LCD FRAMEBUFFER.

(LABORATORY IV)

4/26/2004

Sergey Averchenkov

CS345, Queens College, N.Y.

INTRODUCTION:

This laboratory dealt with different ways in which an image can be produces on the LCD screen of the RC200e device. One way to create an image is to use PAL macros to check current scan position and decide (within a single clock cycle) the color of the current pixel to be drawn. This is probably the simplest way to produce simple shapes such as rectangles, but it is rather difficult to create more complex shapes that require some calculation before the color of the pixel can be decided. This issue can be solved by using a framebuffer. Framebuffer is essentially a section of RAM that holds a copy of an image to be drawn. All rendering calculations are performed in parallel with drawing, so color values are calculated before the actual rendering cycle allowing for greater flexibility in creating an image.

.

Project Specifications:

There are four projects in this laboratory. The first one will create an image "on the fly" using PAL macros to tell the code which pixel is about to be drawn. The second one will examine the values of some of the parameters that will affect the next two projects. The third one will create an image on the fly, but will perform its own synchronization with the video refresh cycle. The last project will create an image by reading pixel values from RAM in real time, thus implementing a classic Framebuffer design.

METHOD:

Project 1 – LCD Seven Segment Display:

A workspace named "Laboratory V" and a project named "SevenSeg Display" were created. Runtime configurations were set as usual for RC200E and SIM. A program was created to read from the keyboard and write an ASCII code in hex to the seven segment display on LCD screen.

One thread reads characters and assigns values to two seven-bit variables that tell which segments to illuminate. This is done by table lookup from a table that contains bit patters that indicate which of the seven segments must be on. Second thread continuously determines the coordinates being refreshed on the LCD and decides what color to display there. It uses macros PalVideoOutGetX(handle) and PalVideoOutGetY(handle) that return the current X and Y coordinate values, and decided what color to draw (using PalVideoOutWrite(handle, pixel)) on the next clock cycle.

Project 2 – Display Video Parameter Values:

Second project called Parameters was created and configured for EDIF and for Simulation. pal_console.hch header file was included in the source code and the pal_console.hcl library file was added to the Linker list. Handel-C program called parameters.hcc that displays the following parameters on the console at run time was created:

- The number of visible pixels per scan line, determined at compile time.
- The number of visible scan lines, determined at compile time.
- The number of visible pixels per scan line, determined at run time.
- The number of visible scan lines, determined at run time.
- The total number of pixels per scan line.
- The total number of scan lines.
- The width (number of bits) needed for a variable that holds the X coordinate of a visible pixel.
- The width needed for a variable that holds the Y coordinate of a visible pixel.
- The width needed for a variable that ranges over all X coordinates on a scan line.
- The width needed for a variable that ranges over all Y coordinates on the display.
- The width needed for a variable that holds a RAM memory address.
- The width needed for a variable that holds a word of RAM data.

Project 3 – Draw a Test Pattern by Synchronizing with HBlank:

Third project named Unbuffered, and the Handel-C source file called unbuffered.hcc were created and added to the workspace. This program uses the PalVideoOutGetHBlank(handle) to synchronize with the beginning of each scan line. It draws a white pixel in the first column of each scan line, and draws vertical bars of alternating colors across the remainder of each line.

Project 4 – Draw an Image From a Framebuffer:

This part of the lab was based on using one of the RC200E's PL1 memory banks as a framebuffer to draw an image on the screen. First, we decided how to map pixel coordinates to memory addresses, and generated a test pattern into RAM. The test pattern consists of a one-pixel wide white border along all four edges of the visible part of the display. Inside the border, vertical bars of alternating colors were drawn.

The code was written to continuously read from the framebuffer and draw the pixels to their proper locations on the screen. Because of the delays involved in reading from the memory and writing to the screen, we had to start processing each scan line during the end of the HBlank period of the previous scan line.

