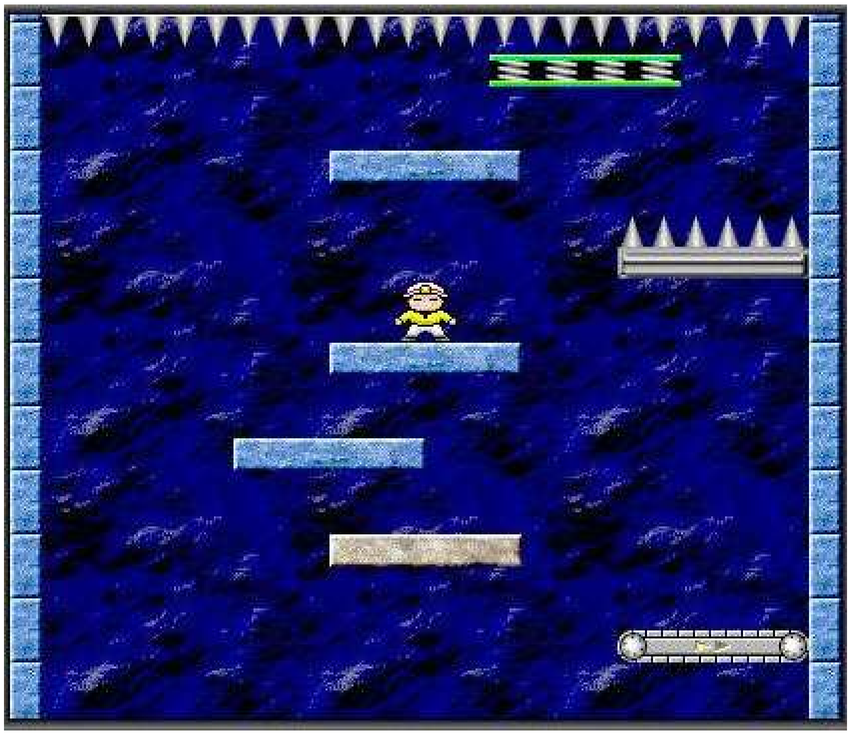
CSEE4840 Final Project Report

Watch Out!



Shangru Li
Zachary Salzbank

Introduction

Watch Out! is a game in which the player must jump from one platform to another in order to avoid hitting the top of the screen. The game gets progressively more difficult as time goes on. The platforms will begin to move faster and the distance between platforms will increase. The game has no ending, as long as the player can survive the increased difficulty level. The player's score increases when they jump from one platform to the next. The goal of the game is to get the highest score.

There are five different types of platforms:

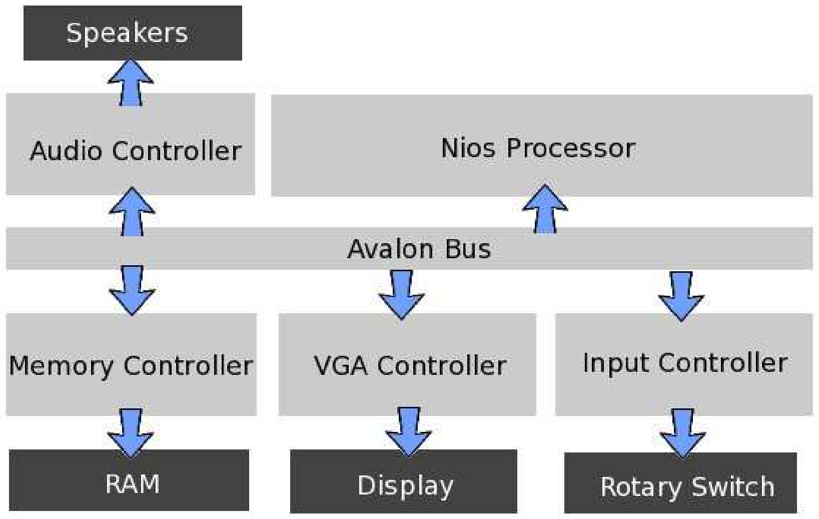
- Brick: A standard platform with no special properties.
- Sand: A platform that disappears after a short period of time.
- Spike: This platform reduces the health of the player and bounces them.
- Bounce: The player bounces when this platform is landed on.
- Health: Results in the player's health being increased by one heart.

Architecture

In order to create the game, multiple components will be required to provide the different functionalities needed. Controllers must be designed for each of these components:

- VGA Display (Video Controller)
- Speakers (Audio Controller)
- Rotary Switch (Input Controller)

These components will be controlled by a NIOS processor that will be programmed onto the FPGA. All of these components are connected together through the Avalon Bus. The main game software will be run on the processor.



Implementation

Video Controller

The objects shown on the VGA display are controlled by the video controller. The controller can be accessed via the Avalon bus. Once selected for usage (reading or writing), the two most significant bits of the address are used to determine what is being changed.

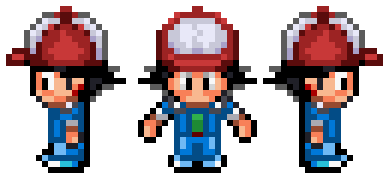
Address MSB	**Function**
00	Set Tile Type
01	Set Player Property
10	Set Tile Scroll Speed
11	Clear Display Update Interrupt

These functions allow the software to control all the aspects of what is being displayed. In general, there are two major sub-components to the display system: the player display and the tile system. In addition, an interrupt is used to notify the software about when to update these components. This interrupt prevents elements on the screen from flashing when they are not supposed to.

When determining what data to display on the screen for a specific pixel, the hardware first determines if the player should be displayed. If so, the player data for the specified coordinate is shown. Otherwise, tile data is shown.

Player Display

The player is the representation of the user in the game. The pixel data for the player is stored in a ROM on the FPGA.



Three properties are available to control the display of the player: x position, y position and facing direction. To select which property is being modified, the following scheme is used:

Address LSB	**Function**
00	X Position
01	Y Position
10	Facing Direction

Writing to these addresses modifies registers on the FPGA. These registers are used to determine if the player is being displayed at a certain coordinate, and if so, what facing direction should be shown on the screen.

Tile System

Everything displayed on the screen other than the player is part of the tile system. The screen is divided into 30 rows of 40 tiles - a total of 1200 tiles. Each tile is 16x16 pixels. Pixel data for each of the 35 types of tiles is stored in a ROM on the FPGA. In addition, there is a RAM consisting of 1280 positions which is able to accommodate 32 rows of 40 tiles. The extra two lines are used to allow the software to update what is being displayed with ample extra time before it is actually shown.

Setting Tiles

In order to modify what is being displayed in a specific tile position, the software modifies the RAM value corresponding to the position of the tile. The value at this position is set to the index of the tile that should be displayed.

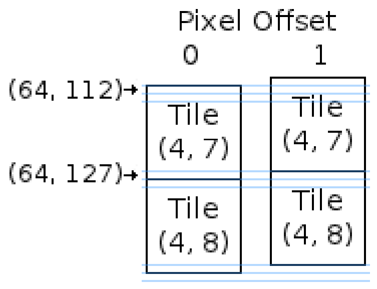
Address[10:6]	**Address[5:0]**
TIle Y Coordinate	Tile X Coordinate

The value from the RAM is used to lookup the specific pixel data in the ROM for the current coordinate being displayed.

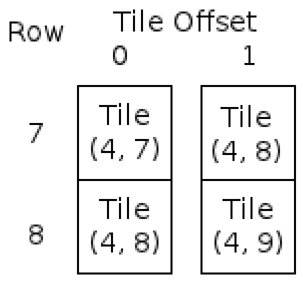
Tile Scrolling

In order to make the platforms look like they are scrolling up, two offset registers are used when determining what data to display for a certain position: the pixel offset and the tile offset.

The *pixel offset register* is used to determine how many pixels each tile should be shifted upwards from their base position. For example: if we were trying to display tile (4, 7) and pixel offset is zero, the pixel at (64, 112) would display data from whatever is referenced in the RAM at (4, 7). However, if we were trying to display the same tile, but pixel offset was set to one, the pixel at (64, 112) would display the second row of pixel data from whatever is referenced in the RAM at (4, 7). The pixel at (64, 127) would display the first row of pixel data from whatever is referenced in the RAM at (4, 8).



The *tile offset register* is used to offset the displayed tile by zero or more full tiles. When the current value of the pixel offset is 15, the tile offset register increases. The tile offset register allows the software to change only the next row of tiles when a new row is to be displayed, instead of changing every row in the RAM.



Both registers are accessible from the software by reading the vga controller (at any address).

Value[15:11]	**Value[10:7]**
Tile Offset	Pixel Offset

These values are helpful for determining if the player has landed on one of the platforms.

Stationary Rows

The two topmost rows in the RAM do not scroll. They are always shown as the top two rows of the display. This area is used to display information to the player, such as score and health.

Scroll Speed

The speed at which the tiles scroll is controlled by software. A *maximum counter value register* is used to determine how often the display is scrolled by one pixel. When the value of a counter reaches the maximum value, a pixel is scrolled. A larger value for this register will result in a slower scrolling speed. A special value for this register is 0, which means that no scrolling should occur.

Screen Refresh Interrupt

As mentioned above, an interrupt is used to notify the software when it can update the data that should be displayed on the screen. This eliminates unwanted flashes during updates. When the final pixel on the screen is drawn in a cycle, the interrupt is raised. To clear the interrupt, a value is written to any address that has the two most significant bits equal to 11.

Pixel Data Storage

Pixel data is stored in two separate ROMs: one for the player, and one for the tile data. The data is stored in MIF files and loaded into memory bits instead of storing it in logic cells.

The player pixels are stored as 25-bit values. Eight bits are used for each of red, green and blue. The final pixel is used to determine transparency (if the pixel should be displayed or not). A transparent pixel will allow the tile behind it to be displayed.

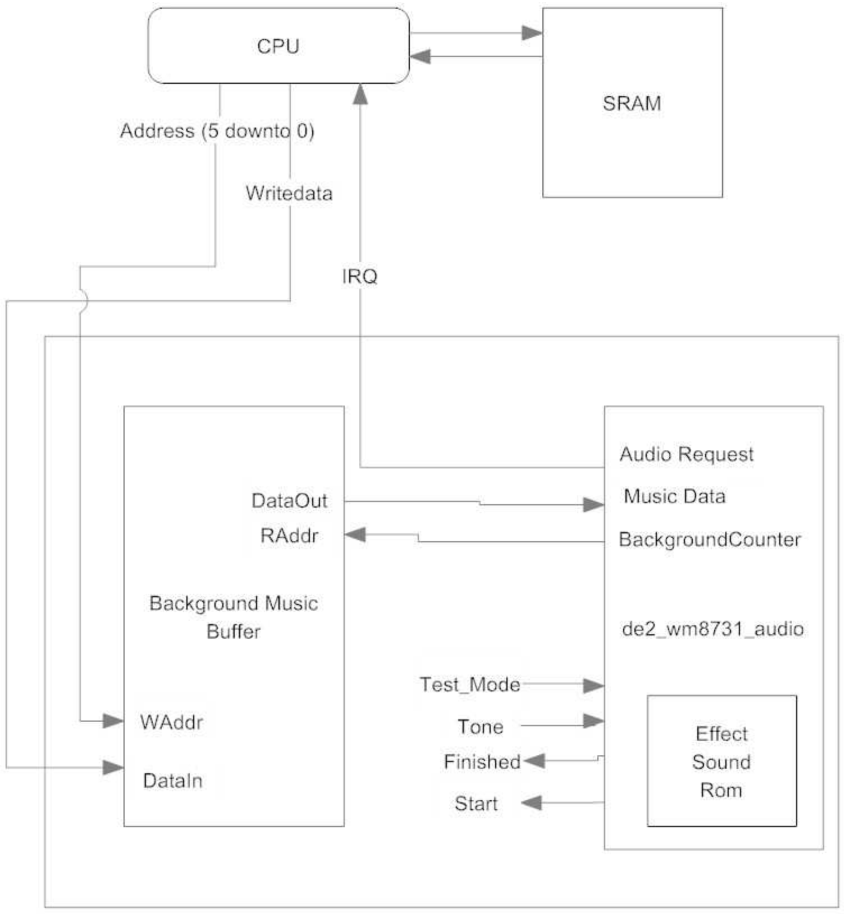
The tile pixels are stored as 16-bit values. Red and blue both use five bits each. Green uses six bits. Each tile is stored directly after the previous tile. This allows the ROM to be indexed by the tile number that should be displayed by simply multiplying the index by 256.

In order to generate the data, C++ scripts were created. They are included with the code below.

Audio Controller

The audio controller receives the CPU command to play a certain sound or receives the data CPU get from SRAM to play background music. The background music is sampled and played at *6000Hz*, and the sound effects are sampled and played at *3000Hz*.

The general architecture of Audio Controller is shown in the diagram below (with key signals shown):



The audio controller has a RAM for buffering the background music. The size of this buffer is *32 × 16 bits*. It has a write address, *Waddr*, which is determined by the CPU for writing new data. It also has a read address *Raddr*, which is controlled by the *de2_wm8731_audio* to provide a *6000Hz* sample frequency to read in background music data. When the data in the RAM is totally consumed (when the background counter reached 31), an interrupt is triggered to tell the CPU to feed more data into the RAM. Then the CPU will provide next 32 data words into this RAM from its own memory SRAM.

The background music data is generated from a piece of Matlab code, which takes in the *.wav* music file and exports series of 16-bit data which describe the amplitude of the music along the time. The sample frequency of the data is *6000Hz*. The data source is a cut piece of the song “It’s My Life”. The length of the music is about 30s. The size of the data of this piece of music is about *200000 × 16 bits* which consume about 80 percent of the SRAM (totally *512 kbytes*).

In addition to the background music, we also include two sound effects: a sound for when the player dies and a sound for when the player is bounced off some types of platforms. The SRAM has been consumed 80 percent by the background music, but we still have nearly 40 percent of the on-chip memory on FPGA after the implementation of the video part, so we use the on-chip memory for storing the two sound effects. The raw data is created in the same way with the background music, the length of each is no more than 1s and they totally consume about 15 percent of the on-chip memory.

Both the background music and sound effects can play at the same time. The CPU first selects whether the test_mode is on. If it is *de2_wm8731_audio* takes the sound effects as the source of data, rather than the background music buffer. The next step is that the CPU will select a *tone* to distinguish between the two sound effects, and finally the CPU gives a *start* command, and then the selected sound will be produced. After the sound effect is totally played, a signal *finished* will be set to tell the controller to resume the background music.

Interface

The detail interface between CPU and audio controller is as follows:
The component has a 6-bit address.

Address MSB	**Function**
1	Feed new data into the background music buffer
0	Set effect music parameters

MSB = ‘0’, address LSB	**Function**
00	Select tones
01	Set *start* signal for effect sound
10	Select whether to play background music only or just play it with the effect sound

MSB = ‘1’ Address (4 downto 0)	**Function**
Buffer Waddr <= Address (4 downto 0)	Writing new data into background music buffer

Sampling Rate Selection

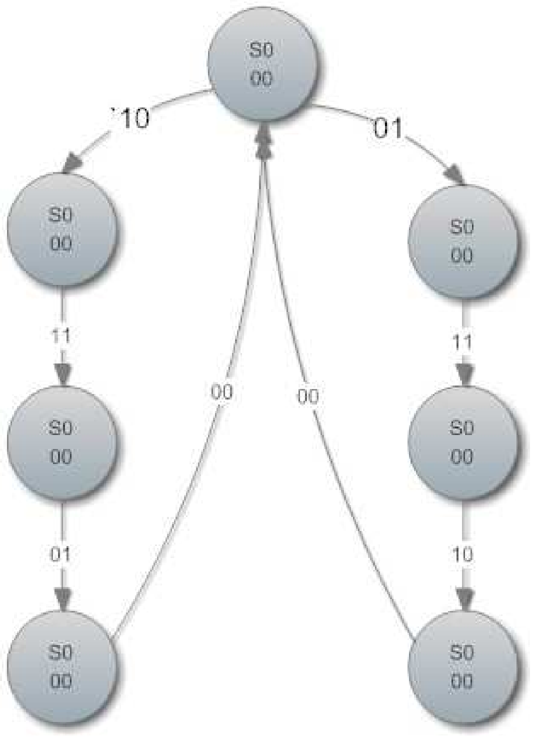
For the background music, we have tested *3000 Hz*, but the music is really intolerably unpleasant to the ears. However when the frequency is too high, for instance *12000 Hz*, the frequency of interrupts will be too high for the CPU to handle, so we have to use a larger buffer to store data for the music controller to consume to reduce the interrupt rate. But each time an interrupt occurs, more data will be written, which may probably make the software hang for a short period. Another negative effect is that the high sample frequency will lead to a smaller amount of music data that can fit in the 512 Kbytes SRAM.

But for the sound effects, we tested that *3000 Hz* is enough to produce a clear sound. To work with the same *lrck_divider* for *6000 Hz*, we just use the higher 11 bits of the 12 bit sin_counter for reading data from the sound effect ROM, which appears to slow down the frequency to half.

