COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

CSEE 4840 Embedded System Design

Battle Tank- Inspired Multidirectional Shooter Video Game

Group Name: Tank

Lupeng Fan	**LF2447**
Yinshen Wang	**YW2561**
Di Yang	**DY2266**
Yichen Zhu	**YZ2582**

3/26/2014

Contents

List of Figures

List of Tables

1. Project Introduction

Our team proposes to build a classic video game named Battle Tank, which is a multi-directional shooter video game.

The player, controlling a tank, must destroy enemy tanks in each level, which enter the playfield from top-right and top-left corners. The enemy tanks attempt to destroy the player's base, as well as the human tank itself. A level is completed when the player destroys all enemy Tanks, but the game ends if the player's base is destroyed or the player loses all available lives. If the player wins, the player can go to the next level, which the enemy goes faster and have a higher IQ for attacking the base and yourself. And also, based on the different music, the background and the speed of enemy tank can be changed by the style of music.

This game will be implemented with System Verilog and C language and shown on the screen through VGA. Besides, we have background audio and sound when collision and shot have happened. When all the enemies have been killed, the players win the game.

2. A high-level hardware structure

The hardware block diagram for the game is shown in the following figure:

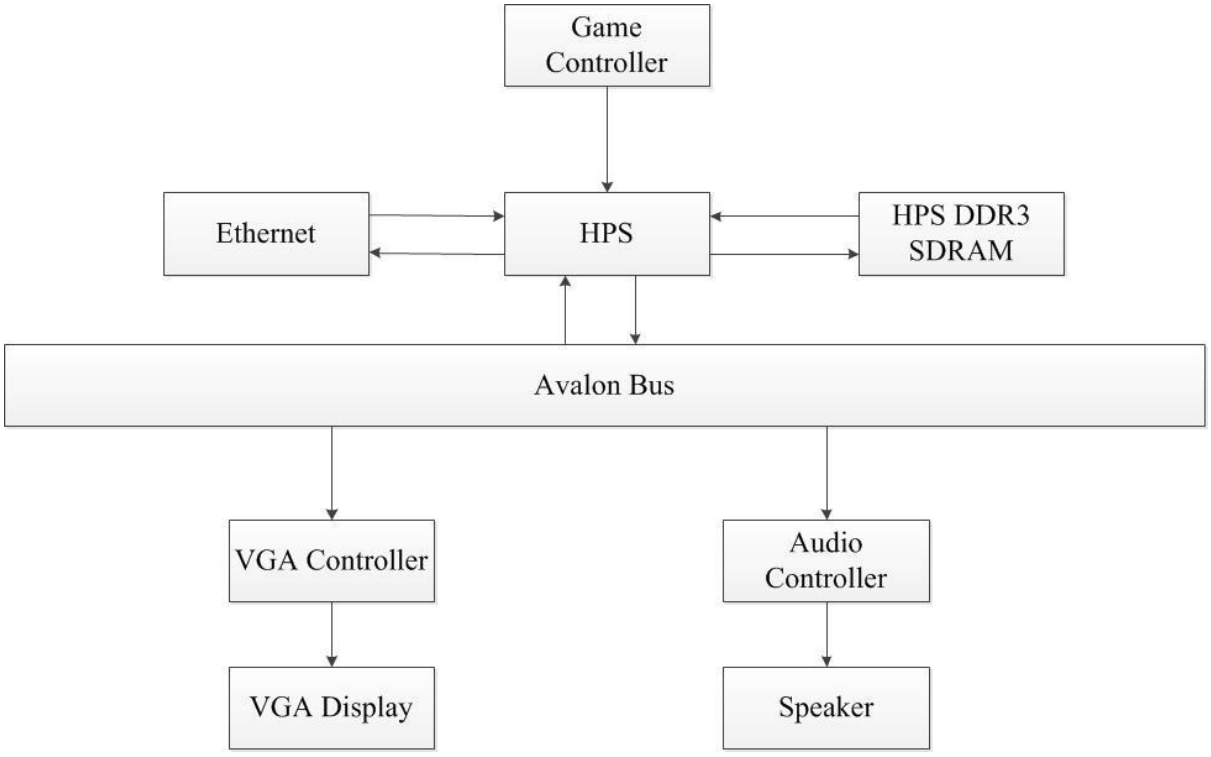

Figure 1: Detailed description of each peripheral will be discussed in the following part.

2.1 VGA Block

There are 9 kinds of elements to be shown:

(1) Two kinds of enemies (each has same appearance but with different color) and different tanks for 2 players.

(2) Background with grey frame for the game.

(3) Steel (Tank cannot move through the steel and the volume of the steel will not be reduced when it is hit by the bullet).

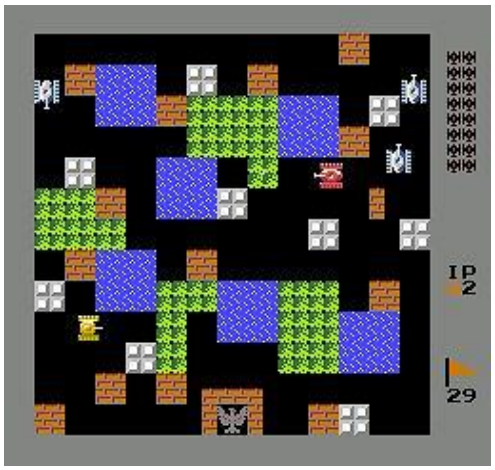

Figure 2: Image of Game

(4) Brick (Tank cannot move through the brick and the volume of the brick will be reduced when it is hit by the bullet).

(5) Water (Tank cannot move through the water but bullet can move through it).

(6) Game information: The number of enemy tank, player tank and current stage.

(7) Bullet and visual effect of explosion.

