Design Report

DESIGN A PROCESSOR FOR CONTROLLING MIXING PROCESS (CMP)

EEX7436

Processor Design

(Design Report)

By

S.A.P. Kavinda

617143597

Submitted to

Department of Electrical and Computer Engineering

Faculty of Engineering Technology

The Open University of Sri Lanka

on

12/21/2023

TABLE OF CONTENTS

1 INTRODUCTION

Mixing is defined as the reduction of inhomogeneity in order to achieve a desired process result. The inhomogeneity can be concentration, phase or temperature. Also, secondary effects such as mass transfer, reaction, and phase properties are usually critical objectives[1]. Mixing operations are necessary step for wide range of industries such as fine chemical, agrichemical, pharmaceuticals, petrochemicals, Biotechnology, polymer processing, paint and automotive, food, cosmetics, pulp and paper, mineral processing and drinking water. In all of these industries, the mixing problem mainly focuses on two factors, known as key variables in mixing.

- Time available to accomplish mixing (The time scale)
- Required scale of homogeneity (The length scale)

Both factors are different in each industry as well as each consumer product. For example, in food industry the time of mixing is depends on the recipe of the food and petroleum and pharmaceutical industries uses some catalyst to reduce the mixing time. Homogeneity is also very dependent on the industry for example in food industry some mixers use to just mix some ingredients without any chemical reactions. But in pharmaceuticals industry homogeneity is critical aspect which consider concentration variation, temperature variations and other filed specific factors also.

To do the mixing operations, process or chemical engineers use mixing machines. There are wide range of mixing machines are available in market that made to do specific things. From the traditional stirred tanks, baffling, the full range of impellers, and other tanks, pipeline mixers, High viscosity mixers, double motion mixers are uses in different industries with suitable changes as suitable to the requirements. But in controlling viewpoint, all these mixers have common attributes such as vessel controlling, sensor measuring, motor controlling (expect in static mixers). Mixing operation is vary with the type of ingredient that intent to mixing. For example, solid-solid, solid-liquid, liquid-liquid, high-viscosity liquid mixing and gas mixing required deferent types of controlling methods. Also, for the evaluate mixing process, process engineers are using field specific theories like Residence time distribution theory. Mean value and variance of some measurement is also used in most of theories as well as directly evaluate the mixing process. For obtain the data for calculations, sensors are using. Most of the cases use a array of sensors instead of use one sensor. For example, Figure 1 shows the Strain gauges mounted on the mixer shaft are a popular and reliable method of measuring torque.

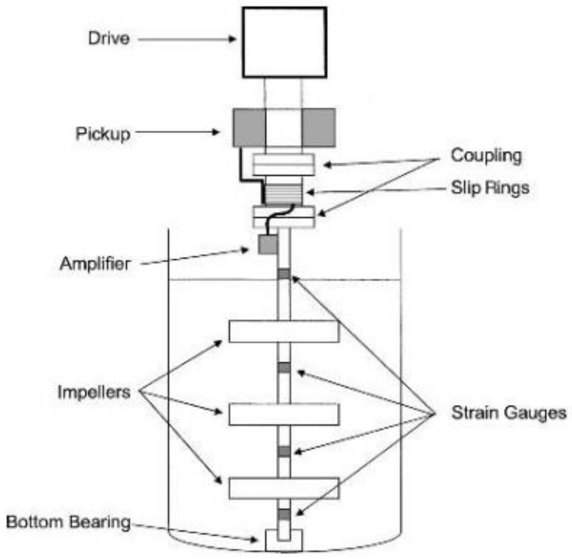


Figure 1: A example for the use of array of sensors within mixers. Strain gauges mounted on the mixer shaft are a popular and reliable method of measuring torque. source [1]

There are wide range of sensors are used to evaluate the mixer such as Platinum electrode conductivity probe or optical probs for measuring solid concentration, strain gauges for measure mass or torques, thermocouples use for temperature measurements and tachometers uses to measure the actual shaft speed. For control and read each type of sensors, needs to adepts their own drivers and required to convert as suitable to the controller.

Expect of the static mixers, all mixers are uses motors to create a motion. There are various types of motors (actuators) used, Electric motors, air motors and Hydraulic Motors. Although theoretically, all the speeds are possible with in motors, practically gain some of difficulties to operate motors with all speeds. Usually, mixer speeds can vary from 3600 rpm (High-Shear Mixers) to 30 rpm. Mixing speed is a critical factor to be ensure the quality of mixer and the time scale of mixer. Also, sometimes users are required to measure the consume power of mixer. Below text is summarized common attributes used in most mixers as well as most industries.

Common attributes used in most of mixers as well as most of industries:

1. Most industries do their mixing operations with sequential step by step process. Each step is defined with its own attributes such as mixing speeds, mixing time, required temperature scales, pressures requirements, and so on.

2. Most industries and mixers are designed with a combination of sub mixing units, that includes dedicated motors (or actuators), Dedicated array of sensors and input vassals. Some industry mixing operations are required to circulation also such as in pharmaceuticals, Fermentation and Cell Culture Industries.