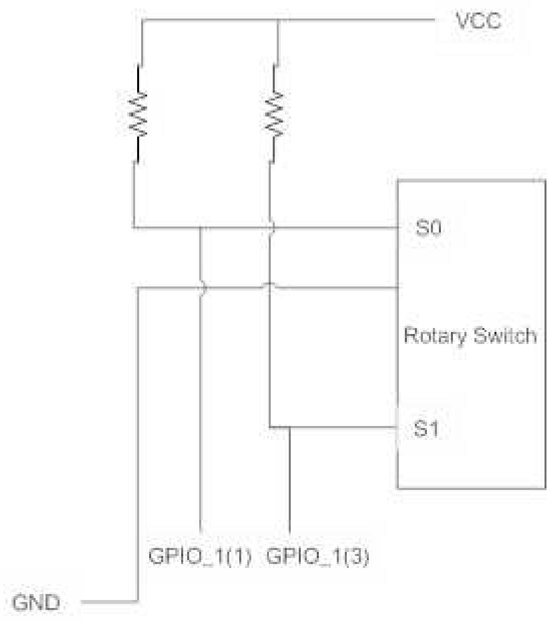
```
                    musicRom  port map (
                                        clk ,
                                        sin_counter (11 downto 1),
                                        sin_out0);
```

Input Controller

The rotary switch is able to control the movement of the player. The switch provides analog signals which are converted to digital signals accessible by the software. The rotary controller has two inputs, *s1* and *s2*, going into the GPIO of the DE2 board. We can track the pattern of the sequence of *s1* and *s2* to determine whether it is rotated clockwise or anticlockwise. If the sequence is *00, 10, 11, 01*, it is anti-clockwise. If the sequence is *00, 01, 11, 10*, it is clockwise. We can use two bits of output to represent different types of movements: output = "0X" means no movement, *output* = *"10"* means clockwise movement and *output* = *"11"* means anti-clockwise movement. So we can make a simple Finite State Machine to act as the digital interface of the rotary controller and the FSM's state diagram is as follows:



The controller is soldered on a single chip and connected with GPIO. The circuit of it is shown below:



In software, we use polling for the rotary controller's input. The CPU reads in the output from this component and moves the character accordingly.

Game Logic

The game logic is the software that controls the above mentioned hardware. There are four distinct parts of the game logic: initial setup, the main loop, the screen refresh method and the game over screen.

Initial Setup

Setting up the screen for the game must be done first. This includes setting the top tile row with the score and health information, and the second row with the spikes. In addition, the initial position of the player is set to the center of the screen, and a large platform is created under the player. Lastly, a variable that determines the probability of a platform being generated on the bottom of the screen is set (see *platform generation* below).

Main Loop

Once the user moves the player with the rotary switch, the main loop of the program begins and continues until the game is over. The main loop serves three purposes: to generate the next platform, update the position of the player and update the score & health.

Platform Generation

Using the probability mentioned above, the code determines whether or not a platform will be generated on the next level that will appear. If a randomly generated integer is less than the probability, a platform is generated. If a platform is to be generated, the type of platform is randomly selected and the X position of the platform is randomly chosen. Although these parameters have been created, the platform does not appear until the row it is in is shown on the screen.

Player Position Updates

The future position of the player is determined during this stage, but the actual position the player is in is not set during this stage. The actual displayed position is changed during screen refresh.

The X position of the player is determined by capturing the state of the rotary switch. The player's future position is set to a different position if the switch has been changed. If at any time the next position is past the bounds of the screen, the position is updated to be that bound.

The Y position is determined based on the state of the player.

- If the player is free falling, the future position is updated to mimic gravity.
- If the player has landed on a platform that does not result in a bounce, the player's next position will be on top of the platform. The tile offset, pixel offset and set of current tiles are used to determine if the player is on a platform.
- It the player is bouncing, the next position will be higher than the current position.

If the Y position reaches the bottom of the screen, the player will bounce. If the player hits the spikes on top, it will stop moving until the platform under it disappears.

Score and Health Updates

The score is updated whenever the player lands on a tile. Each tile provides only one additional point. The scroll speed and platform generation probability are based on the score, so they are updated at this time as well.

The health is decreased if the player hits spikes or the bottom of the screen. It is increased whenever the player lands on the health platform.

Screen Refresh

Each time the screen is refreshed (when an interrupt occurs), the following happens:

- The score and health displays are updated.
- The player position is increased or decreased to reach the future position requested by the main loop.
- If a platform should be generated, its properties are retrieved from the main loop and the tiles for the platform are set.
- Any sand platforms that were landed on are removed if their time has expired.

Doing these things during the screen refresh period ensure that no unwanted flashing occurs.

Game Over

When the player's health reaches zero, the game is over. This results in the screen being cleared and the score of the player being displayed. In addition, the high score for the session of play is also shown. When the user is ready to play again, they can use the rotary switch to start over.

Work Distribution

- Shangru Li worked on the audio controller and rotary controller.

- Zachary Salzbank worked on the video controller and the software.

Lessons Learned

Overall, this was a fun project. The two most tedious parts of the project were making sure the display did not flash (by using the interrupt) and ensuring the audio did not sound distorted. Creating the software was not difficult. The only time we had problems with creating software was when there were underlying issues with the hardware. The tile architecture made the platform generation process much more simple than our initial plan to use sprites for each platform.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


-- this rom is likely to be changed to ram, 'cause CPU will write values to it
entity allocation is
port (Clk : in std_logic;
        addrX : in integer;
        addrY : in integer;
        data : out integer);
end allocation;
architecture imp of allocation is
type X_type is array (0 to 39) of integer;
type Y_type is array (0 to 31) of x_type ;   -- 32 * 40 blocks because there is
an additional line.
constant ROM : Y_type :=
(
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
),
(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
)
);



begin

process (Clk)
begin
if rising_edge(Clk) then
 data <= ROM(addrY)(addrX);
end if;
end process;
end imp;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity datafeed is port (
        clk, reset_n , write , read, chipselect  : in std_logic;
        address: in unsigned (5 downto 0);
        writedata : in unsigned (15 downto 0);
        readdata: out unsigned (15 downto 0);
        request : out std_logic;
        AUD_ADCLRCK  : out  std_logic;   --    Audio CODEC ADC LR Clock
    AUD_ADCDAT   : in   std_logic;   --    Audio CODEC ADC Data
    AUD_DACLRCK  : out  std_logic;   --    Audio CODEC DAC LR Clock
    AUD_DACDAT   : out  std_logic;   --    Audio CODEC DAC Data
    AUD_BCLK     : inout std_logic  --    Audio CODEC Bit-Stream Clock
);

end datafeed ;




architecture rtl of datafeed is
component de2_wm8731_audio is
port (
    clk : in std_logic;       --  Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
    reset_n : in std_logic;
    test_mode : in std_logic;       --    Audio CODEC controller test mode
        start : in std_logic;
        finished : out std_logic;
        tone : in std_logic;
    audio_request : out std_logic;  --    Audio controller request new data
    data : in unsigned(15 downto 0);
    counter_out: out unsigned (4 downto 0);
    -- Audio interface signals
    AUD_ADCLRCK  : out  std_logic;   --    Audio CODEC ADC LR Clock
    AUD_ADCDAT   : in   std_logic;   --    Audio CODEC ADC Data
    AUD_DACLRCK  : out  std_logic;   --    Audio CODEC DAC LR Clock
    AUD_DACDAT   : out  std_logic;   --    Audio CODEC DAC Data
    AUD_BCLK     : inout std_logic  --    Audio CODEC Bit-Stream Clock
  );
end  component;
signal counter : unsigned (4 downto 0);
signal test_mode : std_logic:= '0';
signal tone : std_logic;
type buffer_type is array (0 to 31) of unsigned (15 downto 0);
signal buffer1 : buffer_type;
signal audio_request: std_logic;
signal data: unsigned (15 downto 0);
signal audio_clk : std_logic;
signal start : std_logic;
signal finished : std_Logic;
--signal AUD_ADCDAT, AUD_ADCLRCK, AUD_DACLRCK, AUD_DACDAT, AUD_BCLK : std_logic;
begin
v: de2_wm8731_audio port map (
        audio_clk,
        reset_n,
        test_mode,
        start ,
        finished ,
        tone ,
        audio_request,
        data,
        counter,
    AUD_ADCLRCK,
    AUD_ADCDAT  ,   --    Audio CODEC ADC Data
    AUD_DACLRCK  ,  --    Audio CODEC DAC LR Clock
    AUD_DACDAT ,   --    Audio CODEC DAC Data
    AUD_BCLK   --    Audio CODEC Bit-Stream Clock
);

  process (clk)
begin
        if rising_edge (clk) then
                audio_clk <= not audio_clk;
        end if;
end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        request <= '0';
      else
        if counter = "11111" then
          request <= '1';
        elsif write = '1' and chipselect = '1' then
          request <= '0';  -- important to reset the irq
        end if;
      end if;
    end if;
  end process;

process (clk) begin
        if rising_edge (clk) then
                data <= buffer1(to_integer (counter));

                if chipselect = '1' and write = '1'  then
                        if  address(5) = '1' then
                                buffer1 (to_integer (address(4 downto 0))) <= wr
itedata;
                        elsif address = "000000" then
                                tone <= writedata (0);
                        elsif address = "000001" then
                                start <= writedata (0);
                        end if;
                end if;
        end if;
end process;

process (clk) begin
        if rising_edge (clk) then
                if chipselect = '1' and write = '1'  then
                        if address = "000010" then
                                test_mode <= '1';
                        end if;
                elsif   finished = '1' then
                                test_mode <= '0';
                end if;
        end if;
end process;


end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_raster is

  port (
    clk50     : in std_logic;
    reset : in std_logic;
    chipselect : in std_logic;
    write, read : in std_logic;
    address  :  in std_logic_vector(15 downto 0);
    writedata : in std_logic_vector(15 downto 0);
    readdata : out std_logic_vector(15 downto 0);
        VGA_IRQ : OUT std_logic ;
    VGA_CLK,                          -- Clock
    VGA_HS,                           -- H_SYNC
    VGA_VS,                           -- V_SYNC
    VGA_BLANK,                        -- BLANK
    VGA_SYNC : out std_logic;         -- SYNC
    VGA_R,                            -- Red[9:0]
    VGA_G,                            -- Green[9:0]
    VGA_B : out unsigned(9 downto 0) -- Blue[9:0]
    );

end de2_vga_raster;

architecture rtl of de2_vga_raster is
  -- Address Assignment:
  -- a(15:14) == 00  ->  Set Tile
  --     a(5:0)      ->  X
  --     a(10:6)     ->  Y
  -- a(15:14) == 01  ->  Set Man Property
  --     a(2:0)      ->  Property
  --       000       ->  X
  --       001       ->  Y
  --       010       ->  Direction
  -- a(15:14) == 10  ->  Settings
  --     a(2:0)      ->  Setting
  --       000       ->  Speed


component man_rom
        PORT
        (
                address         : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                clock           : IN STD_LOGIC ;
                q               : OUT STD_LOGIC_VECTOR (24 DOWNTO 0)
        );
end component;


  component man_register is
    port
    (
      clk   : in std_logic;
      reset_n : in std_logic;
      write  : in std_logic;
      property : in std_logic_vector(2 downto 0);
      value : in std_logic_vector(15 downto 0);

      x : out integer;
      y : out integer;
      dir : out integer
    );
  end component;

component tile_rom
        PORT
        (
                address         : IN STD_LOGIC_VECTOR (13 DOWNTO 0);
                clock           : IN STD_LOGIC ;
                q               : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
        );
end component;

component tile_ram_mega
        PORT
        (
                clock           : IN STD_LOGIC ;
                data            : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
                rdaddress_a             : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                rdaddress_b             : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                wraddress               : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                wren            : IN STD_LOGIC  := '1';
                qa              : OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
                qb              : OUT STD_LOGIC_VECTOR (5 DOWNTO 0)
        );
end component;



  -- 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;

  constant MAN_W    : integer := 20;
  constant MAN_H    : integer := 28;

  constant tileWidth : integer := 16;
  constant tileHeight : integer := 16;

  constant statusTileHeight : integer := 2;
  -- Signals for the video controller
  signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)
  signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)
  signal EndOfLine, EndOfField : std_logic;
  signal offset, tileOffset : integer;
  signal counter : unsigned (15 downto 0);
  signal counter_max : unsigned (15 downto 0) := to_unsigned(512, 16);
  signal vga_hblank, vga_hsync,
    vga_vblank, vga_vsync, blank : std_logic;  -- Sync. signals

  signal addrX, addrY: integer;  -- the pixel to read from each tile rom
  signal RGB: STD_LOGIC_VECTOR (15 DOWNTO 0);
  signal tileNumberX: integer; -- the horizontal index of the tile
  signal tileNumberY : integer; -- vertical one
  signal tileNumber: integer ;  -- tile name, now it is "000" to "111"

  signal tileWrite : std_logic;
  signal tileNumberA, tileNumberB : std_logic_vector(5 downto 0);

  signal romAddr, alloAddrY : integer;
  signal clk25 : std_logic := '0';

  signal vga_active : std_logic;
  signal x_pos, y_pos : unsigned(9 downto 0);

  signal write_man, man_on : std_logic;
  signal man_x, man_y, man_dir : integer;
  signal man_address_base, man_address : integer;
  signal man_data     : std_logic_vector(24 downto 0);
  signal man_pixel : integer;
begin
  vga_active <= not vga_hblank and not vga_vblank;
  x_pos <= Hcount-(HSYNC + HBACK_PORCH);
  y_pos <= Vcount-(VSYNC + VBACK_PORCH);

  process (clk50)
  begin
    if rising_edge(clk50) then
      clk25 <= not clk25;

                if chipselect = '1' and write = '1' and address(15 downto 14) =
"10" and address(2 downto 0) = "000" then
                        counter_max <= unsigned(writedata);
                else
                        counter_max <= counter_max;
                end if;
    end if;
  end process;

-- generate the IRQ when the vcount reach the bottom of the screen
 process (clk25)
 begin
        if rising_edge (clk25) then
                if reset = '1' then
                        VGA_IRQ <= '0';
                else
                        if chipselect = '1' and write = '1' and address(15 downt
o 14) = "11" then
                                VGA_IRQ <= '0';
                        else
                                if vga_vsync = '1' then
                                        VGA_IRQ <= '1' ;
                                else
                                        VGA_IRQ <= '0';
                                end if;
                        end if;
                end if;
        end if;
end process;



  write_man <= chipselect and write and '1' when (address(15 downto 14) = "01")
else '0';
  MAN_R: man_register port map (
    clk         => clk50,
    reset_n     => not reset,
    write       => write_man,
    property    => address(2 downto 0),
    value       => writedata,

    x           => man_x,
    y           => man_y,
    dir         => man_dir
  );

  man_address_base <= 0 when man_dir = 0 else
                      MAN_W*MAN_H*1 when man_dir = 1 else
                      MAN_W*MAN_H*2 when man_dir = 2 else
                      0;
  man_address <= man_address_base + man_pixel + 1;

  man_rom_inst : man_rom PORT MAP (
    address      => std_logic_vector(to_unsigned(man_address, 11)),
    clock        => clk25,
    q    => man_data
  );

  ManGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        man_on <= '0';
      elsif(vga_active = '1') then
        if (y_pos >= man_y and y_pos < man_y + MAN_H) and
           (x_pos >= man_x and x_pos < man_x + MAN_W) and
           (man_data(24) = '1') then
          man_on <= '1';
        else
          man_on <= '0';
        end if;
      else
        man_on <= '0';
      end if;
    end if;
  end process ManGen;

  ManPixelGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        man_pixel <= -1;
      elsif(vga_active = '1') then
        if(x_pos + 1 = man_x-1 and y_pos + 1 = man_y-1) then
          man_pixel <= -1;
        elsif   (y_pos + 1 >= man_y and y_pos+ 1 < man_y + MAN_H) and
               (x_pos + 1>= man_x and x_pos + 1< man_x + MAN_W) then
          man_pixel <= man_pixel + 1;
        end if;
      end if;
    end if;
  end process ManPixelGen;

tile_rom_inst : tile_rom PORT MAP (
                address  => std_logic_vector(to_unsigned(romAddr, 14)),
                clock    => not clk25,
                q        => RGB
        );
```

```
tileWrite <= chipselect and write and '1' when address(15 downto 14) = "00" else
 '0';
tile_ram_mega_inst : tile_ram_mega PORT MAP (
                clock    => clk50,
                data     => writedata(5 downto 0),
                rdaddress_a      => std_logic_vector(to_unsigned(alloAddrY, 5))
& std_logic_vector(to_unsigned(tileNumberX, 6)),
                rdaddress_b      => address(10 downto 0),
                wraddress        => address(10 downto 0),
                wren     => tileWrite,
                qa       => tileNumberA,
                qb       => tileNumberB
        );

tileNumber <= to_integer(unsigned(tileNumberA));
readdata(15 downto 11) <= std_logic_vector (to_unsigned (tileOffset, 5));
readdata (10 downto 7) <= std_logic_vector (to_unsigned (offset, 4));
readdata(5 downto 0) <= tileNumberB;


count: process (clk25)
begin
        if rising_edge (clk25) then

        --------------------------
                if reset = '1' or counter = counter_max or counter_max = 0 then
                        counter <= (others => '0');
                elsif vga_vblank = '1' then
                        counter <= counter + 1;
                end if;

                if tileNumberY>=statusTileHeight then
                        romAddr <= ((tileNumber)*256 + (addrY-1)*(tileWidth) + a
ddrX);
                else
                        romAddr <= ((tileNumber)*256 + (addrY-1-offset)*(tileWid
th) + addrX);
                end if;
                addrY <= to_integer (Vcount - VSYNC - VBACK_PORCH - tileHeight *
 tileNumberY + offset + 1); -- plus the offset

        end if;
end process count;


offsetPro: process (clk25)
begin
        if rising_edge(clk25) then
                if reset = '1' then
                        offset <= 0;
                        tileOffset <= statusTileHeight-1;
-- only when the vcount reach the bottom, do we change the offset
-- so there will never the flashing on the tile
-- the sprites flashing issue is solved by the IRQ in software level
                elsif  counter = counter_max and counter_max > 0 then
                        if offset = tileHeight - 1 then
                                offset <=  0;

