

Embedded Systems

Lab 8 – Requirements-MINI PROJECT

11/10/2004

Name: MOTOR SPEED CONTROL

Narayanan Krishnamurthy, Nisheet Gupta

Purpose: The system must be able to control the speed and direction of the motor included in the setup located in the lab. There must be a means for the operator to set the desired speed and direction and to observe the actual speed and direction.

Servo motors are motors with “brains” they are used in abundance in factory automation applications, robotics and precision machine tool control. DC servo motors has 2 inputs sources: the energy to drive it and the command source to tell it what to do. The mini-project to implement the servo motor controller for speed and direction enables us to understand its multifarious uses and applications in the industry.

Inputs:

```
//user inputs
getSpecifiedSpeed()           //user defined set speed
getSpecifiedDirection()      //user defined direction
speedCntlInput()             //input from tachometer for speed control
```

Outputs:

```
setSpecifiedSpeed()          //output to servo
setSpecifiedDirection()
speedCntlOutput()
```

Functions:

Implements a closed loop speed control of a DC servo motor. Using Pulse Width Modulation, the duty cycle of the input signal determines the average DC signal seen by the servo which in turn directly controls its speed. The servo speed control should minimize the deviation in actual speed from the set value, and system response must be comparable to commercial controllers.

Power:

- 6W power supply to the target board

Manufacturing Cost:

- FPGA target board \$299
- LCD Display \$25
- KeyPad \$10
- Power supply \$20
- Total \$354

Specifications Document for Servo Motor Control

November 22, 2004

Nisheet Gupta
Narayanan Krishnamurthy

Table of contents

Table of contents	2
Introduction:	3
Specifications:	3
Use Case:	3
Sequence diagram for viewing motor speed:	4
Sequence diagram for setting motor speed:	4
Collaboration diagram for viewing motor speed:	5
Collaboration diagram for setting motor speed:	6
Class diagram:	7
Testing Plan:	7

Introduction:

In the requirements section, the inputs, outputs, general functions, estimated power consumption, and estimated cost for the motor speed control project were defined from the user's point of view. In the specification section, a bridge is drawn between customer's requirements and the design architecture document. The specification should encompass all the requirements provided by the customer. UML notations for specifying the Stepper Motor Speed Control project, in terms of use cases, object extraction from nouns of the problem statement, sequence of data and control flow, helps in identifying modules and their interactions.

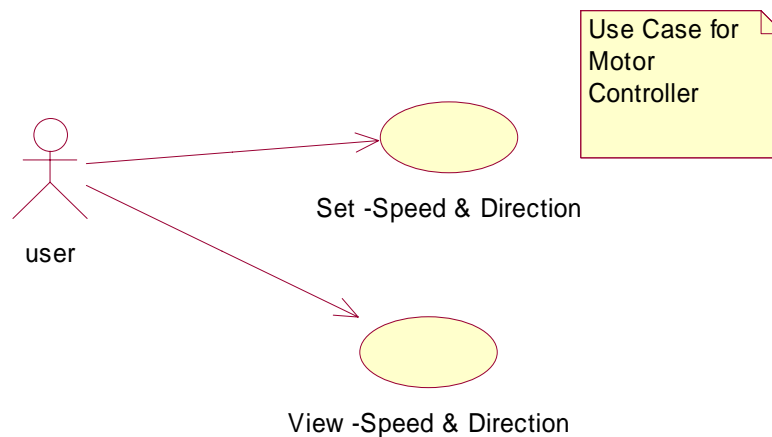
Specifications:

Input speed is in RPM, Display speed is in RPM

User has to specify direction (clockwise/anti-clockwise) and speed to change the speed of the stepper motor, the controller should take care of gradually increasing the motor speed to the new user defined speed.

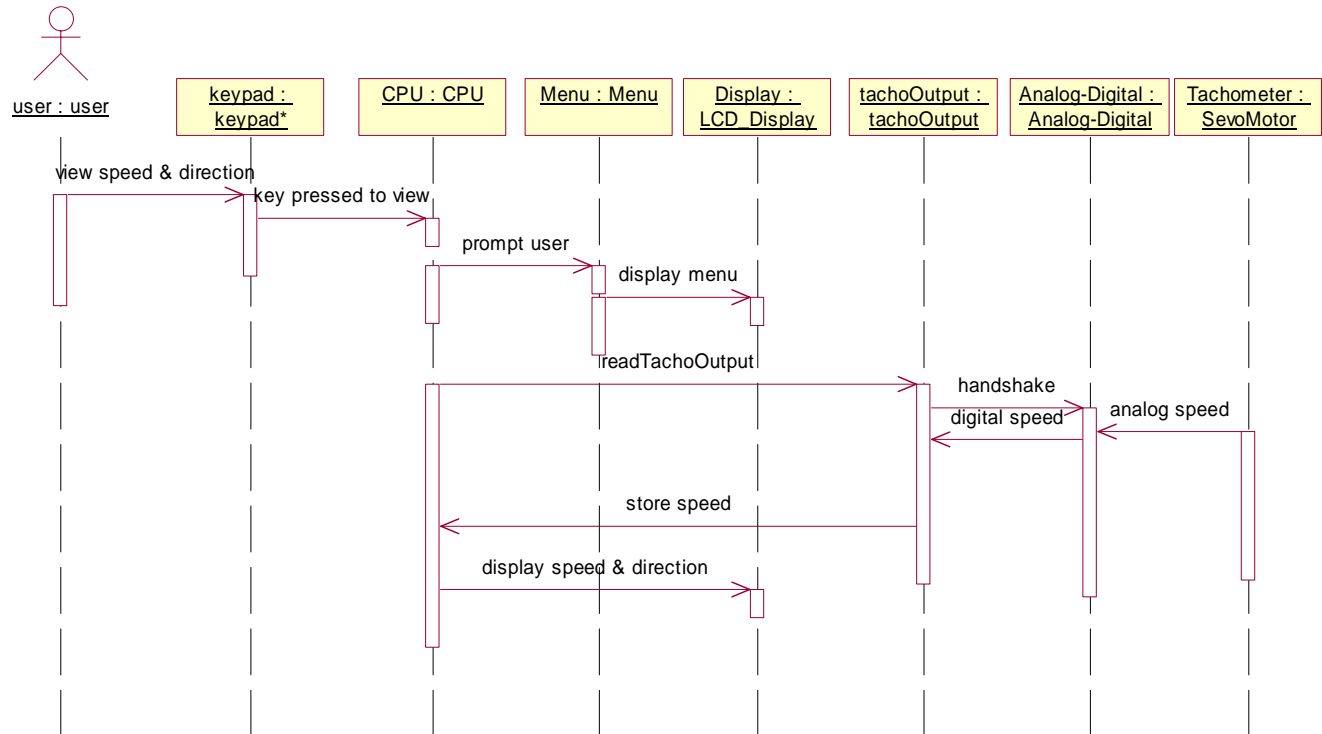
Use Case:

The use case for the stepper motor controller depicts the user interaction with the system, thus the user can view (speed and direction) and change (speed and direction of the motor). The user input is obtained through a keypad interface, where he can input a three digit speed (only a certain range of speeds are permissible, limited by the servo specifications)

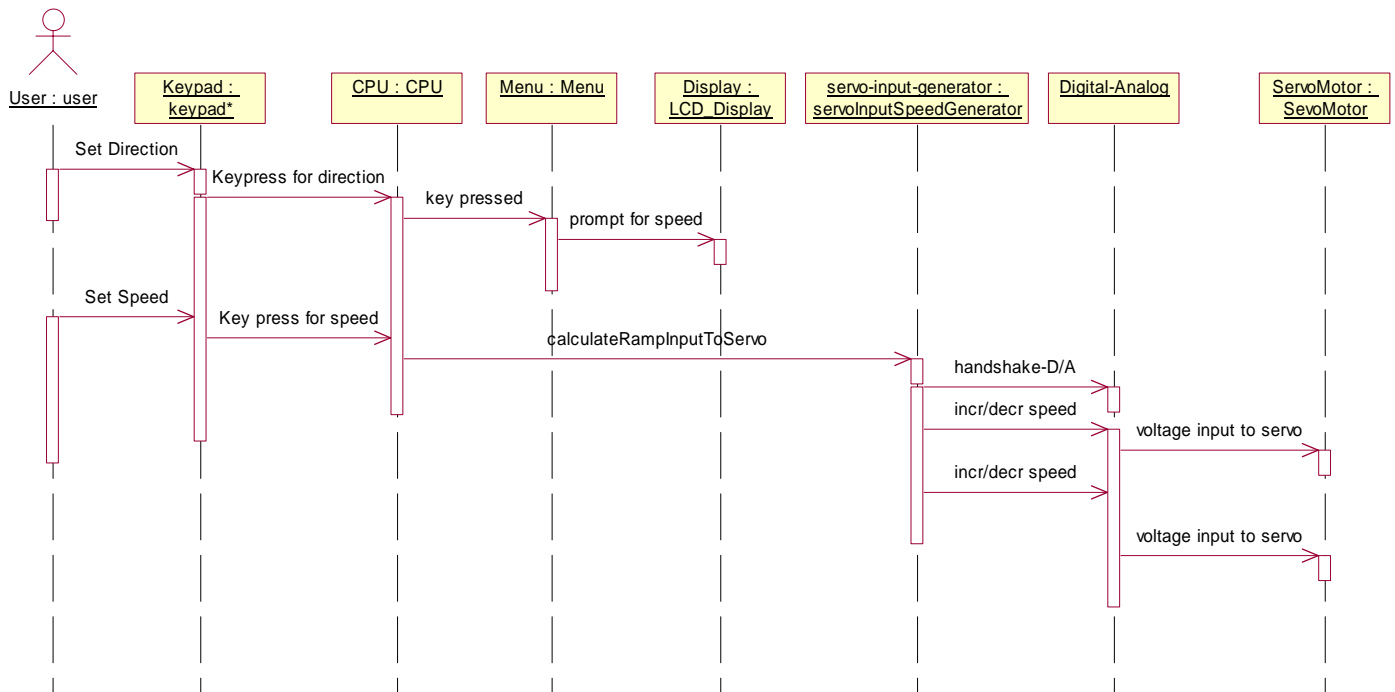


Sequence diagram for viewing motor speed:

Note: Explanation on control flow for sequence and collaboration diagrams given with the annotation for collaboration diagrams.

