Embedded System Design

Video Game -- Crazy Car (Rally X like)

C.C. Liu **ccliu.iic93g@nctu.edu.tw**
Ying-Der Lee **ydlee@twins.ee.nctu.edu.tw**

08/26/2005

1. Introduction

The project has been design to take everything that we learned by the class and labs for embedded system. We use both software and hardware implementation by C and VHDL on Xilinx FPGA system. Our target is to demonstrate our knowledge and techniques of embedded system on FPGA learned in the class.

2. The Design

2.1 Game Description

We created a game called by “Crazy Car”. It is a labyrinth game like a car to find out the correct path in a maze of roads. Your goal is to find the flag hiding in the map. The car moving is controlled by PS/2 keyboard. The game map that we design is large than VGA (648x480) screen. Therefore, we use screen scrolling to display the car moving and game processing.

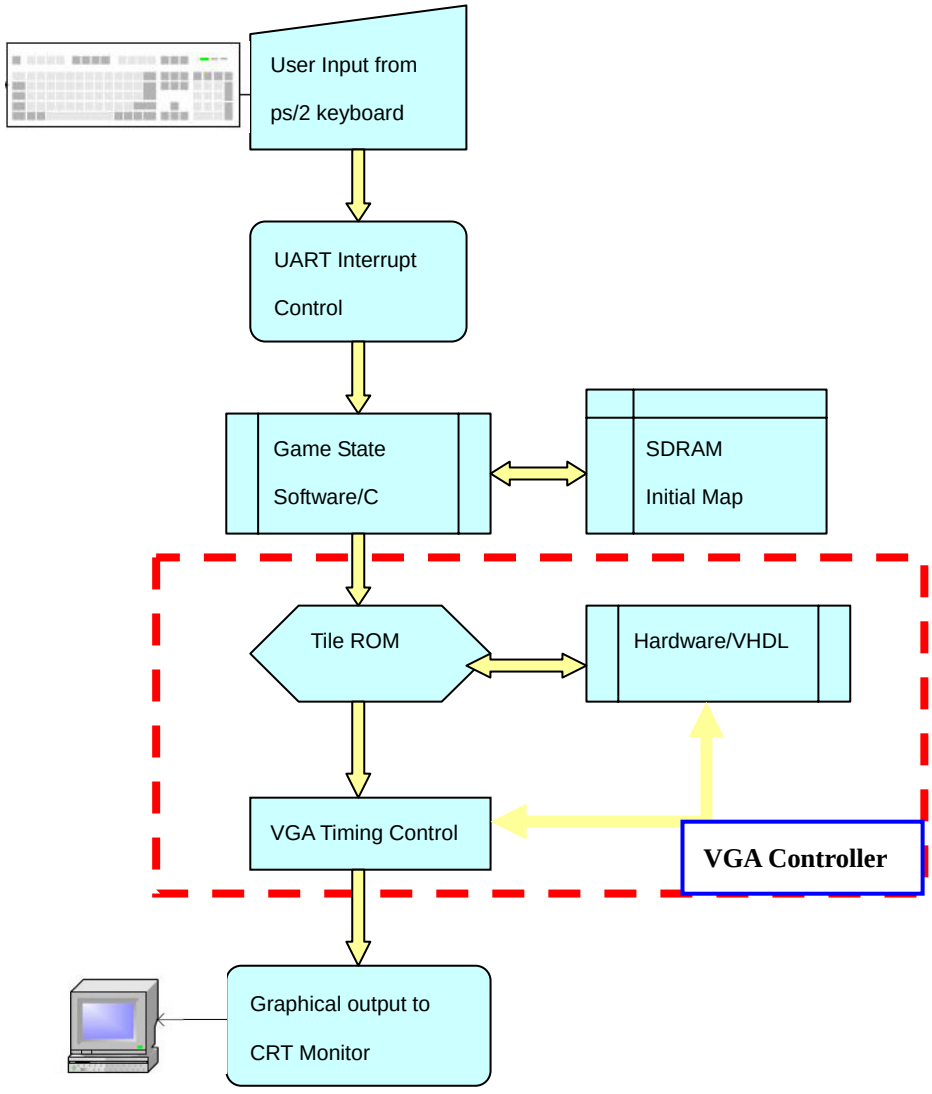
2.2 Hardware/VHDL/Video Control

In our labs in class, we learned to implement an instant video display on CRT screen. So we develop our design on this basis. The block (tile) size is defined by 8x16 pixels, and the components of the game displayed on the screen are constructed of these 4x2 blocks (tile) called “Big-block”. Our tiles are divided into 3 sets including walls, car, and flag. All of them are constructed of 1 Big-block with 32x32 pixels. For video control, we use timing delay to solve char-column and char-raw display non-harmonious issue. Throughout the implementation, vga_timing tracks the time and the address currently displayed on the screen. It is important to keep track of horizontal and vertical lines on the screen in order to synchronize loading and displaying. These component signals are various other components within the VGA. In addition, we add the RGB setting of each tile for color display.

2.3 Software/C/Game State

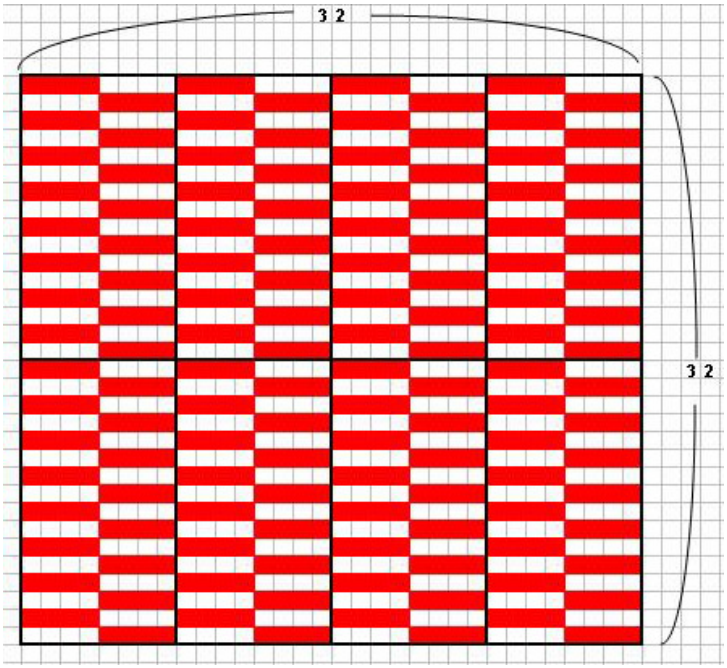
The VGA screen has 80x30 blocks, and the game map of our design is 160x60 that is 4 times of VGA size. That is the map is consisted of 40x30 Big-blocks. Therefore, we set the car in the center of screen and use scrolling screen to display the car moving. The game map is defined and saved in SDRAM initially. Whenever receiving UART interrupt from ps/2 keyboard like up, down, left, or right, we shift the map to corresponding position and the car changes its corresponding direction block. If determining the walls or boundary of the map, we will stop the car moving in the same direction.

