4/7/2004
Ke Xu, Eric Li, Winston Chao

Team Pac-Man
FINAL PROJECT REPORT

Contents

Overview of Pac-Man

Project Name: Pac-Man

Project Group: Ke Xu, Eric Li, Winston Chao

Project Proposal: Our project is a simple Pac-man game-engine programmed in C code implemented on a FPGA chip. The game will require both software programming in C and hardware programming in VHDL. When completed our game will be connected to a PC and take game input from the keyboard and display the graphic results on a PC monitor.

Project Structure: The rough breakdown of our game will consist of game logic, graphics engine, and I/O. The game logic will consist of collision detection, scoring, and enemy AI. The graphics engine will include sprite rendering onto the display. The I/O will include mostly interaction with the keyboard and if we have time maybe a joystick. Currently our plan is to implement the game logic in software and the graphics engine on the hardware. If additional time is available to us, we would like to also implement audio game features within our project.

Design of Pac-Man

Introduction: This game has been designed to include the components and techniques learned from class Labs dealing with the software side as well as with VHDL. Our version of Pac-man will be partially implemented with third party software; however, the graphics display of the game as well as most of the game state/controls will be entirely

made by our team. The game engine portion of our project will be coded in C while the graphical portion of it will be done in VHDL.

Game Description: Our version of Pac-Man will have several modifications to the original game. Enemy units will be configured to attack and defend human players. Our game is based upon a point system. In a one-person game, if the player gobbles up a designated number of pellets in the game, then the player wins. Our game will host a maximum of two human players competing head to head. When the game is in two-player mode, a win condition is given to the highest scoring human player, regardless of AI effect. Additionally, our game state will contain "Magic Pellets" which will offer players special powers such as attack capability, invincibility, wall smashing…etc.

Software/Game Engine: The software side of our project is coded in C. The game engine will store the information of the game map along with the states of the player and enemy units. The game engine is divided into the following major components.

Game State: The game state is represented by the **grid[][]** data-structure, it is essentially an 80x60 two-dimensional integer array. Within each cell, the integer value can be 0,1,2,3,4,5 or 6: which exactly corresponds to the 8-bit blank, wall, and pellet data blocks encoded in hexadecimal within the Character ROM in hardware. This kind of index allows quick conversion between the game state and the graphics display, since the grid element index is also the character tile to be displayed on screen. The game-state is manipulated by changing the non-zero elements into zeros. This change corresponds to mobile units eating the pellets or eating through walls. State data of Human players and AI units are stored in the struct **player**. The game engine keeps track of 2 arrays of player pointers: **allplayers[]** and **allunits[]** for the human players and the AI units respectively.

Note: Game state is customizable see **User Customization** for further information.

Game Clock: In-game movement will be controlled by a central-clock. Movement will be scaled according to the cycles of the clock. For example, movement at normal speed would be one game tile per clock cycle, at reduced speed would be two tiles per one cycle. The game clock is implemented by a for-loop that continuously runs. At each loop, a directional look up will be done for the player(s) and a tile movement will be recorded in the game state if a movement command is detected.

Game Map Generation: Our game will feature a map generator that will basically generate the walls of a particular map along with pellet combinations and layout. Our wall construct is fairly simple. We used a 14x5 integer array called **map** to simulate the walls of the map. Each row of the array corresponds to a particular wall. The information per row is encoded in the following way:

Begin x | Begin y | End x | End y | Wall type (vertical/horizontal)

The walls are generated after the game-engine reads the 14x5 array and populates the specified coordinates on **grid** with wall values. Pellet generation is done by randomly placing normal and specialized pellets. The random placement probability is biased so that roughly .005% of all placed pellets is a special pellet. So this means that while a user can dictate the structure of the walls for a particular game, the pellet arrangement is basically random and cannot be set by the user.

Game Controls: Our keyboard controls will be limited to 10 buttons. The input method is via keyboard through MINICOM to UART, and finally to our character buffer in xuartlite.c. Originally we were going to set the control as the up, down, left, and right buttons . However, during game development it was found that this set of controls is extremely awkward for two player games (players kept getting into each other's way). Our current buttons map to the following pattern:

o->player1 up
l ->player1 down
k->player1 left
;->player1 right
p->player1 pause

w->player2 up
s->player2 down
a->player2 left
d->player 2 right
q->player 2 pause

Note: All player control buttons are user-customizable see **User Customization** for further information.

Artificial Intelligence: It was decided that our AI should be significantly more advanced that the traditional Pac-Man AI. Thus we wrote our own improved version of the Pac-Man AI from scratch. The basic AI unit of our game conforms to the following algorithm.

1. Do a lookup on **allplayers[]**, and search for the closest human player by Euclidean distance
2. Once found, check human for invincibility or attack capability
    2.a If invincibility power found….calculate shortest path to human unit and approach
    2.b If attack power found…..calculate shortest path away from human unit and proceed to run away. At each tile of the path, do a look up on allplayers[] for another human player that is nearer.
        2.b.1 If another player is found, go back to step 2
        2.b.2 If not found, proceed on path to furthest distance, and maintain distance until player's attack capability wears off, then go back to step 2

3. If player isn't attacking or invincible, calculate shortest path to player and proceed to intercept.
4. If player is intercepted by the AI or another AI, proceed to step 1.
5. If all players have attack capability, proceed to staying away from every player.
6. If all players are dead, proceed to deactivate.
7. If any player wins…proceed to deactivate.

Magic Pills: The game engine supports three different kinds of magic pills, all differentiated by color. Each magic pill gives the player that eats it, a special power that lasts for 30 seconds. The effect's duration is indicated by a colored pill which appear under the player name, when the pill disappears the effect wear off.

Red Pill: Allows the player to eat AI's, within the game state, grid value in is 6.
Blue Pill: Allow the player to become invincible, nothing can eat it, grid value is 5.
Green Pill: Allows the player to eat through walls, grid value is 4.

Collision Resolution: The game engine must keep track of a multitude of different collision types between AI units and human units. The basic set of collision resolution logic is described in the following:

Human moves onto tile of another Human: No effect, both avatars inhabit same tile.
Human(s) moves onto tile of AI(s): Human dies.
AI(s) moves onto tile of AI(s): No effect, both avatars inhabit same tile
AI(s) moves onto tile of Human(s): If next human move is to another tile, human lives, else human dies.
Human(s) with attack moves onto AI(s) tile: AI(s) dies
AI(s) moves onto tile of Human(s) with attack: AI(s) dies

Game Customization: Our game is highly customizable. We included a set of parameter macros that allows users to manipulate game aspects such as #humans, #AI, Scoring/Win Condition, and keyboard controls without knowing anything about the coding of the game. The following macros are listed:

#define WIN_SCORE 500 //eat 500 pellets to win, entering 0 will access default win condition
#define PLAYER_COUNT 2 // two player game, can be either 1 or 2
#define AI_COUNT 5 //5 AI's within the game, can be from 1 to 5
#define P1UP 128 //selects ascii value of a character button to be the up button for player 1
#define P1DOWN 127 //control buttons can be any letter or number on the keyboard
#define P1LEFT 126 // But the control buttons of different players has to be different
#define P1RIGHT 125
#define P2UP 124
#define P2DOWN 123
#define P2LEFT 122
#define P2RIGHT 121

Hardware/VHDL: The basic graphical implementation of Pac-Man will borrow heavily from what we learned in Lab 5. Our game graphics will be done entirely within VHDL. A game state will be a matrix of tiles. Each kind of graphical tile will be stored in ROM. Each tile represents an 8x8 block of pixels on the display. The tile is implemented as a 8 bit Hexadecimal value. The tile is the basic building block of walls, pellets, and mobile units. The other major building block of the hardware portion of Pac-Man is the graphical sprits to represent the human and Avatar units:

Tile Engine: The background is implemented using a tile based system where each 8 x 8 square on the screen is one tile. On each tile, we can draw any of various background images including a pellet, a piece of the wall, or alphanumeric characters for displaying messages to the player. The key distinction between the background and the sprites is that the background does not need to be animated. Each tile does not need to move or display motion. All the tiles on the screen together comprise the game board.

We modified the character display hardware from lab 5 for use as our tile engine. As in the character display, we have a ROM that contains all possible tiles for the game. Every byte in the frame buffer specifies a tile to draw in that 8 x 8 space. The original character display did not support colors. To add color to our tile engine, we used the upper 3 bits of the 8 bits for specifying a tile to encode a color. This reduced the total number of different tiles we're allowed to use to only 32 but that is more than sufficient. The three bits of color toggle on red, green, and blue shades respectively, so that if we wanted to color a tile red, we would set its highest bit to 1 followed by two 0's and then the encoding for the tile that we want.

Sprite Engine: The sprites are the animated characters that move smoothly across the foreground. Unlike the tiles, the sprites are not restricted to appear only within 8 x 8 squares of the background grid. They can move between tiles and their motion is smooth. Also, sprites are rendered on top of the tiles so that any time a square contains both a sprite and a tile, the sprite will be drawn over the tile.

The sprite engine we implemented using our own set of hardware components that we hooked up together with the tile engine. We used BRAM to implement a sprite ROM much like the tile ROM. Each sprite occupies a 16 x 16 block. To place a sprite on screen, you must specify the x and y coordinates in terms of pixels of its top left corner and the encoding of the specific sprite that you want to display. The coordinate and encoding information along with the screen line counter from the vga timing component feed together into the sprite ROM to specify which line of the sprite the screen should be drawing. There are also two toggle signals that are turned on when the pixel and line counters of the screen are at the top left corner of the sprite because that is when we want to start using the data coming out of the sprite ROM to draw our sprite. The toggle signals are turned on when the x and y coordinates of the sprite match the values of the pixel and line counters from the vga timing component. At the same time that the signals are toggled, two 4-bit counters also start counting from 0 to 15. One counter is to tell the horizontal pixel toggle when to turn itself off, i.e. to stop drawing the sprite 16 pixels

after where we started drawing it. Similarly, the other counter turns the vertical line toggle signal off after the screen refreshes past the bottom line of the sprite. The pixel data comes out of the sprite ROM 16 bits at a time and goes into a shift register that shifts by 1 bit every cycle for 16 cycles. The most significant bit of the register is used to specify whether or not we light the next pixel. Actually, the register's MSB doesn't directly feed into the RGB signals of the monitor. There is one more component, the arbitrator, that receives both the pixel information of the sprite engine and the tile engine. The arbitrator then decides whether to plot the tile's pixel or the sprite's pixel. The output from the arbitrator then feeds into the RGB signals of the monitor.

There is one more important part of the sprite engine, and that is where we get the coordinate and encoding information needed to draw the sprite. Initially, we planned to use a register that we would expose to software to hold this information so that in our game engine, we could manipulate the position and orientation of the sprite in code. When we talked to Christian about doing this, he said that it's quite complicated and that we should try to find and use some code that already works. We weren't able to find code suited to our purpose, and we didn't have enough time to read through, understand, and modify other people's code that does something similar. In the end, we thought of a way around it, though it's somewhat of a hack. We chose an unused part of our framebuffer and designed that to hold the information that the sprite register would've been used for. Then we added some hardware to detect when we're reading data from that section and use that data to drive the coordinate and encoding signals for the sprites.

Ultimately, we were able to get our tile and sprite engine hardware to work correctly. But we didn't have enough time to make a clean integration between our software and hardware parts and there are some loose ends that we could improve.

Possible additions/corrections

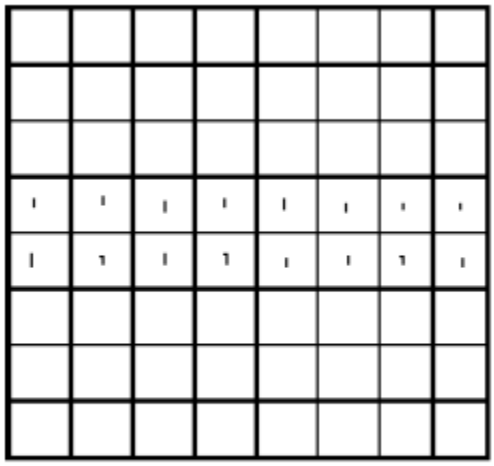
Due to time constraints, we were not able to get some of what we originally wanted done. But here are some components we wanted to add.

The FPGA only had a certain number of BRAMs available to us. Because of this, we weren't able to fit all the sprites we wanted with all possible orientations for each into our sprite ROM. Instead, we had to settle for one orientation of our sprites. One solution to this problem is to add a hardware component that does rotations on the sprites so that we can have all possible orientations without needing additional BRAM resources.