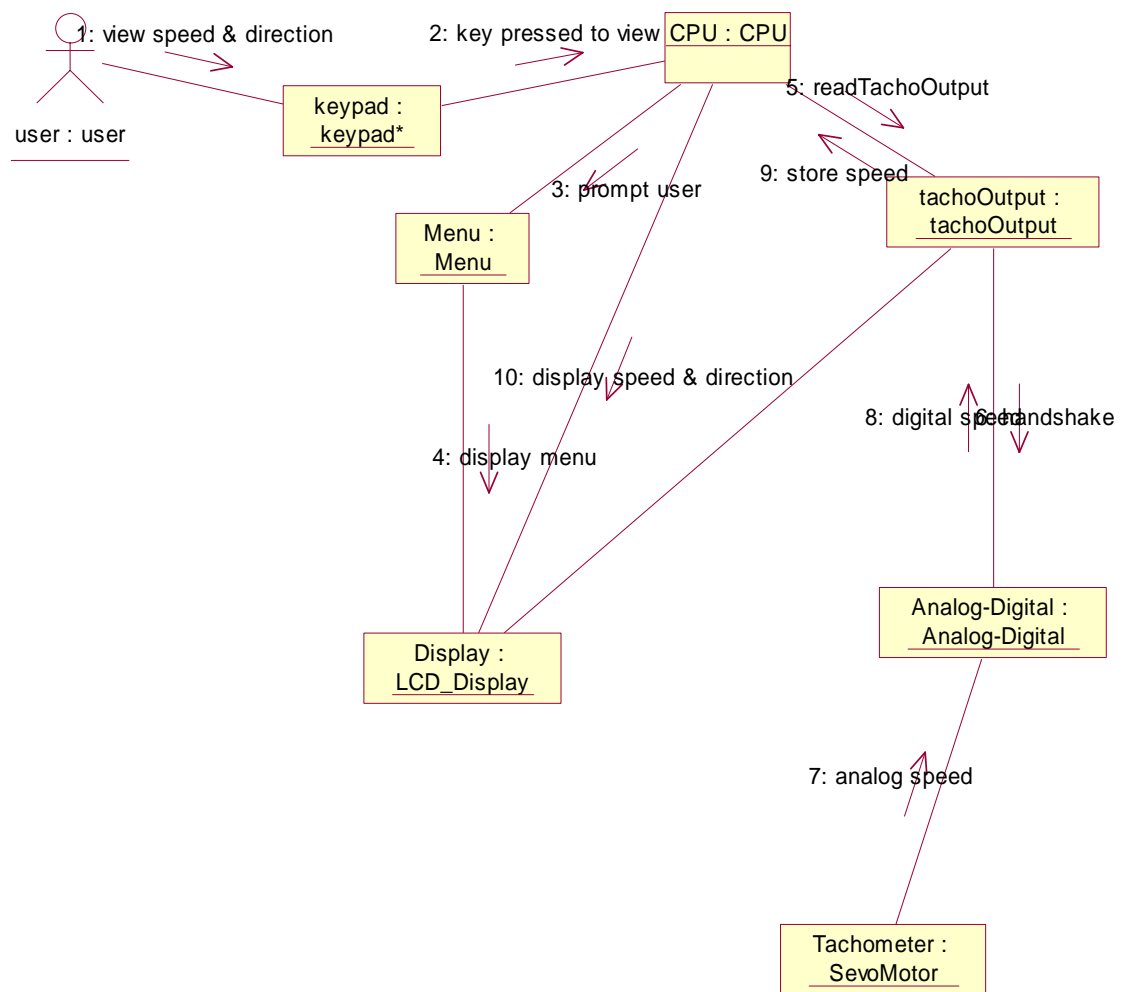


Sequence diagram for setting motor speed:



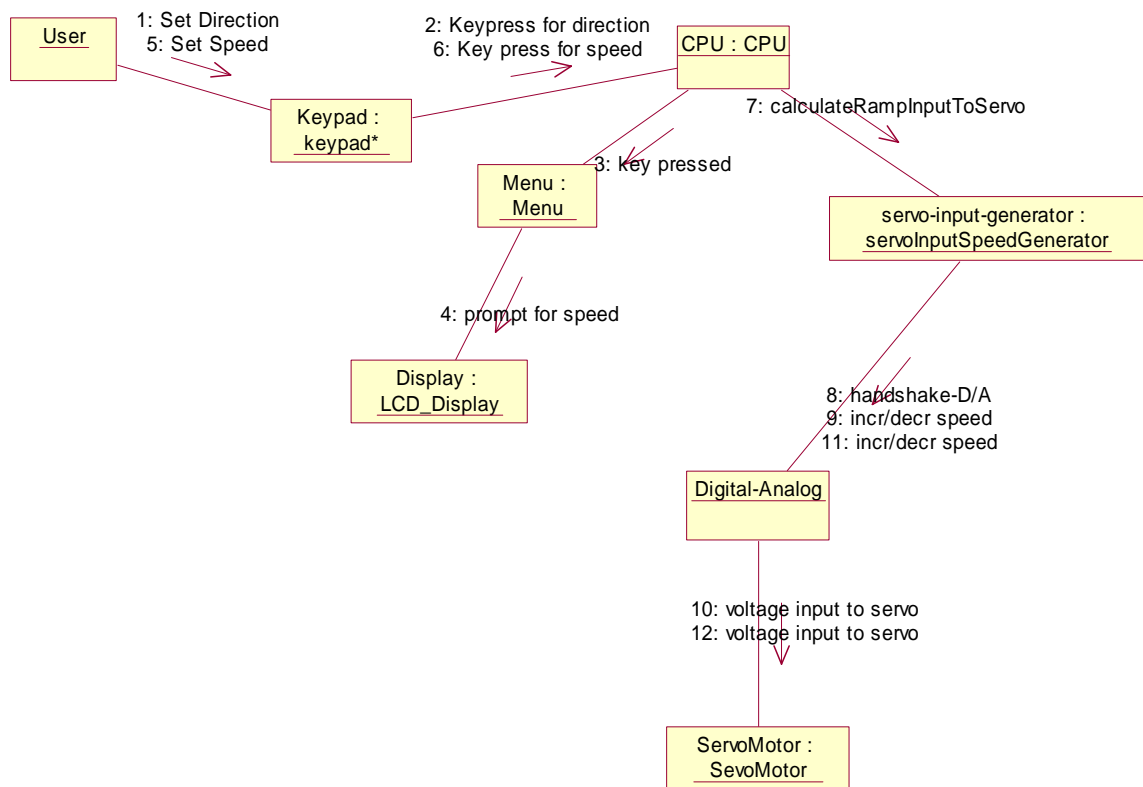
Collaboration diagram for viewing motor speed:

When the user decides to view the current speed and direction of the motor, he/she will press the corresponding key on the keypad. The menu class will handle the job of displaying the key press and the speed & direction on the LCD display. The key press information will also go to the tachoOutput class which will read data from the A/D. The A/D will provide the digital data that it received from the Tachometer to the tachoOutput.



Collaboration diagram for setting motor speed:

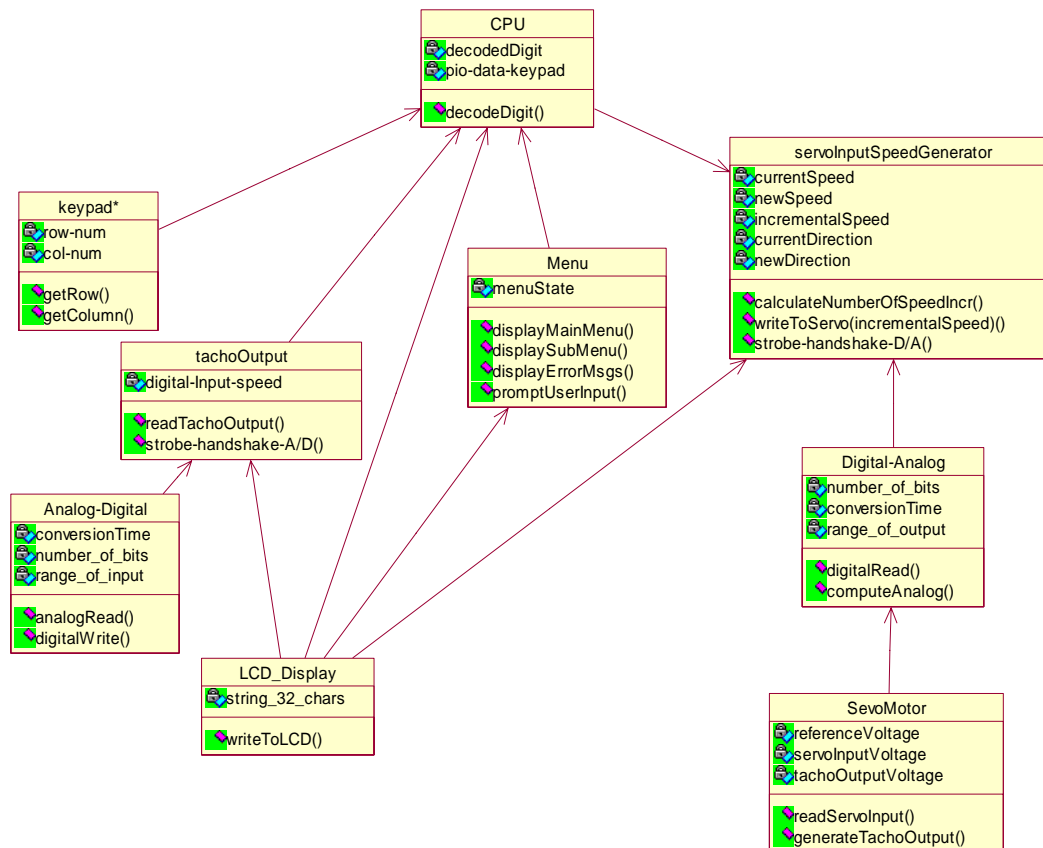
When the user decides to set the current speed and direction of the motor, he/she will press the corresponding key on the keypad. The menu class will handle the job of displaying the key press and the speed & direction on the LCD display. The key press information will also go to the servo-input-generator class which will read data from the D/A. The D/A will provide the analog data that it received from the ServoMotor class to the servo-input-generator class.



Class diagram:

Class diagram lists the objects in the system, and their hierarchy in composition. The attributes list the variables that characterize the class of objects. The classes define behavior or methods that define the interactions between the objects.

The class diagram for the servo-motor controller is given below:



Testing Plan:

1. Check if User input speed is validated
2. Check if Change of motor direction works
3. Check if set speed is achieved with 2% accuracy (takes care of resolution and quantization errors in D-A,A-D conversions)
4. Check for gradual step up of speed of motor to new speed
5. Check User-interface for proper menus, prompts and error messages

Design Document for Servo Motor Control

November 22, 2004

Nisheet Gupta
Narayanan Krishnamurthy

Table of contents

Table of contents.....	2
Introduction: Bridging specifications and design.....	3
Hardware Architecture:.....	3
Master Block Diagram.....	3
Component Function and Interaction.....	4
Software Architecture:.....	5
Class diagram:.....	5
State diagrams.....	6
Digit Handler and Menu State Machines.....	6
Program Flow: Pseudo-code/Flowcharts.....	6
Pseudo code for MAIN_MENU & ENT_DIR states.....	6
Program flow diagram in ENT_SPEED state.....	7
Pseudo code for Calculation of DACValue, DACbase, DACincr.....	8
Hardware Software Interface:.....	9

Introduction: Bridging specifications and design

In the specification section, a bridge was drawn between customer's requirements and the design architecture document. The specification encompassed all the requirements provided by the customer. The specification does not say how the system does things, only what the system does. Describing how the system implements those functions is the purpose of the design/architecture. The architecture is a plan for the overall structure of the system that will be used later to design the components that make up the architecture.

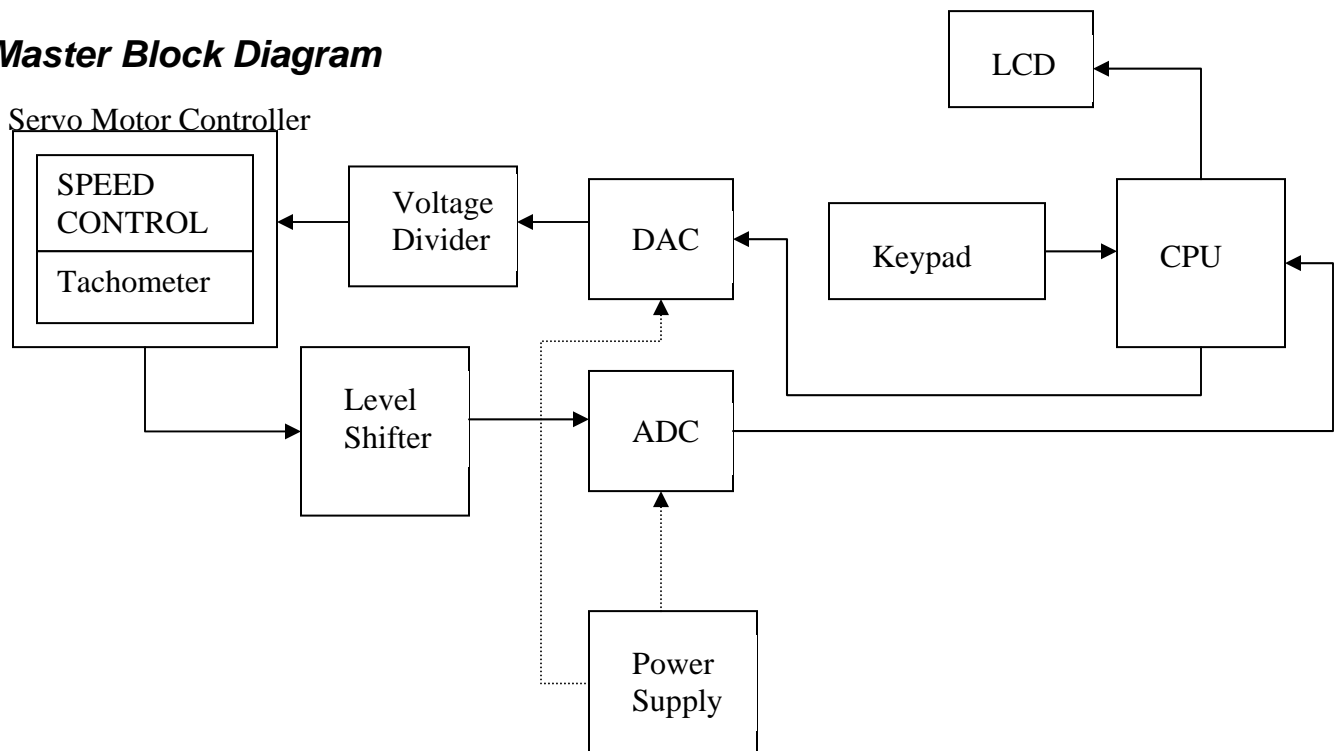
The system block diagram is refined into two block diagrams: one for hardware and another for software. The hardware block diagram will have a central CPU surrounded by memory and I/O devices. In our case the CPU will be the NIOS processor, and the I/O will be coming and going to the A/D converter, D/A converter, the keypad, and the servo motor controller. The software block diagram fairly closely follows the system block diagram.