3. Block diagram



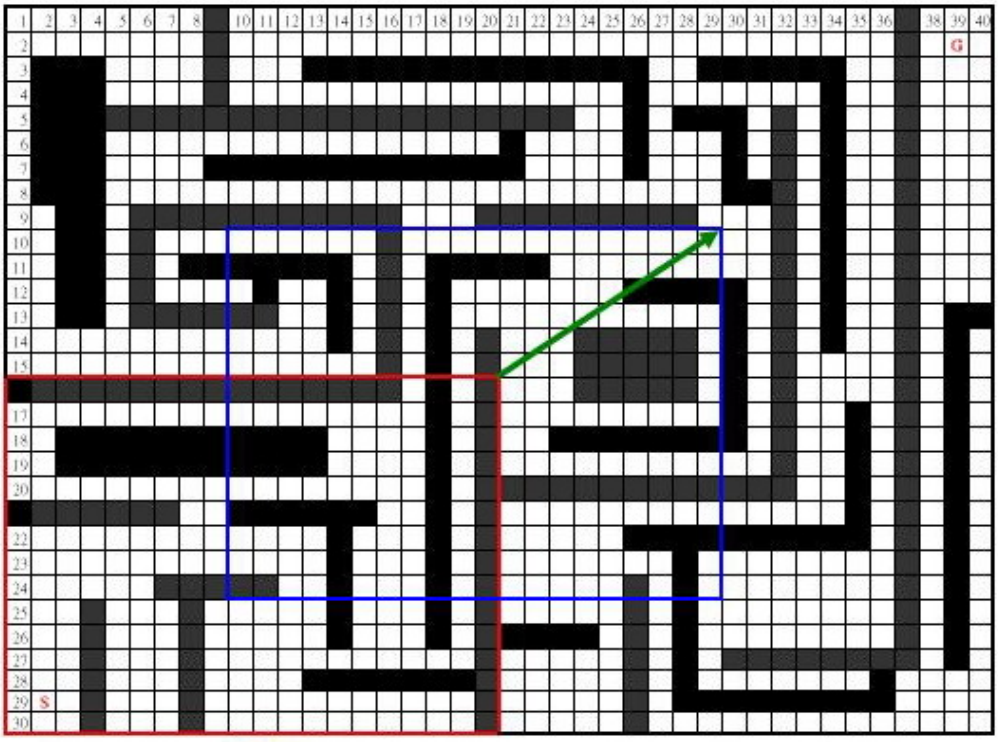
4. Block illustration

Wall



1 Big-block is consisted of 8 tiles (block)

Game map diagram



5. Code

//opb_s3board_vga.vhd

```
--------------------------------------------------------------------------------
--
-- Text-mode VGA controller for the Digilent Spartan 3 starter board
--
-- Uses an OPB interface, e.g., for use with the Microblaze soft core
--
-- Stephen A. Edwards
-- sedwards@cs.columbia.edu
-- Modified by C.C. Liu and Y.D. Lee
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity opb_s3board_vga is

  generic (
    C_OPB_AWIDTH : integer                := 32;
    C_OPB_DWIDTH : integer                := 32;
    C_BASEADDR   : std_logic_vector(31 downto 0) := X"FEFF1000";
    C_HIGHADDR   : std_logic_vector(31 downto 0) := X"FEFF1FFF");

  port (
    OPB_Clk       : in std_logic;
    OPB_Rst       : in std_logic;

    -- OPB signals
    OPB_ABus      : in std_logic_vector (31 downto 0);
    OPB_BE        : in std_logic_vector (3 downto 0);
    OPB_DBus      : in std_logic_vector (31 downto 0);
    OPB_RNW       : in std_logic;
    OPB_select    : in std_logic;
    OPB_seqAddr   : in std_logic;

    VGA_DBus      : out std_logic_vector (31 downto 0);
```

```
    VGA_errAck    : out std_logic;
    VGA_retry     : out std_logic;
    VGA_toutSup   : out std_logic;
    VGA_xferAck   : out std_logic;

    Pixel_Clock_2x : in std_logic;

    -- Video signals
    VIDOUT_CLK    : out std_logic;
    VIDOUT_RED    : out std_logic;
    VIDOUT_GREEN  : out std_logic;
    VIDOUT_BLUE   : out std_logic;
    VIDOUT_HSYNC : out std_logic;
    VIDOUT_VSYNC : out std_logic);

end opb_s3board_vga;

architecture Behavioral of opb_s3board_vga is

  constant BASEADDR : std_logic_vector(31 downto 0) := X"FEFF1000";

  -- Video parameters

  constant HTOTAL : integer := 800;
  constant HSYNC : integer := 96;
  constant HBACK_PORCH : integer := 48;
  constant HACTIVE : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL : integer := 525;
  constant VSYNC : integer := 2;
  constant VBACK_PORCH : integer := 33;
  constant VACTIVE : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  -- 512 X 8 dual-ported Xilinx block RAM
  component RAMB4_S8_S8
    port (
      DOA   : out std_logic_vector (7 downto 0);
```

```
    ADDRA : in std_logic_vector (8 downto 0);
    CLKA  : in std_logic;
    DIA   : in std_logic_vector (7 downto 0);
    ENA   : in std_logic;
    RSTA  : in std_logic;
    WEA   : in std_logic;
    DOB   : out std_logic_vector (7 downto 0);
    ADDRB : in std_logic_vector (8 downto 0);
    CLKB  : in std_logic;
    DIB   : in std_logic_vector (7 downto 0);
    ENB   : in std_logic;
    RSTB  : in std_logic;
    WEB   : in std_logic);
end component;

-- Signals latched from the OPB
signal ABus : std_logic_vector (31 downto 0);
signal DBus : std_logic_vector (31 downto 0);
signal RNW  : std_logic;
signal select_delayed : std_logic;

-- Latched output signals for the OPB
signal DBus_out : std_logic_vector (31 downto 0);

-- Signals for the OPB-mapped RAM
signal ChipSelect : std_logic;          -- Address decode
signal MemCycle1, MemCycle2 : std_logic;  -- State bits
signal RamPageAddress : std_logic_vector(2 downto 0);
signal RamSelect : std_logic_vector (7 downto 0);
signal RST, WE : std_logic_vector (7 downto 0);
signal DOUT0, DOUT1, DOUT2, DOUT3 : std_logic_vector(7 downto 0);
signal DOUT4, DOUT5, DOUT6, DOUT7 : std_logic_vector(7 downto 0);
signal ReadData : std_logic_vector(7 downto 0);

-- Signals for the video controller
signal Pixel_Clock : std_logic;      -- 25 MHz clock divided from 50 MHz
signal LoadNShift : std_logic;       -- Shift register control
signal FontData : std_logic_vector(7 downto 0);   -- Input to shift register
signal ShiftData : std_logic_vector(7 downto 0);  -- Shift register data
```

```
signal VideoData : std_logic;       -- Serial out ANDed with blanking

signal Hcount : std_logic_vector(9 downto 0);  -- Horizontal position (0-640)
signal Vcount : std_logic_vector(9 downto 0);  -- Vertical position (0-480)
signal HBLANK_N, VBLANK_N : std_logic;   -- Blanking signals

signal FontLoad, LoadChar : std_logic;  -- Font/Character RAM read triggers
signal FontAddr : std_logic_vector(10 downto 0);
signal FontAddrTemp,FontAddrTemp2 : std_logic_vector(10 downto 0);

signal CharRamPage : std_logic_vector(2 downto 0);
signal CharRamSelect_N : std_logic_vector(4 downto 0);
signal FontRamPage : std_logic_vector(1 downto 0);
signal FontRamSelect_N : std_logic_vector(2 downto 0);
signal CharAddr : std_logic_vector(11 downto 0);
signal CharColumn : std_logic_vector(9 downto 0);
signal CharRow : std_logic_vector(9 downto 0);
signal Column : std_logic_vector(6 downto 0); -- 0-79
signal Row : std_logic_vector(4 downto 0);   -- 0-29
signal EndOfLine, EndOfField : std_logic;


signal DOUTB0, DOUTB1, DOUTB2, DOUTB3 : std_logic_vector(7 downto 0);
signal DOUTB4, DOUTB5, DOUTB6, DOUTB7 : std_logic_vector(7 downto 0);

signal FontDataTemp : std_logic_vector(7 downto 0);



attribute INIT_00 : string;
attribute INIT_01 : string;
attribute INIT_02 : string;
attribute INIT_03 : string;
attribute INIT_04 : string;
attribute INIT_05 : string;
attribute INIT_06 : string;
