SVGA Network Poker : Group Report

Project Title:	**SVGA Network Poker**

Project Team Members:	**Lyndon Carvalho** **Hin Chan**

Names:	**Lyndon Carvalho (991007361)** **Hin Chan (990953660)**
Professor:	**Prof. Paul Chow**
Course:	**ECE532 Digital Hardware**
Date:	**March 28, 2005**

Contents

Acknowledgements

We would like to thank our course instructor, Professor Paul Chow, for the opportunity to participate in this project and to gain exposure to full system design with Xilinx technology and tools. His lectures and guidance in labs were invaluable to the completion of this project.

We would also like to thank the course teaching assistants, Chris Comis and Lesley Shannon, who helped a lot inside and outside of labs with advice, knowledge and equipment whenever needed. In addition, we want to thank Patrick Akl from the American University of Beirut for his summer project on a reliable Ethernet communication system for Xilinx MicroBlaze FPGA systems. His professional work gave us a strong base to adapt and build upon. His prompt suggestions to problems were also very helpful in troubleshooting.

Finally, we would like to acknowledge our Texas Hold'em poker group (Nigel, Justin, Eric, Hattie, Chris, John, Simon and Ajay) for the inspiration and support for this project.

Tables

Figures

1. Introduction

Computer systems are used everywhere in the world for many different applications. Each system ranges in size, type, power and performance. Three key factors that must be considered in designing such complex systems involve performance, cost and application. Any computer that needs to process large amounts of data at fast speeds must have exceptional performance but comes at a higher cost. Supercomputers are well known in this area to provide highly reliable and fast computing power for applications such as nuclear simulations, computer aided design and even video gaming.

The personal computer is the low cost, general application counterpart to the supercomputer and can be found in almost every home in North America these days. While these systems are lower in cost, they still need to be designed with a general application in mind for different computations dealing with simulation, graphics, gaming, networking and etc.

There are many specific applications that do not need high performance computing or generality in design. Processors are now found in cellular phones, digital cameras, cars, toys, appliances, houses…the list goes on. Each of these applications is very specific in task and demand. They do not require high performance but need low cost, low power and highly reliable components. This need has led to the evolution of systems on chip. These systems include a microprocessor, I/O interfaces, memory, timers, and controllers embedded onto a single device. Such systems are referred to as embedded systems. They have been made more easily deployable with the reduction of CMOS technology nodes used to fabricate silicon devices.

This technology has been adapted to FPGAs that allow the designer to create systems on a programmable chip. The topic of embedded systems and the FPGA to create an application specific system on a programmable chip is the main focus of this project.

2. Overview

2.1. Project Description

This report outlines a two-player network poker card game implemented on the Xilinx Multimedia Board[1] with a Xilinx Virtex-II FPGA[2]. Each running game communicates over an Ethernet connection to a duplicate system running the same hardware on another Xilinx Multimedia Board. In addition, each system displays the game status for each player onto an SVGA monitor and takes input from a keyboard.

The interaction between different players or systems proceeds on a server/client protocol where one system hosts the game while the other system connects to it over the Ethernet. The server side system controls and keeps track of the entire game-play, including the cards dealt, the winner of each hand and each player's cash. The other system functions as a client that connects to the server system and runs its own copy of the game communicating over the Ethernet.

The variant of poker played in this project is Texas Hold'em.[3]

2.2. Goals

The main goals of this project were as follows:

- Develop exposure to full hardware and software embedded system design using state of the art technology and tools from Xilinx
- Gain exposure to EDA design tools and FPGAs to implement complex designs, specifically in the approaches for large digital systems encompassing a lot of gates and a lot of software

- Use good design practices for modular design while adhering to the EDA design flow with everything from design specification to synthesis to programming
- Follow a good modular design process to divide and develop tasks between the team members and integrate the full system successfully
- Propose and meet a strict design schedule with milestones for each week (see Table 2-1)

Table 2-1 Milestones

2005 Dates	Expected Milestones
February 23	▪ Ethernet driver software complete ▪ Base hardware system design complete
March 2	▪ Ethernet control software complete ▪ Ethernet game-play protocol complete
March 9	▪ Poker game-play complete ▪ SVGA hardware custom core complete
March 16	▪ SVGA Video drivers and software core complete ▪ Network poker game-play implementation on FPGA using UART for I/O
March 23	▪ System integration of Ethernet, poker game-play and SVGA display complete ▪ Project demo

2.3. Technical System Description

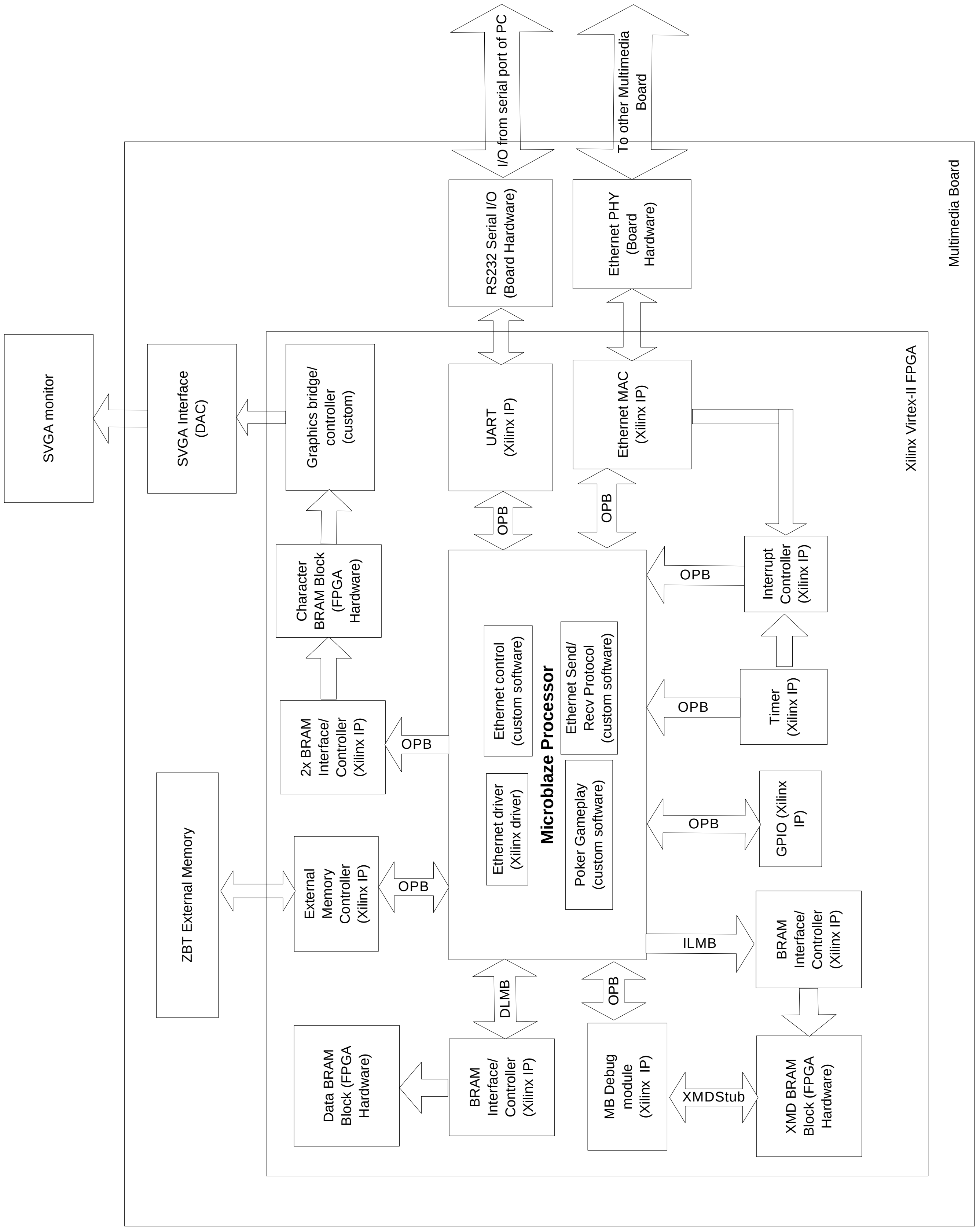
2.3.1. System Topology

This project consists of a fully embedded hardware and software system for playing SVGA Texas Hold'em poker between two players across an Ethernet network. The system has two Xilinx Multimedia Boards that connect through a 10 Mbps Ethernet hub using two RJ45 Ethernet cables. The hub works as a simple repeater transmitting received frames to all ports connected to it. Each board connects to a PC via the JTAG parallel port (for programming and debugging) and the RS232 serial port (for user input and output debugging statements). An SVGA monitor connects to the SVGA output of each board to display the game onto the screen in a user-friendly and aesthetic GUI.

2.3.2. High Level Description

The heart of the system consists of the Xilinx Virtex-II FPGA. The FPGA is programmed with the core system to control other hardware devices on the board for I/O, memory, display and communication functions. The full system block diagram is shown in Figure 2-1. The MicroBlaze[4] soft processor core is central to the embedded system and connects to all the other on chip IP blocks via various buses and software interfaces. The MicroBlaze also runs all the custom designed software for the embedded system. This includes the Ethernet control and communication software as well as the poker game-play functions.

Figure 2-1 System Block Diagram


I/O from serial port of PC
To other Multimedia Board
Multimedia Board
RS232 Serial I/O (Board Hardware)
Ethernet PHY (Board Hardware)
Xilinx Virtex-II FPGA
SVGA monitor
SVGA Interface (DAC)
Graphics bridge/ controller (custom)
UART (Xilinx IP)
Ethernet MAC (Xilinx IP)
OPB
OPB
Character BRAM Block (FPGA Hardware)
Interrupt Controller (Xilinx IP)
OPB
Microblaze Processor
Ethernet control (custom software)
Ethernet Send/ Recv Protocol (custom software)
Ethernet driver (Xilinx driver)
Poker Gameplay (custom software)
2x BRAM Interface/ Controller (Xilinx IP)
OPB
Timer (Xilinx IP)
OPB
GPIO (Xilinx IP)
OPB
ZBT External Memory
External Memory Controller (Xilinx IP)
OPB
ILMB
BRAM Interface/ Controller (Xilinx IP)
DLMB
OPB
Data BRAM Block (FPGA Hardware)
BRAM Interface/ Controller (Xilinx IP)
MB Debug module (Xilinx IP)
XMDStub
XMD BRAM Block (FPGA Hardware)

2.3.3. Hardware IP Blocks

2.3.3.1. Processor

The MicroBlaze (Xilinx IP) is a standard 32-bit RISC Harvard-style soft processor that is especially developed for the Virtex FPGA architecture. It can be used with Xilinx EDK 6.3[5] software to develop a base system on the Xilinx Multimedia Board. It is the brains of the system controlling each IP block through software embedded in memory. The core can be instantiated in Xilinx Platform Studio by following the tutorial in module m01[6] on the ECE532 course website.

2.3.3.2. Universal Asynchronous Receiver/Transmitter (UART)

The UART (Xilinx IP) core handles I/O to and from the system for user input and debugging. It connects to an off chip RS232 serial port. This port directly connects to the serial interface of a PC. The core can be instantiated along with a MicroBlaze base system by following module m01 from the previous section.

2.3.3.3. General Purpose I/O (GPIO)

The GPIO (Xilinx IP) core controls general purpose input and output signals from switches or LED's on the board. The system uses the GPIO with on board switches to manually reset the processor and the SVGA monitor. The core can be added to the base system by following module m02[7] from the ECE532 course website.