3. Mean value and variance of some measurement is also used in most of theories as well as directly evaluate the mixing process.

4. Speed controlling, in and out vassal controlling, pressure controlling, PH controlling, and temperature controlling can be seen in many industries.

5. Measure the array of sensor values for concentration, temperature, speeds, pressure, pH rates and weights are common in many industries.

6. Also required to additional procedures for the stop all operations in safely when occurs an emergence. Cleaning of mixers are also required routing operation in many industries.

Addition to these common attributes, some industries required specialized operation to done the mixing. Below text summarizes the few special requirements for selected industries.

Mixing in the Fine Chemicals and Pharmaceutical Industries: Required to control overmixing and particle crystallization. Also, needs to control the temperature difference within the mixer and ensure that a uniform distribution of heat. For the crystallizations, gas-liquid reactions and some mixing operations are required to control the angle of impellers also. Figure 2 shows the common impellers angles.

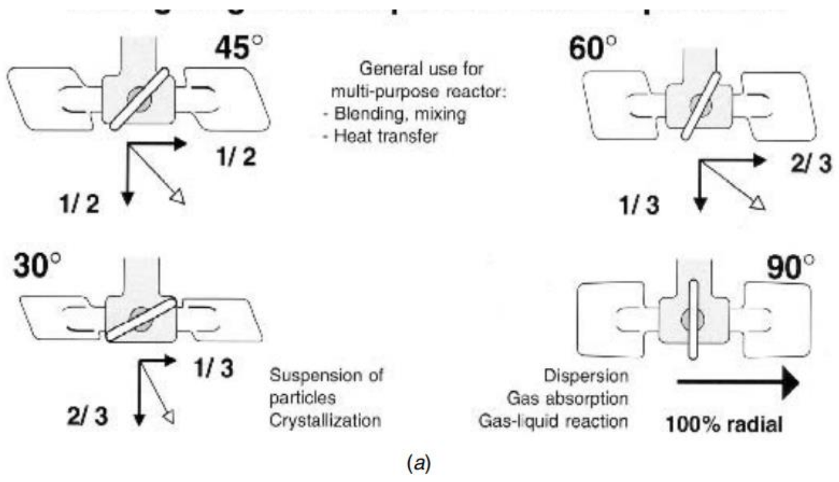


Figure 2: Common impellers angles

Fermentation and Cell Culture Industries: Cell culturing industry required much more controlled environment while the mixing when compared to other industries such as food. The maximum productivity, product concentration, and quality achievable depends primarily on bulk mixing and oxygen mass transfer, which in turn are governed by process operation, impeller type, and fluid properties. The viscosity of the broth will influence the bulk mixing, air dispersion, and power draw by the agitator. Therefore, proper torque measurement, speed and viscosity measurements are required. The dissolved oxygen can be fluctuated with a fixed frequency in a square- or sine-wave fashion by either varying the inlet gas composition or the fermenter head pressure to alter the liquid-phase dissolved oxygen concentration. Therefore, proper gas inlet or pressure controlling mechanism is required and control should be able to be controlling the gas by predefined ways.

Fluid Mixing Technology in the Petroleum Industry: Mixing applications in petroleum industry may be somewhat limited compared to chemical, pharmaceutical, and food manufacturing. In addition, refinery streams are less complex than specialty and fine chemicals in terms of fluid physical properties and process conditions. However, due to large volumes of petroleum streams to be mixed, mixing technology plays an important role in enhancing productivity and profitability. The refining processes involving mixing operations include making emulsion products for oil drilling, absorption of CO_2 from natural gas, crude oil–water homogenization for custody transfer, sludge suspension in crude oil storage tanks, desalting of crude oil, alkylation, caustic–oil contacting for neutralization, pH control, and more.

Mixing in the Pulp and Paper Industry: The pulp and paper industry comprises companies that use wood as raw material and produce pulp, paper, paperboard, and other cellulose-based products[2]. The most common chemical pulping process is the kraft process. Here, wood chips are treated to remove the lignin that binds the cellulose fiber to the wood matrix.

Highly Viscous Fluids, Polymers, and Pastes: Many industrially important products, such as pastes, putties, chewing gum, soap, grease, solid propellant, and some foods, fall into this category. Viscous mixing involves many applications in processes wherein the viscosity is sufficiently high (greater than 10 Pa s-1). Mixing highly viscous fluids required to draw much attention to the heat transfer, speeds and power built-up. Slow impeller speeds to limit heat buildup. These mixing industries required negative speeds also because of stop the fluid speeds. Figure 3 illustrates the requirement of negative speeds and the requirement of controlling the power and speeds of impellers in high-viscous fluids mixing.

