EE552
Final Report

A motor speed controller

Group members:	Alam Fakhre	376285
	Arshid Khan	358261
	Omar Hussain	233203
	Rizwan Muhammad	376356

Abstract

This project involved the design of a motor speed controller. The fundamental components of the system are a keypad, a display unit, and of course, a motor. The display shows at all times, the current motor speed, and the desired operational motor speed. The user inputs the desired operational speed via the keypad, and the motor speed controller chip sends pulses out to the motor to increase or decrease its speed as required.

The controller was designed and written in VHDL, simulated in MAX+plus II, and programmed on the Altera 10k20 FPGA.

Several important accomplishments were made during the course of this project of which the following are a few. A great deal of knowledge was acquired in the area of FPGA design. The degree of comfort and proficiency in designing, understanding and implementing systems in VHDL was also greatly increased. The project provided valuable insight into the interaction and coordination required to successfully complete a project in a group of engineers.

Table of contents

Response to feedback

Unfortunately, feedback on the simulation documentation was not available at the time of the writing of this report, therefore the comments obtained on the resource requirement documentation are dealt with here.

Comments:

1.) Your writing is better.
2.) You haven't clearly indicated the difference between automatic mode (where INC and DEC can be used) and manual mode (where INC and DEC can be used).
3.) I was looking for a more detailed estimate of the number of logic cells in the work left to be done.
4.) What kind of controller are you using? Proportional, PI, PDI?

Responses:

1.) We are gratified that our efforts in this area have been noticed.
2.) The manual mode feature had to be removed due to space constraints, so the issue of the operation of INC and DEC in that mode is now non-existent. That leaves the automatic mode, and in this mode INC and DEC can be pressed at any time. If the motor is at a standstill, pressing the INC or DEC keys causes the user-set speed to increment or decrement by 10rpm – the final user-set speed value is then stored and displayed. When FWD or REV are subsequently pressed, the motor starts and the controller attempts to run the motor at the user-set speed.
   If the motor is running and INC or DEC are pressed, the user-set speed is updated and displayed, and the motor controller attempts to change the motor's speed on the fly in accordance with the new user-set speed value.
   In summary, then, the INC and DEC keys can be pressed at any time, but the system response to these inputs is dependent on the present state of the system at the time of input.
3.) At this point no work remains to be done in terms of VHDL and 1064 from a total of 1152 available logic cells have been used.
4.) The controller is proportional – any other type of controller, PI, PD, PDI, etc. would have been significantly more difficult to implement, and most likely would have consumed more resources.

Overview

This project involves the design of a digital motor speed controller system. The design was simulated in VHDL, programmed into the Altera 10k20 FPGA, and interfaced with external circuitry involving a motor, keypad, and seven segment display array. Fundamentally, the system allows a user to view the current motor speed (also referred to as the measured speed), and to set and view the desired motor speed (also referred to as the user-set speed).

There are two main elements to the user interface for this system:

1.) A seven segment LED display, with three digits for displaying the current motor speed, and three digits for displaying the user-set speed
2.) A 12-key keypad with the following keys

Key name	Key description
INC	Increment motor speed by 10rpm
DEC	Decrement motor speed by 10rpm
FWD	Run motor in forward drive
REV	Run motor in reverse drive
STOP	Stop the motor
MAN	Manual motor control mode ← **OBSOLETE!**

The system is powered on by depressing the main power switch. Initially, when the system is powered up, the motor is at rest, the current speed display shows 0rpm, and the user-set speed display reads 10rpm. 10rpm is the minimum value for the user-set speed, and it is set to this minimum on startup and on every system reset.

In the initial design for this project, there were two modes of motor control: automatic, and manual. The automatic control was to be set by default on startup and on system reset, with the MAN key providing the user with the option of overriding this default mode to enter manual speed control mode. As the project neared completion however, and resources on the FPGA ran low, the option for the manual mode had to be removed, rendering the MAN key obsolete as indicated in the table above. The motor runs in automatic control mode at all times now.

The user enters the desired speed and the digital motor speed controller automatically takes the steps needed to attain this speed. The user-set speed is incremented by pressing the INC key, or decremented via the DEC key. The user-set speed increases or decreases by 10rpm respectively for each stroke of the INC and DEC keys, and any changes made to the user-set speed readily appear on the user-set speed display.

Once the desired speed has been set, the user starts the motor by pressing either of the FWD or REV keys, which cause the motor to run in forward or reverse drive, respectively. The current speed display updates frequently to indicate the motor's approach to the user-set speed.

While the motor is running, the user can press the STOP key to bring the motor to a halt. If the INC and DEC keys are pressed while the motor is running in automatic mode, the user-set speed display is updated to show the new user-set speed, and the controller adjusts the current speed to approach the new user-set speed.

The FWD and REV keys allow the user to select between one of two drive directions for the motor, namely forward and reverse.

If the motor is at a standstill, pressing the FWD and REV keys cause the motor to start in forward or reverse drive, respectively. The speed at which the motor runs is given by the user-set speed which is either entered by the user via INC and DEC as discussed above, or set at a default of 10rpm.

If the motor is running in either forward or reverse drive, the FWD and REV keys are ignored. If the motor is in forward drive for example, and the user wishes to change to reverse drive, pressing the REV key will not do anything. The STOP key must be pressed first, and then the REV key can be pressed to activate reverse drive.

IC data

The following is a list of some of the main features which had to be implemented:

	Feature set / module	Works as determined by simulation?
1	Scanning display	Yes
2	Binary to BCD conversion	Yes
3	Error calculation block	Yes
4	Motor speed measurement	Yes
5	Firing angle delay module	Yes
6	Forward and reverse SCR bridge buffers	Yes
7	SCR protection block	Yes
8	Keypad module	Yes

Total logic cell usage:

	Feature set / module	Number of logic cells used
1	Main system FSM with keypad controller	111
2	Display module	292
3	Firing angle delay, SCR bridge, error calculation module	668
4	Motor speed controller top level entity (instantiates the above)	1064
	Total cell usage:	**1064**

So for the final project, 1064 of 1152 available logic cells, or 92% of the available logic cells have been used.

It was a challenge to fit our project into the FPGA, features such as manual override had to be dropped, and optimizations had to be performed to avoid exceeding the available resources. In order to implement any additional features we would need more cells, perhaps a different FPGA than the one obtained from Altera for this project could be used.

As mentioned earlier, the removal of the manual control means that the motor only runs in auto mode, and the motor speed can be increased and decreased by pressing the INC or DEC keys on the keypad.

Maximum Speed:

- The maximum clock as indicated by the timing analyzer is 13.42MHz.
- The minimum time period from the timing analyzer is 74.5ns.
- We are using a system clock of 500kHz.
- We are also using a separate clock of 180Hz for scanning the six seven segment display units.

Pin Names:

Pin name:	Pin type:
Clock	In
Reset	In
Zero_crossn	In
Zero_crossp	In
In_monopulse	In
Techo_in	In
Sense_line	In
Pulse_out	Out
Led_stop	Out
Led_rev	Out
Led_fwd	Out
Fbout	Out
Rbout	Out
Seg_data	Out
Seg_enable	Out
Drive_line	Out

Design details

In this section some of the problems encountered during this project and their solutions are presented.

One of the first problems upon programming the FPGA with the .sof file from our working simulation was that, essentially nothing seemed to work. The LED displays were very dim, so it appeared initially as if the system was dead. However, after shielding the LEDs from the ambient light present in the lab, it was observed that the display was updating after a fashion.

The display was initially only lighting up five of the six LED segments. This was traced to an error in the output holding registers for the display module which was then rectified. Once all the digits were displaying data (albeit dimly), it was noted that the display was constantly showing 0rom for the measured speed and 10rpm for the set speed regardless of waveforms applied to the tacho input or keys pressed on the keypad.

We soon realised that the reset pushbutton we were using was active low – our assumption of an active high reset had resulted in our system being in a state of continual reset. The VHDL code was changed to reflect this, and the display was tested again.

Now the display seemed to be working properly in conjunction with the keypad and the rest of the system, so we decided to attempt to increase the intensity of the display to make it more legible. We found we could improve the brightness of the segments somewhat by providing the seven segment array power from a separate supply as opposed to that of the FPGA, and by scanning at a slightly lower frequency. Although this caused greater flicker, it allowed the segments to stay on for a longer period of time and thus appear brighter.

During the development of the main system FSM, insufficient attention had been paid to making the modules of our project as independent from one another as possible. This created a great difficulty when we attempted to implement the main system FSM. The initial draft is shown on the following four pages. The main FSM was too complex, with too many states, and we were certain it would create many timing problems. The solution was to distribute the FSM load, in a sense. The entire project was divided into three high level modules, each relatively autonomous and communicating with other modules (if needed) via a very small number of control signals. Each module instantiates and controls the smaller modules of the project which were related in their functions. The main FSM ended up being so simple that it didn’t even need a separate module, it was included as part of the multi purpose firing module.

(Please see the design documentation section on page 15 for diagrams of the modules which can be found throughout the body of that section as indicated in the table of contents, and the final main system FSM is on page 24 in the same section.)

Design verification

This section deals with the simulation of the whole chip for the motor speed controller project (MSC project) in MAX+plus II. The MSC project is composed of numerous VHDL entities, which have been designed to be as autonomous from one another as possible.

Some of the higher level entities designed in this project are; a 12 key scanning keypad entity, a scanning display entity, a speed measurement entity, a firing angle generating entity, an error calculation entity, and an SCR selection and protection entity.

The inputs to the MSC project module are; zero cross pulses (negative and positive), sense and drive lines from a row-matrix keypad (3 x 4), a tachogenerator input (pulses/rev. coming from motor shaft), and a mono stable pulse (used for error calculation timing).

The outputs are; six enable lines for scanning six seven segment display units, seven display data lines a, b, c, d, e and f , two groups of 4-bit SCR firing pulse outputs for forward and reverse drive for the motor, a 3 bit motor status output (going to 3 LEDs which light up when the motor is stopped, or in forward or reverse drive), and a 1-bit output for enabling the external mono stable circuit. The display is an integral part of the motor speed controller system. The current motor speed, and the desired motor speed set by the user are displayed using three digits for each, so that speeds from 0 rpm to 999 rpm can be shown.

The following table is very important for the following section. It demonstrates the correlation between the data to be displayed, the seven segment data output by the MSC chip, and the decimal numbers printed on the simulation waveforms.

Digit displayed	Seven segment data							Decimal number on waveform
	a	b	c	d	E	f	g	
0	0	0	0	0	0	0	1	1
1	1	1	0	0	1	1	1	103
2	1	0	0	1	0	0	0	72
3	1	0	0	0	0	1	0	66
4	0	1	0	0	1	1	0	38
5	0	0	1	0	0	1	0	18
6	0	0	1	0	0	0	0	16
7	1	0	0	0	1	1	1	71
8	0	0	0	0	0	0	0	0
9	0	0	0	0	1	1	0	6
error	0	0	1	1	0	0	0	24

The following test cases were applied in order to verify the correctness of the simulation results:

1. ’Reset’ test case

As shown in Fig. 1 on the following page, ‘reset’ is forced high and as expected, the system resets to a known state.

- 'Led_rev' signal goes high (off state).
- 'Led_stop' signal which represent the motor stop status goes low (active state – the stop LED turns on).
- 'Led_fwd' or motor forward signal goes high (off state).
- 'fbout' or forward SCR firing pulse out are all high (15 = 1111b means all SCRs are off).
- 'rbout' or reverse SCR firing pulse out are all high (15 = 1111b means all SCRs are off).