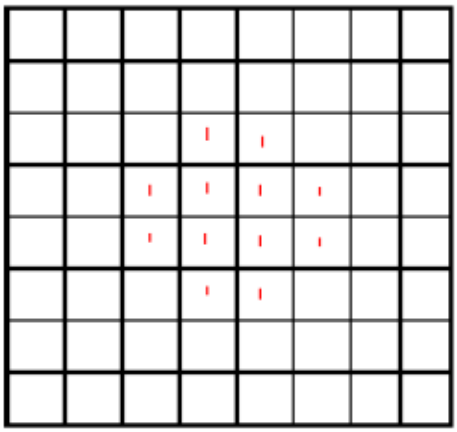
Another constraint is that we currently can only support one sprite on the screen. If we had more time, we could remedy this problem by duplicating the sprite engine hardware and feeding a few more sprite signals into the arbitrator to be rendered on screen. It would be the same exact set of components that we added for our first sprite hooked up in parallel.

Last but not least, we would've liked to have some midi sound effects, but that requires a whole other set of hardware.

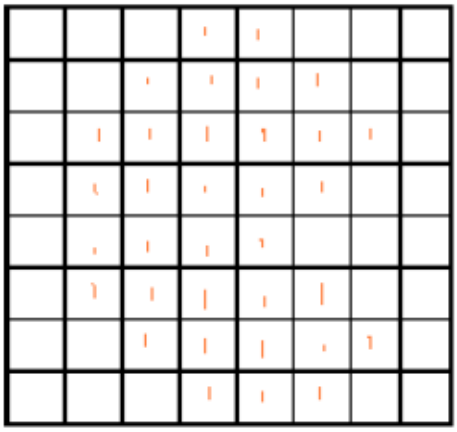
Illustrations:



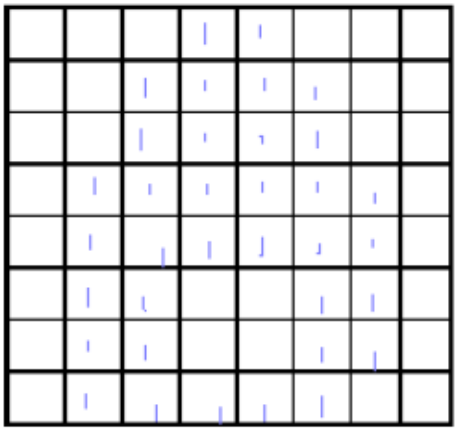
Horizontal Wall



Pellet Tile

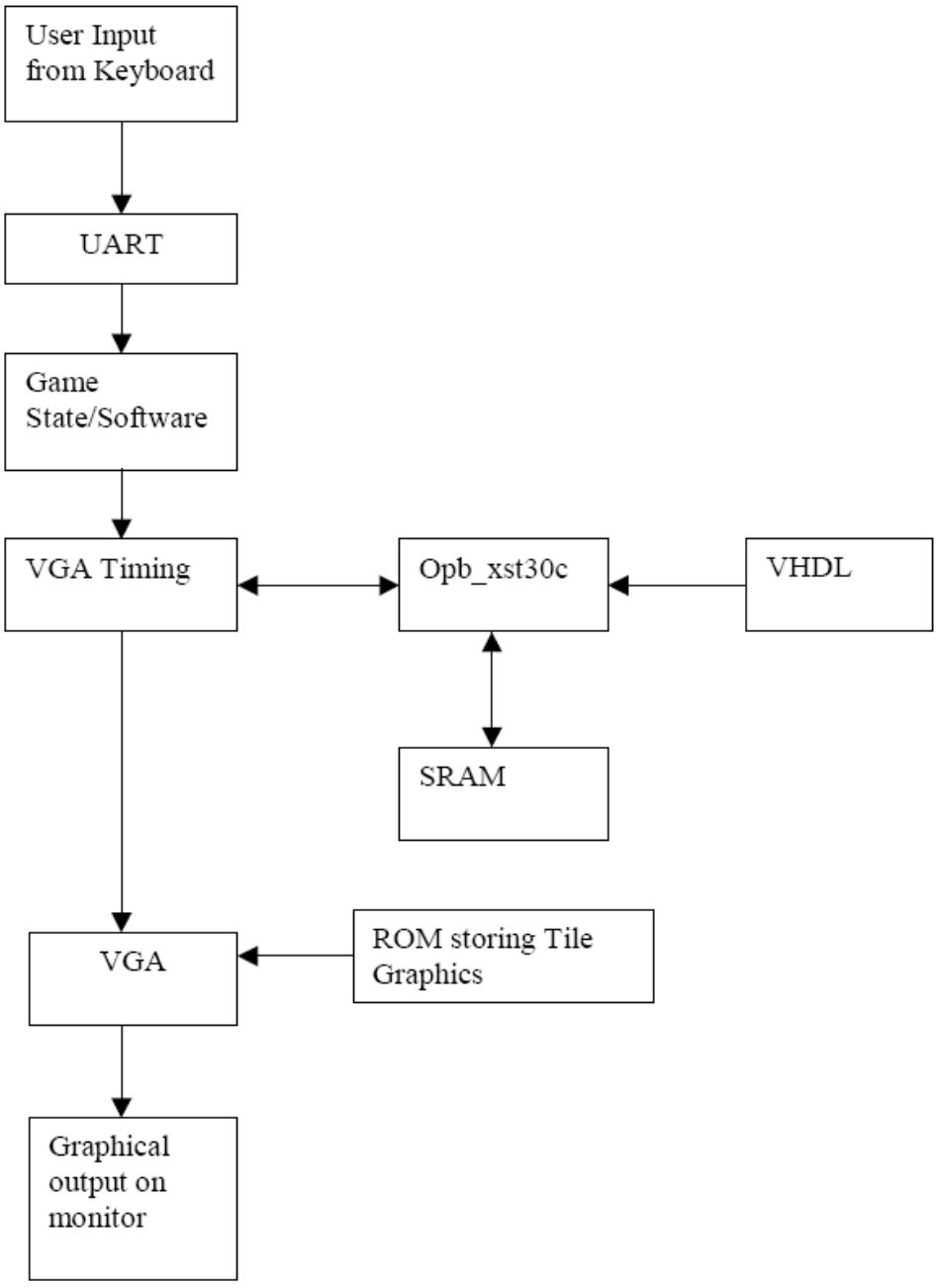


Player Tile



Enemy Unit

Block Diagram:



Workload Distribution

Written Portion:
Proposal: Ke Xu, Eric Li
Design: Ke Xu
Final Report: Ke Xu

Programming/Labs:
Lab1: Eric Li, Ke Xu
Lab2: Eric Li, Ke Xu
Lab3: Ke Xu
Lab4: Eric Li, Ke Xu
Lab5: Eric Li
Lab6: Eric Li

Programming/Project:
Hardware/Graphics: Eric Li
Software/Game-Engine: Ke Xu
Integration: Eric Li, Ke Xu
Testing: Eric Li, Ke Xu

Lessons Learned:

Written Portion/Design: We learned that it is best to start the proposal and design stages early on so as to hammer out a project design that would allow for completion before the deadline. Additionally, we've found that stating specific parameters during the design phase makes the actual development far more routine and mechanical. Certain areas of our project were ambiguously defined, thus we had to spend extra time during development to hammer out aspects that should've been specified during the design phase.

Programming Portion/Software: We learned that it is best to plan out sub-modular components of the game-engine in detail before actually beginning to code. Although these segments of code serve fairly trivial functions, such as score-keeping or refreshing the display, they did take a disproportionate amount of time to implement due to insufficient prior planning. Although these small segments of code usually do not fall under the design document, we recommend that smaller non-critical features should nevertheless be thought out during the beginning of the implementation phase, prior to actual coding. Critical interface code between hardware and software should be as centralized as possible. Since we structured the game engine to have the graphics interface code concentrated in a very small part of the program, integration was

completed within an hour instead of a day. User customization of game parameters, though not originally part of the design document, nevertheless proved extremely helpful in the testing stage. The user customization code, allows for a very quick and simple means of changing the parameters of the game without actually changing any of the game code.

Programming Portion/Hardware: We learned that it is an absolutely necessity to start early on the Hardware portion. The more time allocated for this portion of the project, the better. Additionally, it is apparent that all members of the team should attempt to understand the specifics of lab 4 and 5. We encountered a developmental bottleneck due to the fact that only one of our members understood VHDL enough to code in it without wasting huge amounts of time on it. If every member of our group had the same experience with VHDL, the development cycle would've been much smoother.

Future Advice:

1. Start as early as possible, many of our design decisions and game parameters were done in a hurried fashion and thus were not as well thought out.

2. Set milestones so as to be able to keep track of what is being done and what is not being done.

3. When coding the software side, for each module, plan out the code-details of that module before actually coding it. Knowing what the module is supposed to do is not as effective as knowing how to build it when building it.

4. Get as much experience with VHDL as possible, so as to save effort on the graphical component of the game.

5. Be sure to leave adequate time for testing and integration.

6. Try not to partner with people who simply can't or won't do any work.

File Listing:

./c_source_files/isr.c
./c_source_files/main.c
./myip/opb_xsb300_v1_00_a/hdl/vhdl/asciirom.vhd
./myip/opb_xsb300_v1_00_a/hdl/vhdl/spriterom.vhd
./myip/opb_xsb300_v1_00_a/hdl/vhdl/vga.vhd
./myip/opb_xsb300_v1_00_a/hdl/vhdl/vga_timing.vhd
./myip/opb_xsb300_v1_00_a/data/opb_xsb300_v2_0_0.pao