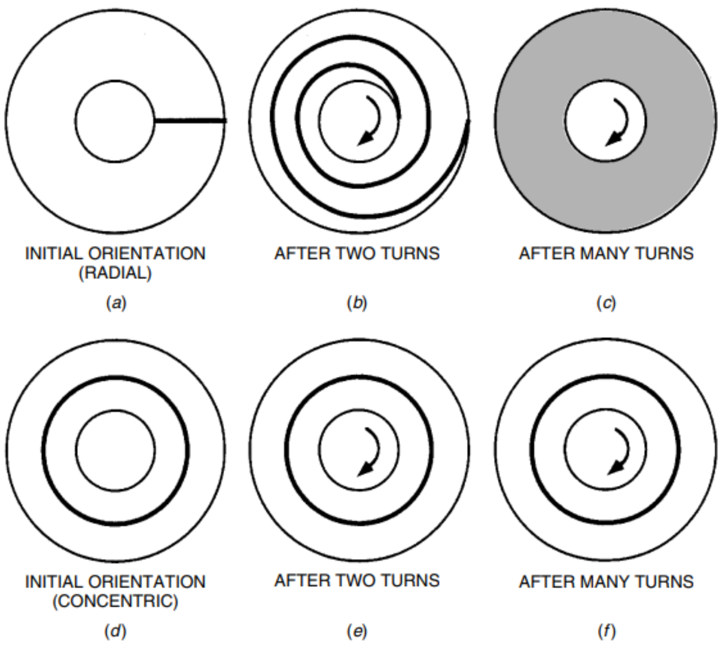


Figure 16-6 Effect of orientation on mixing in a concentric cylinder mixer.

Figure 3: Requirement of negative speeds and the requirement of controlling the power and speeds of impellers in high-viscous fluids mixing.

1.1 Requirement of CMP

1. It may be able to design an algorithm to make mixing operations step by step.
2. Each step or profile could be able to design or edit separately and independently.
3. Could be able to make motors and sensor groups as can control as once. This makes easier the process engineer job.
4. After doing the initial configurations, process engineers can control the mixing operations as steps. Also, could be able to evaluate the mixing process by using sensor values.
5. For the Diagnostic proposal, each operation, input and output could be controlled separately.
6. When wanted, additional functionalities that are required to various industries could be able to design using basic logic functions.

1.2 Internal Functionality of the Processor

Available Resources in the processor:

#	Type of peripherals	Previous count	Changed Count
1	motor	32	31
2	Vessels	32	31
3	temp. sensors	32	31
4	pressure sensors	32	31
5	concentration sensors	32	31
6	general proposed actuators	64	31
7	general proposed sensors	32	31
8	Profiles	16	16

1.3 How the CMP can be used in a mixing machine

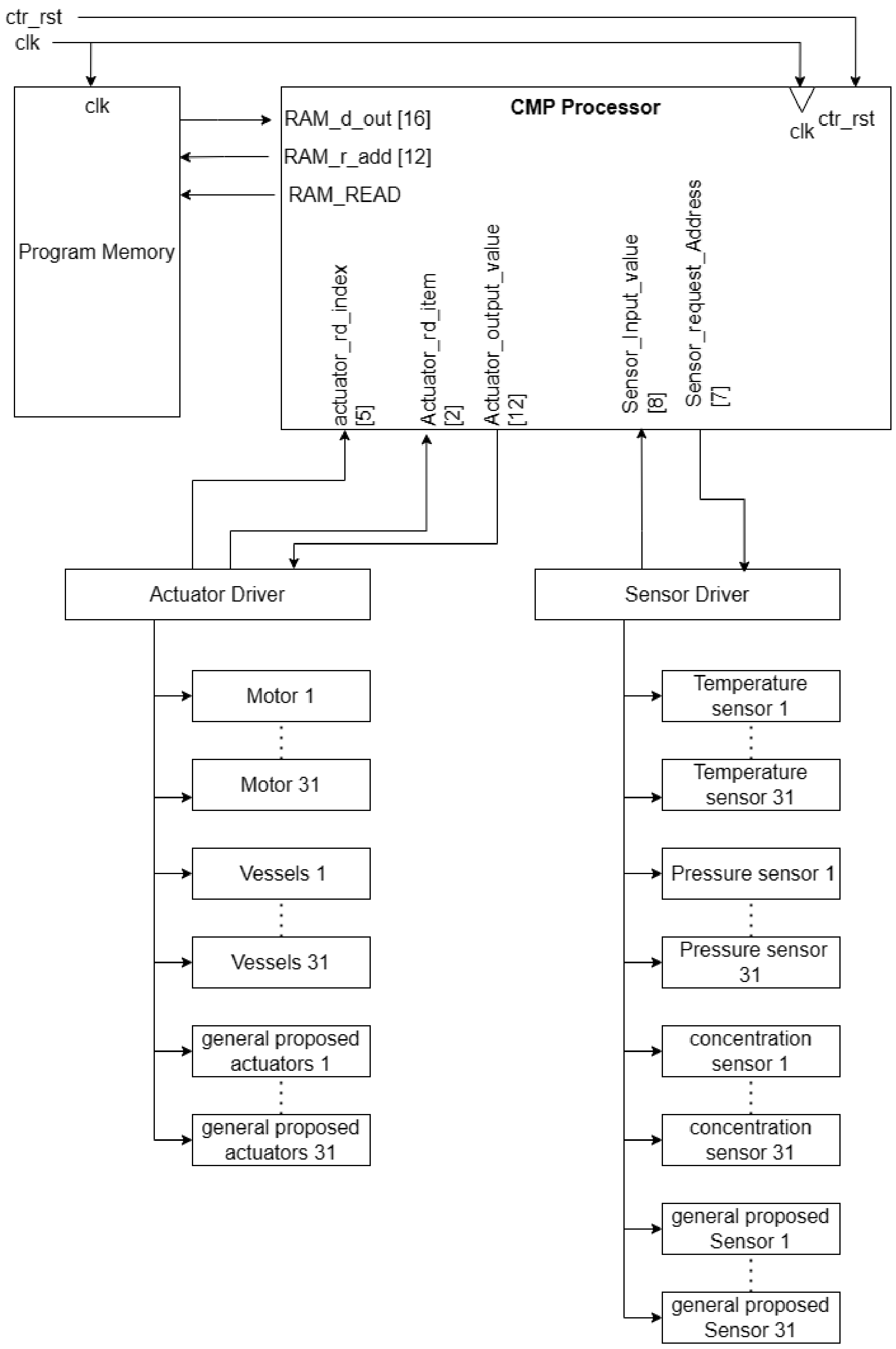
Since Mixing machine has many Actuators to control and many sensors to get inputs, CMP can't control them directly. Also, Actuators and sensors use different protocols and different logic families, needs to convert the CMP's IO pins, as compatible with each actuator and sensors. Drivers are responsible for above both functions. The process of sensor values reading and actuator value providing as follows.