Appendix A: Project 1 – LCD Seven Segment Display Source Code.

```
// sevenseg.hcc
/*****************************************************************
* Project   :    Simulated sevenSeg display using LCD           *
* Date      :    26 Apr 2004                                    *
* File      :    sevenseg.hcc                                   *
* Author    :    Sergey Averchenkov                             *
*****************************************************************
* Desc      :    One thread reads characters and assigns values*
*           :    to two seven-bit variables that tell which    *
*           :    segments to illuminate. This is done by table *
*           :    lookup from a table that contains bit patters *
*           :    that indicate which of the seven segments must*
*           :    be on.                                        *
*           :    Second thread continuously determines the     *
*           :    coordinates being refreshed on the LCD and    *
*           :    decides what color to display there. It uses  *
*           :    macros PalVideoOutGetX( handle ) and          *
*           :    PalVideoOutGetY( handle ) that return the     *
*           :    current X and Y coordinate values, and decided*
*           :    what color to draw (using PalVideoOutWrite(   *
*           :    handle, pixel ) ) on the next clock cycle.    *
*****************************************************************/

#if (defined USE_RC200 || USE_RC200E)
  #define PAL_TARGET_CLOCK_RATE 25175000
#else
  #define PAL_ACTUAL_CLOCK_RATE 25175000
  set clock = external "P1"
    with
    {
      extlib  = "DKSync.dll",
      extinst = "1000",       // Period of 1MHz simulated clock
      extfunc = "DKSyncGetSet"
    };
#endif

#include <pal_master.hch>
#include <pal_keyboard.hch>
#include <stdlib.hch>

static unsigned 7 charMap[] = {   0x7E, // 0
                                  0x18, // 1
                                  0x37, // 2
                                  0x3D, // 3
                                  0x59, // 4
                                  0x6D, // 5
                                  0x6F, // 6
                                  0x38, // 7
                                  0x7F, // 8
                                  0x7D, // 9
                                  0x7B, // A
                                  0x7F, // B
                                  0x66, // C
                                  0x7E, // D
                                  0x67, // E
                                  0x63  // F
                              };
```

```
static macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;

static macro expr VideoOut = PalVideoOutOptimalCT (PAL_ACTUAL_CLOCK_RATE);
static macro proc KeyboardRun(KeyboardPtr);
static macro proc VideoRun(VideoPtr);

static macro expr testSevenSegLeft(sevenSegLeftX, sevenSegY,
   scanX, scanY, valueLeft);
static macro expr testSevenSegRight(sevenSegRightX, sevenSegY,
   scanX, scanY, valueRight);
static macro expr testSevenSeg0(beginX, beginY, currX, currY, value);
static macro expr testSevenSeg1(beginX, beginY, currX, currY, value);
static macro expr testSevenSeg2(beginX, beginY, currX, currY, value);
static macro expr testSevenSeg3(beginX, beginY, currX, currY, value);
static macro expr testSevenSeg4(beginX, beginY, currX, currY, value);
static macro expr testSevenSeg5(beginX, beginY, currX, currY, value);
static macro expr testSevenSeg6(beginX, beginY, currX, currY, value);
static macro expr ScanX = PalVideoOutGetX (VideoOut);
