

Configuration of Nios™ Soft Core & Peripherals
and
Executing Software on a Nios™ Embedded Processor

LAB #4-5 EE 2160
October 11, 2004

BY:
Narayanan Krishnamurthy

TABLE OF CONTENTS

| | |
|--|----|
| TABLE OF CONTENTS..... | 2 |
| Purpose:..... | 3 |
| Procedure: | 3 |
| Design: | 5 |
| Nios softcore processor and its peripherals: | 5 |
| Pin assignments:..... | 6 |
| Block Diagram: | 8 |
| Sorting Algorithm – source code: | 9 |
| //Initialization: | 9 |
| //Main module:..... | 9 |
| //Selection Sort module:..... | 10 |
| // Bubble Sort module: | 10 |
| //LCD Write Routine: | 11 |
| //Print through UART to stdout: | 11 |
| Results:..... | 12 |
| Conclusion: | 13 |

Purpose:

The purpose of lab 4 was to implement a 32 bit Nios Soft Core and peripherals using Quartus II. The reference design was used to expedite the process of adding and assigning pins. In lab 5, C code was created, compiled, and executed on a Nios embedded processor running on an Altera Excalibur board. Nios SDK Unix shell was used to execute and download the code on the FPGA board.

Procedure:

The Nios embedded processor was created using the instructions on the lab handout and the Nios Tutorial provided as part of the lab. Following steps outlay the processor creation process:

1. A top-level Block Design file (.bdf) called **nios_system_module** was created.
2. The MegaWizard Plug-In Manager was then used to launch the Altera SOPC Builder 2.8. The SOPC Builder allows for the creation of the Nios CPU and peripherals.
3. The following modules were then added to the SOPC Builder:
 - Nios 32-Bit CPU
 - Boot Monitor ROM
 - Communications UART
 - Debugging UART
 - Timer
 - Button PIO
 - LCD PIO
 - LED PIO
 - Seven Segment PIO
 - External RAM Bus (Avalon Tri-State Bridge)
 - External RAM Interface
 - External Flash Interface
4. The base address and IRQs were then verified using Figure 27 of the Nios Tutorial. Please refer to Figure 1 in the design section for the various parameters in Nios Processor and peripherals.
5. The design was then synthesized using the Generate command in the SPOC Builder.

6. After the generation of the design, the symbol for the Nios system module was added to the BDF.

The input and output pin connections to the CPU was made in the .BDF file and compiled. The .CSF file was edited to create pin assignments the reference design and the reference .CSF file were used for this purpose. For each I/O pin appearing in the schematic, the corresponding pin assignment statement were copied from the reference design .CSF file into he same place in the .CSF file for the lab project.

Programmer command was then used to download the Nios Processor design into the APEX20KE FPGA in the Excalibur target board.

Nios SDK shell was then used to run the nios-build and nios-run utilities on the Nios development board. As a test for a correct implementation of the Nios processor, the hello_nios file was downloaded and tested on the FPGA board. Following two commands were used to first build and then run output of the file:

```
nb hello_nios.c ←  
nr hello_nios.srec ←
```

The program generated the message “Hello, from Nios!” and caused the 2-digit 7-segment LED to count down from 99 to 0.

As part of the assignment, a C program was written to implement bubble sort and selection sort. The program took a set of 10 unsorted integer values as inputs and sorted them by employing the Selection Sort and the Bubble Sort technique. The output of the programs, showing the values being sorted step-by-step was displayed to both the monitor and the LCD display. Please refer to the design section for the source code and see figure 3 in the results section for the program output.

Design:

Nios softcore processor and its peripherals: The Altera-Excalibur's tutorial was used for creating the 32-bit Nios 33.33 Mhz processor. The related peripherals including ROM, RAM, and the other peripherals listed below were also created as per the tutorial.

| | |
|---|---|
| <ul style="list-style-type: none"> ■ Boot Monitor ROM ■ UART_0 ■ UART_1_debug ■ Timer_0 ■ Button PIO | <ul style="list-style-type: none"> ■ LCD PIO ■ LED PIO ■ Seven Segment PIO ■ External RAM Bus (Avalon Tri-State Bridge) ■ External RAM Interface ■ External Flash Interface |
|---|---|

Note: Check for the IRQ interrupt nos(16-uart_0,17-uart_1_debug,18-timer_0 and 19-button_pio, the priority has to be maintained in this order for the processor to function properly.