Sensors Value Read: CMP sends the required sensor address to get the sensor output value. Driver will locate the sensor and provide appropriate value to CMP.

Actuator Value Set: Actuator values are needs to be continuously update. When Actuator driver continuously send the required actuator item and the index, CMP provides the respective value.

"Ctr_rst" is uses to reset the CMP and it. When CMP powerd up, It necessarily to be reset. Program memory (RAM) stores the instructions, and operands (Program). CMP fetch the data and execute them by getting program memory data.

This processor used RISC (Reduced Instruction Set Architecture).



1.4 ISA of the CMP

Instruction Word content:

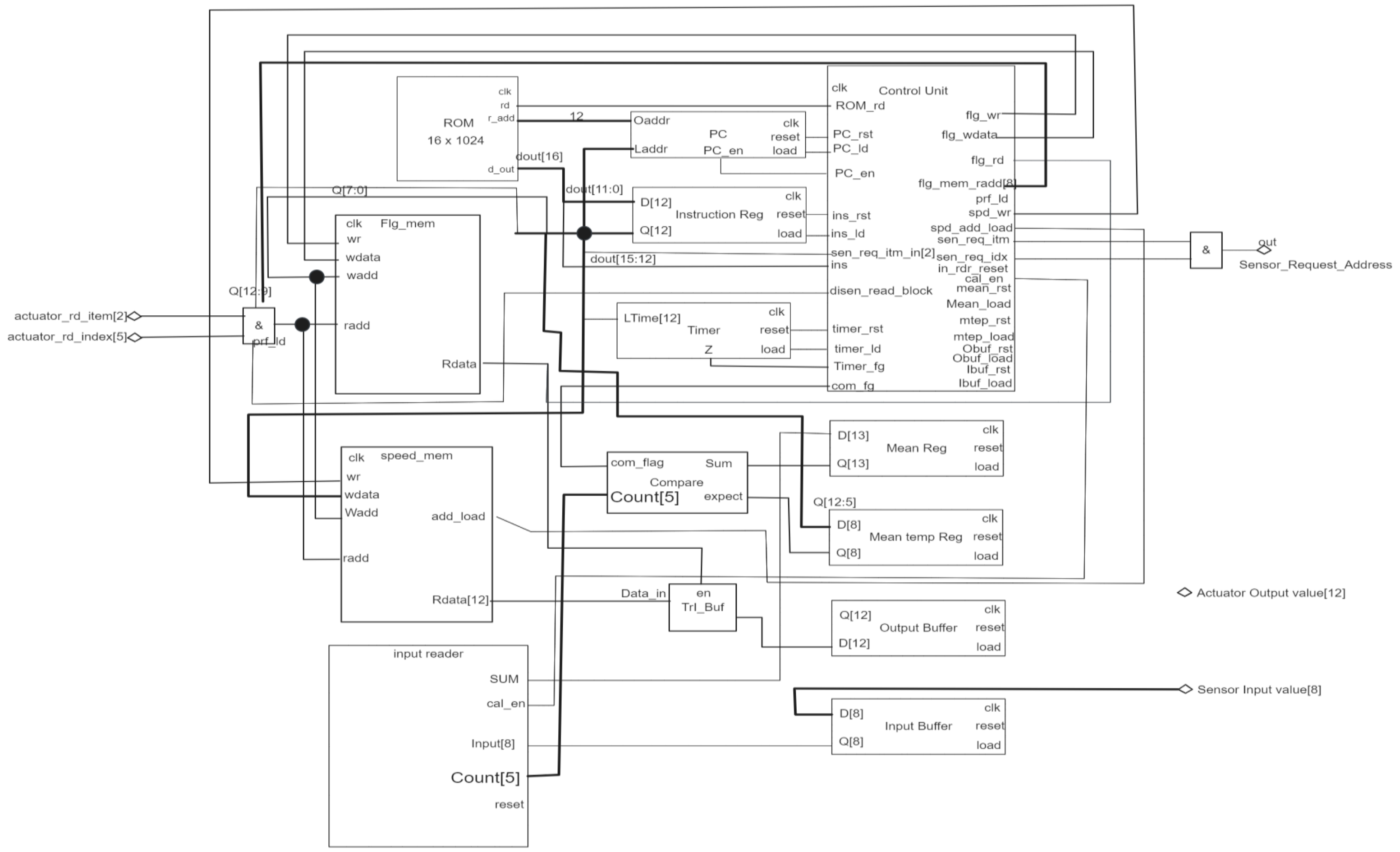
4 bit	12 bit
Opcode	Operands

There are 9 Instructions including reset instruction.

Opcode	**Mnemonic**	**Operands**	**Bytes**	**Operation**
0XXX	Reset	-	2	Reset the processor to initial state.
		Configuration Instructions		
1000	Add2p	P,I,IX	2	Add Items to a Profile ➔ P – Profile Number [4] ➔ I – Item [3] • Outputs: ○ 0 – Motor ○ 1 – Vassals ○ 2 – General Output ○ 3 - None • Input ○ 4-Temperature sensor ○ 5-Pressure sensor ○ 6-Concentration sensor ○ 7-General Input ➔ IX – Index [5] (0-31) ○ 0 Dedicated to indicates all enable peripheral.
1001	Point	P,I,IX	2	Point to the desired Item before set the values. ➔ P – Profile Number ➔ I – Item ➔ IX – Index *Operands same as Add2P
1010	Setval	V	2	Set the pointed value as given 12-bit value V [12] (0-4095)

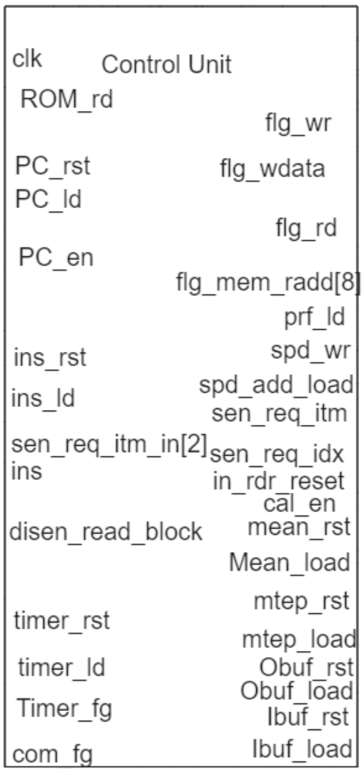
Control Instructions				