main.c

```
//Main Pac-Man Game Engine
//Ke Xu
//ESD TEAM Pac-Man
//5/7/2004


#include "xbasic_types.h"
#include "xio.h"
#include "xintc_l.h"
#include "xuartlite_l.h"

#define TRUE 1
#define FALSE 0
#define W 80
#define H 60
#define VGA_START 0x00800000
#define LAST_LINE 59
#define C_CURSOR 24
#define C_SPACE 32
#define C_BACKSPACE 8
#define C_CRETURN 10
#define C_NEWLINE 13
#define C_RETURN 13
#define BUF_SIZE 10
#define WIN_SCORE 400
#define AI_COUNT 5
```

```
#define PLAYER_COUNT 1
#define P1UP 112
#define P1DOWN 59
#define P1LEFT 108
#define P1RIGHT 39
#define P1PAUSE 111
#define P2UP 119
#define P2DOWN 115
#define P2LEFT 97
#define P2RIGHT 100
#define P2PAUSE 101



// defined in isr.c
extern void uart_handler(void *callback);

// circular character buffer
char buffer[BUF_SIZE];
int head = 0;
int tail = 0;
int count = 0;
int grid[80][60];
unsigned long int next=1;
struct player ai;
struct player ai2;
struct player ai3;
struct player ai4;
struct player ai5;

int rand()
{
  next=next*1103515245+12345;
  return (unsigned int)(next/65536)%32768;


}

int dist(x1,x2,y1,y2){
  return (x2-x1)*(x2-x1)+(y2-y1)*(y2-y1);
}



//basic structure of a tile holding 1 bit
struct tile{
  unsigned int tiletype:1;
  unsigned int x:7;
  unsigned int y:6;
};

struct player{
  int
x,y,prevx,prevy,score,power,direction,alive,invulnerable,chewing,attack
;
  int aiType,it,ct,at;//default no AI
```

```
};


/*
 * setup_interrupts: Initialize the interrupt sources and handlers
 *
 * Should be called once when the system starts
 *
 * The main _interrupt_handler() function from Xilinx
 *
 * Saves and restores CPU context, etc.
 *
 * Sees which interrupts are pending, and for each it
 *     acknowledges the interrupt and
 *     calls a user-defined interrupt handler in
Xintc_InterruptVectorTable
 *
 * Place interrupt service routines in isr.c to ensure they are placed
in
 * the proper memory segment.
 */
void setup_interrupts()
{
  /*
   * Reset the interrupt controller peripheral
   */

  /* Disable the interrupt signal */
  XIntc_mMasterDisable(XPAR_INTC_SINGLE_BASEADDR);

  /* Disable all interrupt sources */
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,0);

  /* Acknowledge all possible interrupt sources
     to make sure none are pending */
  XIntc_mAckIntr(XPAR_INTC_SINGLE_BASEADDR, 0xffffffff);

  /*
   * Install the UART interrupt handler
   */

  XIntc_InterruptVectorTable[XPAR_INTC_MYUART_INTERRUPT_INTR].Handler =
    uart_handler;

  /*
   * Enable interrupt sources
   */

  /* Enable CPU interrupts */
  microblaze_enable_interrupts();

  /* Enable interrupts from the interrupt controller */
  XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);

  /* Tell the interrupt controller to accept interrupts from the UART
*/
```

```
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,
XPAR_MYUART_INTERRUPT_MASK);

  /* Enable UART interrupt generation */
  XUartLite_mEnableIntr(XPAR_MYUART_BASEADDR);
}

//  AI that just tries to to reach no walled areas
void default_AI(struct player *ai){
  int zx=0,zy=0,x=0,y=0;
  ai->prevx=ai->x;
  ai->prevy=ai->y;
  if(grid[(ai->x)+1][ai->y]!=2){
    if(grid[(ai->x)+1][ai->y]==1){
      x=ai->x+1;
      y=ai->y;
    }
    else{
      zx=ai->x+1;
      zy=ai->y;
    }
  }

  if(grid[(ai->x)-1][ai->y]!=2){
    if(grid[(ai->x)-1][ai->y]==1){
      x=ai->x-1;
      y=ai->y;
    }
    else{
      zx=ai->x-1;
      zy=ai->y;
    }
  }

  if(grid[ai->x][(ai->y)+1]!=2){
    if(grid[(ai->x)][ai->y+1]==1){
      y=ai->y+1;
      x=ai->x;
    }
    else{
      zy=ai->y+1;
      zx=ai->x;
    }
  }

  if(grid[ai->x][(ai->y)-1]!=2){
    if(grid[(ai->x)][ai->y-1]==1){
      y=ai->y-1;
      x=ai->x;
    }
    else{
      zy=ai->y-1;
      zx=ai->x;
    }
  }

  if(x!=0 || y!=0){
```

```
    ai->x=x;
    ai->y=y;
    ai->score++;

  }
  else if(zx!=0 || zy!=0){
    ai->x=zx;
    ai->y=zy;
  }
}


void end_game(){
  int i;
  for(i=36; i<46; i++) XIo_Out8(VGA_START + 36, 0);

}

void lost_game(int playernum){
  if(playernum==1){
    XIo_Out8(VGA_START + 35+80, 15+96);
    XIo_Out8(VGA_START + 36+80, 11+96);
    XIo_Out8(VGA_START + 37+80, 0);
    XIo_Out8(VGA_START + 38+80, 13+64);
  }
  else{
    XIo_Out8(VGA_START + 43+80, 15+96);
    XIo_Out8(VGA_START + 44+80, 12+96);
    XIo_Out8(VGA_START + 45+80, 0);
    XIo_Out8(VGA_START + 46+80, 13+64);
  }



}


void won_game(int playernum){

   if(playernum==1){
     XIo_Out8(VGA_START + 35+80, 15+96);
     XIo_Out8(VGA_START + 36+80, 11+96);
     XIo_Out8(VGA_START + 37+80, 0);
     XIo_Out8(VGA_START + 38+80, 14+32);
   }
   else{
     XIo_Out8(VGA_START + 43+80, 15+96);
     XIo_Out8(VGA_START + 44+80, 12+96);
     XIo_Out8(VGA_START + 45+80, 0);
     XIo_Out8(VGA_START + 46+80, 14+32);
   }

}

void make_invulnerable(int playernum,int type){
  int c;
  if(type==1)c=9+112;
```

```
  else c=0;

  if(playernum==2){
    XIo_Out8(VGA_START + 50+80, c);
  }
  else{
    XIo_Out8(VGA_START + 2+80, c);
  }

}

void make_chewing(int playernum, int type){
  int c;
  if(type==1)c=9+32;
  else c=0;
  if(playernum==2){
    XIo_Out8(VGA_START + 52+80, c);
  }
  else{
    XIo_Out8(VGA_START + 4+80, c);
  }

}

void make_attack(int playernum, int type){
  int c;
  if(type==1)c=9+64;
  else c=0;
  if(playernum==2){
    XIo_Out8(VGA_START + 54+80, c);
  }
  else{
    XIo_Out8(VGA_START + 6+80, c);
  }

}



//=====================================================================
=
// Main function
//=====================================================================
=
int main()
{
    // for keeping track of cursor position
    int cursor_x = 1;
    int cursor_y = 3;
    int test_x=1,test_y=3;
    int header_delay=0;
    int unit_delay=0;
    int cursor_delay = 0;
    int player_delay=0;
    char tmp;
    int x,zx,zy;
    int y,i,j,k,newmove, emptyTile,num_players=2;
```

```
   int term_size=12;
   int tile_count=0;
   int win_value=100;
   int movemaker=0;
   int totalscore=0;
   struct player* allunits[AI_COUNT];
   struct player* allplayers[PLAYER_COUNT];
   int player_clock=0;
   int AImoves[4][3];
   int change=FALSE;
   struct player p;
   struct player p2;



// Enable the instruction cache: makes the code run 6 times faster
microblaze_enable_icache();
   setup_interrupts();

   //keep track of AIs
   allunits[0]=&ai;
   if(AI_COUNT>=2)allunits[1]=&ai2;
   if(AI_COUNT>=3)allunits[2]=&ai3;
   if(AI_COUNT>=4)allunits[3]=&ai4;
   if(AI_COUNT>=5)allunits[4]=&ai5;
   //Keep track of players
   allplayers[0]=&p;
   if(PLAYER_COUNT>=2){
     allplayers[1]=&p2;
   }


   // Erase the screen
   for(x=0; x<H*W; x++)
     XIo_Out8(VGA_START + x, 0);


   for(i=0; i<W; i++){
     for(j=0; j<H; j++){
     grid[i][j]=0;
     }
   }

   //initialize player1
   p.prevx=1;
   p.prevy=3;
   p.x=1;
   p.y=3;
   p.score=1;//zero score and no powers initially
   p.power=0;
   p.direction=0;
   p.alive=1;
   p.invulnerable=FALSE;
   p.chewing=FALSE;
   p.attack=FALSE;

   //initialize player2 on otherside of map
```

```
p2.prevx=78;
p2.prevy=58;
p2.x=78;
p2.y=58;
p2.score=1;//zero score and no powers initially
p2.power=0;
p2.direction=0;
p2.alive=1;
p2.invulnerable=FALSE;
p2.chewing=FALSE;
p2.attack=FALSE;


for(i=0; i<=80; i++){
  XIo_Out8( VGA_START +i+2*W, 2+32);
  grid[i][2]=2;
}
for(i=2; i<=59; i++) {
  XIo_Out8( VGA_START +0+i*W, 1+32);
  grid[0][i]=1;
}
for(i=0; i<=79; i++) {
  XIo_Out8( VGA_START +i+59*W, 2+32);
  grid[i][59]=2;
}
for(i=2; i<=59; i++) {
  XIo_Out8( VGA_START +79+i*W, 1+32);
  grid[79][i]=1;
}
/*
 {1,10,27,10,2},//1
 {14,10,14,20,1},//2
 {16,12,16,22,1},//3
 {19,10,19,22,1},//4
 {25,12,25,26,1},//5
*/

for(i=3; i<=27; i++){
  XIo_Out8( VGA_START +i+10*W, 2+32);
  grid[i][10]=2;
}
for(i=10; i<=20; i++) {
  XIo_Out8( VGA_START +14+i*W, 1+32);
  grid[14][i]=1;
}
/*
for(i=19; i<=22; i++) {
  XIo_Out8( VGA_START +16+i*W, 1+32);
  grid[16][i]=1;
}
*/

for(i=10; i<=22; i++) {
  XIo_Out8( VGA_START +19+i*W, 1+32);
  grid[19][i]=1;
}
for(i=13; i<=22; i++) {
```

```
  XIo_Out8( VGA_START +25+i*W, 1+32);
  grid[25][i]=1;
}

/*
 {3,13,3,20,1},//6
 {3,13,24,13,2},//7
 {5,15,24,15,2},//8
 {5,15,5,22,1},//9
 {3,15,5,15,2}, //10
*/


for(i=13; i<=20; i++){
  XIo_Out8( VGA_START +3+i*W, 1+32);
  grid[3][i]=1;
}
for(i=3; i<=12; i++) {
  XIo_Out8( VGA_START +i+13*W, 2+32);
  grid[i][13]=2;
}
/*



    {20,18,20,32,1},//11
  {24,22,27,22,2},//12
  {3,25,8,25,2},//13
  {3,25,3,50,1},//14
  {8,25,8,50,1},//15
*/

for(i=18; i<=30; i++){
  XIo_Out8( VGA_START +22+i*W, 1+32);
  grid[22][i]=1;
}
/*
for(i=24; i<=27; i++) {
  XIo_Out8( VGA_START +i+22*W, 2+32);
  grid[i][22]=2;
}
*/
for(i=3; i<=8; i++) {
  XIo_Out8( VGA_START +i+25*W, 2+32);
  grid[i][25]=2;
}
for(i=25; i<=50; i++) {
  XIo_Out8( VGA_START +3+i*W, 1+32);
  grid[3][i]=1;
}
for(i=25; i<=50; i++) {
  XIo_Out8( VGA_START +8+i*W, 1+32);
  grid[8][i]=1;
}

/*
   {3,55,12,55,2},//16
```

```
   {25,25,25,38,1},//17
   {20,34,20,52,1},//18
   {20,43,25,43,2},//19
   {25,46,25,57,1},//20
*/

for(i=3; i<=12; i++){
  XIo_Out8( VGA_START +i+55*W, 2+32);
  grid[i][55]=2;
}
for(i=25; i<=30; i++) {
  XIo_Out8( VGA_START +25+i*W, 1+32);
  grid[25][i]=1;
}
for(i=34; i<=52; i++) {
  XIo_Out8( VGA_START +20+i*W, 1+32);
  grid[20][i]=1;
}
for(i=20; i<=25; i++) {
  XIo_Out8( VGA_START +i+43*W, 2+32);
  grid[i][43]=2;
}
for(i=46; i<=56; i++) {
  XIo_Out8( VGA_START +25+i*W, 1+32);
  grid[25][i]=1;
}

/*
   {30,39,30,57,1},//21
   {30,57,40,57,2},//22
   {40,32,40,57,1},//23
   {20,32,35,32,2},//24
   {20,30,35,30,2},//25

*/


 for(i=39; i<=56; i++){
  XIo_Out8( VGA_START +30+i*W, 1+32);
  grid[30][i]=1;
}
for(i=30; i<=40; i++) {
  XIo_Out8( VGA_START +i+56*W, 2+32);
  grid[i][56]=2;
}
for(i=32; i<=56; i++) {
  XIo_Out8( VGA_START +40+i*W, 1+32);
  grid[40][i]=1;
}
for(i=20; i<=35; i++) {
  XIo_Out8( VGA_START +i+34*W, 2+32);
  grid[i][34]=2;
}
for(i=20; i<=35; i++) {
  XIo_Out8( VGA_START +i+30*W, 2+32);
  grid[i][30]=2;
}
```

```
/*
{35,32,35,55,1},//26
{20,15,20,25,2},//27
{40,22,40,27,1},//28
{40,3,40,25,1},//29
{43,10,43,25,1},//30
*/
for(i=34; i<=53; i++){
  XIo_Out8( VGA_START +35+i*W, 1+32);
  grid[35][i]=1;
}
for(i=23; i<=54; i++) {
  XIo_Out8( VGA_START +12+i*W, 1+32);
  grid[12][i]=2;
}
for(i=22; i<=27; i++) {
  XIo_Out8( VGA_START +40+i*W, 1+32);
  grid[40][i]=1;
}
for(i=5; i<=6; i++) {
  XIo_Out8( VGA_START +40+i*W, 1+32);
  grid[40][i]=1;
}
for(i=10; i<=25; i++) {
  XIo_Out8( VGA_START +43+i*W, 1+32);
  grid[43][i]=1;
}

/*
            {43,10,63,10,2},//31
            {50,13,72,13,2},//32
            {43,15,65,15,2},//33
            {65,15,65,25,1},//34
            {65,22,76,22,2},//35
*/
 for(i=43; i<=63; i++){
  XIo_Out8( VGA_START +i+10*W, 2+32);
  grid[i][10]=2;
}
 /*
for(i=50; i<=72; i++) {
  XIo_Out8( VGA_START +i+13*W, 2+32);
  grid[i][13]=2;
}
 */
for(i=43; i<=65; i++) {
  XIo_Out8( VGA_START +i+15*W, 2+32);
  grid[i][15]=2;
}
for(i=11; i<=20; i++) {
  XIo_Out8( VGA_START +64+i*W, 1+32);
  grid[65][i]=1;
}
for(i=68; i<=75; i++) {
  XIo_Out8( VGA_START +i+25*W, 2+32);
  grid[i][25]=2;
}
```

```
/*
   {67,3,67,20,1},//36
            {70,10,70,22,1},//37
            {73,3,73,20,1},//38
            {40,5,40,35,1},//39
            {43,25,43,55,1},//40
*/
for(i=5; i<=20; i++){
  XIo_Out8( VGA_START +67+i*W, 1+32);
  grid[67][i]=1;
}
for(i=10; i<=22; i++) {
  XIo_Out8( VGA_START +70+i*W, 1+32);
  grid[70][i]=1;
}
for(i=5; i<=20; i++) {
  XIo_Out8( VGA_START +73+i*W, 1+32);
  grid[73][i]=1;
}
for(i=5; i<=35; i++) {
  XIo_Out8( VGA_START +40+i*W, 1+32);
  grid[40][i]=1;
}
for(i=25; i<=54; i++) {
  XIo_Out8( VGA_START +43+i*W, 1+32);
  grid[43][i]=1;
}

/*
   {45,17,62,17,2},//41
   {45,17,45,32,1},//42
            {47,39,47,57,1},//43
            {48,15,48,39,1},//44
            {50,15,50,42,1},//45

*/
for(i=45; i<=62; i++){
  XIo_Out8( VGA_START +i+18*W, 2+32);
  grid[i][18]=2;
}
for(i=18; i<=32; i++) {
  XIo_Out8( VGA_START +45+i*W, 1+32);
  grid[45][i]=1;
}
for(i=39; i<=56; i++) {
  XIo_Out8( VGA_START +47+i*W, 1+32);
  grid[47][i]=1;
}
for(i=20; i<=40; i++) {
  XIo_Out8( VGA_START +50+i*W, 1+32);
  grid[50][i]=1;
}
for(i=19; i<=42; i++) {
  XIo_Out8( VGA_START +53+i*W, 1+32);
  grid[53][i]=1;
}
```

```
for(i=47; i<=50; i++) {
  XIo_Out8( VGA_START +i+40*W, 2+32);
  grid[i][40]=1;
}



/*
{49,46,50,46,2},//46
            {67,10,67,56,1},//47
            {65,46,77,46,2},//48
            {61,32,61,46,2},//49
            {65,23,65,44,1},//50

for(i=49; i<=50; i++){
  XIo_Out8( VGA_START +i+46*W, 2+32);
  grid[i][46]=2;
}
*/
for(i=10; i<=56; i++) {
  XIo_Out8( VGA_START +67+i*W, 1+32);
  grid[67][i]=1;
}
for(i=50; i<=67; i++) {
  XIo_Out8( VGA_START +i+56*W, 2+32);
  grid[i][56]=2;
}



for(i=62; i<=76; i++) {
  XIo_Out8( VGA_START +i+46*W, 2+32);
  grid[i][46]=2;
}
for(i=34; i<=46; i++) {
  XIo_Out8( VGA_START +61+i*W, 1+32);
  grid[61][i]=1;
}
for(i=23; i<=43; i++) {
  XIo_Out8( VGA_START +64+i*W, 1+32);
  grid[64][i]=1;
}
/*
   {65,23,75,23,2}, //51
   {75,23,75,44,1},//52



for(i=73; i<=75; i++) {
  XIo_Out8( VGA_START +i+23*W, 2+32);
  grid[i][23]=2;
}
*/
for(i=25; i<=43; i++) {
  XIo_Out8( VGA_START +75+i*W, 1+32);
  grid[75][i]=1;
```

```
    }

    for(i=3; i<=35; i++){
      XIo_Out8( VGA_START +i+6*W, 2+32);
      grid[i][6]=2;
    }


    for(i=6; i<=26; i++){
      XIo_Out8( VGA_START +35+i*W, 1+32);
      grid[35][i]=1;
    }









/*
    //*********************paint walls**************************
    for(i=0; i<26; i++){
      if(map[i][0]==map[i][2]){
      for(y=map[i][1]; y<=map[i][3]; y++){
        XIo_Out8( VGA_START + map[i][0]+y*W, map[i][4]+32);
        grid[map[i][0]][y]=1;//establish that the space is occupied by
a wall
      }
      }
      if(map[i][1]==map[i][3]){
      for(x=map[i][0]; x<=map[i][2]; x++){
        XIo_Out8( VGA_START + x+map[i][1]*W, map[i][4]+32);
        grid[x][map[i][1]]=2;
      }
      }
    }
    */


    //*********************paint pellets************************
    i=0;
    for(x=0 ; x<W ; x++){
      for(y=3; y<H; y++){
      if(grid[x][y]==0)emptyTile=TRUE;
      else emptyTile=FALSE;
      //for(j=0; j<term_size; j++){
      //  if((map[j][0]==map[j][2]) && (y>=map[j][1] && y<=map[j][3])
&& x==map[j][0] ) emptyTile=FALSE;
      //  if((map[j][1]==map[j][3]) && (x>=map[j][0] && x<=map[j][2])
&& y==map[j][1]) emptyTile=FALSE;
      //  if(!emptyTile)break;
      //}
      if(emptyTile==TRUE){
        //create invulnerable pellet
        if(i%1183==687){
          grid[x][y]=4;
```

```
      XIo_Out8(VGA_START + x+y*W, 112+9);
    }
    else if(i%1975==567){//create wall crunching pellet
      grid[x][y]=5;
      XIo_Out8(VGA_START + x+y*W, 32+9);
    }
    else if(i%1130==345){//create attack pellet
      grid[x][y]=6;
      XIo_Out8(VGA_START + x+y*W, 64+9);
    }
    else if(y==3 && x==6){
      grid[x][y]=6;
      XIo_Out8(VGA_START + x+y*W, 64+9);
    }
    else{
      grid[x][y]=3;//establish that the space is occupied by pellet
      XIo_Out8(VGA_START + x+y*W, 16+3);
    }
    i++;
  }
  }
}


totalscore=i;
//Randomly place AIs
for(i=0; i<AI_COUNT; i++){
  x=rand()%75+1;
  y=rand()%55+1;

  //check to see if space is not a wall
  while(grid[x][y]!=0 && grid[x][y]!=3){
  //if wall then pick another location
  x=rand()%75+1;
  y=rand()%55+2;
  }
  allunits[i]->prevx=x;
  allunits[i]->prevy=y;
  allunits[i]->x=x;
  allunits[i]->y=y;
  allunits[i]->score=1;
  allunits[i]->direction=0;
  allunits[i]->alive=TRUE;
}

//paint player score tally
if(PLAYER_COUNT>=1){
  XIo_Out8(VGA_START + 0, 15+48);
  XIo_Out8(VGA_START + 1, 11+48);
  for(i=0; i<20; i++){
   XIo_Out8(VGA_START + 2+i, 3+80);
  }

}

if(PLAYER_COUNT==2){
  XIo_Out8(VGA_START + 77, 15+48);
```

```
      XIo_Out8(VGA_START + 78, 12+48);
      for(i=0; i<20; i++){
       XIo_Out8(VGA_START + 57+i, 3+80);
      }

    }


    //animated opening game message
    /*
    while (header_delay<=1000000){
     if(header_delay==0)
     {

     }
     else if(header_delay==1000000){
       for(i=36; i<46; i++) XIo_Out8(VGA_START + i, ' ');
     }

     header_delay= (header_delay==5000000)? 5000000 : (header_delay+1);

    }
    */


    player_delay=0;
    for (;;)
    {
        // disable interrupts while reading count since it can be
        // changed inside the isr concurrently




        microblaze_disable_interrupts();
      if(player_clock!=0  && player_clock!=-1){
      if(player_delay<20){

        if(change)player_delay--;
        if(player_delay==0)change=FALSE;
        if(!change)player_delay++;
        if(p.alive || p2.alive){
        //animate player
          if(p.alive){
            //if(p.direction==3)  XIo_Out8(VGA_START + p.x + (p.y*W),
6+96);
            //if(p.direction==4 || p.direction==0) XIo_Out8(VGA_START +
p.x + (p.y*W), 4+96);
            //if(p.direction==1) XIo_Out8(VGA_START + p.x + (p.y*W),
7+96);
            //if(p.direction==2) XIo_Out8(VGA_START + p.x + (p.y*W),
8+96);
            /*
            if(p.direction==3){
            //XIo_Out8(VGA_START + p.x + (p.y*W), 6+96);
            XIo_Out16(VGA_START+62,p.x*8);
            XIo_Out16(VGA_START+74,p.y*8);
```

```
XIo_Out16(VGA_START+80,0);
}
if(p.direction==4 || p.direction==0){
XIo_Out16(VGA_START+62,p.x*8);
XIo_Out16(VGA_START+74,p.y*8);
XIo_Out16(VGA_START+80,0);
//XIo_Out8(VGA_START + p.x + (p.y*W), 4+96);
}
if(p.direction==1){
XIo_Out16(VGA_START+62,p.x*8);
XIo_Out16(VGA_START+74,p.y*8);
XIo_Out16(VGA_START+80,0);

//XIo_Out8(VGA_START + p.x + (p.y*W), 7+96);
}
if(p.direction==2){
XIo_Out16(VGA_START+62,p.x*8);
XIo_Out16(VGA_START+74,p.y*8);
XIo_Out16(VGA_START+80,0);
//XIo_Out8(VGA_START + p.x + (p.y*W), 8+96);
}
*/


if(p.prevx<p.x){
    //XIo_Out8(VGA_START + p.x + (p.y*W), 6+96);
XIo_Out16(VGA_START+62,(p.prevx*8)+player_clock);
XIo_Out16(VGA_START+74,p.y*8);
XIo_Out16(VGA_START+80,1);
}
if(p.prevx>p.x){
XIo_Out16(VGA_START+62,(p.prevx*8)-player_clock);
XIo_Out16(VGA_START+74,p.y*8);
XIo_Out16(VGA_START+80,2);
//XIo_Out8(VGA_START + p.x + (p.y*W), 4+96);
}
if(p.prevy>p.y){
XIo_Out16(VGA_START+62,p.x*8);
XIo_Out16(VGA_START+74,(p.prevy*8)-player_clock);
XIo_Out16(VGA_START+80,4);

//XIo_Out8(VGA_START + p.x + (p.y*W), 7+96);
}
if(p.prevy<p.y){
XIo_Out16(VGA_START+62,p.x*8);
XIo_Out16(VGA_START+74,(p.prevy*8)+player_clock);
XIo_Out16(VGA_START+80,3);
//XIo_Out8(VGA_START + p.x + (p.y*W), 8+96);
}
if(p.prevy==p.y && p.prevx==p.x){
XIo_Out16(VGA_START+62,p.prevx*8);
XIo_Out16(VGA_START+74,p.prevy*8);
XIo_Out16(VGA_START+80,1);
//XIo_Out8(VGA_START + p.x + (p.y*W), 8+96);
}
```

```
          }

          if(PLAYER_COUNT>1 && p2.alive){
            if(p2.direction==3)  XIo_Out8(VGA_START + p2.x + (p2.y*W),
6+96);
            if(p2.direction==4 || p2.direction==0) XIo_Out8(VGA_START +
p2.x + (p2.y*W), 4+96);
            if(p2.direction==1) XIo_Out8(VGA_START + p2.x + (p2.y*W),
7+96);
            if(p2.direction==2) XIo_Out8(VGA_START + p2.x + (p2.y*W),
8+96);
          }



          //animate enemy

        }






      }
      else {
        if(p.alive){
           if(p.prevx<p.x){
                //XIo_Out8(VGA_START + p.x + (p.y*W), 6+96);
            XIo_Out16(VGA_START+62,(p.prevx*8)+player_clock);
            XIo_Out16(VGA_START+74,p.y*8);
            XIo_Out16(VGA_START+80,5);
            }
            if(p.prevx>p.x){
            XIo_Out16(VGA_START+62,(p.prevx*8)-player_clock);
            XIo_Out16(VGA_START+74,p.y*8);
            XIo_Out16(VGA_START+80,5);
            //XIo_Out8(VGA_START + p.x + (p.y*W), 4+96);
            }
            if(p.prevy>p.y){
            XIo_Out16(VGA_START+62,p.x*8);
            XIo_Out16(VGA_START+74,(p.prevy*8)-player_clock);
            XIo_Out16(VGA_START+80,6);

            //XIo_Out8(VGA_START + p.x + (p.y*W), 7+96);
            }
            if(p.prevy<p.y){
            XIo_Out16(VGA_START+62,p.x*8);
            XIo_Out16(VGA_START+74,(p.prevy*8)+player_clock);
            XIo_Out16(VGA_START+80,7);
            //XIo_Out8(VGA_START + p.x + (p.y*W), 8+96);
            }
            if(p.prevy==p.y && p.prevx==p.x){
```

```
            XIo_Out16(VGA_START+62,p.prevx*8);
            XIo_Out16(VGA_START+74,p.prevy*8);
            XIo_Out16(VGA_START+80,5);
            //XIo_Out8(VGA_START + p.x + (p.y*W), 8+96);
            }

          //XIo_Out8(VGA_START + p.x + (p.y*W), 5+96);

        }
          //if(PLAYER_COUNT>1){
        //if(p2.alive)XIo_Out8(VGA_START + p2.x + (p2.y*W), 5+96);
        //}


        //for(i=0; i<AI_COUNT; i++){
        //if(allunits[i]->alive)XIo_Out8(VGA_START+allunits[i]-
>x+(allunits[i]->y*W),10+16);
        //}
        if(player_delay==40)change=TRUE;
        if(change)player_delay--;
        else player_delay++;

      }

      if(!p.alive){
        XIo_Out16(VGA_START+62,p.x*8);
        XIo_Out16(VGA_START+74,p.y*8);
        XIo_Out16(VGA_START+80,0);
      }



        for(i=0; i<AI_COUNT; i++){
        if(allunits[i]->alive){
          XIo_Out8(VGA_START+allunits[i]->prevx+(allunits[i]-
>prevy*W),0);
          XIo_Out8(VGA_START+allunits[i]->x+(allunits[i]->y*W),10+16);

        }
      }



      if(player_clock==8)player_clock=-1;//reset clock

      }//end if player_Clock
      else if(player_clock==0){  //if player_Clock is equal to 0




      i=1;
      j=p.score;
      if(WIN_SCORE!=0) j=(j*20/WIN_SCORE);
      else j=(j*20)/totalscore;
      for(i=0; i<20; i++){
         if(i<=j)XIo_Out8(VGA_START + 2+i, 3+48);
```

```
}



if(PLAYER_COUNT==2){
  i=1;
  j=p2.score;
  if(WIN_SCORE!=0) j=(j*20/WIN_SCORE);
  else j=(j*20)/totalscore;
  for(i=0; i<20; i++){
    if(i<=j)XIo_Out8(VGA_START + 76-i, 3+48);
  }

}




 //checks duration of player special powers

  // check invulnerability of players
  if(PLAYER_COUNT==2){
    if(p2.invulnerable){
      p2.it--;
      if(p2.it<=0){
      p2.invulnerable=FALSE;
      make_invulnerable(2,0);
      }
    }
  }
  if(p.invulnerable){
    p.it--;
    if(p.it<=0){
      p.invulnerable=FALSE;
      make_invulnerable(1,0);
    }
  }


  //checks wall chewing ability of players
  if(PLAYER_COUNT==2){
    if(p2.chewing){
      p2.ct--;
      if(p2.ct<=0){
      p2.chewing=FALSE;
      make_chewing(2,0);
      }
    }
  }
  if(p.chewing){
    p.ct--;
    if(p.ct<=0){
      p.chewing=FALSE;
      make_chewing(1,0);
    }
  }
```

```
  //checks attack ability
   if(PLAYER_COUNT==2){
    if(p2.attack){
      p2.at--;
      if(p2.at<=0){
      p2.attack=FALSE;
      make_attack(2,0);
      }
    }
  }
  if(p.attack){
    p.at--;
    if(p.at<=0){
      p.attack=FALSE;
      make_attack(1,0);
    }
  }



if(PLAYER_COUNT==2){
  if(!p.alive && !p2.alive){
    lost_game(1);
    lost_game(2);
  }
}
else{
  if(!p.alive)lost_game(1);
}


//*****************************
//user defined win condition
//*****************************
if(p.alive && p.score>=WIN_SCORE){
  won_game(1);
  //kill all AI's
  for(i=0; i<AI_COUNT; i++){
    allunits[i]->alive=FALSE;
  }
  if(PLAYER_COUNT==2 && p2.score<p.score)lost_game(2);
}
if(PLAYER_COUNT==2){
  //tie
  if(p2.alive && p2.score>=WIN_SCORE && p.score<WIN_SCORE){
    won_game(2);
    lost_game(1);
    for(i=0; i<AI_COUNT; i++){
      allunits[i]->alive=FALSE;
    }
  }
}
```

```
k=FALSE;
for(i=0; i<W; i++){
  for(j=0; j<H; j++){
    if(grid[i][j]==3){
      k=TRUE;
      break;
    }
  }
}

//*********************************
//default win condition all pellets have been eaten
//*********************************
if(!k){
  end_game();
  if(PLAYER_COUNT==2){
    if(p.alive && p2.alive){
      if(p.score>p2.score){
      won_game(1);
      }
      else if(p.score<p2.score){
      won_game(2);
      }
    }
  }
  else{
    if(p.alive)won_game(1);
  }
}



for(i=0; i<1000000; i++);


  microblaze_enable_interrupts();
if(count!=0){
  tmp = buffer[head];
    count--;
    head++;
    head = (head >= BUF_SIZE) ? (head - BUF_SIZE) : head;
  }


if(tmp==P1UP){
  p.direction=1;
}
else if(tmp==P1DOWN){
  p.direction=2;
}
else if(tmp==P1LEFT){
  p.direction=3;
}
else if(tmp==P1RIGHT){
```

```
  p.direction=4;
}
else if(tmp==P1PAUSE){
  p.direction=0;
}
else if(tmp==P2UP){
  p2.direction=1;
}
else if(tmp==P2DOWN){
  p2.direction=2;
}
else if(tmp==P2LEFT){
  p2.direction=3;
}
else if(tmp==P2RIGHT){
  p2.direction=4;
}
else if(tmp==P2PAUSE){
  p2.direction=0;
}

//else{
//p.direction=0;
//}


if(p.alive){


if(p.direction==1){
   test_x=p.x;
   test_y=p.y-1;
}
else if(p.direction==2){
   test_x=p.x;
   test_y=p.y+1;
}
else if(p.direction==3){
   test_x=p.x-1;
   test_y=p.y;
}
else if(p.direction==4){
  test_x=p.x+1;
  test_y=p.y;
}
else if(p.direction==0){
  test_x=p.x;
  test_y=p.y;
}

  //Player1 Game logic
x=p.x;
y=p.y;
p.prevx=x;
p.prevy=y;
```

```
      if(grid[test_x][test_y]!=1 && grid[test_x][test_y]!=2 &&
grid[test_x+1][test_y]!=1 && grid[test_x+1][test_y]!=2 &&
grid[test_x+1][test_y+1]!=1 && grid[test_x+1][test_y+1]!=2  &&
grid[test_x][test_y+1]!=1 && grid[test_x][test_y+1]!=2    ){
        XIo_Out8(VGA_START + x + (y*W), 0);
        XIo_Out8(VGA_START + x+1 + (y*W), 0);
        XIo_Out8(VGA_START + x + ((y+1)*W), 0);
        XIo_Out8(VGA_START + x+1 + (y*W), 0);



        if(grid[test_x][test_y]==3){
          p.score++;
          grid[test_x][test_y]=0;
        }
        if(grid[test_x][test_y]==4){
          p.invulnerable=TRUE;
          p.it=200;
          make_invulnerable(1,1);
          p.score++;
          grid[test_x][test_y]=0;
        }
        if(grid[test_x][test_y]==5){
          p.chewing=TRUE;
          p.ct=200;
          make_chewing(1,1);
          p.score++;
          grid[test_x][test_y]=0;
        }
        if(grid[test_x][test_y]==6){
          p.attack=TRUE;
          p.at=200;
          make_attack(1,1);
          p.score++;
          grid[test_x][test_y]=0;
        }


        if(grid[test_x+1][test_y]==3){
          p.score++;
          grid[test_x+1][test_y]=0;
        }
        if(grid[test_x+1][test_y]==4){
          p.invulnerable=TRUE;
          p.it=200;
          make_invulnerable(1,1);
          p.score++;
          grid[test_x+1][test_y]=0;
        }
        if(grid[test_x+1][test_y]==5){
          p.chewing=TRUE;
          p.ct=200;
          make_chewing(1,1);
          p.score++;
          grid[test_x+1][test_y]=0;
        }
        if(grid[test_x+1][test_y]==6){
```

```
  p.attack=TRUE;
  p.at=200;
  make_attack(1,1);
  p.score++;
  grid[test_x+1][test_y]=0;
}

if(grid[test_x+1][test_y+1]==3){
  p.score++;
  grid[test_x+1][test_y+1]=0;
}
if(grid[test_x+1][test_y+1]==4){
  p.invulnerable=TRUE;
  p.it=200;
  make_invulnerable(1,1);
  p.score++;
  grid[test_x+1][test_y+1]=0;
}
if(grid[test_x+1][test_y+1]==5){
  p.chewing=TRUE;
  p.ct=200;
  make_chewing(1,1);
  p.score++;
  grid[test_x+1][test_y+1]=0;
}
if(grid[test_x+1][test_y+1]==6){
  p.attack=TRUE;
  p.at=200;
  make_attack(1,1);
  p.score++;
  grid[test_x+1][test_y+1]=0;
}



if(grid[test_x][test_y+1]==3){
  p.score++;
  grid[test_x][test_y+1]=0;
}
if(grid[test_x][test_y+1]==4){
  p.invulnerable=TRUE;
  p.it=200;
  make_invulnerable(1,1);
  p.score++;
  grid[test_x][test_y+1]=0;
}
if(grid[test_x][test_y+1]==5){
  p.chewing=TRUE;
  p.ct=200;
  make_chewing(1,1);
  p.score++;
  grid[test_x][test_y+1]=0;
}
if(grid[test_x][test_y+1]==6){
  p.attack=TRUE;
  p.at=200;
```

```
          make_attack(1,1);
          p.score++;
          grid[test_x][test_y+1]=0;
        }




        p.x=test_x;
        p.y=test_y;
      }
      else{//hitting a wall
        if(p.chewing){
          if(test_x!=0 && test_x!=79 && test_y!=2 && test_y!=59 &&
test_x+1!=0 && test_x+1!=79 && test_y+1!=2 && test_y+1!=59){
            XIo_Out8(VGA_START + x + (y*W), 0);
            if(grid[test_x][test_y+1]==1 ){
             grid[test_x][test_y+1]=0;
            XIo_Out8(VGA_START + x + ((y+1)*W), 0);
            }
            else{
             grid[test_x][test_y+1]=0;
            XIo_Out8(VGA_START + x+1 + y*W , 0);
            }
            grid[test_x][test_y]=0;
            p.x=test_x;
            p.y=test_y;
          }
        }
      }

      //player moves onto AI
      for(i=0; i<AI_COUNT; i++){
        if(((p.x==allunits[i]->x && p.y==allunits[i]-
>y)||(p.x+1==allunits[i]->x && p.y==allunits[i]->y)||(p.x==allunits[i]-
>x && p.y+1==allunits[i]->y)|| (p.x+1==allunits[i]->x &&
p.y+1==allunits[i]->y ))&& p.attack )allunits[i]->alive=FALSE;
        else if(p.x==allunits[i]->x && p.y==allunits[i]->y && !p.attack
&&!p.invulnerable){
          p.alive=FALSE;
          lost_game(1);
        }
      }

      for(i=0; i<AI_COUNT; i++){
        if(((p.prevx==allunits[i]->x && p.prevy==allunits[i]-
>y)||(p.prevx+1==allunits[i]->x && p.prevy==allunits[i]-
>y)||(p.prevx==allunits[i]->x && p.prevy+1==allunits[i]->y)||
(p.prevx+1==allunits[i]->x && p.prevy+1==allunits[i]->y ))&& p.attack
)allunits[i]->alive=FALSE;
        else if(p.prevx==allunits[i]->x && p.prevy==allunits[i]->y &&
!p.attack &&!p.invulnerable){
          p.alive=FALSE;
          lost_game(1);
        }
      }
```

```
}
if(p2.alive){
//Player 2 game logic
if(p2.direction==1){
   test_x=p2.x;
   test_y=p2.y-1;
}
else if(p2.direction==2){
   test_x=p2.x;
   test_y=p2.y+1;
}
else if(p2.direction==3){
   test_x=p2.x-1;
   test_y=p2.y;
}
else if(p2.direction==4){
  test_x=p2.x+1;
  test_y=p2.y;
}
else if(p2.direction==0){
  test_x=p2.x;
  test_y=p2.y;
}

  //Player2 Game logic
if(PLAYER_COUNT>1){

  x=p2.x;
  y=p2.y;
  if(grid[test_x][test_y]!=1 && grid[test_x][test_y]!=2){
    XIo_Out8(VGA_START + x + (y*W), 0);
    if(grid[test_x][test_y]==3){
      p2.score++;
      grid[test_x][test_y]=0;
    }
    if(grid[test_x][test_y]==4){
      p2.invulnerable=TRUE;
      p2.it=200;
      make_invulnerable(2,1);
      p2.score++;
      grid[test_x][test_y]=0;
    }
    if(grid[test_x][test_y]==5){
      p2.chewing=TRUE;
      p2.ct=200;
      make_chewing(2,1);
      p2.score++;
      grid[test_x][test_y]=0;
    }
```

```
          if(grid[test_x][test_y]==6){
            p2.attack=TRUE;
            p2.at=200;
            make_attack(2,1);
            p2.score++;
            grid[test_x][test_y]=0;
          }
          p2.x=test_x;
          p2.y=test_y;
        }
        else{//hitting a wall
          if(p2.chewing){
            if(test_x!=0 && test_x!=79 && test_y!=2 && test_y!=59){
            XIo_Out8(VGA_START + x + (y*W), 0);
            grid[test_x][test_y]=0;
            p2.x=test_x;
            p2.y=test_y;
            }
          }
        }


         //player moves onto AI
        for(i=0; i<AI_COUNT; i++){
          if(p2.x==allunits[i]->x && p2.y==allunits[i]->y &&
p2.attack)allunits[i]->alive=FALSE;
          else if(p2.x==allunits[i]->x && p2.y==allunits[i]->y &&
!p2.attack && !p2.invulnerable){
            lost_game(2);
            p2.alive=FALSE;
          }
        }

      }
      }


      zx=0;

      //AI game logic
      for(x=0; x<AI_COUNT; x++){
        if(allunits[x]->alive){

        allunits[x]->prevx=allunits[x]->x;
        allunits[x]->prevy=allunits[x]->y;


      //path finding for AI
      i=rand()%3;
      zx=allunits[x]->x;
      zy=allunits[x]->y;

      j=0;

      if(PLAYER_COUNT==2){
        j=(p.x-allunits[x]->x)*(p.x-allunits[x]->x)+ (p.y-allunits[x]-
>y)*(p.y-allunits[x]->y);
```

```
        k=(p2.x-allunits[x]->x)*(p2.x-allunits[x]->x)+ (p2.y-
allunits[x]->y)*(p2.y-allunits[x]->y);
        if(p.alive && p2.alive){
          if(j<k){
            j=0;
          }
          else{
            j=1;
          }
        }
        else if(p.alive && !p2.alive){
          j=0;
        }
        else{
          j=1;
        }
      }
      //horizontal/vertical tracking
      if(i!=0){
        k=1;
      if(allplayers[j]->attack)k=-1;
      newmove=TRUE;

      if(allplayers[j]->x>zx){
        zx+=k;
        if(zx!=0 && zx!=79 && zy!=2 && zy!=59)newmove=FALSE;
        else zx-=k;
      }

      if(allplayers[j]->x<zx  && newmove==TRUE){
        zx-=k;
        if(zx!=0 && zx!=79 && zy!=2 && zy!=59)newmove=FALSE;
        else zx+=k;
      }

      if(allplayers[j]->y>zy && newmove==TRUE){
        zy+=k;
        if(zx!=0 && zx!=79 && zy!=2 && zy!=59)newmove=FALSE;
        else zy-=k;
      }

      if(allplayers[j]->y<zy  && newmove==TRUE){
        zy-=k;
        if(zx!=0 && zx!=79 && zy!=2 && zy!=59)newmove=FALSE;
        else zy+=k;
      }


      if(allplayers[j]->x==zx  && newmove==TRUE){
        if(k!=1){
          zx-=1;
          if(zx==0 || zx==79 || zy==2 || zy==59){
            zx+=2;

          }
        }
      }
```

```
      if(allplayers[j]->y==zy  && newmove==TRUE){
        if(k!=1){
          zy-=1;
          if(zx==0 || zx==79 || zy==2 || zy==59){
            zy+=2;

          }
        }
      }




      }

      //if AI is in defensive mode....make sure not to run off screen
      if(zx!=0 && zx!=79 && zy!=2 && zy!=59){
        allunits[x]->x=zx;
        allunits[x]->y=zy;
      }

      //collision resolution
      if(allunits[x]->x!=allunits[x]->prevx || allunits[x]-
>y!=allunits[x]->prevy){
        //AI moves on player
        if(allunits[x]->x==allplayers[j]->x && allunits[x]-
>y==allplayers[j]->y){
          //if player not attack or invulnerable....player gets killed
          if(!allplayers[j]->invulnerable && !allplayers[j]->attack){
            allplayers[j]->alive=FALSE;
            lost_game(j+1);
          }
          else if(allplayers[j]->attack){//if player is attacking...AI
gets killed
            allunits[x]->alive=FALSE;
          }
        }


        //player moves on AI
         if(allunits[x]->prevx==allplayers[j]->x && allunits[x]-
>prevy==allplayers[j]->y){

           if(!allplayers[j]->invulnerable && !allplayers[j]->attack){
            allplayers[j]->alive=FALSE;
            lost_game(j+1);
          }
          else if(allplayers[j]->attack){
            allunits[x]->alive=FALSE;
          }
        }
```

```
         if(grid[allunits[x]->x][allunits[x]->y]!=1 &&
grid[allunits[x]->x][allunits[x]->y]!=2 && grid[allunits[x]-
>x+1][allunits[x]->y]!=1 && grid[allunits[x]->x+1][allunits[x]->y]!=2
&& grid[allunits[x]->x+1][allunits[x]->y+1]!=1 && grid[allunits[x]-
>x+1][allunits[x]->y+1]!=2 && grid[allunits[x]->x][allunits[x]->y+1]!=1
&& grid[allunits[x]->x][allunits[x]->y+1]!=2){


        if(grid[allunits[x]->prevx][allunits[x]->prevy]!=1 &&
grid[allunits[x]->prevx][allunits[x]->prevy]!=2){
          XIo_Out8(VGA_START + allunits[x]->prevx + (allunits[x]-
>prevy*W),0);
          //grid[allunits[x]->prevx][allunits[x]->prevy]=0;
        }
        else{//AI's eat through walls
           if(grid[allunits[x]->prevx][allunits[x]->prevy]==1){
             grid[allunits[x]->prevx][allunits[x]->prevy+1]=0;
              XIo_Out8(VGA_START + allunits[x]->prevx + ((allunits[x]-
>prevy+1)*W),0);
           }
           if(grid[allunits[x]->prevx][allunits[x]->prevy]==2){
             grid[allunits[x]->prevx+1][allunits[x]->prevy]=0;
              XIo_Out8(VGA_START + allunits[x]->prevx+1 +
((allunits[x]->prevy)*W),0);
           }

             grid[allunits[x]->prevx][allunits[x]->prevy]=0;

          XIo_Out8(VGA_START + allunits[x]->prevx + (allunits[x]-
>prevy*W),0);
        }

         }

      }


        }



      }

      }//end else

      player_clock++;

      //unit_delay=0;
      //while(unit_delay<100000){
      //unit_delay++;
      //}



      }

    return 0;
```

```
}
```