Architectural descriptions are obtained used class diagrams. The classes for the overall architecture are obtained first and then the classes are split into hardware and software classes.

Hardware Architecture:

First, a master block diagram of the hardware, identifying major components was obtained. Please refer to the figure below for the block diagram

Master Block Diagram



Component Function and Interaction

CPU

CPU will act as the main I/O port between the user and the hardware components. CPU will be responsible for processing the 8-bit row and column binary data from the keypad and likewise output the speed and direction information onto the LCD display. CPU also displays error messages on the LCD display. Also, 8-bit data from the ADC goes to the data input of the CPU. By doing this, the CPU will be able to display the current motor speed on the LCD display.

Keypad

Keypad will accept input from the user and will output 8 bit data to the data line of the CPU. This 8 bit data will be used to decode the key press and eventually display it on the LCD display. Also, this 8 bit data will be used to control the speed and direction of the servo motor.

DAC

DAC will convert 8 bit data to an analog voltage signal. The CPU will provide 8 bit data to the input pins of the DAC which will convert this 8 bit data to a 0v-10v analog which will be used to control the speed of the motor. The DAC will be powered by a 10v power supply.

Voltage Divider

Because the servo motor is configured to operate in the range of 0v to 5v, the 0-10v output from the DAC will be converted to 0-5v using a voltage divider. A network of resistors will be used for the voltage conversion.

Servo Motor

The servo motor will take the analog voltage signal from the DAC and will likewise adjust the speed and direction of the motor. The servo motor is composed of two hardware entities:

- Speed control: The analog signal from the DAC will be used to control the speed of the motor.
- Tachometer: The tachometer will output the current speed of the servo motor. This analog output will be the input to the level shifter.

Level Shifter

The level shifter will be used to shift the -16v to 16v analog output from the tachometer to 0v-5v signal. This needs to be done so as to be a proper input to the ADC. Also, the level shifter will be provided with a constant 16v input. An op-amp and a network of registers will be used for this conversion.

ADC

The ADC will be responsible for converting analog voltage to digital data. The analog signal from the level shifter component will be input to the ADC. 8-bit data from the ADC goes to the data input of the CPU. The 8-bit data will then be used by the CPU to display the current motor speed on the LCD display.

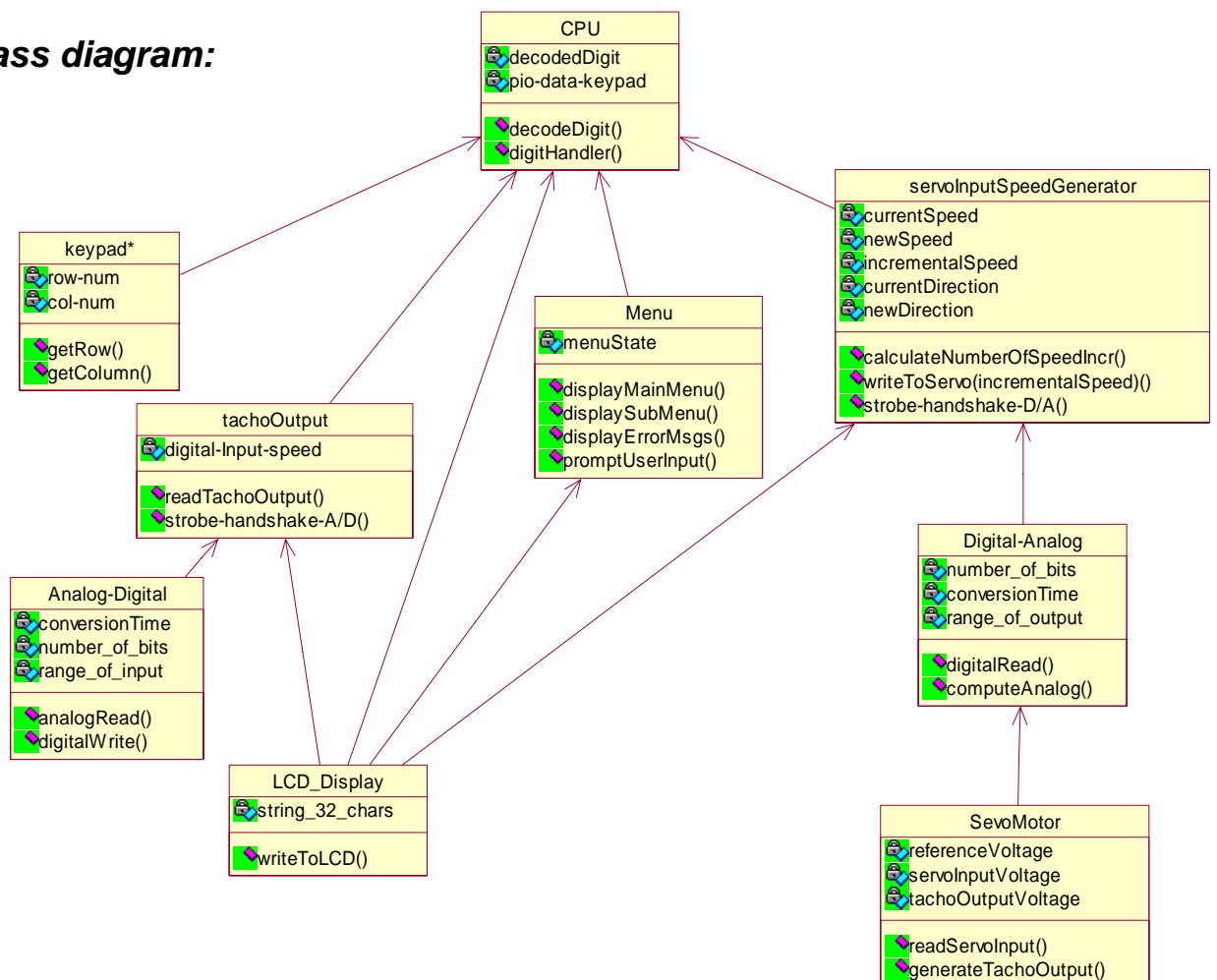
Power Supply

A power supply will be used to power the ADC, DAC, and the level shifter. ADC and DAC will be supplied with 10v while level shifter will receive a 16v input.

Software Architecture:

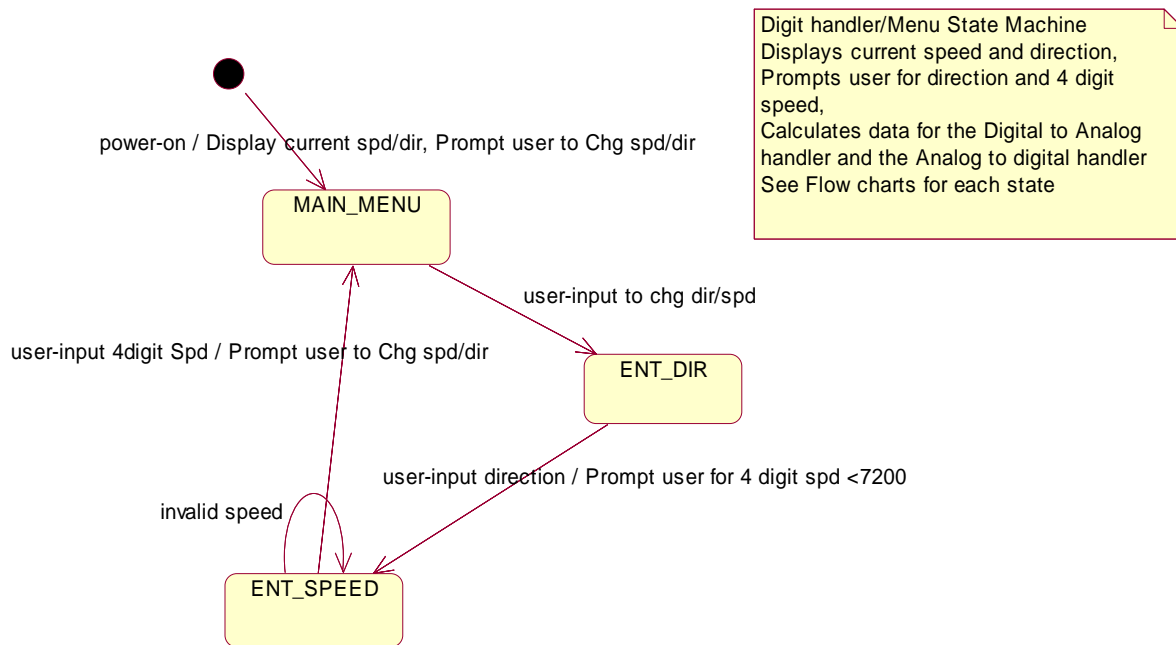
Identifying the Software architecture began with the Class Diagram of the specifications document. By identifying the objects (instantiation of the base class) one identifies the modules (real objects map to hardware realization while abstract objects to software modules) and their interactions.