attribute INIT_07 : string;
attribute INIT_08 : string;
attribute INIT_09 : string;
```

```
  attribute INIT_0a : string;
  attribute INIT_0b : string;
  attribute INIT_0c : string;
  attribute INIT_0d : string;
  attribute INIT_0e : string;
  attribute INIT_0f : string;

  attribute INIT_00 of RAMB4_S8_S8_0 : label is
  -- "21646c726f57206f6c6c6548";
     "2d2c2b2a2222222229282726";
  -- Standard 8x16 font taken from the Linux console font file "lat0-16.psfu"

  attribute INIT_00 of RAMB4_S8_S8_5 : label is
   "00000000181800181818183c3c3c180000000000000000000000000000000000";-- 1+32 --0+32
   --   ! space
   ------------------------------------------------------------------------------start
  attribute INIT_01 of RAMB4_S8_S8_5 : label is
   "ffffffffffffffffffffffffffffff0000000000000000000000000000000000";--  3+32  0x23  --
2+32 0x22
  --white  black
  attribute INIT_02 of RAMB4_S8_S8_5 : label is
   "0f0ff0f00f0ff0f00f0ff0f00f0ff0f00f0ff0f00f0ff0f00f0ff0f00f0ff0f0";-- 0x25  0x24
  --wall wall
--------------------------------------------------------------------------------
  attribute INIT_03 of RAMB4_S8_S8_5 : label is
  "e7e7e7672a35eaf5eaf52a3f1f03010000000000003c3c3f3f3c3c0000000000";-- 0x27  0x26
  -- Car ud2 + Car ud1
  attribute INIT_04 of RAMB4_S8_S8_5 : label is
  "00000000003c3cfcfc3c3c0000000000e7e7e7e654ac57af57af54fcf8c08000";-- 0x29  0x28
  -- Car ud4 + Car ud3
  attribute INIT_05 of RAMB4_S8_S8_5 : label is
   "0001031f3f2af5eaf57a352a67e7e7e70000003c3c3c3f3f3f3f3c3c3c000000";-- 0x2b  0x2a
  -- Car ud6 + Car ud5
  attribute INIT_06 of RAMB4_S8_S8_5 : label is
   "0000003c3c3cfcfcfcfc3c3c3c0000000080c0f8fc54af57af57ac54e6e7e7e7";-- 0x2d  0x2c
  -- Car ud8 + Car ud7
   --------------------------------------------------------------------------------
  attribute INIT_07 of RAMB4_S8_S8_5 : label is
   "af5faf5050ffcfc7c0c0f0f0f0f000007a3d1a1d1a0f030303030f0f0f0f0000";-- 0x2f 0x2e
```

```
  -- Car lr2 + Car lr1
  attribute INIT_08 of RAMB4_S8_S8_5 : label is
   "5ebc58b858f0c0c0c0c0f0f0f0f00000f5faf50a05fff3e303030f0f0f0f0000";-- 0x31 0x30
   -- Car lr4 + Car lr3
  attribute INIT_09 of RAMB4_S8_S8_5 : label is
   "0000f0f0f0f0c0c0c7cfffa050af5faf00000f0f0f0f030303030f1a1d1a3d7a";-- 0x33 0x32
   -- Car lr6 + Car lr5
  attribute INIT_0a of RAMB4_S8_S8_5 : label is
   "0000f0f0f0f0c0c0c0c0f058b858bc5e00000f0f0f0f0303e3f3ff050af5faf5";-- 0x35 0x34
   -- Car lr8 + Car lr7
   --------------------------------------------------------------------------------
  attribute INIT_0b of RAMB4_S8_S8_5 : label is
   "fffffffff7f3f1f0f070301000000000f070301000000000000000000000000";-- 0x37 0x36
   -- P2 P1
  attribute INIT_0c of RAMB4_S8_S8_5 : label is
   "0000000000000000000000000000000080808080808080808080808080800000";-- 0x39 0x38
   -- P4 P3
  attribute INIT_0d of RAMB4_S8_S8_5 : label is
   "000f2bd73b0701010101010101010101000000000000000000000000000000000";-- 0x3b 0x3a
   -- P6 P5
  attribute INIT_0e of RAMB4_S8_S8_5 : label is
   "00000000000000000000000000000000f0d4ebdce080808080808080808080";-- 0x3d 0x3c
   -- P8 P7
   --------------------------------------------------------------------------------
  attribute INIT_0f of RAMB4_S8_S8_5 : label is
   "000000001818001818180cc6c67c000000000006030180c060c183060000000";-- 31+32 --30+32
   --------------------------------------------------------------------------------
  attribute INIT_00 of RAMB4_S8_S8_6 : label is
   "00000000c6c6c6c6fec6c66c38100000000000007cc0dcdededec6c6c67c0000";--A @ -- 0x41 0x40
  attribute INIT_01 of RAMB4_S8_S8_6 : label is
   "000000003c66c2c0c0c0c0c2663c000000000000fc666666667c666666fc0000";--C B
  attribute INIT_02 of RAMB4_S8_S8_6 : label is
   "00000000fe6662606878686266fe000000000000f86c6666666666666cf80000";--E D
  attribute INIT_03 of RAMB4_S8_S8_6 : label is
   "000000003a66c6c6dec0c0c2663c000000000000f06060606878686266fe0000";--G F
  attribute INIT_04 of RAMB4_S8_S8_6 : label is
   "000000003c181818181818183c000000000000c6c6c6c6c6fec6c6c6c60000";--  41+32  --40+32
-- I H
  attribute INIT_05 of RAMB4_S8_S8_6 : label is
```

```
   "00000000e666666c78786c6666e600000000000078cccccc0c0c0c0c0c1e0000";-- K J
  attribute INIT_06 of RAMB4_S8_S8_6 : label is
   "00000000c6c6c6c6c6d6fefeeec6000000000000fe66626060606060f00000";-- M L
  attribute INIT_07 of RAMB4_S8_S8_6 : label is
   "000000007cc6c6c6c6c6c6c6c67c000000000000c6c6c6c6cedefef6e6c60000";-- O N
  attribute INIT_08 of RAMB4_S8_S8_6 : label is
   "00000e0c7cded6c6c6c6c6c6c67c000000000000f0606060607c666666fc0000";-- Q P
  attribute INIT_09 of RAMB4_S8_S8_6 : label is
   "000000007cc6c6060c3860c6c67c000000000000e66666666c7c666666fc0000";--  51+32  --50+32
-- S R
  attribute INIT_0a of RAMB4_S8_S8_6 : label is
   "000000007cc6c6c6c6c6c6c6c6c60000000000003c1818181818185a7e7e0000";-- U T
  attribute INIT_0b of RAMB4_S8_S8_6 : label is
   "000000006ceefed6d6d6c6c6c6c6c60000000000000010386cc6c6c6c6c6c6c60000";-- W V
  attribute INIT_0c of RAMB4_S8_S8_6 : label is
   "000000003c181818183c666666660000000000000c6c66c7c38387c6cc6c60000";-- Y X
  attribute INIT_0d of RAMB4_S8_S8_6 : label is
   "000000003c303030303030303c000000000000fec6c26030180c86c6fe0000";--  [ Z -- 5B 5A
  attribute INIT_0e of RAMB4_S8_S8_6 : label is