isr.c

```
#include "xbasic_types.h"
#include "xio.h"
#include "xparameters.h"
#include "xuartlite_l.h"

extern char buffer[BUF_SIZE];
extern int head;
extern int tail;
extern int count;

/*
 * Interrupt service routine for the UART
 */
void uart_handler(void *callback)
{
  Xuint32 IsrStatus;

  Xuint8 incoming_character;

  /* Check the ISR status register so we can identify the interrupt
source */
  IsrStatus = XIo_In32(XPAR_MYUART_BASEADDR + XUL_STATUS_REG_OFFSET);

  if ((IsrStatus & (XUL_SR_RX_FIFO_FULL | XUL_SR_RX_FIFO_VALID_DATA))
!= 0) {
    /* The input FIFO contains data: read it */
    incoming_character =
      (Xuint8) XIo_In32( XPAR_MYUART_BASEADDR + XUL_RX_FIFO_OFFSET );

    if ( count < BUF_SIZE )
    {
        // interrupts are disabled to prevent simultaneous reading and
        // writing into the character buffer
        microblaze_disable_interrupts();
        buffer[tail] = incoming_character;
        tail++;
        tail = (tail >= BUF_SIZE) ? (tail - BUF_SIZE) : tail;
        count++;
        microblaze_enable_interrupts();
    }
  }

  if ((IsrStatus & XUL_SR_TX_FIFO_EMPTY) != 0) {
    /* The output FIFO is empty: we can send another character */
  }

}
```