                                if tileOffset = 31 then
                                        tileOffset <= statusTileHeight;
                                else
                                        tileOffset <= tileOffset + 1;
                                end if;
                        else
                                offset <= offset + 1;
                        end if;
                end if;

                if tileNumberY>=statusTileHeight then
                        if tileNumberY+tileOffset-1 > 31 then
                                alloAddrY <= tileNumberY+tileOffset-1+statusTile
Height;
                        else
                                alloAddrY <= tileNumberY+tileOffset-1;
                        end if;
                else
                        alloAddrY <= tileNumberY;
                end if;
        end if;
end process offsetPro;


tileX: process (clk25)
begin
        if falling_edge (clk25) then
                if reset = '1' then
                        tileNumberX <= 0;
                else
                        if Hcount = "0000000000" then
                                tileNumberX<= 0;
                        elsif Hcount  > (tileNumberX+1)* tileWidth +  HSYNC + HB
ACK_PORCH - 2 - 1 then
                                tileNumberX <= tileNumberX+1;
                        end if;
                end if;

                addrX <= to_integer (Hcount - HSYNC - HBACK_PORCH - tileWidth *
tileNumberX + 1);
        end if;
end process tileX;

 tileY: process (clk25)
begin
        if falling_edge (clk25) then
                if reset = '1' then
                        tileNumberY <= 0;
                else
                        if Vcount = "0000000000" then
                                tileNumberY<= 0;
                        elsif tileNumberY >= statusTileHeight and Vcount > (tile
NumberY +1)* tileHeight + VSYNC + VBACK_PORCH - offset -1 then
                                tileNumberY <= tileNumberY+1;
                        elsif Vcount > (tileNumberY +1)* tileHeight + VSYNC + VB
ACK_PORCH -1 then
                                tileNumberY <= tileNumberY+1;
                        end if;
                end if;
        end if;
end process tileY;


  HCounter : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        Hcount <= (others => '0');
      elsif 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 (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        Vcount <= (others => '0');
      elsif 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 (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' or EndOfLine = '1' then
        vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
        vga_hsync <= '0';
      end if;
    end if;
  end process HSyncGen;

  HBlankGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        vga_hblank <= '1';
      elsif Hcount = HSYNC + HBACK_PORCH then
        vga_hblank <= '0';
      elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
        vga_hblank <= '1';
      end if;
    end if;
  end process HBlankGen;

  VSyncGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        vga_vsync <= '1';
      elsif EndOfLine ='1' then
        if EndOfField = '1' then
          vga_vsync <= '1';
        elsif Vcount = VSYNC - 1 then
```

```
          vga_vsync <= '0';
        end if;
      end if;
    end if;
  end process VSyncGen;

  VBlankGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        vga_vblank <= '1';
      elsif EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;

--process (clk25)
--  begin
--    if rising_edge(clk25) then
--      if reset = '1' then
--        blank <= '0';
--      elsif EndOfLine = '1' then
--        if Vcount = VSYNC + VBACK_PORCH + VACTIVE then
--         blank <= '0';
--        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
--          blank <= '1';
--        end if;
--      end if;
--    end if;
--  end process;
  -- Registered video signals going to the video DAC

  VideoOut: process (clk25, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk25'event and clk25 = '1' then

          if vga_active = '1' then
                if man_on = '1' then
                        VGA_R <= unsigned(man_data(23 downto 16) & "00");
                        VGA_G <= unsigned(man_data(15 downto 8) & "00");
                        VGA_B <= unsigned(man_data(7 downto 0) & "00");
                else
                        VGA_R <= unsigned (RGB(15 downto 11) & "00000");
                        VGA_G <= unsigned(RGB(10 downto 5) & "0000");
                        VGA_B <= unsigned(RGB(4 downto 0) & "00000");
                end if;
      else
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end process VideoOut;
  VGA_CLK <= clk25;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_wm8731_audio is
port (
    clk : in std_logic;       --  Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
    reset_n : in std_logic;
    test_mode : in std_logic;       --    Audio CODEC controller test mode
        start : in std_logic;
        finished : out std_logic;
        tone : in std_logic;
    audio_request : out std_logic;  --    Audio controller request new data
    data : in unsigned(15 downto 0);
    counter_out: out unsigned (4 downto 0);
    -- Audio interface signals
    AUD_ADCLRCK  : out  std_logic;   --    Audio CODEC ADC LR Clock
    AUD_ADCDAT   : in   std_logic;   --    Audio CODEC ADC Data
    AUD_DACLRCK  : out  std_logic;   --    Audio CODEC DAC LR Clock
    AUD_DACDAT   : out  std_logic;   --    Audio CODEC DAC Data
    AUD_BCLK     : inout std_logic  --    Audio CODEC Bit-Stream Clock
  );
end  de2_wm8731_audio;

architecture rtl of de2_wm8731_audio is

    signal lrck : std_logic;
    signal bclk : std_logic;
    signal xck  : std_logic;

    signal lrck_divider : unsigned(11 downto 0);
    signal bclk_divider : unsigned(3 downto 0);

    signal set_bclk : std_logic;
    signal set_lrck : std_logic;
    signal clr_bclk : std_logic;
    signal lrck_lat : std_logic;

    signal shift_out : unsigned(15 downto 0);

    signal sin_out     : unsigned(15 downto 0);
    signal sin_counter : unsigned(11 downto 0);
        signal backcounter : unsigned (4 downto 0);
        signal testCounter : unsigned (25 downto 0);

        signal endOfTone : unsigned (11 downto 0);
    signal sin_out1, sin_out0 : unsigned(15 downto 0);
        signal merged: unsigned (15 downto 0);
        signal switch : std_logic ;
        signal thresh: unsigned (11 downto 0);
component musicRom is port (Clk: in std_logic;
addr : in unsigned(10 downto 0);
data : out unsigned(15 downto 0));
end component;

component musicRom1 is port (Clk: in std_logic;
addr : in unsigned(10 downto 0);
data : out unsigned(15 downto 0));
end component;


begin
  counter_out<=backcounter;
```

```
    -- LRCK divider
    -- Audio chip main clock is 18.432MHz / Sample rate 48KHz
    -- Divider is 18.432 MHz / 48KHz = 192 (X"C0")
    -- Left justify mode set by I2C controller

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        lrck_divider <= (others => '0');
      elsif lrck_divider = thresh then--"100001000000"  then        -- "C0" minu
s 1
        lrck_divider <= "000000000000";
      else
        lrck_divider <= lrck_divider + 1;
      end if;
    end if;
  end process;

thresh <= "100001000000"  ;

  process (clk)
        begin
                if rising_edge (clk) then
                        if sin_counter = "000000000000" then
                                finished <= '0';
                        elsif sin_counter = endOfTone then
                                finished <= '1';
                        end if;
                end if;
        end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        bclk_divider <= (others => '0');
      elsif bclk_divider = X"B" or set_lrck = '1'  then
        bclk_divider <= X"0";
      else
        bclk_divider <= bclk_divider + 1;
      end if;
    end if;
  end process;


   set_lrck <= '1' when lrck_divider = "100001000000" else '0';
  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        lrck <= '0';
      elsif set_lrck = '1' then
        lrck <= not lrck;
      end if;
    end if;
  end process;

  -- BCLK divider
  set_bclk <= '1' when bclk_divider(3 downto 0) = "0101" else '0';
  clr_bclk <= '1' when bclk_divider(3 downto 0) = "1011" else '0';

  process (clk)
```

```
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      bclk <= '0';
    elsif set_lrck = '1' or clr_bclk = '1' then
      bclk <= '0';
    elsif set_bclk = '1' then
      bclk <= '1';
    end if;
  end if;
end process;



-- Audio data shift output
process (clk)
begin
  if rising_edge(clk) then
              if switch = '0' and test_mode = '1' then
                      merged <= sin_out;
              else
                      merged <= data;
              end if;

    if reset_n = '0' then
      shift_out <= (others => '0');
    elsif set_lrck = '1' then
      if test_mode = '1' then
                      shift_out <= merged;
      else
        shift_out <= data;
      end if;
    elsif clr_bclk = '1' then
      shift_out <= shift_out (14 downto 0) & '0';
    end if;
  end if;
end process;

  -- Audio outputs

  AUD_ADCLRCK  <= lrck;
  AUD_DACLRCK  <= lrck;
  AUD_DACDAT   <= shift_out(15);
  AUD_BCLK     <= bclk;

  -- Self test with Sin wave

  process(clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' or start = '0'then
          sin_counter <= (others => '0');
      elsif lrck_lat = '1' and lrck = '0'  then
          sin_counter <= sin_counter + 1;
                      switch <= not switch ;
        end if;
      end if;
  end process;

  process(clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
```

```
            backcounter<= (others => '0');
        elsif lrck_lat = '1' and lrck = '0'  then
            backcounter <= backcounter + 1;

        end if;
      end if;
    end process;

    process(clk)
    begin
      if rising_edge(clk) then
        lrck_lat <= lrck;
      end if;
    end process;

    process (clk)
    begin
      if rising_edge(clk) then
        if lrck_lat = '1' and lrck = '0' then
          audio_request <= '1';
        else
          audio_request <= '0';
        end if;
      end if;
    end process;

v1: musicRom  port map (
                clk ,
                sin_counter (11 downto 1),
                sin_out0);
v2: musicRom1  port map (
                clk ,
                sin_counter (11 downto 1),
                sin_out1);

endOfTone <= "110110000000" when tone = '0' else
                     "101011100100" when tone = '1' else
                     "110110000000" ;

sin_out <= sin_out0 when tone = '0' else
                   sin_out1 when tone = '1' else
                   sin_out0;

end architecture;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity inputController is

port(
clk              : in   std_logic;
reset_n : in std_logic;
chipSelect: in std_logic;
read: in std_logic;
address : in std_logic;
read_data: out unsigned (15 downto 0);
GPIOinput        : in   std_logic_vector (1 downto 0)
);

end entity;

architecture rtl of inputController is
        signal readdata : unsigned (15 downto 0);
        signal output : unsigned (1 downto 0);
        component rotaryController is port(
                clk              : in   std_logic;
                input    : in   std_logic_vector (1 downto 0);
                reset_n  : in   std_logic;
                output   : out  unsigned (1 downto 0)
        );

end component;
begin
        v1: rotaryController port map (
        clk,
        GPIOinput,
        reset_n,
        output
        );

        process (clk)
        begin
                if rising_edge (clk) then
                        if reset_n = '0' then
                                read_data <= (others => '0');
                        else
                                if chipSelect = '1' then
                                        if address = '0' then
                                                if read = '1' then
                                                        read_data <= UNSIGNED ("
00000000000000"  & OUTPUT); -- readdata;
                                                end if;
                                        end if;
                                end if;
                        end if;
                end if;
        end process;
end rtl;
```

```
--
-- DE2 top-level module that includes the simple VGA raster generator
--
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lab3_vga is

  port (
    -- Clocks

    CLOCK_27,                                         -- 27 MHz
    CLOCK_50,                                         -- 50 MHz
    EXT_CLOCK : in std_logic;                         -- External Clock

    -- Buttons and switches

    KEY : in std_logic_vector(3 downto 0);           -- Push buttons
    SW : in std_logic_vector(17 downto 0);           -- DPDT switches

    -- LED displays

    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
       : out std_logic_vector(6 downto 0);
    LEDG : out std_logic_vector(8 downto 0);         -- Green LEDs
    LEDR : out std_logic_vector(17 downto 0);        -- Red LEDs

    -- RS-232 interface

    UART_TXD : out std_logic;                         -- UART transmitter
    UART_RXD : in std_logic;                          -- UART receiver

    -- IRDA interface

--    IRDA_TXD : out std_logic;                         -- IRDA Transmitter
    IRDA_RXD : in std_logic;                          -- IRDA Receiver

    -- SDRAM

    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                        -- Low-byte Data Mask
    DRAM_UDQM,                                        -- High-byte Data Mask
    DRAM_WE_N,                                        -- Write Enable
    DRAM_CAS_N,                                       -- Column Address Strobe
    DRAM_RAS_N,                                       -- Row Address Strobe
    DRAM_CS_N,                                        -- Chip Select
    DRAM_BA_0,                                        -- Bank Address 0
    DRAM_BA_1,                                        -- Bank Address 0
    DRAM_CLK,                                         -- Clock
    DRAM_CKE : out std_logic;                         -- Clock Enable

    -- FLASH

    FL_DQ : inout std_logic_vector(7 downto 0);       -- Data bus
    FL_ADDR : out std_logic_vector(21 downto 0);  -- Address bus
```

```
FL_WE_N,                                            -- Write Enable
FL_RST_N,                                           -- Reset
FL_OE_N,                                            -- Output Enable
FL_CE_N : out std_logic;                            -- Chip Enable

-- SRAM

SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
SRAM_UB_N,                                      -- High-byte Data Mask
SRAM_LB_N,                                      -- Low-byte Data Mask
SRAM_WE_N,                                      -- Write Enable
SRAM_CE_N,                                      -- Chip Enable
SRAM_OE_N : out std_logic;                      -- Output Enable

-- USB controller

OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
OTG_ADDR : out std_logic_vector(1 downto 0);    -- Address
OTG_CS_N,                                        -- Chip Select
OTG_RD_N,                                        -- Write
OTG_WR_N,                                        -- Read
OTG_RST_N,                                       -- Reset
OTG_FSPEED,                      -- USB Full Speed, 0 = Enable, Z = Disable
OTG_LSPEED : out std_logic;      -- USB Low Speed, 0 = Enable, Z = Disable
OTG_INT0,                                        -- Interrupt 0
OTG_INT1,                                        -- Interrupt 1
OTG_DREQ0,                                       -- DMA Request 0
OTG_DREQ1 : in std_logic;                        -- DMA Request 1
OTG_DACK0_N,                                     -- DMA Acknowledge 0
OTG_DACK1_N : out std_logic;                     -- DMA Acknowledge 1

-- 16 X 2 LCD Module

LCD_ON,                      -- Power ON/OFF
LCD_BLON,                    -- Back Light ON/OFF
LCD_RW,                      -- Read/Write Select, 0 = Write, 1 = Read
LCD_EN,                      -- Enable
LCD_RS : out std_logic;      -- Command/Data Select, 0 = Command, 1 = Data
LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits

-- SD card interface

SD_DAT,                      -- SD Card Data
SD_DAT3,                     -- SD Card Data 3
SD_CMD : inout std_logic;    -- SD Card Command Signal
SD_CLK : out std_logic;      -- SD Card Clock

-- USB JTAG link

TDI,                         -- CPLD -> FPGA (data in)
TCK,                         -- CPLD -> FPGA (clk)
TCS : in std_logic;          -- CPLD -> FPGA (CS)
TDO : out std_logic;         -- FPGA -> CPLD (data out)

-- I2C bus

I2C_SDAT : inout std_logic; -- I2C Data
I2C_SCLK : out std_logic;   -- I2C Clock

-- PS/2 port

PS2_DAT,                     -- Data
```

```
    PS2_CLK : in std_logic;     -- Clock

    -- VGA output

    VGA_CLK,                                             -- Clock
    VGA_HS,                                              -- H_SYNC
    VGA_VS,                                              -- V_SYNC
    VGA_BLANK,                                           -- BLANK
    VGA_SYNC : out std_logic;                            -- SYNC
    VGA_R,                                               -- Red[9:0]
    VGA_G,                                               -- Green[9:0]
    VGA_B : out unsigned(9 downto 0);                   -- Blue[9:0]

    --  Ethernet Interface

    ENET_DATA : inout std_logic_vector(15 downto 0);    -- DATA bus 16Bits
    ENET_CMD,          -- Command/Data Select, 0 = Command, 1 = Data
    ENET_CS_N,                                           -- Chip Select
    ENET_WR_N,                                           -- Write
    ENET_RD_N,                                           -- Read
    ENET_RST_N,                                          -- Reset
    ENET_CLK : out std_logic;                            -- Clock 25 MHz
    ENET_INT : in std_logic;                             -- Interrupt

    -- Audio CODEC

    AUD_ADCLRCK : inout std_logic;                       -- ADC LR Clock
    AUD_ADCDAT : in std_logic;                           -- ADC Data
    AUD_DACLRCK : inout std_logic;                       -- DAC LR Clock
    AUD_DACDAT : out std_logic;                          -- DAC Data
    AUD_BCLK : inout std_logic;                          -- Bit-Stream Clock
    AUD_XCK : out std_logic;                             -- Chip Clock

    -- Video Decoder

    TD_DATA : in std_logic_vector(7 downto 0);  -- Data bus 8 bits
    TD_HS,                                      -- H_SYNC
    TD_VS : in std_logic;                       -- V_SYNC
    TD_RESET : out std_logic;                   -- Reset

    -- General-purpose I/O

    GPIO_0,                                      -- GPIO Connection 0
    GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
    );

end lab3_vga;

architecture datapath of lab3_vga is
  signal clk25 : std_logic := '0';
  signal input: std_logic_vector (1 downto 0);
  signal proc_R, proc_G, proc_B : STD_LOGIC_VECTOR (9 DOWNTO 0);
  component de2_i2c_av_config is
  port (
    iCLK : in std_logic;
    iRST_N : in std_logic;
    I2C_SCLK : out std_logic;
    I2C_SDAT : inout std_logic
  );
  end component;


begin
```

```vhdl
i2c : de2_i2c_av_config port map (
  iCLK     => CLOCK_50,
  iRST_n   => '1',
  I2C_SCLK => I2C_SCLK,
  I2C_SDAT => I2C_SDAT
);
process (CLOCK_50)
begin
  if rising_edge(CLOCK_50) then
    clk25 <= not clk25;
  end if;
end process;

proc : entity work.watch_out
  port map(
    SRAM_ADDR_from_the_sram => SRAM_ADDR,
    SRAM_CE_N_from_the_sram => SRAM_CE_N,
    SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
    SRAM_LB_N_from_the_sram => SRAM_LB_N,
    SRAM_OE_N_from_the_sram => SRAM_OE_N,
    SRAM_UB_N_from_the_sram => SRAM_UB_N,
    SRAM_WE_N_from_the_sram => SRAM_WE_N,

