Fortune Teller

Final Project Report

Leo Gertsenshteyn (lpg2006@columbia.edu)
Russ Santillanes (res2049@columbia.edu)
Kathryn Hagan (kmh2124@columbia.edu)

May 9, 2006

TABLE OF CONTENTS

A. Project Proposal

The Concept:

Fortune Teller is a digital version of a 1950's-era penny arcade fortune teller machine (see "Fortune Tellers" at http://marvin3m.com/arcade/ for an example). The program will display a graphic of a gypsy at run time, and then a key-press from the user will prompt the gypsy to animate and music will play while she gazes into her crystal ball. A random fortune will be displayed below her on the screen.

The Implementation:

- Graphics / animation will be accomplished using run length encoding, reading from a statically stored set of encoded images in the SRAM and decoding them through hardware on the FPGA to display them on the screen through the Video Codec.
- Sound will be output through the onboard Stereo Codec.
- Fortunes will be one sentence long each, stored in a BRAM on the FPGA. Fortunes will be chosen through a random number generator on the FPGA to select which fortune to display.

B. Project Design

I. OVERVIEW

The fortune teller will be composed of three main parts: the graphical display, the textual fortune generation and display, and the audio output. The program will use a custom encoding method to store the video in SRAM. There will also be a main control program, a small C program that will run on the Microblaze CPU. It will wait for a keystroke from the user, generate a fortune when input is received (passing it to the video module to be displayed), and resume waiting for the next user input.

Figure 1 illustrates the overall system architecture.

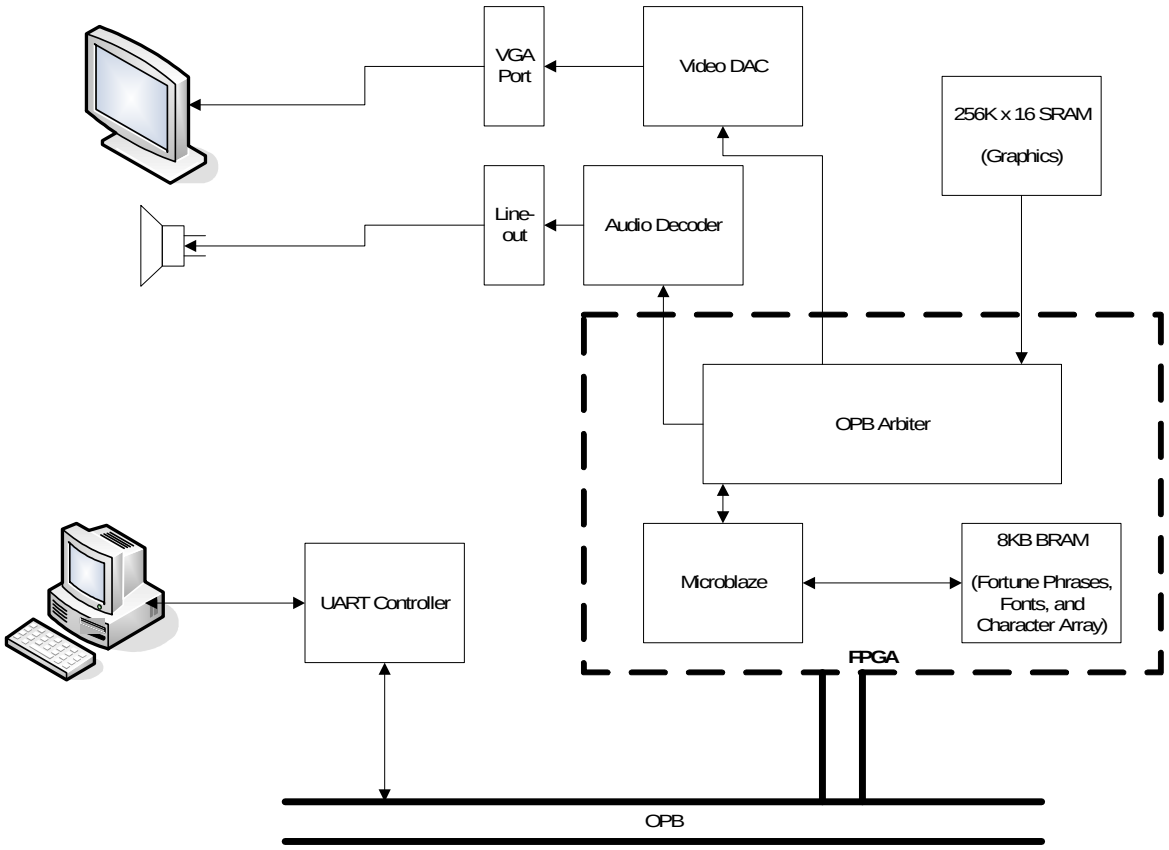


Figure 1. System architecture.

II. DESIGN PROCESS

The following is a list of project milestones, in the order they will be completed.

1. base graphic display (8-bit RGB color, no animation)
2. fortune generation and text display
3. main control program
4. animation
5. audio output

Milestones 1-3 will be complete on April 13 (the 75% mark); the entire project will be complete on May 9.

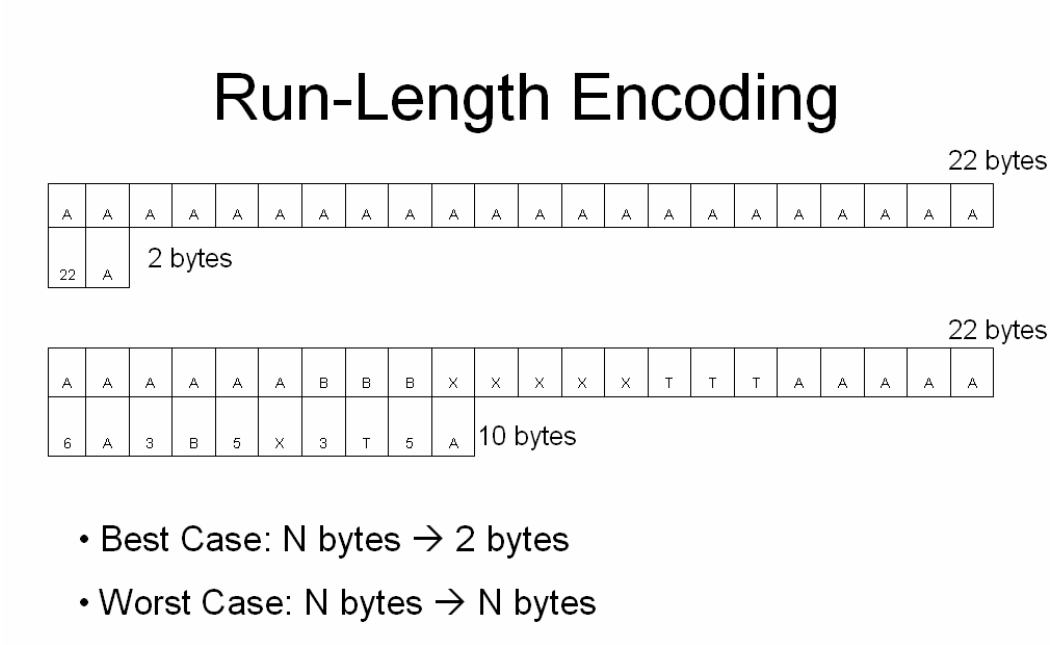
We never hit our milestones. We kept pushing back and pushing back the deadlines until it became impossible to meet them.

The design process we used was one of our main reasons for failure. Due to scheduling conflicts, we did not set up a regular meeting time, either with the TA or as a group. This led to more immediate deadlines getting prioritized and a failure to hold each other accountable and know the status of the project. In addition, though our design document split the project into three main parts, we had no clear division of responsibilities.

Our initial plan of attack was to take existing lab code and marry it with previous groups' project code to attain a great deal of the framework which we wanted. This became a large issue when things didn't

quite “marry” so well. We found that we had been completing the labs without a good idea of what was actually going on behind the scenes, and this was a liability when it came time to adapt them to the project code.

III. ENCODING/DECODING IMAGES



A quick overview of run-length encoding

Our LKR encoder operates as follows:

File format:
  first 16x3 bytes :
    RED, GREEN, BLUE (eight bit color values for color code 0)
    RED, GREEN, BLUE (eight bit color values for color code 1)
    ...
    RED, GREEN, BLUE (eight bit color values for color code 15)
  next however-many bytes:
    [run_length - 1, color_code] (high 4 bits=unsigned run length -1, low 4 bits=unsigned color code)

encoder:
  input = ASCII PPM color image file

It skips PPM metadata, reads RGB values, one at a time, puts those into a hash and keeps track of how many seen total, giving an error if more than 16 total encountered. It then tracks run lengths, and at each color change, pushes these onto a queue. Once the file is done and we know that it was valid, the color table and queue get packed into hex and written to a file.