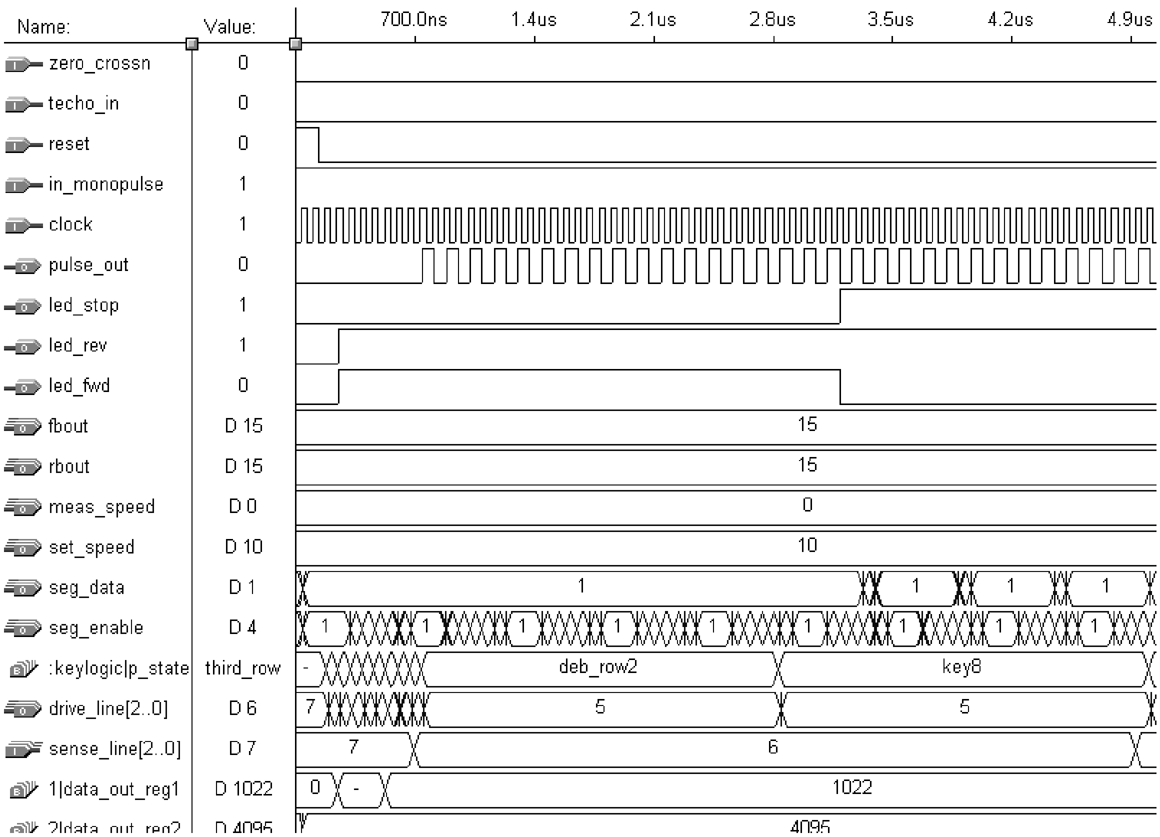

Fig. 1

Also note that the measured speed, ‘meas_speed’ has correctly been reset to 0rpm, and the user-set speed, ‘set_speed’ has correctly been reset to 10rpm. Fig. 2 on the following page, demonstrates that these speeds are displayed correctly. First data is sent out from the MSB to LSB of measured speed (seg_enable goes from 1 to 8, enabling these three digits). Then data is sent out from the MSB to the LSB of user-set speed (seg_enable goes from 1 to 8, enabling these three digits).

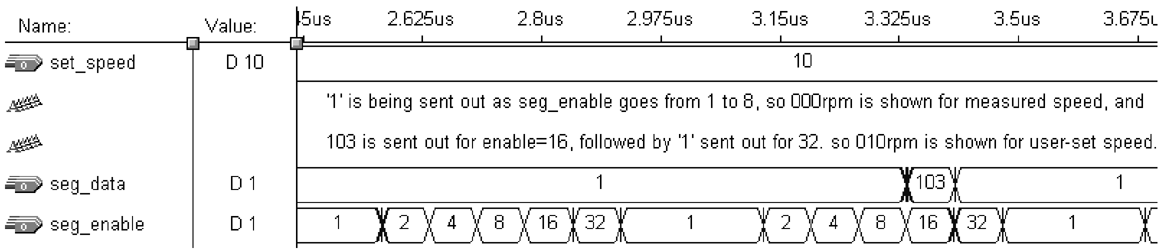

Fig. 2

2. 'Motor forward' test case

As shown by the waveform in Fig. 3, sending sense_line (0) low (so that sense_line[2..0] goes from 7 to 6, while the drive_line is 5, simulating a FWD key press) causes the keypad FSM to detect the key '8' (the internal representation for the forward key), and after detecting the key '8', the led_stop (for motor stop) goes high, and the motor forward (led_fwd) goes low, meaning that the motor is in the forward mode of operation and all other key signals were initialized properly.

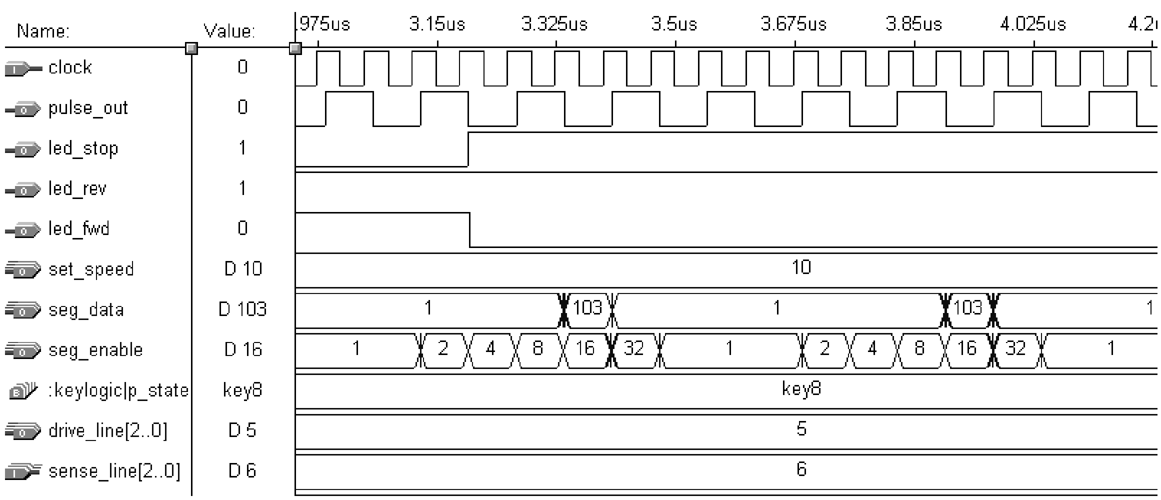

Fig. 3

3. 'Motor stop' test case

In Fig. 4, the sense_line stays at 6, while the drive_line is 6, simulating a STOP key press. The key detected by the keypad entity is key9 as shown. Key9 is the internal representation for the STOP key, so the motor forward output (led_fwd) goes high, the motor reverse (led_rev) output goes high and the motor stop (led_stop) output goes low indicating that the motor has correctly entered the stop mode.

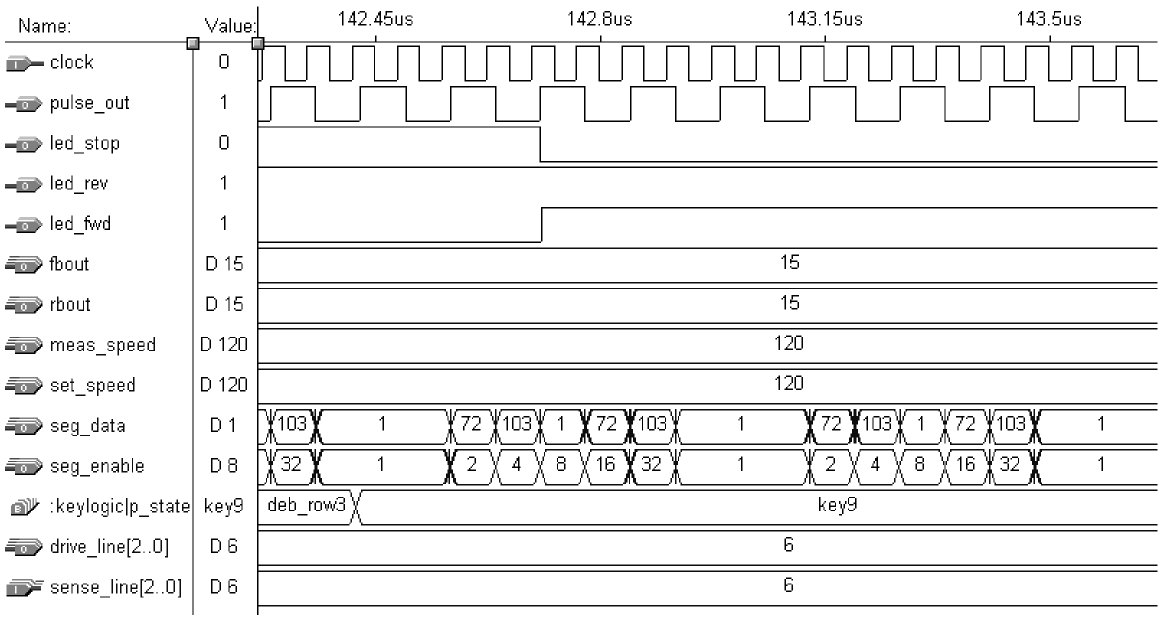

Fig. 4

4. 'Motor reverse' test case

As shown in Fig. 4, the motor was stopped, and as shown in Fig. 5 on the following page, the sense_line stays at 6, while the drive_line is set to 3, simulating a REV key press.
This is detected as key7 by the keypad FSM. When the main system FSM sees that key7, which is the internal representation for reverse, has been pressed, it causes some changes in the output of the motor speed controller. The motor stop output goes high, motor forward output (led_fwd) goes high, and motor reverse (led_rev) output goes low indicating that the motor is properly running in reverse mode.

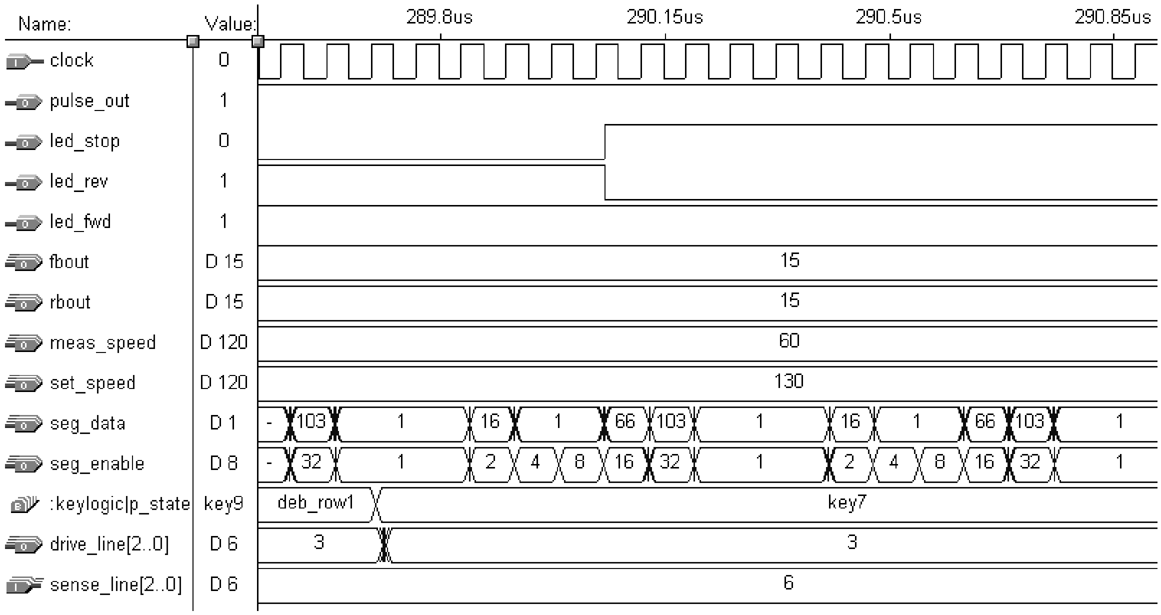

Fig. 5

5. ‘Speed measurement’ test case

This test case demonstrates that the tachogenerator input is being correctly translated into the current speed of the motor in rpm. For the initial setting of the tacho_in signal to a period of 700ns, the speed was correctly being measured as 60rpm. When the period of the tacho_in pulses is halved to 350ns, the measured speed correctly doubles to 120rpm.

