_block (grass, dirt, stone)		8*8
Player (front, back, left, right)		16*32
Bomb (1, 2)		8*8
Explosion (1, 2, 3, 4, 5, 6)		40*40

To store the all of the sprites we want to display, we have a framebuffer in the memory responsible for storage of images in pixels. Each pixel occupy 30 bits in the memory, and for the sprite above, we need to have characters of two types, bombs of two types, explosions and ground blocks. Also, we have to display related information of plays, containing 64*16 pixels. In this way, the total memory used for display is estimated as,
 $30*(8*8*3+16*32*2+8*8*2+40*40*6+64*16*2)/8 = 47.36 \text{ KB}$. Background scene needs $640*480*30/8 = 1.10\text{M B}$. So the memory needed in total is 1.14M B.

A D-SUB connector is used for the communication between VGA and FPGA. Signals $VGA_R[7:0]$, $VGA_G[7:0]$, $VGA_B[7:0]$, VGA_CLK , VGA_BLANK_n , VGA_HS , VGA_VS , VGA_SYNC_n are responsible for the communication between VGA and FPGA blocks. The VGA synchronization signals are provided from FPGA, and the Analog Devices ADV7123 triple 10-bit high-speed video DAC is used to produce the data signals (red, green and blue).



The VGA and FPGA communication protocol are illustrated. The figure shown below indicates the timing specification for VGA synchronization and RGB data and the basic timing requirements for each row that is displayed on a VGA monitor.



Application Layer

The Application Layer includes the translated Bluetooth control information and the position information of the player.

We need to use the bluetooth peripheral to connect the Wii Remote Controller to the SoCKit Board. Here it requires a bluetooth USB dongle, which is capable of translating the Bluetooth 4.0 signal to transmission packages.

In the game, the player controls the figure by its movement, bomb tossing angle and bomb tossing strength. So the package should include the figure's position, direction (facing left or right), throwing bomb signal, tossing angle and tossing strength. After that, the algorithm can calculate the figure and enemy's actions by these inputs, transmit the data to SoCKit board and display on the VGA screen.

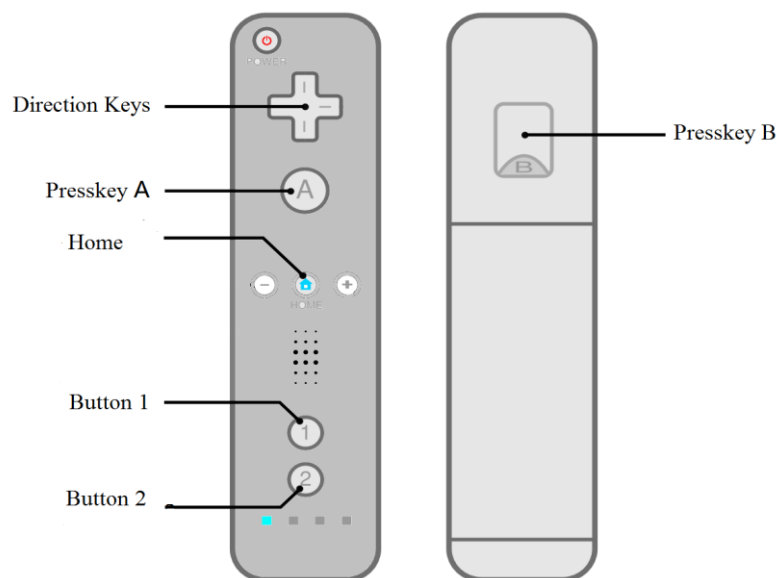
Based on the Bluetooth protocols, our data will transmit on L2CAP packages, which are already implemented by our hardware. The interpretation is achieved by Bluetooth driver, BlueZ. The driver contains the protocol stack of HCI (Human Computer Interface), Logic Link Control and application pro, and RFCOMM baseband. Only the control of the L2CAP will be of consideration in our design.

The designed protocol of Application layer is shown below.

32-23 bit	22-14 bit	13 bit	12-10 bit	9-6 bit	5-1 bit	0 bit
Horizontal Position	Vertical Position	direction	movement	Tossing Angle	Tossing Strength	Toss Signal

Peripheral Block

In this game, we will utilize Wii Remote Controller as a main input to implement game functions. Communication between Wii and SoCKit can be operated through several drivers serially, including USB driver, Bluetooth drivers and Wii drivers. Although the USB driver we use in this game is similar with that in lab 2, Bluetooth driver and Wii driver are needed for the communication between USB and Bluetooth. The Bluetooth controller is designed to be used with devices which follow the Bluetooth Human Interface Device (HID) standard. While there are many documents of the driver in the internet, we have to further modify the drivers to adapt them to the use in Linux.