Figure 1: Nios Processor and Peripherals

Target Device Family: APEX 20KE System Clock Frequency: 33.333 MHz

| Use | Module Name | Description | Base | End | IRQ |
|-------------------------------------|------------------|--|------------|--------------|-----|
| <input checked="" type="checkbox"/> | cpu | Nios Processor - Altera Corporation | | | |
| <input checked="" type="checkbox"/> | boot_monitor_rom | On-Chip Memory (RAM or ROM) | 0x00000000 | 0x000003FF | |
| <input checked="" type="checkbox"/> | uart_0 | UART (RS-232 serial port) | 0x00000400 | 0x0000041F | 16 |
| <input checked="" type="checkbox"/> | uart_1_debug | UART (RS-232 serial port) | 0x00000420 | 0x0000043F | 17 |
| <input checked="" type="checkbox"/> | timer_0 | Interval timer | 0x00000440 | 0x0000045F | 18 |
| <input checked="" type="checkbox"/> | button_pio | PIO (Parallel I/O) | 0x00000460 | 0x0000046F | 19 |
| <input checked="" type="checkbox"/> | lcd_pio | PIO (Parallel I/O) | 0x00000470 | 0x0000047F | |
| <input checked="" type="checkbox"/> | led_pio | PIO (Parallel I/O) | 0x00000480 | 0x0000048F | |
| <input checked="" type="checkbox"/> | seven_seg_pio | PIO (Parallel I/O) | 0x00000490 | 0x0000049F | |
| <input checked="" type="checkbox"/> | ext_ram_bus | Avalon Tri-State Bridge | | | |
| <input checked="" type="checkbox"/> | ext_ram | IDT71V016 SRAM for EP20K200E Nios Development Board | 0x00040000 | 0x0007FFFF | |
| <input checked="" type="checkbox"/> | ext_flash | AMD 29LV800 Flash for EP20K200E Nios Development Board | 0x00100000 | 0x001FFFFFFF | |

Pin assignments: The 32-bit Nios reference design was used to simplify the process of pin assignments, the following pin details of the reference design was copied into the CHIP module of our nios_system_module .CSF file.

Note : Take care in mapping the IO pins to the created processor symbol(check the reference-32bit-Nios design) before generating your .CSF file. Confirm the following options in the CHIP module, of the nios_system_module.csf file, DEVICE = "EP20K200EFC484-2X"; RESERVE_ALL_UNUSED_PINS = "AS INPUT TRI-STATED";

```

CLK_DRV/clk_to_apex : LOCATION = Pin_L6;
DB9_CONNECTOR/rxd : LOCATION = Pin_W8;
DB9_CONNECTOR/txd : LOCATION = Pin_D15;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[0]" : LOCATION = Pin_Y9;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[10]" : LOCATION = Pin_U12;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[11]" : LOCATION = Pin_Y10;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[1]" : LOCATION = Pin_T9;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[2]" : LOCATION = Pin_Y8;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[3]" : LOCATION = Pin_W9;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[4]" : LOCATION = Pin_V9;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[5]" : LOCATION = Pin_U9;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[6]" : LOCATION = Pin_T10;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[7]" : LOCATION = Pin_U10;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[8]" : LOCATION = Pin_V10;
"DIP_SWITCH/all-SW7/out-SW6/out-SW5/out-SW4/out[9]" : LOCATION = Pin_P11;
"DISPLAY/tens_digit-DISPLAY/ones_digit[0]" : LOCATION = Pin_W17;
"DISPLAY/tens_digit-DISPLAY/ones_digit[10]" : LOCATION = Pin_Y17;
"DISPLAY/tens_digit-DISPLAY/ones_digit[11]" : LOCATION = Pin_V8;
"DISPLAY/tens_digit-DISPLAY/ones_digit[12]" : LOCATION = Pin_Y7;
"DISPLAY/tens_digit-DISPLAY/ones_digit[13]" : LOCATION = Pin_U11;
"DISPLAY/tens_digit-DISPLAY/ones_digit[14]" : LOCATION = Pin_R11;
"DISPLAY/tens_digit-DISPLAY/ones_digit[15]" : LOCATION = Pin_D18;
"DISPLAY/tens_digit-DISPLAY/ones_digit[1]" : LOCATION = Pin_U18;
"DISPLAY/tens_digit-DISPLAY/ones_digit[2]" : LOCATION = Pin_Y18;
"DISPLAY/tens_digit-DISPLAY/ones_digit[3]" : LOCATION = Pin_W18;
"DISPLAY/tens_digit-DISPLAY/ones_digit[4]" : LOCATION = Pin_U8;
"DISPLAY/tens_digit-DISPLAY/ones_digit[5]" : LOCATION = Pin_T11;
"DISPLAY/tens_digit-DISPLAY/ones_digit[6]" : LOCATION = Pin_R10;
"DISPLAY/tens_digit-DISPLAY/ones_digit[7]" : LOCATION = Pin_C18;
"DISPLAY/tens_digit-DISPLAY/ones_digit[8]" : LOCATION = Pin_V17;
"DISPLAY/tens_digit-DISPLAY/ones_digit[9]" : LOCATION = Pin_V18;
FLASH/a16 : LOCATION = Pin_F3;
FLASH/ce_n : LOCATION = Pin_E1;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[0]" : LOCATION = Pin_G17;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[10]" : LOCATION = Pin_A3;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[11]" : LOCATION = Pin_A4;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[12]" : LOCATION = Pin_C3;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[13]" : LOCATION = Pin_C1;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[14]" : LOCATION = Pin_D3;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[15]" : LOCATION = Pin_D2;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[16]" : LOCATION = Pin_C2;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[17]" : LOCATION = Pin_D1;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[18]" : LOCATION = Pin_B3;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[19]" : LOCATION = Pin_E3;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[1]" : LOCATION = Pin_A8;

