Paul Kafka

EEL4930 - Embedded Microprocessor System Design

12/9/13 - Fall 2013

Final Project

LCD - Custom Characters

Table of Contents

1. Introduction

The objective of this project was to create an FPGA-based NIOS II system design of one's choice. This report will discuss a project that implements custom characters on the LCD component. Below, Figure 1 is the schedule created to complete this task.

Term Research Project - LCD - Gantt Chart														
			Novemeber								December			
	Name	Dates	1-5	6	7-12	13	14-19	20	21-26	27	1-3	4	4-9	9
1	Weekly Report 1	11/6/2013		¥										
1.a	Research	11/1-5/2013												
2	Weekly Report 2	11/13/2013				¥								
2.a	Devlope Algorithm	11/7-12/2013												
3	Weekly Report 3	11/20/2013						¥						
3.a	Convert Algorithm to C	11/14-19/2013												
4	Weekly Report 4	11/27/2013								¥				
4.a	Implement the code	11/21-26/2013												
5	Presentation	12/4/2013										¥		
5.a	Prepare for Presentation	12/1-3/2013												
6	Report Due	12/9/2013												¥
6.a	Write Final Report	12/4-9/2013												

Figure 1: Gantt Chart

2. System Information

The Altera DE2 Board with Cyclone II FPGA is shown in Figure 2. The SOPC Builder in Quartus II, Nios II processor, Monitor Program, and HAL system library drivers were the necessary elements that were used to interact with the LCD and create custom characters for display.

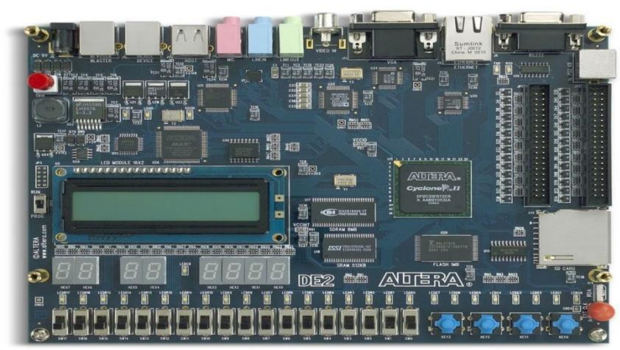


Figure 2: DE2 Board

The SOPC Builder instantiates the character LCD Core component for the Nios II processor. The Nios II processor also contains some simple I/O peripherals. The Monitor Program then takes this custom computer built by the SOPC Builder and provides users with the ability to add, compile, and run their c programs on the Nios processor. In the Monitor Program, one could select a basic computer or media converter for quick and easier implementation, instead of the custom computer. Lastly, the HAL system drivers enable one to access the LCD controller using the ANSI C standard library functions for the Nios processor.

3. LCD Research

LCD is an acronym for Liquid-Crystal Display. The liquid crystals have Light modulating properties. The LCD Core supports a clock frequency of 50 MHz. The Character LCD Core sends characters to the LCD according to the Character Generator ROM Pattern of the LCD.

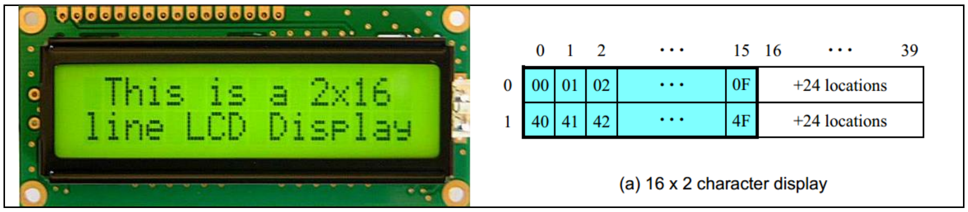


Figure 3: Locations

Data can be sent to the display as ASCII character codes, which are automatically converted by the 16x2 character display into bit patterns using a built-in font. After the location of the cursor has been set, a character can be loaded into this location by writing its ASCII value into the Data register. As characters are written, the cursor position is updated. The LCD controller is an output-only device.