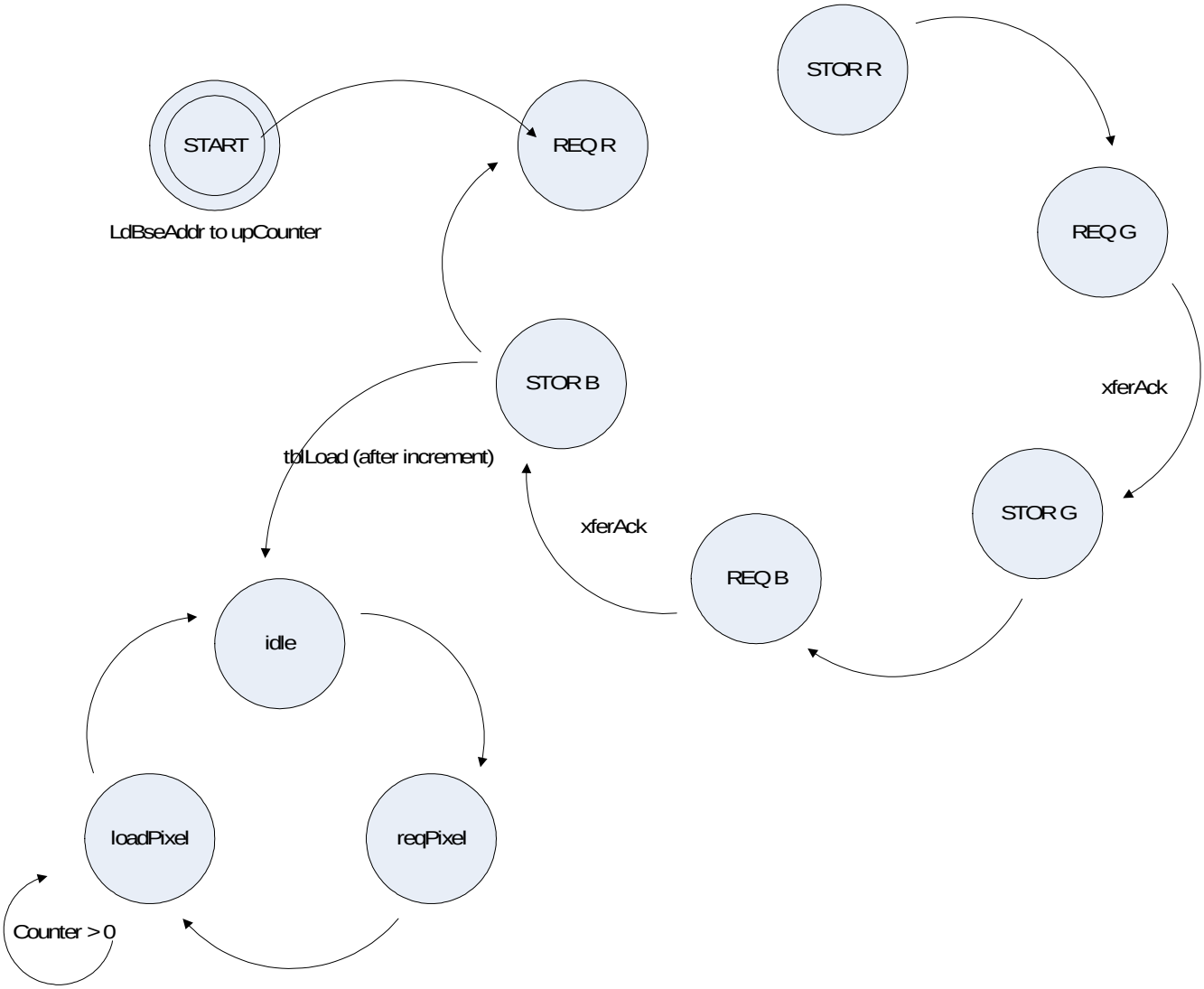


Figure 1a. State Machine for Decoder

Decoder:

There is more than enough time during the VGA SYNC and BACK_PORCH periods to load the color table into memory so that color codes can be decoded on the fly as they come from the RAM. Once the table is filled, the decoder idles until it's time to start pulling pixels. The run counter gets loaded when we receive xferAck and have the next run-length-encoded byte in hand.

IV. GRAPHICAL DISPLAY

The video module will provide an interface for the main program to write text to the screen, as well as displaying and animating the gypsy graphic. (The graphic will be based on the one on the title page of this document, a public-domain piece of internet clip-art from the following webpage: http://serp.la.asu.edu/clipart_dir/clipartidx.html .)

The SRAM is the store for the encoded images. It is set up without an arbiter, because the FPGA is the only thing reading or writing to it. It is set-up to be treated like a ROM through a tri-state buffer and the FPGA fetches parcels of memory to be sliced back together. The SRAM contains only the graphical data. We used Lab 6 and heavily modified it to include the BRAM as well as tested read and writes through the Microblaze.

Initially we were going to take the Scorched_Earf code and pare it down to what we required, because they had handled a great deal of the SRAM implementation already. This became complicated when they used a different approach to building a project (no Xilinx Platform Studio support) a convoluted make process, and then once we tried porting to XPS we received errors of a completely interstellar nature.

At first we made the SRAM work and function as an OPB peripheral in the spirit of Lab 6. We then modified that to attach the SRAM pins to the VGA peripheral, bypassing the OPB bus entirely. This is where we fell short in our implementation. We were unable to properly get the SRAM to act as ROM and while we could read and write in a C program, we were unable to make the jump to a hardware implementation.

Because of memory constraints, we will be storing/using the BRAM for the fortunes and the fonts.
As seen in the figure below, the display will be split between a large area for the main graphic and a text area that will display the fortune. (8-bit/pixel depth RGB @ 640 x 480)

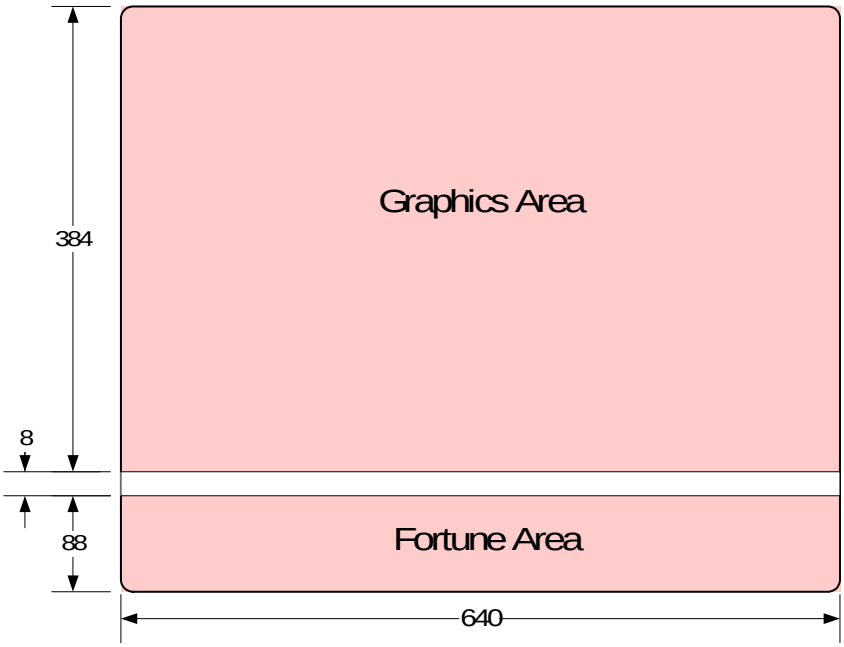


Figure 2. Division of screen real estate.

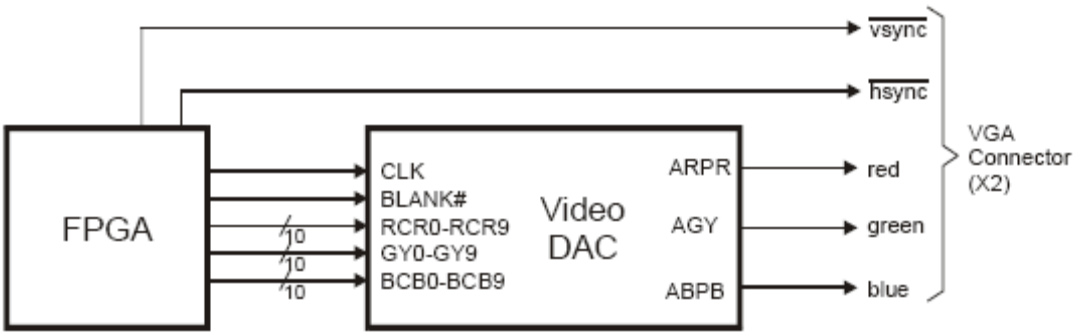


Figure 3. Block diagram for the video controller.

SRAM timing diagrams (from the data sheets):

TIMING DIAGRAMS
READ CYCLE (See Note 2)

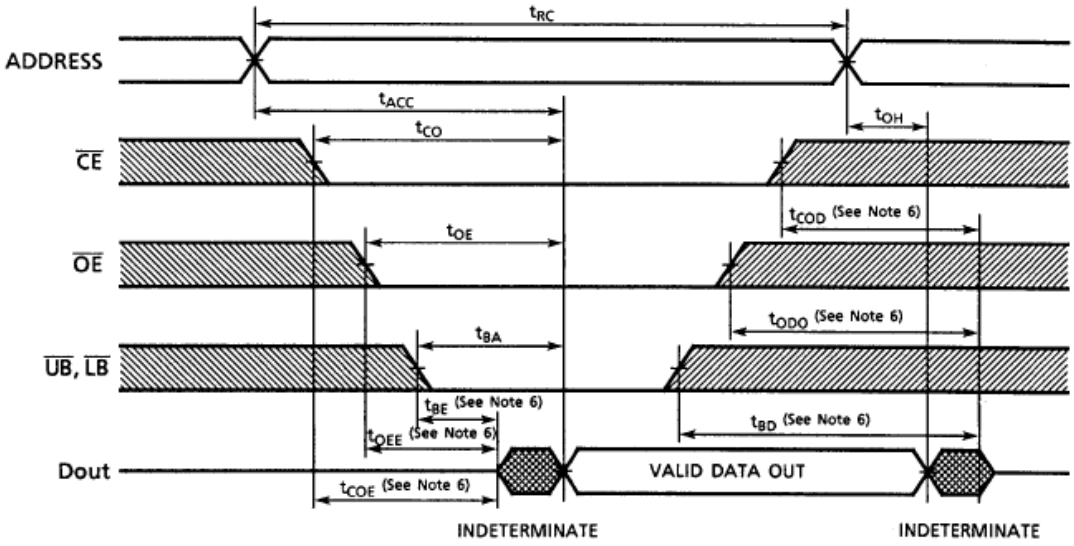


Figure 4.