```

```

"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[2]" : LOCATION = Pin_B8;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[3]" : LOCATION = Pin_A7;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[4]" : LOCATION = Pin_B7;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[5]" : LOCATION = Pin_B6;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[6]" : LOCATION = Pin_A6;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[7]" : LOCATION = Pin_A5;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[8]" : LOCATION = Pin_B5;
"FLASH/high_address_bits-SRAM_Lo/address-FLASH/a0-APEX/a0[9]" : LOCATION = Pin_B4;
FLASH/we_n : LOCATION = Pin_H13;
HEADER_SWITCH/enable1_n : LOCATION = Pin_V7;
LCD_HEADER/interface_pins[0] : LOCATION = Pin_U21;
LCD_HEADER/interface_pins[10] : LOCATION = Pin_N15;
LCD_HEADER/interface_pins[1] : LOCATION = Pin_P17;
LCD_HEADER/interface_pins[2] : LOCATION = Pin_U1;
LCD_HEADER/interface_pins[3] : LOCATION = Pin_U2;
LCD_HEADER/interface_pins[4] : LOCATION = Pin_T2;
LCD_HEADER/interface_pins[5] : LOCATION = Pin_T3;
LCD_HEADER/interface_pins[6] : LOCATION = Pin_U4;
LCD_HEADER/interface_pins[7] : LOCATION = Pin_U19;
LCD_HEADER/interface_pins[8] : LOCATION = Pin_R18;
LCD_HEADER/interface_pins[9] : LOCATION = Pin_W20;
"LED2/in-LED1/in[0]" : LOCATION = Pin_T18;
"LED2/in-LED1/in[1]" : LOCATION = Pin_T19;
RESET_SWITCH/out : LOCATION = Pin_F12;
"SRAM_Hi/be_n-SRAM_Lo/be_n[0]" : LOCATION = Pin_F5;
"SRAM_Hi/be_n-SRAM_Lo/be_n[1]" : LOCATION = Pin_F2;
"SRAM_Hi/be_n-SRAM_Lo/be_n[2]" : LOCATION = Pin_F4;
"SRAM_Hi/be_n-SRAM_Lo/be_n[3]" : LOCATION = Pin_H5;
SRAM_Hi/cs_n : LOCATION = Pin_E2;
"SRAM_Hi/data-SRAM_Lo/data[0]" : LOCATION = Pin_C4;
"SRAM_Hi/data-SRAM_Lo/data[10]" : LOCATION = Pin_C5;