Class diagram:



State diagrams

Digit Handler and Menu State Machines



Program Flow: Pseudo-code/Flowcharts

The program flow for the 3 states indicated in the state diagram are given below:

Pseudo code for MAIN_MENU & ENT_DIR states

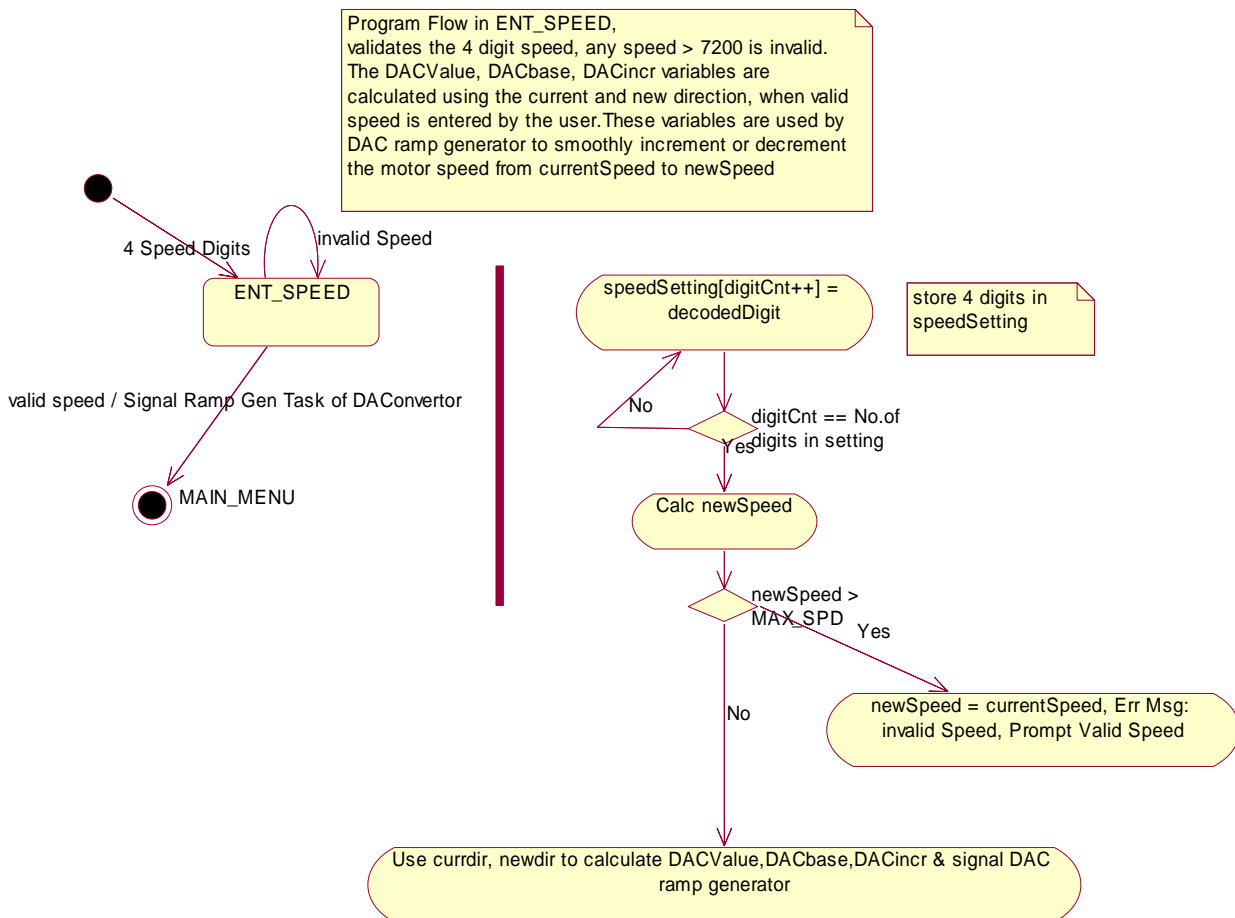
```

if(decodedDigit == CLOCKWISE)
{
    newdir = 1;
    state = ENT_SPD;
}
else if(decodedDigit == ANTICLOCK)
{
    newdir = 0;
    state = ENT_SPD;
}
else
{
    Prompt User ("Bad Direction,
    currentSpeed, currentDirection");
}
  
```

```

if(decodedDigit == CHG_SPD_DIR)
{
    state = ENT_DIR;
    Prompt User("enter 8-clockwise 9-anticlock Direction");
}
else
{
    Prompt User("Incorrect Choice",
    currentSpeed, currentDirection);
}
  
```

Program flow diagram in ENT_SPEED state



Pseudo code for Calculation of DACValue, DACbase, DACincr...

```
//NOTE: currrdir, newdir holds 0 –Anticlock 1-Clockwise,
//      currentSpeed,newSpeed holds 4-digit speed < MAX_SPEED

#define MAX_SPEED 7200
#define SPEED_RESOL 128
if(currrdir == newdir)
{
    if(currrdir) //clockwise
    {
        DACbase=SPEED_RESOL+((currentSpeed*SPEED_RESOL)/MAX_SPEED);
        if(newSpeed>currentSpeed)
        {
            DACValue=(((newSpeed-currentSpeed)*SPEED_RESOL)/MAX_SPEED);
            DACincr=0; // to incr speed
        }
        else
        {
            DACValue=(((currentSpeed - newSpeed)*SPEED_RESOL)/MAX_SPEED);
            DACincr=1; // to dcr speed
        }
    }
else //anticlock
{
    DACbase=SPEED_RESOL-((currentSpeed*SPEED_RESOL)/MAX_SPEED);
    if(newSpeed>currentSpeed)
    {
        DACValue=(((newSpeed-currentSpeed)*SPEED_RESOL)/MAX_SPEED);
        DACincr=1; // to incr speed
    }
    else
    {
        DACValue=(((currentSpeed - newSpeed)*SPEED_RESOL)/MAX_SPEED);
        DACincr=0; // to dcr speed
    }
}
}
else // direction is different
{
    if(newdir) // clockwise
    { //current direction anticlock
        DACbase=SPEED_RESOL-((currentSpeed*SPEED_RESOL)/MAX_SPEED);
        DACValue= SPEED_RESOL +((newSpeed*SPEED_RESOL)/MAX_SPEED) -DACbase;
        DACincr=0; //incr to new speed
    }
else // anticlockwise
{
    //current direction is clockwise
    DACbase=SPEED_RESOL+((currentSpeed*SPEED_RESOL)/MAX_SPEED);
    DACValue = DACbase - SPEED_RESOL +((newSpeed*SPEED_RESOL)/MAX_SPEED);
    DACincr=1; //decr to newspeed
}
}
}
```

Hardware Software Interface:

The hardware software interfacing comprises of modules that directly talk with the hardware components mentioned in the master block diagram. Note the hardware inputs and outputs to subsequent stages are modified/conditioned using voltage dividers and level shifters as indicated in the *Component Function and Interaction* section.

The software modules that interface the hardware elements viz. Keypad, A/D and D/A convertors take care of the setup/initialization(includes Chipselect/Chipenable logic generation) and also implements other handshake, data conditioning specific for the communicating hardware.

The keypad interacts with the keypad_pio_ISR_handler, while the Analog to Digital and Digital to Analog Convertors would interact with their respective interface modules that take care of data conditioning and generation to ramp the speed of the motor to the new speed smoothly and to display the speed measured from the tachometer.

Implementation & Testing Document for Servo Motor Control

December 10, 2004

Nisheet Gupta
Narayanan Krishnamurthy

Table of Contents

Table of Contents	2
Introduction	3
Preliminary Investigation	3
Input Calibration.....	3
Output Calibration	3
Hardware Implementation	3
Voltage Dividers for the Servo Speed Controller.....	4
Level Shifter	5
Wiring Schematic for DAC	7
Wiring Schematic for DAC	8
Quartus Block Diagram(s).....	9
PIO Creation in CPU for ADC and DAC.....	10
New Pin Assingments.....	10
Pin Assignments for DAC and ADC.....	11
Software Implementation	11
Description on Software architecture	11
//lab8.h header file	12
//OS_CPU_C.C TimerTickHook.....	13
//Lab8.c Main file that handles 3 tasks and 2 ISR's	14
Testing and Results.....	22
Testing DAC- Incremental speed change and Analog Voltage ramping to Servo.....	23

Implementation & Testing

Introduction

So far, the requirements, specifications, and design/architecture of the motor speed control system were obtained. After the hardware and software had been identified in the design/architecture part of the design flow, it is time to do the implementation and testing to actually verify the correctness and functionality of the design/architecture. The hardware implementation of this section will show the physical connections, screenshots from Quartus, and the real VHDL code. This will also include circuit schematics for the external analog components that were used and how they connect to the Excalibur board and other components. The software implementation section will provide the real C code that runs. Also, a listing of all tests that were performed and the results will be provided.

Preliminary Investigation

The process of designing the speed controller for the stepper motor is as follows:

1. The motor was first Calibrated, i.e. the input and output characteristics were determined.

Input Calibration

2. A reference **Vref** of 2.5 voltage was chosen and the -ve input to the differential amplifier of the stepper motor, and a **Vin** voltage change [0-5v] to the +ve terminal was chosen as the dynamic range such that, the **motor rotates clockwise** for **Vin > Vref [2.5 – 5V]** and for **Vin < Vref [0-2.5V]** it **rotates anticlockwise**. **Vref = Vin = 2.5V** the **motor stops rotating**.

Output Calibration

3. The Tacho output was measured for the **Vin[0-5V]** **Vtacho** output was found to be in the range of **[-16 to 16V]**

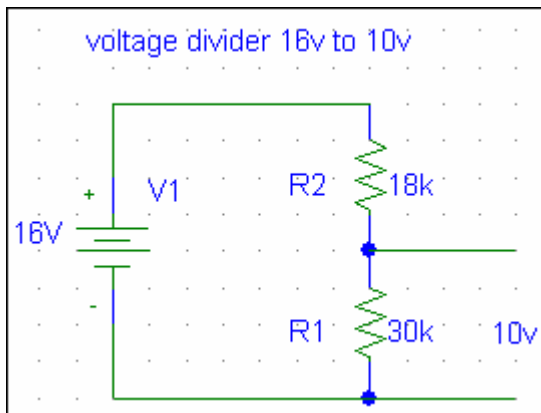
Hardware Implementation

The calibration process helped us identify the need for a level shifter to bring the tacho output to the operating range of the A/D converter determined by **[-Vref(gnd) to Vref+(5V)]**

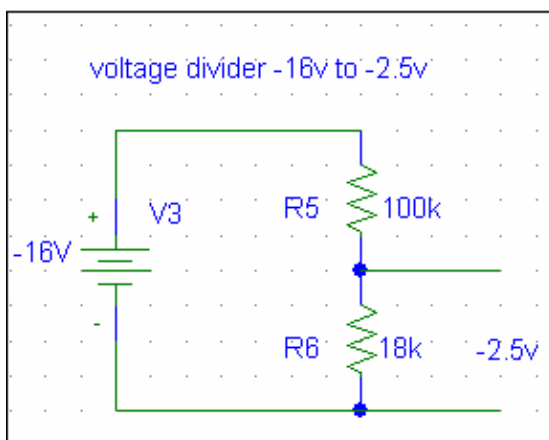
The Circuits were designed to operate from a single **power supply voltages of $\pm 16V$** , therefore a number of voltage dividers were designed to bring the interfacing voltages to the correct ranges. The circuit diagrams for the level shifter(an opamp summing amplifier) and the voltage divider circuits are given hereon.

There were a total of four voltage dividers that were implemented to obtain a particular output voltage from an input voltage. The schematics for the voltage dividers and there usage is described below:

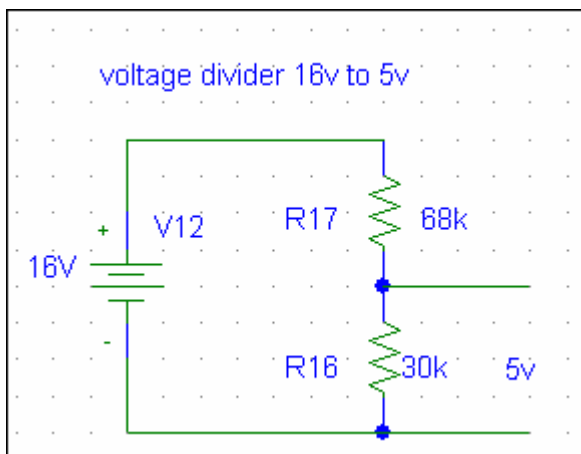
Voltage Dividers for the Servo Speed Controller



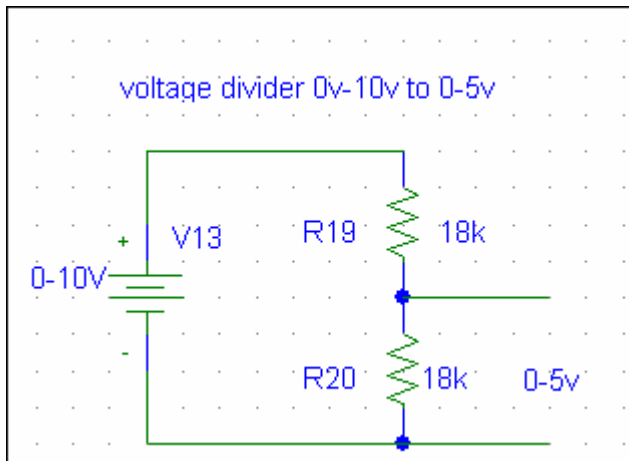
The 16v to 10v voltage divider was used to provide 10v supply to the DAC