1011	control	P	2	On Profile Process ➔ P – Profile Number [4]
1100	Findmean	P,I	2	Finding the Mean of given Profile Item List. ➔ P – Profile [4] ➔ I – Item [2] ○ 0-Temperature sensor ○ 1-Pressure sensor ○ 2-concntration sensor ○ 3-General Inputs Store the results in mean register, a 8 Bit register.
1101	compare	V	2	Compare the values with Mean register value, and set the compare flag according to following conditions, $Compare\ Flag \begin{cases} 0\ ; V - Mean\ reg > 0 \\ 1\ ; V - Mean\ reg =< 0 \end{cases}$ V[8]
1110	JIC	addr	2	Jump if compare flag is 1 Addr[12]
1111	waitt	clk	2	Wait given Number of clocks. Clk[12]

1.5 Block diagram for the CMP



2 CONTROL UNIT OF THE PROCESSOR

Control Unit Controls the all activities of the processor and it maintain the fetch, decode and execution cycle. It control the all buffers, registers and control the input and output process. Program counter controlling, memory controlling, and control each functions using inbuilt FSM. It maintains the timing and synchronization of each blocks.



Below table illustrates the states changes and the signal controlling in control unit. \

Initial State	**Idel State**	**Fetch One: Send address**

PC_rst <='1';
ins_rst <='1';
timer_rst <='1';
in_rdr_reset <='1';
mtep_rst <='1';
Obuf_rst <='1';
Ibuf_rst <='1';
mean_rst <='1';
disen_read_block <= '0';
	PC_rst <='0';
	ins_rst <='0';
	timer_rst <='0';
	in_rdr_reset <='0';
	mtep_rst <='0';
	Obuf_rst <='0';
	Ibuf_rst <='0';
	mean_rst <='0';
		ROM_rd <= '1';
		ins_ld <= '1';
		Mean_load <='0';
		mtep_load <='0';
		PC_ld <= '0';
		PC_en <='0';