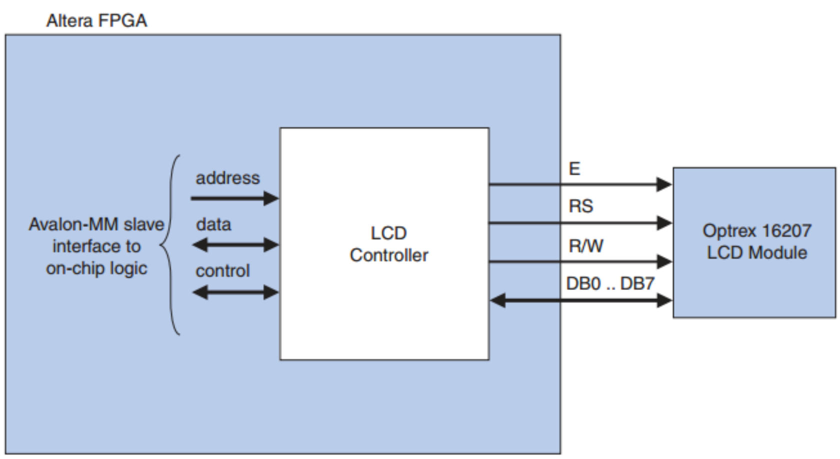


Figure 4: LCD Controller

4. Implementation

a. Initialization

Before the LCD component can be used, it has to initialize. The code for this initialization is in code section of the report. The flow chart is depicted in Figure 5.