   "000000003c0c0c0c0c0c0c0c0c3c0000000000000000060c183060c00000000000";-- 61+32 --60+32
  attribute INIT_0f of RAMB4_S8_S8_6 : label is
   "00ff0000000000000000000000000000000000000000000000000000000c66c3810";
  attribute INIT_00 of RAMB4_S8_S8_7 : label is
   "0000000076cccccc7c0c78000000000000000000000000000000001830303000";
  attribute INIT_01 of RAMB4_S8_S8_7 : label is
   "000000007cc6c0c0c0c67c00000000000000000007c666666666c786060e00000";
  attribute INIT_02 of RAMB4_S8_S8_7 : label is
   "000000007cc6c0c0fec67c000000000000000000076cccccccc6c3c0c0c1c0000";
  attribute INIT_03 of RAMB4_S8_S8_7 : label is
   "0078cc0c7ccccccccccc760000000000000000000f060606060f060646c380000";-- 71+32 --70+32
  attribute INIT_04 of RAMB4_S8_S8_7 : label is
   "000000003c1818181818380018180000000000000e666666666766c6060e00000";
  attribute INIT_05 of RAMB4_S8_S8_7 : label is
   "00000000e6666c78786c666060e00000003c666606060606060e0006060000";
  attribute INIT_06 of RAMB4_S8_S8_7 : label is
   "00000000c6d6d6d6d6feec0000000000000000000018343030303030303030700000";
  attribute INIT_07 of RAMB4_S8_S8_7 : label is
   "000000007cc6c6c6c6c67c000000000000000000006666666666666dc0000000000";
  attribute INIT_08 of RAMB4_S8_S8_7 : label is
```

```
  "001e0c0c7ccccccccccc76000000000000f060607c6666666666dc0000000000";-- 81+32 --80+32
 attribute INIT_09 of RAMB4_S8_S8_7 : label is
  "000000007cc60c3860c67c00000000000000000000f06060606676dc0000000000";
 attribute INIT_0a of RAMB4_S8_S8_7 : label is
  "0000000076ccccccccccccc0000000000000000001c3630303030fc3030100000";
 attribute INIT_0b of RAMB4_S8_S8_7 : label is
  "000000006cfed6d6d6c6c60000000000000000000183c66666666660000000000";
 attribute INIT_0c of RAMB4_S8_S8_7 : label is
  "00f80c067ec6c6c6c6c6c60000000000000000000c66c3838386cc60000000000";
 attribute INIT_0d of RAMB4_S8_S8_7 : label is
  "000000000e18181818701818180e000000000000fec6603018ccfe0000000000";-- 91+32 --90+32
 attribute INIT_0e of RAMB4_S8_S8_7 : label is
  "00000000701818181800e1818187000000000000001818181818181818180000";
 attribute INIT_0f of RAMB4_S8_S8_7 : label is
  "000000003c1818183c666666660066000000000000000000000000000dc760000"; -- 95+32 --94+32



begin  -- Behavioral

  ---------------------------------------------------------------------------
  --
  -- Instances of the block RAMs
  -- Each is 512 bytes, so 4K total
  --
  ---------------------------------------------------------------------------

  -- First 2.5K is used for characters (5 block RAMs)
  -- Remaining 1.5K holds the font    (3 block RAMs)
  --

  -- Port A is used for communication with the OPB
  -- Port B is for video

  RAMB4_S8_S8_0 : RAMB4_S8_S8
    port map (
      DOA   => DOUT0,
      ADDRA => ABus(8 downto 0),
      CLKA  => OPB_Clk,
```

```
    DIA   => DBus(7 downto 0),
    ENA   => '1',
    RSTA  => RST(0),
    WEA   => WE(0),
    DOB   => DOUTB0,
    ADDRB => CharAddr(8 downto 0),
    CLKB  => Pixel_Clock,
    DIB   => X"00",
    ENB   => '1',
    RSTB  => CharRamSelect_N(0),
    WEB   => '0');

RAMB4_S8_S8_1 : RAMB4_S8_S8
  port map (
    DOA   => DOUT1,
    ADDRA => ABus(8 downto 0),
    CLKA  => OPB_Clk,
    DIA   => DBus(7 downto 0),
    ENA   => '1',
    RSTA  => RST(1),
    WEA   => WE(1),
    DOB   => DOUTB1,
    ADDRB => CharAddr(8 downto 0),
    CLKB  => Pixel_Clock,
    DIB   => X"00",
    ENB   => '1',
    RSTB  => CharRamSelect_N(1),
    WEB   => '0');

  RAMB4_S8_S8_2 : RAMB4_S8_S8
  port map (
    DOA   => DOUT2,
    ADDRA => ABus(8 downto 0),
    CLKA  => OPB_Clk,
    DIA   => DBus(7 downto 0),
    ENA   => '1',
    RSTA  => RST(2),
    WEA   => WE(2),
    DOB   => DOUTB2,
```

```
  ADDRB => CharAddr(8 downto 0),
  CLKB  => Pixel_Clock,
  DIB   => X"00",
  ENB   => '1',
  RSTB  => CharRamSelect_N(2),
  WEB   => '0');

RAMB4_S8_S8_3 : RAMB4_S8_S8
port map (
  DOA   => DOUT3,
  ADDRA => ABus(8 downto 0),
  CLKA  => OPB_Clk,
  DIA   => DBus(7 downto 0),
  ENA   => '1',
  RSTA  => RST(3),
  WEA   => WE(3),
  DOB   => DOUTB3,
  ADDRB => CharAddr(8 downto 0),
  CLKB  => Pixel_Clock,
  DIB   => X"00",
  ENB   => '1',
  RSTB  => CharRamSelect_N(3),
  WEB   => '0');

RAMB4_S8_S8_4 : RAMB4_S8_S8
port map (
  DOA   => DOUT4,
  ADDRA => ABus(8 downto 0),
  CLKA  => OPB_Clk,
  DIA   => DBus(7 downto 0),
  ENA   => '1',
  RSTA  => RST(4),
  WEA   => WE(4),
  DOB   => DOUTB4,
  ADDRB => CharAddr(8 downto 0),
  CLKB  => Pixel_Clock,
  DIB   => X"00",
  ENB   => '1',
  RSTB  => CharRamSelect_N(4),
```

```
  WEB   => '0');

RAMB4_S8_S8_5 : RAMB4_S8_S8
port map (
  DOA   => DOUT5,
  ADDRA => ABus(8 downto 0),
  CLKA  => OPB_Clk,
  DIA   => DBus(7 downto 0),
  ENA   => '1',
  RSTA  => RST(5),
  WEA   => WE(5),
  DOB   => DOUTB5,
  ADDRB => FontAddr(8 downto 0),
  CLKB  => Pixel_Clock,
  DIB   => X"00",
  ENB   => '1',
  RSTB  => FontRamSelect_N(0),
  WEB   => '0');

RAMB4_S8_S8_6 : RAMB4_S8_S8
port map (
  DOA   => DOUT6,
  ADDRA => ABus(8 downto 0),
  CLKA  => OPB_Clk,
  DIA   => DBus(7 downto 0),
  ENA   => '1',
  RSTA  => RST(6),
  WEA   => WE(6),
  DOB   => DOUTB6,
  ADDRB => FontAddr(8 downto 0),
  CLKB  => Pixel_Clock,
  DIB   => X"00",
  ENB   => '1',
  RSTB  => FontRamSelect_N(1),
  WEB   => '0');

RAMB4_S8_S8_7 : RAMB4_S8_S8
port map (
  DOA   => DOUT7,