The -16v to -2.5v was used to create Vref for the tachometer



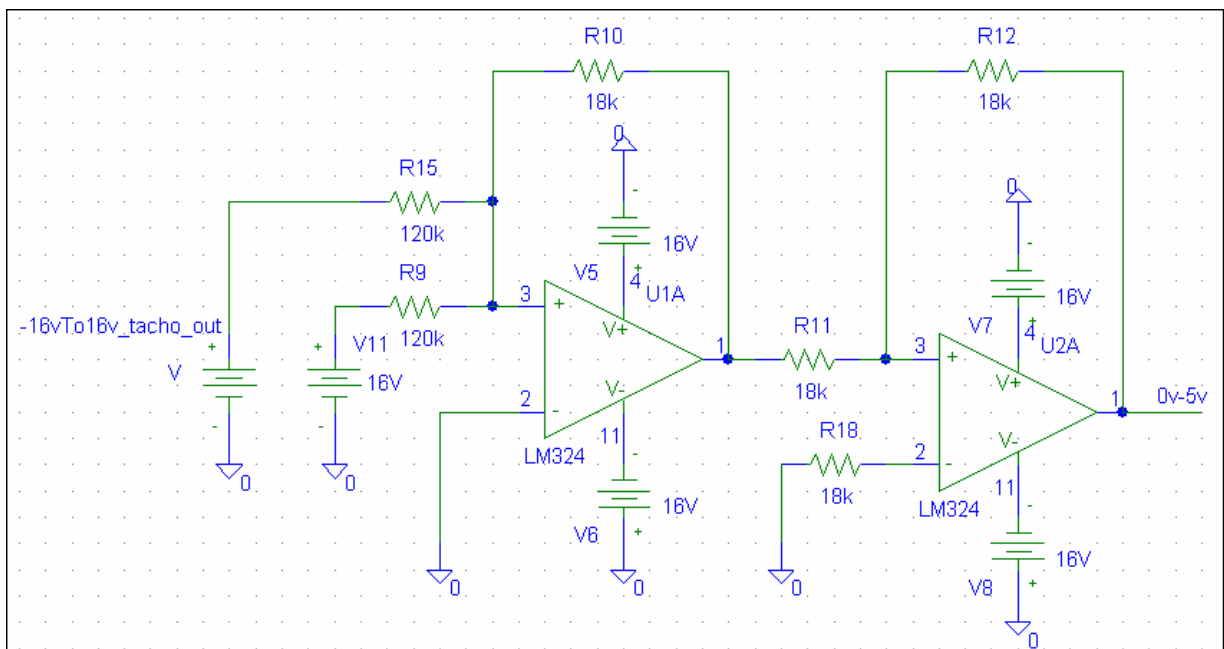
The 16v to 5v was used for creating 5v for Vref of ADC



The 0-10v output from the DAC is converted to 0-5v for creating a 2.5vref for the speed control.

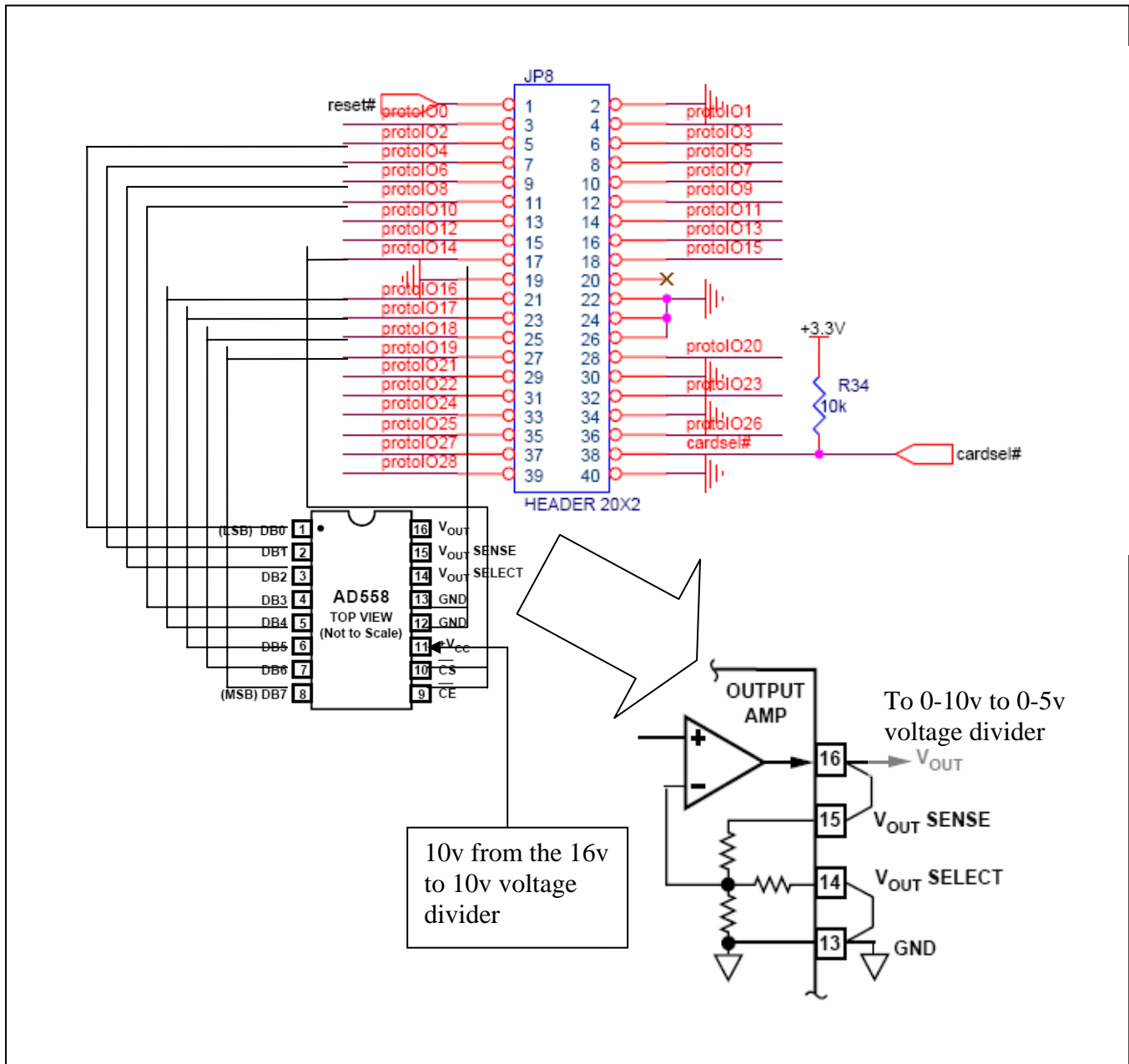
Level Shifter

The output of the tachometer was from -16v to 16v. A level shifter was implemented to convert this to 0v to 5v range. For this, an LM348 op-amp (labeled as LM324 in the schematic below) was used. Since the chip has 4 internal op-amps, a summing amplifier was created first and the output from it was input to an inverting amplifier. This was done since it was noticed that the output of the summing amplifier was inverting.



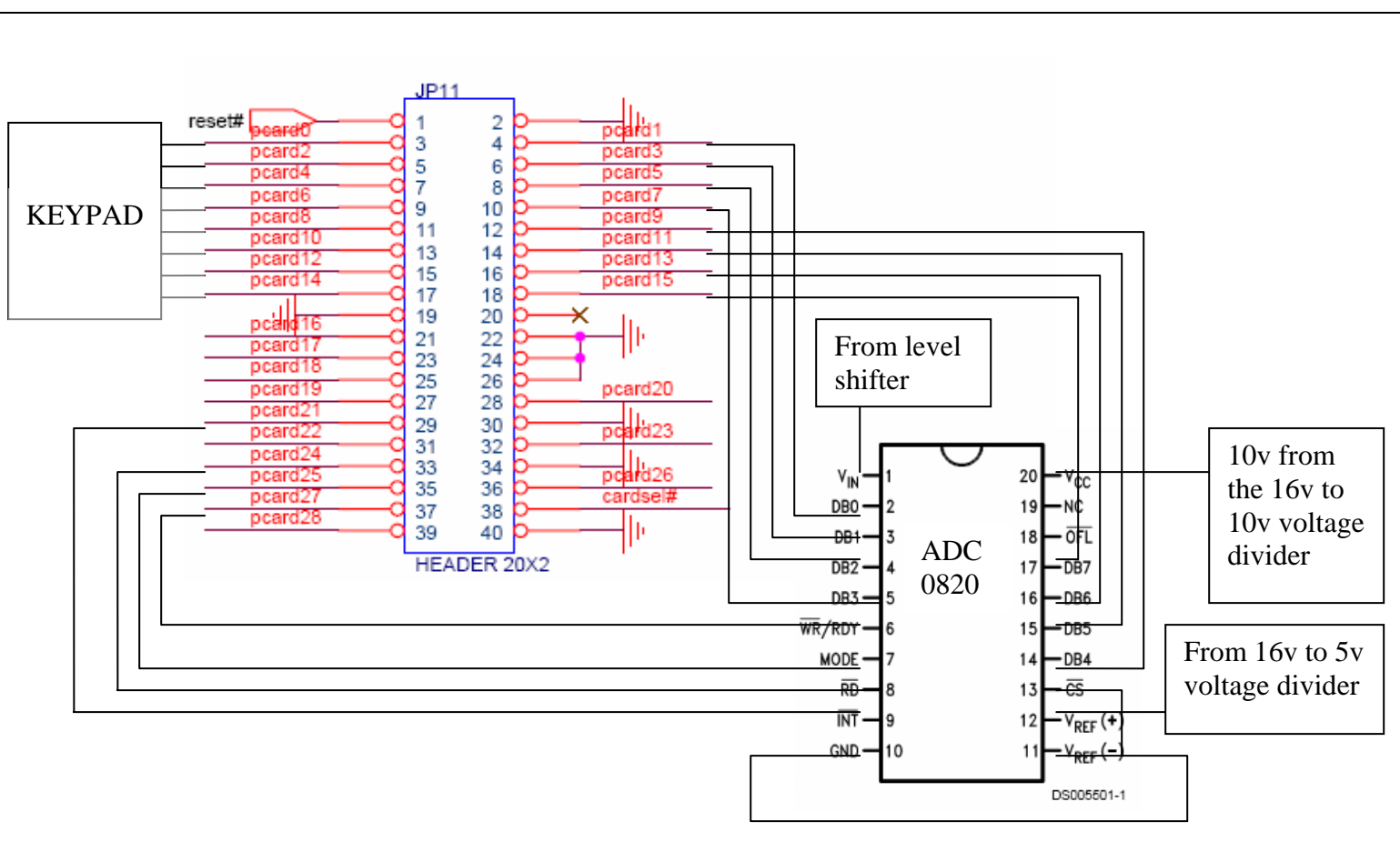
Wiring Schematic for DAC

The AD558 DAC was connected to the JP8 header as shown below. The 0-7 bits of digital input were connected between the JP8 and the DAC. To obtain the 0-10v output from the DAC, pin 15 and 16 as well as pin 13 and 14 were connected to each other. The output voltage was tapped from pin 16 whereas pin 13 was put to ground.