    AUD_ADCDAT_to_the_game_sound =>AUD_ADCDAT,
    AUD_ADCLRCK_from_the_game_sound =>AUD_ADCLRCK,
    AUD_BCLK_to_and_from_the_game_sound =>AUD_BCLK,
    AUD_DACDAT_from_the_game_sound =>AUD_DACDAT,
    AUD_DACLRCK_from_the_game_sound =>AUD_DACLRCK,

    VGA_BLANK_from_the_vga => VGA_BLANK,
    VGA_B_from_the_vga => proc_B,
    VGA_CLK_from_the_vga => VGA_CLK,
    VGA_G_from_the_vga => proc_G,
    VGA_HS_from_the_vga => VGA_HS,
    VGA_R_from_the_vga => proc_R,
    VGA_SYNC_from_the_vga => VGA_SYNC,
    VGA_VS_from_the_vga => VGA_VS,
    clk => CLOCK_50,
        GPIOinput_to_the_rotary => input,
    reset_n => KEY(0)
  );
  input <= (NOT GPIO_1(1)) & (NOT GPIO_1(3));
      VGA_R <= unsigned(proc_R);
      VGA_G <= unsigned(proc_G);
      VGA_B <= unsigned(proc_B);

HEX7     <= "0001001"; -- Leftmost
HEX6     <= "0000110";
HEX5     <= "1000111";
HEX4     <= "1000111";
HEX3     <= "1000000";
HEX2     <= (others => '1');
HEX1     <= (others => '1');
HEX0     <= ("00000"&input);--;(others => '1');          -- Rightmost
LEDG     <= (others => '1');
LEDR     <= (others => '0');
LCD_ON   <= '1';
LCD_BLON <= '1';
LCD_RW <= '1';
LCD_EN <= '0';
LCD_RS <= '0';

SD_DAT3 <= '1';
```

```
  SD_CMD <= '1';
  SD_CLK <= '1';

  UART_TXD <= '0';
  DRAM_ADDR <= (others => '0');
  DRAM_LDQM <= '0';
  DRAM_UDQM <= '0';
  DRAM_WE_N <= '1';
  DRAM_CAS_N <= '1';
  DRAM_RAS_N <= '1';
  DRAM_CS_N <= '1';
  DRAM_BA_0 <= '0';
  DRAM_BA_1 <= '0';
  DRAM_CLK <= '0';
  DRAM_CKE <= '0';
  FL_ADDR <= (others => '0');
  FL_WE_N <= '1';
  FL_RST_N <= '0';
  FL_OE_N <= '1';
  FL_CE_N <= '1';
  OTG_ADDR <= (others => '0');
  OTG_CS_N <= '1';
  OTG_RD_N <= '1';
  OTG_RD_N <= '1';
  OTG_WR_N <= '1';
  OTG_RST_N <= '1';
  OTG_FSPEED <= '1';
  OTG_LSPEED <= '1';
  OTG_DACK0_N <= '1';
  OTG_DACK1_N <= '1';

  TDO <= '0';

  ENET_CMD <= '0';
  ENET_CS_N <= '1';
  ENET_WR_N <= '1';
  ENET_RD_N <= '1';
  ENET_RST_N <= '1';
  ENET_CLK <= '0';

  TD_RESET <= '0';

  --I2C_SCLK <= '1';

 -- AUD_DACDAT <= '1';
  AUD_XCK <= clk25;

  -- Set all bidirectional ports to tri-state
  DRAM_DQ     <= (others => 'Z');
  FL_DQ       <= (others => 'Z');
  SRAM_DQ     <= (others => 'Z');
  OTG_DATA    <= (others => 'Z');
  LCD_DATA    <= (others => 'Z');
  SD_DAT      <= 'Z';
  I2C_SDAT    <= 'Z';
  ENET_DATA   <= (others => 'Z');
  AUD_ADCLRCK <= 'Z';
  AUD_DACLRCK <= 'Z';
  AUD_BCLK    <= 'Z';
  GPIO_0      <= (others => 'Z');
 -- GPIO_1      <= (others => 'Z');

end datapath;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity man_register is
  port
  (
    clk    : in std_logic;
    reset_n : in std_logic;
    write  : in std_logic;
    property : in std_logic_vector(2 downto 0);
    value : in std_logic_vector(15 downto 0);

    x : out integer;
    y : out integer;
    dir : out integer
  );
end man_register;

architecture rtl of man_register is
    signal x_val : unsigned(9 downto 0) := to_unsigned(230, 10);
    signal y_val : unsigned(9 downto 0) := to_unsigned(0, 10);
    signal dir_val : unsigned(1 downto 0) := to_unsigned(0, 2);
begin
  x <= to_integer(x_val);
  y <= to_integer(y_val);
  dir <= to_integer(dir_val);

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        x_val <= (others => '0');
        y_val <= (others => '0');
        dir_val <= (others => '0');
      else
        if write = '1' then
          case property is
             when "000" => x_val       <= unsigned(value(9 downto 0));
             when "001" => y_val       <= unsigned(value(9 downto 0));
             when "010" => dir_val    <= unsigned(value(1 downto 0));
             when others =>
              x_val <= x_val;
          end case;
        end if;
      end if;
    end if;
  end process;
end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity musicControl is port (
clk              : in   std_logic;
reset_n : in std_logic;
chipSelect: in std_logic;
write: in std_logic;
address : in std_logic;
writedata: in unsigned (15 downto 0);
AUD_ADCLRCK  : out std_logic;       --    Audio CODEC ADC LR Clock
AUD_ADCDAT   : in  std_logic;       --    Audio CODEC ADC Data
AUD_DACLRCK  : out std_logic;       --    Audio CODEC DAC LR Clock
AUD_DACDAT   : out std_logic;       --    Audio CODEC DAC Data
AUD_BCLK     : inout std_logic      --    Audio CODEC Bit-Stream Clock

);
end musicControl;

architecture rtl of musicControl is
signal tone: unsigned (15 downto 0);
signal counter : unsigned (24 downto 0);
signal tone_out, tone_out0,tone_out1,tone_out2,tone_out3 : unsigned (3 downto 0)
;
signal counter_tone : unsigned (1 downto 0);
 signal audio_clock : unsigned(1 downto 0) := "00";
signal audio_request : std_logic;
signal Stop : std_logic ;
signal stopCount : unsigned (2 downto 0);
  component de2_wm8731_audio is
   port (
    clk : in std_logic;               --    Audio CODEC Chip Clock AUD_XCK
    reset_n : in std_logic;
    test_mode : in std_logic;         --    Audio CODEC controller test mode
    audio_request : out std_logic;    --    Audio controller request new data
    data : in std_logic_vector(15 downto 0);
        tone: in unsigned (3 downto 0);
        stop: in std_logic;
    -- Audio interface signals
    AUD_ADCLRCK  : out std_logic;       --    Audio CODEC ADC LR Clock
    AUD_ADCDAT   : in  std_logic;       --    Audio CODEC ADC Data
    AUD_DACLRCK  : out std_logic;       --    Audio CODEC DAC LR Clock
    AUD_DACDAT   : out std_logic;       --    Audio CODEC DAC Data
    AUD_BCLK     : inout std_logic      --    Audio CODEC Bit-Stream Clock
  );
  end component;

begin

V1: de2_wm8731_audio port map (
    clk => audio_clock(1),
    reset_n => '1',
    test_mode => '0',                 -- Output a sine wave
    audio_request => audio_request,
    data => "0000000000000000",
        tone => tone_out,
        stop => stop,
    -- Audio interface signals
    AUD_ADCLRCK  => AUD_ADCLRCK,
    AUD_ADCDAT   => AUD_ADCDAT,
    AUD_DACLRCK  => AUD_DACLRCK,
    AUD_DACDAT   => AUD_DACDAT,
    AUD_BCLK     => AUD_BCLK
  );

process (CLk)
  begin
    if rising_edge(CLk) then
      audio_clock <= audio_clock + "1";
    end if;
  end process;

process (clk)
begin
if rising_edge (clk) then
        if reset_n = '0' then
                tone <= (others => '0');
        else
                if chipSelect = '1' then
                        if address = '0' then
                                if write = '1' then
                                        tone <= writedata;
                                end if;
                        end if;
                end if;
        end if;
end if;
end process;


process (clk)
begin
if rising_edge (clk) then
        if reset_n = '0' then
                counter<= (others => '0');
        else
                counter <= counter +1;
        end if;
end if;
end process;

process (clk)
begin
if rising_edge (clk) then
        if reset_n = '0' then
                stop <= '0';
        else
                if (write = '1' and chipSelect = '1')then
                        stop <= '0';
                elsif (stopCount = "111") then
                        stop <= '1';
                end if;
        end if;
end if;
end process;

process (counter (24))
begin
if rising_edge (counter (24)) then
        if reset_n = '0' then
                counter_tone <= (others => '0');
        else
                if (counter_tone = "11") then
                        stopCount <= stopCount +1;
                end if;
                counter_tone <= counter_tone +1;
        end if;
end if;
end process;

with tone select tone_out <=
tone_out1 when x"0000",
tone_out2 when x"0001",
tone_out3 when x"0002",
tone_out0 when others;

with counter_tone select tone_out0 <=
    "0000" when "00",
    "0001" when "01",
    "0010" when "10",
    "0011" when others;

with counter_tone select tone_out1 <=
    "0001" when "00",
    "0001" when "01",
    "0010" when "10",
    "0001" when others;
with counter_tone select tone_out2 <=
    "0010" when "00",
    "0011" when "01",
    "0110" when "10",
    "0011" when others;
with counter_tone select tone_out3 <=
    "0000" when "00",
    "0001" when "01",
    "0000" when "10",
    "0011" when others;

end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity musicRom1 is
port (Clk: in std_logic;
addr : in unsigned(10 downto 0);
data : out unsigned(15 downto 0));
end musicRom1;

architecture imp of musicRom1 is
type rom_type is array (0 to 1394) of unsigned(15 downto 0);
constant ROM : rom_type :=(
    X"061c",  X"0a1f",  X"b417",  X"4370",  X"e7cf",  X"c2b6",  X"3d6a",  X"f1c3",  X"c
d64",  X"42b7",  X"df69",  X"f516",  X"f1c2",  X"46ec",  X"ecfd",  X"b10d",  X"8000",
X"c06c",  X"e483",  X"5dec",  X"bc6c",  X"ec76",  X"4c0a",  X"bece",  X"f270",  X"3a8e"
,  X"bffb",  X"0b6f",  X"371e",  X"840c",  X"4d75",  X"106b",  X"a738",  X"3e13",  X"0f
21",  X"b407",  X"4205",  X"f99b",  X"dbe0",  X"fc62",  X"34e9",  X"f874",  X"aca9",  X
"7bdf",  X"cfce",  X"d725",  X"5f4e",  X"cc20",  X"dd1c",  X"5109",  X"c601",  X"ec4a",
  X"3df8",  X"c92f",  X"f992",  X"4be0",  X"856c",  X"31b4",  X"32fb",  X"91da",  X"341
2",  X"2410",  X"a59c",  X"3b30",  X"0c0b",  X"cbd9",  X"046a",  X"3082",  X"02d4",
X"a2b2",  X"6a4e",  X"e0b1",  X"d13f",  X"5766",  X"d55f",  X"d765",  X"4bd9",  X"cc91"
,  X"eaa1",  X"35a8",  X"d4fa",  X"f09d",  X"4fe6",  X"968e",  X"0f1d",  X"54a4",  X"8d
0b",  X"20fc",  X"3a51",  X"a271",  X"2a24",  X"2629",  X"bd58",  X"039c",  X"271a",
X"12e4",  X"a75c",  X"58a5",  X"f0d4",  X"cab0",  X"4da2",  X"e0c5",  X"d315",  X"432b"
,  X"d535",  X"e952",  X"2f25",  X"dc8e",  X"ecd1",  X"46bf",  X"b1b2",  X"eafc",  X"6b
2d",  X"97eb",  X"072f",  X"4e75",  X"ab59",  X"0cdd",  X"3ddd",  X"bbf0",  X"03bc",
X"1e73",  X"1ee9",  X"ab4a",  X"464e",  X"06d7",  X"c07e",  X"45f6",  X"f1fb",  X"ca50"
,  X"3d74",  X"e2ce",  X"dde3",  X"307c",  X"e18d",  X"e850",  X"3c14",  X"d3bc",  X"c
1b2",  X"71ee",  X"bb19",  X"db64",  X"58f3",  X"bf15",  X"ec73",  X"4f0f",  X"c4bb",
X"f8ea",  X"1a5e",  X"2b13",  X"ae59",  X"316d",  X"20fe",  X"b476",  X"3f7e",  X"04d5"
,  X"bf6e",  X"3a9c",  X"f2fa",  X"d101",  X"3380",  X"e7b4",  X"dec4",  X"3273",  X"f7
70",  X"abdf",  X"62fc",  X"e15b",  X"bcd7",  X"593d",  X"dc40",  X"cf6d",  X"536f",  X
"d2f6",  X"ea6b",  X"1414",  X"3440",  X"bbdd",  X"0f47",  X"400c",  X"aef5",  X"2a24",
  X"1ed3",  X"b836",  X"2abf",  X"11a3",  X"c28c",  X"2a8e",  X"fe93",  X"d365",  X"262
1",  X"10b9",  X"a38d",  X"4599",  X"1150",  X"a094",  X"4870",  X"ff64",  X"b79f",  X
"4bb1",  X"edce",  X"d894",  X"0d77",  X"3b98",  X"d2a5",  X"e63c",  X"5b84",  X"b557"
,  X"0774",  X"3f1e",  X"b944",  X"09f9",  X"2e3c",  X"b96a",  X"19d8",  X"17ce",  X"c
a33",  X"1548",  X"26d6",  X"aa06",  X"193c",  X"3e0c",  X"9abb",  X"29bb",  X"227c",
 X"ac45",  X"32bd",  X"0ece",  X"cce8",  X"04bb",  X"361f",  X"f59d",  X"bd6c",  X"6362
",  X"d28c",  X"e3d2",  X"4bb0",  X"c93f",  X"ec2d",  X"3ed1",  X"c4df",  X"ffdf",  X"2
a42",  X"ce34",  X"0346",  X"34c9",  X"bd30",  X"e904",  X"6221",  X"a4a4",  X"032a",
 X"47ae",  X"aec2",  X"0f7d",  X"36d0",  X"c21a",  X"fb5b",  X"2e3d",  X"077b",  X"b810
",  X"5414",  X"ed9c",  X"d200",  X"48ba",  X"dd17",  X"df62",  X"3de3",  X"d082",  X
"f127",  X"2ce4",  X"d22c",  X"ff4f",  X"2d61",  X"dce4",  X"c116",  X"70d9",  X"be9b",
  X"d63a",  X"5d12",  X"bd6a",  X"ef0d",  X"4d06",  X"bfbe",  X"fa65",  X"282f",  X"047
1",  X"cc65",  X"3c0a",  X"f5eb",  X"dd20",  X"36f7",  X"e00b",  X"ef13",  X"281d",  X
"d6b8",  X"ffbf",  X"1c92",  X"d704",  X"1070",  X"0a37",  X"0773",  X"b11a",  X"5372",
  X"ef7c",  X"b941",  X"5741",  X"cb8d",  X"e6e4",  X"47b5",  X"c482",  X"0711",  X"1a
90",  X"f998",  X"f38d",  X"162d",  X"fb6f",  X"f5be",  X"15ae",  X"f122",  X"fe71",  X
"0887",  X"f075",  X"00e9",  X"04ba",  X"ee6e",  X"0e92",  X"ef83",  X"2a0b",  X"ae39",
  X"2a57",  X"1fd4",  X"9b0f",  X"5abf",  X"cffc",  X"e5cf",  X"473e",  X"be25",  X"1a40
",  X"137a",  X"e447",  X"1f6f",  X"f648",  X"fee7",  X"113e",  X"ec54",  X"0ec7",  X"f
9ff",  X"f2a1",  X"1126",  X"e5fc",  X"0f79",  X"f589",  X"0346",  X"f732",  X"2ca9",  X
"b564",  X"0d5c",  X"3ba5",  X"7f52",  X"7bbf",  X"9fa0",  X"1ffb",  X"1a48",  X"c532",
  X"4a35",  X"cd06",  X"1861",  X"0ba3",  X"e8ae",  X"2254",  X"dd09",  X"ff60",  X"f90
1",  X"f4b4",  X"0cdb",  X"ed1a",  X"0beb",  X"f78d",  X"029f",  X"04e2",  X"eec5",  X"
3405",  X"c060",  X"0110",  X"2f74",  X"9b6d",  X"6813",  X"8e09",  X"66ab",  X"ab2d",
  X"4db7",  X"c70e",  X"2df0",  X"e852",  X"088d",  X"0698",  X"f4f6",  X"0e08",  X"e8c
e",  X"125a",  X"e996",  X"0e44",  X"f05d",  X"0420",  X"fd38",  X"ff45",  X"070a",  X"
edeb",  X"2753",  X"dd31",  X"0077",  X"0a68",  X"e021",  X"0e12",  X"e872",  X"0b68",
  X"fe52",  X"0cad",  X"fcc3",  X"0638",  X"03a5",  X"02b2",  X"0655",  X"fe03",  X"047
f",  X"f6d9",  X"05c4",  X"f59f",  X"ff2e",  X"fdae",  X"f8c7",  X"02b1",  X"fa3d",  X"0
6df",  X"f6b3",  X"0f88",  X"f59a",  X"fdab",  X"0382",  X"f318",  X"0688",  X"f2c4",  X
```