```

```
    ADDRA => ABus(8 downto 0),
    CLKA  => OPB_Clk,
    DIA   => DBus(7 downto 0),
    ENA   => '1',
    RSTA  => RST(7),
    WEA   => WE(7),
    DOB   => DOUTB7,
    ADDRB => FontAddr(8 downto 0),
    CLKB  => Pixel_Clock,
    DIB   => X"00",
    ENB   => '1',
    RSTB  => FontRamSelect_N(2),
    WEB   => '0');


------------------------------------------------------------------------------
--
-- OPB-RAM controller
--
------------------------------------------------------------------------------

-- Unused OPB control signals
VGA_errAck <= '0';
VGA_retry <= '0';
VGA_toutSup <= '0';

-- Latch the relevant OPB signals from the OPB, since they arrive late
LatchOPB: process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Abus <= ( others => '0' );
    DBus <= ( others => '0' );
    RNW <= '1';
    select_delayed <= '0';
  elsif OPB_Clk'event and OPB_Clk = '1' then
    ABus <= OPB_ABus;
    DBus <= OPB_DBus;
    RNW <= OPB_RNW;
    select_delayed <= OPB_Select;
```

```
  end if;
end process LatchOPB;

-- Address bits 31 downto 12 is our chip select
-- 11 downto 9 is the RAM page select
-- 8 downto 0 is the RAM byte select

ChipSelect <=
  '1' when select_delayed = '1' and
     (ABus(31 downto 12) = BASEADDR(31 downto 12)) and
     MemCycle1 = '0' and MemCycle2 = '0' else
  '0';

RamPageAddress <= ABus(11 downto 9);

RamSelect <=
  "00000001" when RamPageAddress = "000" else
  "00000010" when RamPageAddress = "001" else
  "00000100" when RamPageAddress = "010" else
  "00001000" when RamPageAddress = "011" else
  "00010000" when RamPageAddress = "100" else
  "00100000" when RamPageAddress = "101" else
  "01000000" when RamPageAddress = "110" else
  "10000000" when RamPageAddress = "111" else
  "00000000";

MemCycleFSM : process(OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    MemCycle1 <= '0';
    MemCycle2 <= '0';
  elsif OPB_Clk'event and OPB_Clk = '1' then
    MemCycle2 <= MemCycle1;
    MemCycle1 <= ChipSelect;
  end if;
end process MemCycleFSM;

VGA_xferAck <= MemCycle2;
```

```
WE <=
RamSelect when ChipSelect = '1' and RNW = '0' and OPB_Rst = '0' else
  "00000000";

RST <=
 not RamSelect when ChipSelect = '1' and RNW = '1' and OPB_Rst = '0' else
  "11111111";

ReadData <= DOUT0 or DOUT1 or DOUT2 or DOUT3 or
           DOUT4 or DOUT5 or DOUT6 or DOUT7 when MemCycle1 = '1'
           else "00000000";

-- DBus(31 downto 24) is the byte for addresses ending in 0

GenDOut: process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    DBus_out <= ( others => '0');
  elsif OPB_Clk'event and OPB_Clk = '1' then
    DBus_out <= ReadData & ReadData & ReadData & ReadData;
  end if;
end process GenDOut;

VGA_DBus <= DBus_out;

------------------------------------------------------------------------------
--
-- Video controller
--
------------------------------------------------------------------------------

-- Pixel clock divider

Pixel_clk_divider : process (Pixel_Clock_2x, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Pixel_Clock <= '0';
  elsif Pixel_Clock_2x'event and Pixel_Clock_2x = '1' then
    Pixel_Clock <= not Pixel_Clock;
```

```
  end if;
end process Pixel_clk_divider;

-- Horizontal and vertical counters

HCounter : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Hcount <= (others => '0');
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine = '1' then
      Hcount <= (others => '0');
    else
      Hcount <= Hcount + 1;
    end if;
  end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter: process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Vcount <= (others => '0');
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK
```

```
HSyncGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_HSYNC <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine = '1' then
      VIDOUT_HSYNC <= '1';
    elsif Hcount = HSYNC - 1 then
      VIDOUT_HSYNC <= '0';
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    HBLANK_N <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if Hcount = HSYNC + HBACK_PORCH - 1 then
      HBLANK_N <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE - 1 then
      HBLANK_N <= '0';
    end if;
  end if;
end process HBlankGen;

VSyncGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_VSYNC <= '1';
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine ='1' then
      if EndOfField = '1' then
        VIDOUT_VSYNC <= '1';
      elsif VCount = VSYNC - 1 then
        VIDOUT_VSYNC <= '0';
      end if;
    end if;
  end if;
```

```
  end process VSyncGen;

  VBlankGen : process (Pixel_Clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      VBLANK_N <= '0';
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
      if EndOfLine ='1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          VBLANK_N <= '1';
        elsif VCount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          VBLANK_N <= '0';
        end if;
      end if;
    end if;
  end process VBlankGen;

  -- RAM read triggers and shift register control

  --LoadChar   <= '1' when Hcount(2 downto 0) = X"5" else '0';
  FontLoad   <= '1' when Hcount(2 downto 0) = X"6" else '0';
  LoadNShift <= '1' when Hcount(2 downto 0) = X"7" else '0';




-------------------------------------------------------------------------------