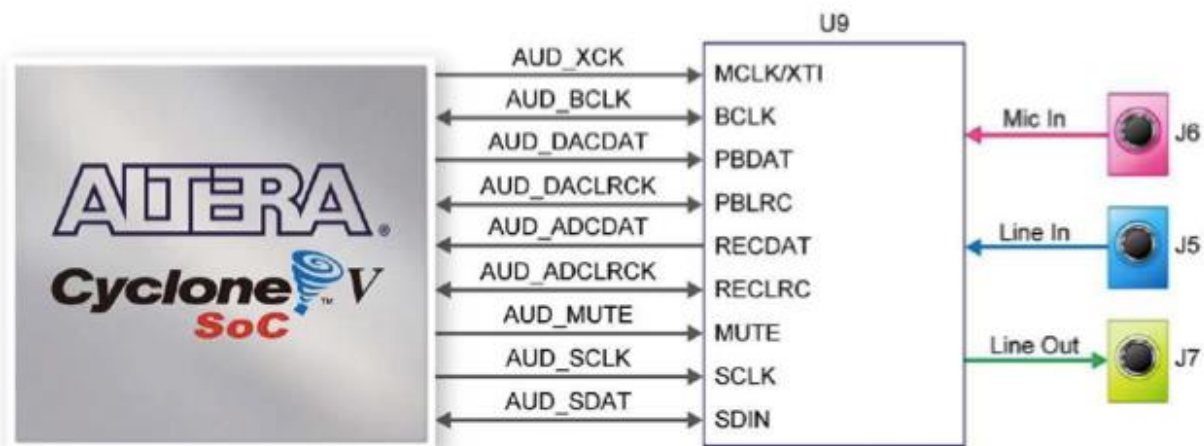
Wii remote input outperform traditional Keyboard and Mouse input. The game controlled by Wii makes it possible for a wireless remote input, which gives user a more convenient, better and more comfortable feeling in operation. All the game process can be implemented with only one hand! Wii remote brings more fun. In order deal with basic control in the game, we will use direction keys, Press-key A and B, as well as Button 1 and 2. The four direction keys are responsible for movement along

the ground surface, for example moving up and down to the hill in the game scene. When the figures want to launch bombs onto each other, we would press Key A while rotating the wrist with Wiimote upwards for a certain angle corresponding to the horizontal direction. The rotation will end when releasing the key. This angle of rotation corresponds to the direction of bombing. After aiming on the target, bombs can be launched by swing Wiimote with Key B pressed. The speed of launched bombs depends on how fast to swing Wiimote. Also, Button 1 and 2 act as shortcut to switch the types of bombs, the figure will be using during the current round. So far, we have developed two kinds of bombs to choose from, each one of which has different characteristics and effects in both image and force. We will probably come up with more kinds of bombs for choose from in the future designs. There will be also a function button in our Wiimote. Home button is responsible for options and settings for the game, including image settings, color settings, audio settings and so on.

Audio Block

The audio data of our project includes background music and special sound effects. The background music is always on for this game. The special sound effects include tossing the bomb and bomb explosion. The HPS obtains audio data from the DDR3 SDRAM, sending the instruction to codec to play the sound. For our project, the bit rate of background music is 64kbps, while the sample rate is 44.1 kHz and the duration is 1 minute. The two sound effects are both 0.5 seconds. The data size for our audio files is 1.5Mb for background music and 32kB for sound effects.

The board provides high-quality 24-bit audio via the Analog Devices SSM2603 audio CODEC (Encoder/Decoder). This chip supports microphone-in, line-in, and line-out ports, with a sample rate adjustable from 8 kHz to 96 kHz. The SSM2603 is controlled via a serial I2C bus interface, which is connected to pins on the Cyclone V SoC FPGA.



Parameter	Limit		Unit	Description
	t _{MIN}	t _{MAX}		
t _{SCS}	600		ns	Start condition setup time
t _{SCH}	600		ns	Start condition hold time
t _{WH}	600		ns	SCLK pulse width high
t _{PL}	1.3		μs	SCLK pulse width low
f _{SCLK}	0	526	kHz	SCLK frequency
t _{OS}	100		ns	Data setup time
t _{OH}		900	ns	Data hold time
t _{tr}		300	ns	SDIN and SCLK rise time
t _{tf}		300	ns	SDIN and SCLK fall time
t _{HCS}	600		ns	Stop condition setup time

Pins Connection:

Figure: Pin Connection Between FGPA and Audio Codec.

Figure: The Timing Diagram for SSM2603

Reference

http://www.analog.com/static/imported-files/data_sheets/SSM2603.pdf

http://www.altera.com/literature/ug/ug_cv_soc_dev_kit.pdf

http://en.wikipedia.org/wiki/Wii_Remote

[http://en.wikipedia.org/wiki/Worms_\(series\)](http://en.wikipedia.org/wiki/Worms_(series))