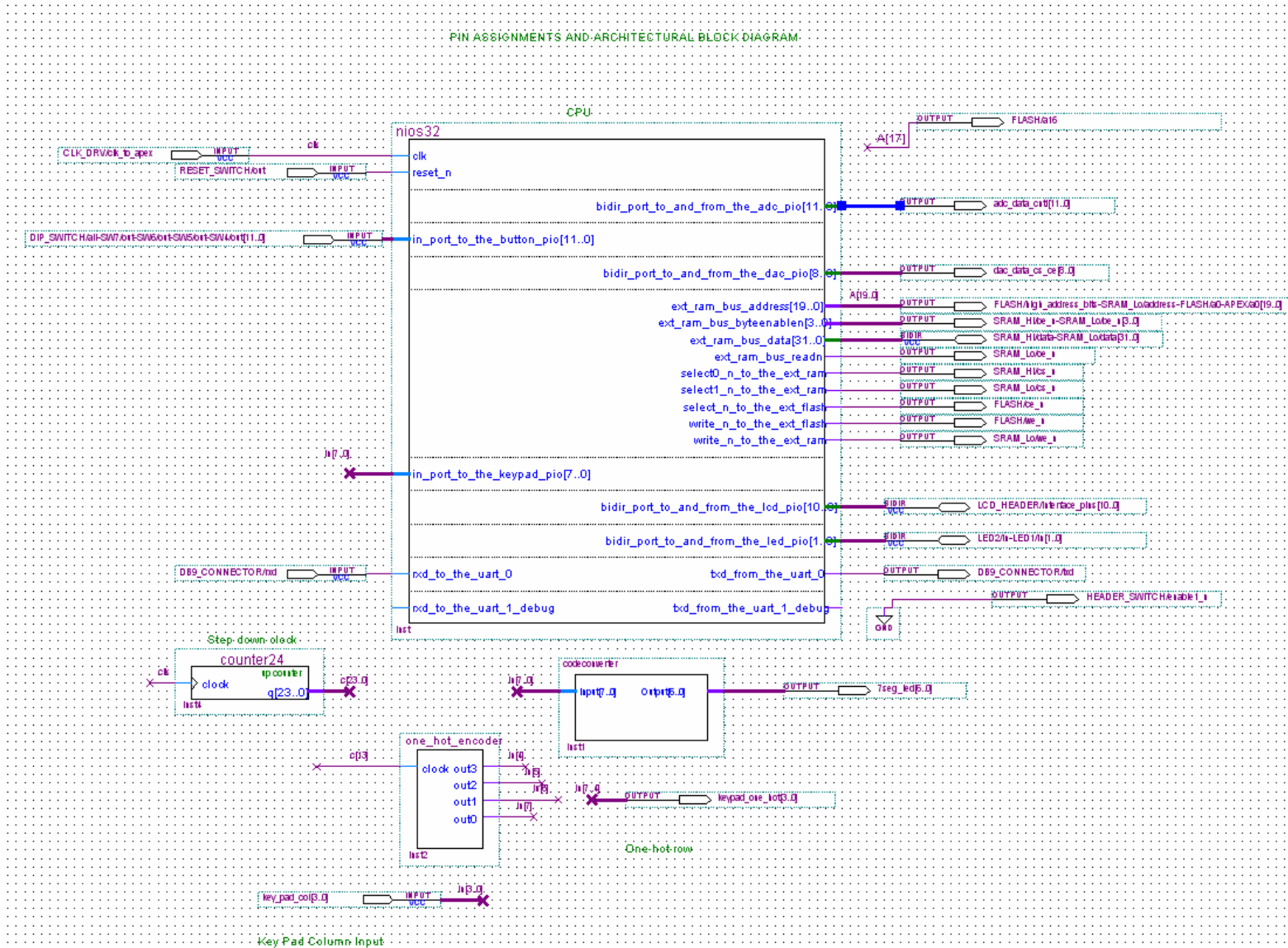


Wiring Schematic for DAC

The ADC0820 ADC was used to convert the analog input from the level shifter to the corresponding bits. Pin V_{in} was connected to the output of the level shifter and the 0-7 bit outputs were connected to the JP11. Keypad was also connected to the JP11 pins. The V_{ref} (+) pin was connected to 5v output from the voltage divider. V_{ref} (-) was connected to the ground.



Quartus Block Diagram(s)



PIO Creation in CPU for ADC and DAC

Altera SOPC Builder - nios32

File System Module View Help

System Contents **Nios** More "cpu" Settings System Generation

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

Use	Module Name	Description	Base	End	IRQ
<input checked="" type="checkbox"/>	boot_monitor_rom	On-Chip Memory (RAM or ROM)	0x00000000	0x000003FF	
<input checked="" type="checkbox"/>	cpu	Nios Processor - Altera Corporation			
<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"/>	timer1	Interval timer	0x00000440	0x0000045F	18
<input checked="" type="checkbox"/>	button_pio	PIO (Parallel I/O)	0x00000460	0x0000046F	19
<input checked="" type="checkbox"/>	keypad_pio	PIO (Parallel I/O)	0x000004A0	0x000004AF	20
<input checked="" type="checkbox"/>	adc_pio	PIO (Parallel I/O)	0x000004B0	0x000004BF	21
<input checked="" type="checkbox"/>	dac_pio	PIO (Parallel I/O)	0x00000490	0x0000049F	
<input checked="" type="checkbox"/>	ext_flash	AMD 29LV800 Flash for EP20K200E Nios Development Bo...	0x00100000	0x001FFFFFF	
<input checked="" type="checkbox"/>	ext_ram	IDT71V016 SRAM for EP20K200E Nios Development Board	0x00040000	0x0007FFFF	
<input checked="" type="checkbox"/>	ext_ram_bus	Avalon Tri-State Bridge			
<input checked="" type="checkbox"/>	lcd_pio	PIO (Parallel I/O)	0x00000470	0x0000047F	
<input checked="" type="checkbox"/>	led_pio	PIO (Parallel I/O)	0x00000480	0x0000048F	

All Available Components

Add... Check

Move Up Move Down

Done checking for updates.

Exit < Prev Next > Generate

Pin Assignments for DAC and ADC

CHIP(nios_system_module)

{ The portion below was added to the previous pin assignments:

```

dac_data_cs_ce[0] : LOCATION = Pin_P4;
dac_data_cs_ce[0] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[1] : LOCATION = Pin_V11;
dac_data_cs_ce[1] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[2] : LOCATION = Pin_K19;
dac_data_cs_ce[2] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[3] : LOCATION = Pin_N22;
dac_data_cs_ce[3] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[8] : LOCATION = Pin_P21;
dac_data_cs_ce[8] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[4] : LOCATION = Pin_P20;
dac_data_cs_ce[4] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[5] : LOCATION = Pin_K15;
dac_data_cs_ce[5] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[6] : LOCATION = Pin_R5;
dac_data_cs_ce[6] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[6] : IO_STANDARD = LVTTTL;
dac_data_cs_ce[7] : LOCATION = Pin_N6;
dac_data_cs_ce[7] : IO_STANDARD = LVTTTL;
adc_data_cntl[0] : LOCATION = Pin_P19;
adc_data_cntl[0] : IO_STANDARD = LVTTTL;
adc_data_cntl[1] : LOCATION = Pin_R20;
}

adc_data_cntl[1] : IO_STANDARD = LVTTTL;
adc_data_cntl[2] : LOCATION = Pin_N1;
adc_data_cntl[2] : IO_STANDARD = LVTTTL;
adc_data_cntl[3] : LOCATION = Pin_M2;
adc_data_cntl[3] : IO_STANDARD = LVTTTL;
adc_data_cntl[4] : LOCATION = Pin_T22;
adc_data_cntl[4] : IO_STANDARD = LVTTTL;
adc_data_cntl[5] : LOCATION = Pin_T20;
adc_data_cntl[5] : IO_STANDARD = LVTTTL;
adc_data_cntl[6] : LOCATION = Pin_T21;
adc_data_cntl[6] : IO_STANDARD = LVTTTL;
adc_data_cntl[7] : LOCATION = Pin_P1;
adc_data_cntl[7] : IO_STANDARD = LVTTTL;
adc_data_cntl[8] : LOCATION = Pin_T1;
adc_data_cntl[8] : IO_STANDARD = LVTTTL;
adc_data_cntl[9] : LOCATION = Pin_U3;
adc_data_cntl[9] : IO_STANDARD = LVTTTL;
adc_data_cntl[9] : IO_STANDARD = LVTTTL;
adc_data_cntl[10] : LOCATION = Pin_R2;
adc_data_cntl[10] : IO_STANDARD = LVTTTL;
adc_data_cntl[11] : LOCATION = Pin_N16;
adc_data_cntl[11] : IO_STANDARD = LVTTTL;

```

Software Implementation

Description on Software architecture

The modules for the software were identified from the class diagram given in the specifications document.

3 multi tasks were identified, the menu-task, DAC-task and ADC-task, these tasks were created with DAC with highest priority 1, as it generates the voltage for the speed control, the ADC task has priority 2 – it reads the tacho output and calculates the speed corresponding to the analog voltage read by the ADC which converts it to bits 0-255 bits. The ADC ISR handler retrieves the digital value of the speed when the INT line of ADC triggers the interrupt.

Both the DAC and ADC tasks run only when the user has requested a speed change, The DAC task generates the ramp output to the motor, thus taking the motor from the current speed to the new speed gradually. **See Results of the DAC output in the section Testing DAC- Incremental speed change and Analog Voltage ramping to Servo Servo**

The ADC task is scheduled the same time the DAC increment speed task is scheduled, therefore the ADC measures the incremented speed and displays it on the LCD. **Note: the user would not be able to modify motor speed when the speed of the motor is being changed.(This feature is implemented in software)**

The KeyPad isr processes the decodedDigit while the ADC isr calculates the digital speed from the measured voltage of the tacho output and displays it on the LCD.

//lab8.h header file

```
// ISR related

#define PIO_DATA_MASK 0xFF
#define PIO_DATA_ROW_MASK 0xF0 //rows on msn
#define PIO_DATA_COL_MASK 0x0F //columns on lsn

#define ADC_DATA_MASK 0x0FF
#define ADC_ISR_MASK 0x800 //1000 0000 0000
#define ADC_MODE_MASK 0x100 //BIWISE OR 0001 0000 0000
#define ADC_CS_WREN_MASK 0x9FF //BITWISE & 1001 1111 1111
#define ADC_CS_WRDIS_MASK 0x400 //BITWISE OR 0100 0000 0000

#define TWR 200 // min 600 ns, 30ns clock a for loop

// note row msn col msn make this up
//r4--1,c4--1
#define ONE 17 //00010001
#define TWO 18 //00010010
#define THREE 20 //00010100
#define FOUR 33 //00100001
#define FIVE 34 //00100010
#define SIX 36 //00100100
#define SEVEN 65 //01000001
#define EIGHT 66 //01000010
#define NINE 68 //01000100
#define ZERO_DIGIT 130 //10000010

#define CLOCKWISE 81
#define ANTICLOCK 82
#define NO_DIRECTION 83

#define MAIN_MENU 91
#define ENT_DIR 92
#define ENT_SPD 93
#define SPD_BEING_CHGD 94

#define MAX_SPEED 7200
#define SPEED_RESOL 128
#define NO_DIGITS_IN_SETTING 4

#define CS_CE_ENA_MASK 0x0FF
#define LATCH_DAC_MASK 0x1FF

//define VOL_PER_BIT 0.039 // for 8 bit representation for 0-10volt NOT USED
```

//OS_CPU_C.C TimerTickHook

```

/*
*****
*
*           TICK HOOK
*****
*/
static int DACTick   = 40;    // run every 200 msecs
static int menuTick  = 200;   // run every 1 secs
static int ADCTick   = 10;    // run every 100 msecs
static int pioTick   = 150;   // hold rcv_pio for 0.75 secs

void OSTimeTickHook (void)
{
  ONT_EXT OS_EVENT *schADC;
  ONT_EXT OS_EVENT *schMenu;
  ONT_EXT OS_EVENT *schDAC;

  extern int rcvd_pio;
  extern int DACValue;
  extern int lastSpeedRead;

  if(rcvd_pio)      // to reset rcv_pio_data only after 150 ticks
    pioTick--;
  if(!pioTick)
  {
    rcvd_pio = 0;  // reset rcv_pio_data to accept new inputs
    pioTick = 150;
  }

  if(DACValue)
  {
    DACTick--;
    if(!DACTick)
    {
      OSSemPost(schDAC);
      DACTick = 40;
    }
  }
  else if(lastSpeedRead) // read current speed after last speed increment
  {
    lastSpeedRead =0;
    OSSemPost(schADC);
  }

  if((DACValue)) // read current speed only when being changed
  {
    ADCTick--;
    if(!ADCTick)
    {
      OSSemPost(schADC);
      ADCTick = 10;
    }
  }
}