"SRAM_Hi/data-SRAM_Lo/data[11]" : LOCATION = Pin_D6;
"SRAM_Hi/data-SRAM_Lo/data[12]" : LOCATION = Pin_C6;
"SRAM_Hi/data-SRAM_Lo/data[13]" : LOCATION = Pin_F9;
"SRAM_Hi/data-SRAM_Lo/data[14]" : LOCATION = Pin_H10;
"SRAM_Hi/data-SRAM_Lo/data[15]" : LOCATION = Pin_D7;
"SRAM_Hi/data-SRAM_Lo/data[16]" : LOCATION = Pin_C7;
"SRAM_Hi/data-SRAM_Lo/data[17]" : LOCATION = Pin_E9;
"SRAM_Hi/data-SRAM_Lo/data[18]" : LOCATION = Pin_E10;
"SRAM_Hi/data-SRAM_Lo/data[19]" : LOCATION = Pin_D9;
"SRAM_Hi/data-SRAM_Lo/data[1]" : LOCATION = Pin_H11;
"SRAM_Hi/data-SRAM_Lo/data[20]" : LOCATION = Pin_C8;
"SRAM_Hi/data-SRAM_Lo/data[21]" : LOCATION = Pin_F10;
"SRAM_Hi/data-SRAM_Lo/data[22]" : LOCATION = Pin_G11;
"SRAM_Hi/data-SRAM_Lo/data[23]" : LOCATION = Pin_C9;
"SRAM_Hi/data-SRAM_Lo/data[24]" : LOCATION = Pin_C10;
"SRAM_Hi/data-SRAM_Lo/data[25]" : LOCATION = Pin_H12;
"SRAM_Hi/data-SRAM_Lo/data[26]" : LOCATION = Pin_D10;
"SRAM_Hi/data-SRAM_Lo/data[27]" : LOCATION = Pin_G12;
"SRAM_Hi/data-SRAM_Lo/data[28]" : LOCATION = Pin_G13;
"SRAM_Hi/data-SRAM_Lo/data[29]" : LOCATION = Pin_F11;
"SRAM_Hi/data-SRAM_Lo/data[2]" : LOCATION = Pin_G10;
"SRAM_Hi/data-SRAM_Lo/data[30]" : LOCATION = Pin_B11;
"SRAM_Hi/data-SRAM_Lo/data[31]" : LOCATION = Pin_B10;
"SRAM_Hi/data-SRAM_Lo/data[3]" : LOCATION = Pin_D8;
"SRAM_Hi/data-SRAM_Lo/data[4]" : LOCATION = Pin_E7;
"SRAM_Hi/data-SRAM_Lo/data[5]" : LOCATION = Pin_D4;

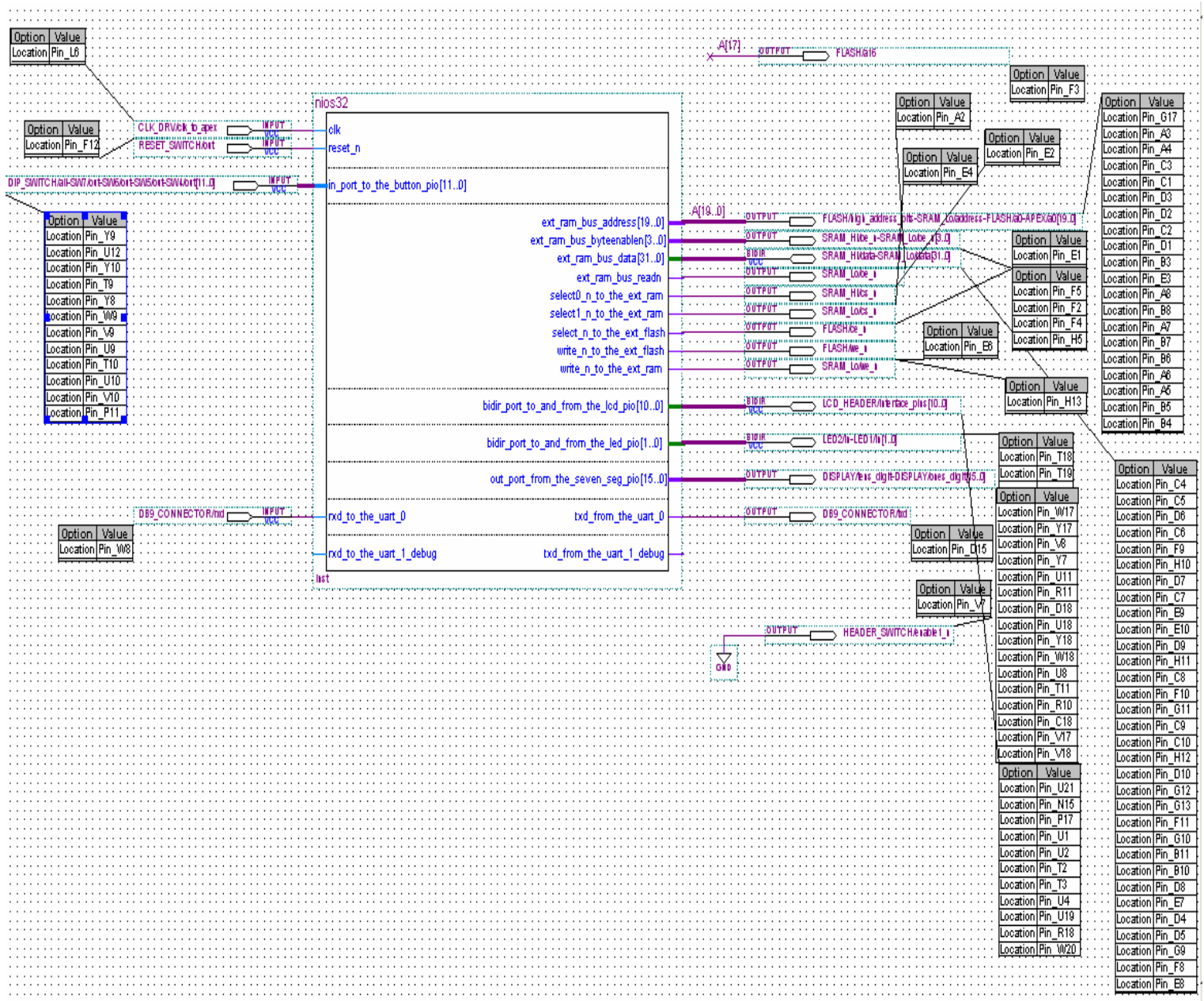
```

```
"SRAM_Hi/data-SRAM_Lo/data[6]" : LOCATION = Pin_D5;
"SRAM_Hi/data-SRAM_Lo/data[7]" : LOCATION = Pin_G9;
"SRAM_Hi/data-SRAM_Lo/data[8]" : LOCATION = Pin_F8;
"SRAM_Hi/data-SRAM_Lo/data[9]" : LOCATION = Pin_E8;
SRAM_Lo/cs_n : LOCATION = Pin_E4;
SRAM_Lo/oe_n : LOCATION = Pin_A2;
SRAM_Lo/we_n : LOCATION = Pin_E6;
```