```
"019d",  X"fcde",  X"02af",  X"0267",  X"fe92",  X"077b",  X"0494",  X"033b",  X"06da",
  X"ffe7",  X"ffd0",  X"0040",  X"fc3f",  X"fe74",  X"fc6e",  X"fe75",  X"fefa",  X"fea9",
  X"ff56",  X"fdbc",  X"0119",  X"0258",  X"f6a3",  X"0423",  X"f5ec",  X"008f",  X"f95b
",  X"fd24",  X"fe26",  X"ff83",  X"0264",  X"fae2",  X"035a",  X"0716",  X"02c4",  X"0
b4b",  X"00b6",  X"03a5",  X"fe7c",  X"feb5",  X"fd5f",  X"fcc5",  X"fde1",  X"fe0e",  X
"ff4d",  X"fdc0",  X"ffae",  X"fec0",  X"0425",  X"fb7d",  X"00fe",  X"f8a2",  X"fdba",
X"fe16",  X"fcd0",  X"ff51",  X"ff64",  X"01fa",  X"fbb7",  X"feec",  X"03b7",  X"0203",
 X"0878",  X"027f",  X"02c2",  X"0163",  X"00e0",  X"ff52",  X"fdc8",  X"fe58",  X"fd51"
,  X"fdb0",  X"fd09",  X"ff59",  X"fcb5",  X"0339",  X"fa3b",  X"0036",  X"faf4",  X"fc6
3",  X"febe",  X"fe7b",  X"fdc9",  X"004d",  X"00ba",  X"fd52",  X"fd3a",  X"05f1",  X"f
eeb",  X"0b94",  X"008c",  X"0470",  X"ffac",  X"fff1",  X"fe7b",  X"fdfb",  X"fd8e",  X
"fcc9",  X"fedc",  X"fdf9",  X"ffd8",  X"fe4d",  X"0302",  X"fc40",  X"fd80",  X"0001",
X"fe3d",  X"008c",  X"ff9a",  X"019e",  X"0230",  X"0176",  X"0121",  X"fdae",  X"0cfe",
  X"ff4d",  X"0aa7",  X"02b7",  X"05b3",  X"02c4",  X"031d",  X"002c",  X"00ac",  X"ff1
a",  X"fe88",  X"ffc6",  X"fddf",  X"0013",  X"fec1",  X"0188",  X"fe35",  X"feda",  X"0
2f1",  X"fe04",  X"01b1",  X"00d6",  X"00d5",  X"025b",  X"001f",  X"033c",  X"f974",
X"0e05",  X"fb6a",  X"09db",  X"044f",  X"056a",  X"04e1",  X"03f5",  X"018d",  X"006d"
,  X"ffff",  X"fe5d",  X"fe07",  X"0047",  X"fe91",  X"002f",  X"00b9",  X"013f",  X"ff9b
",  X"038f",  X"ffef",  X"00f4",  X"0134",  X"fedf",  X"04cf",  X"fab8",  X"0855",  X"f2
04",  X"1307",  X"fd15",  X"0812",  X"09b6",  X"04a0",  X"06d5",  X"0394",  X"02bf",
X"0022",  X"ff18",  X"ff3a",  X"fed7",  X"ffde",  X"ff1b",  X"ffe6",  X"0151",  X"ffd2",
 X"0182",  X"ffba",  X"0054",  X"01e3",  X"016f",  X"0111",  X"01e6",  X"fe35",  X"046d
",  X"f651",  X"0362",  X"2166",  X"d911",  X"38a6",  X"d6c2",  X"3535",  X"d727",  X
"299e",  X"d78b",  X"2378",  X"d9d6",  X"2074",  X"df06",  X"1a3a",  X"e5a5",  X"11da"
,  X"f2bd",  X"0570",  X"060e",  X"f68a",  X"0fad",  X"f4d2",  X"0ad5",  X"fc18",  X"08
0a",  X"fa6b",  X"0ac6",  X"e3dd",  X"4a48",  X"c22e",  X"3463",  X"0060",  X"ec86",
X"39ee",  X"bb0b",  X"46ee",  X"c03b",  X"2939",  X"ec2f",  X"fa43",  X"1709",  X"e0df"
,  X"234c",  X"e9c4",  X"067b",  X"05c1",  X"f7d3",  X"08be",  X"045e",  X"fd07",  X"0
91b",  X"fe87",  X"fcaa",  X"0bda",  X"dc3f",  X"39a9",  X"f30f",  X"d913",  X"60c9",
X"b021",  X"38a6",  X"0452",  X"d3aa",  X"3f09",  X"cc95",  X"126d",  X"0f91",  X"d765"
,  X"293c",  X"e83b",  X"fbb1",  X"19ca",  X"e2a9",  X"0fde",  X"0c29",  X"e84c",  X"1a
2b",  X"f76e",  X"fad7",  X"1002",  X"f52a",  X"fae3",  X"0d03",  X"11f8",  X"d7aa",  X
"3007",  X"f139",  X"f810",  X"25b3",  X"e034",  X"117b",  X"0a52",  X"e97c",  X"1aa7",
  X"f6e5",  X"f94f",  X"161c",  X"ee53",  X"0f7e",  X"0712",  X"da36",  X"36b5",  X"e2e
9",  X"f85c",  X"26bf",  X"d9d9",  X"1470",  X"0eb8",  X"e72a",  X"14c6",  X"0ba0",  X
"e6b6",  X"1d0e",  X"fb1f",  X"f578",  X"19b6",  X"eaf6",  X"0b63",  X"0546",  X"f131",
  X"1210",  X"f648",  X"01ad",  X"06f7",  X"f864",  X"0b15",  X"0a0f",  X"d045",  X"32c
a",  X"eeb9",  X"e7dd",  X"2fe0",  X"cf5b",  X"201d",  X"057e",  X"e9df",  X"2224",  X"
f4d2",  X"0745",  X"1375",  X"f00e",  X"16be",  X"f7e8",  X"ff2a",  X"0ca9",  X"f06f",
X"102f",  X"f8e4",  X"0127",  X"09ca",  X"f676",  X"0ea9",  X"f684",  X"1990",  X"d832"
,  X"1eae",  X"026f",  X"e255",  X"2a14",  X"d1fd",  X"266d",  X"f37f",  X"034c",  X"16
f7",  X"e931",  X"244e",  X"ece8",  X"1008",  X"023e",  X"f953",  X"11f4",  X"ef3f",  X
"14ec",  X"ef99",  X"1178",  X"f44c",  X"0902",  X"feb7",  X"0534",  X"fd3e",  X"1410",
  X"e796",  X"111f",  X"0223",  X"eee6",  X"1815",  X"e1ce",  X"20de",  X"e6dd",  X"1da
6",  X"f31b",  X"0c1c",  X"ff3f",  X"02ce",  X"0192",  X"00ae",  X"00a2",  X"fea1",  X"
0228",  X"ff84",  X"003b",  X"006d",  X"fecb",  X"02e5",  X"fe02",  X"0559",  X"f9c3",
 X"0e30",  X"f161",  X"0e2e",  X"fe28",  X"fbad",  X"05fd",  X"f928",  X"096e",  X"f9a9"
,  X"0b88",  X"fa3d",  X"0e41",  X"fae7",  X"0963",  X"fea1",  X"0639",  X"ff59",  X"01
d8",  X"0064",  X"0147",  X"00f8",  X"00ce",  X"0057",  X"0181",  X"febe",  X"019b",
X"fda0",  X"03e3",  X"fd2a",  X"0459",  X"02e7",  X"fed5",  X"057b",  X"fded",  X"0683"
,  X"fd39",  X"096c",  X"f708",  X"0e1f",  X"f847",  X"0a5f",  X"ff93",  X"088b",  X"017
6",  X"0641",  X"ff48",  X"03ef",  X"000f",  X"00e1",  X"fe8c",  X"016e",  X"fed7",  X"
0349",  X"004f",  X"02e8",  X"0042",  X"014f",  X"0477",  X"fe6d",  X"04e8",  X"ff15",
 X"050c",  X"fe09",  X"074c",  X"f92a",  X"0e03",  X"fea4",  X"0475",  X"073d",  X"01c1
",  X"08bf",  X"ff84",  X"06f1",  X"fe91",  X"04a9",  X"fe3d",  X"0357",  X"fda9",  X"0
303",  X"0030",  X"0451",  X"ff5b",  X"0409",  X"ff3f",  X"028d",  X"ff85",  X"029a",
X"ff64",  X"02ed",  X"fedf",  X"0394",  X"fd02",  X"067d",  X"0d48",  X"ed5f",  X"1ff6",
  X"eac0",  X"1d1b",  X"edef",  X"15b2",  X"f325",  X"0c6c",  X"fb9c",  X"02cc",  X"049
4",  X"fc0f",  X"0acf",  X"fa53",  X"0a8b",  X"fabb",  X"0657",  X"01ac",  X"feee",  X"0
823",  X"fc67",  X"057d",  X"feed",  X"00a4",  X"0123",  X"f901",  X"2055",  X"e2e3",
X"1d85",  X"fda2",  X"fbde",  X"1716",  X"e793",  X"1c0c",  X"ece3",  X"0d22",  X"00ad"
,  X"f963",  X"0dcd",  X"f3e1",  X"0c4a",  X"fd46",  X"01cd",  X"02b3",  X"03ab",  X"fe
ea",  X"03c8",  X"0352",  X"fe1b",  X"06a4",  X"fd1e",  X"03d3",  X"f8e8",  X"19b7",  X
```

```
"f75d",  X"fd99",  X"1a2c",  X"ea0c",  X"174f",  X"fdf4",  X"f7e5",  X"11b4",  X"f469",
 X"0743",  X"05ba",  X"f8af",  X"0b55",  X"fcf4",  X"0160",  X"090b",  X"fc2a",  X"0417
",  X"0357",  X"fee2",  X"046c",  X"0030",  X"00b1",  X"0107",  X"014e",  X"fd73",  X"
09f4",  X"02df",  X"f544",  X"159e",  X"f7c9",  X"022f",  X"0bdd",  X"f5f1",  X"08e8",
X"02f8",  X"f961",  X"092f",  X"fd5c",  X"ff45",  X"0659",  X"fd2c",  X"02dc",  X"0573",
  X"f845",  X"0483",  X"0be3",  X"f498",  X"0a0c",  X"0358",  X"f8ef",  X"09ea",  X"002
5",  X"ffaf",  X"0ad9",  X"fd1e",  X"0431",  X"0707",  X"fcfc",  X"065d",  X"006b",  X"
0023",  X"0569",  X"feb4",  X"01b1",  X"008b",  X"0024",  X"015c",  X"00a9",  X"ff4d",
 X"04e8",  X"fb57",  X"fd20",  X"132c",  X"f2c4",  X"0785",  X"045f",  X"fa99",  X"0b73
",  X"ff85",  X"0204",  X"0060",  X"00d2",  X"ffc4",  X"001b",  X"fff4",  X"0003",  X"ff
ff",  X"ffff",  X"0003",  X"fff5",  X"0014",  X"ffdf",  X"002e",  X"ffbf",  X"005b",  X"ff
6c",  X"0184",  X"0280",  X"f801",  X"11c3",  X"f630",  X"091b",  X"00d1",  X"ff3e",  X
"088d",  X"fe0f",  X"0923",  X"fd0e",  X"09a0",  X"fcb8",  X"07ca",  X"fedd",  X"0349",
 X"0159",  X"0086",  X"042e",  X"fe42",  X"0313",  X"ff01",  X"0264",  X"ff81",  X"022e
",  X"ff97",  X"0240",  X"012f",  X"fbff",  X"0c73",  X"f82f",  X"0934",  X"fbd2",  X"05
c5",  X"ffae",  X"0625",  X"0255",  X"0168",  X"0449",  X"0139",  X"0437",  X"010c",
X"049f",  X"ffb8",  X"01c4",  X"0004",  X"008c",  X"0123",  X"00af",  X"010e",  X"00d0"
,  X"0146",  X"007e",  X"015a",  X"0082",  X"ffc6",  X"0541",  X"fff7",  X"0359",  X"ff5
d",  X"02df",  X"0148",  X"0303",  X"002b",  X"01e3",  X"004d",  X"0228",  X"0083",
X"0159",  X"0042",  X"0095",  X"007b",  X"0071",  X"ffc7",  X"001e",  X"ffed",  X"0008"
,  X"ffff",  X"fff9",  X"000e",  X"ffe7",  X"0028",  X"ffb3",  X"0196",  X"006a",  X"0161
",  X"00a9",  X"016b",  X"0048",  X"00ad",  X"0029",  X"015b",  X"00d7",  X"032b",  X
"0169",  X"030e",  X"00c1",  X"010b",  X"0107",  X"00da",  X"0130",  X"00ba",  X"0177"
,  X"fff3",  X"0001",  X"fffc",  X"0002",  X"fffe",  X"0002",  X"fffe",  X"0002",  X"fffc"
,  X"0006",  X"fff5",  X"0011",  X"ffe5",  X"002a",  X"ffb6",  X"01f9",  X"006d",  X"02f
e",  X"0113",  X"047e",  X"00f7",  X"01c0",  X"00f8",  X"015f",  X"0027",  X"00c6",  X
"ffac",  X"002f",  X"ffe0",  X"0013",  X"fff5",  X"0006",  X"fffd",  X"0000",  X"0001",
X"fffa",  X"000b",  X"ffea",  X"0027",  X"ffbf",  X"006a",  X"ff47",  X"0209",  X"ffb1",
 X"0385",  X"010f",  X"03b3",  X"0242",  X"00ae",  X"00e0",  X"00f9",  X"00e1",  X"006
5",  X"0067",  X"ffca",  X"0017",  X"0020",  X"0017",  X"fff4",  X"0006",  X"fffc",  X"0
003",  X"fffa",  X"000b",  X"ffed",  X"0021",  X"ffca",  X"0056",  X"ff6e",  X"0154",  X
"007b",  X"01fd",  X"0440",  X"ff84",  X"0283",  X"ff9a",  X"0218",  X"ff95",  X"0212",
 X"ffa8",  X"01d4",  X"0005",  X"013b",  X"00cb",  X"0073",  X"0197",  X"ff82",  X"0045
",  X"ffd7",  X"0017",  X"fff7",  X"ffff",  X"000b",  X"ffe9",  X"0026",  X"ffba",  X"00c
5",  X"0172",  X"ffaa",  X"0269",  X"ffa4",  X"01cb",  X"0021",  X"00ff",  X"0128",  X"
ffa8",  X"011d",  X"ffcd",  X"001c",  X"ffec",  X"000a",  X"fffa",  X"0003",  X"ffff",  X"
0000",  X"0000",  X"ffff",  X"0000",  X"0000",  X"fffe",  X"0004",  X"fff9",  X"000b",
X"fff0",  X"01ee",  X"ffb1",  X"01bd",  X"002c",  X"0098",  X"0186",  X"ff9a",  X"0037",
  X"ffde",  X"0015",  X"fff3",  X"0008",  X"fffb",  X"0002",  X"ffff",  X"0001",  X"ffff",
  X"0002",  X"fffb",  X"0011",  X"01d8",  X"ffbe",  X"00c4",  X"0058",  X"ffe0",  X"0011
",  X"fff5",  X"0007",  X"fffc",  X"0003",  X"ffff",  X"0000",  X"ffff",  X"0000",  X"000
0",  X"ffff",  X"0000",  X"ffff",  X"0000",  X"ffff",  X"ffff",  X"ffff",  X"ffff",  X"ffff",
  X"0000",  X"ffff",  X"ffff",  X"ffff",  X"ffff",  X"ffff",  X"ffff",  X"ffff",  X"ffff",  X"f
fff",  X"0000",  X"ffff",  X"0000",  X"ffff",  X"0000",  X"0000",  X"ffff",  X"ffff",  X"00
00",  X"0000",  X"0000",  X"ffff",  X"ffff",  X"0000",  X"0000",  X"ffff",  X"0000",  X"0
000",  X"ffff",  X"ffff",  X"0000",  X"0000",  X"ffff",  X"ffff"