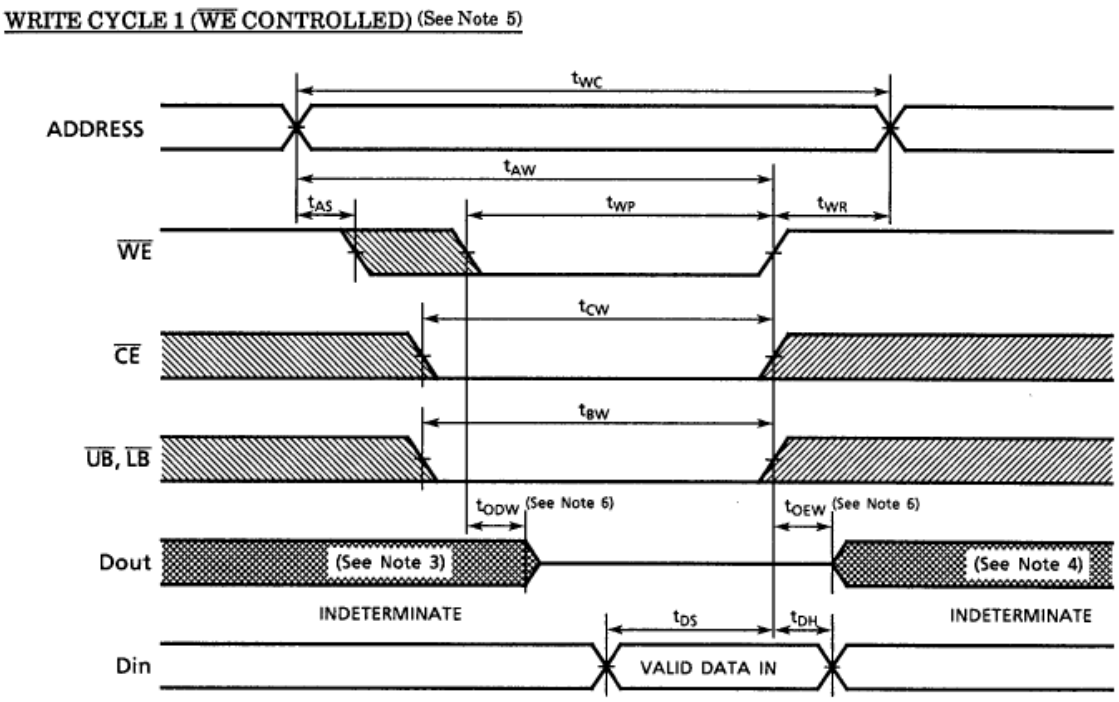


Figure 5.

V. FORTUNE GENERATION AND DISPLAY

Fortunes are made in any old text editor, keeping in mind proper spacing and hex padding allowances and then converting said file to hex. This hex code is then statically loaded into the BRAM similar to the way the font is loaded.

The problem with dealing with a limited number of BRAMs is as follows:

- 8KB of BRAM (16 BRAMs; .5K ea.)
    - 4KB (8 BRAMs) Reserved for Microblaze
    - 4KB Free
        - 5 (2.5K) for characters
        - 3 (1.5K) for font
    - How/Where to put our fortunes?

    - Solution: Shrink the area able to display text (reduce the character buffer)
        - Frees up 4 BRAMs for Fortunes

Other groups (Scorched) have disabled the Microblaze's cache, freeing up the BRAMs. Since we will most likely need the cache for the Decoder, we have decided to limit the space of the character buffer. This lines up perfectly with our decision to split the screen in two parts. If we were to use text overlay, as originally suggested, we would run into BRAM issues on a larger scale.

Text writing will utilize a peripheral similar to that which was used in lab 2 (the simple terminal emulator); this peripheral will take a C string as input and write it to the text area of the screen.

VI. SOUND

Audio was originally planned for this project, and was designated as our component to “drop” if time ran out. Sound would have been fairly simple to implement however as a number of groups have interfaced with the codec to output some audible noise with relative ease.

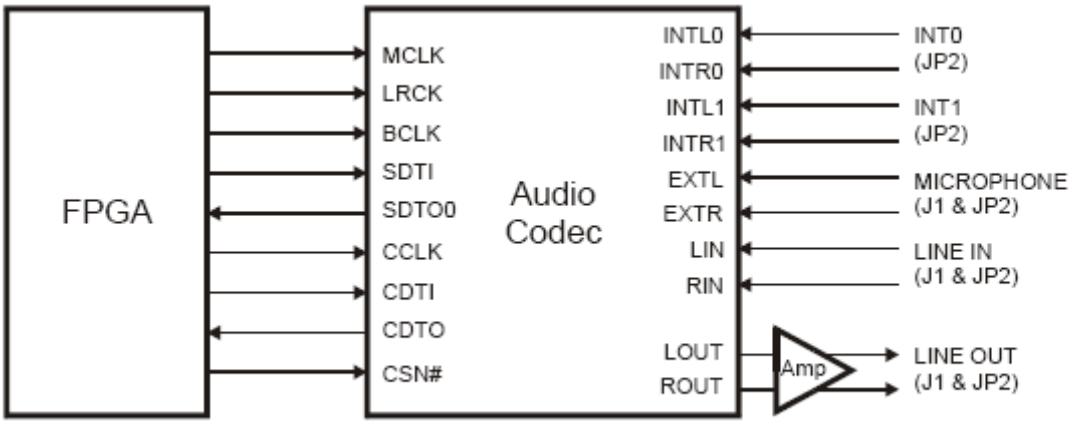


Figure 6. Block diagram of the audio codec.

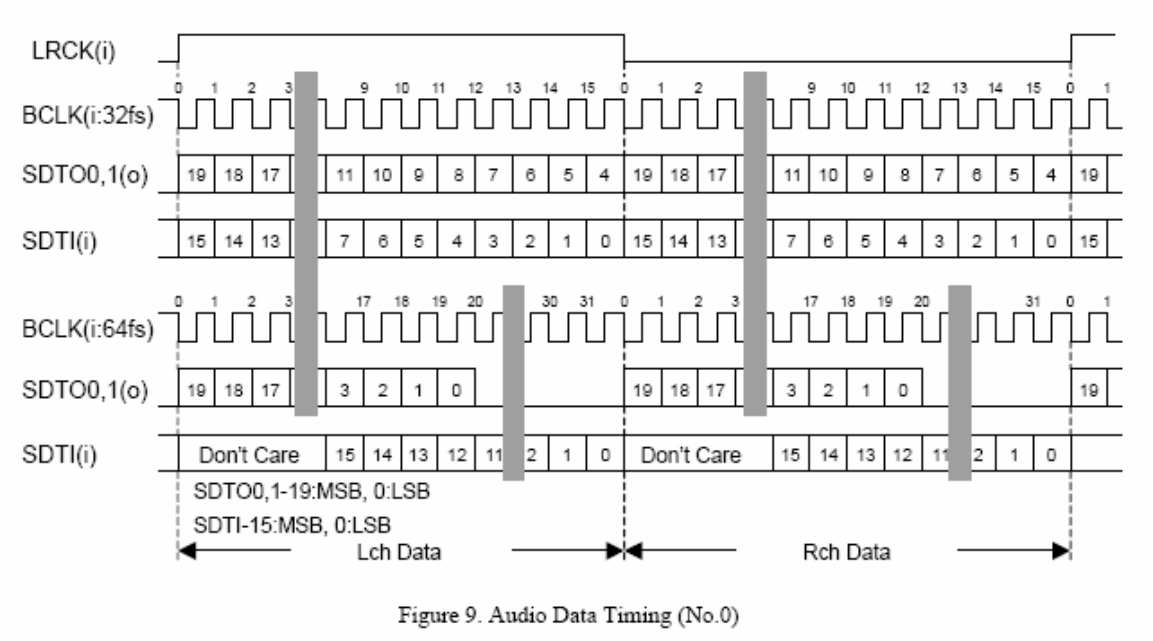


Figure 7. Audio timing diagram.

C. Lessons Learned

Be more hands on early on – by being scared of the problem or the tools we shortchanged ourselves of

valuable time and resources available to us (TAs, Prof. Edwards, etc) By waiting too long, our time horizon closes, and these resources become less and less accessible to us. If you don't learn how the tools function, you're going to make a flawed unit, or have a great deal more trouble using the instruments given to you.

Communicate, goal set, and hold accountable other team members for their respective pieces – set intermediate goals so we have some way to gauge how much progress we're making because a good understanding of how the system is supposed to work does not necessarily translate into ease of actually coding it.

Porting previous groups' code can be an exercise in version futility – Too often we realized too late that porting the old codes to the new versions of XPS and using the new libraries ended up compounding the problem that we couldn't find out what was going on until we fixed the broken platform. Perhaps a more formal in-depth instruction into the utility would be useful – incidentally, there was a great deal of support for the XSB boards in Windows, and using the linux platform made it difficult to take advantage of that.

Set up regular meetings with the TA and regular times for the group to meet -- and stick to them. The less familiar you are with VHDL, the harder this project can become and these meetings can help to alleviate a great deal of these stresses.

A steady iterative process, going from large-scale to small scale, would allow for a much smoother, more intuitive, more reliable, more easily verifiable solution. – I would encourage each group to ask of each component and connection on their schematics, "Do we know exactly what it would take to make that happen? No, really, do we? Do we need t map it into memory? Do we need to map it into some pins? How long will it take to get data from there to whatever is talking to it and back?"
Then, sit down and get them talking, regardless of what they are saying, before filling in any meaningful data or processing.