Fetch Two: Load Data	**Decode**	

--None	ROM_rd <='0';
	ins_ld<='0';
	ins_reg <= ins;
	
	if(ins_reg(0)='0') then Next_state<=Initial;	

	elsif(ins_reg = "1000") then Next_state <= Add2p;
	elsif(ins_reg = "1001") then Next_state <= Point;
	elsif(ins_reg= "1010") then Next_state <= Setval;
	elsif(ins_reg="1011") then Next_state<= control;
	elsif(ins_reg = "1100") then Next_state<=Findmean;
	elsif(ins_reg = "1101") then Next_state <= compare;
	elsif(ins_reg = "1110") then Next_state <= JIC;
	elsif(ins_reg = "1111") then Next_state <= waitt;
	else Next_state <= Initial;	

Add2p	Add2p_two	Add2p_three

--address is already in instruction reg Q pin.
disen_read_block <='1';
flg_wdata <= '1';
flg_wr <= '1';
next_state <= Add2p_two
	C_en <= '1'; --Increment Program counter
	next_state<=Add2p_three;
		C_en <= '0';
		flg_wdata <= '0';
		flg_wr <= '0';
		disen_read_block <='0';
		next_state <= Fetch_one;

Point	Point_two	Setval

pd_add_load <= '1';
next_state<=Point_two;
PC_en <= '1';
	spd_add_load <= '0';
	Pc_en <= '0';
	next_state<=Fetch_one;
		spd_wr <= '1';
		next_state<=Setval_two;
		PC_en <= '1';

Setval_two	Setval_three	Control

C_en <='0';
next_state<=Setval_three;
--save data
	d_wr <= '0';
	next_state <= Fetch_one;
		disen_read_block <= '1';
		PC_en <= '1';
		next_state<=Control_two;

Control_two	Findmean	Findmean_2

PC_en <= '0';
disen_read_block<= '0';
next_state<=Fetch_one;
	itm <= sen_req_itm_in;
	disen_read_block <= '1';
	flag_wr<= '0';
	
	idx <= "00000";
	
	PC_en<='1'
	Next_state<=Findmean2
		cal_en <= '0'
		PC_en<='0'
		if(idx = "11111") then
		disen_read_block <= '0';
		next_state <= Fetch_one;
		Mean_load <='1';
		else
		flg_mem_radd <= '1' & itm(5:4) & idx;
		next_state<=findmean22;

Findmean22	Findmean3	Findmean4

next_state <= Findmean3;
-- wait till flg_mem_out
	if(flg_rd = '1') then
	sen_req_itm <= itm(5:4);
	sen_req_idx <= idx;
	next_state <= find_mean4;
	elsif(flg_rd = '0') then
	idx <= idx+1;
	next_state <= Findmean2;
		next_state<= find_mean5;
		--wait untill sensor men out
		next_state <= Findmean5

Findmean5	Findmean6	Findmean7

Ibuf_load <= '1';
next_state <= Findmean6;
	next_state <= Findmean7;	next_state <= Findmean2;
		cal_en <= '1'
		idx <= idx+1;

Compare	JIC	Wait

mtep_load <='1';
next_state <= Fetch_one;
	if(com_fg = '1') then
	PC_ld <= '1';
	next_state <= Fetch_one;
	else
	next_state <= Fetch_one;
		timer_ld <='1';
		next_state <= waitt2

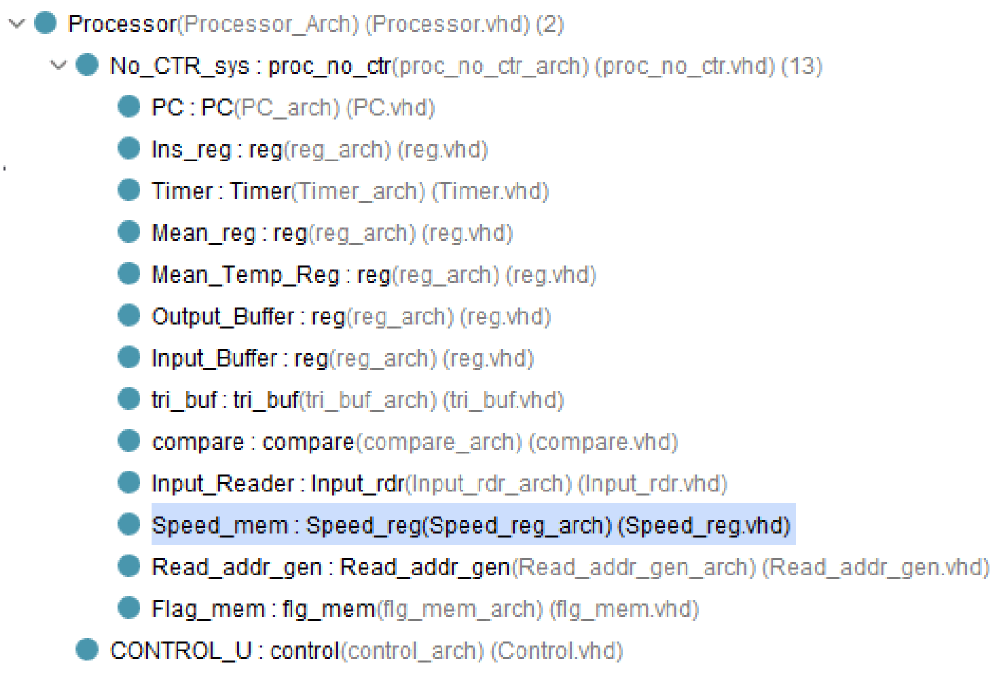
Waitt2		

timer_ld <='0';
if(timer_fg = 1) then
state_next <= Fetch_one;		