-------------------------------------------------------------------------------

  -- Correction of 4 needed to calculate the character address before the
  -- character is displayed
  CharColumn <= Hcount - HSYNC - HBACK_PORCH + 4;
  Column <= CharColumn(9 downto 3);  -- /8
  CharRow <= Vcount - VSYNC - VBACK_PORCH;
  Row <= CharRow(8 downto 4);  -- / 16

  -- Column + Row * 80
  CharAddr <= Column +
              ("0" & Row(4 downto 0) & "000000") +
```

```
            ("000" & Row(4 downto 0) & "0000");

  CharRamPage <= CharAddr(11 downto 9);
  CharRamSelect_N <=
    "11110" when CharRamPage = "000" else
    "11101" when CharRamPage = "001" else
    "11011" when CharRamPage = "010" else
    "10111" when CharRamPage = "011" else
    "01111" when CharRamPage = "100" else
    "11111";

  -- Most significant bit of character ignored
  FontAddr(10 downto 4) <=
    (DOUTB0(6 downto 0) or DOUTB1(6 downto 0) or DOUTB2(6 downto 0) or
     DOUTB3(6 downto 0) or DOUTB4(6 downto 0));
  FontAddr(3 downto 0) <= CharRow(3 downto 0);

  -- Unusual addressing: font only holds 96 of 128 possible characters
  -- First 32 characters appear twice
  FontRamPage <= FontAddr(10 downto 9);
  FontRamSelect_N <=
    "110" when FontRamPage = "00" else
    "110" when FontRamPage = "01" else
    "101" when FontRamPage = "10" else
    "011" when FontRamPage = "11" else
    "111";
--------------------------------------------------------------------------------

     FontData <= DOUTB5 or DOUTB6 or DOUTB7;
DelayFontData: process (Pixel_Clock, OPB_Rst)
begin
     if OPB_Rst = '1' then
          FontDataTemp <= X"00";
     elsif Pixel_Clock'event and Pixel_Clock = '1' then
          if FontLoad = '1' then
               FontDataTemp <= FontData;
               FontAddrTemp2 <= FontAddr;
          end if;
     end if;
```

```
end process DelayFontData;


-------------------------------------------------------------------------
  -- Shift register

  ShiftRegister: process (Pixel_Clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      ShiftData <= X"AB";
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
      if LoadNShift = '1' then
        ShiftData <= FontDataTemp;
        FontAddrTemp <= FontAddrTemp2;
      else
        ShiftData <= ShiftData(6 downto 0) & ShiftData(7);
      end if;
    end if;
  end process ShiftRegister;

  VideoData <= ShiftData(7);

  -- Video signals going to the "video DAC"

  VideoOut: process (Pixel_Clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      VIDOUT_RED   <= '0';
      VIDOUT_BLUE  <= '0';
      VIDOUT_GREEN <= '0';
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
      if VideoData = '1' and HBLANK_N = '1' and VBLANK_N = '1' then
           if FontAddrTemp(10 downto 9) = "01" then
           if FontAddrTemp(8 downto 4) = "00100" then
                VIDOUT_RED   <= '1';
                VIDOUT_GREEN <= '0';
                VIDOUT_BLUE  <= '0';
           elsif FontAddrTemp(8 downto 4) = "00011" then
                VIDOUT_RED   <= '0';
```

```
                VIDOUT_GREEN <= '0';
                VIDOUT_BLUE  <= '1';
            elsif FontAddrTemp(8 downto 4) >= "00110" and  FontAddrTemp(8 downto 4) <= "10101"
then
                VIDOUT_RED   <= '1';
                VIDOUT_GREEN <= '1';
                VIDOUT_BLUE  <= '0';
            elsif FontAddrTemp(8 downto 4) >= "10110" and  FontAddrTemp(8 downto 4) <= "11101"
then
                VIDOUT_RED   <= '1';
                VIDOUT_GREEN <= '0';
                VIDOUT_BLUE  <= '1';
            else
                VIDOUT_RED   <= '1';
                VIDOUT_GREEN <= '1';
                VIDOUT_BLUE  <= '1';
                end if;
            else
                VIDOUT_RED   <= '1';
            VIDOUT_GREEN <= '1';
            VIDOUT_BLUE  <= '1';
            end if;
        elsif VideoData = '0' and HBLANK_N = '1' and VBLANK_N = '1' then
            if FontAddrTemp(10 downto 9) = "01" then
            if FontAddrTemp(8 downto 4) = "00100" then
                VIDOUT_RED   <= '1';
                VIDOUT_GREEN <= '1';
                VIDOUT_BLUE  <= '1';
            elsif FontAddrTemp(8 downto 4) = "00011" then
                VIDOUT_RED   <= '0';
                VIDOUT_GREEN <= '0';
                VIDOUT_BLUE  <= '1';
            elsif FontAddrTemp(8 downto 4) >= "00110" and  FontAddrTemp(8 downto 4) <= "10101"
then
                VIDOUT_RED   <= '0';
                VIDOUT_GREEN <= '0';
                VIDOUT_BLUE  <= '0';
            elsif FontAddrTemp(8 downto 4) >= "10110" and  FontAddrTemp(8 downto 4) <= "11101"
then
```

```
                VIDOUT_RED   <= '0';
                VIDOUT_GREEN <= '0';
                VIDOUT_BLUE  <= '0';
           else
                VIDOUT_RED   <= '0';
                VIDOUT_GREEN <= '0';
                VIDOUT_BLUE  <= '0';
                end if;
           else
                VIDOUT_RED   <= '0';
           VIDOUT_GREEN <= '0';
           VIDOUT_BLUE  <= '0';
           end if;
      else
        VIDOUT_RED   <= '0';
        VIDOUT_GREEN <= '0';
        VIDOUT_BLUE  <= '0';
      end if;
    end if;
  end process VideoOut;


  VIDOUT_CLK <= Pixel_Clock;

end Behavioral;
```