(8) Bonus for another player tank.

(9) Home base.

All those elements can be stored as fixed figures in the memory on the board and we can use them as saved resources on the FPGA. To realize dynamic display and show the result, we can refresh the sprite periodically. As the screen we use is 640*480 pixels, 32*32 pixels are used for showing each element. As a result, 20*15 areas can be shown on the screen. Each 32*32 pixels are used for showing tank, home base, steel, brick and water. 20*20 pixels are used for showing game information.

The bullet and visual effect of explosion are on the top level, followed by steel, water, brick and home base, then enemy tanks, player tanks and bonus, the last one are game information and background with grey frame. The generation of enemy tank or player tank leads to new sprite because motion of each tank should be treated independently.

The levels of sprites are shown as following:

- Bullet and visual effect of explosion
- Steel, water, brick and home base
- Enemy tanks, player tanks and bonus.
- Game information (current stage, the number of enemy tank and player tank).
- Background with grey frame.

The difficulties for the VGA design:

- Limit the quantity of enemy tanks and prevent all tanks from appearing at the same place.
- When the tank or brick are shot by the bullet, they should disappear. And new tanks should be generated and game information should be changed.

The VGA display is very important for this design because it shows basic graphics of the game. As a result, we can control the display of each sprite by software.
Memory:

For a video frame buffer, each pixel has 30 bits. Tanks, steel, water, brick, bonus, bullet, explosion and home base are in 32*32 size. Game information (10 digits) is in 20*20 size. So total memory for displaying:

30*(32*32*(1*2+1*2+1+1+1+1+1+1+1)+20*20*10)=457920 bits=57.24k bytes.

VGA connections between FPGA and VGA:

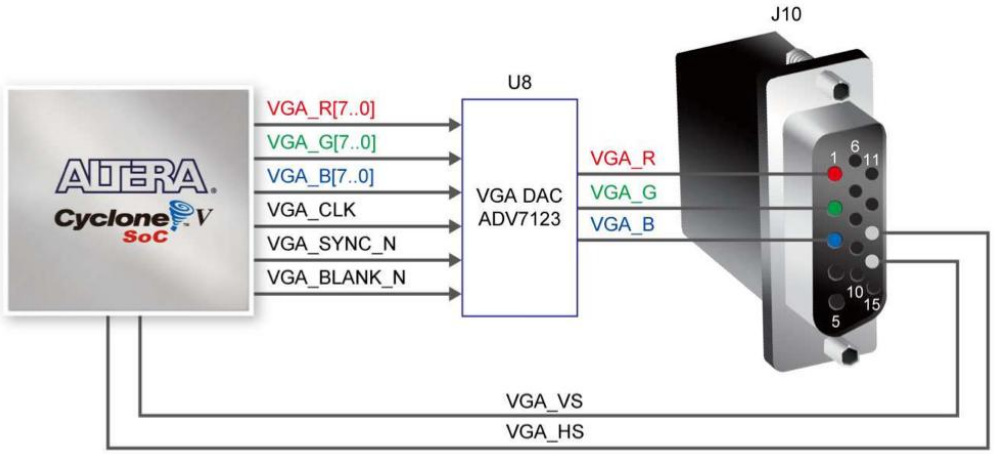

Figure 3: ADV7123 Pin Connection with FPGA

The VGA synchronization signals are provided directly from the SoC FPGA, and the analog device ADV7123 is used to produce the analog data signals. Considering hardware-to-hardware blocks, defined signal and pin assignments are shown in the following figure. We have signal VGA_R[7:0], VGA_G[7:0], VGA_B[7:0], VGA_CLK, VGA_BLANK_n, VGA_HS, VGA_VS, VGA_SYNC_n.

Signal Name	FPGA Pin No.	Description	I/O Standard
VGA_R[0]	PIN_AG5	VGA Red[0]	3.3V
VGA_R[1]	PIN_AA12	VGA Red[1]	3.3V
VGA_R[2]	PIN_AB12	VGA Red[2]	3.3V
VGA_R[3]	PIN_AF6	VGA Red[3]	3.3V
VGA_R[4]	PIN_AG6	VGA Red[4]	3.3V
VGA_R[5]	PIN_AJ2	VGA Red[5]	3.3V
VGA_R[6]	PIN_AH5	VGA Red[6]	3.3V
VGA_R[7]	PIN_AJ1	VGA Red[7]	3.3V
VGA_G[0]	PIN_Y21	VGA Green[0]	3.3V
VGA_G[1]	PIN_AA25	VGA Green[1]	3.3V
VGA_G[2]	PIN_AB26	VGA Green[2]	3.3V
VGA_G[3]	PIN_AB22	VGA Green[3]	3.3V
VGA_G[4]	PIN_AB23	VGA Green[4]	3.3V
VGA_G[5]	PIN_AA24	VGA Green[5]	3.3V
VGA_G[6]	PIN_AB25	VGA Green[6]	3.3V
VGA_G[7]	PIN_AE27	VGA Green[7]	3.3V
VGA_B[0]	PIN_AE28	VGA Blue[0]	3.3V
VGA_B[1]	PIN_Y23	VGA Blue[1]	3.3V
VGA_B[2]	PIN_Y24	VGA Blue[2]	3.3V
VGA_B[3]	PIN_AG28	VGA Blue[3]	3.3V
VGA_B[4]	PIN_AF28	VGA Blue[4]	3.3V
VGA_B[5]	PIN_V23	VGA Blue[5]	3.3V
VGA_B[6]	PIN_W24	VGA Blue[6]	3.3V
VGA_B[7]	PIN_AF29	VGA Blue[7]	3.3V
VGA_CLK	PIN_W20	VGA Clock	3.3V
VGA_BLANK_n	PIN_AH3	VGA BLANK	3.3V
VGA_HS	PIN_AD12	VGA H_SYNC	3.3V
VGA_VS	PIN_AC12	VGA V_SYNC	3.3V
VGA_SYNC_n	PIN_AG2	VGA SYNC	3.3V