D. Complete Listing

1. main.C file which implements user key input and textual display.

```
/*
 *Leo Gertsenshteyn  -- lpg2006@columbia.edu
 *Russel Santillanes -- res2049@columbia.edu
 *Kathryn Hagan      -- kmh2124@columbia.edu / wirehead@notapattern.net
 *
 *CSEE W4840 Lab 2 (modified into project code)
 *04/12/2006
*/


#include "xbasic_types.h"
#include "xio.h"

#include "xintc_l.h"
#include "xuartlite_l.h"
#include "xparameters.h"
```

```
/* Defined in mymicroblaze/xparameters.h */
/* #define XPAR_VGA_BASEADDR 0xFEFF1000 */
/* #define XPAR_VGA_HIGHADDR 0xFEFF1fff */

#define FONT_OFFSET 0xA00

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

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

int cursorx = 0, cursory = 0;    // cursor position
unsigned char saved_cursor;  // character sharing space with the cursor

int BUFFER_SIZE = 256;
char buffer[256];        // character buffer
int volatile next_empty = 0;  // pointer to next empty slot in buffer
int volatile next_data = 0;   // pointer to next character to be read from buffer

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

  Xuint8 incoming_character;

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

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

    // TODO: check for full buffer ?

    // add new character to buffer
    buffer[next_empty] = incoming_character;

    // advance the buffer pointer
    ++next_empty;

    // check for overflow
    if(next_empty >= BUFFER_SIZE) next_empty = 0;
```

```
  }

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

}

/* Write a text string on the display */
void put_string(char *string, int row, int column)
{
  char *p = &(CHAR(row, column));
  while (*p++ = *string++)
    ;
}

void write_fortune(char *string, int startrow, int startcol) {
  char *p = &(CHAR(startrow, startcol));
  int i=0;
  int j;
  int col=startcol;
  while(string[i] != '\0') {
    *p++ = string[i++];
    for (j=0; j<100000000 ; j++)  ; /* delay (this mimics typing) */
  }
}

void generate_fortune(char* fortune) {
  strcpy(fortune, "generated fortune");
  //int test = rand(10);
}


void draw_graphic() {
  int offset=10
  // ASCII art…  commented because it doesn't fit in the BRAM
  /* put_string("..;;..", 0, offset+22);
  put_string("..;;;;;;ii;;..", 1, offset+18);
  put_string(";;;;iiiiiiii;;ii..", 2, offset+16);
  put_string(";;iiii..  ;;::....;;;;", 3, offset+12);
  put_string(";;ii..  ....;;....;;;;..", 4, offset+12);
  put_string("..ii;;::  ;;....ii  ,,;;;;", 5, offset+10);
  put_string("..;;;;;;;;;;;;.........;;", 6, offset+10);
  put_string("..;;;;LLEEff..  ;;LL..::..", 7, offset+10);
  put_string(";;LLWWWWLL;;    ..tt..ttWW;;", 8, offset+10);
  put_string(";;ii####GG;;ttLL....tt;;ttKK..", 9, offset+8);
  put_string("ii;;KKWWDDii;;tt..::iittjjKKii", 10, offset+8);
  put_string(";;;;iiWWWWGG;;ii..    ....;;KKLLii", 11, offset+6);
  put_string(";;ii;;EE##LL....        ..;;KKEE..", 12, offset+6);
```

```
  put_string("..;;;;EEWWGG......iiii......WWKKDD..ii", 13, offset+6);
  put_string(";;  ;;ffWWKK;;  ;;;;ii;;  GGKK##WWKKWW..", 14, offset+6);
  put_string("....  iiiiKKDDWW..;;LLGG,,tt##DDEEEELL::", 15, offset+4);
  put_string(";;  ..;;;;WW####GG;;    ..GG##KKWWWWGG,,", 16, offset+4);
  put_string(";;  ....;;DDLLff##GGLLffjjffGGttDD##ff", 17, offset+4);
  put_string("....  ....ii;;ttKKGGfftttttttiiWW;;iiGGGG", 18, offset+2);
  put_string("....  ;;ii;;;;LLDDttii;;;;....ttLLtttttt..", 19, offset+2);
  put_string(";;    ;;jjiiiiLLii;;;;ii;;..;;ttttii;;ttff", 20, offset+2);
  put_string(";;..  ;;LLLLttii..;;....    ..::ttttttiiii..", 21, offset+2);
  put_string("ii..;;ii;;ii;;..            ..iiiiiiLLiiLL", 22, offset+4);
  put_string("    ;;ffii;;iitt;;..              ..tt;;;;ii;;;;  ", 23, 10);
  put_string("  ttKK;;;;..;;ffii..              ..ii;;  ..GGff  ", 24, 10);
  put_string("  LLLL  ....iiii;;;;              ..ii;;..  jjKK;;", 25, 10);
  put_string("  GGLL..iiiiKKLL;;;;..          ..;;iiii..  ttLLjj", 26, 10);
  put_string(";;ffKK  EE..::;;iiii....      ..;;fftt;;iiiiLLGGff", 27, 10);
  put_string("    LLLL;;      iitt;;........;;ffii..  ,,GGDD;;tt", 28, 10);
  put_string("                ;;ttiiiittttttii..    ..GGLLii;;", 29, 10); */

  put_string("~~ Madame Xilinx knows all! ~~", 4, 40);
  put_string("(press a key to see your fortune)", 10, 40);
}

int main()
{
  volatile unsigned char *p;
  unsigned char v, vv;
  int i, j, k;
  unsigned char *to, *from;

  /* display initial graphic */
  //print("Hello World!\r\n");
  draw_graphic();

#if 0
  v = 0;
  for ( p = (unsigned char *) XPAR_VGA_BASEADDR ;
        p <= (unsigned char *) XPAR_VGA_HIGHADDR ;
        ++p ) {
#if 0
    print("writing to ");
    putnum((int) p);
    print(" value ");
    putnum(v);
    print("\r\n");
#endif
    *p = v;
    v += 1;
  }
  print("wrote to memory\r\n");
```

```
  v = 0;
  for ( p = (unsigned char *) XPAR_VGA_BASEADDR ;
        p <= (unsigned char *) XPAR_VGA_HIGHADDR ;
        ++p ) {
    vv = *p;
    if (vv != v) {
      print("read from ");
      putnum((int) p);
      print(" returned ");
      putnum(vv);
      print(" expected ");
      putnum(v);
      print("\r\n");
    }
    v += 1;
  }

  print("memory test completed. from ");
  putnum(XPAR_VGA_BASEADDR);
  print(" to ");
  putnum(XPAR_VGA_HIGHADDR);
  print("\r\n");
#endif

  /* Enable UART interrupts and register uart_handler as the ISR */
  XIntc_RegisterHandler( XPAR_INTC_BASEADDR, XPAR_MYUART_DEVICE_ID,
                         (XInterruptHandler)uart_handler, (void *)0);
  XIntc_mEnableIntr( XPAR_INTC_BASEADDR, XPAR_MYUART_INTERRUPT_MASK);
  XIntc_mMasterEnable( XPAR_INTC_BASEADDR );
  XIntc_Out32(XPAR_INTC_BASEADDR + XIN_MER_OFFSET,
XIN_INT_MASTER_ENABLE_MASK);
  microblaze_enable_interrupts();
  XUartLite_mEnableIntr(XPAR_MYUART_BASEADDR);

  for (;;) {
    unsigned char new_data;
    char buf[2];
    buf[1] = '\0';

    //print("waiting for input...\r\n");

    // get any new characters from the buffer
    if(next_empty != next_data) {
      // get character from buffer
      new_data = buffer[next_data];

      // update pointer and check for overflow
      ++next_data;
```

```
    if(next_data >= BUFFER_SIZE) next_data = 0;

    //print("got character");

    // wait for space, newline or CR
    if ((new_data == ' ') || (new_data == '\n') || (new_data == '\r'))
         {
           print("Displaying message\r\n");
        char phrase[255];
        generate_fortune(phrase);
           write_fortune(phrase, 27, 5);
        new_data = NULL;
         }
   }

   //CHAR(cursory, cursorx) = CHAR(cursory, cursorx);

   for (i = 0 ; i < 10000000 ; i++ )  ; /* delay */
  }

  return 0;
}
```

2. Makefile that includes target to download data to the SRAM.

```
## file: sram_incl.make
## include in full Makefile with the line “include sram_incl.make”
## sram download stuff
XSLOAD = /usr/cad/xess/bin/xsload
XESS_BOARD = XSB-300E

SRAM_BITFILE = implementation/sram.bin
SRAM_HEXFILE = implementation/sram.hex

full_download : $(DOWNLOAD_BIT)
	$(XSLOAD) -fpga -b $(XESS_BOARD) $(DOWNLOAD_BIT)

download-sram : $(DOWNLOAD_BIT) $(SRAM_HEXFILE)
	$(XSLOAD) -ram -b $(XESS_BOARD) $(SRAM_HEXFILE)
	$(XSLOAD) -fpga -b $(XESS_BOARD) $(DOWNLOAD_BIT)
```

3. main.C file that tests SRAM access (SRAM as an OPB peripheral)

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

#include "xintc_l.h"
#include "xuartlite_l.h"
#include "xparameters.h"

/* Defined in mymicroblaze/xparameters.h */
/* #define XPAR_VGA_BASEADDR 0xFEFF1000 */
/* #define XPAR_VGA_HIGHADDR 0xFEFF1fff */

#define FONT_OFFSET 0xA00

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

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

int uart_interrupt_count = 0;
char uart_character;

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

  Xuint8 incoming_character;

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

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

    uart_character = incoming_character;
    ++uart_interrupt_count;
  }

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

```
}

/* Write a text string on the display */
void put_string(char *string, int row, int column)
{
  char *p = &(CHAR(row, column));
  while (*p++ = *string++)
    ;
}

int main()
{
  volatile unsigned char *p;
  unsigned char v, vv;
  int i, j, k;
  print("Hello World!\r\n");

#if 0
  v = 0;
  for ( p = (unsigned char *) XPAR_VGA_BASEADDR ;
        p <= (unsigned char *) XPAR_VGA_HIGHADDR ;
        ++p ) {
#if 0
    print("writing to ");
    putnum((int) p);
    print(" value ");
    putnum(v);
    print("\r\n");
#endif
    *p = v;
    v += 1;
  }
  print("wrote to memory\r\n");

  v = 0;
  for ( p = (unsigned char *) XPAR_VGA_BASEADDR ;
        p <= (unsigned char *) XPAR_VGA_HIGHADDR ;
        ++p ) {
    vv = *p;
    if (vv != v) {
      print("read from ");
      putnum((int) p);
      print(" returned ");
      putnum(vv);
      print(" expected ");
      putnum(v);
      print("\r\n");
    }
```

```
    v += 1;
   }

   print("memory test completed. from ");
   putnum(XPAR_VGA_BASEADDR);
   print(" to ");
   putnum(XPAR_VGA_HIGHADDR);
   print("\r\n");
#endif


   /* Clear the screen */

   /*
   for ( p = &(CHAR(0,0)) ; p < &(CHAR(29,80)) ; ++p)
     *p = ' ';
    */

   /* Announce ourselves */

   put_string("Welcome to CSEE 4840 Lab 2", 15, 27);

   /* Enable UART interrupts and register uart_handler as the ISR */
   XIntc_RegisterHandler( XPAR_INTC_BASEADDR, XPAR_MYUART_DEVICE_ID,
                          (XInterruptHandler)uart_handler, (void *)0);
   XIntc_mEnableIntr( XPAR_INTC_BASEADDR, XPAR_MYUART_INTERRUPT_MASK);
   XIntc_mMasterEnable( XPAR_INTC_BASEADDR );
   XIntc_Out32(XPAR_INTC_BASEADDR + XIN_MER_OFFSET,
XIN_INT_MASTER_ENABLE_MASK);
   microblaze_enable_interrupts();
   XUartLite_mEnableIntr(XPAR_MYUART_BASEADDR);

   int addr;

   /* test SRAM access */
   print("testing SRAM...\r\n");
   j = 0;
   /*for ( i = 0 ; i < 215 ; i++ ) {
    addr = XPAR_SRAM_PERIPHERAL_BASEADDR + (i << 2);
     XIo_Out16(addr, j);
     j += 17;
     }*/

   j = 0;
   for ( i = 0 ; i <10 ; i++) {
     addr = XPAR_SRAM_PERIPHERAL_BASEADDR + (i << 2);
     k = XIo_In16(addr);
     //putnum(addr);
     //print(" got ");
```

```
    //putnum(k);
    //if ( k != j ) {
      putnum(addr);
      print(" got ");
      putnum(k);
      //print(" expected ");
      //putnum(j);
      print("\r\n");
      //}
    //print("\r\n");
    j += 17;
  }

  return 0;
}
```