//TestApp.c

```
/*
 *  * Copyright (c) 2004 Xilinx, Inc.  All rights reserved.
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
 * COURTESY TO YOU.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
 * ONE POSSIBLE   IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
 * STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
 * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION
 * XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO
 * THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO
 * ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE
 * FROM CLAIMS OF INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.
 */
// Modified by C.C. Liu  Y.D. Lee
// Located in: microblaze_0/include/xparameters.h
#include "xparameters.h"

#include "xutil.h"
#include "xintc_l.h"
#include "xuartlite_l.h"
#include "xgpio_l.h"
////////////////////////////////////////////////////////////////////////////
#define uCAR 0x01
#define dCAR 0x02
#define lCAR 0x03
#define rCAR 0x04
#define pennant 0x05
#define wall 0x06
#define road 0x07
#define white 0x08

#define up 0x75
#define down 0x72
#define left 0x6B
#define right 0x74
```

```
////////////////////////////////////////////////////////////////////////////
#define FONT_OFFSET 0xA00

/* Address of a particular character on the screen (rows are 80) */
#define CHAR(c,r) \
   (((unsigned char *)(XPAR_VGA_BASEADDR))[(c) + ((r) << 6) + ((r) << 4)])

/* Start of font memory */
#define FONT ((unsigned char *)XPAR_VGA_BASEADDR + FONT_OFFSET)

////////////////////////////////////////////////////////////////////////////
void print_car(unsigned char uctemp);
void print_map(void);
unsigned char KBdir(void);

int x_col_map = 1;
int y_row_map = 28;
unsigned char exFlag = 0;
int x_flag = 38;
int y_flag = 0;

//40 col 30 row
unsigned char map[30][40] = {
 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0,},
 { 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,},
 { 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,},
 { 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,},
 { 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,},
 { 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,},
 { 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,},
```

```
 { 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,},
 { 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,},
 { 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0,},
 { 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,},
 { 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,},
 { 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,},
 { 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0,},
 { 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0,
0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,},
 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,},
 { 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,},
 { 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0,
1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,},
 { 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,},
 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,},
 { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,},
 { 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,},
 { 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,},
 { 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,},
 { 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,},
 { 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,},
 { 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 0,},
 { 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,},
 { 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
 { 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,}};

////////////////////////////////////////////////////////////////////////////

int uart_interrupt_count = 0;
char uart_character;

/* UART interrupt service routine */
void uart_int_handler(void *baseaddr_p) {

   /* While UART receive FIFO has data */
   while (!XUartLite_mIsReceiveEmpty(XPAR_RS232_BASEADDR)) {
     /* Read a character */
     uart_character = XUartLite_RecvByte(XPAR_RS232_BASEADDR);
     ++uart_interrupt_count;
   }
}

/* Must be a power of two */
#define SCANCODE_BUFFER_SIZE 16
unsigned char scancode_buffer[SCANCODE_BUFFER_SIZE];
int scancode_buffer_head = 0;
int scancode_buffer_tail = 0;

int character_available()
{
  int result;
  microblaze_disable_interrupts();
  result = scancode_buffer_head != scancode_buffer_tail;
  microblaze_enable_interrupts();
  return result;
}
```

```
unsigned char get_character()
{
  unsigned char result;
  microblaze_disable_interrupts();
  result = scancode_buffer[scancode_buffer_tail];
  scancode_buffer_tail = (scancode_buffer_tail + 1) & (SCANCODE_BUFFER_SIZE - 1);
  microblaze_enable_interrupts();
  return result;
}

int keyboard_interrupt_count = 0;
unsigned int scan_code = 0;
unsigned int keyboard_bit = 11;

/* PS/2 Keyboard interrupt service routine */
void ps2_int_handler(void *baseaddr_p)
{
  int next_head;
  keyboard_interrupt_count++;

  /* We're reading a scancode and the keyboard clock fell: sample the data */
  scan_code = (scan_code >> 1) |
      (XGpio_mReadReg(XPAR_PS2IO_BASEADDR, XGPIO_DATA_OFFSET) << 9);
  if (--keyboard_bit == 0) {
    /* keycode is ready */
    next_head = (scancode_buffer_head + 1) & (SCANCODE_BUFFER_SIZE - 1);
    if (next_head != scancode_buffer_tail) {
      /* buffer is not full; add the character */
      scancode_buffer[scancode_buffer_head] = scan_code;
      scancode_buffer_head = next_head;
    }
    keyboard_bit = 11;
    scan_code = 0;
  }

  /* Acknowledge the interrupt */
  XGpio_mWriteReg(XPAR_PS2IO_BASEADDR, XGPIO_ISR_OFFSET, 1);
}
```

```
/* Write a text string on the display */
void put_string(char *string, int column, int row)
{
  char *p = &(CHAR(column, row));
  while (*p++ = *string++)
    ;
}
void put_block(char Btype, int column, int row)
{
     unsigned char i,j;
     char uCARs[8] = {0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d};
     char dCARs[8] = {0x2a,0x27,0x28,0x2d,0x26,0x2b,0x2c,0x29};
     char lCARs[8] = {0x2e,0x2f,0x30,0x31,0x32,0x33,0x34,0x35};
     char rCARs[8] = {0x2e,0x2f,0x30,0x31,0x32,0x33,0x34,0x35};
     char pennants[8] = {0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d};
     char walls[8] = {0x24,0x24,0x24,0x24,0x24,0x24,0x24,0x24};
     char roads[8] = {0x22,0x22,0x22,0x22,0x22,0x22,0x22,0x22};
     char white_walls[8] = {0x23,0x23,0x23,0x23,0x23,0x23,0x23,0x23};