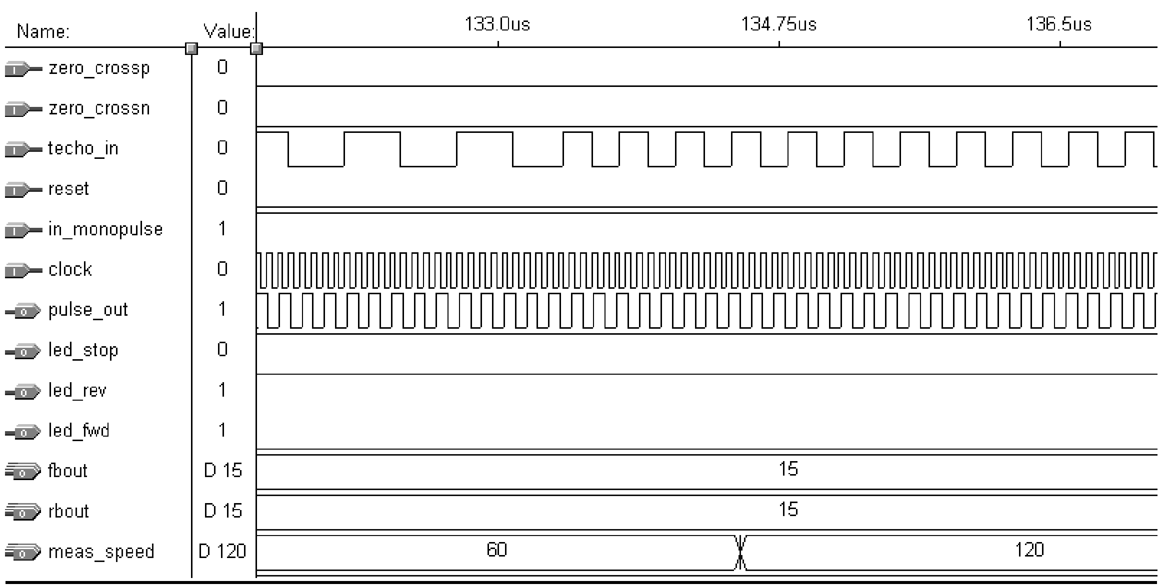

Fig. 6

6. 'Set speed increment' test case

Fig.7 demonstrates that the user-set speed is incremented correctly. The drive_line and sense_line are set to 3, which should indicate to the keypad FSM that INC has been pressed. Indeed, the key_state soon changes to key1, which is the internal representation for the INC key. The main FSM takes the appropriate actions, which result in the user-set speed being increased from 10rpm to 20rpm.

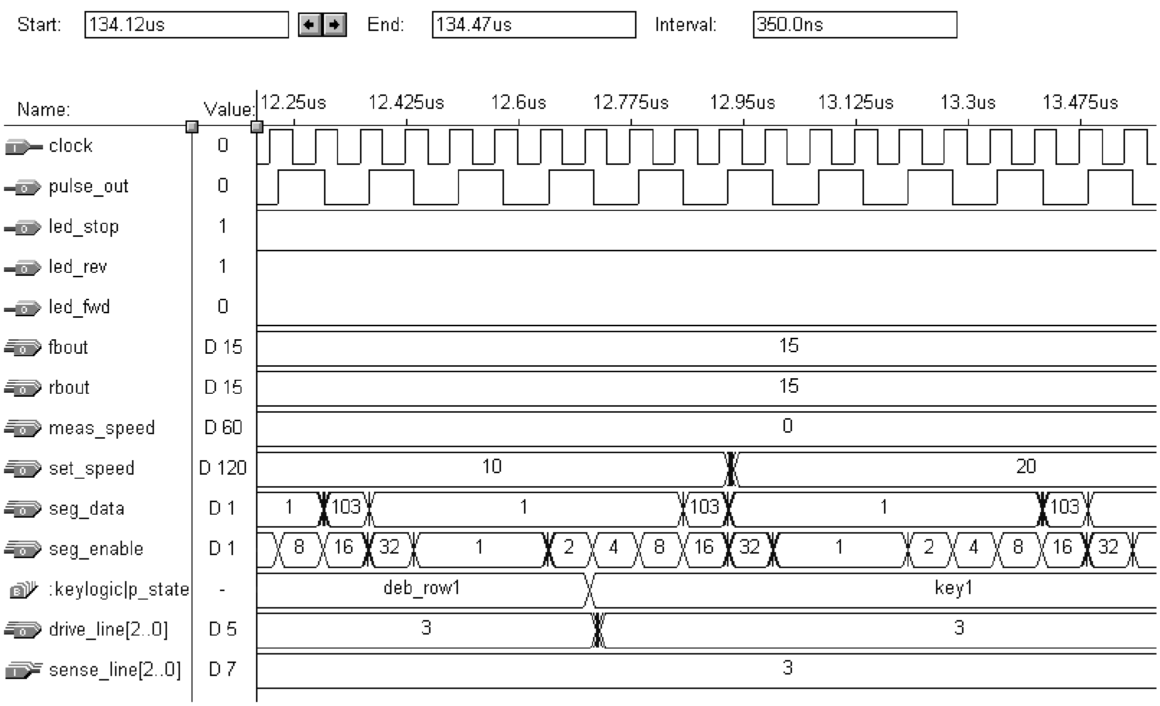

Fig. 7

Fig. 8 on the following page, shows that the display is correctly updated at approximately 18us, which is 3us after the user-set speed was incremented as discussed above.

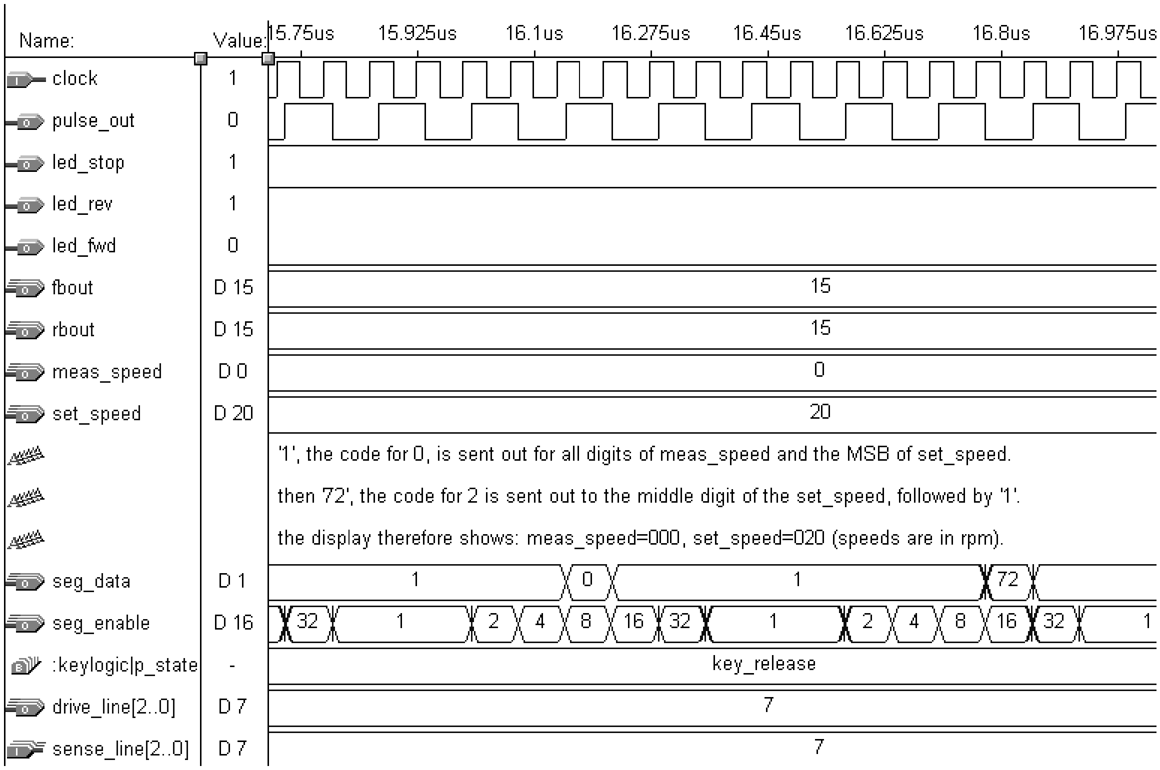

Fig. 8

7. Set speed decrement test case

Fig.9 on the following page, demonstrates that the user-set speed is decremented correctly. The drive_line is set to 5 while the sense_line is 3, which should indicate to the keypad FSM that DEC has been pressed. Indeed, the key_state soon changes to key2, which is the internal representation for the DEC key. The main FSM takes the appropriate actions, which result in the user-set speed being decreased from 130rpm to 120rpm.

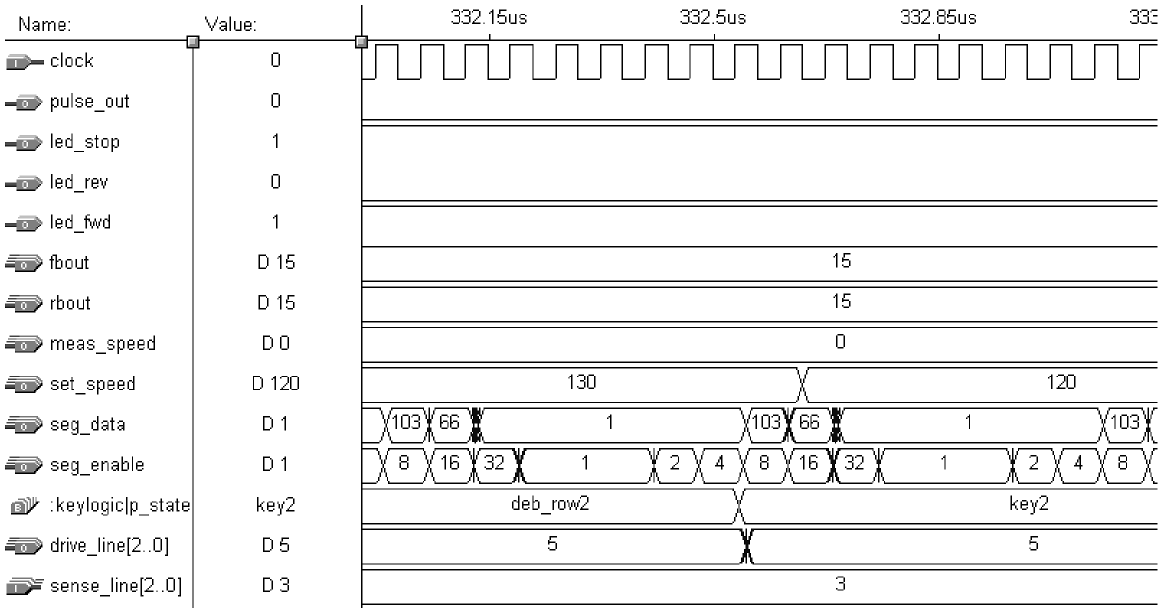

Fig. 9

Fig. 10 shows that the display is correctly updated at approximately 18us, which is 3us after the user-set speed was incremented as discussed above.