);
begin
process (Clk)
begin
if rising_edge(Clk) then
data <= ROM(TO_INTEGER(addr));
end if;
end process;
end imp;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity musicRom is
port (Clk: in std_logic;
addr : in unsigned(10 downto 0);
data : out unsigned(15 downto 0));
end musicRom;

architecture imp of musicRom is
type rom_type is array (0 to 1728) of unsigned(15 downto 0);
constant ROM : rom_type :=(
    X"0001",  X"ffff",  X"0001",  X"fffd",  X"000c",  X"0004",  X"ffc5",  X"003a",  X"ffc
5",  X"0085",  X"feda",  X"0150",  X"0016",  X"fea7",  X"012c",  X"0027",  X"fd99",  X
"022a",  X"00d1",  X"fdac",  X"01a0",  X"00bf",  X"fd8c",  X"01ef",  X"ff3a",  X"ff35",
 X"fd07",  X"0b0f",  X"f7f3",  X"fbcd",  X"0931",  X"fd75",  X"f963",  X"069a",  X"fe3f"
,  X"ffad",  X"036c",  X"fbfd",  X"00cd",  X"00c9",  X"01c5",  X"f442",  X"0bc0",  X"09
ce",  X"e995",  X"0a04",  X"0842",  X"f27a",  X"02f8",  X"04ce",  X"fdc6",  X"ffec",  X
"0181",  X"ff37",  X"ff87",  X"036d",  X"ef5a",  X"ffe0",  X"0c0a",  X"e304",  X"0fba",
 X"0825",  X"ee8e",  X"03ca",  X"0889",  X"f9b2",  X"00b6",  X"03fe",  X"fe6c",  X"fcf9"
,  X"0605",  X"e6a1",  X"2335",  X"0022",  X"dadd",  X"1c81",  X"0834",  X"e463",  X"
09f8",  X"0a38",  X"f80f",  X"ffa1",  X"06e9",  X"faf8",  X"ffb5",  X"00d5",  X"e359",
X"3fcd",  X"d903",  X"ef37",  X"2067",  X"ff19",  X"dfb7",  X"188c",  X"0051",  X"fabc",
  X"fd40",  X"0da6",  X"f452",  X"0423",  X"f45f",  X"f6af",  X"3cef",  X"c201",  X"0973
",  X"18d8",  X"f923",  X"e0a6",  X"21e3",  X"f67e",  X"0029",  X"f941",  X"129a",  X"
edf9",  X"0bc9",  X"e47d",  X"104d",  X"2dfa",  X"b91c",  X"1ad2",  X"12ad",  X"f2da",
 X"e496",  X"279b",  X"f01b",  X"01f3",  X"f934",  X"1465",  X"e9af",  X"0f02",  X"dd5b
",  X"21a6",  X"1f4f",  X"b8e1",  X"22b5",  X"0f3d",  X"efb1",  X"e6fd",  X"2a45",  X"e
b57",  X"0431",  X"fae2",  X"11f0",  X"ec23",  X"0cad",  X"e039",  X"1447",  X"3696",
X"abb7",  X"1da6",  X"162e",  X"f8ab",  X"d666",  X"2dbc",  X"f2a3",  X"fe73",  X"00c2"
,  X"0c07",  X"ee95",  X"0698",  X"f331",  X"05b9",  X"230a",  X"c4eb",  X"2e65",  X"e
d49",  X"ffd0",  X"d833",  X"3a0e",  X"ba93",  X"2cd6",  X"19bf",  X"c7f3",  X"0d1b",
X"1ed1",  X"ed23",  X"e4fe",  X"2c84",  X"e93b",  X"f3b5",  X"22ce",  X"ee4a",  X"f388"
,  X"0e23",  X"012a",  X"edd7",  X"1421",  X"0a72",  X"d54d",  X"2bf8",  X"eef1",  X"0
18d",  X"ee74",  X"21cb",  X"ddee",  X"edb4",  X"4d3d",  X"e0c8",  X"cb52",  X"3d41",
 X"07a6",  X"c75d",  X"14c4",  X"25b3",  X"c48e",  X"ff08",  X"5dca",  X"a564",  X"04a5
",  X"2eca",  X"f645",  X"c2fc",  X"473f",  X"0690",  X"ae84",  X"45b6",  X"02f3",  X"d
b89",  X"fb5c",  X"3393",  X"c4ef",  X"eeac",  X"6d83",  X"ac41",  X"e576",  X"641c",
X"d257",  X"d34c",  X"31a1",  X"02cc",  X"bd26",  X"2acf",  X"3f0c",  X"b9c9",  X"da8f"
,  X"6e20",  X"b522",  X"f406",  X"2532",  X"0be0",  X"d3ba",  X"0e19",  X"2eb0",  X"d
588",  X"eb91",  X"35a6",  X"e5f2",  X"dca2",  X"013e",  X"4d0b",  X"dd57",  X"ba64",
 X"7baf",  X"bff8",  X"e10b",  X"26f5",  X"0c57",  X"bc82",  X"24ac",  X"127a",  X"22a8
",  X"b7c7",  X"0e9c",  X"3908",  X"b808",  X"1c93",  X"f514",  X"0d30",  X"fb10",  X"
f0df",  X"2c9d",  X"f737",  X"c84a",  X"47c7",  X"dcf4",  X"f8a8",  X"d6c5",  X"3bbd",
X"3079",  X"85a8",  X"5203",  X"279c",  X"a06c",  X"1924",  X"3586",  X"b85b",  X"04da
",  X"1d2e",  X"0aaf",  X"06bf",  X"e805",  X"06fa",  X"0264",  X"00a6",  X"f00b",  X"0
767",  X"db1a",  X"33a9",  X"212f",  X"9c29",  X"4e11",  X"0c06",  X"ccbe",  X"fc8d",
X"4549",  X"a630",  X"1c40",  X"032b",  X"e97a",  X"2576",  X"35e0",  X"b202",  X"febe
",  X"570d",  X"8832",  X"3277",  X"022c",  X"fbd9",  X"13de",  X"d1a4",  X"3bf6",  X"
cbdf",  X"2dc1",  X"e21b",  X"ffa0",  X"1c78",  X"da5e",  X"1896",  X"cf18",  X"1a16",
 X"0ea5",  X"3587",  X"d444",  X"d30e",  X"6546",  X"9db5",  X"0c53",  X"29f6",  X"d58
f",  X"fb82",  X"49c5",  X"9638",  X"589e",  X"cc44",  X"3c00",  X"be37",  X"2f87",  X
"0649",  X"c47a",  X"36ee",  X"bf32",  X"0a46",  X"1a1d",  X"35e9",  X"eca5",  X"acfc",
  X"739a",  X"b7cb",  X"e7e8",  X"3ff9",  X"eb0b",  X"b571",  X"79cb",  X"b84d",  X"f24
e",  X"26fc",  X"15b1",  X"dd44",  X"e575",  X"63a9",  X"8396",  X"47ca",  X"ee6b",  X
"d4df",  X"1425",  X"3740",  X"f166",  X"0d95",  X"bd15",  X"4fa8",  X"c7ee",  X"041c",
  X"1784",  X"ec2e",  X"ce9f",  X"414a",  X"194b",  X"a574",  X"4385",  X"1761",  X"e5
39",  X"c87e",  X"76b9",  X"9b6f",  X"09bb",  X"3112",  X"c7ca",  X"e1a7",  X"50ad",
X"ef7c",  X"e28b",  X"47d5",  X"b213",  X"1828",  X"22c1",  X"dcbf",  X"f752",  X"2524"
,  X"cf80",  X"dde9",  X"650c",  X"ca4d",  X"dcf2",  X"405a",  X"2746",  X"8c05",  X"5f
d3",  X"ff01",  X"abbb",  X"3446",  X"1eea",  X"a381",  X"1b0a",  X"4edf",  X"ade3",  X
"fad6",  X"5beb",  X"d03d",  X"c08d",  X"8000",  X"a425",  X"f95b",  X"34e1",  X"fe4a",
  X"a36a",  X"531a",  X"0c46",  X"b7ac",  X"1f7c",  X"381f",  X"d5c1",  X"cc11",  X"765
```

```
a",  X"a1b1",  X"04e4",  X"2c18",  X"0017",  X"bdab",  X"23fd",  X"2f9f",  X"b7fb",  X"
0e28",  X"2a3b",  X"fb93",  X"bcc2",  X"4e01",  X"f18e",  X"cc11",  X"37c1",  X"f97c",
 X"ec29",  X"e6f5",  X"2869",  X"00dc",  X"d97d",  X"1df4",  X"113b",  X"f9d8",  X"cea1
",  X"46f0",  X"d6e5",  X"f751",  X"1ded",  X"fc0f",  X"ef2d",  X"f10e",  X"1e2d",  X"fc
f1",  X"e04f",  X"1b24",  X"09c6",  X"0925",  X"caa9",  X"3800",  X"f1d2",  X"e166",  X
"21e9",  X"fca7",  X"f8c1",  X"e8e1",  X"1612",  X"034c",  X"f128",  X"056a",  X"0839",
  X"0f26",  X"e536",  X"fed7",  X"29c6",  X"c9ba",  X"1c13",  X"ff3c",  X"0642",  X"e9f0
",  X"0a04",  X"f1e9",  X"1e55",  X"e238",  X"05cc",  X"0da5",  X"0fa6",  X"d99f",  X"0
c29",  X"2474",  X"c39b",  X"2d84",  X"edba",  X"1143",  X"e40e",  X"1bf9",  X"d887",
 X"2092",  X"fe90",  X"e066",  X"2418",  X"f397",  X"0e40",  X"d47f",  X"3d8d",  X"cfe0
",  X"0ee5",  X"0bc0",  X"fcee",  X"fad9",  X"ffd0",  X"0b86",  X"d155",  X"4039",  X"d
0ab",  X"08d5",  X"0a88",  X"0552",  X"f72e",  X"e7b2",  X"3ff6",  X"bd0c",  X"257c",
X"fc88",  X"ff53",  X"f669",  X"0c8c",  X"ff33",  X"d9f4",  X"39eb",  X"ddb1",  X"f8d5",
  X"17c5",  X"f529",  X"1053",  X"d81e",  X"27fb",  X"f586",  X"e891",  X"20fe",  X"f32
9",  X"02e4",  X"f93c",  X"0fc7",  X"e6eb",  X"fa64",  X"29d0",  X"df68",  X"f9f5",  X"
1d73",  X"ec28",  X"1441",  X"dcc2",  X"1f33",  X"f9f9",  X"ea51",  X"1d2a",  X"f147",
 X"096a",  X"f38b",  X"10b8",  X"f3ae",  X"e83a",  X"201b",  X"0177",  X"e13e",  X"16cc
",  X"0134",  X"01e4",  X"07a1",  X"def1",  X"2d89",  X"deb7",  X"061e",  X"0e84",  X"
f6ae",  X"01cc",  X"f874",  X"1359",  X"debb",  X"00d2",  X"1ac7",  X"edc6",  X"0089",
 X"0362",  X"fcf9",  X"1e4d",  X"ecc4",  X"e080",  X"3e6d",  X"d4f6",  X"fdb6",  X"19d7
",  X"f3ec",  X"f859",  X"086f",  X"07b6",  X"e11d",  X"05bc",  X"12d5",  X"f8c0",  X"f
4d1",  X"0f0c",  X"f73c",  X"0b01",  X"1ee6",  X"b6f6",  X"37a7",  X"0028",  X"d3ba",
X"2904",  X"f48e",  X"fe40",  X"fb1d",  X"1443",  X"e553",  X"f291",  X"2333",  X"e999"
,  X"07ca",  X"f94e",  X"0691",  X"ffa1",  X"0daf",  X"118d",  X"bea3",  X"3f0f",  X"eed
0",  X"e064",  X"2ad6",  X"e981",  X"03c6",  X"fe58",  X"0e81",  X"e560",  X"fd04",  X
"1802",  X"eb9d",  X"0c26",  X"f9f7",  X"022b",  X"001a",  X"fd71",  X"2412",  X"d83c",
  X"fcc5",  X"2e3d",  X"cdec",  X"1497",  X"0875",  X"edb3",  X"085a",  X"02be",  X"02a
7",  X"e0cc",  X"1a81",  X"fe08",  X"f76a",  X"0a24",  X"efe8",  X"0daa",  X"fdcd",  X"f
9ee",  X"2354",  X"e072",  X"f35a",  X"2eec",  X"d51a",  X"0b19",  X"1114",  X"ed7e",
X"04fd",  X"0134",  X"02c5",  X"e8ce",  X"1208",  X"ff02",  X"f9af",  X"0ba1",  X"f326",
  X"021d",  X"06f9",  X"fb80",  X"0995",  X"10de",  X"d3aa",  X"1f4f",  X"0706",  X"db4
2",  X"1daa",  X"f8d3",  X"fc1d",  X"0254",  X"0102",  X"fa45",  X"f9e2",  X"0d00",  X"
f80c",  X"06b2",  X"fd3f",  X"fcdd",  X"049b",  X"fa02",  X"06cf",  X"004c",  X"1098",
X"e297",  X"062c",  X"16b7",  X"dd8d",  X"1420",  X"0280",  X"f687",  X"062f",  X"fc18"
,  X"00d8",  X"faa3",  X"06e4",  X"fd95",  X"00f1",  X"054a",  X"f78a",  X"feda",  X"07
d1",  X"fb7e",  X"fe3d",  X"0c97",  X"fe20",  X"ea7c",  X"1460",  X"fcae",  X"f119",  X
"1366",  X"f770",  X"ff7c",  X"025b",  X"fe58",  X"0142",  X"f9c8",  X"078e",  X"fda5",
 X"0071",  X"038f",  X"fb5c",  X"ff8e",  X"0136",  X"00c3",  X"fc52",  X"0603",  X"08b8
",  X"ea41",  X"0804",  X"09d5",  X"ef6e",  X"0b25",  X"0003",  X"fbc2",  X"043a",  X"
fe2a",  X"ff9f",  X"fc5a",  X"0397",  X"0119",  X"fe52",  X"02a3",  X"0108",  X"f95b",
X"0215",  X"03e7",  X"f947",  X"04cd",  X"051d",  X"fd52",  X"f478",  X"0c9d",  X"fd12"
,  X"f8dd",  X"09e1",  X"fbe7",  X"0032",  X"01f1",  X"fe8b",  X"fe7d",  X"ff26",  X"019
0",  X"01a0",  X"fe0f",  X"040e",  X"fdde",  X"f8e2",  X"0942",  X"fc05",  X"fe8b",  X"
0263",  X"031f",  X"ff85",  X"f4d1",  X"0d22",  X"fb1a",  X"fbbd",  X"07d0",  X"fbb4",
X"0139",  X"ff95",  X"01b7",  X"fcfb",  X"0063",  X"ff09",  X"0280",  X"016c",  X"fcc3",
  X"0499",  X"f87a",  X"0535",  X"ff81",  X"fcbf",  X"047e",  X"fc6a",  X"0761",  X"f712
",  X"00ac",  X"07ee",  X"f5b3",  X"0641",  X"ffc7",  X"fd60",  X"0269",  X"0033",  X"f
fc9",  X"fdaf",  X"010c",  X"feb7",  X"034b",  X"fe9b",  X"ff87",  X"0385",  X"f9b6",  X
"03eb",  X"fff4",  X"fc47",  X"043f",  X"fdcb",  X"0519",  X"f9bd",  X"ff75",  X"073e",
 X"f743",  X"0444",  X"0157",  X"fd87",  X"0121",  X"00a6",  X"00bb",  X"fc95",  X"00bc
",  X"00fa",  X"01f1",  X"fdc1",  X"ff88",  X"03d1",  X"fb78",  X"0065",  X"033f",  X"fc
36",  X"019d",  X"ff9c",  X"03a4",  X"fd01",  X"fb8e",  X"08af",  X"f9d7",  X"0036",  X
"03e6",  X"fd2d",  X"0014",  X"0110",  X"00eb",  X"fcec",  X"011b",  X"0013",  X"012b",
  X"ff68",  X"ffcc",  X"0031",  X"0010",  X"ff5e",  X"005a",  X"ffe0",  X"ff72",  X"00b6"
,  X"0074",  X"0269",  X"f9e8",  X"038e",  X"0121",  X"fc66",  X"02fb",  X"ff4f",  X"ffe
c",  X"ff6a",  X"029f",  X"fd15",  X"005b",  X"fe6c",  X"035e",  X"00af",  X"fbb0",  X"0
3c0",  X"fe7f",  X"ffd6",  X"001c",  X"ffe7",  X"ffb1",  X"009f",  X"ff59",  X"03de",  X"
fb96",  X"fdc2",  X"06ba",  X"fb80",  X"0017",  X"01eb",  X"ff93",  X"feaa",  X"0226",
X"ff65",  X"fdae",  X"0088",  X"0254",  X"ffa4",  X"fd87",  X"0320",  X"fe42",  X"0008",
  X"00c3",  X"fee5",  X"fffc",  X"ffc9",  X"012d",  X"0328",  X"f9b3",  X"01cc",  X"03c3
",  X"faac",  X"0320",  X"ff4f",  X"0006",  X"ffe6",  X"018e",  X"ff5e",  X"fce8",  X"02e
f",  X"fe58",  X"0279",  X"fec1",  X"fefa",  X"021b",  X"fe59",  X"0090",  X"ff45",  X"0
0c5",  X"fdd7",  X"03a1",  X"0190",  X"f81b",  X"06d1",  X"fe23",  X"fe12",  X"0237",
X"003b",   X"fe72",   X"001c",   X"02ce",   X"fcfd",   X"00d4",   X"febf",   X"0216",   X"ff4f",
  X"006f",   X"febb",   X"0095",   X"0191",   X"fce3",   X"0292",   X"ff39",   X"ff44",   X"ff65
",   X"0476",   X"fbcd",   X"fe3a",   X"0675",   X"fa51",   X"01ea",   X"00ac",   X"ffaa",   X"fe
3d",   X"0268",   X"0049",   X"fd7c",   X"017a",   X"ff63",   X"012d",   X"ff6c",   X"ff14",   X
"0162",   X"ff52",   X"ffec",   X"004b",   X"ffeb",   X"0035",   X"fe3f",   X"0250",   X"0198",
 X"f9cb",   X"0561",   X"ff90",   X"fd17",   X"0235",   X"0055",   X"fed0",   X"ff5a",   X"0299"
,   X"fe40",   X"ffb1",   X"007a",   X"ffe7",   X"00fa",   X"feb7",   X"001c",   X"00a6",   X"ffbf
",   X"0027",   X"ff26",   X"0141",   X"ffc3",   X"fde1",   X"02c8",   X"013b",   X"f9c4",   X"0
5ff",   X"feb3",   X"fe06",   X"015f",   X"00ac",   X"fec3",   X"ffa1",   X"022a",   X"fe42",   X"
0041",   X"002d",   X"ff6d",   X"0145",   X"ffa5",   X"feab",   X"0123",   X"0070",   X"ff15",
X"ffda",   X"0133",   X"ff2f",   X"ff72",   X"fff5",   X"0367",   X"fb63",   X"0096",   X"0426",
 X"fba9",   X"00df",   X"019a",   X"ff73",   X"fddb",   X"02c7",   X"ff62",   X"ff42",   X"ff55",
  X"01e2",   X"ff34",   X"ff4e",   X"00b8",   X"0006",   X"000c",   X"ff53",   X"006b",   X"006
9",   X"ffe2",   X"fec3",   X"0097",   X"0352",   X"fa85",   X"01be",   X"0333",   X"fe25",   X"
faf2",   X"088f",   X"fc88",   X"fb97",   X"06bb",   X"fd30",   X"fea8",   X"0195",   X"003b",
X"feb9",   X"0190",   X"fe9c",   X"005e",   X"0112",   X"fe87",   X"0045",   X"00ba",   X"ffee",
  X"feef",   X"0136",   X"ff8e",   X"ffc1",   X"007b",   X"ff52",   X"00bc",   X"ffce",   X"ff5f",
  X"0104",   X"ff5b",   X"002c",   X"ffbf",   X"006e",   X"ffc2",   X"ffe0",   X"005c",   X"ffb6"
,   X"0024",   X"ffe2",   X"0020",   X"ffe6",   X"0027",   X"ff86",   X"00f2",   X"fefd",   X"005
4",   X"010c",   X"fdbf",   X"020a",   X"ffae",   X"fe57",   X"0275",   X"fe84",   X"ffaf",   X"0
163",   X"fef4",   X"ffda",   X"0120",   X"feba",   X"00c1",   X"ffe0",   X"ffe1",   X"0031",   X
"ff7b",   X"00df",   X"ff34",   X"0027",   X"00cf",   X"fe96",   X"0138",   X"ff7c",   X"ffea",
X"0046",   X"ffde",   X"ffcc",   X"0098",   X"ff3d",   X"00a9",   X"ff80",   X"0075",   X"ff79",
 X"0067",   X"001e",   X"ff6e",   X"0038",   X"011b",   X"fd3f",   X"03ba",   X"fca0",   X"01d3
",   X"0003",   X"fef6",   X"00ef",   X"ffb8",   X"ffdb",   X"0012",   X"0028",   X"ffd6",   X"00
04",   X"ffff",   X"003d",   X"ff79",   X"00a2",   X"ff7c",   X"004b",   X"ffe6",   X"fffc",   X"0
022",   X"ffb2",   X"008f",   X"ff31",   X"00c9",   X"ffb0",   X"ff89",   X"011a",   X"fedd",   X
"007a",   X"007c",   X"feda",   X"0126",   X"ff77",   X"ffb0",   X"00ed",   X"ff03",   X"0088",
 X"001b",   X"ff83",   X"005a",   X"0034",   X"ff3b",   X"00f7",   X"ff4d",   X"003d",   X"000e"
,   X"0000",   X"ffb1",   X"0096",   X"ff5a",   X"0082",   X"ffb3",   X"001a",   X"001a",   X"ffa
e",   X"007c",   X"ff6d",   X"0098",   X"ff7c",   X"0041",   X"0030",   X"ff5e",   X"00d9",   X"
ff42",   X"0069",   X"ffff",   X"ffa4",   X"0091",   X"ff7d",   X"0046",   X"ffef",   X"000b",   X
"ffe1",   X"001c",   X"0012",   X"ffaf",   X"0075",   X"ff93",   X"0042",   X"0000",   X"ffa5",
 X"00ca",   X"fed4",   X"0148",   X"ff18",   X"0011",   X"00e0",   X"fe9a",   X"0128",   X"ffc4"
,   X"ff1f",   X"01a7",   X"fe3a",   X"014f",   X"ff67",   X"fffd",   X"005c",   X"ff85",   X"0080
",   X"ff87",   X"005e",   X"ffd6",   X"ffec",   X"004c",   X"ff88",   X"00a4",   X"ff2d",   X"00f
0",   X"ff2a",   X"0068",   X"004b",   X"fef8",   X"0174",   X"feb4",   X"0098",   X"0048",   X"
ff28",   X"00cd",   X"ffbc",   X"ffaf",   X"0084",   X"ffd1",   X"ff90",   X"00e6",   X"ff2b",   X
"0039",   X"009f",   X"fec6",   X"013b",   X"ff68",   X"ffa0",   X"0139",   X"fe74",   X"0135",
 X"ff9c",   X"ff80",   X"0111",   X"feed",   X"0097",   X"0013",   X"ff6d",   X"00ad",   X"ff97"
,   X"ffff",   X"0050",   X"ff9d",   X"0042",   X"fff5",   X"ffe3",   X"0026",   X"fff9",   X"ffd0"
,   X"0065",   X"ff83",   X"006f",   X"ffb8",   X"001c",   X"0008",   X"ffde",   X"002c",   X"ffd
a",   X"0013",   X"0006",   X"ffdd",   X"003b",   X"ffc5",   X"0019",   X"001a",   X"ffc7",   X"
0023",   X"0022",   X"ff93",   X"008b",   X"ff9d",   X"0004",   X"0061",   X"ff6c",   X"007a",
 X"ffd4",   X"ffe3",   X"0038",   X"ffdf",   X"fffd",   X"000e",   X"000e",   X"ffc0",   X"005f",
  X"ffb3",   X"000a",   X"0043",   X"ff97",   X"004d",   X"ffff",   X"ffb7",   X"0061",   X"ffc7"
,   X"ffed",   X"0055",   X"ffa0",   X"002e",   X"0020",   X"ffa6",   X"005f",   X"ffca",   X"0002
",   X"0018",   X"fff0",   X"fff4",   X"0024",   X"ffde",   X"0007",   X"001a",   X"ffd2",   X"00
29",   X"ffef",   X"fff7",   X"001a",   X"ffe2",   X"001b",   X"ffea",   X"0011",   X"fff4",   X"00
02",   X"000a",   X"ffea",   X"0019",   X"fff1",   X"fffe",   X"0013",   X"ffe7",   X"0013",   X"f
ffd",   X"fff4",   X"0014",   X"ffef",   X"0004",   X"0009",   X"fff0",   X"000c",   X"fffc",   X"f
ff5",   X"0008",   X"ffec",   X"fffa",   X"fff2",   X"ffe9",   X"ffe8",   X"ffe9",   X"ffd1",   X"ffe
2",   X"ffca",   X"ffc7",   X"ffca",   X"ffaf",   X"ffba",   X"ffa6",   X"ffa0",   X"ff9f",   X"ff89"
,   X"ff8d",   X"ff7d",   X"ff77",   X"ff6e",   X"ff67",   X"ff5a",   X"ff56",   X"ff4f",   X"ff3f",
  X"ff46",   X"ff2e",   X"ff34",   X"ff29",   X"ff1e",   X"ff24",   X"ff10",   X"ff1a",   X"ff09",
 X"ff0e",   X"ff09",   X"ff00",   X"ff0c",   X"fef9",   X"ff07",   X"fefe",   X"feff",   X"ff03",
X"fef9",   X"ff02",   X"fef2",   X"fefb",   X"feec",   X"fee9",   X"fee5",   X"fed4",   X"fed4",
X"fec2",   X"feb9",   X"feac",   X"fe9c",   X"fe8e",   X"fe7c",   X"fe6d",   X"fe59",   X"fe4b",
 X"fe39",   X"fe26",   X"fe1a",   X"fe03",   X"fdf7",   X"fde7",   X"fdd4",   X"fdcd",   X"fdb9",
  X"fdb2",   X"fda7",   X"fd9b",   X"fd99",   X"fd8e",   X"fd8d",   X"fd8a",   X"fd8a",   X"fd8c
",   X"fd8f",   X"fd94",   X"fd9a",   X"fda4",   X"fdac",   X"fdb8",   X"fdc5",   X"fdd2",   X"fde
1",   X"fdf1",   X"fe03",   X"fe14",   X"fe29",   X"fe3b",   X"fe50",   X"fe65",   X"fe77",   X"fe
8c",   X"fea0",   X"feb0",   X"fec4",   X"fed4",   X"fee3",   X"fef5",   X"ff00",   X"ff0d",   X"ff
```