static macro expr ScanY = PalVideoOutGetY (VideoOut);
static macro expr MaxX = PalVideoOutGetVisibleX (VideoOut, ClockRate);
static macro expr MaxY = PalVideoOutGetVisibleY (VideoOut);

/*******************************************************
 *            Seven segment configuration:             *
 *******************************************************/
      // Horizontal spacing between adjacent segments
      static macro expr segmentSpacing = 15;
      // Width of the horizontal segment
      static macro expr horizSegmentWidth = 75;
      // Height of the horizontal segment
      static macro expr horizSegmentHeight = 15;
      // Width of the vertical segment
      static macro expr vertSegmentWidth = 15;
      // Height of the vertical segment
      static macro expr vertSegmentHeight = 70;
/*******************************************************
 *******************************************************/

static macro expr SevenSegLeftX  = 1 * (MaxX / 4) - (horizSegmentWidth + 2
    * vertSegmentWidth + 2 * segmentSpacing) / 2;
static macro expr SevenSegRightX = 3 * (MaxX / 4) - (horizSegmentWidth + 2
    * vertSegmentWidth + 2 * segmentSpacing) / 2;
static macro expr SevenSegY = (MaxY / 2) - (2 * vertSegmentHeight + 3
    * horizSegmentHeight + 4 * segmentSpacing) / 2;

static unsigned 8 displayChar = 0;

// main()
// -------------------------------------------------------------
/*
 *    Main entry of the program. Runs all threads.
 */
void main (void)
{
  PalKeyboard *KeyboardPtr;

  PalVersionRequire(1, 0);
  PalVideoOutRequire(1);
```

```
  PalPS2PortRequire(2);

  par
  {
    PalVideoOutRun(VideoOut, ClockRate);
    PalKeyboardRun(&KeyboardPtr, PalPS2PortCT(1), ClockRate);
    seq
    {
      par
      {
        PalVideoOutEnable(VideoOut);
        PalKeyboardEnable(KeyboardPtr);
      }
      par
      {
        KeyboardRun(KeyboardPtr);
        VideoRun(VideoOut);
      }
    }
  }
}
// -------------------------------------------------------------


// -------------------------------------------------------------
// VideoRun()
// -------------------------------------------------------------
/*
 *    This thread outputs the seven segment display to the
 *    LCD screen. This method does not return.
 */
macro proc VideoRun(VideoOut)
{
  while(1)
  {
    if(testSevenSegLeft (SevenSegLeftX,  SevenSegY, ScanX,
         ScanY, charMap[displayChar[7:4]]) ||
       testSevenSegRight(SevenSegRightX, SevenSegY, ScanX,
         ScanY, charMap[displayChar[3:0]]))
    {
      PalVideoOutWrite (VideoOut, 0xFF0000);
    }
    else
    {
      PalVideoOutWrite (VideoOut, 0x000000);
    }
  }
}
// -------------------------------------------------------------


// -------------------------------------------------------------
// KeyboardRun()
// -------------------------------------------------------------
/*
 *    This thread reads the keyboard and updates the displayChar.
 *    This method does not return.
 */
macro proc KeyboardRun(KeyboardPtr)
```

```
{
  unsigned temp;
  while(1)
  {
    PalKeyboardReadASCII(KeyboardPtr, &temp);
    if(temp != 0)
    {
      displayChar = temp;
    }
  }
}
// -------------------------------------------------------------


macro expr testSevenSegLeft(sevenSegLeftX, sevenSegY, scanX,
  scanY, valueLeft) =
(
  (testSevenSeg0(sevenSegLeftX, sevenSegY, scanX, ScanY, valueLeft[6]) ||
   testSevenSeg1(sevenSegLeftX, sevenSegY, scanX, ScanY, valueLeft[5]) ||
   testSevenSeg2(sevenSegLeftX, sevenSegY, scanX, ScanY, valueLeft[4]) ||