2.3.3.4. Timer & Interrupt Controller

The Timer (Xilinx IP) and Interrupt Controller (Xilinx IP) cores are used by the system for timing events and interrupting processor operation to service requests from other connected peripherals such as the EMAC. The cores can be added to the base system by following module m03[8] from the ECE532 course website.

2.3.3.5. Ethernet

Communication between server and client boards is controlled with the help of the Ethernet Media Access Controller (EMAC) IP core which connects to an on board Ethernet physical transceiver chip. This EMAC (Xilinx IP) core is responsible for all sending and receiving of Ethernet frames. It can be added to the base MicroBlaze system by following the tutorial in module m04[9] on the ECE532 course website.

2.3.3.6. Microblaze Debug Module (MDM)

The MicroBlaze Debug Module (Xilinx IP) core performs debugging on the hardware and software system during run time. It can be instantiated with the MicroBlaze base system by following module m01.

2.3.3.7. ZBT External Memory Controller (EMC)

The ZBT External Memory Controller (Xilinx IP) core controls the on-board ZBT memory chip to store/load instructions and data that require larger capacity than what is available from the on-chip BRAM. The core can be instantiated with the base system by ensuring that ZBT_512Kx32 is selected with the UARTLITE option in the Configure Additional IO Interfaces screen of the Xilinx Platform Studio Base System Builder Wizard used in module m01. After instantiating the core, add the relevant LOC constraints for the ZBT memory chip in the system.ucf file. The constraints file for the system is shown in the appendix.

2.3.3.8. SVGA

The SVGA controller is based on the reference design from Xilinx[10]. The particular type of controller used in this project is the character mode SVGA with colour support.

Instead of specifying the state of each pixel as in a bitmap mode, the screen is broken down into areas of 8x8 pixels. In each area, a previously defined character can be drawn. The defined characters are stored in a read-only memory block. The character to be drawn in each location is indicated by writing a value to the corresponding location in the character ram.

With the colour support, the active pixels in each character area can be assigned a colour; this colour is the same for all the pixels in the area. The colour of each area can be set by writing the appropriate value to the location in the colour video memory.

In our system, the SVGA controller is attached to the OPB bus. The video memory is mapped to an address range and thus is always accessible for writing.

The output of the controller drives the pins of the FPGA that are connected to a triple 8-bit video DAC on the Multimedia Board. The pin connections are specified in the system.ucf file found in the appendix.

2.3.4. Software Blocks

All the hardware IP blocks are controlled by the designer using software embedded on the MicroBlaze processor. Xilinx provides low level software drivers that the designer can use to code the higher level software of the system. Complete documentation of these low level functions, data structures and software components can be found in the Xilinx Device Driver Documentation[11].

2.3.4.1. Ethernet

The EMAC interfaces with the MicroBlaze by means of low level software drivers provided by Xilinx. These Ethernet drivers provide functions to control the EMAC using software programmed by the designer. The description of these drivers can be found in the EMAC section of the Xilinx Device Driver Documentation. These drivers facilitate the software coding needed to control the sending and receiving of Ethernet frames from board to board in a reliable and timely fashion. The control software was adapted from a project completed by Patrick Akl from the American University of Beirut for A Reliable Communication System for Xilinx MicroBlaze FPGA Systems[12]. Finally, the software system contains a custom designed protocol to standardize communication between each board. This protocol identifies frame types before sending and after receiving to ensure correct processing of data communicated between server and client machines.

2.3.4.2. Poker Gameplay

The game in our system proceeds upon a server/client protocol. One server board is responsible for all the administrative aspects of the poker game while the other board acts as a client and solely relies on the server for the necessary information, such as the cards dealt for the hand. All the intelligence for the game is run on the server side and communicated to the client machine via the Ethernet.

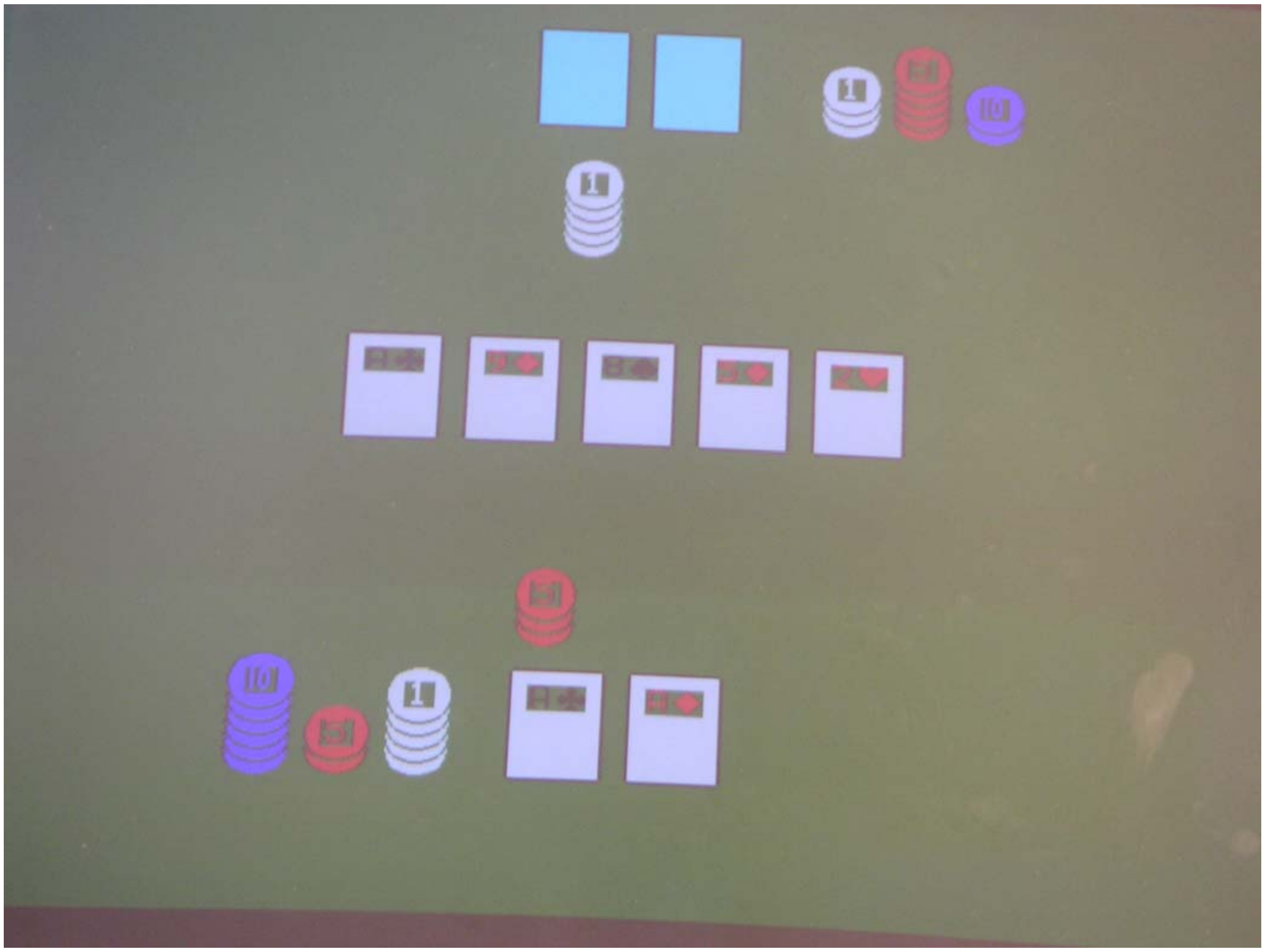
2.3.4.3. SVGA

To facilitate the ease of drawing characters onto the SVGA monitor, several custom functions were created. These functions draw characters by writing directly to the video memory of the SVGA controller.

3. Outcome

3.1. Results

Figure 3-1 SVGA Display



The results of the project were excellent. The hardware and software systems were built and tested separately and integrated successfully to create a fun and interactive network poker game with graphics similar to the figure above.

Players start with $100 each and can raise a maximum of $9 each round. Randomization is achieved through a counter that runs upon server startup and stops when a client connection is received. This counter value is used as the seed for the random number generator for the game. The opposing player's cards are face down to maintain ambiguity and the board cards are displayed in the centre of the screen to both players. Players can call, check, raise or fold each round by entering the appropriate letter when prompted at the XILINXPORT window as well as in the GUI. Player bets are shown with chip stacks in front of their cards and the current pot is displayed with a chip stack to the right side of the board cards. The program quits when one player ends a game with $0 cash and cannot meet the $1/$2 blinds.

The network communication of 10Mbps with error checking and recovery was adequate for the game and provided reliable and fast enough data transfer to support all the gaming needs. The character mode SVGA was sufficient for drawing cards and chips since the graphics do not need great detail and speed. Cards and chips are easier to draw and do not require external memory for storage of each pixel.

3.2. Improvements & Future Work

Some suggested enhancements to this system are as follows:

1. Greater than two network players/boards

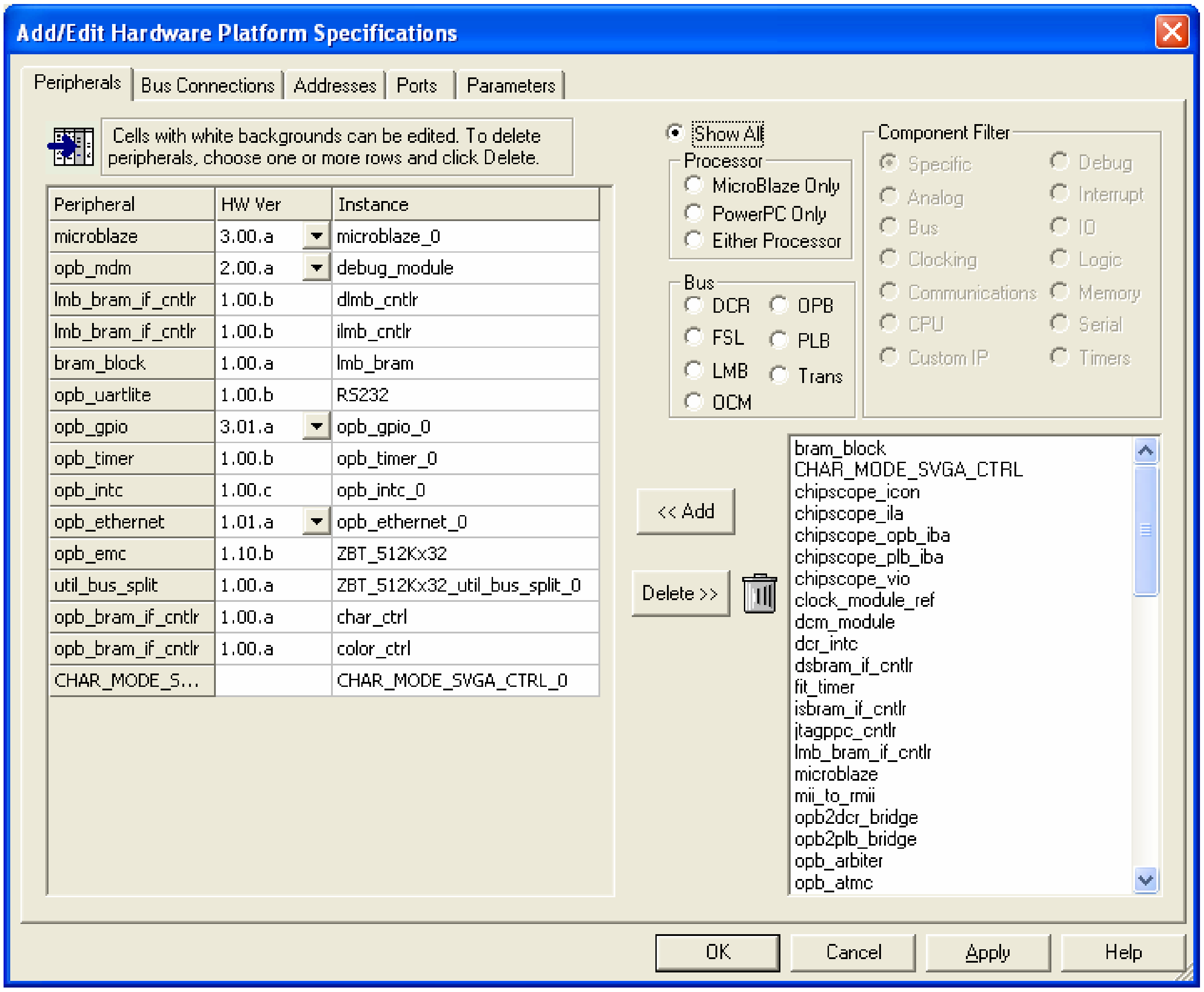
- Connect more than two boards to the hub and modify server to receive and deal multiple clients with different MAC addresses