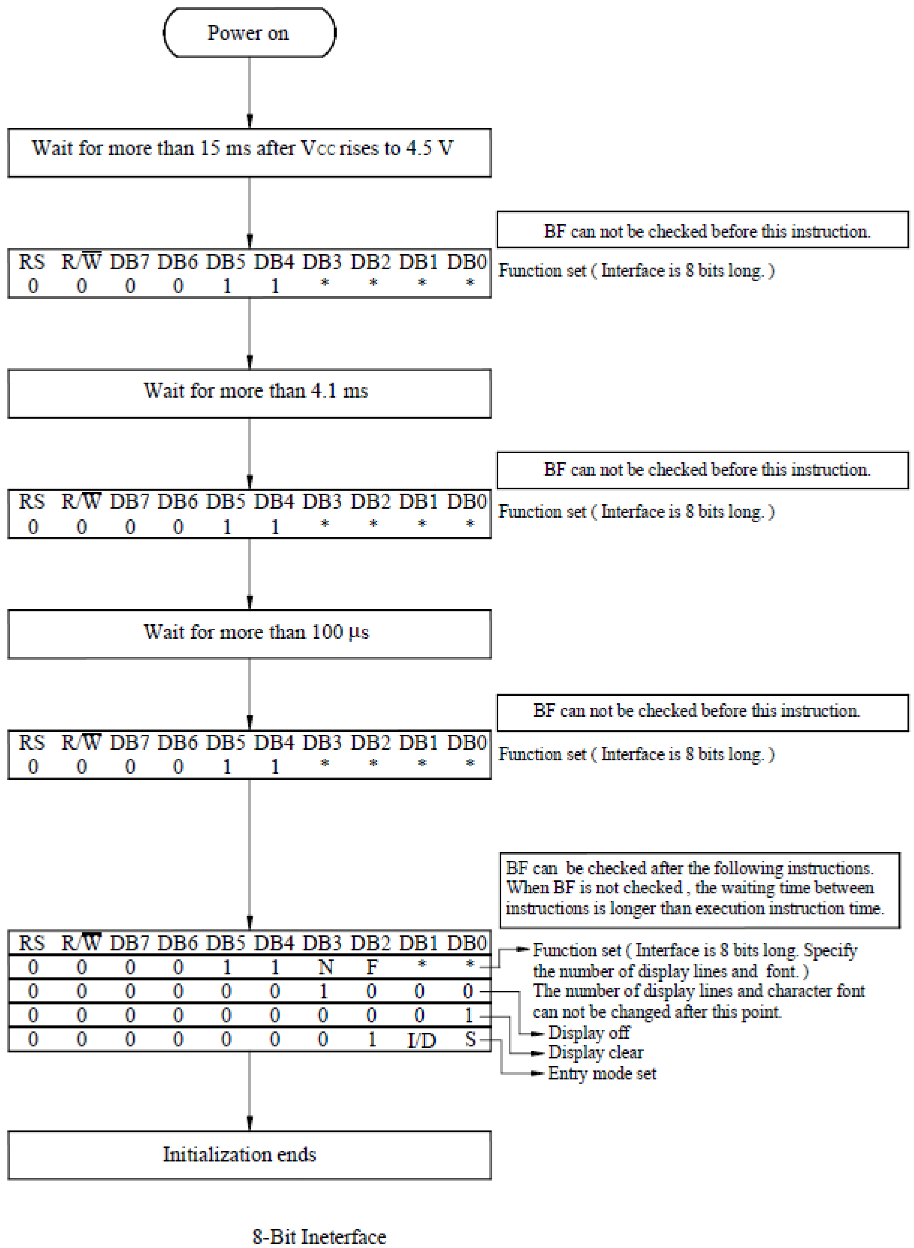


Figure 5: Initialization Steps

b. Character Creation

There are three types of memory used for the LCD. The Character Generator ROM (CGROM) contains predefined characters, like ASCII fonts. From character codes, a 5x8 dot character pattern is generated. These patterns cannot be changed (read only memory). On the other hand, Character Generator RAM (CGRAM), is the exact same as the ROM version, but custom dot patterns can be saved and rewritten. There are only a few locations available for custom characters in the CGRAM as shown in Figure 6. Custom character can be created by masking the CGROM characters, but it is much easier to just create new ones. Lastly, the memory that displays the characters is the Display Data RAM (DDRAM). The CGROM and CGRAM data is accessed by the DDRAM.