   testSevenSeg3(sevenSegLeftX, sevenSegY, scanX, ScanY, valueLeft[3]) ||
   testSevenSeg4(sevenSegLeftX, sevenSegY, scanX, ScanY, valueLeft[2]) ||
   testSevenSeg5(sevenSegLeftX, sevenSegY, scanX, ScanY, valueLeft[1]) ||
   testSevenSeg6(sevenSegLeftX, sevenSegY, scanX, ScanY, valueLeft[0]))
);
macro expr testSevenSegRight(sevenSegRightX, sevenSegY, scanX,
  scanY, valueRight) =
(
  (testSevenSeg0(sevenSegRightX, sevenSegY, scanX, ScanY, valueRight[6]) ||
   testSevenSeg1(sevenSegRightX, sevenSegY, scanX, ScanY, valueRight[5]) ||
   testSevenSeg2(sevenSegRightX, sevenSegY, scanX, ScanY, valueRight[4]) ||
   testSevenSeg3(sevenSegRightX, sevenSegY, scanX, ScanY, valueRight[3]) ||
   testSevenSeg4(sevenSegRightX, sevenSegY, scanX, ScanY, valueRight[2]) ||
   testSevenSeg5(sevenSegRightX, sevenSegY, scanX, ScanY, valueRight[1]) ||
   testSevenSeg6(sevenSegRightX, sevenSegY, scanX, ScanY, valueRight[0]))
);
macro expr testSevenSeg0(beginX, beginY, currX, currY, value) =
(
  (value &&
  (currX >= beginX) &&
  (currX <= (beginX + vertSegmentWidth)) &&
  (currY >= (beginY + horizSegmentHeight
  + segmentSpacing)) &&
  (currY <= (beginY + horizSegmentHeight
  + segmentSpacing + vertSegmentHeight)))
);
macro expr testSevenSeg1(beginX, beginY, currX, currY, value) =
(
  (value &&
  (currX >= (beginX + vertSegmentWidth
  + segmentSpacing)) &&
  (currX <= (beginX + vertSegmentWidth
  + segmentSpacing + horizSegmentWidth)) &&
  (currY >= (beginY)) &&
  (currY <= (beginY + horizSegmentHeight)))
);
macro expr testSevenSeg2(beginX, beginY, currX, currY, value) =
(
  (value &&
```

```
  (currX >= (beginX + vertSegmentWidth
  + horizSegmentWidth + 2 * segmentSpacing)) &&
  (currX <= (beginX + vertSegmentWidth
  + horizSegmentWidth + 2
     * segmentSpacing + vertSegmentWidth)) &&
  (currY >= (beginY + horizSegmentHeight
  + segmentSpacing)) &&
  (currY <= (beginY + horizSegmentHeight
  + segmentSpacing + vertSegmentHeight)))
);
macro expr testSevenSeg3(beginX, beginY, currX, currY, value) =
(
  (value &&
  (currX >= (beginX + vertSegmentWidth + horizSegmentWidth
  + 2 * segmentSpacing)) &&
  (currX <= (beginX + vertSegmentWidth + horizSegmentWidth
  + 2 * segmentSpacing + vertSegmentWidth)) &&
  (currY >= (beginY + 2 * horizSegmentHeight
  + 3 * segmentSpacing + 1 * vertSegmentHeight)) &&
  (currY <= (beginY + 2 * horizSegmentHeight
  + 3 * segmentSpacing + 2 * vertSegmentHeight)))
);
macro expr testSevenSeg4(beginX, beginY, currX, currY, value) =
(
  (value &&
  (currX >= (beginX + vertSegmentWidth
  + segmentSpacing)) &&
  (currX <= (beginX + vertSegmentWidth
  + segmentSpacing + horizSegmentWidth)) &&
  (currY >= (beginY + 2 * horizSegmentHeight
  + 4 * segmentSpacing + 2 * vertSegmentHeight)) &&
  (currY <= (beginY + 3 * horizSegmentHeight
  + 4 * segmentSpacing + 2 * vertSegmentHeight)))
);
macro expr testSevenSeg5(beginX, beginY, currX, currY, value) =
(
  (value &&
  (currX >= beginX) &&
  (currX <= (beginX + vertSegmentWidth)) &&
  (currY >= (beginY + 2 * horizSegmentHeight
  + 3 * segmentSpacing + 1 * vertSegmentHeight)) &&
  (currY <= (beginY + 2 * horizSegmentHeight
  + 3 * segmentSpacing + 2 * vertSegmentHeight)))
);
macro expr testSevenSeg6(beginX, beginY, currX, currY, value) =
(
  (value &&
  (currX >= (beginX + vertSegmentWidth + segmentSpacing)) &&
  (currX <= (beginX + vertSegmentWidth + segmentSpacing
  + horizSegmentWidth)) &&
  (currY >= (beginY + 1 * horizSegmentHeight
  + 2 * segmentSpacing + 1 * vertSegmentHeight)) &&
  (currY <= (beginY + 2 * horizSegmentHeight
  + 2 * segmentSpacing + 1 * vertSegmentHeight)))
);
```