asciirom.vhd

```
----------------------------------------------------------------------
-- ASCII ROM component
--
-- Created by Eric Li
--
-- W4840 Lab #5
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity asciirom is

    port (  clk : in std_logic;
            addr : in std_logic_vector(9 downto 0);
            data : out std_logic_vector(7 downto 0)
         );

end asciirom;

architecture behavior of asciirom is

    type rom_type is array (0 to 255) of std_logic_vector (7 downto 0);

    constant ROM : rom_type :=

    (


    -- 0 0x00 '^@ Empty tile' --
        "00000000",
        "00000000",
        "00000000",
        "00000000",
        "00000000",
        "00000000",
        "00000000",
        "00000000",

    -- 1 0x01 '^A' vertical wall--
        "11111111",
        "11111111",
        "11111111",
        "11111111",
        "11111111",
        "11111111",
        "11111111",
        "11111111",

    -- 2 0x02 '^B' Horizontal wall --
        "11111111",
        "11111111",
        "11111111",
        "11111111",
```

```
    "11111111",
    "11111111",
    "11111111",
    "11111111",

 -- 3 0x03 '^C' Pellet--
    "00000000",
    "00000000",
    "00000000",
    "00011000",
    "00011000",
    "00000000",
    "00000000",
    "00000000",

-- 4 0x04 '^D  Pacman left-orient' --
    "00011000",
    "00111100",
    "01101111",
    "01111000",
    "01111000",
    "01111111",
    "00111100",
    "00011000",

-- 5 0x05 '^E  Pacman horizontal closed-mouth --
    "00011000",
    "00111100",
    "01101110",
    "01111110",
    "01111110",
    "01111110",
    "00111100",
    "00011000",

-- 6 0x06 '^F  Packman right-orient' --
    "00011000",
    "00111100",
    "11110110",
    "00011110",
    "00011110",
    "11111110",
    "00111100",
    "00011000",

-- 7 0x07 '^G  packman up-orient' --
    "01000010",
    "01100110",
    "01111110",
    "11101111",
    "11111111",
    "11111111",
    "01111110",
    "00111100",

-- 8 0x08 '^H  Packman down orient' --
    "00111100",
```

```
    "01111110",
    "11111111",
    "11111111",
    "11101111",
    "01111110",
    "01100110",
    "01000010",

-- 9 0x09 '^I' Enemy eyes/mouth closed   --
    "00000000",
    "00000000",
    "00011000",
    "00111100",
    "00111100",
    "00011000",
    "00000000",
    "00000000",

-- 10 0x0a '^J' --
    "00111100",
    "01111110",
    "01011010",
    "01111110",
    "11111111",
    "11000011",
    "11000011",
    "11111111",

-- 11 0x31 '1' --
    "00011000",
    "00111000",
    "00011000",
    "00011000",
    "00011000",
    "00011000",
    "01111110",
    "00000000",

-- 12 0x32 '2' --
    "01111100",
    "11000110",
    "00000110",
    "00011100",
    "00110000",
    "01100110",
    "11111110",
    "00000000",

-- 13 0x0d 'X' --
    "11000011",
    "01100110",
    "00111100",
    "00011000",
    "00011000",
    "00111100",
    "01100110",
    "11000011",
```

```
-- 14 0x0e 'check' --
    "00000000",
    "00000001",
    "00000010",
    "00000100",
    "01001100",
    "01101100",
    "00111000",
    "00110000",

-- 15 0x50 'P' --
    "11111100",
    "01100110",
    "01100110",
    "01111100",
    "01100000",
    "01100000",
    "11110000",
    "00000000",

-- 16 0x10 '^P' --
    "10000000",
    "11100000",
    "11111000",
    "11111110",
    "11111000",
    "11100000",
    "10000000",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
```

```
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
```

```
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",

-- 17 0x11 '^Q' --
    "00000010",
    "00001110",
    "00111110",
    "11111110",
    "00111110",
    "00001110",
    "00000010",
    "00000000",
```

```
    -- 18 0x12 '^R' --
        "00011000",
        "00111100",
        "01111110",
        "00011000",
        "00011000",
        "01111110",
        "00111100",
        "00011000"

    );

begin

    process (clk)
    begin
        if (clk'event and clk = '1') then
            data <= ROM(conv_integer(addr(6 downto 0)));
        end if;
    end process;

end behavior;
```