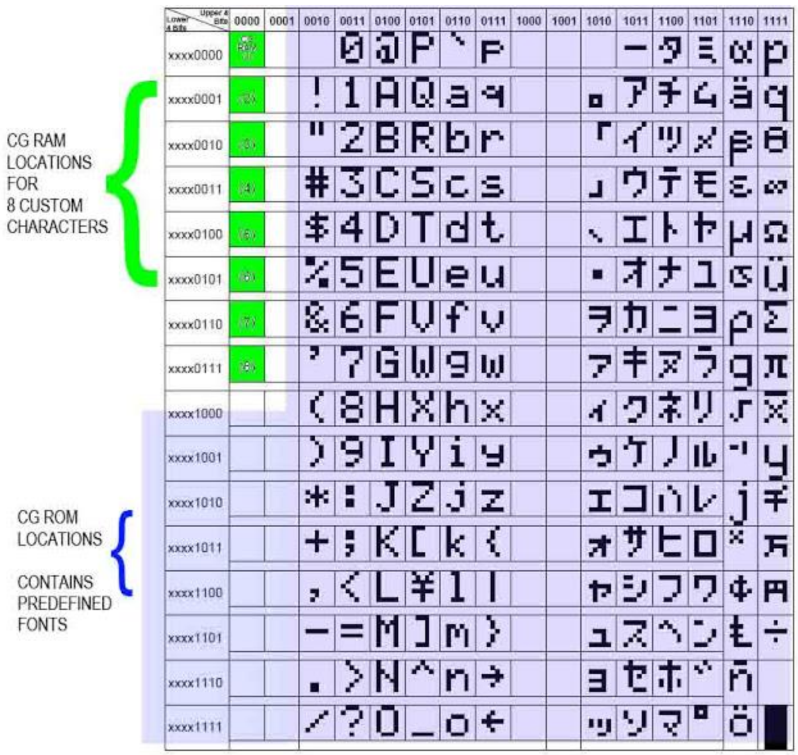


Figure 6: Memory

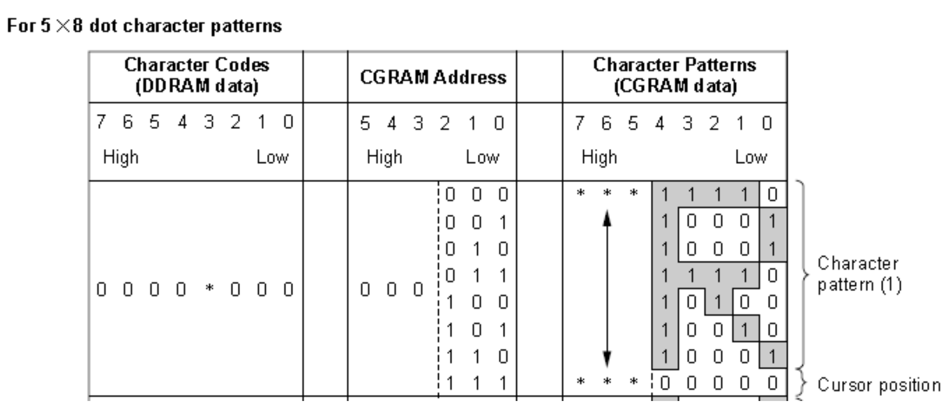


Figure 7: Char Dot Patterns

Once the data is sent and displayed using HAL functions as shown in the code section Figure 8 shows an example of a character being created. Each row's columns are counted as bits and a number is generated. These numbers are then sent to the CGRAM and then displayed.

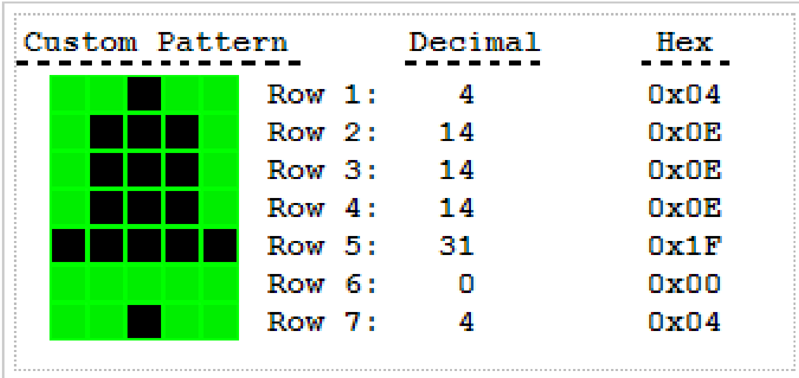


Figure 8: Custom Pattern

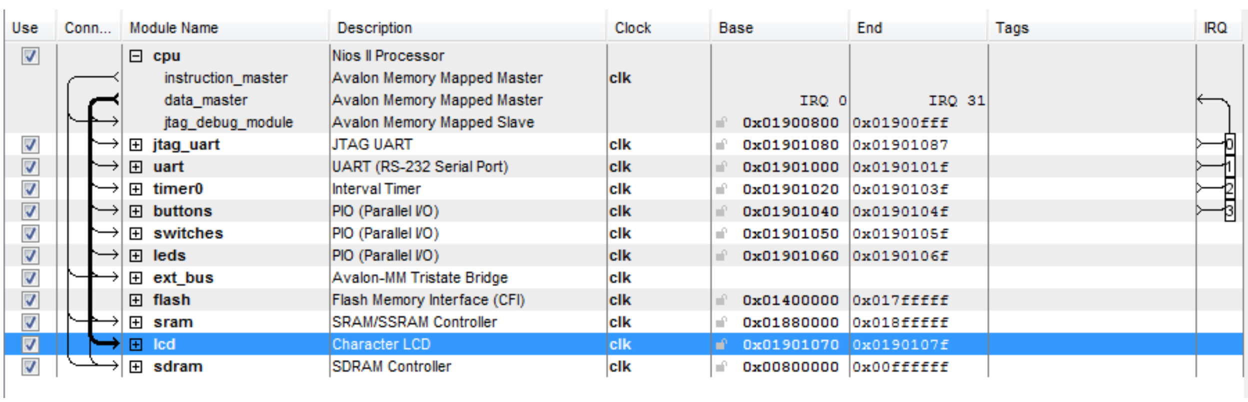
The LCD controller is accompanied by the following software files. These files define the low-level interface to the hardware and provide the HAL drivers. It is said that application developers should not modify these files.