Block Diagram:

The CPU interfaces with the external FLASH, seven segment LED display, the UART debug interface, the button interface and the LCD display see figure 2 for the details.

Figure 2: CPU Block diagram



Sorting Algorithm – source code:

//Initialization:

```
#include <stdio.h>
#define arr_size 10
#include "pio_lcd16207.h"
#include "nios.h"

// prototyping the functions
void selectionSort(int numbers[], int array_size);
void bubbleSort(int numbers[], int array_size);

void printArray(int nos[], int array_size, int scnt);
void outPutToLCD (int numbers[],int array_size, int swap_cnt, char []);

// global variable
int  oNum[]={3,2,4,1,7,8,6,9,5,0};
char bsort[]={'B','S'};
char ssort[]={'S','S'};
```

//Main module:

```
int main()
{
    int choice=0;
    int i,as;

    int arrNum[]={3,2,4,1,7,8,6,9,5,0};

    as= arr_size;

    nr_pio_lcdinit(na_lcd_pio); // initialize the lcd to display

    for(i=0;i<arr_size;i++)
        arrNum[i] = oNum[i];
    printf("Unsorted Array\n");
    printArray(arrNum,arr_size,choice);
    printf("Sorting using Bubble Sort\n");
    bubbleSort(arrNum,as);

    for(i=0;i<arr_size;i++)
        arrNum[i] = oNum[i];
    printf("Unsorted Array\n");
    printArray(arrNum,arr_size,choice);
    printf("Sorting using Selection Sort\n");
    selectionSort(arrNum,as);
} //main
```

//Selection Sort module:

```

void selectionSort(int numbers[], int array_size)
{
    int swpCnt=0;
    int i=0;
    int j;

    int min, temp;
    // printArray(numbers,array_size,i);
    for (i = 0; i <= array_size-1; i++)
    {
        min = i;
        for (j = i+1; j < array_size; j++)
        {
            if (numbers[j] < numbers[min])
                min = j;
        }
        swpCnt++;
        temp = numbers[i];
        numbers[i] = numbers[min];
        numbers[min] = temp;
        printArray(numbers,array_size,swpCnt);
        outPutToLCD(numbers,array_size,swpCnt,ssort);
        nr_delay(1400);
    }
}