3 DESIGN CYCLE

1. Analyzed the problem, identify the requirement of CMP.
2. Designed working procedure of the CMP.
3. Design an ISA for the CMP.
4. Designed the block diagram for the processor indicating all input and output signals.
5. Create VHDL code for each entities.
6. Verify the model using simulations.
7. Error correcting, debugging.
8. Design control unit and model it using VHDL.
9. Synthesize the design and debug the design by observing simulations.
10. Implementation on FPGA board.

4 COMPILED VERSION OF VHDL CODING



```
Processor(Processor_Arch) (Processor.vhd) (2)
    No_CTR_sys : proc_no_ctr(proc_no_ctr_arch) (proc_no_ctr.vhd) (13)
        PC : PC(PC_arch) (PC.vhd)
        Ins_reg : reg(reg_arch) (reg.vhd)
        Timer : Timer(Timer_arch) (Timer.vhd)
        Mean_reg : reg(reg_arch) (reg.vhd)
        Mean_Temp_Reg : reg(reg_arch) (reg.vhd)
        Output_Buffer : reg(reg_arch) (reg.vhd)
        Input_Buffer : reg(reg_arch) (reg.vhd)
        tri_buf : tri_buf(tri_buf_arch) (tri_buf.vhd)
        compare : compare(compare_arch) (compare.vhd)
        Input_Reader : Input_rdr(Input_rdr_arch) (Input_rdr.vhd)
        Speed_mem : Speed_reg(Speed_reg_arch) (Speed_reg.vhd)
        Read_addr_gen : Read_addr_gen(Read_addr_gen_arch) (Read_addr_gen.vhd)
        Flag_mem : flg_mem(flg_mem_arch) (flg_mem.vhd)
    CONTROL_U : control(control_arch) (Control.vhd)
```

Program Counter

```vhdl
library ieee;
use ieee.std_logic_1164.all;
--use ieee.numeric_std.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PC is
generic(
width: integer:=12);
port(
clk,reset,load : in std_logic;
Laddr: in std_logic_vector(width-1 downto 0);
Oaddr: out std_logic_vector(width-1 downto 0);
PC_en: in std_logic);
end  entity PC;

architecture PC_arch of PC is

signal Oaddr_TEMP: std_logic_vector(width-1 downto 0);

begin
count_proc: process(clk,reset,load) begin
if(reset = '1') then Oaddr_TEMP<=(others=>'0');
elsif(rising_edge(clk)) then
    if(load='1') then Oaddr_TEMP <= Laddr;
    elsif(PC_en='1') then Oaddr_TEMP <= Oaddr_TEMP +1;
end if;
end if;
end process;

Oaddr <= Oaddr_TEMP;
end architecture;
```

Register

```
library ieee;
use ieee.std_logic_1164.all;


entity reg is
generic( width: integer:=12);
port(
    clk,reset,load : in std_logic;
    D : in std_logic_vector(width-1 downto 0);
    Q : out std_logic_vector(width-1 downto 0));
end entity reg;

architecture reg_arch of reg is begin
reg_proc: process(clk,load,reset) begin
    if(reset='1') then Q <= (others => '0');
    elsif(rising_edge(clk)) then
        if(load='1') then Q <= D;
       end if; end if;
       end process reg_proc;
end architecture reg_arch;
```

Timer

```vhdl
library ieee;
use ieee.std_logic_1164.all;
--use ieee.numeric_std.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Timer is
generic(
width: integer:=12);
port(
clk,reset,load : in std_logic;
Ltime: in std_logic_vector(width-1 downto 0);
Z : out std_logic);

end  entity Timer;

architecture Timer_arch of Timer is

signal T: std_logic_vector(width-1 downto 0);

begin

Timer_proc: process(clk,reset,load,T) begin
if(reset='1') then T <= (others=>'0');
elsif (rising_edge(clk)) then
    if(load = '1') then T <= LTime;
    elsif(T /= 0) then T <= T-1;
end if;end if;
end process;

Flag_proc: Process(T) begin
if (T = 0) then Z <= '1';
else Z <= '0';
end if;
end process;

end architecture Timer_arch;
```

Tri-State Buffer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tri_buf is
 Port (

 data_in: in std_logic_vector(11 downto 0);
 en: in std_logic;
 tri_out: out std_logic_vector(11 downto 0)
 );
end tri_buf;

architecture tri_buf_arch of tri_buf is

begin
tri_proc: process(en) is begin
if(en='1') then  tri_out<=data_in;
else tri_out<=(others=>'Z');
end if; end process;
end tri_buf_arch;
```

Compare Unit