spriterom.vhd

```
--------------------------------------------------------------------------------
-- Sprite ROM component
--
-- Created by Eric Li
--
-- W4840 final project
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity spriterom is

    port (  clk : in std_logic;
            addr : in std_logic_vector(6 downto 0);
            data : out std_logic_vector(15 downto 0)
         );

end spriterom;

architecture behavior of spriterom is

    type rom_type is array (0 to 127) of std_logic_vector (15 downto
0);

    constant ROM : rom_type :=

    (

    -- 0
```

```
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",

-- 1
    "0000011111100000",
    "0001111111111000",
    "0011111111111100",
    "0111100111111110",
    "0111100111111110",
    "1111111111111100",
    "1111111111100000",
    "1111111100000000",
    "1111110000000000",
    "1111111100000000",
    "1111111111100000",
    "0111111111111100",
    "0111111111111110",
    "0011111111111100",
    "0001111111111000",
    "0000011111100000",

-- 2
    "0000011111100000",
    "0001111111111000",
    "0011111111111100",
    "0111111110011110",
    "0111111110011110",
    "0011111111111111",
    "0000011111111111",
    "0000000011111111",
    "0000000000111111",
    "0000000011111111",
    "0000011111111111",
    "0011111111111111",
    "0111111111111110",
    "0011111111111100",
    "0001111111111000",
    "0000011111100000",

-- 3
    "0000011111100000",
    "0001111111111000",
    "0011111111111100",
```

```
    "0111111110011110",
    "0111111110011110",
    "1111111111111111",
    "1111111011111111",
    "1111111011111111",
    "1111110001111111",
    "1111110001111111",
    "1111110001111111",
    "0111100000111111",
    "0111100000111110",
    "0011100000111100",
    "0001100000111000",
    "0000000000000000",

-- 4
    "0000000000000000",
    "0001110000011000",
    "0011110000011100",
    "0111110000011110",
    "0111110000011110",
    "1111111000111111",
    "1111111000111111",
    "1111111000111111",
    "1111111101111111",
    "1111111101111111",
    "1111001111111111",
    "0111001111111111",
    "0111111111111110",
    "0011111111111100",
    "0001111111111000",
    "0000011111100000",

-- 5
    "0000011111100000",
    "0001111111111000",
    "0011111111111100",
    "0111100111111110",
    "0111100111111110",
    "1111111111111111",
    "1111111111111111",
    "1111111111111111",
    "1111111111111111",
    "1111111111111111",
    "1111111111111111",
    "0111111111111111",
    "0111111111111110",
    "0011111111111100",
    "0001111111111000",
    "0000011111100000",

-- 6
    "0000011111100000",
    "0001111111111000",
    "0011111111111100",
    "0111111111111110",
    "0111111111111110",
    "1111111111111111",
```

```
        "1111111111111111",
        "1111111111111111",
        "1111111111111111",
        "1111111111111111",
        "1111001111111111",
        "0111001111111111",
        "0111111111111110",
        "0011111111111100",
        "0001111111111000",
        "0000011111100000",

    -- 7
        "0000011111100000",
        "0001111111111000",
        "0011111111111100",
        "0111111110011110",
        "0111111110011110",
        "1111111111111111",
        "1111111111111111",
        "1111111111111111",
        "1111111111111111",
        "1111111111111111",
        "1111111111111111",
        "0111111111111111",
        "0111111111111110",
        "0011111111111100",
        "0001111111111000",
        "0000011111100000"
    );

begin

    process (clk)
    begin
        if (clk'event and clk = '1') then
            data <= ROM(conv_integer(addr(6 downto 0)));
        end if;
    end process;

end behavior;
```