Table 1: ADV7123 Pin and Signal Specification

Timing diagram for VGA horizontal timing specification:

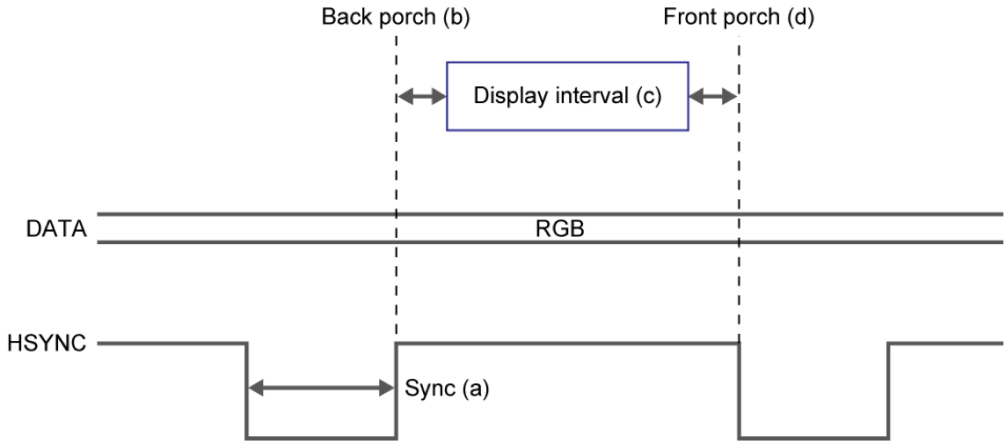

Figure 4: VGA horizontal Timing Diagram

For ADV7123, which receives the digital signals from FPGA and convert them to analog signal, timing diagram for it is shown in the following figure:

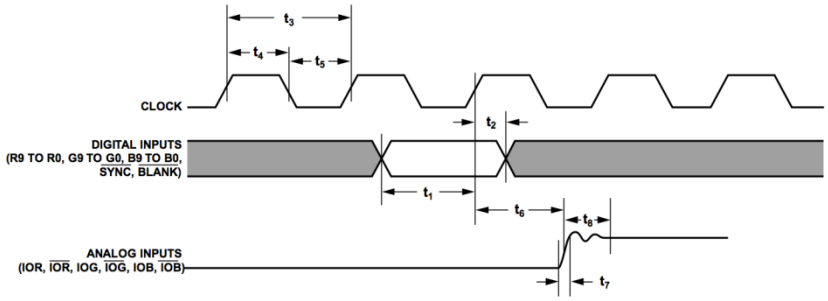

Figure 5:ADV7123 Timing Diagram

2.2 Audio Block

The audio output of our project includes both background music, which can be either turn on or off, and special sound effects i.e. fire and explosion. The CPU gets audio data from DDR3 SRAM, computes and sends commands to audio controller to play the sound. 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. For our project, the audio chip is configured in Master Mode. The audio interface is configured as I2S and 16-bit mode. Furthermore, we decide to use 8kHz sampling rate and duration of the background music is 30s and sound effect is 0.5s respectively. Since our data bus is 16bit, sampling rate is 8kHz and music duration is 30s & two sound effects is 0.5 . The data size is 480kByte for background music and 16kByte for sound effects.

Signals/Pins Connections:

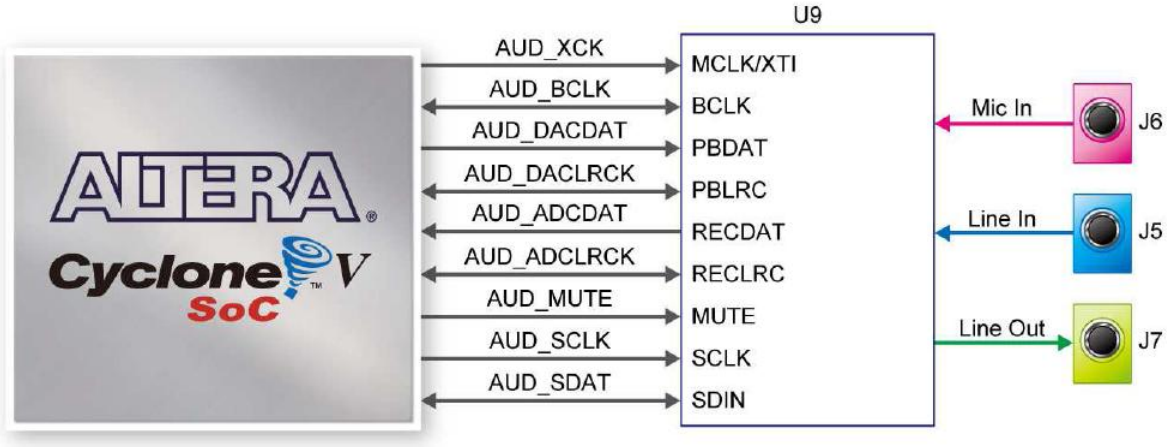

Figure 6: Pin Connections Between FPGA and Audio CODEC

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
AUD_ADCLRCK	PIN_AG30	Audio CODEC ADC LR Clock	3.3V
AUD_ADCDAT	PIN_AC27	Audio CODEC ADC Data	3.3V
AUD_DACLRCK	PIN_AH4	Audio CODEC DAC LR Clock	3.3V
AUD_DACDAT	PIN_AG3	Audio CODEC DAC Data	3.3V
AUD_XCK	PIN_AC9	Audio CODEC Chip Clock	3.3V
AUD_BCLK	PIN_AE7	Audio CODEC Bit-Stream Clock	3.3V
AUD_I2C_SCLK	PIN_AH30	I2C Clock	3.3V
AUD_I2C_SDAT	PIN_AF30	I2C Data	3.3V
AUD_MUTE	PIN_AD26	DAC Output Mute, Active Low	3.3V