4. SRAM OPB peripheral in VHDL (opb_sram.vhd).

```
------------------------------------------------------------------------------
--
-- OPB peripheral: SRAM controller
--
-- (modified from BRAM controller by Stephen A. Edwards)
-- Stephen A. Edwards
-- sedwards@cs.columbia.edu
--
------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity opb_sram is

  generic (
    C_OPB_AWIDTH : integer                  := 32;
    C_OPB_DWIDTH : integer                  := 32;
    C_BASEADDR   : std_logic_vector(0 to 31) := X"00000000";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"FFFFFFFF");

  port (
    OPB_Clk     : in  std_logic;
    OPB_Rst     : in  std_logic;
    OPB_ABus    : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE      : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus    : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW     : in  std_logic;
    OPB_select  : in  std_logic;
    OPB_seqAddr : in  std_logic;        -- Sequential Address
    Sln_DBus    : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck  : out std_logic;        -- (unused)
    Sln_retry   : out std_logic;        -- (unused)
    Sln_toutSup : out std_logic;        -- Timeout suppress
    Sln_xferAck : out std_logic;
    PB_D        : inout std_logic_vector(15 downto 0);
    PB_A        : out std_logic_vector(17 downto 0);
    PB_WE       : out std_logic;
    PB_OE       : out std_logic;
    PB_LB       : out std_logic;
    PB_UB       : out std_logic;
    RAM_CE      : out std_logic);       -- Transfer acknowledge
```

```
end opb_sram;

architecture Behavioral of opb_sram is

  constant RAM_AWIDTH : integer := 18;  -- Number of address lines on the RAM
  constant RAM_DWIDTH : integer := 16;  -- Number of data lines on the RAM

 -- component RAMB4_S8
 --   port (
 --     DO   : out std_logic_vector(RAM_DWIDTH-1 downto 0);
 --     ADDR : in  std_logic_vector(RAM_AWIDTH-1 downto 0);
  --    CLK  : in  std_logic;
---     DI   : in  std_logic_vector(RAM_DWIDTH-1 downto 0);
 --     EN   : in  std_logic;
--      RST  : in  std_logic;
--      WE   : in  std_logic);
 -- end component;

 component OBUF_F_24
    port (
      O : out std_logic;
      I : in std_logic);
  end component;

  component IOBUF_F_24
    port (
      O : out std_logic;
      IO : inout std_logic;
      I : in std_logic;
      T : in std_logic);
  end component;

  signal RNW : std_logic;
  --signal RAM_DI, RAM_DO : std_logic_vector(0 to RAM_DWIDTH-1);
  --signal ABus : std_logic_vector(0 to RAM_AWIDTH-1);
  signal chip_select : std_logic;
  signal output_enable : std_logic;
  signal tri_iob: std_logic;
  signal OE, WE, LB, UB : std_logic;
  signal saddin : std_logic_vector(0 to 17);
  signal sramout : std_logic_vector(0 to 15);
  signal sramin : std_logic_vector(0 to 15);
  signal iobuf : std_logic;

  -- Critical: Sln_xferAck is generated directly from state bit 0!

type opb_state is (Idle, Selected, Read, Xfer);
signal present_state, next_state : opb_state;
```

```
begin

 -- RAMBlock : RAMB4_S8
 -- port map (
 --   DO   => RAM_DO,
  --  ADDR => ABus,
  --  CLK  => OPB_Clk,
  --  DI   => RAM_DI,
  --  EN   => '1',
  --  RST  => RST,
  --  WE   => WE);

  address: for i in 0 to 17 generate
  OBUFBlock  : OBUF_F_24
    port map (
      O => PB_A(i),
      I  => saddin(i));
  end generate;

  data : for j in 0 to 15 generate
  IOBUFBlock : IOBUF_F_24
    port map (
      O => sramout(j),
      IO => PB_D(j),
      I => sramin(j),
      T => tri_iob);
  end generate;



  register_opb_inputs: process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      sramin <= (others => '0');
      saddin <= (others => '0');
      RNW <= '0';
    elsif OPB_Clk'event and OPB_Clk = '1' then
      sramin <= OPB_DBus(0 to RAM_DWIDTH-1);
      saddin <= OPB_ABus(C_OPB_AWIDTH-3-(RAM_AWIDTH-1) to C_OPB_AWIDTH-3);
      RNW <= OPB_RNW;
    end if;
  end process register_opb_inputs;

  register_opb_outputs: process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');

    elsif OPB_Clk'event and OPB_Clk = '1' then
```

```
    if output_enable = '1' then
       Sln_DBus(0 to RAM_DWIDTH-1) <= sramout;
    else
      Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
    end if;
   end if;
 end process register_opb_outputs;

 -- Unused outputs
 Sln_errAck  <= '0';
 Sln_retry   <= '0';
 Sln_toutSup <= '0';
 Sln_DBus(RAM_DWIDTH to C_OPB_DWIDTH-1) <= (others => '0');

 chip_select <=
   '1' when OPB_select = '1' and
      OPB_ABus(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) =
      C_BASEADDR(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) else
   '0';

 RAM_CE <= '0';
 PB_WE <= WE;
 PB_OE <= OE;
 PB_UB <= UB;
 PB_LB <= LB;
-- iobuf <= '0' when output_enable = '1' else '1';
--iobuf <= output_eable;
 -- Sequential part of the FSM
 fsm_seq : process(OPB_Clk, OPB_Rst)
 begin
   if OPB_Rst = '1' then
     present_state <= Idle;
   elsif OPB_Clk'event and OPB_Clk = '1' then
     present_state <= next_state;

   end if;
 end process fsm_seq;

 -- Combinational part of the FSM
 fsm_comb : process(OPB_Rst, present_state, chip_select, OPB_Select, RNW)
 begin                       -- Default values

   Sln_xferAck <= '0';
   next_state <= present_state;
   output_enable <= '0';
   tri_iob <= '1';
   WE <= '1';
   UB <= '0';
   LB <= '0';
```

```
    OE <= '1';

    case present_state is

      when Idle =>
        if chip_select = '1' then
          next_state <= Selected;
        end if;

      when Selected =>
        if OPB_Select = '1' then
          if RNW = '1' then           -- read
            OE <= '0';
            next_state <= Read;
          else
            -- write
            WE <= '0';
            tri_iob <= '0';
            next_state <= Xfer;
          end if;
        else
          next_state <= Idle;
        end if;

      when Read =>

        if OPB_Select = '1' then
          OE <= '0';
          output_enable <= '1';
          next_state <= Xfer;
        else
          next_state <= Idle;
        end if;

      -- State encoding is critical here: xfer must only be true here
      when Xfer =>
        Sln_xferAck <= '1';
        next_state <= Idle;

     end case;

end process fsm_comb;

end Behavioral;

-- Local Variables:
-- compile-command: "ghdl -a opb_sram.vhd"
-- End:
```

5. VGA OPB peripheral that interacts directly with the SRAM (opb_xsb_300e_vga.vhd)

```
--------------------------------------------------------------------------------
--
-- Text-mode VGA controller for the XESS-300E
--
-- Uses an OPB interface, e.g., for use with the Microblaze soft core
--
-- Stephen A. Edwards
-- sedwards@cs.columbia.edu
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity opb_xsb300e_vga is

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

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

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

    VGA_DBus      : out std_logic_vector (31 downto 0);
    VGA_errAck    : out std_logic;
    VGA_retry     : out std_logic;
    VGA_toutSup   : out std_logic;
    VGA_xferAck   : out std_logic;

    Pixel_Clock   : in std_logic;

    -- Video signals
    VIDOUT_CLK    : out std_logic;
```

```
    VIDOUT_RED     : out std_logic_vector(9 downto 0);
    VIDOUT_GREEN   : out std_logic_vector(9 downto 0);
    VIDOUT_BLUE    : out std_logic_vector(9 downto 0);
    VIDOUT_BLANK_N : out std_logic;
    VIDOUT_HSYNC_N : out std_logic;
    VIDOUT_VSYNC_N : out std_logic;

    -- SRAM signals
    Sln_DBus    : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck  : out std_logic;        -- (unused)
    Sln_retry   : out std_logic;        -- (unused)
    Sln_toutSup : out std_logic;        -- Timeout suppress
    Sln_xferAck : out std_logic;
    PB_D        : inout std_logic_vector(15 downto 0);
    PB_A        : out std_logic_vector(17 downto 0);
    PB_WE       : out std_logic;
    PB_OE       : out std_logic;
    PB_LB       : out std_logic;
    PB_UB       : out std_logic;
    RAM_CE      : out std_logic);       -- Transfer acknowledge

end opb_xsb300e_vga;

architecture Behavioral of opb_xsb300e_vga is

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

  -- Video parameters

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

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

  -- 512 X 8 dual-ported Xilinx block RAM
  component RAMB4_S8_S8
    port (
      DOA   : out std_logic_vector (7 downto 0);
      ADDRA : in std_logic_vector (8 downto 0);
      CLKA  : in std_logic;
      DIA   : in std_logic_vector (7 downto 0);
      ENA   : in std_logic;
```

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

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

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

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

-- Signals for the video controller
signal LoadNShift : std_logic;        -- Shift register control
signal FontData : std_logic_vector(7 downto 0);   -- Input to shift register
signal ShiftData : std_logic_vector(7 downto 0);  -- Shift register data
signal VideoData : std_logic;         -- Serial out ANDed with blanking
--signal RedData : std_logic_vector(7 downto 0); -- Red value from video memory
--signal GreenData : std_logic_vector(7 downto 0);--Grn value from video memory
--signal BlueData : std_logic_vector(7 downto 0);--Blue value from video memory


-- signals for the video decoder
signal decoded_SRAM_data : std_logic_vector(23 downto 0);
signal raw_SRAM_data : std_logic_vector(15 downto 0);
signal next_pixel_to_display : std_logic;

signal Hcount : std_logic_vector(9 downto 0);  -- Horizontal position (0-800)
signal Vcount : std_logic_vector(9 downto 0);  -- Vertical position (0-524)
signal HBLANK_N, VBLANK_N : std_logic;   -- Blanking signals
```