2. Keyboard Input
   - Completely isolate system with keyboard input directly into board PS2 port
3. Sound
   - Introduce sound effects of shuffling, dealing, betting
4. Higher Stakes
   - Allow no limit betting
   - Have a larger pot and player cash
5. Faster Communication
   - Increase speed of Ethernet communications from 10Mbps to 100Mbps
6. Player Interaction
   - WebCam over the network to view poker faces, will require faster communication
   - Allow text messages or speech across network, will require faster communication
7. Better Graphics
   - Less blinking between turns because of redrawing entire screen
   - Use bitmapped SVGA over character mode SVGA for better graphics
8. Save Games
   - Use ZBT memory to save game states for retrieval later

4. Description of Blocks

4.1. Hardware Blocks

Figure 4-1 Hardware IP Cores



Add/Edit Hardware Platform Specifications

Peripherals | Bus Connections | Addresses | Ports | Parameters

Cells with white backgrounds can be edited. To delete peripherals, choose one or more rows and click Delete.

Peripheral	HW Ver	Instance
microblaze	3.00.a	microblaze_0
opb_mdm	2.00.a	debug_module
lmb_bram_if_cntlr	1.00.b	dlmb_cntlr
lmb_bram_if_cntlr	1.00.b	ilmb_cntlr
bram_block	1.00.a	lmb_bram
opb_uartlite	1.00.b	RS232
opb_gpio	3.01.a	opb_gpio_0
opb_timer	1.00.b	opb_timer_0
opb_intc	1.00.c	opb_intc_0
opb_ethernet	1.01.a	opb_ethernet_0
opb_emc	1.10.b	ZBT_512Kx32
util_bus_split	1.00.a	ZBT_512Kx32_util_bus_split_0
opb_bram_if_cntlr	1.00.a	char_ctrl
opb_bram_if_cntlr	1.00.a	color_ctrl
CHAR_MODE_S...		CHAR_MODE_SVGA_CTRL_0

[✓ Show All]

Processor: MicroBlaze Only, PowerPC Only, Either Processor

Bus: DCR, OPB, FSL, PLB, LMB, Trans, OCM

Component Filter: [✓ Specific], Debug, Analog, Interrupt, Bus, IO, Clocking, Logic, Communications, Memory, CPU, Serial, Custom IP, Timers

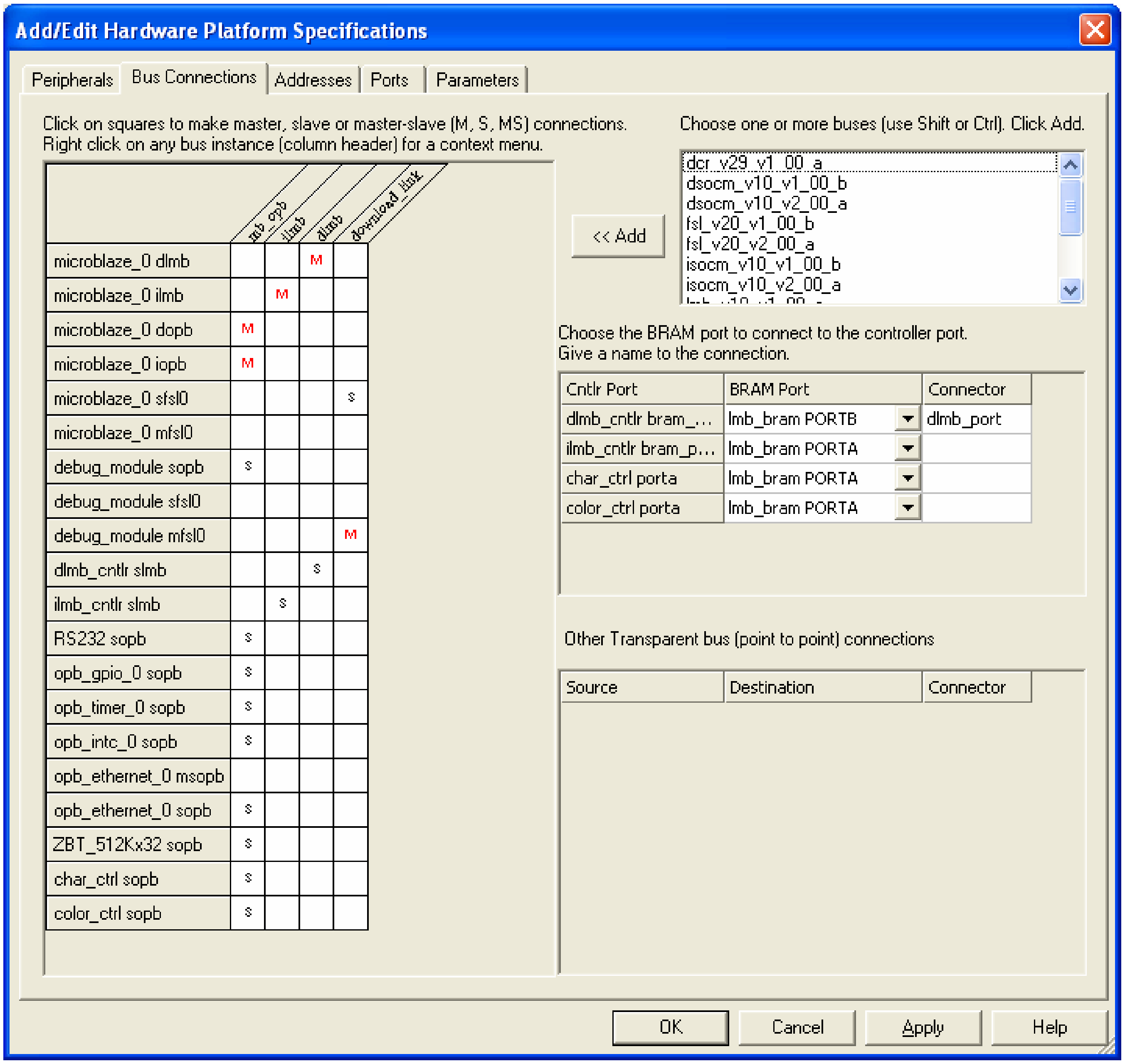
<< Add

Delete >>

bram_block
CHAR_MODE_SVGA_CTRL
chipscope_icon
chipscope_ila
chipscope_opb_iba
chipscope_plb_iba
chipscope_vio
clock_module_ref
dcm_module
dcr_intc
dsbram_if_cntlr
fit_timer
isbram_if_cntlr
jtagppc_cntlr
lmb_bram_if_cntlr
microblaze
mii_to_rmii
opb2dcr_bridge
opb2plb_bridge
opb_arbiter
opb_atmc

OK | Cancel | Apply | Help

Figure 4-2 Hardware Bus Connections



Add/Edit Hardware Platform Specifications

Peripherals | Bus Connections | Addresses | Ports | Parameters

Click on squares to make master, slave or master-slave (M, S, MS) connections.
Right click on any bus instance (column header) for a context menu.

	mb_opb	ilmb	dlmb	download_link
microblaze_0 dlmb			M	
microblaze_0 ilmb		M		
microblaze_0 dopb	M			
microblaze_0 iopb	M			
microblaze_0 sfsl0				S
microblaze_0 mfsl0				
debug_module sopb	S			
debug_module sfsl0				
debug_module mfsl0				M
dlmb_cntlr slmb			S	
ilmb_cntlr slmb		S		
RS232 sopb	S			
opb_gpio_0 sopb	S			
opb_timer_0 sopb	S			
opb_intc_0 sopb	S			
opb_ethernet_0 msopb				
opb_ethernet_0 sopb	S			
ZBT_512Kx32 sopb	S			
char_ctrl sopb	S			
color_ctrl sopb	S			

Choose one or more buses (use Shift or Ctrl). Click Add.

<< Add

dcr_v29_v1_00_a
dsocm_v10_v1_00_b
dsocm_v10_v2_00_a
fsl_v20_v1_00_b
fsl_v20_v2_00_a
isocm_v10_v1_00_b
isocm_v10_v2_00_a

Choose the BRAM port to connect to the controller port.
Give a name to the connection.

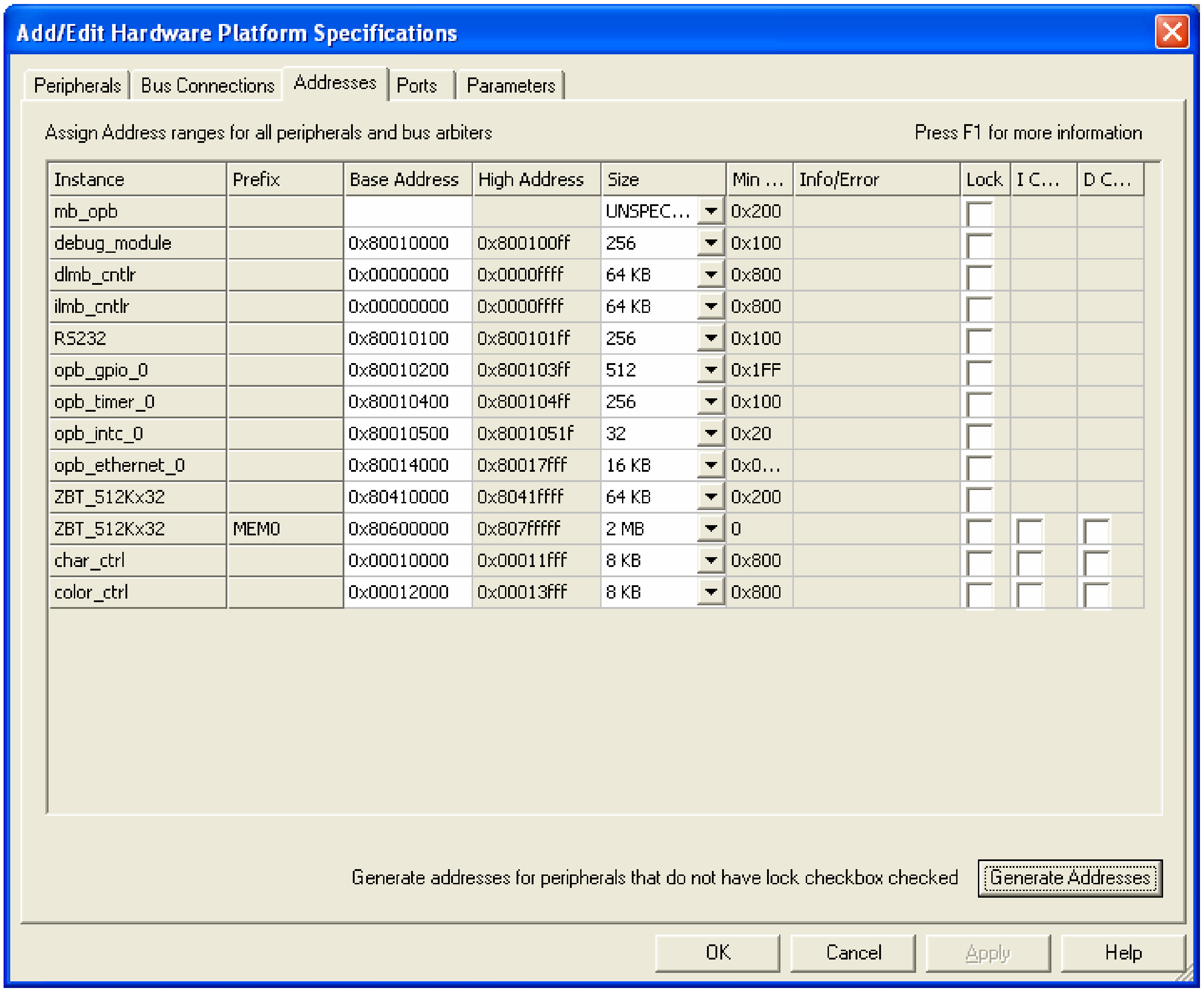
Cntlr Port	BRAM Port	Connector
dlmb_cntlr bram_...	lmb_bram PORTB	dlmb_port
ilmb_cntlr bram_p...	lmb_bram PORTA	
char_ctrl porta	lmb_bram PORTA	
color_ctrl porta	lmb_bram PORTA	