Table 2: Pin Assignments for Audio CODEC

SSM2603 Timing Diagram:

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_{PH}	600		ns	SCLK pulse width high
t_{PL}	1.3		µs	SCLK pulse width low
f_{SCLK}	0	526	kHz	SCLK frequency
t_{DS}	100		ns	Data setup time
t_{DH}		900	ns	Data hold time
t_{RT}		300	ns	SDIN and SCLK rise time
t_{FT}		300	ns	SDIN and SCLK fall time
t_{HCS}	600		ns	Stop condition setup time

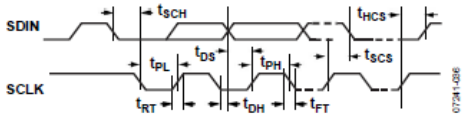

Figure 7: I²C interface Timing Diagram

Parameter	Limit		Unit	Description
	t_{MIN}	t_{MAX}		
t_{DST}	30		ns	PBDAT setup time to BCLK rising edge
t_{DHT}	10		ns	PBDAT hold time to BCLK rising edge
t_{DL}		10	ns	RECLRC/PBLRC propagation delay from BCLK falling edge
t_{DDA}		10	ns	RECDAT propagation delay from BCLK falling edge
t_{BCLKR}	10		ns	BCLK rising time (10 pF load)
t_{BCLKF}	10		ns	BCLK falling time (10 pF load)
t_{BCLKDS}	45:55:00	55:45:00		BCLK duty cycle (normal and USB mode)

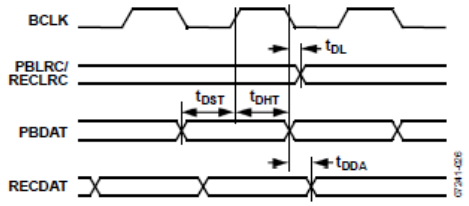

Figure 4. Digital Audio Interface Master Mode Timing

Figure 8: Digital Audio Interface Master Mode Timing

Parameter	Limit		Unit	Description
	t_{MIN}	t_{MAX}		
t_{XTIY}	54		ns	MCLK/XTI clock cycle time
t_{MCLKDS}	40:60	60:40		MCLK/XTI duty cycle
t_{XTIH}	18		ns	MCLK/XTI clock pulse width high
t_{XTIL}	18		ns	MCLK/XTI clock pulse width low
t_{COP}		20	ns	CLKOUT propagation delay from MCLK/XTI falling edge
$t_{COPDIV2}$		20	ns	CLKODIV2 propagation delay from MCLK/XTI falling edge

[1] CLKDIV2 bit (Register R8, Bit D6) is set to 0

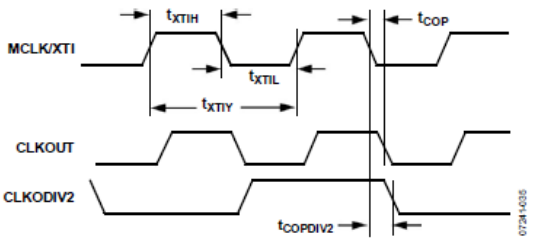

Figure 9: Master Clock Timing

The digital audio input can support the following four digital audio communication protocols: right-justified mode, left-justified mode, I²S mode, and digital signal processor (DSP) mode.

For our project we choose to use I^2S Mode:

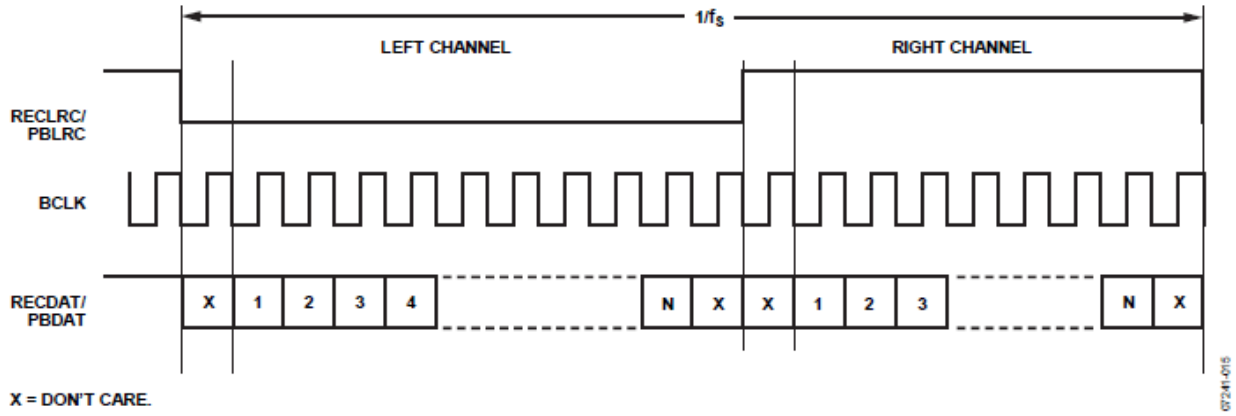

Figure 10: I2S Audio Input Mode

The software control interface provides access to the user-selectable control registers and can operate with a 2-wire (I2C) interface.
Within each control register is a control data-word consisting of 16 bits, MSB first. Bit B15 to Bit B9 are the register map address, and Bit B8 to Bit B0 are register data for the associated register map.