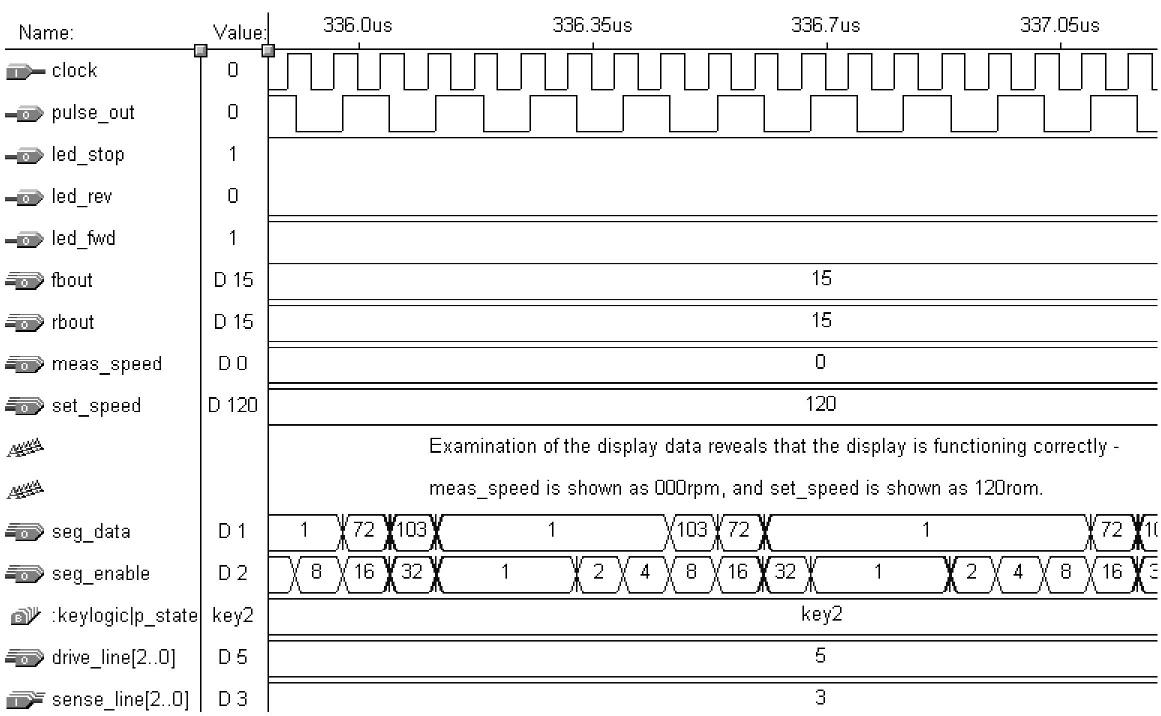

Fig. 10

The annotated waveforms on the following pages labeled Page A through H contain the remaining details of the design verification (Page F, the signal index page, is intentionally in front).

Design documentation

In this section, the VHDL code for the entire project is presented. Diagrams of the modules depicting their interface have been included where needed.

There are three high level modules as indicated in the following pages, and these three modules are instantiated by the top level entity whose diagram and code are presented immediately following this page.

The top level entity is named msc_project.vhd – it is essentially a container for all the modules of the project. The following three entities are instantiated by msc_project.vhd and thus are at the same hierarchy level:

1) Mainpa.vhd: this file controls all the modules related to the firing angle, SCR protection and firing, error calculation, and it also contains the main system FSM for the chip.
2) Keypad.vhd: this file contains the keypad controller which determines which key has been pressed, performs debouncing, etc. It has an internal, local FSM which is necessary to coordinate its internal functions, e.g. upon receiving an INC or DEC key, it updates the user-set speed value accordingly.
3) Dblock.vhd: this file contains the seven segment display module (which has an FSM inside it). It also contains a binary to BCD conversion module. This module receives the tacho input and converts it to a measurement of the motor's speed in rpm. It also receives the user-set speed from keypad.vhd and converts this data to seven segment fromat before displaying it.

Diagram of top level entity for the project

msc_project.vhd

```
--*************************************************************
-- MAIN BLOCK OF THE MOTOR SPEED CONTROLLER PROJECT (MSC_PROJECT)
--*************************************************************


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity msc_project is

-------------------------------------------
generic (     in_width     :      positive := 10;
              out_width    :      positive := 10;
                     sense_width :positive := 3;
                     drive_width  :      positive := 3;
                     pulse_width  :      positive :=  4;
                     seg_width    :      positive := 7;
                     segen_width  :      positive :=  6;
                     code_width   :      positive := 4;
                     speed_width :       positive := 10
              );
-------------------------------------------

port(clock                 : in   std_logic;
       reset               : in   std_logic;
       zero_crossn         : in   std_logic;
       zero_crossp         : in   std_logic;
       in_monopulse : in   std_logic;
       techo_in            : in   std_logic;
       sense_line          : in   std_logic_vector(sense_width-1 downto 0);
       pulse_out           : out  std_logic;
       led_stop            : out  std_logic;
       led_rev             : out  std_logic;
       led_fwd                    : out  std_logic;
       fbout               : out  std_logic_vector(pulse_width-1 downto 0);
    rbout                  : out  std_logic_vector(pulse_width-1 downto 0);
       seg_data            : out  std_logic_vector(seg_width-1 downto 0);
       seg_enable          : out  std_logic_vector(segen_width-1 downto 0);
       drive_line          : out  std_logic_vector(drive_width-1 downto 0)
       );

end msc_project;

architecture structure of msc_project is

--*********************************************
component mainpa           --Main logic of firing block, error calculation, SCR
selection and firing
------------------------------------
generic (out_width : integer := 10);
------------------------------------

port(clock                 : in   std_logic;
       reset               : in   std_logic;
       in_monopulse : in   std_logic;
       key_valid           : in   std_logic;
       zero_crossn         : in   std_logic;
       zero_crossp   : in   std_logic;
       fbout               : out  std_logic_vector(out_width-7 downto 0);
```

```
      rbout              : out  std_logic_vector(out_width-7 downto 0);
      pulse_out          : out  std_logic;
      key_ack                  : out  std_logic;
      led_stop           : out  std_logic;
      led_rev                  : out  std_logic;
      led_fwd            : out  std_logic;
      set_speed          : in   std_logic_vector(out_width-1 downto 0);
      current_speed : in   std_logic_vector(out_width-1 downto 0);
      key_in             : in   std_logic_vector(out_width-7 downto 0)
      );
end component;

--**********************************************
component keypad          --Key pad controller
-----------------------------------------------
generic(senseline_width   :      positive     :=3;
             driveline_width     :      positive     :=3;
             code_length         :      positive     :=4;
             output_length :      positive     :=10;
             debouncing_delay:   positive     :=30);
-----------------------------------------------
      port(
          clock,reset     : in      STD_LOGIC;
             sense_line          : in     STD_LOGIC_VECTOR(senseline_width-1
DOWNTO 0);
             key_ack         : in      STD_LOGIC;
             key_valid       : buffer  STD_LOGIC;
             drive_line          : out    STD_LOGIC_VECTOR(driveline_width-1
DOWNTO 0);
             key_code            : buffer  STD_LOGIC_VECTOR(code_length-1
DOWNTO 0);
             s_speed         : buffer  STD_LOGIC_VECTOR(output_length-1 DOWNTO
0)
             );
end component;

--**********************************************
component dblock          --Display bolck (includes speed measurement, bin2BCD
conversion and scanning display)
generic (speed_width :positive := 10;
             rpmout_width : positive :=10;
             seg_width     : positive :=7;
             segen_width   : positive :=6);
-----------------------------------------------
port(clock               : in   std_logic;
      reset              : in   std_logic;
      techo_in           : in   std_logic;
      usrs_speed         : in   std_logic_vector    (speed_width-1 downto 0);
      seg_data           : out  std_logic_vector    (seg_width-1 downto 0);
      seg_enable         : out  std_logic_vector    (segen_width-1 downto 0);
      rpmout             : buffer std_logic_vector  (rpmout_width-1 downto 0)
      );
end component;

--********************************************
--Signals definition
--********************************************
signal valid_key    :      std_logic;
signal code_key            :      std_logic_vector(code_width-1 downto 0);
signal ack_key             :      std_logic;
signal curnt_speed  :      std_logic_vector(speed_width-1 downto 0);
signal usrset_speed :      std_logic_vector(speed_width-1 downto 0);
--********************************************
```

```
begin
----------------------------------------------
mainbl : mainpa
----------------------------------------------
port map (
             clock               =>     clock,
             reset               =>     reset,
             in_monopulse =>     in_monopulse,
             key_valid           =>     valid_key,
             zero_crossn         =>     zero_crossn,
             zero_crossp  =>     zero_crossp,
             fbout               =>     fbout,
             rbout               =>     rbout,
             pulse_out           =>     pulse_out,
             key_ack                    =>     ack_key,
             led_stop            =>     led_stop,
             led_rev                    =>     led_rev,
             led_fwd             =>     led_fwd,
             set_speed           =>     usrset_speed,
             current_speed =>     curnt_speed,
             key_in              =>     code_key
                    );

-----------------------------------------------
keylogic : keypad
-----------------------------------------------
port map (
             clock               =>     clock,
             reset               =>     reset,
             sense_line          =>  sense_line,
             key_ack         =>  ack_key,
             key_valid       =>  valid_key,
             drive_line          =>     drive_line,
             key_code            =>     code_key,
             s_speed         =>  usrset_speed
                    );

-----------------------------------------------
displayblock : dblock
-----------------------------------------------
port map (
             clock               =>     clock,
             reset               =>     reset,
             techo_in            =>     techo_in,
             usrs_speed          =>     usrset_speed,
             seg_data            =>     seg_data,
             seg_enable          =>     seg_enable,
             rpmout              =>     curnt_speed
                    );
------------------------------------------------
end structure;
```

Diagram of high level entity – mainpa.vhd

mainpa.vhd

```
--- NEW FILE NAME : MAINPA.VHD
--- NEW FILE NAME : FAPN2.VHD
-- FILE NAME : FADMNPCn2.VHD
-- date :NOV 6,98
-- FIRING ANGLE DELAY(ALPH PDODUCER) MODULE FOR SINGLE PHASE
-- SCR 2F(WHERE F IS THE FREQUENCY OF LINE  AC, 60HZ)
-- FIRINIG ANGLE DELAY MODULE  IS COMBINATION OF ALL COMPONENTS
-- INPUTS: LOAD_DATA(ACTIVE HIGH)
-- INPUTS : TWO ZERO CROSS INPUTS FROM +ZERO CROSS AND -VE ZERO CROSS
-- CLCOK (RISING EDGE  ACTIVE) SYSTEM CLOCK
-- INPUTS: LOAD_DATAP, LOAD_DATAN(LOAD +VE AND -VE FIRING ANGLE
--              : VALUE IN TERMS OF BINARY COUNTS.
-- RESET: ACTIVE HIGH
-- OUTPUT: OUTPUT_READYP(ACTIVE HIGH)
--             :OUTPUT_READYN(ACTIVE HIGH)
-- FIRING PULSES OUTPUITS:TWO SCR FIRING PULSES OUT ONNE FOR +VE AC
-- CYCLE(OUTPUT_PFAP) AND ONE FOR -VE AC CYCLE(OUTPUT_PFAN) ARE
-- ACTIVE HIGH


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

--*********************************************************************
-- main program in which we are calling  main fsm and error calculation*
--*and scr firing angle timing generator circuit and scr decoder and   *
--*scr protection circuit , we are calling all as an component              *
--*********************************************************************


entity mainpa is
generic(out_width: integer:= 10);
port(
       clock,reset,in_monopulse,key_valid: in std_logic;
       zero_crossn,zero_crossp : in std_logic;
       fbout,rbout                        : out std_logic_vector(out_width-7
downto 0);
       pulse_out,key_ack          : out std_logic;
       led_stop,led_rev,led_fwd: out std_logic;
       set_speed, current_speed: in std_logic_vector(out_width-1 downto 0);
       key_in                             : in std_logic_vector(out_width-7
downto 0)

       );

end mainpa ;


architecture behav of mainpa is

--********************************************************************
--*main program in which we are calling error calculation entity  and *
--*firing angle timing generator circuit and also the scr decoder and *
--*and scr protection circuit.
                *
--********************************************************************

component fapn2 is
generic( out_width: integer:= 10);
```

```
port(
      clock,reset,mfor,mrev,mstop,in_monopulse: in std_logic;
      zero_crossn,zero_crossp   : in std_logic ;
      fbout,rbout                      : out std_logic_vector(out_width-7
downto 0);
      pulse_out                        : out std_logic;
      set_speed,current_speed   : in std_logic_vector(out_width-1 downto 0)

      );
end component ;

--*****************************************************************
--* calling of main fsm which is providing the top entity stop,forward*
--* and reversed  and after getting the input from main fsm the top   *
--* entity is doing all calculation.
         *
--* ****************************************************************

component mainfsm is
port(
             clock, reset, key_valid          : in   std_logic;
             key_in                                 : in   std_logic_vector(3
downto 0);
             stopout, fwdout, revout          : out  std_logic;
             stop_buf, fwd_buf, rev_buf : buffer std_logic;
             key_ack, led_stop, led_rev, led_fwd           : out  std_logic
             );

end component;

--*****************************************************************
--* inter connection signal b/w the entities
*
--*****************************************************************

             signal fwdoutfa                          : std_logic;
             signal revoutfa                          : std_logic;
             signal stopfa                     : std_logic;

begin

--*****************************************************************
--* port map of the main fsm with extenal and inside  the fapn2       *
--*****************************************************************

mainfsm1: mainfsm port map (
                                  clock        =>           clock,
                                  reset        =>           reset,
                                  key_valid    =>           key_valid,
                                  key_in              =>           key_in,
                                  stopout      =>           stopfa,
                                  fwdout              =>           fwdoutfa,
                                  revout              =>           revoutfa,
                                  key_ack      =>           key_ack,
                                  led_stop     =>           led_stop,
                                  led_rev      =>           led_rev,
                                  led_fwd      =>           led_fwd
                                  );

--*****************************************************************
--* port map of error calculation and firing angle and scr protection *
--* and scr decoder generator entity.
         *
```

```
--*********************************************************************


fapn2c: fapn2 port map(
                                  clock         =>          clock,
                                  reset         =>          reset,
                                  mfor          =>          fwdoutfa,
                                  mrev          =>          revoutfa,
                                  mstop         =>          stopfa,
                                  in_monopulse =>     in_monopulse,
                                  zero_crossn =>            zero_crossn,
                                  zero_crossp =>            zero_crossp,
                                  fbout         =>          fbout,
                                  rbout         =>          rbout,
                                  pulse_out     =>          pulse_out,
                                  set_speed     =>          set_speed,
                                  current_speed =>    current_speed
                                  );

end behav;
```