Appendix B: Project 2 – Display Video Parameter Values.

```
// display_params.hcc
/*****************************************************************
* Project   :    Display video and RAM constants in the console    *
* Date      :    26 Apr 2004                                       *
* File      :    display_params.hcc                                *
* Author    :    Sergey Averchenkov                                *
*****************************************************************
* Desc      :    This program displays the following parameters    *
*           :    on the console at run time:                       *
*           :       The number of visible pixels per scan line,    *
*           :     determined at compile time.                       *
*           :       The number of visible scan lines,               *
*           :     determined at compile time.                       *
*           :       The number of visible pixels per scan line,    *
*           :     determined at run time.                           *
*           :       The number of visible scan lines,               *
*           :     determined at run time.                           *
*           :       The total number of pixels per scan line.      *
*           :       The total number of scan lines.                 *
*           :       The width (number of bits) needed for a variable*
*           :     that holds the X coordinate of a visible pixel.   *
*           :       The width needed for a variable that holds the Y*
*           :     coordinate of a visible pixel.                    *
*           :       The width needed for a variable that ranges over*
*           :     all X coordinates on a scan line.                 *
*           :       The width needed for a variable that ranges over*
*           :     all Y coordinates on the display.                 *
*           :       The width needed for a variable that holds a    *
*           :     RAM memory address.                               *
*           :       The width needed for a variable that holds a    *
*           :     word of RAM data.                                 *
*****************************************************************/

#if (defined USE_RC200 || USE_RC200E)
  #define PAL_TARGET_CLOCK_RATE 25175000
#else
  #define PAL_ACTUAL_CLOCK_RATE 25175000
  set clock = external "P1"
    with
    {
      extlib  = "DKSync.dll",
      extinst = "1000",       // Period of 1MHz simulated clock
      extfunc = "DKSyncGetSet"
    };
#endif

#include <pal_master.hch>
#include <pal_console.hch>
#include <stdlib.hch>

static macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
static macro proc microsec_delay(msec);

macro expr VideoOut = PalVideoOutOptimalCT(PAL_ACTUAL_CLOCK_RATE);
macro proc ConsoleRun(ConsolePtr);
```

```
// main()
// ---------------------------------------------------------------
/*
 *    Main entry of the program. Runs all threads.
 */
void main (void)
{
  PalConsole *ConsolePtr;

  PalVersionRequire(1, 0);
  PalVideoOutRequire(1);

  par
  {
    PalConsoleRun(&ConsolePtr, PAL_CONSOLE_FONT_NORMAL,
      PalVideoOutOptimalCT(ClockRate), ClockRate);
    seq
    {

      PalConsoleEnable(ConsolePtr);
      PalConsoleClear(ConsolePtr);
      ConsoleRun(ConsolePtr);
    }
  }
}
// ---------------------------------------------------------------


// ConsoleRun()
// ---------------------------------------------------------------
/*
 *    Runs console output thread.
 */
macro proc ConsoleRun(ConsolePtr)
{
  static ram unsigned char String1[13]  = "Visible X: ";
  static ram unsigned char String2[13]  = "Visible Y: ";
  static ram unsigned char String3[11]  = "Total X: ";
  static ram unsigned char String4[11]  = "Total Y: ";
  static ram unsigned char String5[14]  = "Visible Xct: ";
  static ram unsigned char String6[14]  = "Visible Yct: ";
  static ram unsigned char String7[18]  = "Visible X width: ";
  static ram unsigned char String8[18]  = "Visible Y width: ";
  static ram unsigned char String9[14]  = "Max X width: ";
  static ram unsigned char String10[14] = "Max Y width: ";
  static ram unsigned char String11[24] = "Max RAM address width: ";
  static ram unsigned char String12[21] = "Max RAM data width: ";

  macro expr VisibleX = PalVideoOutGetVisibleX(VideoOut, ClockRate);
  macro expr VisibleY = PalVideoOutGetVisibleY(VideoOut);
  macro expr TotalX = PalVideoOutGetTotalXCT(VideoOut, ClockRate);
  macro expr TotalY = PalVideoOutGetTotalYCT(VideoOut);
  macro expr VisibleXct = PalVideoOutGetVisibleXCT(VideoOut, ClockRate);
  macro expr VisibleYct = PalVideoOutGetVisibleYCT(VideoOut);
  macro expr VisibleXWidth = PalVideoOutGetXWidthCT(VideoOut);
  macro expr VisibleYWidth = PalVideoOutGetYWidthCT(VideoOut);
  macro expr MaxXWidth = PalVideoOutGetMaxXWidthCT();
  macro expr MaxYWidth = PalVideoOutGetMaxYWidthCT();
  macro expr MaxRAMAddressWidth = PalPL1RAMGetMaxAddressWidthCT();
```

```
    macro expr MaxRAMDataWidth = PalPL1RAMGetMaxDataWidthCT();

    PalConsolePutString(ConsolePtr, String1);
    PalConsolePutUInt(ConsolePtr, 0@VisibleX);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String2);
    PalConsolePutUInt(ConsolePtr, 0@VisibleY);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String3);
    PalConsolePutUInt(ConsolePtr, TotalX);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String4);
    PalConsolePutUInt(ConsolePtr, TotalY);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String5);
    PalConsolePutUInt(ConsolePtr, VisibleXct);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String6);
    PalConsolePutUInt(ConsolePtr, VisibleYct);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String7);
    PalConsolePutUInt(ConsolePtr, VisibleXWidth);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String8);
    PalConsolePutUInt(ConsolePtr, VisibleYWidth);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String9);
    PalConsolePutUInt(ConsolePtr, MaxXWidth);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String10);
    PalConsolePutUInt(ConsolePtr, MaxYWidth);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String11);
    PalConsolePutUInt(ConsolePtr, MaxRAMAddressWidth);
    PalConsolePutChar(ConsolePtr, '\n');

    PalConsolePutString(ConsolePtr, String12);
    PalConsolePutUInt(ConsolePtr, MaxRAMDataWidth);
    PalConsolePutChar(ConsolePtr, '\n');
}
```