altera_avalon_lcd_16207_regs.h — This file defines the core's register map, providing symbolic constants to access the low-level hardware.

altera_avalon_lcd_16207.h, **altera_avalon_lcd_16207.c** — These files implement the LCD controller device drivers for the HAL system library

system.h – Has the SOPC LCD Configurations

Figure 9 shows an example of the SOPC configuration. Notice the LCD component is added.



Use	Conn...	Module Name	Description	Clock	Base	End	Tags	IRQ
☑		cpu	Nios II Processor					
		instruction_master	Avalon Memory Mapped Master	clk				
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave		0x01900800	0x01900fff		
☑		jtag_uart	JTAG UART	clk	0x01901080	0x01901087		0
☑		uart	UART (RS-232 Serial Port)	clk	0x01901000	0x0190101f		1
☑		timer0	Interval Timer	clk	0x01901020	0x0190103f		2
☑		buttons	PIO (Parallel I/O)	clk	0x01901040	0x0190104f		3
☑		switches	PIO (Parallel I/O)	clk	0x01901050	0x0190105f		
☑		leds	PIO (Parallel I/O)	clk	0x01901060	0x0190106f		
☑		ext_bus	Avalon-MM Tristate Bridge	clk				
☑		flash	Flash Memory Interface (CFI)	clk	0x01400000	0x017fffff		
☑		sram	SRAM/SSRAM Controller	clk	0x01880000	0x018fffff		
☑		lcd	Character LCD	clk	0x01901070	0x0190107f		
☑		sdram	SDRAM Controller	clk	0x00800000	0x00ffffff		

Figure 9: SOPC Builder

The Monitor program is set up for Custom Computer and assembles the c program which implements the created character display. Figure 10 illustrates the configuration.