```

// Bubble Sort module:

```

void bubbleSort(int numbers[], int array_size)
{
    int swpCnt=0;
    int i=0;
    int j, temp;
    // printArray(numbers,array_size,i);
    for (i = (array_size - 1); i >= 0; i--)
    {
        for (j = 1; j <= i; j++)
        {
            if (numbers[j-1] > numbers[j])
            {
                temp = numbers[j-1];
                numbers[j-1] = numbers[j];
                numbers[j] = temp;
                swpCnt++;
                printArray(numbers,array_size,swpCnt);
                outPutToLCD(numbers,array_size,swpCnt,bsort);
                nr_delay(1400);
            }
        }
    }
}

```

//LCD Write Routine:

```
void outPutToLCD (int n[],int a_size, int swp, char type[])
{ // form the 32 character string
  char str[128];

  sprintf(str,"arr%c%c:%d%d%d%d%d%d%d%d%d%dswp%d:%d%d%d%d%d%d%d%d%d",type[
0],type[1],
oNum[0],oNum[1],oNum[2],oNum[3],oNum[4],oNum[5],oNum[6],oNum[7],oNum[8],oN
um[9], swp,n[0],n[1],n[2],n[3],n[4],n[5],n[6],n[7],n[8],n[9]);
  nr_pio_lcdwritescreen(str);
}
```

//Print through UART to stdout:

```
void printArray(int nos[], int a_size, int scnt)
{
  int i;
  printf("swap count: %d\n",scnt);
  for(i =0;i<a_size;i++)
    printf("%d\t",nos[i]);

  printf("\n");
}
```

Results:

The simulation results in figure 3 confirm the performance comparison of the 2 sorting algorithms, bubble sort sorted the given array of numbers {3,2,4,1,7,8,6,9,5,0} in ascending order in 19 swaps while the Selection sort did the same in 10 swaps.

The sequence sort needs 'n' swaps in the worst case for sorting 'n' numbers. At every pass it finds the minimum value of the series and stores it in the first index and increments the index of the minimum to the next location. Bubble sort in comparison, compares adjacent elements for every pass and hence needs '(n-1)!' swaps in the worst case for sorting 'n' sequence numbers.

Figure 3: Sorting algorithm output

```

nios-run: Terminal mode <Control-C exits>
-----
Unsorted Array
swap count: 0
3      2      4      1      7      8      6      9      5      0
Sorting using Bubble Sort
swap count: 1
2      3      4      1      7      8      6      9      5      0
swap count: 2
2      3      1      4      7      8      6      9      5      0
swap count: 3
2      3      1      4      7      6      8      9      5      0
swap count: 4
2      3      1      4      7      6      8      5      9      0
swap count: 5
2      3      1      4      7      6      8      5      0      9
swap count: 6
2      1      3      4      7      6      8      5      0      9
swap count: 7
2      1      3      4      6      7      8      5      0      9
swap count: 8
2      1      3      4      6      7      5      8      0      9
swap count: 9
2      1      3      4      6      7      5      0      8      9
swap count: 10
1      2      3      4      6      7      5      0      8      9
swap count: 11
1      2      3      4      6      5      7      0      8      9
swap count: 12
1      2      3      4      6      5      0      7      8      9
swap count: 13
1      2      3      4      5      6      0      7      8      9
swap count: 14
1      2      3      4      5      0      6      7      8      9
swap count: 15
1      2      3      4      0      5      6      7      8      9
swap count: 16
1      2      3      0      4      5      6      7      8      9
swap count: 17
1      2      0      3      4      5      6      7      8      9
swap count: 18
1      0      2      3      4      5      6      7      8      9
swap count: 19
0      1      2      3      4      5      6      7      8      9
Unsorted Array
swap count: 0
3      2      4      1      7      8      6      9      5      0
Sorting using Selection Sort
swap count: 1
0      2      4      1      7      8      6      9      5      3
swap count: 2
0      1      4      2      7      8      6      9      5      3
swap count: 3
0      1      2      4      7      8      6      9      5      3
swap count: 4
0      1      2      3      7      8      6      9      5      4
swap count: 5
0      1      2      3      4      8      6      9      5      7
swap count: 6
0      1      2      3      4      5      6      9      8      7
swap count: 7
0      1      2      3      4      5      6      9      8      7
swap count: 8
0      1      2      3      4      5      6      7      8      9
swap count: 9
0      1      2      3      4      5      6      7      8      9
swap count: 10
0      1      2      3      4      5      6      7      8      9

```

Conclusion:

The labs 4 & 5 taught us to create the Nios softcore processor and enabled us to code in 'C' and run our programs in the 32 bit processor.

The bubble sort and selection sort algorithms were implemented and the sorting procedure was displayed in the 16X2 character LCD screen.