```
signal FontLoad, LoadChar : std_logic;  -- Font/Character RAM read triggers
signal FontAddr : std_logic_vector(10 downto 0);
signal CharRamPage : std_logic_vector(2 downto 0);
signal CharRamSelect_N : std_logic_vector(4 downto 0);
signal FontRamPage : std_logic_vector(1 downto 0);
signal FontRamSelect_N : std_logic_vector(2 downto 0);
signal CharAddr : std_logic_vector(11 downto 0);
signal CharColumn : std_logic_vector(9 downto 0);
signal CharRow : std_logic_vector(9 downto 0);
signal Column : std_logic_vector(6 downto 0); -- 0-79
signal Row : std_logic_vector(4 downto 0);   -- 0-29
signal EndOfLine, EndOfField : std_logic;

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

attribute INIT_00 : string;
attribute INIT_01 : string;
attribute INIT_02 : string;
attribute INIT_03 : string;
attribute INIT_04 : string;
attribute INIT_05 : string;
attribute INIT_06 : string;
attribute INIT_07 : string;
attribute INIT_08 : string;
attribute INIT_09 : string;
attribute INIT_0a : string;
attribute INIT_0b : string;
attribute INIT_0c : string;
attribute INIT_0d : string;
attribute INIT_0e : string;
attribute INIT_0f : string;

attribute INIT_00 of RAMB4_S8_S8_0 : label is
 "21646c726f57206f6c6c6548";

-- Standard 8x16 font taken from the Linux console font file "lat0-16.psfu"

attribute INIT_00 of RAMB4_S8_S8_5 : label is
 "0000000018180018181 83c3c3c180000000000000000000000000000000000";
attribute INIT_01 of RAMB4_S8_S8_5 : label is
 "000000006c6cfe6c6c6cfe6c6c0000000000000000000000000002466666600";
attribute INIT_02 of RAMB4_S8_S8_5 : label is
 "0000000086c66030180cc6c200000000000010107cd616167cd0d0d67c101000";
attribute INIT_03 of RAMB4_S8_S8_5 : label is
 "00000000000000000000003018181800000000076ccccccdc76386c6c380000";
attribute INIT_04 of RAMB4_S8_S8_5 : label is
 "0000000030180c0c0c0c0c0c18300000000000000c18303030303030180c0000";
attribute INIT_05 of RAMB4_S8_S8_5 : label is
```

"00000000000018187e1818000000000000000000000663cff3c660000000000";
attribute INIT_06 of RAMB4_S8_S8_5 : label is
"000000000000000fe0000000000000000000030181818000000000000000000";
attribute INIT_07 of RAMB4_S8_S8_5 : label is
"0000000000c06030180c06000000000000000000018180000000000000000000";
attribute INIT_08 of RAMB4_S8_S8_5 : label is
"000000007e1818181818187838180000000000007cc6e6e6d6d6cecec67c0000";
attribute INIT_09 of RAMB4_S8_S8_5 : label is
"000000007cc60606063c0606c67c000000000000fec6c06030180c06c67c0000";
attribute INIT_0a of RAMB4_S8_S8_5 : label is
"000000007cc6060606fcc0c0c0fe0000000000001e0c0c0cfecc6c3c1c0c0000";
attribute INIT_0b of RAMB4_S8_S8_5 : label is
"000000030303030180c0606c6fe0000000000007cc6c6c6c6fcc0c060380000";
attribute INIT_0c of RAMB4_S8_S8_5 : label is
"00000000780c0606067ec6c6c67c0000000000007cc6c6c6c67cc6c6c67c0000";
attribute INIT_0d of RAMB4_S8_S8_5 : label is
"000000003018180000001818000000000000000001818000000181800000000";
attribute INIT_0e of RAMB4_S8_S8_5 : label is
"00000000000000fe0000fe00000000000000000060c18306030180c06000000";
attribute INIT_0f of RAMB4_S8_S8_5 : label is
"0000000018180018181800cc6c67c000000000006030180c060c183060000000";
attribute INIT_00 of RAMB4_S8_S8_6 : label is
"00000000c6c6c6c6fec6c66c38100000000000007cc0dcdededec6c6c67c0000";
attribute INIT_01 of RAMB4_S8_S8_6 : label is
"000000003c66c2c0c0c0c0c2663c000000000000fc666666667c666666fc0000";
attribute INIT_02 of RAMB4_S8_S8_6 : label is
"00000000fe6662606878686266fe000000000000f86c6666666666666cf80000";
attribute INIT_03 of RAMB4_S8_S8_6 : label is
"000000003a66c6c6dec0c0c2663c000000000000f06060606878686266fe0000";
attribute INIT_04 of RAMB4_S8_S8_6 : label is
"000000003c18181818181818183c000000000000c6c6c6c6c6c6fec6c6c6c60000";
attribute INIT_05 of RAMB4_S8_S8_6 : label is
"00000000e666666c78786c6666e600000000000078cccccc0c0c0c0c0c0c1e0000";
attribute INIT_06 of RAMB4_S8_S8_6 : label is
"00000000c6c6c6c6c6c6d6fefeeec6000000000000fe66626060606060f00000";
attribute INIT_07 of RAMB4_S8_S8_6 : label is
"000000007cc6c6c6c6c6c6c6c6c67c000000000000c6c6c6c6cedefef6e6c60000";
attribute INIT_08 of RAMB4_S8_S8_6 : label is
"00000e0c7cded6c6c6c6c6c6c6c67c000000000000f0606060607c666666fc0000";
attribute INIT_09 of RAMB4_S8_S8_6 : label is
"000000007cc6c6060c3860c6c67c000000000000e66666666c7c666666fc0000";
attribute INIT_0a of RAMB4_S8_S8_6 : label is
"000000007cc6c6c6c6c6c6c6c6c6c60000000000000003c1818181818185a7e7e0000";
attribute INIT_0b of RAMB4_S8_S8_6 : label is
"000000006ceefed6d6d6c6c6c6c6c600000000000010386cc6c6c6c6c6c6c6c60000";
attribute INIT_0c of RAMB4_S8_S8_6 : label is
"000000003c181818183c6666666000000000000c6c66c7c38387c6cc6c60000";
attribute INIT_0d of RAMB4_S8_S8_6 : label is
"000000003c30303030303030303c000000000000fec6c26030180c86c6fe0000";

```
attribute INIT_0e of RAMB4_S8_S8_6 : label is
 "000000003c0c0c0c0c0c0c0c0c3c0000000000000060c183060c00000000000";
attribute INIT_0f of RAMB4_S8_S8_6 : label is
 "00ff0000000000000000000000000000000000000000000000000000c66c3810";
attribute INIT_00 of RAMB4_S8_S8_7 : label is
 "0000000076cccccc7c0c78000000000000000000000000000001830303000";
attribute INIT_01 of RAMB4_S8_S8_7 : label is
 "000000007cc6c0c0c0c67c0000000000000000007c666666666c786060e00000";
attribute INIT_02 of RAMB4_S8_S8_7 : label is
 "000000007cc6c0c0fec67c00000000000000000076cccccccc6c3c0c0c1c0000";
attribute INIT_03 of RAMB4_S8_S8_7 : label is
 "0078cc0c7ccccccccccc760000000000000000f060606060f060646c380000";
attribute INIT_04 of RAMB4_S8_S8_7 : label is
 "000000003c1818181818380018180000000000e666666666766c6060e00000";
attribute INIT_05 of RAMB4_S8_S8_7 : label is
 "00000000e6666c78786c666060e00000003c66660606060606060e0006060000";
attribute INIT_06 of RAMB4_S8_S8_7 : label is
 "00000000c6d6d6d6d6feec0000000000000000001834303030303030700000";
attribute INIT_07 of RAMB4_S8_S8_7 : label is
 "000000007cc6c6c6c6c67c0000000000000000006666666666666dc0000000000";
attribute INIT_08 of RAMB4_S8_S8_7 : label is
 "001e0c0c7ccccccccccc76000000000000f060607c6666666666dc0000000000";
attribute INIT_09 of RAMB4_S8_S8_7 : label is
 "000000007cc60c3860c67c00000000000000000f06060606676dc0000000000";
attribute INIT_0a of RAMB4_S8_S8_7 : label is
 "0000000076ccccccccccccc000000000000000001c3630303030fc3030100000";
attribute INIT_0b of RAMB4_S8_S8_7 : label is
 "000000006cfed6d6d6c6c60000000000000000000183c666666666000000000";
attribute INIT_0c of RAMB4_S8_S8_7 : label is
 "00f80c067ec6c6c6c6c6c600000000000000000c66c3838386cc60000000000";
attribute INIT_0d of RAMB4_S8_S8_7 : label is
 "000000000e18181818701818180e000000000000fec6603018ccfe0000000000";
attribute INIT_0e of RAMB4_S8_S8_7 : label is
 "00000000701818181806e1818187000000000000018181818181818181800000";
attribute INIT_0f of RAMB4_S8_S8_7 : label is
 "000000003c1818183c66666666006600000000000000000000000000dc760000";


----------------------------------------------------------------------
--BEGIN SRAM STUFF
----------------------------------------------------------------------

constant RAM_AWIDTH : integer := 18;  -- Number of address lines on the RAM
constant RAM_DWIDTH : integer := 16;  -- Number of data lines on the RAM

-- component RAMB4_S8
--   port (
--     DO   : out std_logic_vector(RAM_DWIDTH-1 downto 0);
--     ADDR : in  std_logic_vector(RAM_AWIDTH-1 downto 0);
```

```
  --    CLK  : in  std_logic;
---     DI   : in  std_logic_vector(RAM_DWIDTH-1 downto 0);
 --     EN   : in  std_logic;
--      RST  : in  std_logic;
--      WE   : in  std_logic);
 -- end component;

  component OBUF_F_24
    port (
      O : out std_logic;
      I : in std_logic);
  end component;

  component IOBUF_F_24
    port (
      O : out std_logic;
      IO : inout std_logic;
      I : in std_logic;
      T : in std_logic);
  end component;

  signal sram_RNW : std_logic;
  --signal RAM_DI, RAM_DO : std_logic_vector(0 to RAM_DWIDTH-1);
  --signal ABus : std_logic_vector(0 to RAM_AWIDTH-1);
  signal chip_select : std_logic;
  signal output_enable : std_logic;
  signal tri_iob: std_logic;
  signal OE, sram_WE, LB, UB : std_logic;
  signal saddin : std_logic_vector(0 to 17);
  signal sramout : std_logic_vector(0 to 15);
  signal sramin : std_logic_vector(0 to 15);
  signal iobuf : std_logic;

  signal SRAM_START_ADDRESS : std_logic_vector(0 to 17);