Appendix C: Project 3 – Draw a Test Pattern by Synchronizing with HBlank.

```
// unbuffered.hcc
/*****************************************************************
* Project   :    Unbuffered drawing                                *
* Date      :    26 Apr 2004                                       *
* File      :    unbuffered.hcc                                    *
* Author    :    Sergey Averchenkov                                *
*****************************************************************
* Desc      :    This program uses the PalVideoOutGetHBlank(   *
*           :    handle ) to synchronize with the beginning of *
*           :    each scan line. It draws a white pixel in the *
*           :    first column of each scan line, and draws     *
*           :    vertical bars of alternating colors across the*
*           :    remainder of each line.                           *
*****************************************************************/

#if (defined USE_RC200 || USE_RC200E)
  #define PAL_TARGET_CLOCK_RATE 25175000
#else
  #define PAL_ACTUAL_CLOCK_RATE 25175000
  set clock = external "P1"
    with
    {
      extlib  = "DKSync.dll",
      extinst = "1000",       // Period of 1MHz simulated clock
      extfunc = "DKSyncGetSet"
    };
#endif

#include <pal_master.hch>
#include <stdlib.hch>

static macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
static macro expr VideoOut = PalVideoOutOptimalCT(PAL_ACTUAL_CLOCK_RATE);
static macro expr ScanX = PalVideoOutGetX(VideoOut);
static macro expr PL1RAM   = PalPL1RAMCT(0);
static macro proc VideoRun(VideoPtr);

// main()
// -------------------------------------------------------------
/*
 *    Main entry of the program. Runs all threads.
 */
void main (void)
{
  PalVersionRequire(1, 0);
  PalVideoOutRequire(1);
  PalPL1RAMRequire(1);

  par
  {
    PalVideoOutRun(VideoOut, ClockRate);
    seq
    {
      PalVideoOutEnable(VideoOut);
      VideoRun(VideoOut);
    }
  }
```

```
}
// ------------------------------------------------------------


// ------------------------------------------------------------
// VideoRun()
// ------------------------------------------------------------
/*
 *    This thread outputs the alternating color vertical bars
 *    to the LCD screen. This method does not return.
 */
macro proc VideoRun(VideoOut)
{
  while(1)
  {
    if(PalVideoOutGetHBlank(VideoOut))
    {
      PalVideoOutWrite(VideoOut, 0xFFFFFF);
    }
    else if(ScanX[5] == 0)
    {
      PalVideoOutWrite(VideoOut, 0xFF0000);
    }
    else
    {
      PalVideoOutWrite(VideoOut, 0x0000FF);
    }
  }
}
// ------------------------------------------------------------
```