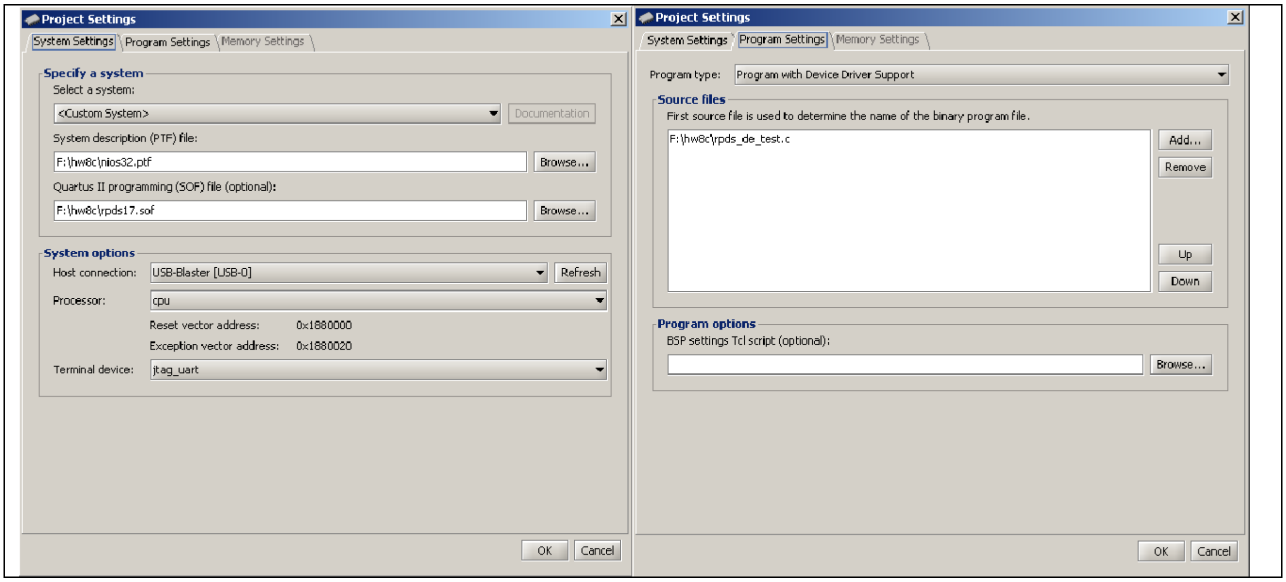


Figure 10: Monitor Program

5. Code

a. Header Code

```
#ifndef RPDS_DE_TEST_H_
#define RPDS_DE_TEST_H_

#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
#include "sys/alt_flash.h"
#include "altera_avalon_pio_regs.h"

#ifdef LCD_NAME
    /* LCD constants */
    #define LCD_WR_COMMAND_REG  0
    #define LCD_WR_DATA_REG     2
#endif

/* Memory constants */
#define SRAM_MAX_WORDS      8000
#define FLASH_MAX_WORDS     1000
#define SDRAM_MAX_WORDS     1000000


#endif /*RPDS_DE_TEST_H_*/
```