  -- Critical: Sln_xferAck is generated directly from state bit 0!

  type opb_state is (Idle, Selected, Read, Xfer);
  signal present_state, next_state : opb_state;


begin  -- Behavioral

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

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

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

RAMB4_S8_S8_0 : RAMB4_S8_S8
  port map (
    DOA   => DOUT0,
    ADDRA => ABus(8 downto 0),
    CLKA  => OPB_Clk,
    DIA   => DBus(7 downto 0),
    ENA   => '1',
    RSTA  => RST(0),
    WEA   => WE(0),
    DOB   => DOUTB0,
    ADDRB => CharAddr(8 downto 0),
    CLKB  => Pixel_Clock,
    DIB   => X"00",
    ENB   => '1',
    RSTB  => CharRamSelect_N(0),
    WEB   => '0');

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

  RAMB4_S8_S8_2 : RAMB4_S8_S8
  port map (
    DOA   => DOUT2,
    ADDRA => ABus(8 downto 0),
    CLKA  => OPB_Clk,
    DIA   => DBus(7 downto 0),
    ENA   => '1',
```

```
  RSTA  => RST(2),
  WEA   => WE(2),
  DOB   => DOUTB2,
  ADDRB => CharAddr(8 downto 0),
  CLKB  => Pixel_Clock,
  DIB   => X"00",
  ENB   => '1',
  RSTB  => CharRamSelect_N(2),
  WEB   => '0');

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

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

RAMB4_S8_S8_5 : RAMB4_S8_S8
port map (
  DOA   => DOUT5,
  ADDRA => ABus(8 downto 0),
  CLKA  => OPB_Clk,
```

```
    DIA   => DBus(7 downto 0),
    ENA   => '1',
    RSTA  => RST(5),
    WEA   => WE(5),
    DOB   => DOUTB5,
    ADDRB => FontAddr(8 downto 0),
    CLKB  => Pixel_Clock,
    DIB   => X"00",
    ENB   => '1',
    RSTB  => FontRamSelect_N(0),
    WEB   => '0');

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

  RAMB4_S8_S8_7 : RAMB4_S8_S8
  port map (
    DOA   => DOUT7,
    ADDRA => ABus(8 downto 0),
    CLKA  => OPB_Clk,
    DIA   => DBus(7 downto 0),
    ENA   => '1',
    RSTA  => RST(7),
    WEA   => WE(7),
    DOB   => DOUTB7,
    ADDRB => FontAddr(8 downto 0),
    CLKB  => Pixel_Clock,
    DIB   => X"00",
    ENB   => '1',
    RSTB  => FontRamSelect_N(2),
    WEB   => '0');

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

```
--
-------------------------------------------------------------------------------

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

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

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

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

RamPageAddress <= ABus(11 downto 9);

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

MemCycleFSM : process(OPB_Clk, OPB_Rst)
begin
```

```
  if OPB_Rst = '1' then
    MemCycle1 <= '0';
    MemCycle2 <= '0';
  elsif OPB_Clk'event and OPB_Clk = '1' then
    MemCycle2 <= MemCycle1;
    MemCycle1 <= ChipSelect;
  end if;
end process MemCycleFSM;

VGA_xferAck <= MemCycle2;

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

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

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

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

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

VGA_DBus <= DBus_out;

------------------------------------------------------------------------------
--
-- video decoder
--
------------------------------------------------------------------------------

SRAM_START_ADDRESS <= (others => '0');     -- hardwired for now

SRAM_address : process(Hcount)  -- assert address to SRAM
begin
-- if(hcount == 0)   ? restarting screendraw
     PB_A <= SRAM_START_ADDRESS;   -- plus whatever...
end process SRAM_address;
```

```
read_SRAM : process(sln_xferAck)
begin
  raw_SRAM_data <= PB_D;
end process read_SRAM;

video_decode : process(raw_SRAM_data)
begin
  --decoded_SRAM_data <= (some voodoo) <= raw_SRAM_data; -- decode the raw data
  decoded_SRAM_data(15 downto 0) <= raw_SRAM_data; -- decode the raw data
end process video_decode;

update_pixel : process(Pixel_Clock)
begin
  if Pixel_Clock'event and Pixel_Clock = '1' then
    next_pixel_to_display <= decoded_SRAM_data(0);
  end if;
end process update_pixel;

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

-- Horizontal and vertical counters

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

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

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

```
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

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

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

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

-- The -1 correction doesn't appear here to correct for the
-- registered video signal outputs.

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

VSyncGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_VSYNC_N <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine ='1' then
      if EndOfField = '1' then
        VIDOUT_VSYNC_N <= '0';
```

```
      elsif VCount = VSYNC - 1 then
        VIDOUT_VSYNC_N <= '1';
      end if;
    end if;
  end if;
end process VSyncGen;

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

-- RAM read triggers and shift register control

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

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

-- Column + Row * 80
CharAddr <= Column +
          ("0" & Row(4 downto 0) & "000000") +
          ("000" & Row(4 downto 0) & "0000");

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

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

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

  FontData <= DOUTB5 or DOUTB6 or DOUTB7;

  -- Shift register

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

  VideoData <= ShiftData(7);

  -- Registered video signals going to the video DAC

  VideoOut: process (Pixel_Clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      VIDOUT_BLANK_N <= '0';
      VIDOUT_RED   <= "0000000000";
      VIDOUT_BLUE  <= "0000000000";
      VIDOUT_GREEN <= "0000000000";
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
      VIDOUT_BLANK_N <= VBLANK_N and HBLANK_N;
    --if VideoData = '1' then
    if next_pixel_to_display = '1' then
--      if (Vcount(6) xor Hcount(6)) = '1' then
```

```
      VIDOUT_RED   <= "1111111111";
      VIDOUT_GREEN <= "1111111111";--GreenData & "00";
      VIDOUT_BLUE  <= "1111111111";--BlueData & "00";
      -- display next_pixel_to_display
      --VIDOUT_GREEN <= "1111111111";
      --VIDOUT_BLUE  <= "1111111111";
    else
      VIDOUT_RED   <= "0000000000";
      VIDOUT_GREEN <= "0000000000";
      VIDOUT_BLUE  <= "0000000000";
    end if;
  end if;
end process VideoOut;

VIDOUT_CLK <= Pixel_Clock;


------------------------------------------------------------------------------
--BEGIN SRAM STUFF
------------------------------------------------------------------------------

address: for i in 0 to 17 generate
OBUFBlock  : OBUF_F_24
  port map (
    O => PB_A(i),
    I  => saddin(i));
end generate;

data : for j in 0 to 15 generate
IOBUFBlock : IOBUF_F_24
  port map (
    O => sramout(j),
    IO => PB_D(j),
    I => sramin(j),
    T => tri_iob);
end generate;



register_opb_inputs: process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    sramin <= (others => '0');
    saddin <= (others => '0');
    sram_RNW <= '0';
  elsif OPB_Clk'event and OPB_Clk = '1' then
    sramin <= OPB_DBus( RAM_DWIDTH-1 downto 0);
    saddin <= OPB_ABus(C_OPB_AWIDTH-3 downto C_OPB_AWIDTH-3-(RAM_AWIDTH-1));
    sram_RNW <= OPB_RNW;
```

```
  end if;
end process register_opb_inputs;

register_opb_outputs: process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');

  elsif OPB_Clk'event and OPB_Clk = '1' then
   if output_enable = '1' then
      Sln_DBus(0 to RAM_DWIDTH-1) <= sramout;
   else
     Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
   end if;
  end if;
end process register_opb_outputs;

-- Unused outputs
Sln_errAck  <= '0';
Sln_retry   <= '0';
Sln_toutSup <= '0';
Sln_DBus(RAM_DWIDTH to C_OPB_DWIDTH-1) <= (others => '0');

chip_select <=
  '1' when OPB_select = '1' and
      OPB_ABus(C_OPB_AWIDTH-3-RAM_AWIDTH downto 0) =
      C_BASEADDR(C_OPB_AWIDTH-3-RAM_AWIDTH downto 0) else
  '0';

RAM_CE <= '0';
PB_WE <= sram_WE;
PB_OE <= OE;
PB_UB <= UB;
PB_LB <= LB;
-- iobuf <= '0' when output_enable = '1' else '1';
--iobuf <= output_eable;
 -- Sequential part of the FSM
 fsm_seq : process(OPB_Clk, OPB_Rst)
 begin
   if OPB_Rst = '1' then
     present_state <= Idle;
   elsif OPB_Clk'event and OPB_Clk = '1' then
     present_state <= next_state;

   end if;
 end process fsm_seq;

 -- Combinational part of the FSM
 fsm_comb : process(OPB_Rst, present_state, chip_select, OPB_Select, sram_RNW)
```

```
begin                    -- Default values

  Sln_xferAck <= '0';
  next_state <= present_state;
  output_enable <= '0';
  tri_iob <= '1';
  sram_WE <= '1';
  UB <= '0';
  LB <= '0';
  OE <= '1';

  case present_state is

     when Idle =>
       if chip_select = '1' then
         next_state <= Selected;
       end if;

    when Selected =>
       if OPB_Select = '1' then
         if sram_RNW = '1' then          -- read
           OE <= '0';
           next_state <= Read;
         else
           -- write
           sram_WE <= '0';
           tri_iob <= '0';
           next_state <= Xfer;
         end if;
       else
         next_state <= Idle;
       end if;

     when Read =>

       if OPB_Select = '1' then
         OE <= '0';
         output_enable <= '1';
         next_state <= Xfer;
       else
         next_state <= Idle;
       end if;

     -- State encoding is critical here: xfer must only be true here
     when Xfer =>
       Sln_xferAck <= '1';
       next_state <= Idle;

    end case;
```

```
end process fsm_comb;


end Behavioral;

-- Only for syntax debugging:
--
-- Local Variables:
-- compile-command: "ghdl -a --ieee=synopsys opb_xsb300e_vga.vhd"
-- End:
```

6. Perl script that converts a PPM file to a compressed image using our run-length encoding (ppm2lkr.pl)