SDIN generates the serial control data-word, SCLK clocks the serial data, and CSB determines the I2C device address. If the CSB pin is set to 0, the address selected is 0011010; if 1, the address is 0011011.

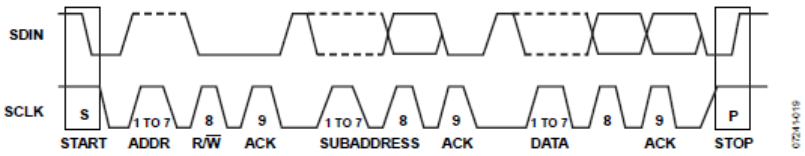

Figure 28. 2-Wire I²C Generalized Clocking Diagram

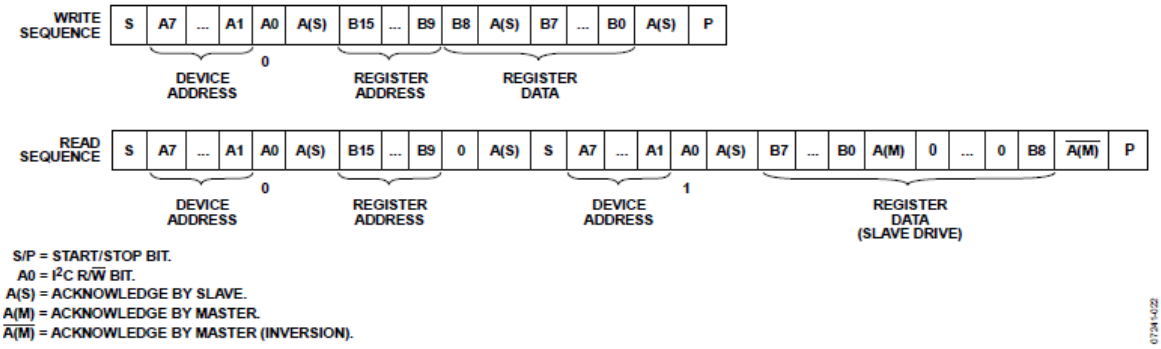

Figure 11: I2C Write and Read Sequences

SSM2603 Register Map:

Reg.	Address	Name	D8	D7	D6	D5	D4	D3	D2	D1	D0	Default
R0	0x00	Left-channel ADC input volume	LRINBOTH	LINMUTE	0	LINVOL[5:0]						010010111
R1	0x01	Right-channel ADC input volume	RLINBOTH	RINMUTE	0	RINVOL[5:0]						010010111
R2	0x02	Left-channel DAC volume	LRHPBOTH	0	LHPVOL[6:0]							001111001
R3	0x03	Right-channel DAC volume	RLHPBOTH	0	RHPVOL[6:0]							001111001
R4	0x04	Analog audio path	0	SIDETONE_ATT[1:0]		SIDETONE_EN	DACSEL	Bypass	INSEL	MUTEMIC	MICBOOST	000001010
R5	0x05	Digital audio path	0	0	0	0	HPOR	DACMU	DEEMPH[1:0]		ADCHPF	000001000
R6	0x06	Power management	0	PWROFF	CLKOUT	OSC	Out	DAC	ADC	MIC	LINEIN	010011111
R7	0x07	Digital audio I/F	0	BCLKINV	MS	LRSWAP	LRP	WL[1:0]		Format[1:0]		000001010
R8	0x08	Sampling rate	0	CLKODIV2	CLKDIV2	SR[3:0]				BOSR	USB	000000000
R9	0x09	Active	0	0	0	0	0	0	0	0	Active	000000000
R15	0x0F	Software reset	Reset[8:0]									000000000
R16	0x10	ALC Control 1	ALCSEL[1:0]		MAXGAIN[2:0]			ALCL[3:0]				001111011
R17	0x11	ALC Control 2	0	DCY[3:0]				ATK[3:0]				000110010
R18	0x12	Noise gate	0	NGTH[4:0]					NGG[1:0]		NGAT	000000000

Table 3: SSM2603 Register Map

2.3 Game Controller

The game uses two PC game controllers to get the operation signals from the players. One controller will be connected directly to the SoCKit Board and communicate with the HPS through the USB port on the board. The other one will be connected to a PC and communicate with the HPS through the Ethernet between the PC and the SoCKit board. Arrow keys of the controller will be used to direct the tank to dodge the bullets from the enemies. And a shooting button will be used to shoot the enemy tanks to protect the homebase from being attacked. A cancle button to undo is also needed. Like lab2, a game controller driver needs to be built in System Verilog. And the interface for the game controller will be implemented in C program. The schematic diagram of the USB circuitry is shown below.

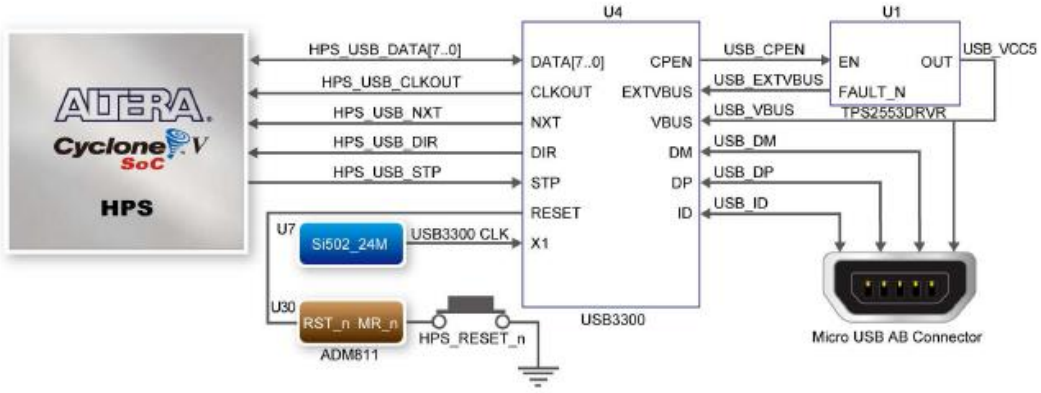

Figure 12: Connections between CycloneV SoC FPGA and USB OTG PHY

The Sockit board provides USB interfaces using the SMSC USB3300 controller. A SMSC USB3300 device in a 32-pin QFN package device is used to interface to a single Type AB Micro-USB connector. This device supports UTMI+ Low Pin Interface (ULPI) to communicate to USB 2.0controller in HPS. The ULPI interface consists of 12 interface pins;