vga.vhd

```
--------------------------------------------------------------------------------
--
-- VGA video generator
--
-- Uses the vga_timing module to generate hsync etc.
-- Massages the RAM address and requests cycles from the memory
controller
-- to generate video using one byte per pixel
--
```

```
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
-- Modified for W4840 Lab #5 by: Eric Li, Ke Xu, Winston Chao
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity vga is
  port (
    clk            : in std_logic;
    pix_clk        : in std_logic;
    rst            : in std_logic;
    video_data     : in std_logic_vector(15 downto 0);
    video_addr     : out std_logic_vector(19 downto 0);
    video_req      : out std_logic;
    VIDOUT_CLK     : out std_logic;
    VIDOUT_RCR     : out std_logic_vector(9 downto 0);
    VIDOUT_GY      : out std_logic_vector(9 downto 0);
    VIDOUT_BCB     : out std_logic_vector(9 downto 0);
    VIDOUT_BLANK_N : out std_logic;
    VIDOUT_HSYNC_N : out std_logic;
    VIDOUT_VSYNC_N : out std_logic);
end vga;

architecture Behavioral of vga is

  constant H_ACTIVE      : integer := 640;
  constant V_ACTIVE      : integer := 480;
  constant H_TOTAL       : integer := 800;
  constant V_TOTAL       : integer := 524;

  -- Fast low-voltage TTL-level I/O pad with 12 mA drive

  component OBUF_F_12
    port (
      O : out STD_ULOGIC;
      I : in STD_ULOGIC);
  end component;

  -- Basic edge-sensitive flip-flop

  component FD
    port (
      C : in std_logic;
      D : in std_logic;
      Q : out std_logic);
  end component;

  -- Force instances of FD into pads for speed

  attribute iob : string;
  attribute iob of FD : component is "true";
```

```
component vga_timing
  port (
    h_sync_delay         : out std_logic;
    v_sync_delay         : out std_logic;
    blank                : out std_logic;
    vga_ram_read_address : out std_logic_vector (19 downto 0);
    pixel_clock          : in std_logic;
    reset                : in std_logic;
    char_line            : out std_logic_vector(2 downto 0);
    pix_ld               : out std_logic;
    vreq                 : out std_logic;
    char_ld              : out std_logic;
    char_sh              : out std_logic;
    px_count             : out std_logic_vector(10 downto 0);
    ln_count             : out std_logic_vector(9 downto 0)
  );
end component;

-- Add ascii rom component
component asciirom
  port (
    clk     : in std_logic;
    addr    : in std_logic_vector(9 downto 0);
    data    : out std_logic_vector(7 downto 0)
  );
end component;

-- Add sprite rom component
component spriterom
  port (
    clk     : in std_logic;
    addr    : in std_logic_vector(6 downto 0);
    data    : out std_logic_vector(15 downto 0)
  );
end component;

signal r                      : std_logic_vector (9 downto 0);
signal g                      : std_logic_vector (9 downto 0);
signal b                      : std_logic_vector (9 downto 0);
signal blank                  : std_logic;
signal hsync                  : std_logic;
signal vsync                  : std_logic;
signal vga_ram_read_address   : std_logic_vector(19 downto 0);
signal vreq                   : std_logic;
signal vga_shreg              : std_logic_vector(15 downto 0);

signal ascii_addr             : std_logic_vector(9 downto 0);
signal char_data              : std_logic_vector(7 downto 0);
signal color_value            : std_logic_vector(9 downto 0);
signal char_line              : std_logic_vector(2 downto 0);
signal char_shreg             : std_logic_vector(7 downto 0);
signal pix_ld                 : std_logic;
signal char_ld                : std_logic;
signal char_sh                : std_logic;
signal color_val              : std_logic_vector(2 downto 0);
signal color_val_next         : std_logic_vector(2 downto 0);
signal color_val_next_next    : std_logic_vector(2 downto 0);
```

```
  signal sprite_addr            : std_logic_vector(6 downto 0);
  signal sprite_data            : std_logic_vector(15 downto 0);

  signal sprite_px_toggle       : std_logic;
  signal sprite_px_counter      : std_logic_vector(3 downto 0);
  signal sprite_ln_toggle       : std_logic;
  signal sprite_ln_counter      : std_logic_vector(3 downto 0);
  signal px_count               : std_logic_vector(10 downto 0);
  signal ln_count               : std_logic_vector(9 downto 0);
  signal sprite_reg             : std_logic_vector(15 downto 0);

  signal sprite_x               : std_logic_vector(10 downto 0);
  signal sprite_y               : std_logic_vector(9 downto 0);
  signal sprite_encoding        : std_logic_vector(2 downto 0);
  signal sprite_color           : std_logic_vector(8 downto 0);

  signal r_tile_val             : std_logic_vector(9 downto 0);
  signal g_tile_val             : std_logic_vector(9 downto 0);
  signal b_tile_val             : std_logic_vector(9 downto 0);
  signal r_spr_val              : std_logic_vector(9 downto 0);
  signal g_spr_val              : std_logic_vector(9 downto 0);
  signal b_spr_val              : std_logic_vector(9 downto 0);


begin

  -- vga timing instance
  st : vga_timing port map (
    pixel_clock => pix_clk,
    reset => rst,
    h_sync_delay => hsync,
    v_sync_delay => vsync,
    blank => blank,
    vga_ram_read_address => vga_ram_read_address,
    char_line => char_line,
    pix_ld => pix_ld,
    vreq => vreq,
    char_ld => char_ld,
    char_sh => char_sh,
    px_count => px_count,
    ln_count => ln_count
  );

  -- FIXME: This should be disabled during blanking to reduce memory
traffic

  -- Generate video_req (to the RAM controller) by delaying vreq by
  -- a cycle synchronized with the pixel clock

  process (clk)
  begin
    if clk'event and clk='1' then
      video_req <= pix_clk and vreq;
    end if;
  end process;
```

```
  -- The video address is the upper 19 bits from the VGA timing
generator
  -- because we are using two pixels per word and the RAM address
counts words

  video_addr <= '0' & vga_ram_read_address(19 downto 1);

  -- The video shift register: either load it from RAM or shift it up a
byte
  -- The load and shift signals come from vga_timing
  process (pix_clk)
  begin
    if pix_clk'event and pix_clk='1' then
      if char_ld = '1' then
        vga_shreg <= video_data;
      elsif char_sh = '1' then
        -- Shift the low byte of read video data into the high byte
        vga_shreg <= vga_shreg(7 downto 0) & "00000000";
      end if;
    end if;
  end process;

  process (pix_clk)
  begin
    if ( pix_clk'event and pix_clk = '1' ) then
      if ( px_count = 500 and ln_count = 0 ) then
        sprite_x <= video_data(10 downto 0);
      end if;
      if ( px_count = 600 and ln_count = 0 ) then
        sprite_y <= video_data(9 downto 0);
      end if;
      if ( px_count = 700 and ln_count = 0 ) then
        sprite_encoding <= video_data(2 downto 0);
      end if;
    end if;
  end process;



  -- Generate the address line into the ROM
  ascii_addr <= vga_shreg(14 downto 8) & char_line;

  -- Use the top 3 bits of the vga_shreg as color bits
  color_val_next_next <= vga_shreg(14 downto 12);

  -- Instance of the ascii ROM
  char_loader : asciirom port map (
    clk => pix_clk,
    addr => ascii_addr,
    data => char_data
  );


  -- Instance of the sprite ROM
  sprite_loader : spriterom port map (
    clk => pix_clk,
    addr => sprite_addr,
```

```
    data => sprite_data
  );

--  sprite_x <= "00000000000";

--  process ( pix_clk )
--  begin
--    if ( pix_clk'event and pix_clk = '1' ) then
--      if ( px_count = "00000000000" and ln_count = "0000000000" )
then
--        if ( sprite_x = H_ACTIVE ) then
--          sprite_x <= "00000000000";
--          sprite_y <= sprite_y + 16;
--        else
--          sprite_x <= sprite_x + 2;
--        end if;
--        if ( sprite_y = V_ACTIVE ) then
--          sprite_y <= "0000000000";
--        end if;
--      end if;
--    end if;
--  end process;

  sprite_color <= "111101000";
  --sprite_encoding <= px_count(3 downto 2) + px_count(2);

  -- Generate sprite_addr
  sprite_addr <= sprite_encoding & sprite_ln_counter;


  -- Sprite pixel toggle
  process (pix_clk)
  begin
    if ( rst = '1' ) then
      sprite_px_toggle <= '0';
    else
      if ( pix_clk'event and pix_clk = '1' ) then
        if ( (sprite_x = px_count + 1) or
             (sprite_x = "00000000000" and px_count = H_TOTAL - 1) )
then
          sprite_px_toggle <= '1';
        elsif ( sprite_px_counter = "0000" ) then
          sprite_px_toggle <= '0';
        end if;
      end if;
    end if;
  end process;

  -- Sprite pixel counter
  process (pix_clk)
  begin
    if ( rst = '1' ) then
      sprite_px_counter <= "0000";
    else
      if ( pix_clk'event and pix_clk = '1' ) then
        if ( sprite_px_toggle = '1' ) then
          sprite_px_counter <= sprite_px_counter + 1;
```

```
        end if;
      end if;
    end if;
  end process;

  -- Sprite line toggle
  process (pix_clk)
  begin
    if ( rst = '1' ) then
      sprite_ln_toggle <= '0';
    else
      if ( pix_clk'event and pix_clk = '1' ) then
        if ( (sprite_y = ln_count + 1 and px_count = H_TOTAL - 1) or
             (sprite_y = "0000000000" and ln_count = V_TOTAL - 1 and
px_count = H_TOTAL - 1 ) ) then
          sprite_ln_toggle <= '1';
        elsif ( sprite_ln_counter = "0000" and px_count = H_TOTAL - 1 )
then
          sprite_ln_toggle <= '0';
        end if;
      end if;
    end if;
  end process;

  -- Sprite line counter
  process (pix_clk)
  begin
    if ( rst = '1' ) then
      sprite_ln_counter <= "0000";
    else
      if ( pix_clk'event and pix_clk = '1' ) then
        if ( sprite_ln_toggle = '1' and ((sprite_y = 0 and px_count =
H_TOTAL - 1) or (not(sprite_y = 0) and px_count = H_TOTAL - 2)) ) then
          sprite_ln_counter <= sprite_ln_counter + 1;
        end if;
      end if;
    end if;
  end process;

  -- delay the color value by 2 ticks
  process (pix_clk)
  begin
    if ( pix_clk'event and pix_clk = '1' ) then
      color_val <= color_val_next;
      color_val_next <= color_val_next_next;
    end if;
  end process;

  -- 16 bit shift register to hold each line of pixels from a sprite
  process (pix_clk)
  begin
    if ( pix_clk'event and pix_clk = '1' ) then
      if ( (px_count = sprite_x - 1) or
           (sprite_x = 0 and px_count = H_TOTAL - 1 and
sprite_ln_toggle = '1') ) then
        sprite_reg <= sprite_data;
      elsif ( sprite_px_toggle = '1' ) then
```

```
        sprite_reg <= sprite_reg(14 downto 0) & '0';
      end if;
    end if;
  end process;

  -- 8 bit shift register to hold each line of pixels from a char
  process (pix_clk)
  begin
    if ( pix_clk'event and pix_clk = '1' ) then
      if ( pix_ld = '1' ) then
        char_shreg <= char_data;
      else
        char_shreg <= char_shreg(6 downto 0) & '0';
      end if;
    end if;
  end process;

  -- use the MSB of the 8 bit sr to plot the pixel
--  color_value <= "1110000000" when char_shreg(7) = '1' else
--                 "0000000000";
  r_tile_val <= color_val(2) & color_val(2) & color_val(2) & "1111111"
when char_shreg(7) = '1' else "0000000000";
  g_tile_val <= color_val(1) & color_val(1) & color_val(1) & "1111111"
when char_shreg(7) = '1' else "0000000000";
  b_tile_val <= color_val(0) & color_val(0) & color_val(0) & "1111111"
when char_shreg(7) = '1' else "0000000000";

  r_spr_val <= r_tile_val when sprite_reg(15) = '0' else sprite_color(8
downto 6) & "1111111";
  g_spr_val <= g_tile_val when sprite_reg(15) = '0' else sprite_color(5
downto 3) & "1111111";
  b_spr_val <= b_tile_val when sprite_reg(15) = '0' else sprite_color(2
downto 0) & "1111111";

  --r_spr_val <= sprite_color(8 downto 6) & "1111111";
  --g_spr_val <= sprite_color(5 downto 3) & "1111111";
  --b_spr_val <= sprite_color(2 downto 0) & "1111111";

  r <= r_spr_val when ( sprite_px_toggle = '1' and sprite_ln_toggle =
'1' ) else r_tile_val;
  g <= g_spr_val when ( sprite_px_toggle = '1' and sprite_ln_toggle =
'1' ) else g_tile_val;
  b <= b_spr_val when ( sprite_px_toggle = '1' and sprite_ln_toggle =
'1' ) else b_tile_val;

  -- Video clock I/O pad to the DAC

  vidclk : OBUF_F_12 port map (
    O => VIDOUT_clk,
    I => pix_clk);

  -- Control signals: hsync, vsync, and blank

  hsync_ff : FD port map (
    C => pix_clk,
    D => not hsync,
    Q => VIDOUT_HSYNC_N );
```

```
  vsync_ff : FD port map (
    C => pix_clk,
    D => not vsync,
    Q => VIDOUT_VSYNC_N );

  blank_ff :  FD port map (
    C => pix_clk,
    D => not blank,
    Q => VIDOUT_BLANK_N );

  -- Three digital color signals

  rgb_ff : for i in 0 to 9 generate

    r_ff : FD port map (
      C => pix_clk,
      D => r(i),
      Q => VIDOUT_RCR(i) );

    g_ff : FD port map (
      C => pix_clk,
      D => g(i),
      Q => VIDOUT_GY(i) );

    b_ff : FD port map (
      C => pix_clk,
      D => b(i),
      Q => VIDOUT_BCB(i) );

  end generate;

end Behavioral;
```