Diagram of main FSM for the project

(instantiated by mainpa.vhd)

mainfsm.vhd

```
--mainfsm.vhd

LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

ENTITY mainfsm IS
PORT(
             clock, reset, key_valid          : IN   STD_LOGIC;
             key_in                                  : IN   STD_LOGIC_VECTOR(3
downto 0);
             stopout, fwdout, revout          : OUT  STD_LOGIC;
             stop_buf, fwd_buf, rev_buf : buffer std_logic;
             key_ack, led_stop, led_rev, led_fwd           : OUT  STD_LOGIC);

END mainfsm;

ARCHITECTURE behavioural OF mainfsm IS

--constant key_stop: STD_LOGIC_VECTOR(3 downto 0):= "1001";
--constant key_fwd: STD_LOGIC_VECTOR(3 downto 0):= "1000";
--constant key_rev: STD_LOGIC_VECTOR(3 downto 0):= "0111";

type state_type is (START, READKEY, CHECKKEY, STOP, FWD, REV, SETFWD, SETREV,
ACK, HNDSHK, HNDSHK2);
signal state :      state_type;

BEGIN


FSM: process(clock, reset, state, key_valid, key_in)
begin
       if reset = '1' then state <= START;
       elsif rising_edge(clock) then
             case state is
                    when START =>
                           state <= READKEY;
                           stop_buf <= '1';
                           fwd_buf <= '0';
                           rev_buf <= '0';


                    when READKEY =>
                           if key_valid = '1' then state <= CHECKKEY;
                           else state <= READKEY;
                           end if;

                    when CHECKKEY =>
                           if key_in = "1001" then
                                                 state <= STOP;
                                  elsif key_in = "1000" then state <= FWD;
                                         elsif key_in = "0111" then state <=
REV;
                                                 else state <= ACK;
                           end if;

                    when STOP =>
                           state <= ACK;
                           stop_buf <= '1';
                           rev_buf <= '0';
```

```
                        fwd_buf <= '0';

                when FWD =>
                        if rev_buf = '1' then state <= ACK;
                                else state <= SETFWD;
                        end if;

                when REV =>
                        if fwd_buf = '1' then state <= ACK;
                                else state <= SETREV;
                        end if;

                when SETFWD =>
                        state <= ACK;
                        stop_buf <= '0';
                        fwd_buf <= '1';

                when SETREV =>
                        state <= ACK;
                        stop_buf <= '0';
                        rev_buf <= '1';

                when ACK =>
                        state <= HNDSHK;
                        key_ack <= '1';

                when HNDSHK =>
                        if key_valid = '0' then state <= HNDSHK2;
                        else state <= HNDSHK;
                        end if;
                        key_ack <= '1';

        when HNDSHK2 =>
                        state <= READKEY;
                        key_ack <= '0';

                when OTHERS =>
        end case;
    end if;

end process FSM;

OUTPUT: process (clock)
begin

    if rising_edge(clock) then
        if stop_buf = '1' then stopout <= '1';
                                        fwdout <= '0';
                                        revout <= '0';
                                        led_stop <= '0';
                                        led_fwd <= '1';
                                        led_rev <= '1';
                elsif fwd_buf = '1' then stopout <= '0';
                                                fwdout <= '1';
                                                revout <= '0';
                                        led_stop <= '1';
                                        led_fwd <= '0';
                                        led_rev <= '1';

                        elsif rev_buf = '1' then stopout <= '0';
                                                        fwdout <= '0';
                                                        revout <= '1';
                                        led_stop <= '1';
```

```
                                        led_fwd <= '1';
                                        led_rev <= '0';
            end if;

      end if;

end process OUTPUT;

END behavioural;
```

fapn2.vhd

```
--- NEW FILE NAME : FAPN2.VHD
-- FILE NAME : FADMNPCn2.VHD
-- date :NOV 6,98
-- FIRING ANGLE DELAY(ALPH PDODUCER) MODULE FOR SINGLE PHASE
-- SCR 2F(WHERE F IS THE FREQUENCY OF LINE  AC, 60HZ)
-- FIRINIG ANGLE DELAY MODULE  IS COMBINATION OF ALL COMPONENTS
-- INPUTS: LOAD_DATA(ACTIVE HIGH)
-- INPUTS : TWO ZERO CROSS INPUTS FROM +ZERO CROSS AND -VE ZERO CROSS
-- CLCOK (RISING EDGE  ACTIVE) SYSTEM CLOCK
-- INPUTS: LOAD_DATAP, LOAD_DATAN(LOAD +VE AND -VE FIRING ANGLE
--              : VALUE IN TERMS OF BINARY COUNTS.
-- RESET: ACTIVE HIGH
-- OUTPUT: OUTPUT_READYP(ACTIVE HIGH)
--             :OUTPUT_READYN(ACTIVE HIGH)
-- FIRING PULSES OUTPUITS:TWO SCR FIRING PULSES OUT ONNE FOR +VE AC
-- CYCLE(OUTPUT_PFAP) AND ONE FOR -VE AC CYCLE(OUTPUT_PFAN) ARE
-- ACTIVE HIGH


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

ENTITY FAPN2 IS
GENERIC( OUT_WIDTH: INTEGER:= 10);
PORT (
       CLOCK,RESET,MFOR,MREV,MSTOP,IN_MONOPULSE: IN STD_LOGIC;
       ZERO_CROSSN,ZERO_CROSSP : IN STD_LOGIC ;
       FBOUT,RBOUT : OUT STD_LOGIC_VECTOR(OUT_WIDTH-7 DOWNTO 0);
       PULSE_OUT: OUT STD_LOGIC;
       SET_SPEED,CURRENT_SPEED : IN STD_LOGIC_VECTOR(OUT_WIDTH-1 DOWNTO 0)

       );

END  FAPN2 ;


ARCHITECTURE BEHAV OF FAPN2 IS



COMPONENT FADMNP IS
GENERIC (PNDATA_WIDTH: INTEGER:= 12);
PORT(
       CLOCK ,ZERO_CROSSN,ZERO_CROSSP,RESET: IN STD_LOGIC ;
       OUTPUT_PFAN,OUTPUT_PFAP: OUT STD_LOGIC ;
       LOAD_DATAN: IN STD_LOGIC;
       LOAD_DATAP: IN STD_LOGIC;
       OUTPUT_READYP:OUT STD_LOGIC;
       OUTPUT_READYN:OUT STD_LOGIC;
       FA_DATAPN:  IN STD_LOGIC_VECTOR(PNDATA_WIDTH-1 DOWNTO 0)
       );

END COMPONENT;

COMPONENT ERCALN2 is
generic ( ERROR_width:integer:=12);
port(
       clock,REVM,FORM,STOPM,ODRP,ODRN,RESET,IN_MONOPULSE: in std_logic ;
--     ERROR_DONE,MAX_DELAY,count_max,stop_buff,chk_buff: BUFFER STD_LOGIC;
```

```
       LDFAP,LDFAN,PULSE_OUT: OUT STD_LOGIC;
--MONO_PULSE: BUFFER STD_LOGIC;
--     q_out: buffer std_logic_vector(15 downto 0);
       CURRENT_SPEED: IN std_logic_vector(ERROR_WIDTH-3 downto 0);
       SET_SPEED:   IN STD_LOGIC_VECTOR(ERROR_WIDTH-3 DOWNTO 0);
--     ERROR_CAL_BUF:BUFFER STD_LOGIC_VECTOR(ERROR_WIDTH-1 DOWNTO 0);
--     ERROR_CALA_BUF1:BUFFER STD_LOGIC_VECTOR(ERROR_WIDTH-1 DOWNTO 0);
--     ERROR_CALB_BUF2:BUFFER STD_LOGIC_VECTOR(ERROR_WIDTH-1 DOWNTO 0);
       ERROR_OUTPUT: buffer STD_LOGIC_VECTOR(ERROR_WIDTH-1 DOWNTO 0)
       );
end  COMPONENT;

COMPONENT PASCRDU  is

PORT(
             FORCU,REVCU,APFADU,ANFADU: IN STD_LOGIC ;
             FSCRBDATA,RSCRBDATA: OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
                    );
END COMPONENT ;

SIGNAL OR1AP: STD_LOGIC;
SIGNAL OR2AN: STD_LOGIC;
SIGNAL LDPS: STD_LOGIC;
SIGNAL LDNS: STD_LOGIC;
SIGNAL ERR_OUTPUT: STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL NFAS: STD_LOGIC;
SIGNAL PFAS: STD_LOGIC;
--SIGNAL REVM1: STD_LOGIC;
--SIGNAL FORM1: STD_LOGIC;
BEGIN
--COMPONENT ERCALN1
--generic ( ERROR_width:integer:=12);

ER_CALU: ERCALN2 port MAP (
       clock => CLOCK,
       REVM => MREV,
       FORM => MFOR,
       STOPM => MSTOP,
       ODRP => OR1AP,
       ODRN => OR2AN,
       RESET => RESET,
       PULSE_OUT => PULSE_OUT,
       IN_MONOPULSE => IN_MONOPULSE,
--     ERROR_DONE,MAX_DELAY,count_max,stop_buff,chk_buff: BUFFER STD_LOGIC;
       LDFAP=> LDPS,
       LDFAN=> LDNS,
--     q_out: buffer std_logic_vector(15 downto 0);
       CURRENT_SPEED => CURRENT_SPEED,
       SET_SPEED => SET_SPEED,
--     ERROR_CAL_BUF:BUFFER STD_LOGIC_VECTOR(ERROR_WIDTH-1 DOWNTO 0);
--     ERROR_CALA_BUF1:BUFFER STD_LOGIC_VECTOR(ERROR_WIDTH-1 DOWNTO 0);
--     ERROR_CALB_BUF2:BUFFER STD_LOGIC_VECTOR(ERROR_WIDTH-1 DOWNTO 0);
       ERROR_OUTPUT => ERR_OUTPUT
       );

FAM_UNIT: FADMNP PORT MAP(
       CLOCK=> CLOCK,
       ZERO_CROSSN => ZERO_CROSSN,
       ZERO_CROSSP => ZERO_CROSSP,
       RESET=> RESET,
       OUTPUT_PFAN=> PFAS,
    OUTPUT_PFAP=> NFAS,
       LOAD_DATAN=> LDNS,
```

```
        LOAD_DATAP=> LDPS,
        OUTPUT_READYP=> OR1AP,
        OUTPUT_READYN=> OR2AN,
        FA_DATAPN => ERR_OUTPUT
        );

SCRPU: PASCRDU PORT MAP (
                FORCU => MFOR,
                REVCU => MREV,
                APFADU => PFAS,
                ANFADU => NFAS,
                FSCRBDATA=>FBOUT,
                RSCRBDATA=>RBOUT
                        );
END BEHAV;
```