Appendix D: Project 4 – Draw an Image From a Framebuffer.

```
// framebuffer.hcc
/*****************************************************************
* Project   :    Buffered drawing using PL1RAM                    *
* Date      :    26 Apr 2004                                      *
* File      :    framebuffer.hcc                                  *
* Author    :    Sergey Averchenkov                               *
*****************************************************************
* Desc      :    This code continuously reads from the            *
*           :    framebuffer and draw the pixels to their proper*
*           :    locations on the screen. Because of the delays *
*           :    involved in reading from the memory and writing*
*           :    to the screen, we had to start processing each *
*           :    scan line during the end of the HBlank period  *
*           :    of the previous scan line.                       *
*****************************************************************/
#if (defined USE_RC200 || USE_RC200E)
  #define PAL_TARGET_CLOCK_RATE 25175000
#else
  #define PAL_ACTUAL_CLOCK_RATE 25175000
  set clock = external "P1"
    with
    {
      extlib  = "DKSync.dll",
      extinst = "1000",       // Period of 1MHz simulated clock
      extfunc = "DKSyncGetSet"
    };
#endif

#include <pal_master.hch>
#include <stdlib.hch>

static macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
static macro expr VideoOut = PalVideoOutOptimalCT(PAL_ACTUAL_CLOCK_RATE);
static macro expr PL1RAM   = PalPL1RAMCT (0);
static macro expr MaxRAMAddressWidth = PalPL1RAMGetMaxAddressWidthCT();
static macro expr MaxRAMDataWidth = PalPL1RAMGetMaxDataWidthCT();
static macro expr MaxXWidth = PalVideoOutGetMaxXWidthCT();
static macro expr MaxYWidth = PalVideoOutGetMaxYWidthCT();
static macro expr VisibleX = PalVideoOutGetVisibleXCT(VideoOut, ClockRate);
static macro expr VisibleY = PalVideoOutGetVisibleYCT(VideoOut);
static macro expr MaxX = PalVideoOutGetTotalXCT(VideoOut, ClockRate);
static macro expr MaxY = PalVideoOutGetTotalYCT(VideoOut);
static macro expr ScanX = PalVideoOutGetX(VideoOut);
static macro expr ScanY = PalVideoOutGetY(VideoOut);

static macro proc VideoRun(VideoPtr);
static macro proc BuildImage();
void write_pixel(unsigned (MaxXWidth) x, unsigned (MaxYWidth) y,
  unsigned 24 color);

// main()
// -------------------------------------------------------------
/*
 *    Main entry of the program. Runs all threads.
 */
void main (void)
{
```

```
  PalVersionRequire(1, 0);
  PalVideoOutRequire(1);

  par
  {
    PalVideoOutRun(VideoOut, ClockRate);
    PalPL1RAMRun(PL1RAM, ClockRate);
    seq
    {
      par
      {
        PalVideoOutEnable(VideoOut);
        PalPL1RAMEnable(PL1RAM);
      }
      VideoRun(VideoOut);
      BuildImage();
    }
  }
}
// -------------------------------------------------------------


// -------------------------------------------------------------
// VideoRun()
// -------------------------------------------------------------
/*
 *    This thread outputs the alternating color vertical bars
 *    to the LCD screen. This method does not return.
 */
macro proc VideoRun(VideoOut)
{
  unsigned (MaxRAMDataWidth) data;
  unsigned (MaxRAMAddressWidth) addr;
  addr = 0;

  while(ScanY < MaxY - 1) { delay; }
  while(ScanX < MaxX - 2) { delay; }

  par
  {
    PalPL1RAMSetReadAddress(PL1RAM, addr);
    addr++;
  }
  par
  {
    PalPL1RAMRead(PL1RAM, &data);
    PalPL1RAMSetReadAddress(PL1RAM, addr);
    addr++;
  }
  while(1)
  {
    par
    {
      PalPL1RAMSetReadAddress(PL1RAM, addr);
      PalPL1RAMRead(PL1RAM, &data);
      PalVideoOutWrite(VideoOut, data<-24);
      addr = (addr < (MaxX * MaxY)-1) ? addr + 1 : 0;
    }
  }
```

```
}
// ------------------------------------------------------------


// ------------------------------------------------------------
// BuildImage()
// ------------------------------------------------------------
/*
 *    Procedure generates a test pattern in the framebuffer.
 */
macro proc BuildImage()
{
  unsigned (MaxXWidth) X;
  unsigned (MaxYWidth) Y;

  for(Y = 0; Y < VisibleY; Y++)
  {
    for(X = 0; X < VisibleX; X++)
    {
      write_pixel(X, Y, (X[3] ? 0xFF0000 : 0x0000FF));
    }
  }

  for(X = 0; X < VisibleX; X++)
  {
    write_pixel(X, 0, 0xFFFFFF);
    write_pixel(X, VisibleY - 1, 0xFFFFFF);
  }

  for(Y = 0; Y < VisibleY; Y++)
  {
    write_pixel(0, Y, 0xFFFFFF);
    write_pixel(VisibleX - 1, Y, 0xFFFFFF);
  }
}
// ------------------------------------------------------------


// ------------------------------------------------------------
// write_pixel()
// ------------------------------------------------------------
/*
 *    Procedure a pixel to the framebuffer.
 */
void write_pixel(unsigned (MaxXWidth) x, unsigned (MaxYWidth) y,
  unsigned 24 color)
{
  unsigned (MaxRAMAddressWidth) address;
  address = adju(y, MaxRAMAddressWidth) * MaxX + adju(x,
    MaxRAMAddressWidth);
  PalPL1RAMSetWriteAddress(PL1RAM, address);
  PalPL1RAMWrite(PL1RAM, 0@color);
}
```