vga_timing.vhd

```
--------------------------------------------------------------------------------
--
-- VGA timing and address generator
--
-- Fixed-resolution address generator.  Generates h-sync, v-sync, and
blanking
-- signals along with a 20-bit RAM address.  H-sync and v-sync signals
are
-- delayed two cycles to compensate for the DAC pipeline.
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
-- Modified for W4840 Lab #5 by: Eric Li, Ke Xu, Winston Chao
--------------------------------------------------------------------------------

library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vga_timing is
  port (
    pixel_clock  : in std_logic;
    reset        : in std_logic;
    h_sync_delay : out std_logic;
    v_sync_delay : out std_logic;
    blank        : out std_logic;
    vga_ram_read_address : out std_logic_vector(19 downto 0);
    char_line    : out std_logic_vector(2 downto 0);
    pix_ld       : out std_logic;
    vreq         : out std_logic;
    char_ld      : out std_logic;
    char_sh      : out std_logic;
    px_count     : out std_logic_vector(10 downto 0);
    ln_count     : out std_logic_vector(9 downto 0)
  );
end vga_timing;

architecture Behavioral of vga_timing is

  constant SRAM_DELAY : integer := 3;

-- 640 X 480 @ 60Hz with a 25.175 MHz pixel clock
  constant H_ACTIVE      : integer := 640;
  constant H_FRONT_PORCH : integer := 16;
  constant H_BACK_PORCH  : integer := 48;
  constant H_TOTAL       : integer := 800;

  constant V_ACTIVE      : integer := 480;
  constant V_FRONT_PORCH : integer := 11;
  constant V_BACK_PORCH  : integer := 31;
  constant V_TOTAL       : integer := 524;

  signal line_count  : std_logic_vector (9 downto 0);   -- Y coordinate
  signal pixel_count : std_logic_vector (10 downto 0);  -- X coordinate

  signal h_sync : std_logic;  -- horizontal sync
  signal v_sync : std_logic;  -- vertical sync

  signal h_sync_delay0 : std_logic; -- h_sync delayed 1 clock
  signal v_sync_delay0 : std_logic; -- v_sync delayed 1 clock

  signal h_blank : std_logic;       -- horizontal blanking
  signal v_blank : std_logic;       -- vertical blanking

  -- flag to reset the ram address during vertical blanking
  signal reset_vga_ram_read_address : std_logic;

  -- flag to hold the address during horizontal blanking
  signal hold_vga_ram_read_address : std_logic;

  signal ram_address_counter : std_logic_vector (19 downto 0);
```

```
begin

  -- Pass pixel and line counters out to vga
  px_count <= pixel_count;
  ln_count <= line_count;

  -- Pixel counter

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      pixel_count <= "00000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_TOTAL - 1) then
        pixel_count <= "00000000000";
      else
        pixel_count <= pixel_count + 1;
      end if;
    end if;
  end process;

  -- Horizontal sync

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      h_sync <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_ACTIVE + H_FRONT_PORCH - 1) then
        h_sync <= '1';
      elsif pixel_count = (H_TOTAL - H_BACK_PORCH - 1) then
        h_sync <= '0';
      end if;
    end if;
  end process;

  -- Line counter

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      line_count <= "0000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if ((line_count = V_TOTAL - 1) and (pixel_count = H_TOTAL - 1))
then
        line_count <= "0000000000";
      elsif pixel_count = (H_TOTAL - 1) then
        line_count <= line_count + 1;
      end if;
    end if;
  end process;

  -- Vertical sync

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
```

```
      v_sync <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if line_count = (V_ACTIVE + V_FRONT_PORCH -1) and
         pixel_count = (H_TOTAL - 1) then
        v_sync <= '1';
      elsif line_count = (V_TOTAL - V_BACK_PORCH - 1) and
            pixel_count = (H_TOTAL - 1) then
        v_sync <= '0';
      end if;
    end if;
  end process;

  -- Add two-cycle delays to h/v_sync to compensate for the DAC
pipeline

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      h_sync_delay0 <= '0';
      v_sync_delay0 <= '0';
      h_sync_delay  <= '0';
      v_sync_delay  <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      h_sync_delay0 <= h_sync;
      v_sync_delay0 <= v_sync;
      h_sync_delay  <= h_sync_delay0;
      v_sync_delay  <= v_sync_delay0;
    end if;
  end process;

  -- Horizontal blanking

  -- The constants are offset by two to compensate for the delay
  -- in the composite blanking signal

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      h_blank <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_ACTIVE - 2) then
        h_blank <= '1';
      elsif pixel_count = (H_TOTAL - 2) then
        h_blank <= '0';
      end if;
    end if;
  end process;

  -- Vertical Blanking

  -- The constants are offset by two to compensate for the delay
  -- in the composite blanking signal

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      v_blank <= '0';
```

```
    elsif pixel_clock'event and pixel_clock = '1' then
      if line_count = (V_ACTIVE - 1) and pixel_count = (H_TOTAL - 2)
then
        v_blank <= '1';
      elsif line_count = (V_TOTAL - 1) and pixel_count = (H_TOTAL - 2)
then
        v_blank <= '0';
      end if;
    end if;
  end process;

  -- Composite blanking

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      blank <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if (h_blank or v_blank) = '1' then
        blank <= '1';
      else
        blank <= '0';
      end if;
    end if;
  end process;

  -- RAM address counter

  -- Two control signals:

  -- reset_ram_read_address is active from the end of each field until
the
  -- beginning of the next

  -- hold_vga_ram_read_address is active from the end of each line to
the
  -- start of the next

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      reset_vga_ram_read_address <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if line_count = V_ACTIVE - 1 and
         pixel_count = ( (H_TOTAL - 1 ) - SRAM_DELAY ) then
        -- reset the address counter at the end of active video
        reset_vga_ram_read_address <= '1';
      elsif line_count = V_TOTAL - 1 and
            pixel_count =((H_TOTAL -1) - SRAM_DELAY) then
        -- re-enable the address counter at the start of active video
        reset_vga_ram_read_address <= '0';
      end if;
    end if;
  end process;

  process ( pixel_clock, reset )
  begin
```

```
    if reset = '1' then
      hold_vga_ram_read_address <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = ((H_ACTIVE - 1) - SRAM_DELAY) then
        -- hold the address counter at the end of active video
        hold_vga_ram_read_address <= '1';
      elsif pixel_count = ((H_TOTAL - 1) - SRAM_DELAY) then
        -- re-enable the address counter at the start of active video
        hold_vga_ram_read_address <= '0';
      end if;
    end if;
  end process;

  -- generate the counter that the ram address to be loaded is based on
  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      ram_address_counter <= "00000000000000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if reset_vga_ram_read_address = '1' then
        ram_address_counter <= "00000000000000000000";
      else
        if (pixel_count = (H_TOTAL-3) - SRAM_DELAY) then
          if ( not(line_count(2 downto 0) = "111") ) then
            ram_address_counter <= ram_address_counter - 80;
          end if;
        else
          if ( hold_vga_ram_read_address = '0' ) then
            if ( pixel_count(2 downto 0) = "000" ) then
              ram_address_counter <= ram_address_counter + 1;
            end if;
          end if;
        end if;
      end if;
    end if;
  end process;

  vga_ram_read_address <= ram_address_counter;

  -- generate the vreq signal based on the pixel_count
  vreq <= '1' when pixel_count(3 downto 0) = "1011" else '0';

  -- generate the load and shift signals for the load shift
  -- register, char_ld is basically vreq delayed by 2 cycles
  -- to account for the delay through the memory
  char_ld <= '1' when pixel_count(3 downto 0) = "1101" else '0';

  -- char_sh is just 8 cycles after char_ld
  char_sh <= '1' when pixel_count(3 downto 0) = "0101" else '0';

  -- char_line tells what line of the character to output
  char_line <= "000" when (line_count = V_TOTAL - 1) else
               line_count(2 downto 0) + 1 when (pixel_count = H_TOTAL -
2) else
               line_count(2 downto 0);

  -- pix_ld tells the 8 bit sr when to load a new line of 8 pixels
```

```
  pix_ld <= '1' when pixel_count(2 downto 0) = "111" else '0';


end Behavioral;
```

opb_xsb300_v2_0_0.pao

```
lib opb_xsb300_v1_00_a opb_xsb300
lib opb_xsb300_v1_00_a memoryctrl
lib opb_xsb300_v1_00_a vga
lib opb_xsb300_v1_00_a vga_timing
lib opb_xsb300_v1_00_a pad_io
lib opb_xsb300_v1_00_a asciirom
lib opb_xsb300_v1_00_a spriterom
```