Other Transparent bus (point to point) connections

Source	Destination	Connector

OK | Cancel | Apply | Help

Figure 4-3 Hardware Memory Map



Assign Address ranges for all peripherals and bus arbiters

Press F1 for more information

Instance	Prefix	Base Address	High Address	Size	Min ...	Info/Error	Lock	I C...	D C...
mb_opb				UNSPEC...	0x200		☐		
debug_module		0x80010000	0x800100ff	256	0x100		☐		
dlmb_cntlr		0x00000000	0x0000ffff	64 KB	0x800		☐		
ilmb_cntlr		0x00000000	0x0000ffff	64 KB	0x800		☐		
RS232		0x80010100	0x800101ff	256	0x100		☐		
opb_gpio_0		0x80010200	0x800103ff	512	0x1FF		☐		
opb_timer_0		0x80010400	0x800104ff	256	0x100		☐		
opb_intc_0		0x80010500	0x8001051f	32	0x20		☐		
opb_ethernet_0		0x80014000	0x80017fff	16 KB	0x0...		☐		
ZBT_512Kx32		0x80410000	0x8041ffff	64 KB	0x200		☐		
ZBT_512Kx32	MEM0	0x80600000	0x807fffff	2 MB	0		☐	☐	☐
char_ctrl		0x00010000	0x00011fff	8 KB	0x800		☐	☐	☐
color_ctrl		0x00012000	0x00013fff	8 KB	0x800		☐	☐	☐

Generate addresses for peripherals that do not have lock checkbox checked

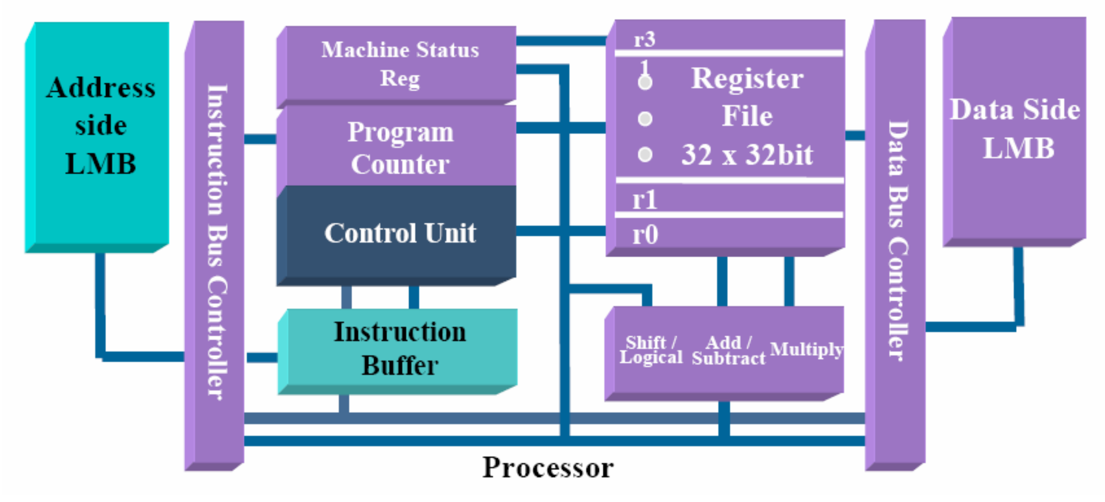
4.1.1. Processor

The Xilinx MicroBlaze soft processor IP core is central to the SVGA Network Poker system. It is a 32-bit RISC processor clocked by the 27MHz clock crystal on the Xilinx Multimedia Board. The MicroBlaze includes the following features[13]:

- Thirty-two 32-bit general purpose RAM based registers with separate instructions for data and memory access
- Separate instruction (ILMB) and data (DLMB) local memory buses
- Built-in interfaces to fast on-chip memory and to IBM's industry-standard On-chip Peripheral Bus (OPB)
- Support for both on-chip BRAM and external memory

The MicroBlaze core is shown in the figure below. It controls the game-play, I/O and communication over the Ethernet with embedded software and Xilinx drivers stored in its instruction/data memory. This software controls the hardware cores in the system.

Figure 4-4 MicroBlaze Processor Core



4.1.2. Ethernet

An Ethernet interface is established using the Ethernet Medium Access Controller (EMAC) provided as a Xilinx IP core[14]. This core can support 32-bit master or slave interfaces on the OPB. The connection speed is 10 Mbps. It supports the IEEE Std. 802.3 Media Independent Interface (MII) to industry standard Physical Layer (PHY) devices. The MII of the EMAC connects to an external 10/100 physical interface (PHY) transceiver. The physical interface is created using a LevelOne LXT972 3.3 Volt PHY[15]. It is an IEEE-compliant Fast Ethernet transceiver that directly supports both 100BASE-TX and 10BASE-T applications.

4.1.3. PC to Board Communication

I/O of the system consists of serial communication with the processor using a Xilinx IP Universal Asynchronous Receiver/Transmitter (UART) core[16] interfaced on the OPB. This core controls the off chip RS232 serial port directly connected to the serial interface of a PC. This can be used as the main communication for game play and game display (using ASCII characters).

4.1.4. Timer

The TC (Timer/Counter) is a 32-bit timer module that attaches to the OPB with byte-enable support[17]. It has two programmable interval timers with interrupt generation capabilities and configurable counter width. The timer is used to timeout acknowledge frames not received by the EMAC after a frame has been sent out across the network. This interrupt causes the interrupt controller to resend the frame until an acknowledge message is received. If no acknowledge message is received after 10 unsuccessful attempts the system will notify the user of a send error.

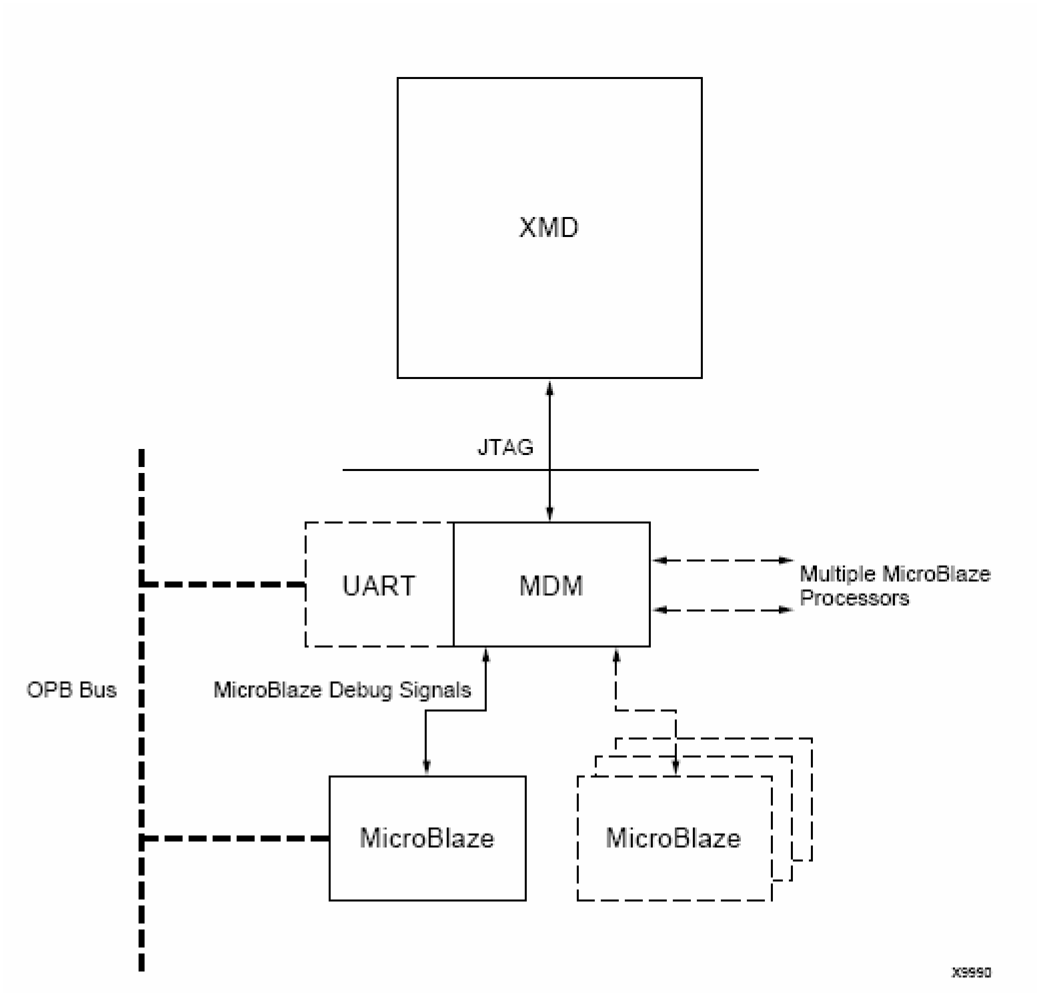
4.1.5. Interrupt Controller

The Interrupt Controller core is a simple, parameterized interrupt controller that attaches to the OPB (On-chip Peripheral Bus)[18]. The controller connects to the Timer and EMAC IP cores for interrupt driven event handling. The timer interrupts with acknowledge timeouts indicating Ethernet frame error and the need for retransmission. The EMAC interrupts with incoming Ethernet frames that require processing for game-play.

4.1.6. Debug Module

A MicroBlaze Debug Module (MDM) is included as a slave peripheral on the OPB. This is a Xilinx IP used to access the internal structure of the processor for debugging and troubleshooting during the design and testing phase. The MDM interfaces with the Xilinx MicroBlaze Debugger (XMD) software debug stub stored in the FPGA BRAM. The XMD interface can be used for command line control and debugging of the MicroBlaze as well as for running complex verification test scripts to test the complete system. This MDM can be interfaced with GNU debugger (GDB) for efficient debugging of software embedded onto the processor.