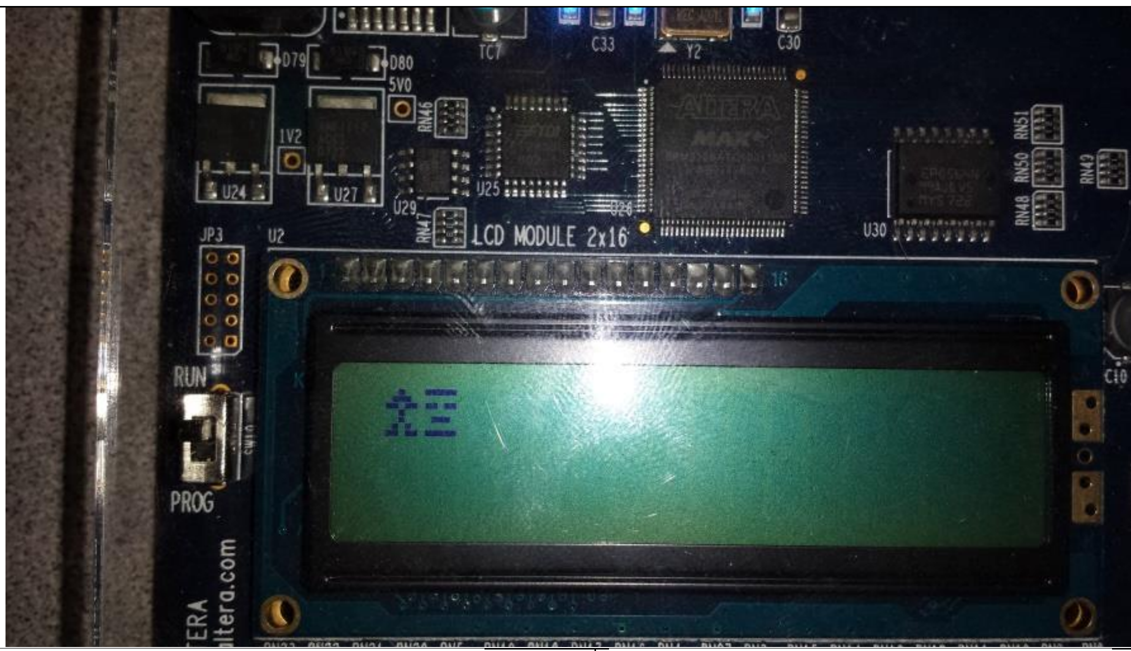
b. Implementation Code

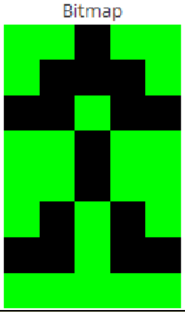
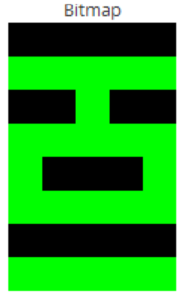
```c
#include "rpds_de_test.h"

static void buttons_isr( void* context, alt_u32 id ) {
  volatile int *function = (volatile int*) context;

  *function = IORD_ALTERA_AVALON_PIO_EDGE_CAP( BUTTONS_BASE );
  IOWR_ALTERA_AVALON_PIO_EDGE_CAP( BUTTONS_BASE, 0 );
  IOWR_ALTERA_AVALON_PIO_IRQ_MASK( BUTTONS_BASE, 0xF );
}

void lcd_init( void ) {
  /* Set Function Code Four Times -- 8-bit, 2 line, 5x7 mode */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x38 );
  usleep(4100);   /* Wait 4.1 ms */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x38 );
  usleep(100);    /* Wait 100 us */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x38 );
  usleep(5000);   /* Wait 5.0 ms */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x38 );
  usleep(100);
  /* Set Display to OFF */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x08 );
  usleep(100);
  /* Set Display to ON */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x0C );
  usleep(100);
  /* Set Entry Mode -- Cursor increment, display doesn't shift */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x06 );
  usleep(100);
  /* Set the cursor to the home position */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x02 );
  usleep(2000);
  /* Clear the display */
  IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x01 );
  usleep(2000);
}

void customChar(int ** cChar, int pos) {
    int i;
    IOWR (LCD_BASE, LCD_WR_COMMAND_REG, 0x48 + (pos *8));
    usleep(10000);

    for (i=0; i<8; i++) {
        IOWR(LCD_BASE, LCD_WR_DATA_REG, cChar[i]);
        usleep(100);
    }

    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x80 + pos);
    usleep(100);

    IOWR(LCD_BASE, LCD_WR_DATA_REG, pos +1);
    usleep(500000);
}
```

```c
int main( void ) {
  int * person[8] = { 0x04, 0x0e, 0x1b, 0x04, 0x04, 0x0a, 0x1b, 0x00};
  int * face[8] = { 0x1f, 0x00, 0x1b, 0x00, 0x0e, 0x00, 0x1f, 0x00 };

  while (1) {
    lcd_init();
    customChar(person, 0x00);
    customChar(face, 0x01);
  }
  return(0);
}
```

6. Results

Below is picture proof of the result. Since there is no user interaction (buttons or switches), a picture was taken instead of a video of the outcome. As shown, the characters created were displayed correctly.



Bitmap	Decimal	Bitmap	Decimal
	4 14 27 4 4 10 27 0		31 0 27 0 14 0 31 0

```c
int * person[8] = { 0x04, 0x0e, 0x1b, 0x04,
0x04, 0x0a, 0x1b, 0x00};
```

```c
int * face[8] = { 0x1f, 0x00, 0x1b,
0x00, 0x0e, 0x00, 0x1f, 0x00 };
```

7. Limitations

An attempt using the media computer was made, but it was easier to use the custom computer. There is a limit on the amount of custom character that can be stored in the CGRAM. The HAL device drivers make it unnecessary for you to access the IR and DR registers directly, therefore, Altera does not publish details about the register map. The altera_avalon_lcd_16207_regs.h is given but its implementation file isn't.

8. Conclusion

Knowledge was gained on how to initialize and use the LCD component using the Custom Computer built from the SOPC Builder in Quartus II for the Nios II processor. In the future, people can either create many custom fonts, or make character animations on the LCD.

9. References

[1] Character LCD Core for Altera DE2 Board, Altera Corporation – University Program October 2006

[2] Media Computer System for the Altera DE2-115 Board, Altera Corporation – University Program July 2010

[3] HD44780U (LCD-II) Dot Matrix Liquid Crystal Display Controller/Driver Hitachi, Data Sheet

[4] Dot Matrix Character LCD Module User's Manual, Optrex Corporation

[5] Springer, Rapid Prototyping of Digital Systems SOPC Edition