8 bi-directional data pins, 3 control pins, and a 60 MHz clock. As defined by OTG mode, the PHY can operate in Host or Device modes. When operating in Host mode, the interface will supply the power to the device through the Micro-USB interface. In our project, we will make the PHY work in the device mode as what we need are just signals from the game controller. The pin assignments for the associated interface are listed in Tables below.

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
HPS_USB_CLKOUT	PIN_N16	60MHz Reference Clock Output	3.3V
HPS_USB_DATA[0]	PIN_E16	HPS USB_DATA[0]	3.3V
HPS_USB_DATA[1]	PIN_G16	HPS USB_DATA[1]	3.3V
HPS_USB_DATA[2]	PIN_D16	HPS USB_DATA[2]	3.3V
HPS_USB_DATA[3]	PIN_D14	HPS USB_DATA[3]	3.3V
HPS_USB_DATA[4]	PIN_A15	HPS USB_DATA[4]	3.3V
HPS_USB_DATA[5]	PIN_C14	HPS USB_DATA[5]	3.3V
HPS_USB_DATA[6]	PIN_D15	HPS USB_DATA[6]	3.3V
HPS_USB_DATA[7]	PIN_M17	HPS USB_DATA[7]	3.3V
HPS_USB_DIR	PIN_E14	Direction of the Data Bus	3.3V
HPS_USB_NXT	PIN_A14	Throttle the Data	3.3V
HPS_USB_RESET_PHY	PIN_G17	HPS USB PHY Reset	3.3V
HPS_USB_STP	PIN_C15	Stop Data Stream on theBus	3.3V

Table 4: Table for USB OTG PHY Pin Assignments

PIN	*NAME*	*TYPE*	*ACTIVE LEVEL*	*DESCRIPTION*
1	GND	Ground	N/A	Ground
2	GND	Ground	N/A	Ground
3	CPEN	Output,CMOS	High	EX. Supply EN
4	VBUS	I/O,Analog	N/A	VBUS PIN
5	ID	Input,Analog	N/A	ID PIN
6	VDD3.3	Power	N/A	3.3V Supply
7	DP	I/O,Analog	N/A	D+ PIN
8	DM	I/O,Analog	N/A	D- PIN
9	RESET	Input,CMOS	High	Transceiver reset
10	EXTVBUS	Input,CMOS	High	External bus detect
11	NXT	Output,CMOS	High	Data throttling
12	DIR	Output,CMOS	N/A	Data direction
13	STP	Input,CMOS	High	Data stop
14	CLKOUT	Output,CMOS	N/A	60MHz CLK output
15	VDD1.8	Power	N/A	1.8V for digital circuit
16	VDD3.3	Power	N/A	0.1uF cap. connected
17	DATA[7]	I/O,CMOS Pull-low	N/A	8-bit bi-directional data bus
18	DATA[6]	I/O,CMOS Pull-low	N/A	8-bit bi-directional data bus

19	**DATA[5]**	**I/O,CMOS Pull-low**	**N/A**	**8-bit bi-directional data bus**
20	**DATA[4]**	**I/O,CMOS Pull-low**	**N/A**	**8-bit bi-directional data bus**
21	**DATA[3]**	**I/O,CMOS Pull-low**	**N/A**	**8-bit bi-directional data bus**
22	**DATA[2]**	**I/O,CMOS Pull-low**	**N/A**	**8-bit bi-directional data bus**
23	**DATA[1]**	**I/O,CMOS Pull-low**	**N/A**	**8-bit bi-directional data bus**
24	**DATA[0]**	**I/O,CMOS Pull-low**	**N/A**	**8-bit bi-directional data bus**
25	**VDD3.3**	**Power**	**N/A**	**0.1uF cap. connected**
26	**VDD1.8**	**Power**	**N/A**	**1.8V for digital circuit**
27	**XO**	**Output,Analog**	**N/A**	**Crystal Pin**
28	**XI**	**Input,Analog**	**N/A**	**Crystal Pin**
29	**VDDA1.8**	**Power**	**N/A**	**1.8V for analog circuit**
30	**VDD3.3**	**Power**	**N/A**	**Analog 3.3V supply**
31	**REG_EN**	**I/O,CMOS Pull-low**	**N/A**	**On-chip 1.8V regulator enable**
32	**RBIAS**	**Analog,CMOS**	**N/A**	**External bias resistor**
	GND FLAG	**Ground**	**N/A**	**Ground**

Table 5: Table for USB3300 Pin Assignments

2.4 Ethernet Block

We need to use the Ethernet peripheral to connect the PC or laptop with the SoCKit Board in order to achieve the multiplayer mode. The network design will be similar to that of the lab 2. In the game, each player controls the tank by the direction and shot. Thus, the communication network needs to transmit the packets between PC and SoCKit Board. The packets should include its own tank's position, direction and shot signal. After that, the algorithm can calculate the tanks and the enemy performances by these inputs and show the graphs by VGA blocks.