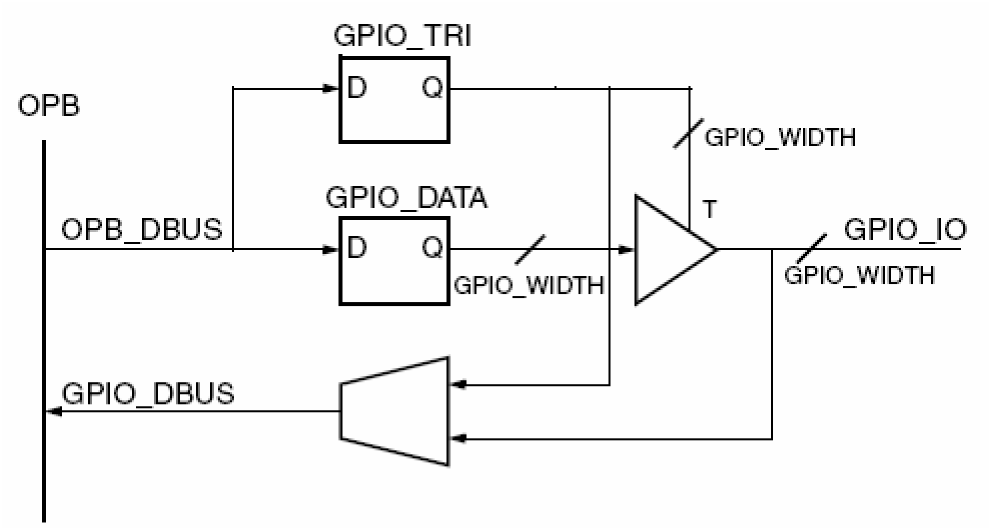
Figure 4-5 Microblaze MDM Target[19]



4.1.7. GPIO

The General Purpose Input/Output (GPIO) core is connected to the OPB[20]. It has two input signals controlled by DIP switches on the Multimedia Board. SW0 is the manual active low reset for the MicroBlaze processor. SW1 is the manual active low reset for the SVGA monitor. The GPIO is a simple peripheral consisting of two registers and a multiplexer for reading register contents and the GPIO I/O signals.

Figure 4-6 GPIO Block Diagram

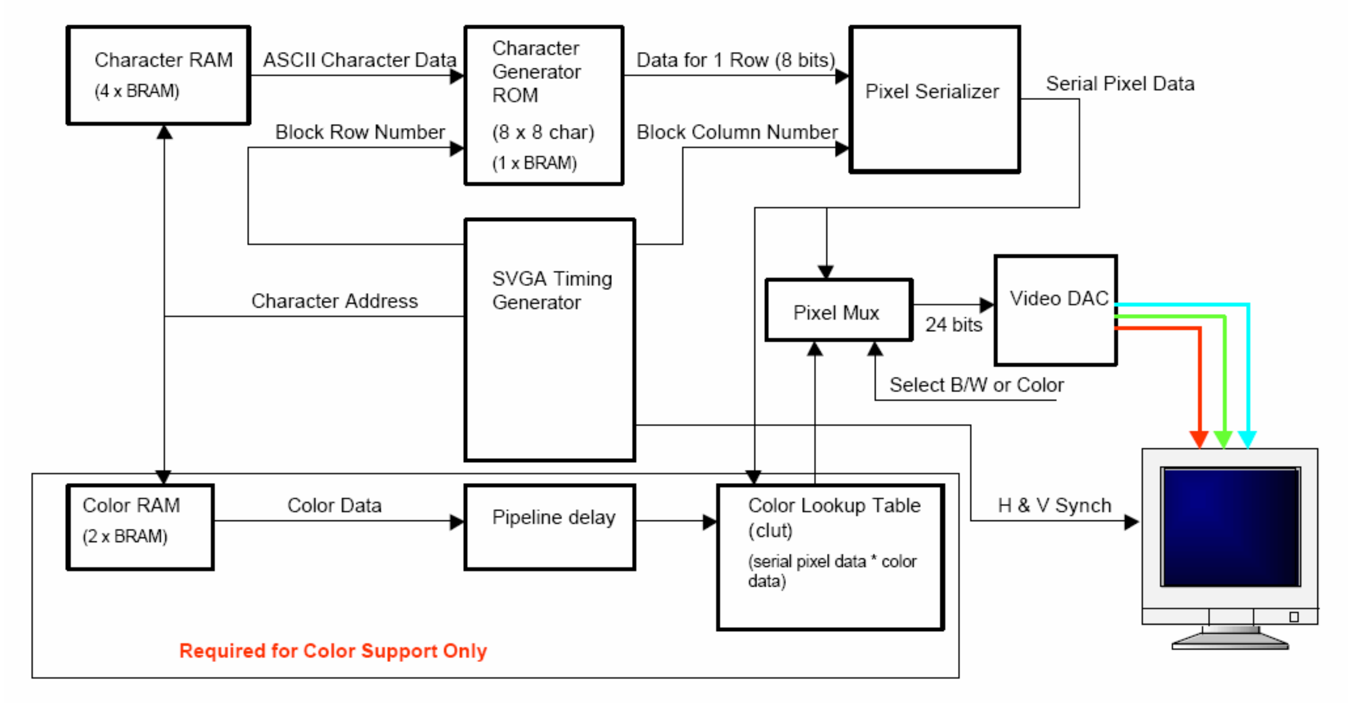


4.1.8. ZBT Memory

The External Memory Controller IP module supports data transfers between the On-chip Peripheral Bus (OPB) and external synchronous and asynchronous memory devices[21]. The EMC on the board connects to a Samsung K7N163601A 512Kx36 Pipelined N*t*RAM[22] for storage of all the software assembly code compiled by EDK for executing the Ethernet communications and poker gameplay. The pin-out from the FPGA to the memory bank can be found in the Multimedia Board User Guide. The LOC constraints are added to the system.ucf file shown in the appendix.

4.1.9. SVGA

Figure 4-7 Character Mode SVGA Block Diagram



4.1.9.1. Graphics Controller

The graphics controller in our project is adapted from the Xilinx example available referenced at the end of this document. The design comes in a Verilog format, thus the "Import Peripheral Wizard" was used to convert it to a core suitable for EDK. The outputs of the design are connected to the pins of the FPGA to allow it to drive the video DAC. See appendix for LOC constraints applied in the system.ucf file.

Modelsim simulations were used to test this module.

4.1.9.1.1 CHAR_MODE_SVGA_CTRL

This Verilog module comes from COLOR_CHAR_MODE_SVGA_CTRL.v in the original Xilinx example. The name was shortened because the

"Import Peripheral Wizard" of EDK limits the length of the name of cores. This module is the top level module for the controller. It instantiates all the following modules and makes the appropriate connections.

To connect it with the OPB, all address and data lines were increased to a width of 32 bits to match the bus. Only the least significant lines are connected to the relevant sub-modules. Also, to reduce the number of connections, the colour RAM write clock and the character RAM write clock are connected together.

The last change to this module was to disable the composite sync on green. The composite sync signal is intended for older analog monitors without separate vertical and horizontal sync inputs. However, we found that this feature caused some problems with our monitors and caused all black areas to be displayed as light green.

4.1.9.1.2 CHARACTER_MODE

This module comes directly from CHARACTER_MODE.v in the Xilinx example. Its purpose is to generate serial pixel data to drive the DAC. It also instantiates the character ROM module that contains the definition of available characters and the character video RAM module.

No changes were made from the original.

4.1.9.1.3 CHAR_GEN_ROM

This module is adapted from CHAR_GEN_ROM.v in the Xilinx reference design. It instantiates a 2k x 8 ROM using the Virtex-II's block RAM. This ROM stores all the characters that are available to be drawn. The ROM contents are specified inside the Verilog file using "defparam" statements.

For our project, the initial contents of the ROM were modified to include our own custom characters and graphics. Our characters are actually made up of 2x2 arrays of regular-sized characters put together to make larger, more visible characters. Also, the "//synthesis translate_off" command was removed to make XST properly process the "defparam" statements. Finally, the black-box module declaration for RAMB16_S9 was removed.

4.1.9.1.4 CHAR_RAM

This module is adapted from CHAR_RAM.v of the Xilinx design. It creates an 8k x 8 character video RAM from the block RAMs of the Virtex-II. The character to be drawn at each character location is determined by writing the address of the character in the character ROM to the corresponding memory location inside the video RAM.

Compared to the original module from Xilinx, the initial state of the RAM was changed to the different blank character address we had from changing the character ROM. Also, the "//synthesis translate_off" command was removed to make XST properly process the "defparam" statements. Finally, the black-box module declaration for RAMB16_S2_S2 was removed.

4.1.9.1.5 SVGA_TIMING_GENERATION

This module is derived from SVGA_TIMING_GENERATION.v of the Xilinx design example. This module is responsible for generating the correct timing information and control signals for the SVGA outputs.

In our project, the only revision we made is to directly include the timing parameters needed as opposed to including "svga_defines.v" like the original version. The parameters the project uses generate a resolution of 800x600 with a refresh rate of 72MHz.

4.1.9.1.6 COLOR_RAM

This module is from COLOR_RAM.v of the original Xilinx design. Its purpose is to create an 8k x 4 colour RAM from the Virtex-II block RAM's. The colour RAM stores the colour information for each character. Since each address location addresses 4 bits, only 16 colours can be used. All the active pixels in a character have the colour that is specified in the colour RAM. The initial default colour is white.

The "//synthesis translate_off" command was removed to make XST properly process the "defparam" statements. Also, the black-box module declaration for RAMB16_S4_S4 was removed.

4.1.9.1.7 COLOR_PIPE

This module is copied from COLOR_PIPE.v of the reference character mode design. It delays the colour data information to synchronize the character data information.

No changes were made to this module from the original.

4.1.9.1.8 CLUT

This module is based on CLUT.v of the Xilinx example. This module generates the colour information that drives the DAC based on the 4 bits from the colour RAM. The 16 colours implemented are based on the DOS 16 colour palette.

The only change made to this module from the original is to make all "inactive" pixels green. The reason for this was to create a green background to simulate a card table.

Table 4-1 SVGA Colours

Color	Index
Black	0
Navy	1
Green	2
Teal	3
Maroon	4
Purple	5
Olive	6
Silver	7
Gray	8
Blue	9
Lime Green	10
Cyan	11
Red	12
Magenta	13
Yellow	14
White	15

4.1.9.1.9 DRIVE_DAC_DATA

This module is duplicated from DRIVE_DAC_DATA.v of the Xilinx reference design. This module is used to pass the correct data to the DAC.

No changes were made to this module from the original.

4.1.9.2. OPB_BRAM_IF_CNTRL

To interface the graphics controller to the MicroBlaze processor, the opb_bram_if_cntrl, ver 1.00.a, is used. This allows the MicroBlaze access to the video RAM of the controller through the OPB. Since only the least significant bits of the data and address lines are used inside the SVGA controller, byte operations must be used.

In our system, two BRAM controllers are used: one to write to the character video RAM and one to write to the colour video RAM.