Diagram of high level entity – keypad.vhd

keypad.vhd

```
-- keypad.vhd  implements Keypad controller.The keypad works with drive
-- and sense lines.
-- Each of the four sense lines is pulled up with a 10kohm resister.
-- The drive lines drive "011"->"101"->"110" and once a sense of '0' is
-- seen on any of the sense_line then we know that a key has been pressed.
-- Debouncing occurs right after a key is pressed or released. Each
-- time the sense line inputs change because of a keypress or keyrelease,
-- we wait 1 ms before sampling the drive_line and sense_line values.
-- Hopefully, any oscillations would have died out during this time. A
-- counter is used to count from 0 up to delay each time the clock
-- signal goes high, since 1 ms delay is required, the maximum counter
-- value depends on the clock frequency.en the
-- Then the kaypad logic waits for the key to be released and for a
-- signal to continue normal operation.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;


ENTITY keypad IS
-----------------------------------------------
             GENERIC(senseline_width    :      positive:=3;    -- Width of sense
line
                          driveline_width     :      positive:=3;    -- Width
of drive line
                          code_length         :      positive:=4;    -- Width
of code generated.
                          output_length :      positive:=10;
                          debouncing_delay:   positive:=30);      -- Maximum
counter value to generate
-------------------------------------------------
                                                   -- required delay of
debouncing.

       PORT(
           clock,reset     : IN      STD_LOGIC;
             sense_line          : IN     STD_LOGIC_VECTOR(senseline_width-1
DOWNTO 0);
             key_ack         : IN      STD_LOGIC;
             key_valid       : BUFFER  STD_LOGIC;
             drive_line          : OUT    STD_LOGIC_VECTOR(driveline_width-1
DOWNTO 0);
             key_code            : BUFFER  STD_LOGIC_VECTOR(code_length-1
DOWNTO 0);
             s_speed         : BUFFER  STD_LOGIC_VECTOR(output_length-1 DOWNTO
0)
             );
END keypad;

ARCHITECTURE behaviour OF keypad IS
        TYPE state_type IS(start,first_row,second_row,third_row,deb_row1,

       deb_row2,deb_row3,key1,key2,key3,key4,key5,key6,key7,key8,key9,key_releas
e);
        TYPE        state_type2 IS
       (r_state,s_state,add_state,sub_state,wait_state);
```

```
--*****************************************************
--Signal Used
--******************************************************
       SIGNAL      p_state           :     state_type;
     SIGNAL  state      :  state_type2;
       SIGNAL      c_enable     :     STD_LOGIC;
       SIGNAL             c_reset            :     STD_LOGIC;
       SIGNAL             c_done       :     STD_LOGIC;
       SIGNAL             ld_reg       :     STD_LOGIC;
       SIGNAL             sclr         :     STD_LOGIC;
       SIGNAL             s_sclr       :     STD_LOGIC;
       SIGNAL             s_ldreg            :     STD_LOGIC;
       SIGNAL             operation    :     STD_LOGIC;
       SIGNAL             in_release   :     STD_LOGIC;
       SIGNAL      count     :  STD_LOGIC_VECTOR(7 DOWNTO 0);
       SIGNAL      int_code     :     STD_LOGIC_VECTOR(code_length-1 DOWNTO
0);
     SIGNAL  addertoff    :     STD_LOGIC_VECTOR(output_length-1 DOWNTO 0);
       SIGNAL      inc_ten      :     STD_lOGIC_VECTOR(output_length-1 DOWNTO
0):= "0000001010";
--********************************************************

       CONSTANT UPPER_LIMIT :STD_LOGIC_VECTOR(output_length-1 DOWNTO 0):=
"1111100111";
     CONSTANT LOWER_LIMIT :STD_LOGIC_VECTOR(output_length-1 DOWNTO 0):=
"0000000000";

-- Counter used for produncing delay before key is sampled. This delay is
required to wait for
-- bouncingto die.
--------------------------------------------------------------------------------
-------------------
COMPONENT counter
   generic( counter_limit:positive:= 8  );
    port (
       reset, Clock : in std_logic;
        enable       : In std_logic;
       Q : buffer std_logic_vector(7 DOWNTO 0);
       done : OUT STD_lOGIC

    ) ;
 END COMPONENT;
--------------------------------------------------------------------------------
--------------------
COMPONENT bit4reg
--------------------------------------------
             GENERIC(shiftreg_width:positive:=8);
--------------------------------------------
PORT(
       clock,clear     : IN STD_LOGIC;
       load               : IN STD_LOGIC;
              D                         : IN STD_LOGIC_VECTOR(shiftreg_width-1
DOWNTO 0);
              Q              : BUFFER   STD_LOGIC_VECTOR(shiftreg_width-1
DOWNTO 0)
              );
   END COMPONENT;
--------------------------------------------------------------------------------
------------------------
COMPONENT adder IS
------------------------------------------
      GENERIC( adder_width : integer:=  10);
------------------------------------------
```

```
      PORT(
              a          : IN     STD_LOGIC_VECTOR(adder_width-1 DOWNTO 0);
              b       : IN    STD_LOGIC_VECTOR(adder_width-1 DOWNTO 0);
              opt     : IN   STD_LOGIC;
              result       : OUT STD_LOGIC_VECTOR(adder_width-1 DOWNTO 0 )
              );

END COMPONENT;
--------------------------------------------------------------------------------
----------------------

BEGIN
inc_ten <= "0000001010";
--------------------------------------------------------------------------------
----------------------
-- Counter used for produncing delay before key is sampled. This delay is
required to wait for bouncing
-- to die.
count_deb:counter
-----------------------------------------------
generic map( counter_limit => debouncing_delay)
-----------------------------------------------
    port map(
             reset  =>     c_reset,
                    Clock  =>     clock,
             enable        =>     c_enable,
             Q             =>     open,
             done   =>     c_done
             ) ;
--------------------------------------------------------------------------------
---------------------
-- A register used to hold code of the key pressed. It is cleared when key
acknowledge is recieved.
valid_reg:bit4reg
----------------------------------------------
      GENERIC MAP(shiftreg_width => code_length)
----------------------------------------------
  PORT MAP(
                    clock  =>     clock,
                    clear  =>     sclr,
                    load   =>     ld_reg,
                    D             =>     int_code,
                    Q             =>     key_code
                    );
--------------------------------------------------------------------------------
----------------------
-- Main FSM to accomlish keypad controller.
 scanner:
PROCESS (clock,p_state,sense_line) IS
BEGIN
 IF ( reset = '1' ) THEN
      p_state <= start;


 ELSIF rising_edge(clock) THEN


    CASE p_state IS
             WHEN start =>                  -- All register and conter is
intialized and clear.
                          p_state <= first_row;
```

```
            WHEN first_row  =>            -- Scan first row of key_pad by
driving "011" on drine_line.
                          IF (sense_line="011"OR sense_line= "101" OR
sense_line ="110") THEN
                                  p_state <= deb_row1;
                          ELSIF (sense_line = "111") THEN
                                  p_state <= second_row;
                          ELSE
                                  p_state <= start;
                          END IF;

            WHEN second_row =>            -- Scan second row of key_pad by
driving "011" on drine_line.

                    IF (sense_line="011"OR sense_line= "101" OR sense_line
="110") THEN
                          p_state <= deb_row2;
                    ELSIF (sense_line = "111") THEN
                          p_state <= third_row;
                    ELSE
                          p_state <= start;
                    END IF;

            WHEN third_row =>               -- Scan third row of key_pad by
driving "011" on drine_line.

                    IF (sense_line="011"OR sense_line= "101" OR sense_line
="110") THEN
                        p_state <= deb_row3;
                    ELSIF (sense_line = "111") THEN
                          p_state <= first_row;
                    ELSE
                          p_state <= start;
                    END IF;

            WHEN deb_row1 =>            -- Key is detected on the first row of
the keypad.Stay in this
                                                          -- state until any
oscillations would have died out.
                    IF ( c_done = '1' ) THEN
                          If (sense_line="011") THEN
                                p_state  <= key1;
                          ELSIF (sense_line="101") THEN
                                p_state <= key4;
                          ELSIF (sense_line <="110" ) THEN
                                p_state <= key7;
                          ELSE
                                p_state <= start;
                          END IF;
                    ELSE
                          p_state <= deb_row1;
                    END IF;

            WHEN deb_row2 =>                -- Key is detected on the second
row of the keypad.Stay in this
                                                          -- state until any
oscillations would have died out.
            IF ( c_done = '1' ) THEN
                          If (sense_line="011") THEN
                                p_state  <= key2;
                          ELSIF (sense_line="101") THEN
                                p_state <= key5;
                          ELSIF (sense_line <="110" ) THEN
```

```
                                p_state <= key8;
                         ELSE
                                p_state <= start;
                         END IF;
             ELSE
                         p_state <= deb_row2;
                  END IF;

             WHEN deb_row3 =>                 -- Key is detected on the third
row of the keypad.Stay in this
                                                          -- state until any
oscillations die out.
             IF (c_done = '1' ) THEN
                         If (sense_line="011") THEN
                                p_state  <= key3;
                         ELSIF (sense_line="101") THEN
                                p_state <= key6;
                         ELSIF (sense_line <="110" ) THEN
                                p_state <= key9;
                         ELSE
                                p_state <= start;
                         END IF;
                  ELSE
                         p_state <= deb_row3;
                  END IF;

             WHEN key1 =>                     -- Key 1 has been detected after
oscillations died out.
                                                          -- Generate code
for key 1.
                  IF sense_line = "111" THEN
                         p_state <= key_release;
             ELSE
                       p_state <= key1 ;
                  END IF;

       WHEN key4=>                            -- Key 4 has been detected after
oscillations died out.
                                                          -- Generate code
for key 4.
                  IF sense_line = "111" THEN
                         p_state <= key_release;
                  ELSE
                       p_state <= key4;
                  END IF;

          WHEN key7=>                                 -- Key 7 has been detected
after oscillations died out.
                                                          -- Generate code
for key 7.
             IF  sense_line = "111"   THEN
                         p_state <= key_release;
                  ELSE
                       p_state <= key7;
                  END IF;

       WHEN key2=>                            -- Key 2 has been detected after
oscillations died out.
                                                          -- Generate code
for key 2.
                  IF (sense_line = "111") THEN
                         p_state <= key_release;
                  ELSE
```

```
                        p_state <= key2;
                    END IF;

       WHEN key5=>                               -- Key 5 has been detected after
oscillations died out.
                                                              -- Generate code
for key 5.
                    IF (sense_line = "111" ) THEN
                          p_state <= key_release;
                    ELSE
                        p_state <= key5;
                    END IF;

       WHEN key8=>                               -- Key 8 has been detected after
oscillations died out.
                                                              -- Generate code
for key 8.
                    IF (sense_line = "111" ) THEN
                          p_state <= key_release;
                    ELSE
                        p_state <= key8;
                    END IF;

        WHEN key3=>                              -- Key 3 has been detected after
oscillations died out.
                                                              -- Generate code
for key 3.
                          IF sense_line = "111" THEN
                          p_state <= key_release;
                    ELSE
                        p_state <= key3;
                    END IF;

        WHEN key6=>
                                                              -- Key 6 has been
detected after oscillations died out.
                                                              -- Generate code
for key 6.
                    IF sense_line = "101"  THEN
                          p_state <= key_release;
                    ELSE
                        p_state <= key6;
                    END IF;

             WHEN key9=>                              -- Key 9 has been detected
after oscillations died out.
                                                              -- Generate code
for key 9.
                    IF (sense_line = "111" ) THEN
                          p_state <= key_release;
                    ELSE
                        p_state <= key9;
                  END IF;
             WHEN key_release =>
                                                              -- Key has beec
releaded.stay in state until any oscillations
                                                              -- die out.
              IF (c_done = '1' ) THEN
                          IF key_ack = '1' THEN
                                p_state <= start;
                          ELSE
                                p_state <= key_release;
```

```
                        END IF;
                ELSE
                        p_state <= key_release;
                END IF;
        END case;
        END IF;
END PROCESS scanner;
-------------------------------------------------------------------------------
WITH p_state  SELECT
        drive_line <= "111" WHEN start | key_release,
                          "011" WHEN first_row | deb_row1 | key1 | key4 |
key7,
                          "101" WHEN second_row | deb_row2 | key2 | key5 |
key8,
                          "110" WHEN third_row | deb_row3 | key3 | key6 |
key9;
-------------------------------------------------------------------------------
coding:
WITH p_state SELECT
         int_code <=  "0001" WHEN key1,
                          "0010" WHEN key2,
                          "0011" WHEN key3,
                          "0100" WHEN key4,
                  "0101" WHEN key5,
                          "0110" WHEN key6,
                          "0111" WHEN key7,
                          "1000" WHEN key8,
                          "1001" WHEN key9,
                          "0000" WHEN OTHERS;
 ------------------------------------------------------------------------------
validating:
WITH p_state SELECT
         key_valid <= '1' WHEN key1 | key2 | key3 | key4 | key5 | key6 | key7 |
key8 | key9 | key_release,
                          '0' WHEN OTHERS;
-------------------------------------------------------------------------------
WITH p_state SELECT
     c_reset <= '1'   WHEN start | key1 | key2 | key3 | key4 | key5 | key6 |
key7 | key8 | key9,
                '0'   WHEN OTHERS;
-------------------------------------------------------------------------------
WITH p_state SELECT
        c_enable <= '1'   WHEN deb_row1 | deb_row2 | deb_row3 | key_release,
                '0'   WHEN OTHERS;
-------------------------------------------------------------------------------
WITH p_state SELECT
         ld_reg <=    '1' WHEN key1 | key2 | key3 | key4 | key5 | key6 | key7 |
key8 | key9,
                          '0' WHEN OTHERS;
-------------------------------------------------------------------------------
WITH p_state SELECT
         sclr   <=    '1' WHEN start,
                          '0' WHEN OTHERS;

-------------------------------------------------------------------------------
k_reg:bit4reg
             GENERIC MAP(shiftreg_width => output_length)
        PORT MAP(
        clock => clock ,clear => s_sclr,
                load  => s_ldreg,
                D => s_speed,
        Q => addertoff
         );
```

```
---------------------------------------------------------------------------
 k_adder: lpm_add_sub
   GENERIC MAP(LPM_WIDTH => output_length,
      LPM_REPRESENTATION => "UNSIGNED"
             )
   PORT MAP(dataa =>addertoff, datab => inc_ten,
      add_sub => operation,
     result => s_speed
         );
---------------------------------------------------------------------------
inc_dec:
PROCESS (clock,reset,int_code)
BEGIN
       IF ( reset = '1' ) THEN
              state <= s_state;

       ELSIF rising_edge(clock) THEN
              CASE state IS
                     WHEN r_state =>
                                   state <= s_state;
                     WHEN s_state =>
                      if ( key_code = "0001" ) and (s_speed /= UPPER_LIMIT)THEN
                                   state <= add_state;
                            elsif ( key_code = "0010" ) and (s_speed /=
LOWER_LIMIT) THEN
                                   state <= sub_state;
                            else
                                   state <= s_state;
                            end if;
                     WHEN add_state=>
                     state <= wait_state;
                     WHEN sub_state =>
                     state <= wait_state;
                     WHEN wait_state=>
                            if ( key_valid = '1' ) THEN
                                   state <= wait_state;
                            else
                                   state <= s_state;
                            end if;


                     END CASE;
              END IF;

       END PROCESS inc_dec;
---------------------------------------------------------------------------
       WITH state SELECT
               s_sclr  <= '1' WHEN r_state,
                                   '0' WHEN OTHERS;

---------------------------------------------------------------------------
       WITH state SELECT
               operation <= '0' WHEN sub_state,
                                     '1' WHEN OTHERS;
---------------------------------------------------------------------------
    WITH state SELECT
        s_ldreg <= '1' WHEN add_state | sub_state,
                            '0' WHEN OTHERS;

---------------------------------------------------------------------------
END behaviour;
```