Based on Ethernet, we will use the standard UDP packets with IP headers specifically designed. Due to UDP doesn't have handshaking dialogues, it is possible that the packet loss and packet corruption will take place during the transmission. However, we must guarantee the packets to be transmitted safely. Thus, we decide to use an ACK protocol to check whether the packets have been lost. That's mean that when a packet has been sent, the sender wait for the ACK signal from receiver to acknowledge the transmission successfully. Then the sender can transmit another packet. If the sender hasn't received the ACK by the constant

time (For example: 500ms), we need to send the current packet again. Multiple failures to transmit the packet will change the game to single player mode. If the received packet is wrong, such as the invalid position, we will not send the ACK and wait for the packet again. In the worst case, if the corruption of the packet occurs multiple times, we will change the game to the single player mode.

And also, we need to design the three sections (own tank's position, direction and shot signal) in Application Layer. Other bits will be reserved for additional use.

The designed protocol of the Application Layer is shown below. Note that additional fields may be added during design.

31-28 bit	27-18 bit	17-9 bit	8 bit	7-4 bit	3-0 bit
Data Type ID(can be reserved)	Horizontal Position	Vertical Position	Shot Signal	direction	reserved

Table 6: Table for Designed Protocol of the Application Layer

Signals/Pins Connections:

The board provides Ethernet support through the chip of Micrel KSZ9021RN PHY and HPS Ethernet MAC function. The KSZ9021RN chip with integrated 10/100/1000 Mbps Gigabit Ethernet transceiver supports Reduced gigabit media independent interface (RGMII) MAC interfaces. Figure 13 shows the connection of pins between the Gigabit Ethernet PHY and Cyclone V SoC FPGA.

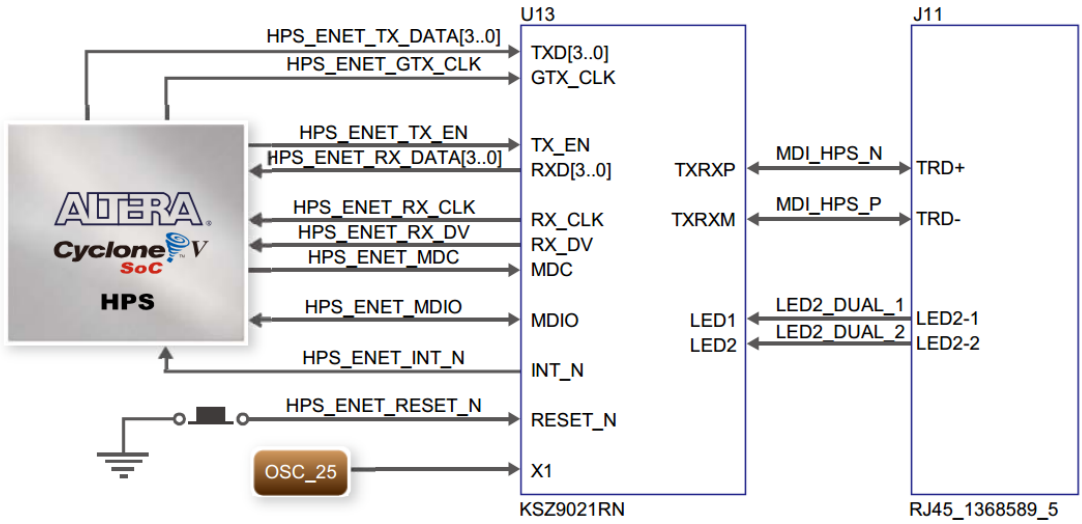

Figure 13: Signals/Pins Connection

RGMII Signal Definition:

The following table describes the RGMII signals.

RGMII Signal Name (per spec)	RGMII Signal Name (per KSZ9021RL/RN)	Pin Type (with respect to PHY)	Pin Type (with respect to MAC)	Description
TXC	GTX_CLK	Input	Output	Transmit Reference Clock (125MHz for 1000Mbps, 25MHz for 100Mbps, 2.5MHz for 10Mbps)
TX_CTL	TX_EN	Input	Output	Transmit Control
TXD[3:0]	TXD[3:0]	Input	Output	Transmit Data [3:0]
RXC	RX_CLK	Output	Input	Receive Reference Clock (125MHz for 1000Mbps, 25MHz for 100Mbps, 2.5MHz for 10Mbps)
RX_CTL	RX_DV	Output	Input	Receive Control
RXD[3:0]	RXD[3:0]	Output	Input	Receive Data [3:0]

Table 7: RGMII Pin and Signal Specification

The following table shows the KSZ9021RL/RN pin-out assignments for MDI/MDI-X pin mapping

Pin (RJ-45 pair)	MDI			MDI-X		
	1000Base-T	100Base-TX	10Base-T	1000Base-T	100Base-TX	10Base-T
TXRXP/M_A (1,2)	A+/-	TX+/-	TX+/-	B+/-	RX+/-	RX+/-
TXRXP/M_B (3,6)	B+/-	RX+/-	RX+/-	A+/-	TX+/-	TX+/-
TXRXP/M_C (4,5)	C+/-	Not used	Not used	D+/-	Not used	Not used
TXRXP/M_D (7,8)	D+/-	Not used	Not used	C+/-	Not used	Not used

Table 8: KSZ9021RL/RN Pin and Signal Specification

Timing Diagrams:

RGMII Timing

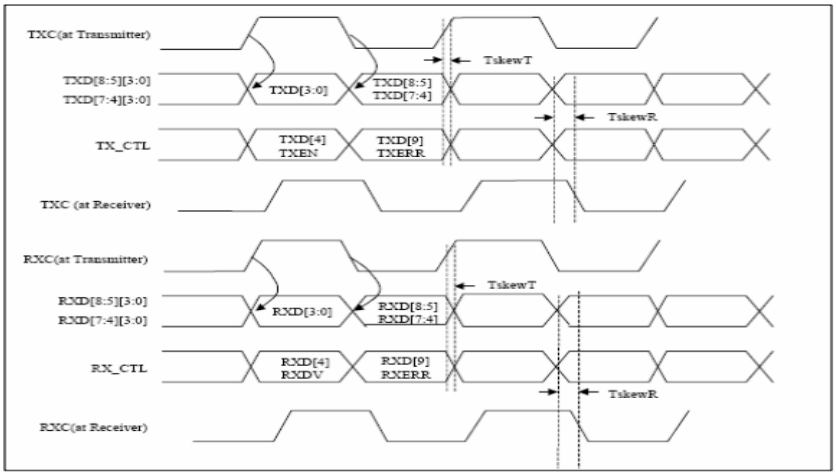

Figure 14: RGMII Timing

MDC/MDIO Timing

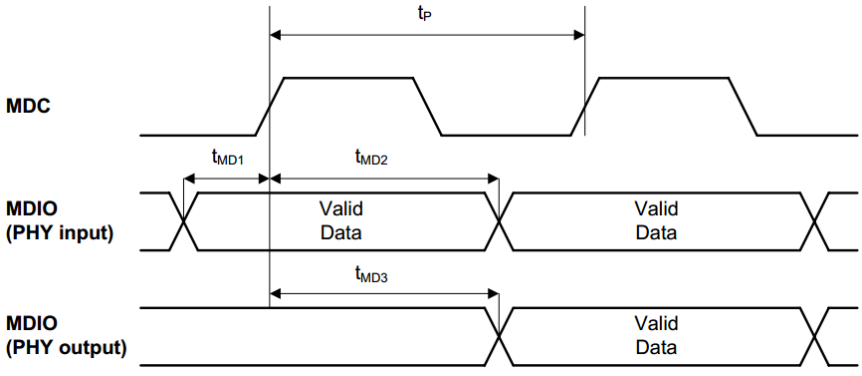

Figure 15: MDC/MDIO Timing

Register Map:

The IEEE 802.3 Specification provides a 32 register address space for the PHY. Register 0 thru 15 are standard PHY registers, defined per the specification. Register 16 thru 31 are vendor specific registers.

- Register 11(Bh) for Extended Register – Control
- Register 12 (Ch) for Extended Register – Dtat Write
- Register 13 (Dh) for Extended Register – Dtat Read

Register Number(Hex)	Description
IEEE Defined Registers	
0(0h)	Basic Control
1(1h)	Basic Status
2(2h)	PHY Identifier 1
3(3h)	PHY Identifier 2
4(4h)	Auto-Negotiation Advertisement
5(5h)	Auto-Negotiation Link Partner Ability
6(6h)	Auto-Negotiation Expansion
7(7h)	Auto-Negotiation Next Page
Register Number(Hex)	**Description**
8(8h)	Auto-Negotiation Link Partner Next Page Ability
9(9h)	1000Base-T Control
10(Ah)	1000Base-T Status
11(Bh)	Extended Rigister-Control
12(Ch)	Extended Rigister-Date Write
13(Dh)	Extended Rigister-Date Read
14(Eh)	Reserved
15(Fh)	Extended- MII Status
Vendor Specific Registers	
16(10h)	Reserved
17(11h)	Remote Loopback, LED Mode
18(12h)	LinkMD –Cable Diagnostic
19(13h)	Digital PMA/PCS Status
20(14h)	Reserved
21(15h)	RXER Counter
22(16h)-26(1Ah)	Reserved
27(1Bh)	Interrupt Control/Status
28(1Ch)	Digital Debug Control 1
29(1Dh)-30(1Eh)	Reserved
31(1Fh)	PHY Control
Extended Register	
256(100h)	Common Control
257(101h)	Strap Status
258(102h)	Operation Mode Strap Override
259(103h)	Operation Mode Strap Status
260(104h)	RGMII Clock and Control Pad Skew
261(105h)	RGMII RX Data Pad Skew

263(107h)	Analog Test Register

Table 9: PHY Register Map

3. Algorithm Description

This typical Battle Tank game has 2 modes – single player mode and double player mode. The double player mode can be two players using same keyboard or through Ethernet connection. The game follow the typical game rules and it comprises 3 levels based on game difficulties (i.e. No. and speed of opponents and map layout). The opponent tanks are randomly generated at the top-right and top-left corners. Besides, they are moving in random direction and shoot randomly. Furthermore, the obstructions in the map includes brick walls (can be destroyed by tank shots), steel walls (cannot be destroyed) and river (tank shots can pass trough it).

In summary, the game features are:

- USB keyboard control, capable of fast movement and shots.
- VGA graphic display using RGB color and simple 2D graphic effects for actions and scenes.
- Sound effects for shooting\moving\winning\died actions and background music.
- Three large scenes with interactive environmental objects.
- A simple algorithm for opponents routing plan.
- Multiple selection of game modes, including single player and double mode (single keyboard input, multi keyboard input through Ethernet (optional))

4. Critical Path

For our project, the critical path is determined by the sequence of operation as follows:

Firstly, player presses a button on the game controller connected to a computer . Then, the computer will send the data of button pressed to the FPGA through Ethernet and the Ethernet controller store the number code of the button for the reading operation of CPU. CPU read the button word and determine which action should be executed and put it on the bus and write it in the RAM of VGA controller. The VGA controller tells VGA which array and where the array should be displayed next. After that, the CPU will write the command in the RAM of audio controller, and lastly, the audio controller reads the sound effect data from DDR3 RAM and sends to audio codec to play the sound effect. This is the longest path in this game, so if the player enter the button too fast, then the VGA may fail to display the right array and play the sound.