```

```

menuTick--;
if(!menuTick)
{
    OSSemPost(schMenu);
    menuTick = 400;
}
}

```

//Lab8.c Main file that handles 3 tasks and 2 ISR's

```

#include "./os_cpu.h"
#include "./os_cfg.h"
#include "./ucos_ii.h"
#include "nios.h"
#include "lab8.h"
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include "pio_lcd16207.h"

int state = MAIN_MENU;
int pio_data = 0;
int rcvd_pio = 0;
int currentSpeed = 0;
int speedSetting[] = {0,0,0,0};
int newSpeed = 0;
int digitCnt=0;
int DACValue= 0;
int DACincr=0;
int incrdecr=0;
int DACbase=0;
int rcv_speed=0;
int curdir =0; // 1-CLOCKWISE 0-ANTICLOCK
int newdir =0;
char currentDir[] ="ND"; //NO_DIRECTION
int lastSpeedRead=0;
#ifdef ONT_GLOBALS
#define ONT_EXT
#else
#define ONT_EXT extern
#endif

//*****
// DATA TYPES
//*****
typedef struct {
    char TaskName[30];
    INT16U TaskCtr;
    INT16U TaskExecTime;
    INT32U TaskTotExecTime;
} TASK_USER_DATA;

//*****
// VARIABLES

```

```

//*****
ONT_EXT TASK_USER_DATA TaskUserData[10];

// allocate memory for tasks' stacks
#define STACKSIZE 2048
OS_STK Stack1[STACKSIZE];
OS_STK Stack2[STACKSIZE];
OS_STK Stack3[STACKSIZE];

// global variables used by the tasks of bubblesort and user-interface
OS_EVENT *schDAC; // #1 priority
OS_EVENT *schADC;
OS_EVENT *schMenu; // #3 priority

//*****
// FUNCTION PROTOTYPES
//*****

int calculateSpeed(int);
void setDirection(int dir);

int decodeDigit();

void setDirection(int dir)
{
    if(dir==CLOCKWISE) // set clockwise if dir else anticlock
    {
        currentDir[0]='C';currentDir[1]='W';currdir =1;
    }
    else if(dir == ANTICLOCK)
    {
        currentDir[0]='A';currentDir[1]='C';currdir =0;
    }
    else if(dir == NO_DIRECTION)
    {
        currentDir[0]='N';currentDir[1]='D';currdir =2;
    }
}

int calculateSpeed(int rcv_adc)
{
    if(rcv_adc > SPEED_RESOL)
    {
        currentSpeed = ceil(((rcv_adc-SPEED_RESOL)*MAX_SPEED)/SPEED_RESOL);
        setDirection(CLOCKWISE);
    }
    else if(rcv_adc < SPEED_RESOL)
    {
        currentSpeed = ceil(((SPEED_RESOL-rcv_adc)*MAX_SPEED)/SPEED_RESOL);
        setDirection(ANTICLOCK);
    }
    else
    {
        currentSpeed = 0;
        setDirection(NO_DIRECTION);
    }
}

```

```

    return currentSpeed;
}

void adcPIO_ISR(int context)
{
    char strLCD[]={"\0"};
    np_pio *pio = (np_pio *)context;
    int calc_speed;
    rcv_speed = pio->np_piodata & ADC_DATA_MASK ;
    calc_speed = calculateSpeed(rcv_speed);
    printf("The Current speed:%d\n", calc_speed);
    sprintf(strLCD,"Speed:%d,Dir:%c%cSPEED BEING CHGD",calc_speed,currentDir[0],currentDir[1]);
    nr_pio_lcdwritescreen(strLCD);
    pio->np_pioedgecapture = 0;          // clear the irq condition
}

void ADCHandlerTask (void *i)
{
    np_pio *adc_ptr = na_adc_pio;    // ptr for adc_pio
    INT32U Time;
    INT8U Reply;
    int pio_data;
    int j;
    int temp=0;
    int a=0;
    for (;;) // infinite loop
    {
        OSSemPend(schADC, 0, &Reply);
        //Time = OSTimeGet();
        //printf( "user interface Task: %d\n", Time );
        //setting up ADC

        pio_data = ADC_ISR_MASK & adc_ptr->np_piodata;
        pio_data =((pio_data |ADC_MODE_MASK)& ADC_CS_WREN_MASK);
        adc_ptr->np_piodata = pio_data;
        for(j=0;j<TWR;j++) //dummy loop 33Mhz -> 30ns clock therefore 40 cycles ->1200ns
            temp = a ;
        adc_ptr->np_piodata = (pio_data | ADC_CS_WRDIS_MASK);
    }
}

void keypadPIO_ISR(int context)
{
    int decodedDigit;
    np_pio *pio = (np_pio *)context;
    char strLCD[]={"\0"};
    int i;
    int *ptr;
    nr_delay(5); // 10 msec delay to take care of transients and debounce
    pio_data = pio->np_piodata;
    pio_data = pio_data & PIO_DATA_MASK;
    if(rcvd_pio != pio_data && (pio_data & PIO_DATA_COL_MASK))
    {
        rcvd_pio = pio_data;
        decodedDigit= decodeDigit();
    }
}

```

```

switch(state)
{
case MAIN_MENU:
    if(decodedDigit == 1)
    {
        state = ENT_DIR;
        printf("enter 8-clockwise 9-anticlock\n");
        sprintf(strLCD,"Spd:%d,Dir:%c%cEnter 8-CW 9-AC",currentSpeed,currentDir[0],currentDir[1]);
        nr_pio_lcdwritescreen(strLCD);
    }
    else
    {
        sprintf(strLCD,"Speed:%d,Dir:%c%cIncorrect Choice",currentSpeed,currentDir[0],currentDir[1]);
        nr_pio_lcdwritescreen(strLCD);
    }

    break;
case ENT_DIR:
    if(decodedDigit == 8)
    {
        newdir = 1;
        //setDirection(CLOCKWISE);
        state = ENT_SPD;
    }
    else if(decodedDigit == 9)
    {
        newdir = 0;
        //setDirection(ANTICLOCK);
        state = ENT_SPD;
    }
    else
    {
        sprintf(strLCD,"Speed:%d,Dir:%c%cBad Direction",currentSpeed,currentDir[0],currentDir[1]);
        nr_pio_lcdwritescreen(strLCD);
    }
    break;
case ENT_SPD:
    speedSetting[digitCnt++] = decodedDigit;
if(digitCnt == NO_DIGITS_IN_SETTING)
    {
        digitCnt=0;

        //ptr = &speedSetting[0];
        //printf("New SpeedSetting is %d%d%d%d\n",*ptr++,*ptr++,*ptr++,*ptr);

        newSpeed = (speedSetting[0]*1000 + speedSetting[1]*100 + speedSetting[2]*10 + speedSetting[3]);
        printf("new speed setting:%d\n",newSpeed);

        if(newSpeed > MAX_SPEED)
        { printf("invalid speed\n");
          newSpeed = currentSpeed;
          sprintf(strLCD,"Speed:%d,Dir:%c%cENT Spd<7200rpm",currentSpeed,currentDir[0],currentDir[1]);
          nr_pio_lcdwritescreen(strLCD);
        }
    }
}

```

```

else // valid newspeed processing
{
    printf("curdir:%d newdir:%d curspeed:%d newspeed:%d\n",curdir,newdir,currentSpeed,newSpeed);

    if(curdir == newdir)
    {
        if(curdir) //clockwise
        {
            DACbase=SPEED_RESOL+ceil((currentSpeed*SPEED_RESOL)/MAX_SPEED);
            if(newSpeed>currentSpeed)
            {
                DACValue=ceil(((newSpeed-currentSpeed)*SPEED_RESOL)/MAX_SPEED);
                DACincr=0; // to incr speed
            }
            else
            {
                DACValue=ceil(((currentSpeed - newSpeed)*SPEED_RESOL)/MAX_SPEED);
                DACincr=1; // to dcr speed
            }
        }
        else //anticlock
        {
            DACbase=SPEED_RESOL-ceil((currentSpeed*SPEED_RESOL)/MAX_SPEED);
            if(newSpeed>currentSpeed)
            {
                DACValue=ceil(((newSpeed-currentSpeed)*SPEED_RESOL)/MAX_SPEED);
                DACincr=1; // to incr speed
            }
            else
            {
                DACValue=ceil(((currentSpeed - newSpeed)*SPEED_RESOL)/MAX_SPEED);
                DACincr=0; // to dcr speed
            }
        }
    }
    else // direction is different
    {
        if(newdir) // clockwise
        { //current direction anticlock
            DACbase=SPEED_RESOL-ceil((currentSpeed*SPEED_RESOL)/MAX_SPEED);
            DACValue= SPEED_RESOL +ceil((newSpeed*SPEED_RESOL)/MAX_SPEED) -DACbase;
            DACincr=0; //incr to new speed
        }
        else // anticlockwise
        {
            //current direction is clockwise
            DACbase=SPEED_RESOL+ceil((currentSpeed*SPEED_RESOL)/MAX_SPEED);
            DACValue = DACbase - SPEED_RESOL +ceil((newSpeed*SPEED_RESOL)/MAX_SPEED);
            DACincr=1; //dcr to newspeed
        }
    }
    state = SPD_BEING_CHGD;
    printf("Enter 1 to chg speed_dir\n");
    printf("dacval:%d dacincr:%d dacbase %d\n",DACValue, DACincr, DACbase);
}

```

```

        }//valid speed

        }// four digits
        break;

    case SPD_BEING_CHGD:
        // No processing, User not allowed to change speed when speed is being changed
        break;

    }// switch
} //if
pio->np_pioedgecapture = 0;          // clear the irq condition
}

void DACHandlerTask (void *i)
{
    np_pio *dac_pointer = na_dac_pio;    // ptr for dac_pio
    INT32U Time;
    INT8U Reply;
    int j;
    int temp=0;
    int a = 1;
    CPUInit(); // highest priority task - initialize and enable timer
    for (;;) // infinite loop
    {
        OSSemPend(schDAC, 0, &Reply);
        if(DACValue)
        {
            if(DACincr) // decrement
            {
                if(incrdecr < DACValue)
                {
                    incrdecr++; printf("bit value of currentspd:%d \n", (DACbase-incrdecr));
                    //Time = OSTimeGet();
                    //printf( "user interface Task: %d\n", Time );
                    dac_pointer->np_piodata = CS_CE_ENA_MASK&(DACbase-incrdecr) ;

                    for(j = 0; j<9;j++) // dummy loop 33Mhz -> 30ns clock therefore 7 cycles ->210ns 8-cycle dummy to
                    be safe
                        temp = a ;

                    dac_pointer->np_piodata = LATCH_DAC_MASK&(DACbase-incrdecr);
                }
                else
                {
                    incrdecr =0; DACValue =0; lastSpeedRead =1; state=MAIN_MENU;
                }
            }
            else // increment
            {
                if(incrdecr < DACValue)
                {
                    incrdecr++; printf("bit value of currentspd:%d\n", (DACbase+incrdecr));
                    //Time = OSTimeGet();

```

```

//printf( "user interface Task: %d\n", Time );
dac_pointer->np_piodata = CS_CE_ENA_MASK&(DACbase+incrdecr) ;

for(j = 0; j<9;j++) // dummy loop 33Mhz -> 30ns clock therefore 7 cycles ->210ns 8-cycle dummy to
be safe
    temp = a ;

    dac_pointer->np_piodata = LATCH_DAC_MASK&(DACbase+incrdecr);
}
else
{
    incrdecr =0; DACValue =0; lastSpeedRead =1;state=MAIN_MENU;
}
}

} // valid DACValue
} // for
}