```
1b",  X"ff22",  X"ff2f",  X"ff36",  X"ff3e",  X"ff46",  X"ff4c",  X"ff51",  X"ff57",  X"ff5
c",  X"ff60",  X"ff65",  X"ff69",  X"ff6e",  X"ff73",  X"ff77",  X"ff7c",  X"ff82",  X"ff86
",  X"ff8d",  X"ff93",  X"ff99",  X"ffa1",  X"ffa5",  X"ffae",  X"ffb4",  X"ffba",  X"ffc1"
,  X"ffc6",  X"ffcd",  X"ffd4",  X"ffdb",  X"ffe2",  X"ffeb",  X"fff0",  X"fff9",  X"fffe",
 X"0003",  X"000a",  X"000c",  X"0013",  X"0016",  X"0019",  X"001d",  X"001f",  X"002
2",  X"0025",  X"0028",  X"002b",  X"002f",  X"0033",  X"0039",  X"003f",  X"0045",  X
"004e",  X"0057",  X"0062",  X"006d",  X"007b",  X"0089",  X"0098",  X"00a9",  X"00ba"
,  X"00cd",  X"00e0",  X"00f3",  X"0108",  X"011b",  X"012f",  X"0143",  X"0156",  X"0
168",  X"0179",  X"0189",  X"0197",  X"01a4",  X"01af",  X"01b9",  X"01c1",  X"01c9",
 X"01cf",  X"01d4",  X"01d9",  X"01dd",  X"01e0",  X"01e3",  X"01e5",  X"01e7",  X"01e
8",  X"01ea",  X"01eb",  X"01ec",  X"01ee",  X"01ee",  X"01ef",  X"01f0",  X"01f0",  X"
01f0",  X"01ef",  X"01ee",  X"01ec",  X"01ea",  X"01e7",  X"01e3",  X"01de",  X"01d8",
 X"01d1",  X"01c8",  X"01bf",  X"01b4",  X"01a8",  X"019b",  X"018d",  X"017e",  X"016
d",  X"015c",  X"0149",  X"0136",  X"0122",  X"010e",  X"00f9",  X"00e5",  X"00d0",  X
"00bb",  X"00a5",  X"0091",  X"007d",  X"006b",  X"0059",  X"0048",  X"0039",  X"002b"
,  X"001f",  X"0014",  X"000a",  X"0002",  X"fffc",  X"fff6",  X"fff1",  X"ffed",  X"ffea"
,  X"ffe7",  X"ffe5",  X"ffe4",  X"ffe2",  X"ffe1",  X"ffe0",  X"ffdf",  X"ffdd",  X"ffdc",
  X"ffdb",  X"ffda",  X"ffd8",  X"ffd6",  X"ffd4",  X"ffd2",  X"ffd0",  X"ffce",  X"ffcc",
 X"ffcb",  X"ffca",  X"ffc9",  X"ffc9",  X"ffc9",  X"ffca",  X"ffcb",  X"ffcd",  X"ffd0",
X"ffd3",  X"ffd6",  X"ffda",  X"ffdf",  X"ffe3",  X"ffe8",  X"ffee",  X"fff3",  X"fff9",  X"
fffe"

);
begin
process (Clk)
begin
if rising_edge(Clk) then
data <= ROM(TO_INTEGER(addr));
end if;
end process;
end imp;
```

```
-- Quartus II VHDL Template
-- Four-State Moore State Machine

-- A Moore machine's outputs are dependent only on the current state.
-- The output is written only when the state changes.  (State
-- transitions are synchronous.)

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rotaryController is

port(
clk              : in   std_logic;
input    : in   std_logic_vector (1 downto 0);
reset_n  : in   std_logic;
output   : out  unsigned (1 downto 0)
);

end entity;

architecture rtl of rotaryController is

-- Build an enumerated type for the state machine
type state_type is (s0, s1, s2, s3, s4 , s5, s6);

-- Register to hold the current state
signal state   : state_type;
--signal counter : unsigned (3 downto 0);
begin


-- Logic to advance to the next state
process (clk)
begin
if reset_n = '0' then
state <= s0;
elsif (rising_edge(clk)) then
case state is
when s0=>
if input = "01" then
state <= s1;
elsif input = "10" then
state <= s4;
else
state <= s0;
end if;
when s1=>
if input = "11" then
state <= s2;
elsif input = "00" then
state <= s0;
else
state <= s1;
end if;
when s2=>
if input = "10" then
state <= s3;
elsif input = "00" then
state <= s0;
else
state <= s2;
```

```
end if;
when s3 =>
if input = "00" then
state <= s0;
else
state <= s3;
end if;
when s4 =>
if input = "11" then
state <= s5;
elsif input = "00" then
state <= s0;
else
state <= s4;
end if;
when s5 =>
if input = "01" then
state <= s6;
elsif input = "00" then
state <= s0;
else
state <= s5;
end if;


when s6 =>
if input = "00" then
state <= s0;
else
state <= s6;
end if;
end case;
end if;
end process;

-- Output depends solely on the current state
process (state)
begin
case state is
when s0 =>
output <= "00";
when s1 =>
output <= "00";
when s2 =>
output <= "00";
when s3 =>
output <= "10";
when s4 =>
output <= "00";
when s5 =>
output <= "00";
when s6 =>
output <= "11";

end case;
end process;

end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity sram_controller is

  port (
    signal chipselect : in std_logic;
    signal write, read : in std_logic;
    signal address  :  in std_logic_vector(17 downto 0);
    signal readdata : out std_logic_vector(15 downto 0);
    signal writedata : in std_logic_vector(15 downto 0);
    signal byteenable : in std_logic_vector(1 downto 0);

    signal SRAM_DQ   : inout std_logic_vector(15 downto 0);
    signal SRAM_ADDR : out std_logic_vector(17 downto 0);
    signal SRAM_UB_N, SRAM_LB_N : out std_logic;
    signal SRAM_WE_N, SRAM_CE_N : out std_logic;
    signal SRAM_OE_N            : out std_logic
    );

end sram_controller;

architecture dp of sram_controller is
begin

  SRAM_DQ <= writedata when write = '1' else (others => 'Z');
  readdata <= SRAM_DQ;
  SRAM_ADDR <= address;
  SRAM_UB_N <= not byteenable(1);
  SRAM_LB_N <= not byteenable(0);
  SRAM_WE_N <= not write;
  SRAM_CE_N <= not chipselect;
  SRAM_OE_N <= not read;

end dp;
```

```c
#ifndef CONTROL_H_
#define CONTROL_H_

#define CONTROL_RIGHT 2
#define CONTROL_LEFT 3

#define CONTROL_DIR() IORD_16DIRECT (ROTARY_BASE, 0)

#endif /*CONTROL_H_*/
```

```
#include <io.h>
#include <system.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/alt_irq.h>
#include <alt_types.h>
#include "vga.h"
#include "tiles.h"
#include "control.h"
#include "sound.h"
#define GRAVITY 3
#define X_STEP 48

#define MAX_LEVEL 40
#define LEVEL_INC 5
#define MIN_SPEED 0x1FFF
#define START_SPEED 0xFFFF
#define SPEED_INC ((START_SPEED/MAX_LEVEL)*2)

int seeder = 2020;
int highScore = 0;
int score = 0;
int health = 8;
int man_x, next_man_x;
int man_y, next_man_y;
int man_y_vel;
int genPlatform = 0, drawPlatform = 0;
int platformType, platformX, lastY = 0;
int tileOffset, pixOffset;
int tiles[40][32];
int onTile = 0, scoreUpdated = 0;
int tile_under = TILE_BLACK;
int bouncing = 0;
int bounce_vel = 0;
int topHit = 0;
int onSand = 0, sand_row = −1, sandCounter = 0, bFrom = −1, speed, row_under = 0
;
int level, genLevelProb;

int  genY(int actualY) {
    int topy;
    GET_OFFSETS();
    topy = tileOffset + actualY − 1;

    if(topy > 31) topy −= 30;
    if(topy < 2) topy += 30;

    return topy;
}

void setupScreen() {
    srand(seeder);
    SET_SPEED (0);

    level = MAX_LEVEL;
    genPlatform = drawPlatform = lastY = onTile = scoreUpdated =
        bouncing = bounce_vel = topHit = onSand = sandCounter = 0;

    score = 0;
    health = 8;
    man_x = (MAX_X - MIN_X)/2 - MAN_W/2;
    man_y = TILE_H*15 - MAN_H;
    man_y_vel = 0;
    int x, y;
    for(x = 0; x < 40; x++) {
        SET_TILE(x, 0, TILE_WHITE);
        SET_TILE(x, 1, TILE_SPIKE_REV);
    }
    for(x = 0; x < 40; x++) {
        for(y = 2; y <= 31; y++) {
            SET_TILE(x, y, TILE_BLACK);
        }
    }

    puttiles(0, 0, 3, TILE_HEALTH1);
    puttiles(33, 0, 3, TILE_SCORE1);

    for(x = 10; x <=30; x = x + 3) {
         puttiles(x, genY(28), 3, TILE_BRICK1);
    }

    next_man_x = man_x;
    next_man_y = man_y;

    genLevelProb = (MAX_LEVEL*(level-(MAX_LEVEL/2)) - 100);
}

void bounceTo(int bVel, int bounceFrom) {
    //printf("Starting bounce\n");
    makeSound (SOUNDOFBOUNCE);
    bounce_vel = bVel;
    bouncing = 1;
    bFrom = bounceFrom;
}


void waitForInput(int count) {
   muteTheBackground ();
   while(count) {
        if(CONTROL_DIR() == CONTROL_LEFT || CONTROL_DIR() == CONTROL_RIGHT) {
               count --;
        }
    }
}

void gameOver() {
    int x, y;

    seeder += score;

    if(score > highScore) {
        highScore = score;
    }

    SET_SPEED(0);
    for(x = 0; x < 40; x++) {
        for(y = 0; y <= 31; y++) {
            SET_TILE(x, y, TILE_WHITE);
        }
    }

    puttiles(18, genY(10), 4, TILE_GAME_OVER1);

    puttiles(17, genY(17), 3, TILE_SCORE1);
    setScore(score, 20, genY(17));
    puttiles(16, genY(18), 2, TILE_HIGH1);
    puttiles(18, genY(18), 3, TILE_SCORE1);
    setScore(highScore, 21, genY(18));

    man_x = (MAX_X - MIN_X)/2 - MAN_W/2;
    man_y = TILE_H*9 - MAN_H;