Counter.vhd

```
-- Arshid Mahmood Khan.
-- I.D # 358261.
-- EE 552 FALL 98.
-- counter.vhd  Synchronous down counter with asynchronous rese



library ieee;

use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter is
    generic( counter_limit:positive:= 8              );
    port (
        reset, Clock : in std_logic;
        enable       : In std_logic;
        Q : buffer std_logic_vector(7 DOWNTO 0);
        done : OUT STD_lOGIC
    ) ;
end counter;

architecture behav of counter is
begin


ctr1 :process(clock)
       begin
                    if rising_edge(clock)  then
                    if reset = '1' then
                           Q <= (0 => '1', others => '0');
                    done <= '0';
                 elsif Q =  counter_limit-1 Then
                           done <= '1';
             elsif enable = '1' THEN
                           Q <= Q + "01";
                     end if;
        end if;
    end process ctr1;

end  behav;
```

Bit4reg.vhd

```
-- proj_8breg.vhd.
-- 8 bit regester.


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

ENTITY bit4reg IS

		GENERIC(shiftreg_width:positive:=8);
	PORT(
		 clock,clear	: IN STD_LOGIC;
		 load		: IN STD_LOGIC;
		 D				: IN
STD_LOGIC_VECTOR(shiftreg_width-1 DOWNTO 0);
		 Q		: BUFFER	STD_LOGIC_VECTOR(shiftreg_width-1
DOWNTO 0)
		 );

END bit4reg;
ARCHITECTURE behaviour OF bit4reg IS
BEGIN
	eightbitreg:
	PROCESS (clock)
	BEGIN
		IF rising_edge(clock) THEN
			IF clear = '1' THEN
			Q <= ( others => '0');
			ELSIF load = '1' THEN
				Q <= D;

			END IF;
		END IF;
	END PROCESS eightbitreg;
END behaviour;
```

Diagram of high level entity – dblock.vhd

Sp_mment.vhd

```
-- Arshid Mahmood Khan.
--
-- sp_mment.vhd
-- sp_mment.vhd is TOP LEVEL file for speed measurement block

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY sp_mment IS
       PORT(
              clock,reset         : IN   STD_LOGIC;
              tacho_input     : IN    STD_LOGIC;
              rpm_out                 : OUT    STD_LOGIC_VECTOR(9 DOWNTO 0));
END sp_mment;

ARCHITECTURE mix OF sp_mment IS

COMPONENT tech_ctr
    generic( counter_width:positive:= 8);
      port (
     reset, tacho_in : in std_logic;
     enable : in std_logic;
     Q : buffer std_logic_vector(counter_width-1 downto 0)
    ) ;
end COMPONENT;

 COMPONENT counter is
    generic( counter_limit:positive:= 8);
    port (
        reset, Clock : in std_logic;
        enable       : In std_logic;
        Q : buffer std_logic_vector(14 DOWNTO 0);
        done : OUT STD_lOGIC
    ) ;
end COMPONENT;

SIGNAL tacho_reg : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL reg_mult,dec_sixty  : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL int_reset: STD_LOGIC;
SIGNAL enable_ctr,load_reg,ctr_done : STD_LOGIC;
signal ld_op_reg : STD_LOGIC;
signal tmp_rpm_out : STD_LOGIC_VECTOR(15 DOWNTO 0);
signal rpm : STD_LOGIC_VECTOR(9 DOWNTO 0);
signal low :std_logic:= '0';

type state_type is (reset_state, mment_state, hold_state, done_state,
load_state, dumb_state);
signal state :state_type;

BEGIN

       dec_sixty <= "00111100";

tacho_pulse_ctr: tech_ctr
    generic MAP( counter_width => 8)
      port MAP(
        reset =>int_reset,
```

```
            tacho_in=>tacho_input,
        enable => enable_ctr,
        Q => tacho_reg
    ) ;

comp_pulses_reg: lpm_ff
      generic map(LPM_WIDTH =>8)
      port map(data => tacho_reg,
            clock => clock,
            enable => load_reg,
            q => reg_mult);

     multilplication:lpm_mult
            GENERIC MAP(LPM_WIDTHA => 8,
            LPM_WIDTHB => 8,
            LPM_WIDTHS => 8,
            LPM_WIDTHP => 16,
            LPM_REPRESENTATION => "UNSIGNED",
            LPM_PIPELINE => 0,
            LPM_TYPE => L_MULT,
            LPM_HINT => UNUSED)
            PORT MAP(dataa =>reg_mult,
            datab => dec_sixty,
            result => tmp_rpm_out);

      delay_counter : counter
    generic MAP( counter_limit => 10)
    port MAP(
        reset =>int_reset , Clock => clock,
        enable =>enable_ctr,
        Q => open,
        done =>ctr_done
    ) ;


rpm <= tmp_rpm_out(9 downto 0);

comp_op_reg: lpm_ff
      generic map(LPM_WIDTH =>10)
      port map(data => rpm,
            enable => ld_op_reg,
            clock => clock,
            q => rpm_out);


mment_FSM:
    PROCESS (state, clock ,reset, ctr_done)
    BEGIN
        if reset = '1' then
                state <= reset_state;
        elsif rising_edge(clock) then

    CASE state IS
      WHEN reset_state =>
                  state <= dumb_state;

            WHEN dumb_state =>
                  state <= mment_state;

      WHEN mment_state=>
            IF ctr_done = '0' THEN
                        state <= mment_state;
                  ELSE
```

```
                          state <= hold_state;
                    END IF;

        WHEN hold_state =>
                    state <= done_state;

        WHEN  done_state =>
              state <= load_state;

              WHEN load_state =>
                    state <= reset_state;

        WHEN others =>
         END CASE;
end if;

END PROCESS mment_FSM;

with state select
        int_reset <= '1' when reset_state|dumb_state,
                          '0' when others;

with state select
        enable_ctr <= '1' when mment_state,
                          '0' when others;

with state select
        load_reg <= '1' when hold_state,
                          '0' when others;

with state select
        ld_op_reg <= '1' when load_state,
                          '0' when others;


END mix;
```