```

```

void backgroundMenu(void *i)
{
    INT32U Time;
    INT8U Reply;
    char strLCD[] ={"\0"};
    for (;;) // infinite loop
    {
        OSSemPend(schMenu, 0, &Reply);
        //Time = OSTimeGet();
        //printf( "user interface Task: %d\n", Time );

        switch(state)
        {
            case MAIN_MENU:
                //nr_delay(1000);
                sprintf(strLCD,"Spd:%d,Dir:%c%cEntr1-chgspd_dir",currentSpeed,currentDir[0],currentDir[1]);
                nr_pio_lcdwritescreen(strLCD);
                break;
            case ENT_SPD:
                //nr_delay(1000);
                sprintf(strLCD,"Spd:%d,Dir:%c%cEntr Spd <7200",currentSpeed,currentDir[0],currentDir[1]);
                nr_pio_lcdwritescreen(strLCD);
                break;
            case SPD_BEING_CHGD:
                //ADC task takes care of display - NO processing here
                break;
        }
    }
}

```

```

int decodeDigit()
{int dDigit;

    switch(pio_data)
    {
        case ONE:

```

```

    dDigit = 1;
    break;
case TWO:
    dDigit = 2;
    break;
case THREE:
    dDigit = 3;
    break;
case FOUR:
    dDigit = 4;
    break;
case FIVE:
    dDigit = 5;
    break;
case SIX:
    dDigit = 6;
    break;
case SEVEN:
    dDigit = 7;
    break;
case EIGHT:
    dDigit = 8;
    break;
case NINE:
    dDigit = 9;
    break;
case ZERO_DIGIT:
    dDigit = 0;
    break;
default:
    // only 0-9 decoded other keys given hex e value
    dDigit = 0xe;
    break;
}
printf("the digit is: %d\n", dDigit);
return dDigit;
}

int main ()
{
    char ld1 = '1';
    char ld2 = '2';
    char ld3 = '3';
    char strLCD[]={'\0'};
    np_pio *pio_pointer = na_keypad_pio; // ptr for keypad pio
    np_pio *dac_pointer = na_dac_pio; // ptr for dac_pio
    np_pio *adc_pointer = na_adc_pio; // ptr for adc_pio

    // init the LCD
    nr_pio_lcdinit(na_lcd_pio);
    //ISR init, setting up our keypad_pio isr

    nr_installuserisr(na_keypad_pio_irq,keypadPIO_ISR, (int)pio_pointer);
    nr_installuserisr(na_adc_pio_irq,adcPIO_ISR, (int)pio_pointer);

    //setting up keypad isr

```

```

pio_pointer->np_piodirection = 0; // all 8 lines are input
pio_pointer->np_pioedgecapture = 0; // clears any previous isr's
// use colum bits for interrupt generation
pio_pointer->np_piointerruptmask = PIO_DATA_COL_MASK;

//setting up direction of DAC PIO
dac_pointer->np_piodirection = 0x1FF;

//setting up adc isr
adc_pointer->np_piodirection = 0; // all 8 lines are input
adc_pointer->np_pioedgecapture = 0; // clears any previous isr's
// use colum bits for interrupt generation
adc_pointer->np_piointerruptmask = ADC_ISR_MASK;
//0111 0000 0000
adc_pointer->np_piodirection = 0x700;

// needed by uC/OS
OSInit();
schDAC = OSSemCreate(0);
schADC = OSSemCreate(1);
schMenu = OSSemCreate(1);

// create the tasks in uC/OS and assign decreasing priority to them
OSTaskCreate(DACHandlerTask, (void *)&Id1, (void *)&Stack1[STACKSIZE-1], 1);
OSTaskCreate(ADCHandlerTask, (void *)&Id2, (void *)&Stack2[STACKSIZE-1], 2);
OSTaskCreate(backgroundMenu, (void *)&Id3, (void *)&Stack3[STACKSIZE-1], 3);

// start the os
OSStart();
}

```

Testing and Results

The following tests were performed to check the implementation of the design

1. First the user input speed was validated. Only input values in the range of
2. Check if Change of motor direction works
3. Check if set speed is achieved with 2% accuracy (takes care of resolution and quantization errors in D-A,A-D conversions)
4. Check for gradual step up of speed of motor to new speed
5. Check User-interface for proper menus, prompts and error messages

The DAC was tested and the speed ramp to new speed was checked and the results are given below in the next section

Testing DAC- Incremental speed change and Analog Voltage ramping to Servo

```
[SOPC Builder]$ nb os_cpu_a.s os_cpu_c.c ucos_ii.c lab8.c
```

Beginning Build

Sources:

```
os_cpu_a.s
os_cpu_c.c
ucos_ii.c
lab8.c
```

```
# 2004.12.06 22:20:55 (*) nios-elf-as --defsym __nios32__=1 --defsym __timeStam
p__=44075 --gstabs -I .. -I ../.. -I ../inc -I ../inc -I ../inc -I ../
../inc -I ../inc -I ../inc -m32 os_cpu_a.s -o os_cpu_a.s.o
```

```
# 2004.12.06 22:20:55 (*) nios-elf-gcc -I .. -I ../.. -I ../inc -I ../inc -I
../inc -I ../inc -I ../inc -I ../inc -W -Wno-multichar -g -mno
-zero-extend -O2 -m32 os_cpu_c.c -o os_cpu_c.c.o -c
```

```
# 2004.12.06 22:20:56 (*) nios-elf-gcc -I .. -I ../.. -I ../inc -I ../inc -I
../inc -I ../inc -I ../inc -I ../inc -W -Wno-multichar -g -mno
-zero-extend -O2 -m32 ucos_ii.c -o ucos_ii.c.o -c
```

```
# 2004.12.06 22:20:57 (*) nios-elf-gcc -I .. -I ../.. -I ../inc -I ../inc -I
../inc -I ../inc -I ../inc -I ../inc -W -Wno-multichar -g -mno
-zero-extend -O2 -m32 lab8.c -o lab8.c.o -c
```

```
# 2004.12.06 22:20:58 (*) nios-elf-ld -e _start -u _start -g -T /cygdrive/c/alte
ra/excalibur/sopc_builder/bin/excalibur.ld ../lib/obj32/nios_jumptostart.s.o os
_cpu_a.s.o os_cpu_c.c.o ucos_ii.c.o lab8.c.o --start-group -l nios32 -l c -l m -
l gcc --end-group -L /cygdrive/c/altera/excalibur/sopc_builder/bin/nios-gnupro/ni
os-elf/lib/m32 -L /cygdrive/c/altera/excalibur/sopc_builder/bin/nios-gnupro/lib/g
cc-lib/nios-elf/2.9-nios-010801-20030520/m32 -L ../lib -L ../lib -L ../lib
-L ../lib -L ../lib -L ../lib -L ../inc -L ../inc -L ../inc -L ..
../inc -L ../inc -L .. -o lab8.out
```

```
# 2004.12.06 22:20:59 (*) nios-elf-objcopy -O srec lab8.out lab8.srec
```

```
# 2004.12.06 22:20:59 (*) nios-elf-nm lab8.out | sort > lab8.nm
```

```
# 2004.12.06 22:20:59 (*) nios-elf-objdump -d --source lab8.out > lab8.objdump
```

Finishing Build

```
/cygdrive/x/lab8/lab8_1/cpu_sdk/src
```

```
[SOPC Builder]$ nr lab8.srec
```

```
nios-run: Ready to download lab8.srec over COM1: at 115200 bps
```

```
nios-run: Downloading.....
```

nios-run: Terminal mode (Control-C exits)

```

-----
HL: 224
HL: 224
HL: 224
HL: 224
HL: 224
the digit is: 1
enter 8-clockwise 9-anticlock
the digit is: 8
the digit is: 2
the digit is: 5
the digit is: 6
the digit is: 8
new speed setting:2568
currdir:0 newdir:1 curspeed:0 newspeed:2568
Enter 1 to chg speed_dir
dacval:45 dacincr:0 dabase 128
bit value of currentspd:129
bit value of currentspd:130
bit value of currentspd:131
bit value of currentspd:132
bit value of currentspd:133
bit value of currentspd:134
bit value of currentspd:135
bit value of currentspd:136
bit value of currentspd:137
bit value of currentspd:138
bit value of currentspd:139
bit value of currentspd:140
bit value of currentspd:141
bit value of currentspd:142
bit value of currentspd:143
bit value of currentspd:144
bit value of currentspd:145
bit value of currentspd:146
bit value of currentspd:147
bit value of currentspd:148
bit value of currentspd:149
bit value of currentspd:150
bit value of currentspd:151
bit value of currentspd:152
bit value of currentspd:153
bit value of currentspd:154
bit value of currentspd:155
bit value of currentspd:156
bit value of currentspd:157
bit value of currentspd:158
bit value of currentspd:159
bit value of currentspd:160
bit value of currentspd:161
bit value of currentspd:162
bit value of currentspd:163
bit value of currentspd:164
bit value of currentspd:165
bit value of currentspd:166
bit value of currentspd:167
bit value of currentspd:168
bit value of currentspd:169
bit value of currentspd:170
bit value of currentspd:171
bit value of currentspd:172
bit value of currentspd:173

```

Note: *The bit value of currentspd is the digital input to the DAC:*

- 255 bits corresponds the maximum speed in the clockwise direction, while
- 128 bits is the zero speed and
- 0 bits corresponds to a maximum speed in the anticlockwise direction

Final Document for Servo Motor Control

December 10, 2004

Nisheet Gupta
Narayanan Krishnamurthy

Final Document

The motor speed control system is used to control the speed and direction of the motor included in the setup located in the lab. The operator was able to set the desired speed and direction and observe the actual speed and direction. The setting of speed and direction was done using a keypad and the actual speed and direction was observed on the motor and the LCD screen.

The digital output from the keypad was used by the CPU to provide a digital input in proper form to the ADC. Also, the digital data was displayed on the LCD screen. The 0-10v output of the DAC was divided to obtain 0-5v. The 0-5v was input to the non-inverting input and a -2.9v reference voltage was input to the inverting input of the internal op-amp which resides in the speed control unit of the servo controller. The speed of the motor was then observed to change depending on the user's input.

The second phase of the design was displaying the actual speed on the LCD. For this, the +16v output of the tachometer was level shifted to 0v-5v. The 0v-5v was input to the ADC which converted it to 8-bit digital signal. The 8-bit digital signal was input to the CPU which converted it to a proper form needed to be displayed on the LCD.

Input speed is in RPM, Display speed is in RPM. User has to specify direction (clockwise/anti-clockwise) and speed to change the speed of the stepper motor, the controller should take care of gradually increasing the motor speed to the new user defined speed.

One of the parameters that changed over the course of the design is the reference voltage that is used for the speed control unit. Before the design implementation, a voltage of -2.5v was chosen to be a reference voltage. Because of the limited resistor values that were available, a value of -2.9v was obtained instead of -2.5v.

After measuring the output of the level shifter, it was noticed that an inverting amplifier instead of a non-inverting amplifier had been implemented. To fix this, the output of the inverting amplifier was fed into another inverting amplifier to obtain a non-inverting output.

During testing it was noticed that the user could enter data during the ramping of the motor. Since this is undesirable, the code was changed so that the user could not enter any values while the motor speed was being ramped up or down.

DAC was checked independently of the system. Various input values were provided and based on that, the ramping of the voltage was seen. There were a total of four voltage dividers that were implemented to obtain a single power supply. These voltage dividers were independently checked and were verified to perform correctly. After final soldering, the functionality of DAC was verified again but incorrect results were obtained. Also, the voltage dividers were functioning incorrectly. As a last resort, the voltage supply to the DAC was removed and the voltage divider functionality was checked again and verified to be correct. This led us to conclude that during final soldering, a short had occurred in the JP8 terminal. Due to the lack of time needed for debugging the circuit, it was decided to stop the implementation of the motor control system.

Summary of the final report

1. The DAC functioning with the multitasked DAC Task in software was successfully verified and tested for ramping the motor speed to new speed from current speed (Results shown in the Implementation and testing Document)
2. The ADC and the level shifter (summing amplifier) were built and tested.
3. Final Soldering of JP8 has a short which could not be identified, thus the integrated ADC is not getting VCC because of the short, and hence the integrated ADC and DAC does not function though individually they had worked before integration.