     char *p = &(CHAR(column, row));
     char *q = &(CHAR(column, (row+1)));
     for (j=0;j<=3;j++)
     {
          switch (Btype)
          {
               case road:
                    *p++ = roads[j];
                    break;
               case wall:
                    *p++ = walls[j];
                    break;
               case white:
                    *p++ = white_walls[j];
                    break;
               case uCAR:
                    *p++ = uCARs[j];
                    break;
               case dCAR:
                    *p++ = dCARs[j];
                    break;
```

```
            case lCAR:
                  *p++ = lCARs[j];
                  break;
            case rCAR:
                  *p++ = rCARs[j];
                  break;
            case pennant:
                  *p++ = pennants[j];
                  break;
      }
}
for (j=4;j<=7;j++)
{
      switch (Btype)
      {
            case road:
                  *q++ = roads[j];
                  break;
            case wall:
                  *q++ = walls[j];
                  break;
            case white:
                  *q++ = white_walls[j];
                  break;
            case uCAR:
                  *q++ = uCARs[j];
                  break;
            case dCAR:
                  *q++ = dCARs[j];
                  break;
            case lCAR:
                  *q++ = lCARs[j];
                  break;
            case rCAR:
                  *q++ = rCARs[j];
                  break;
            case pennant:
                  *q++ = pennants[j];
                  break;
```

```
          }
     }
}

int main (void) {
//   int i,j;
//   int x_col = 0;
//   int y_row = 0;
      unsigned char uctemp;
//   unsigned char exFlag = 0;



     print("Hello World!\r\n");    //put string to terminal screen

//   put_string("The Digilent Spartan 3 board says \"Hello from NCTU\"", 0, 1);


   /* Enable MicroBlaze interrupts */
   microblaze_enable_interrupts();
   /* Register the UART interrupt handler in the vector table */
   XIntc_RegisterHandler(XPAR_OPB_INTC_0_BASEADDR,XPAR_OPB_INTC_0_RS232_INTERRUPT_INTR,
      (XInterruptHandler)uart_int_handler,(void *)0);
   /* Register the keyboard interrupt handler */
   XIntc_RegisterHandler(XPAR_OPB_INTC_0_BASEADDR, XPAR_OPB_INTC_0_SYSTEM_PS2C_INTR,
      (XInterruptHandler)ps2_int_handler, (void *)0);
   /* Start the interrupt controller */
   XIntc_mMasterEnable(XPAR_OPB_INTC_0_BASEADDR);
   /* Enable timer and UART interrupt requests in the interrupt controller */
   XIntc_mEnableIntr(XPAR_OPB_INTC_0_BASEADDR,
                  XPAR_RS232_INTERRUPT_MASK | XPAR_SYSTEM_PS2C_MASK);
   /* Enable UART interrupts */
   XUartLite_mEnableIntr(XPAR_RS232_BASEADDR);

//  for (;;) {
//    if (character_available()) {
//         print("Got scancode ");
//         putnum(get_character());
//         print("\r\n");
```

```
//    }
//    for (i = 0 ; i < 1000 ; i++ )  ; /* delay */
//  }

//print the Map
print_map();
print_car(up);

put_string("~~ START GAME ~~",33,19);

for(;;)
{
     uctemp = KBdir();
     if (uctemp)
     {
          switch (uctemp)
          {
               case up:
                    y_row_map -= 1;
                    if ((map[y_row_map][x_col_map]==1)||(y_row_map<0))
                    {
                         y_row_map +=1;
                    }
                    break;
               case down:
                    y_row_map += 1;
                    if  ((map[y_row_map][x_col_map]==1)||(y_row_map>29))
                    {
                         y_row_map -=1;
                    }
                    break;
               case left:
                    x_col_map -= 1;
                    if  ((map[y_row_map][x_col_map]==1)||(x_col_map<0))
                    {
                         x_col_map += 1;
                    }
                    break;
               case right:
```

```
                    x_col_map += 1;
                    if  ((map[y_row_map][x_col_map]==1)||(x_col_map>39))
                    {
                         x_col_map -= 1;
                    }
                    break;
               default: break;
          }
          print_map();
          print_car(uctemp);
     }
     if ((x_col_map == x_flag)&&(y_row_map == y_flag))
     {
          put_string("~~ YOU WIN ^_^",35,19);
          return 0;
     }

}

     print("Goodbye! \r\n");
     return 0;
}

void print_map(void)
{
     unsigned char i,j;
     for (i=0;i<15;i++)
     {
          for (j=0;j<20;j++)
          {
               if
(((x_col_map-9+j)<0)||((y_row_map-7+i)<0)||((x_col_map-9+j)>39)||((y_row_map-7+i)>29))
               {
                    put_block(white,4*j,2*i);
                    continue;
               }
               if(map[y_row_map-7+i][x_col_map-9+j] == 2)
               {
                    put_block(pennant, 4*j, 2*i);
```

```
                }
                else if(map[y_row_map-7+i][x_col_map-9+j] == 1)
                {
                     put_block(wall, 4*j, 2*i);
                }
                else
                {
                     put_block(road, 4*j, 2*i);
                }
           }
     }
}
void print_car(unsigned char uctemp)
{
     switch(uctemp)
     {
          case up:
               put_block(uCAR,36,14);
               break;
          case down:
               put_block(dCAR,36,14);
               break;
          case left:
               put_block(lCAR,36,14);
               break;
          case right:
               put_block(rCAR,36,14);
               break;
     }
}

// Control
unsigned char KBdir(void)
{
     unsigned char uctemp;
     if (character_available())
     {
          uctemp = get_character();
          if (exFlag == 1)
```

```
        {
            switch (uctemp)
            {
                case 0xf0:
                    scancode_buffer_head = scancode_buffer_tail;
                    exFlag = 0;
                    return 0;
                case up:
                    exFlag = 0;
                    return up;
                case down:
                    exFlag = 0;
                    return down;
                case left:
                    exFlag = 0;
                    return left;
                case right:
                    exFlag = 0;
                    return right;
                default: return 0;
            }
        }
        if (uctemp == 0xe0)
        {
            exFlag = 1;
            return 0;
        }
        return 0;
    }
}
```