```
#!/usr/bin/perl -w

# convert ppm to LKR

$colors = 0;
$i=0;
foreach (@ARGV){ # go through the command line arguments
   if(m/-o/i){ $ofile = $ARGV[$i+1]; }
   if(m/-i/i){ $ifile = $ARGV[$i+1]; }
   $i++;
}
($ifile && $ofile) ||    # both must be defined
   die "Usage: ppm2lkr.pl -i infile -o outfile\n";
open(INPUT, "$ifile")
   || die "Couldn't open $ifile for reading.\n";
open(OUTPUT, ">$ofile")
   || die "Couldn't open $ofile for writing.\n";

$all = join("",<INPUT>);

close(INPUT);

@data = split(/\s+/, $all);   #split into non-whitespace tokens
$format = shift @data;
$width = shift @data;
$height = shift @data;
$maxval = shift @data;
$maxval <= 256 ||
   die "PPM input file uses a maxval greater than 256 and is incompatible.\n";
($lr, $lg, $lb)=(-1,-1,-1); # guaranteed never to matter
$last = "";
$last_ct = 0;
$N_FLAG = 0;
for $i (1 .. ($height*$width)) {
   ($r,$g,$b, @data) = @data;
   $RGB = "$r:$g:$b";
   unless($colors{$RGB}){ #unless we've seen it before...
         $colors{$RGB} = ++$colors;
           $colors <= 16
              || die "PPM input file uses more than 16 distinct colors!\n";
   }

  #"$r:$g:$b"
   if($i > 1){ #past the boundary condition
         if($RGB eq $last){
```

```
                #in a run
            $last_ct++;
            if($last_ct == 16){
                    #we have maxed a run
                push(@working, 16);
                push(@working, $RGB);
                $last_ct=0;
            } # if (last_ct == 16)
        }else{
                #just hit a color boundary
                # pack the last one, start a new count
            push(@working, $last_ct);
            push(@working, $last);
            $last_ct=1; #start counting on this new color
        } # if we hit a color boundary
    }else{ #very first pixel
        $last_ct = 1; #nothing else special needs to happen
    } #end internal processing loop
    $last = $RGB;  # we need to know for next time
} # end for all pixels
push (@working, $last_ct);  #the last one! just push it!
push (@working, $last);

#pack the color table into the file
%codes = reverse(%colors); #1-to-1, this is ok!
for $i (0 .. 15){
    if ($codes{$i} && $codes{$i} =~ m/^(\d+):(\d+):(\d+)$/) {
        $buf=pack("H3", $1, $2, $3);
    }
    else{
        $buf=pack("H3", 0, 0, 0);
    }
    print OUTPUT $buf;
}

#we're done dumping the color table
while(@working){
    # range of 4-bits covers 0-15, whereas values from 1-16
    $run = shift(@working)-1;
    $color = $colors{shift(@working)};
    $buf = pack ("H2", $run, $color);
    print OUTPUT $buf;
}
```

7. Image decoder component in VHDL (decoder.vhdl)

```
----------------------------------------------------------------------------
--
-- LKR hardware decoder unit for the Leo-Kat-Russ image encoding format
--
-- Leo Gertsenshteyn
-- lpg2006@columbia.edu
--
----------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity LKR_decoder is
  generic (
    AWIDTH : natural := 20);
  port (
    OPB_Clk        : in std_logic;
    Pixel_Clock    : in std_logic;
    xferAck    : in std_logic;
    SRAM_data   : in std_logic_vector(15 downto 0);

    Go, Reset   : in std_logic;
    Base_addr   : in std_logic_vector(AWIDTH-1 downto 0);
    SRAM_addr   : out std_logic_vector(AWIDTH-1 downto 0);
    SRAM_CE     : out std_logic;

    -- Video signals
    VIDOUT_RED     : out std_logic_vector(9 downto 0);
    VIDOUT_GREEN   : out std_logic_vector(9 downto 0);
    VIDOUT_BLUE    : out std_logic_vector(9 downto 0));

end LKR_decoder;



architecture Behavioral of LKR_decoder is

  type color is array (0 to 2) of std_logic_vector(7 downto 0);
  type state is (init, start, reqRed, stoRed, reqGrn, stoGrn,
                 reqBlu, stoBlu, idling, decode_pix, load_pix);
  type color_table is array (0 to 15) of color;

  signal to_SRAM_addr: std_logic_vector(AWIDTH-1 downto 0);
  signal color_tbl : color_table;
  signal counting : integer := 0;
```

```
  signal index: integer := 0;
  signal cstate, nstate : state := idling;


  signal countdown: std_logic_vector(3 downto 0);
  signal SRAM_current: std_logic_vector(7 downto 0);
  signal cdwn_clr,cdwn_enable, cdwn_ld, cdwn_up: std_logic;
  signal d_addr_clr, d_addr_enable, d_addr_ld: std_logic;

  component  UD_COUNTER1
    generic (WIDTH: natural := 4);
    port ( CLK,CLEAR,LOAD,CE,UP : in std_logic;    -- INPUT and OUTPUT declaration
          DIN : in std_logic_vector(WIDTH-1 downto 0);
          Q : out std_logic_vector(WIDTH-1 downto 0));
  end component;


begin  -- Behavioral

d_addr : UD_COUNTER1 generic map(WIDTH => AWIDTH)
  port map (
    CLK=>OPB_Clk,
    CLEAR=>d_addr_clr,
    LOAD=>d_addr_ld,
    CE=>d_addr_enable,
    UP=>'1',
    DIN=> Base_addr,
    Q=> to_SRAM_addr
  );

down_ctr : UD_COUNTER1
  port map (
    CLK=>OPB_Clk,
    CLEAR=>cdwn_clr,
    LOAD=>cdwn_ld,
    CE=>cdwn_enable,
    UP=>cdwn_up,
    DIN=>(others=>'0'),
    Q=> countdown
  );

SRAM_addr <= to_S;


SYNC_PROC: process (OPB_Clk, Reset)
begin
  if (RESET='1') then
    CSTATE <= init;
  elsif (OPB_Clk'event and OPB_Clk = '1') then
```

```
    CSTATE <= NSTATE;
  end if;
end process;

COMB_PROC: process (CSTATE, Go, xferAck)
begin
  NSTATE<= init;
  cdwn_clr <= '0';
  d_addr_clr <= '0';
  cdwn_up <= '0'; -- this is not in any way helpful except when invalid state
  d_addr_ld <= '0';
  SRAM_CE <= '0';
  index <= CONV_INTEGER(to_SRAM_addr(7 downto 0));

  case CSTATE is

    --init state
    when init =>
      if Go='1' then
        NSTATE <= start;
      end if;

    --starting up state
    when start =>
      d_addr_ld<='1';
      d_addr_enable <='0';
      NSTATE <= reqRed;

    when reqRed =>
      SRAM_CE <= '1';
      if xferAck = '1' then
        NSTATE <= stoRed;
      end if;
    when stoRed =>
      color_tbl(index)(0) <= SRAM_data(15 downto 8) when to_SRAM_addr(0) = '0' else
                             SRAM_data(7 downto 0);
      d_addr_enable <= '1';
      NSTATE <= reqGrn;

    when reqGrn =>
      SRAM_CE <= '1';
      if xferAck = '1' then
        NSTATE <= stoGrn;
      end if;
    when stoGrn =>
      color_tbl(index)(1) <= SRAM_data(15 downto 8) when to_SRAM_addr(0) = '0' else
                             SRAM_data(7 downto 0);
      d_addr_enable <= '1';
      NSTATE <= reqBlu;
```

```
    when reqBlu =>
      SRAM_CE <= '1';
      if xferAck = '1' then
        NSTATE <= stoBlu;
      end if;
    when stoBlu =>
      color_tbl(index)(2) <= SRAM_data(15 downto 8) when to_SRAM_addr(0) = '0' else
                             SRAM_data(7 downto 0);
      d_addr_enable <= '1';
      if (to_SRAM_addr(7 downto 0) = "110000") then
        NSTATE <= idling;
      else
        NSTATE <= reqRed;
      end if;

    when idling =>
      NSTATE <= load_pix;

    when load_pix =>
      SRAM_CE <= '1';
      NSTATE <= idling when Go = '0' else
           decode_pix when xferAck = '1' else
           load_pix;

    when decode_pix =>
      SRAM_current <= SRAM_data(15 downto 8) when to_SRAM_addr(0) = '0' else
                      SRAM_data(7 downto 0);
      VIDOUT_RED <= "00" & color_tbl(CONV_INTEGER(SRAM_data(3 downto 0)))(0);
      VIDOUT_GREEN <= "00" & color_tbl(CONV_INTEGER(SRAM_data(3 downto 0)))(1);
      VIDOUT_BLUE <= "00" & color_tbl(CONV_INTEGER(SRAM_data(3 downto 0)))(2);
      NSTATE <= idling when Go = '0' else
           load_pix;

  end case;
end process;


end Behavioral;

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

--*************************************************************************
--*                                                                       *
--*      Synchronous Binary 4bits Up-Down Counter without Carry/Borrow        *
--*                                        Device : XC9536-PC44   *
--*                                        Author : Seiichi Inoue *
--*************************************************************************
```

```
library ieee;                          -- Library declaration
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity UD_COUNTER1 is
  generic( WIDTH : natural := 4);
  port ( CLK,CLEAR,LOAD,CE,UP : in std_logic;    -- INPUT and OUTPUT declaration
         DIN : in std_logic_vector(WIDTH-1 downto 0);
         Q : out std_logic_vector(WIDTH-1 downto 0));
end UD_COUNTER1;

architecture UD_COUNTER_ARCH of UD_COUNTER1 is
signal Q_IN : std_logic_vector(WIDTH-1 downto 0);      -- Internal counter signal
begin
  Q <= Q_IN;                           -- Set output
  process( CLEAR, CLK, LOAD, CE, UP ) begin
    if CLEAR='1' then                  -- CLEAR = ON ?
      Q_IN <= (others => '0');                 -- Yes. Counter clear
    elsif CLK='1' and CLK'event then           -- Clock in ?
      if LOAD='1' then                 -- Yes. LOAD = ON ?
        Q_IN <= DIN;                   -- Set Input to Output
      else                             -- LOAD = OFF
        if CE='1' then                 -- Count Enable ?
          if UP='1' then               -- Yes. Up count ?
            Q_IN <= Q_IN + '1';              -- Yes. Count-up
          else                         -- Not count-up
            Q_IN <= Q_IN - '1';              -- Count-down
          end if;
        end if;                        -- Not CE = 1
      end if;
    end if;
  end process;
end UD_COUNTER_ARC;
```