Conv.vhd

```
--**************************************************
--conv.vhd
--top level file for the bin2BCD converter module
--***************************************************

LIBRARY lpm;
USE lpm.lpm_components.ALL;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY conv IS                                  --Entity description
-----------------------------------------
generic (signal_width : positive := 10;
              speed_width  :     positive := 10
             );
-----------------------------------------
      PORT(
             clock, reset        : IN   STD_LOGIC;
             meas_sp                           : IN
      STD_LOGIC_VECTOR(speed_width-1 downto 0);
             set_sp                     : IN   STD_LOGIC_VECTOR(speed_width-1
downto 0);
             meas_spout                 : OUT  STD_LOGIC_VECTOR(speed_width+1
downto 0);
             set_spout                  : OUT  STD_LOGIC_VECTOR(speed_width+1
downto 0)
             );
END conv;
--**************************************************
ARCHITECTURE mixed OF conv IS
--**************************************************
component convfsm is
             PORT(
                    clock        : IN          STD_LOGIC;
                    re_set              : IN          STD_LOGIC;
                    done         : IN          STD_LOGIC;
                    measrd       : BUFFER      STD_LOGIC;
                    loadip       : OUT         STD_LOGIC;
                    ack_done     : OUT         STD_LOGIC;
                    start        : OUT         STD_LOGIC;
                    sel                 : OUT         STD_LOGIC;
                    op_measrd    : OUT         STD_LOGIC;
                    op_usr       : OUT         STD_LOGIC
                    );
end component;
--***************************************************
component mux_conv is
-------------------------------------
generic (data_width : positive := 10);
-------------------------------------
      port(
             A                   : IN          STD_LOGIC_VECTOR(data_width-1
downto 0);
             B                   : IN          STD_LOGIC_VECTOR(data_width-1
downto 0);
             sel                 : IN          STD_LOGIC;
             Y                   : BUFFER      STD_LOGIC_VECTOR(data_width-1
downto 0));
end component;
--****************************************************
component onebin2bcd is
```

```
-------------------------------------
generic (data_width : positive := 10;
              BCD_width   : positive := 4);
--------------------------------------
   port(
                    Clock        :      IN STD_LOGIC;
                    Reset        :      IN STD_LOGIC;
                    start        :      IN STD_LOGIC;
                    Ack_done     :      IN STD_LOGIC;
                    Done         :      OUT STD_LOGIC;
                    Data_In      :      IN STD_LOGIC_VECTOR       (data_width+1
downto 0);
                    Bcd_1        :      BUFFER STD_LOGIC_VECTOR   (BCD_width-1
downto 0);
                    Bcd_2        :      BUFFER STD_LOGIC_VECTOR   (BCD_width-1
downto 0);
                    Bcd_3        :      BUFFER STD_LOGIC_VECTOR   (BCD_width-1
downto 0);
                    bcd_data_out:       out std_logic_vector      (data_width+1
downto 0)
             );
END component;
--***************************************************
--Signals used
--***************************************************
signal load_ip             : std_logic;
signal tmpsel              : std_logic;
signal tmpstart            : std_logic;
signal tmp_done            : std_logic;
signal tmp_ackdone  : std_logic;
signal op_measrd    : std_logic;
signal op_usr       : std_logic;
signal tmpmeas_sp   : STD_LOGIC_VECTOR(signal_width-1 DOWNTO 0);
signal tmpset_sp    : STD_LOGIC_VECTOR(signal_width-1 DOWNTO 0);
signal muxout       : STD_LOGIC_VECTOR(signal_width-1 DOWNTO 0);
signal bcd_out             : STD_LOGIC_VECTOR(signal_width+1 DOWNTO 0);
signal tmpmuxout    : STD_LOGIC_VECTOR(signal_width+1 DOWNTO 0);
signal pad                 : STD_LOGIC_VECTOR(signal_width-9 downto 0);
--****************************************************

BEGIN                                 --     mapping the components

pad <= "00";
--****************************************************
reg_measspeed: lpm_ff
--****************************************************
       generic map(LPM_WIDTH =>10)
-------------------------------
       port map(     data   =>     meas_sp,
                            clock  =>     clock,
                            sload  =>     load_ip,
                            q             =>     tmpmeas_sp
                    );
--******************************************************
reg_setspeed: lpm_ff
--******************************************************
       generic map(LPM_WIDTH =>10)
--------------------------------
       port map(     data   =>     set_sp,
                            clock  =>     clock,
                            sload  =>     load_ip,
                            q             =>     tmpset_sp
                    );
```

```
--****************************************************
comp_mux: mux_conv
--****************************************************
      port map (   A           =>    tmpmeas_sp,
                         B           =>    tmpset_sp,
                         sel   =>    tmpsel,
                         Y           =>    muxout
                   );
--****************************************************
tmpmuxout <= pad & muxout;
--****************************************************
comp_bin2bcd: onebin2bcd
      port map(    Clock       =>    clock,
                         Reset       =>    reset,
                         Start       =>    tmpstart,
                         Ack_done    =>    tmp_ackdone,
                         Done        =>    tmp_done,
                         Data_In     =>    tmpmuxout,
                         bcd_data_out=>     bcd_out
                   );
--****************************************************
comp_fsm: convfsm
--****************************************************
      port map (   clock       =>    clock,
                         re_set            =>    reset,
                         done        =>    tmp_done,
                         loadip            =>    load_ip,
                         ack_done    =>    tmp_ackdone,
                         start       =>    tmpstart,
                         sel         =>    tmpsel,
                         op_measrd   =>    op_measrd,
                         op_usr            =>    op_usr
                   );
--****************************************************
out_measspeed: lpm_ff
--****************************************************
      generic map(LPM_WIDTH=>12)
------------------------------
      port map(    data  =>    bcd_out,
                         clock =>    clock,
                         enable      =>    op_measrd,
                         q           =>    meas_spout
                   );
--****************************************************
out_setspeed: lpm_ff
--****************************************************
      generic map(LPM_WIDTH =>12)
-------------------------------
      port map(    data  =>    bcd_out,
                         clock =>    clock,
                         enable      =>    op_usr,
                         q           =>    set_spout
                   );
-------------------------------------------------------
END mixed;
```

Display.vhd

```
--**************************************************
--display.vhd
--display module
--**************************************************

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY display IS
----------------------------------------
generic (speed_width      : positive := 10;
              seg_width          : positive := 7;
              segen_width : positive := 6;
              bcdcrnt_width      : positive := 12;
              bcdset_width : positive := 12;
              select_width : positive := 3;
              tmpselect_width: positive := 3;
              bcd_dig_width      : positive := 4
             );
----------------------------------------
      PORT(
             clock, reset : IN   STD_LOGIC;
             crnt_speed         : IN   STD_LOGIC_VECTOR(speed_width+1 downto
0);
             set_speed          : IN   STD_LOGIC_VECTOR(speed_width+1 downto
0);
             segdata                   : OUT  STD_LOGIC_VECTOR(seg_width-1
downto 0);
             enable             : OUT  STD_LOGIC_VECTOR(segen_width-1 downto
0)
             );
END display;
--**************************************************
ARCHITECTURE mixed OF display IS
--**************************************************
component dispfsm is
PORT(
             startdisplay : IN         STD_LOGIC;
             countdone          : IN         STD_LOGIC;
             clock              : IN         STD_LOGIC;
             reset              : IN         STD_LOGIC;
             load               : BUFFER     STD_LOGIC;
             cnt_reset          : BUFFER     STD_LOGIC
      );
end component;
--**************************************************
component modcount is
----------------------------------------
generic (counter_width : positive:= 3);
----------------------------------------
port(
             reset  : in std_logic;
             clock  : in std_logic;
             Q             : buffer std_logic_vector(counter_width-1 downto 0)
             );
end component;
--**************************************************

component mux is
```

```
generic (in_width : positive := 4;                   --input width
             out_width : positive := 4);                    --output width
       PORT(
             A            : IN        STD_LOGIC_VECTOR(in_width-1 downto 0);
             B            : IN        STD_LOGIC_VECTOR(in_width-1 downto 0);
             C            : IN        STD_LOGIC_VECTOR(in_width-1 downto 0);
             D            : IN        STD_LOGIC_VECTOR(in_width-1 downto 0);
             E            : IN        STD_LOGIC_VECTOR(in_width-1 downto 0);
             F            : IN        STD_LOGIC_VECTOR(in_width-1 downto 0);
             sel          : IN        STD_LOGIC_VECTOR(2 downto 0);
             Y            : BUFFER    STD_LOGIC_VECTOR(out_width-1 downto
0));
END component;

--*****************************************************
component sevenseg is
----------------------------------------
generic (seg_width        : positive := 7;
             BCD_width          : positive  := 4);
----------------------------------------
    port (
        clock :      in std_logic;
             bcddata :    in std_logic_vector (BCD_width-1 downto 0);
             segout       :      out std_logic_vector(seg_width-1 downto 0)
             );
END component;
--*****************************************************
--Signal Used
--*****************************************************
signal count_done   : std_logic;
signal load_inputs  : std_logic;
signal reset_cnt    : STD_LOGIC;
signal tmpreset            : STD_LOGIC;
signal startdisp    : STD_LOGIC;
signal bcdcrnt             : STD_LOGIC_VECTOR  (bcdcrnt_width-1 DOWNTO 0);
signal bcdset       : STD_LOGIC_VECTOR  (bcdset_width-1 DOWNTO 0);
signal selection    : STD_LOGIC_VECTOR  (select_width-1 DOWNTO 0);
signal tmpselection : STD_LOGIC_VECTOR  (tmpselect_width-1 DOWNTO 0);
signal bcd_dig             : STD_LOGIC_VECTOR  (bcd_dig_width-1 DOWNTO 0);
signal tmpsegdata   : STD_LOGIC_VECTOR  (seg_width-1 DOWNTO 0);
signal tmpenable    : STD_LOGIC_VECTOR  (segen_width-1 DOWNTO 0);
signal tmpenable2   : STD_LOGIC_VECTOR  (segen_width-1 DOWNTO 0);
--*****************************************************
BEGIN

startdisp <= '1';
-----------------------------------------------------
reg_crntspeed: lpm_ff
-------------------------------
       generic map(LPM_WIDTH =>12)
-------------------------------
       port map(
                    data   =>     crnt_speed,
                    clock  =>     clock,
                    enable        =>     load_inputs,
                    q             =>     bcdcrnt
                    );
--------------------------------------------------------
reg_setspeed: lpm_ff
-------------------------------
       generic map(LPM_WIDTH =>12)
-------------------------------
       port map(
```

```
                          data          =>     set_speed,
                          clock         =>     clock,
                          enable               =>     load_inputs,
                          q                    =>     bcdset);
                          tmpreset      <= (reset_cnt or reset
                   );
---------------------------------------------------------
count: modcount
       port map (
                          reset         =>     tmpreset,
                          clock         =>     clock,
                          Q                    =>     selection);
                          count_done    <= (selection(2) and selection(0)
                   );
----------------------------------------------------------
localfsm: dispfsm
       port map (
                          startdisplay=>      startdisp,
                          countdone     =>     count_done,
                          clock         =>     clock,
                          reset         =>     reset,
                          load          =>     load_inputs,
                          cnt_reset     =>     reset_cnt
                   );
-----------------------------------------------------------
multiplex: mux

       PORT MAP(A    => bcdcrnt(11 downto 8),
                   B      => bcdcrnt(7 downto 4),
                   C      => bcdcrnt(3 downto 0),
                   D      => bcdset(11 downto 8),
                   E      => bcdset(7 downto 4),
                   F      => bcdset(3 downto 0),
                   sel => selection,
                   Y      => bcd_dig);

------------------------------------------------------------
decoder: lpm_decode
------------------------------
GENERIC MAP  (LPM_WIDTH   =>     3,
              LPM_DECODES =>     6)
------------------------------
       PORT MAP(
                   data   =>     selection,
                   eq            =>     tmpenable
                   );
------------------------------------------------------------
bcdto7seg: sevenseg
       port map (
              clock  => clock,
              bcddata => bcd_dig,
              segout       => tmpsegdata);
-------------------------------------------------------------
reg_segdata: lpm_ff
-------------------------------
       generic map(LPM_WIDTH => 7)
-------------------------------
       port map(
              data   =>     tmpsegdata,
              clock  =>     clock,
              q             =>     segdata
                   );
------------------------------------------------------------
```

```
reg_enable: lpm_ff
------------------------------
      generic map(LPM_WIDTH => 6)
------------------------------
      port map(
            data   =>     tmpenable,
            clock  =>     clock,
            q             =>     enable
                  );
---------------------------------------------------------
END mixed;
```

All VHDL files for the project (in alphabetical order)

…all files could not be included with this report. There are approximately forty files associated with this project, and including them in this document created a file over 100 pages long. We are fairly apprehensive of being able to print out the present report, which is already close to sixty pages in length.

The entire VHDL codebase for this project can be made available in the same directory as the PostScript version of this report on the WWW if this is desired.