4.1.9.3. Video DAC

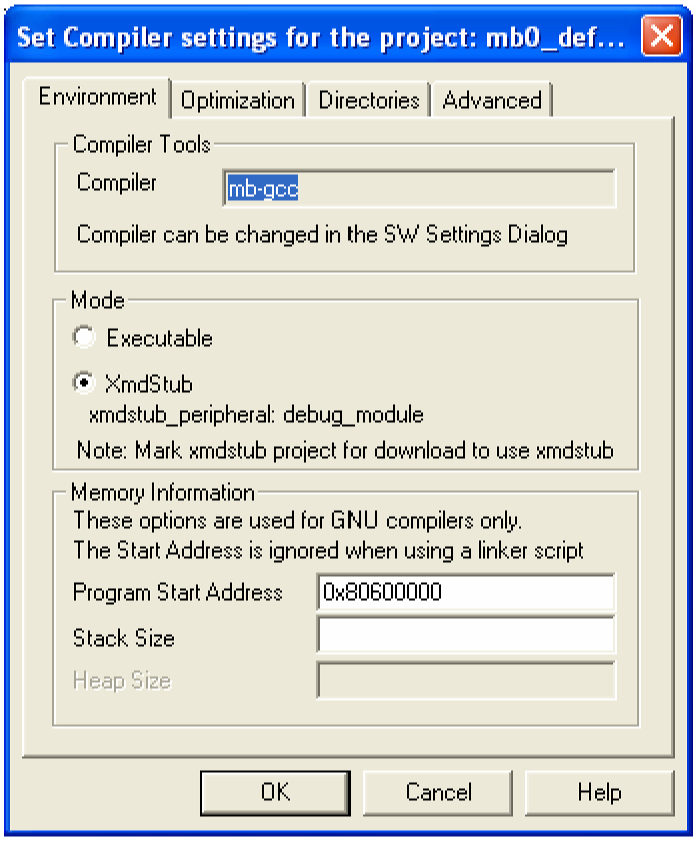
The Xilinx MicroBlaze Multimedia board contains a triple 8-bit video DAC for driving an SVGA monitor. The DAC converts the 8 bits of green, red and blue colour information into analog signals. It also provides the ability for composite sync on green for older monitors that require it.

4.2. Software Blocks

The XMDstub is used to download the software application into ZBT memory with the command: *dow mb0_default/executable.elf*

The software application must be built and compiled with the start address set to the first address of the ZBT bank (MEM0) in the hardware memory map. This can be done from the compiler options window in the applications tab of EDK. The software project in the applications tab must also not be marked to initialize BRAMs since the software code will be stored in external ZBT memory.

Figure 4-8 Compiler Options


Set Compiler settings for the project: mb0_def...

Compiler Tools
Compiler: mb-gcc
Compiler can be changed in the SW Settings Dialog

Mode
Executable [✓ XmdStub]
xmdstub_peripheral: debug_module
Note: Mark xmdstub project for download to use xmdstub

Memory Information
These options are used for GNU compilers only.
The Start Address is ignored when using a linker script
Program Start Address: 0x80600000
Stack Size:
Heap Size:

4.2.1. Ethernet

The Ethernet drivers embedded into the processor along with the game communication functions will define and initialize each system as a server or client depending on the software downloaded on each machine.

A network protocol analyzer called Ethereal[23] was used to monitor network traffic and Ethernet frames between each board. Ethereal can be installed on any PC and must be connected to the hub between each board. This analyzer can help to debug frame contents in transit.

4.2.1.1. Ethernet Drivers

The Ethernet drivers for the EMAC come as part of Xilinx embedded drivers for software application involving network communications. A detailed description of these drivers can be found in the Xilinx Device Driver Documentation.

Each board contains a unique 48-bit serial number that can be used as the MAC address for the board. The EMAC also contains a hard coded MAC Address in a register and can be obtained with a special procedure. However, the controller also has a configurable MAC Address that is used for user convenience. The MAC address for each board can be set in the SingleMB_MACAddresses.h file by changing the LocalAddress variable to the desired MAC address. The EMAC is initialized with low level drivers in the SingleMB_IntializationFunctions.c file. It is configured for automatic insertion of some Ethernet Header and Trailer fields such as the CRC and the source MAC Address. The MAC address is also configured here by running the driver function XEmac_SetMacAddress(EmacPtr, LocalAddress).

4.2.1.2. Ethernet Control Protocol

The sending and receiving of frames using the layer 0 Xilinx drivers are not enough to ensure successful transmission of data from one board to the next. In case a frame is lost, the lower level protocol does not retransmit or inform of an errors. In addition, the receiver cannot distinguish between duplicate frames or frames which are received out of order. The layer 0 drivers can only support Ethernet frames of maximum 1500 bytes in size so there is a need for fragmentation and reassembly for messages larger than this limit. The need for flow control and higher payload size requires the use of a higher layers (layer 1 and 2) Ethernet protocol in the system. These layers are responsible for full control over sending and receiving frames correctly and in order. Tables 4-1 to 4-3 outline the structure of Ethernet frames at these higher layers. These layers are directly adapted from the project completed by Patrick Akl for a simple reliable communication system for Xilinx FPGAs. Detailed descriptions of the layer 1 and 2 control protocol can be found in his user guide referenced at the end of this report.

Table 4-2 Layer 1 Ethernet Frame

Destination Address	Source Address	Frame Length	Payload	Frame Check Sequence
6 bytes	6 bytes	2 bytes	46 to 1500 bytes (auto padding)	4 bytes (auto insertion by EMAC)

Table 4-3 Layer 2 Ethernet Frame

Frame Type	Sequence Number	Message Length	Fragment Payload
2 bytes	4 bytes	4 bytes	Any size less than 2^{32} bytes

Table 4-4 Layer 2 Frame Types

Frame Type	Meaning
0	Non Fragmented Message
1	ACK Message
2	First Fragment
3	Internal Fragment
4	Last Fragment

4.2.1.3. Poker Gameplay Protocol

The poker game communication needs another protocol on top of the lower level layers to establish a standard between server and client. This layer 3 protocol is custom designed for the purposes of game coordination. Table 4-4 outlines the basic types of poker frames sent from server to client and vice versa.

- Cards Frame: sent from server to client that holds each players cards in hand as well as the 5 board cards
- Cash Frame: sent from server to client that holds the cash balance of each player and pot amount during each round
- Turn Frame: specifies to client the game status (pre-flop, flop, turn, river and etc.)
- DoBet Frame: sent by server to client to request a bet from the client machine for the current round, hold value of maximum current bet on the table to indicate whether the client can just check or must call to stay in the game
- BetValue Frame: sent from client to server to establish what bet the client made after getting a DoBet frame from the server
- Status Frame: sent from server to client to inform client user of choices and outcomes made by server machine
- Connect Frame: sent from client to server to establish a connection and deal the first hand of poker

The SingleMB_HandlerFunctions.c file contains the FifoRecvHandler function that is the interrupt service routine run when the EMAC interrupts the processor with a received frame. User code has been added here in the form of a case statement to determine the poker protocol frame type and process it accordingly.

The file send.c contains the corresponding send functions for each poker protocol frame type.

Table 4-5 Layer 3 Frame Types

Frame Type	1st element	2nd element	3rd element	Frame Type
1	Player1 cards	Player2 cards	Board Cards	Cards Frame
2	Player1 cash	Player2 cash	Pot Value	Cash Frame
3	Turn Integer	-	-	Turn Frame
4	Maximum Current Bet	-	-	DoBet Frame
5	Player2 Bet Amount	-	-	BetValue Frame
6	Status Message	Player Number	Amount	Status Frame
7	-	-	-	Connect Frame

Table 4-6 Layer 3 Turn Type

Frame Type	Meaning
0	Pre Flop
1	Flop
2	Turn
3	River
4	Round over, show all cards

Table 4-7 Layer 3 Status Message Type

Frame Type	Meaning
0	Player Calls/Checks
1	Player Raises
2	Player Wins
3	Player Folds

4.2.2. Poker Gameplay

The entire game is controlled by a custom created C code. Since the game operates on a server/client connection, the server is responsible for managing all the various aspects of the game-play, while the client simply displays the appropriate information as instructed by the server.

4.2.2.1. Server

- server_main.c: This file contains the main function for running the game. It first initializes the Ethernet connection by calling the appropriate Ethernet driver functions. It runs the game by continuously looping through dealing the cards, getting bets, showing more cards and determining a winner. It also calls upon the functions in send.c for sending the appropriate information to the client at the correct times.
- dealholdem.c, shuffle.c, fastran2.c: Together, these files deal the cards for a single round. The cards are shuffled by calling on the functions in shuffle.c and dealholdem.c assigns random cards to each player and as community cards. Randomization is done by calling fastran2.c.
- betting.c: This file contains functions related to betting, such as removing the blinds from the two players left of the button, and prompting for a bet from the player.
- getCombos.c, getf.c, getstr7s.c: These files contain the functions necessary for evaluating hands to determine the type of hand each player has made.
- printHands.c: This file contains functions for displaying the status of the game. It uses the SVGA drivers to draw information on the SVGA monitor as well as displaying information to the terminal.
- winners.c: This file contains functions for determining the winner or winners and then assigning the cash properly.

4.2.2.2. Client

- client_main.c: This file contains the main() function for running the client side of the game. It simply waits in a while loop and displays new information using printHands.c whenever information is received from the server.

4.2.3. SVGA Drivers

4.2.3.1. display.h

This header file contains define statements and other initialization statements that contains the following information:

- the height and width of the screen in terms of characters of 8x8 pixels,
- the starting address as defined in the character ROM of each character,
- the index of the colours, and
- the location on the screen to draw the cards, chip stacks, names and amounts.

4.2.3.2. display.c

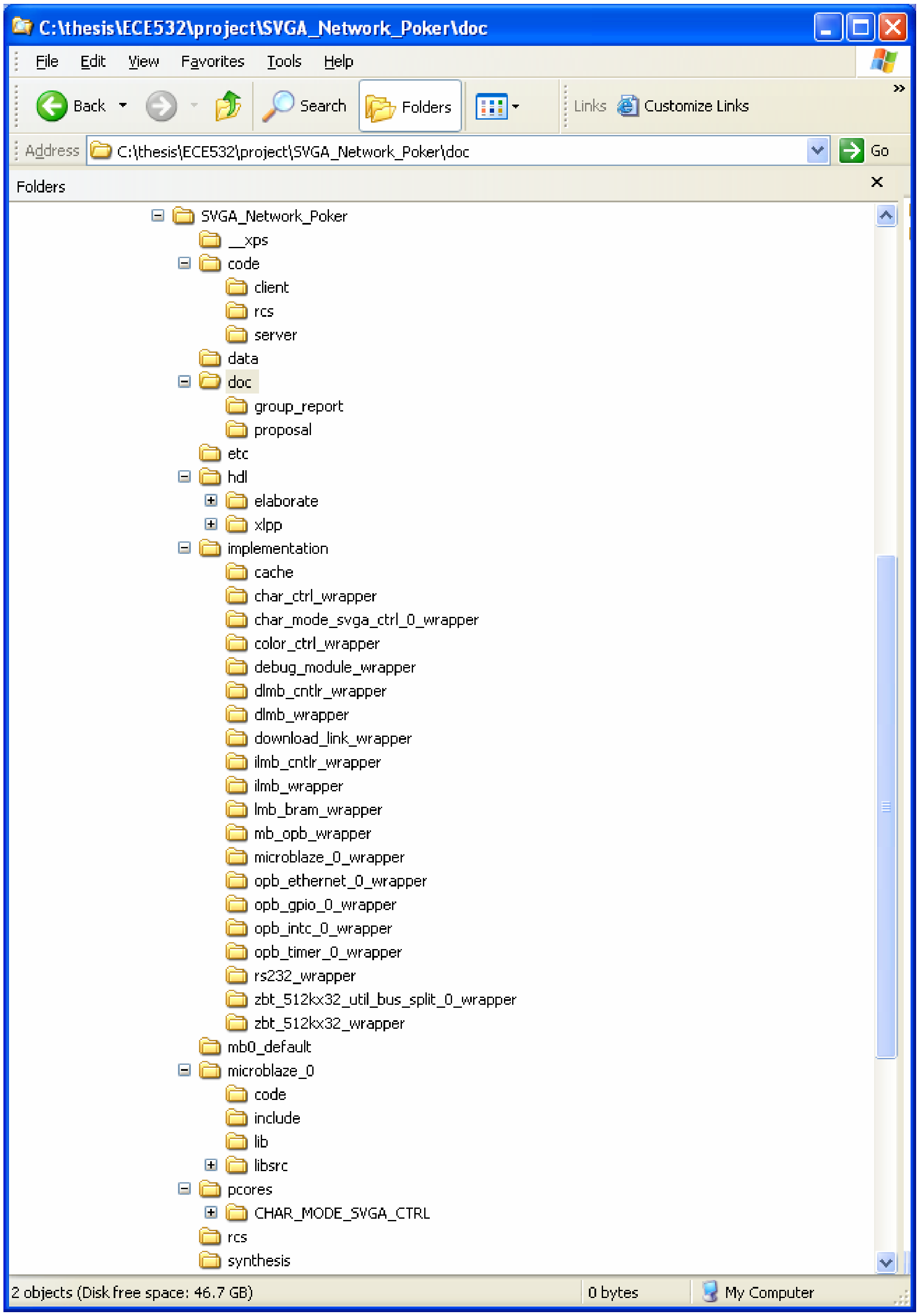
This file contains functions to draw characters onto the screen by writing into the memory RAM. At the lower level, the drawChar() and drawBigChar functions simply draws a 1x1 character or a 2x2 character onto the screen at the specified row and column with the specified colour by writing to memory. The other functions call on these two functions to draw more sophisticated items:

- drawBlankScreen() clears the screen by drawing a blank character at all locations.
- drawCard() draws a card at the specified row and column given a card value and a card suit.