    SET_MAN_DIR (MAN_FORWARD);
    SET_MAN_X(man_x);
    SET_MAN_Y(man_y);

    waitForInput(10000);

    setupScreen();
    SET_SPEED (START_SPEED);
}

void updateManY() {
    int x_tile, y_tile;

    x_tile = (man_x + MAN_W)/TILE_W;
    if(x_tile >= 40) x_tile = 39;
    y_tile = (man_y + MAN_H + pixOffset)/TILE_H + tileOffset - 1;
    if(y_tile > 31) y_tile -= 30;
    if(y_tile < 2) y_tile += 30;

    tile_under = tiles[x_tile][y_tile];
    if(tile_under == TILE_BLACK) {
        x_tile = (man_x)/TILE_W;
        if(x_tile >= 40) x_tile = 39;
        tile_under = tiles[x_tile][y_tile];
    }

    if(tile_under != TILE_BLACK && !bouncing) {
        onTile = 1;
    } else {
        onTile = 0;
    }

    if(!bouncing) {
        if(onTile) {
            row_under = y_tile;

            if(tile_under == TILE_SPIKE || tile_under == TILE_SPRING) {
                bounceTo(15, tile_under);
            } else if(tile_under >= TILE_SAND1 && tile_under <= TILE_SAND3 && !o
nSand) {
              onSand = 1;
              sand_row = y_tile;
              sandCounter = 0x7FFF;
            }
        } else {
            next_man_y += GRAVITY;
        }
    } else {
        //printf("Bouncing\n");
        if(!bounce_vel) {
            bouncing = 0;
           // printf("Done Bouncing\n");
        }
    }
}
```

```
static void drawScreen(void * context, alt_u32 id){
    int x, new_y, pth = topHit;
    GET_OFFSETS();

    setScore(score, SCORE_X, 0);
    setHealth(health);

    if (man_x < next_man_x){
        man_x += 8;
        if(man_x > MAX_X - MAN_W) {
             man_x = next_man_x = MAX_X - MAN_W;
        }
        SET_MAN_DIR (MAN_RIGHT);
    } else if (man_x > next_man_x){
        man_x -= 8;
        if(man_x < MIN_X) {
            man_x = next_man_x = MIN_X;
        }
        SET_MAN_DIR (MAN_LEFT);
    } else {
        SET_MAN_DIR (MAN_FORWARD);
    }

    new_y = man_y;
    if(bouncing) {
        new_y -= bounce_vel;
        if(bounce_vel >= 1 ) bounce_vel --;
    } else if (onTile) {
        new_y =  row_under - tileOffset + 1;
        if(new_y < 2) new_y += 30;
        new_y = new_y*16 - MAN_H - pixOffset;
    } else if (man_y < next_man_y){
        new_y += 1;
    } else if (man_y > next_man_y){
        new_y -= 1;
    }


    topHit = 0;
    if(new_y > (MAX_Y - MAN_H)) {
        new_y = (MAX_Y - MAN_H);
        bounceTo(20, TILE_BLACK);
        health--;
    } else if(new_y < (TILE_H * 2)) {
        new_y = (TILE_H * 2);
        topHit = 1;

        if(pth != topHit && topHit && (!bouncing || (bouncing && bFrom != TILE_S
PIKE))) {
            health--;
        }
    }

    man_y = new_y;

    SET_MAN_X(man_x);
    SET_MAN_Y(man_y);

    if(lastY != tileOffset) {
        genPlatform = 0;
        for(x = 0; x < 40; x++) {
            SET_TILE(x, tileOffset, TILE_BLACK);
        }
```

```c
        if(drawPlatform) {
            lastY = tileOffset;

            putPlatform(platformX, tileOffset, platformType);
            drawPlatform = 0;
        }
    }

    if(sandCounter < 0) {
        onSand = 0;
        for(x = 0; x < 40; x++) {
            SET_TILE(x, sand_row, TILE_BLACK);
        }
    }

    if(health == 0) {
        makeSound (SOUNDOFDEATH);
        gameOver();
    }

    RESET_INTERRUPT();
}

void generatePlatform() {
    if(!genPlatform) {
        genPlatform = 1;
        drawPlatform = (rand()%(MAX_LEVEL*MAX_LEVEL) <= genLevelProb);

        if(drawPlatform) {
               platformType = rand()%100;
               if(platformType >= 0 && platformType < 40) {
                    platformType = TILE_BRICK1;
               } else if(platformType >= 40 && platformType < 60) {
                    platformType = TILE_SAND1;
               } else if(platformType >= 60 && platformType < 80) {
                    platformType = TILE_SPIKE;
               } else if(platformType >= 80 && platformType < 95) {
                    platformType = TILE_SPRING;
               } else {
                    platformType = TILE_POWERUP;
                    if(health == 8) {
                        genPlatform = 0;
                        drawPlatform = 0;
                    }
               }

               platformX =  rand()%35;
        }
    }
}

void updateManPosition() {
    if(next_man_x == man_x) {
        if(CONTROL_DIR() == CONTROL_LEFT) {
            next_man_x -= X_STEP;
        } else if(CONTROL_DIR() == CONTROL_RIGHT) {
            next_man_x += X_STEP;
        }

        if(next_man_x < MIN_X) next_man_x = MIN_X;
        else if(next_man_x > (MAX_X - MAN_W)) next_man_x = (MAX_X - MAN_W);
    }
    updateManY();
}

void updateScore() {
   if(onTile == 1 && !scoreUpdated) {
        score++;
        if(tile_under == TILE_SPIKE) {
            health--;
        } else if (tile_under == TILE_POWERUP) {
            if(health < 8) {
                health++;
            }
        }

        level = MAX_LEVEL - score/LEVEL_INC;
        if(level <= 0) level = 1;

        speed = START_SPEED - MAX_LEVEL*SPEED_INC + level*SPEED_INC;
        if(speed < MIN_SPEED) speed = MIN_SPEED;
        SET_SPEED(speed);

        genLevelProb = (MAX_LEVEL*(level-(MAX_LEVEL/2)) - 100);
        if(genLevelProb < MAX_LEVEL*2) genLevelProb = MAX_LEVEL*2;

        scoreUpdated = 1;
   } else if(onTile == 0) {
        scoreUpdated = 0;
   }
}

int main() {
    alt_irq_register(VGA_IRQ, NULL, (void*)drawScreen);
    alt_irq_register(GAME_SOUND_IRQ, NULL, (void*)newBackGroundMusic );
    setupScreen();

    waitForInput(1);
    SET_SPEED (START_SPEED);

    while(1) {
       generatePlatform();
       updateManPosition();
       updateScore();
       if(onSand) sandCounter --;
    }

    return 0;
}
```

```
#ifndef SOUND_H_
#define SOUND_H_

#include "back.h"

int vadd = 0;

#define SOUNDOFDEATH 0
#define SOUNDOFBOUNCE 1

void makeSound (int sound ){
       IOWR_16DIRECT (GAME_SOUND_BASE, 2, 0);
       IOWR_16DIRECT (GAME_SOUND_BASE, 0 , sound);

       IOWR_16DIRECT (GAME_SOUND_BASE, 4, 1);
       IOWR_16DIRECT (GAME_SOUND_BASE, 2, 1);
}

void muteTheBackground (){
  int i =0;
  while (i < 32){
        IOWR_16DIRECT (GAME_SOUND_BASE, 64+i *2 , 0);
        i++;
  }
  return ;
}

static void newBackGroundMusic (void * context, alt_u32 id)
{
  int i =0;
  while (i < 32){
        IOWR_16DIRECT (GAME_SOUND_BASE, 64+i *2 , value[vadd]);
        vadd++;
        if (vadd >= 208212){
            vadd = 0;
        }
    i++;
  }
}

#endif /*SOUND_H_*/
```

```
#ifndef TILES_H_
#define TILES_H_

#include "vga.h"

#define TILE_0 0
#define TILE_1 1
#define TILE_2 2
#define TILE_3 3
#define TILE_4 4
#define TILE_5 5
#define TILE_6 6
#define TILE_7 7
#define TILE_8 8
#define TILE_9 9
#define TILE_BLACK 10
#define TILE_BRICK1 11
#define TILE_BRICK2 12
#define TILE_BRICK3 13
#define TILE_GAME_OVER1 14
#define TILE_GAME_OVER2 15
#define TILE_GAME_OVER3 16
#define TILE_GAME_OVER4 17
#define TILE_HEALTH1 18
#define TILE_HEALTH2 19
#define TILE_HEALTH3 20
#define TILE_HEART 21
#define TILE_HIGH1 22
#define TILE_HIGH2 23
#define TILE_POWERUP 24
#define TILE_SAND1 25
#define TILE_SAND2 26
#define TILE_SAND3 27
#define TILE_SCORE1 28
#define TILE_SCORE2 29
#define TILE_SCORE3 30
#define TILE_SPIKE_REV 31
#define TILE_SPIKE 32
#define TILE_SPRING 33
#define TILE_WHITE 34

#define SCORE_X 36
#define HEALTH_X 3

void puttiles(int x, int y, int count, int startTile) {
    int i = 0;
    for(i = 0; i<count; i++) {
        SET_TILE((x+i), y, (startTile+i));
    }
}

void setScore(int score, int x, int y) {
    int i;
    for(i = 3; i >= 0; i--) {
        SET_TILE((x+i), y, score%10);
        score = score/10;
    }
}

void setHealth(int health) {
    int i;
    for(i = 0; i < 8; i++) {
        SET_TILE((HEALTH_X+i), 0, (i < health) ? TILE_HEART : TILE_WHITE);
    }
}

void putPlatform(int x, int y, int platform) {
    int i;
    for(i = 0; i < 3; i++) {
        SET_TILE((x+i), y, (platform == TILE_SPIKE || platform == TILE_SPRING ||
 platform == TILE_POWERUP) ? platform : platform + i);
    }
    for(i = 0; i < 3; i++) {
        SET_TILE((x+3+i), y, (platform == TILE_SPIKE || platform == TILE_SPRING
|| platform == TILE_POWERUP) ? platform : platform + i);
    }
}

#endif /*TILES_H_*/
```

```
#ifndef VGA_H_
#define VGA_H_

#include <system.h>

#define MIN_X 4
#define MAX_X 639
#define MIN_Y 32
#define MAX_Y 480

#define MAN_W 20
#define MAN_H 28
#define TILE_W 16
#define TILE_H 16

extern int tiles[40][32];

#define PROPERTY_BITS 2
#define PROPERTY_TILE 0
#define PROPERTY_MAN 1
#define PROPERTY_SETTING 2
#define PROPERTY_IRQ 3

#define CREATE_ADDRESS(property, address) \
    (0x1FFFF & ((((property) << (16-PROPERTY_BITS)) | (address))*2))

#define SET_VGA_PROP(property, address, value) \
    IOWR_16DIRECT(VGA_BASE, CREATE_ADDRESS(property, address), (value))
#define GET_VGA_PROP(property, address) \
    IORD_16DIRECT(VGA_BASE, CREATE_ADDRESS(property, address))


/* Tile Settings */
#define CREATE_TILE_ADDR(x, y) \
    (((y & 0x1F) << 6) | (x & 0x3F))
#define SET_TILE(x, y, value) \
    tiles[x][y] = value; \
    SET_VGA_PROP(PROPERTY_TILE, CREATE_TILE_ADDR(x, y), (value))
#define GET_TILE(x, y) \
    (0x2F & GET_VGA_PROP(PROPERTY_TILE, CREATE_TILE_ADDR(x, y)))

#define GET_OFFSETS() \
    alt_u16 readdata = IORD_16DIRECT(VGA_BASE, 0) & 0xffc0; \
    tileOffset = ((readdata & 0xf800) >> 11); \
    pixOffset = ((readdata & 0x0780) >> 7)

#define RESET_INTERRUPT() \
    SET_VGA_PROP(PROPERTY_IRQ, 0, 0)

/* Man Settings */
#define MAN_FORWARD 0
#define MAN_LEFT 1
#define MAN_RIGHT 2

#define SET_MAN_PROP(param, value) SET_VGA_PROP(PROPERTY_MAN, param, value)

#define SET_MAN_X(x) SET_MAN_PROP(0, x)
#define SET_MAN_Y(y) SET_MAN_PROP(1, y)
#define SET_MAN_DIR(d) SET_MAN_PROP(2, d)

/* Settings */
#define SETTING_SPEED 0
```

```
#define SET_SETTING(param, value) SET_VGA_PROP(PROPERTY_SETTING, param, value)
#define SET_SPEED(s) \
    speed = s; \
    SET_SETTING(SETTING_SPEED, s)

#endif /*VGA_H_*/
```

```
/* GIMP RGB C-Source image dump (1.c) */

static const struct {
  unsigned int   width;
  unsigned int   height;
  unsigned int   bytes_per_pixel; /* 3:RGB, 4:RGBA */
  unsigned char  pixel_data[16 * 16 * 3 + 1];
} gimp_image = {
  16, 16, 3,
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376|||DDD\16\16\16\0\0\0\0\0\0\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\\\\\\\224\224\224\213\213\213\0"
  "\0\0\0\0\0\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\262\262\262\0\0\0\0\0\0\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\262\262\262\0\0"
  "\0\0\0\0\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\262\262\262\0\0\0\0\0\0\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\262\262\262\0\0"
  "\0\0\0\0\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\262\262\262\0\0\0\0\0\0\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\262\262\262\0\0"
  "\0\0\0\0\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\366"
  "\366\366\366\366\366\255\255\255\0\0\0\0\0\0\366\366\366\366\366\366\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
  "\0\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
  "\0\0\0\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376\376"
  "\376",
};
```

```
mif: mif.cpp $(FILE)
        cp -f $(FILE) input.c
        g++ -fno-stack-protector mif.cpp
        ./a.out > out.txt
        rm -f input.c a.out

constant: constant.cpp $(FILE)
        cp -f $(FILE) input.c
        g++ -fno-stack-protector constant.cpp
        ./a.out > out.txt
        rm -f input.c a.out

tile: tile.cpp $(FILE)
        cp -f $(FILE) input.c
        g++ -fno-stack-protector tile.cpp
        ./a.out > out.txt
        mv out.txt $(FILE).tile.txt
        rm -f input.c a.out
```

```
#include <iostream>
#include <vector>
#include <string>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "input.c"

using namespace std;

const char *byte_to_binary(int x) {
  static char b[9];
  b[0] = 0;

  int z;
  for (z = 256; z > 0; z >>= 1) {
    strcat(b, ((x & z) == z) ? "1" : "0");
  }

  return b;
}

void removeFirst (char *s) {
      if (*s == '\0') return;
          *s = *(s+1);
              removeFirst (s+1);
}


string f(int i) {
  char d[9];
  sprintf(d, "%s", byte_to_binary(i));
  removeFirst(d);
  return d;
}

string f2(int i) {
  char d[2];
  sprintf(d, "%02x", i);
  return d;
}

int main() {
  int pixels = gimp_image.width*gimp_image.height;
  char pcount[100];
  sprintf(pcount, "%d", pixels-1);
  string out = "";
  //out += "type _type is array(0 to " + (string) pcount + ") of std_logic_vecto
r(" + ((gimp_image.bytes_per_pixel==3) ? "23" : "24") + " downto 0);\n";
  //out += "constant _image : _type := (";
  if(gimp_image.bytes_per_pixel==4){
    //out += "\"";
    out += (gimp_image.pixel_data[3] != 0x00) ? "1" : "0";
    out += f(gimp_image.pixel_data[0]);
    out += f(gimp_image.pixel_data[1]);
    out += f(gimp_image.pixel_data[2]);
  } else {
    //out += "x\"";
    out += f2(gimp_image.pixel_data[0]);
    out += f2(gimp_image.pixel_data[1]);
    out += f2(gimp_image.pixel_data[2]);
  }
  out += "\t";
  int i = gimp_image.bytes_per_pixel;
  while(i < pixels*gimp_image.bytes_per_pixel) {
    if(gimp_image.bytes_per_pixel==4){
      //out += "\", \"";
      out += (gimp_image.pixel_data[i+3] != 0x00) ? "1" : "0";
      out += f(gimp_image.pixel_data[i+0]);
      out += f(gimp_image.pixel_data[i+1]);
      out += f(gimp_image.pixel_data[i+2]);
      i = i + 4;
    } else {
      //out += "\", x\"";
      out += f2(gimp_image.pixel_data[i+0]);
      out += f2(gimp_image.pixel_data[i+1]);
      out += f2(gimp_image.pixel_data[i+2]);
      i = i + 3;
    }
  out += "\t";
  }

  //out += "\");";
  cout << out << endl;

  return 0;
}
```

```
for f in inputs/*.tile.c; do
  echo "Processing $f file..";
  make tile FILE=$f;
  mv $f.tile.txt outputs/
done

n=0
i=0
cat tile_mif_start.txt > tiles.mif
for f in outputs/*.tile.c.tile.txt; do
  echo "$n --> $f"
  while read line; do
    if [ -n "$line" ]; then
      echo -e "\t$i   :   $line;" >> tiles.mif;
      i=$(($i + 1));
    fi
  done < $f
  n=$(($n + 1));
done

if [ "$i" -ne "12288" ]; then
  echo -e "\t[$i..12287]   :   0;" >> tiles.mif;
fi

echo "END;" >> tiles.mif
```

```
#include <iostream>
#include <vector>
#include <string>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "input.c"

using namespace std;

int main() {
  int pixels = gimp_image.width*gimp_image.height;
  string out = "";
  int i = 0;
  int o;
  char d[9];
  while(i < pixels*gimp_image.bytes_per_pixel) {
      o = (((gimp_image.pixel_data[i+0] >> 3)) << 11) & 0xF800;
      o += (((gimp_image.pixel_data[i+1] >> 2)) << 5) & 0x07E0;
      o += (gimp_image.pixel_data[i+2] >> 3) & 0x001F;
      sprintf(d, "%d", o);
      out += d;
      i = i + 3;
        out += "\n";
  }
  cout << out << endl;

  return 0;
}
```