- drawChipStack() uses the stack characters to draw a single stack of chips at the specified location with the specified value and colour.
- drawStacks() takes an integer value and draws three stacks of chips of different values at the specified location using drawChipStack().
- drawText() draws a "\n" terminated string at the specified location with the specified colour. For our project, only capital letters and numbers are implemented in the character ROM so only these can be drawn.
- drawInt() is similar to drawText() except it takes an integer as its input.

5. Design Tree

Figure 5-1 Design Tree


C:\thesis\ECE532\project\SVGA_Network_Poker\doc
File Edit View Favorites Tools Help
Back
Search
Folders
Links
Customize Links
Address
C:\thesis\ECE532\project\SVGA_Network_Poker\doc
Go
Folders

```
SVGA_Network_Poker
    __xps
    code
        client
        rcs
        server
    data
    doc
        group_report
        proposal
    etc
    hdl
        elaborate
        xlpp
    implementation
        cache
        char_ctrl_wrapper
        char_mode_svga_ctrl_0_wrapper
        color_ctrl_wrapper
        debug_module_wrapper
        dlmb_cntlr_wrapper
        dlmb_wrapper
        download_link_wrapper
        ilmb_cntlr_wrapper
        ilmb_wrapper
        lmb_bram_wrapper
        mb_opb_wrapper
        microblaze_0_wrapper
        opb_ethernet_0_wrapper
        opb_gpio_0_wrapper
        opb_intc_0_wrapper
        opb_timer_0_wrapper
        rs232_wrapper
        zbt_512kx32_util_bus_split_0_wrapper
        zbt_512kx32_wrapper
    mb0_default
    microblaze_0
        code
        include
        lib
        libsrc
    pcores
        CHAR_MODE_SVGA_CTRL
    rcs
    synthesis
```

2 objects (Disk free space: 46.7 GB) 0 bytes My Computer

Table 5-1 Design Tree Description

Directory/File	Description
./_xps	Option files for bitinit, libgen, simgen and platgen
./code/client	Software code for client system
./code/server	Software code for server system
./code/rcs	Backup of original Ethernet code
./data	Constraints file located here
./doc	Documentation files
./etc	Option files for bitgen and downloading
./hdl	Verilog and VHDL files for the hardware system
./implementation	Synthesis, NGD, Map, PAR files directory
./mb0_default	Software executable.elf file to be downloaded to ZBT memory located here
./microblaze_0	Software libraries and drivers
./pcores	Custom core directory containing the character mode SVGA hardware IP core
./rcs	Archives of original Ethernet communication hardware system
./synthesis	Intermediary include files, scripts, logs and project files for synthesis
./system.xmp	XPS project file
./system.mhs	Microblaze Hardware Specification file
./system.mss	Microblaze Software Specification file
./readme	Contains description of design tree
./system.make	System Makefile
./data/system.ucf	System Constraints File

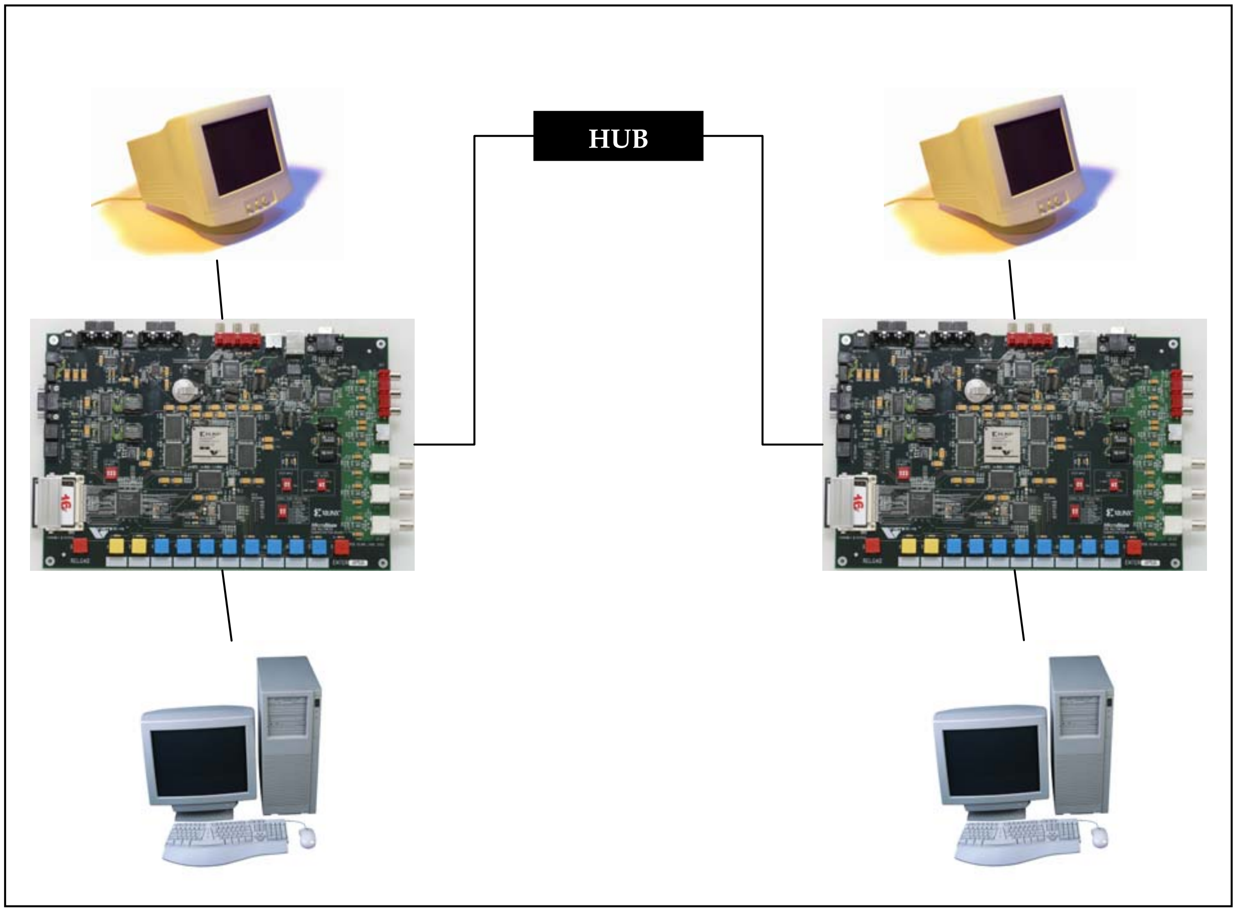
6. System Setup

6.1. Equipment Needed

- 2 Xilinx Multimedia Boards
- 1 10Mbps Ethernet hub with at least 2 ports
- 2 SVGA Monitors
- 2 PC workstations with EDK6.3 installed
- 2 RJ45 Ethernet cables

6.2. Hardware Setup Procedure

Figure 6-1 Hardware Setup



6.3. Software Setup Procedure

- Open system.xmp in EDK on the two different PC workstations
- Download each system onto its respective board
- Go to applications tab and change source file in Project: mb0_default to the server or client directory server_main.c or client_main.c file depending on which system will be the client and which system will be the server
- Build and compile the software for each system (ignore warnings)
- Run XMD
- Download software onto each board with command:
  dow mb0_default/executable.elf
- Open XILINXPORT window on each machine and connect to COM1
- Run server machine first and then run client machine next
- Game will commence

7. Conclusion

With silicon technologies shrinking every two years as predicted by Moore's Law, more and more hardware can be packed into a single FPGA. Large-scale systems on a programmable chip can be used for many embedded applications with the flexibility to change the design in the field. Having good EDA tools is key to ensuring the success of a fully embedded hardware and software system. Xilinx EDK and ISE are examples of such tools and were used extensively to create the SVGA Network Poker system. Following good modular design practices and flows the system was built and integrated successfully to produce a fun network game. Like all systems there is much room for future work and improvements. However, this project provides a solid base and background into developing large complex applications and hardware composed of high-density logic.

Appendix

System.ucf

```
Net sys_clk PERIOD = 37037 ps;

Net RS232_RX LOC=C8;
Net RS232_TX LOC=C9;
Net RS232_req_to_send LOC=B8;

Net sys_clk LOC=AH15;
Net sys_rst LOC=AH7;
Net sys_rst TIG;

Net opb_gpio_0_GPIO_IO<1> LOC=B27;
Net opb_gpio_0_GPIO_IO<0> LOC=D10;

Net PHY_slew1 LOC=G16;
Net PHY_slew2 LOC=C16;
Net opb_ethernet_0_PHY_crs LOC=F20;
Net opb_ethernet_0_PHY_col LOC=C23;

Net opb_ethernet_0_PHY_tx_data<3> LOC=C22;
Net opb_ethernet_0_PHY_tx_data<2> LOC=B20;
Net opb_ethernet_0_PHY_tx_data<1> LOC=B21;
Net opb_ethernet_0_PHY_tx_data<0> LOC=G20;

Net opb_ethernet_0_PHY_tx_en LOC=G19;
Net opb_ethernet_0_PHY_tx_clk LOC=H16;
Net opb_ethernet_0_PHY_tx_er LOC=D21;
Net opb_ethernet_0_PHY_rx_er LOC=D22;
Net opb_ethernet_0_PHY_rx_clk LOC=C17;
Net opb_ethernet_0_PHY_dv LOC=B17;

Net opb_ethernet_0_PHY_rx_data<0> LOC=B16;
Net opb_ethernet_0_PHY_rx_data<1> LOC=F17;
Net opb_ethernet_0_PHY_rx_data<2> LOC=F16;
Net opb_ethernet_0_PHY_rx_data<3> LOC=D16;

Net opb_ethernet_0_PHY_Mii_clk LOC=D17;
Net opb_ethernet_0_PHY_Mii_data LOC=A17;

Net ZBT_512Kx32_Mem_A<29> LOC=T23;
Net ZBT_512Kx32_Mem_A<29> FAST;
Net ZBT_512Kx32_Mem_A<28> LOC=U23;
Net ZBT_512Kx32_Mem_A<28> FAST;
Net ZBT_512Kx32_Mem_A<27> LOC=AB29;
Net ZBT_512Kx32_Mem_A<27> FAST;
Net ZBT_512Kx32_Mem_A<26> LOC=AA29;
Net ZBT_512Kx32_Mem_A<26> FAST;
Net ZBT_512Kx32_Mem_A<25> LOC=AA27;
Net ZBT_512Kx32_Mem_A<25> FAST;
Net ZBT_512Kx32_Mem_A<24> LOC=AB27;
Net ZBT_512Kx32_Mem_A<24> FAST;
Net ZBT_512Kx32_Mem_A<23> LOC=H25;
Net ZBT_512Kx32_Mem_A<23> FAST;
Net ZBT_512Kx32_Mem_A<22> LOC=G25;
Net ZBT_512Kx32_Mem_A<22> FAST;
Net ZBT_512Kx32_Mem_A<21> LOC=G28;
Net ZBT_512Kx32_Mem_A<21> FAST;
Net ZBT_512Kx32_Mem_A<20> LOC=H29;
Net ZBT_512Kx32_Mem_A<20> FAST;
Net ZBT_512Kx32_Mem_A<19> LOC=U27;
Net ZBT_512Kx32_Mem_A<19> FAST;
Net ZBT_512Kx32_Mem_A<18> LOC=T27;
Net ZBT_512Kx32_Mem_A<18> FAST;
Net ZBT_512Kx32_Mem_A<17> LOC=V29;
Net ZBT_512Kx32_Mem_A<17> FAST;
Net ZBT_512Kx32_Mem_A<16> LOC=U29;
Net ZBT_512Kx32_Mem_A<16> FAST;
Net ZBT_512Kx32_Mem_A<15> LOC=T24;
Net ZBT_512Kx32_Mem_A<15> FAST;
Net ZBT_512Kx32_Mem_A<14> LOC=T25;
Net ZBT_512Kx32_Mem_A<14> FAST;
Net ZBT_512Kx32_Mem_A<13> LOC=U28;
Net ZBT_512Kx32_Mem_A<13> FAST;
Net ZBT_512Kx32_Mem_A<12> LOC=F28;
Net ZBT_512Kx32_Mem_A<12> FAST;
Net ZBT_512Kx32_Mem_A<11> LOC=L23;
Net ZBT_512Kx32_Mem_A<11> FAST;
Net ZBT_512Kx32_Mem_DQ<31> LOC=T30;
Net ZBT_512Kx32_Mem_DQ<31> FAST;
Net ZBT_512Kx32_Mem_DQ<30> LOC=P28;
Net ZBT_512Kx32_Mem_DQ<30> FAST;
Net ZBT_512Kx32_Mem_DQ<29> LOC=R25;
Net ZBT_512Kx32_Mem_DQ<29> FAST;
Net ZBT_512Kx32_Mem_DQ<28> LOC=R29;
Net ZBT_512Kx32_Mem_DQ<28> FAST;
Net ZBT_512Kx32_Mem_DQ<27> LOC=R27;
Net ZBT_512Kx32_Mem_DQ<27> FAST;
Net ZBT_512Kx32_Mem_DQ<26> LOC=R23;
Net ZBT_512Kx32_Mem_DQ<26> FAST;
Net ZBT_512Kx32_Mem_DQ<25> LOC=N30;
Net ZBT_512Kx32_Mem_DQ<25> FAST;
Net ZBT_512Kx32_Mem_DQ<24> LOC=K26;
Net ZBT_512Kx32_Mem_DQ<24> FAST;
Net ZBT_512Kx32_Mem_DQ<23> LOC=M25;
Net ZBT_512Kx32_Mem_DQ<23> FAST;
Net ZBT_512Kx32_Mem_DQ<22> LOC=J29;
Net ZBT_512Kx32_Mem_DQ<22> FAST;
Net ZBT_512Kx32_Mem_DQ<21> LOC=K27;
Net ZBT_512Kx32_Mem_DQ<21> FAST;
Net ZBT_512Kx32_Mem_DQ<20> LOC=L24;
Net ZBT_512Kx32_Mem_DQ<20> FAST;
Net ZBT_512Kx32_Mem_DQ<19> LOC=H27;
Net ZBT_512Kx32_Mem_DQ<19> FAST;
Net ZBT_512Kx32_Mem_DQ<18> LOC=H26;
Net ZBT_512Kx32_Mem_DQ<18> FAST;
Net ZBT_512Kx32_Mem_DQ<17> LOC=K25;
Net ZBT_512Kx32_Mem_DQ<17> FAST;
Net ZBT_512Kx32_Mem_DQ<16> LOC=H28;
Net ZBT_512Kx32_Mem_DQ<16> FAST;
Net ZBT_512Kx32_Mem_DQ<15> LOC=J25;
Net ZBT_512Kx32_Mem_DQ<15> FAST;
Net ZBT_512Kx32_Mem_DQ<14> LOC=J26;
Net ZBT_512Kx32_Mem_DQ<14> FAST;
Net ZBT_512Kx32_Mem_DQ<13> LOC=J28;
Net ZBT_512Kx32_Mem_DQ<13> FAST;
Net ZBT_512Kx32_Mem_DQ<12> LOC=K24;
Net ZBT_512Kx32_Mem_DQ<12> FAST;
Net ZBT_512Kx32_Mem_DQ<11> LOC=J27;
Net ZBT_512Kx32_Mem_DQ<11> FAST;
Net ZBT_512Kx32_Mem_DQ<10> LOC=K29;
Net ZBT_512Kx32_Mem_DQ<10> FAST;
Net ZBT_512Kx32_Mem_DQ<9> LOC=L25;
Net ZBT_512Kx32_Mem_DQ<9> FAST;
Net ZBT_512Kx32_Mem_DQ<8> LOC=L26;
Net ZBT_512Kx32_Mem_DQ<8> FAST;
Net ZBT_512Kx32_Mem_DQ<7> LOC=P30;
Net ZBT_512Kx32_Mem_DQ<7> FAST;
Net ZBT_512Kx32_Mem_DQ<6> LOC=P23;
Net ZBT_512Kx32_Mem_DQ<6> FAST;
Net ZBT_512Kx32_Mem_DQ<5> LOC=P27;
Net ZBT_512Kx32_Mem_DQ<5> FAST;
Net ZBT_512Kx32_Mem_DQ<4> LOC=T29;
Net ZBT_512Kx32_Mem_DQ<4> FAST;
Net ZBT_512Kx32_Mem_DQ<3> LOC=R24;
Net ZBT_512Kx32_Mem_DQ<3> FAST;
Net ZBT_512Kx32_Mem_DQ<2> LOC=R28;
Net ZBT_512Kx32_Mem_DQ<2> FAST;
Net ZBT_512Kx32_Mem_DQ<1> LOC=U30;
Net ZBT_512Kx32_Mem_DQ<1> FAST;
Net ZBT_512Kx32_Mem_DQ<0> LOC=T28;
Net ZBT_512Kx32_Mem_DQ<0> FAST;
Net ZBT_512Kx32_Mem_BEN<0> LOC=G29;
Net ZBT_512Kx32_Mem_BEN<0> FAST;
Net ZBT_512Kx32_Mem_BEN<1> LOC=F29;
Net ZBT_512Kx32_Mem_BEN<1> FAST;
Net ZBT_512Kx32_Mem_BEN<2> LOC=H24;
Net ZBT_512Kx32_Mem_BEN<2> FAST;
Net ZBT_512Kx32_Mem_BEN<3> LOC=J24;
Net ZBT_512Kx32_Mem_BEN<3> FAST;
Net ZBT_512Kx32_Mem_WEN LOC=F26;
Net ZBT_512Kx32_Mem_WEN FAST;
Net ZBT_512Kx32_Mem_OEN<0> LOC=F30;
Net ZBT_512Kx32_Mem_OEN<0> FAST;
Net ZBT_512Kx32_Mem_CEN<0> LOC=G26;
Net ZBT_512Kx32_Mem_CEN<0> FAST;
Net ZBT_512Kx32_Mem_CKEN LOC=G30;
Net ZBT_512Kx32_Mem_CKEN FAST;
Net ZBT_512Kx32_Mem_ADV_LDN LOC=K23;
Net ZBT_512Kx32_Mem_ADV_LDN FAST;
Net ZBT_CLOCK LOC=G27;
Net ZBT_CLOCK FAST;

NET "CHAR_MODE_SVGA_CTRL_0_pixel_clock"  LOC = "AD16"  ;
NET "CHAR_MODE_SVGA_CTRL_0_reset"  LOC = "F14"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_COMP_SYNCH_N"  LOC = "A26"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_HSYNCH_N"  LOC = "F24"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLANK_N"  LOC = "A25"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLUE_P<0>"  LOC = "C30"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLUE_P<1>"  LOC = "B30"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLUE_P<2>"  LOC = "G23"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLUE_P<3>"  LOC = "H23"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLUE_P<4>"  LOC = "D28"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLUE_P<5>"  LOC = "E28"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLUE_P<6>"  LOC = "D29"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_BLUE_P<7>"  LOC = "C29"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_GREEN_P<0>"  LOC = "G21"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_GREEN_P<1>"  LOC = "G22"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_GREEN_P<2>"  LOC = "B25"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_GREEN_P<3>"  LOC = "A24"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_GREEN_P<4>"  LOC = "D25"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_GREEN_P<5>"  LOC = "C24"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_GREEN_P<6>"  LOC = "F22"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_GREEN_P<7>"  LOC = "F23"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_PIXEL_CLOCK_P"  LOC = "A27"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_RED_P<0>"  LOC = "E23"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_RED_P<1>"  LOC = "E22"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_RED_P<2>"  LOC = "H20"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_RED_P<3>"  LOC = "H21"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_RED_P<4>"  LOC = "B24"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_RED_P<5>"  LOC = "B23"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_RED_P<6>"  LOC = "D23"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_OUT_RED_P<7>"  LOC = "D24"  ;
NET "CHAR_MODE_SVGA_CTRL_0_VGA_VSYNCH_N"  LOC = "E24"  ;
```

References

[1] Multimedia Board User Guide: ../references/Multimedia_UserGuide.pdf
[2] Virtex-II Datasheet: ../datasheets/VirtexII.pdf
[3] Texas Hold'em Rule: http://www.pokertips.org/rules/texas.php
[4] Microblaze Datasheet: ../datasheets/MicroBlaze.pdf
[5] EDK Documentation: http://www.xilinx.com/ise/embedded/edk_docs.htm
[6] Module m01: ../references/m01.pdf
[7] Module m02: ../references/m02.pdf
[8] Module m03: ../references/m03.pdf
[9] Module m04: ../references/m04.pdf
[10] SVGA Reference Design: ../references/COLOR_CHAR_MODE.zip
[11] Device Drivers Documentation: ../references/xilinx_drivers.pdf
[12] Simple Reliable Communication for Xilinx FPGAs:
../references/Simple_Reliable_Communication_for_Xilinx_FPGAs.pdf
[13] MicroBlaze Processor Reference Guide: ../references/mb_ref_guide.pdf
[14] EMAC Datasheet: ../datasheets/opb_emac.pdf
[15] LXT972 Datasheet: ../datasheets/LXT972.pdf
[16] UARTLITE Datasheet: ../datasheets/opb_uartlite.pdf
[17] Timer/Counter Datasheet: ../datasheets/opb_timer.pdf
[18] Interrupt Controller Datasheet: ../datasheets/opb_intc.pdf
[19] EST Guide: ../references/est_guide.pdf
[20] GPIO Datasheet: ../datasheets/opb_gpio.pdf
[21] EMC Datacheet: ../datasheets/opb_emc.pdf
[22] ZBT Datasheet: ../datasheets/ZBT_k7n163601a.pdf
[23] Ethereal: http://www.ethereal.com