```
library ieee;
use ieee.std_logic_1164.all;
--use ieee.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity compare is
    port (

Expect: in std_logic_vector(7 downto 0);
Sum: in std_logic_vector(12 downto 0);
count: in std_logic_vector(4 downto 0);
com_flag: out std_logic
    );
end entity compare;

architecture compare_arch of compare is
signal T1,T2,T3,T4,T5,res : std_logic_vector(12 downto 0); -- temp value for
multiplication

begin

mult_proc: process(Expect,count) is begin
    if(count(0)='1') then T1<= ("00000" & Expect ); else T1<=(others=>'0'); end
if;
    if(count(1)='1') then T2<= ("0000" & Expect & "0"); else T2<=(others=>'0');
end if;
    if(count(2)='1') then T3<= ("000" & Expect & "00"); else T3<=(others=>'0');
end if;
    if(count(3)='1') then T4<= ("00" & Expect & "000"); else T4<=(others=>'0');
end if;
    if(count(4)='1') then T5<= ("0" & Expect & "0000"); else T5<=(others=>'0');
end if;
end process;

res <= T1+T2+T3+T4+T5;

Compare_proc: process(res,sum) is begin
    if(res<sum) then com_flag<='0';
    else com_flag<='1';
    end if; end process;

end architecture compare_arch;
```

Input Reader Unit

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Input_rdr is
port(

    Input: in std_logic_vector(7 downto 0);
    cal_en,reset: in std_logic;
    count: inout std_logic_vector(4 downto 0);
    sum: inout std_logic_vector(12 downto 0)
);
end entity;

architecture Input_rdr_arch of Input_rdr is

begin

read_proc: process(cal_en,reset) begin

if(reset = '1') then
    sum<=(others=>'0');
    count<=(others=>'0');

elsif(cal_en='1') then
    sum <= sum+input;
    count <= count +1;

end if;
end process;


end architecture Input_rdr_arch;
```

Speed Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use STD.textio.all;
use IEEE.std_logic_textio.all;
use ieee.numeric_std.ALL;

entity Speed_reg is
generic(
    Dwidth: integer:=12;
    Awidth: integer:=11);
port(
    clk,wr : in std_logic;
    wdata : in std_logic_vector(Dwidth-1 downto 0);
    Wadd_T,RAdd : in std_logic_vector(Awidth-1 downto 0);
    rdata: out std_logic_vector(Dwidth-1 downto 0);
    spd_add_load: in std_logic);

end entity;

architecture Speed_reg_arch of Speed_reg is
type array_type is array(0 to 2**Awidth-1) of std_logic_vector(Dwidth-1 downto
0);

signal array_reg: array_type := (others=>(others=>'0'));
signal Wadd : std_logic_vector(Awidth-1 downto 0);
begin

Addr_buf_proc: process(spd_add_load) begin
    if(spd_add_load = '1') then Wadd <= Wadd_T; end if;
    end process;


speed_proc: process(clk,wr,wdata,wadd,radd) begin
if(rising_edge(clk)) then
    if(wr='1') then
        array_reg(to_integer(unsigned(wadd))) <= wdata;
      end if;
     rdata <= array_reg(to_integer(unsigned(radd)));
end if;
end process;
end architecture;
```

Read address Generator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Read_addr_gen is
 Port (
 Item: in std_logic_vector(1 downto 0);
 Idx: in std_logic_vector(4 downto 0);
 add_out: out std_logic_vector(11 downto 0);
 profile: in std_logic_vector(3 downto 0);
 disable: in std_logic;
 flg_mem_radd: in std_logic_vector(7 downto 0)

  );
end Read_addr_gen;

architecture Read_addr_gen_arch of Read_addr_gen is
signal profile_T: std_logic_vector(3 downto 0);
type state_type is (state1,state2);

begin
--Block_proc: process(disable) is begin


--if(disable = '0') then add_out <= profile_T & '0' & Item & Idx;
--elsif(disable = '1') then add_out <= profile & flg_mem_radd; --
Profile_T<=profile;
--end if;
--end process;


--UPDATE_proc: process(disable) is begin
--if(disable = '1') then Profile_T<=profile;
--end if;
--end process;
--end Read_addr_gen_arch;
add_out <=  (profile_T & '0' & Item & Idx) when (disable = '0') else
            (profile & flg_mem_radd) when (disable = '1');

profile_T <= profile when (disable = '1');
end Read_addr_gen_arch;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use STD.textio.all;
use IEEE.std_logic_textio.all;
use ieee.numeric_std.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity flg_mem is
generic(
    Dwidth: integer:=16;
    Awidth: integer:=12);
port(
    clk,wr : in std_logic;
    wdata : in std_logic;
    Wadd, radd : in std_logic_vector(Awidth-1 downto 0);
    rdata: out std_logic);
--    profile: in std_logic_vector(3 downto 0));

end entity;

architecture flg_mem_arch of flg_mem is
type array_type is array(0 to 2**Awidth-1) of std_logic_vector(Dwidth-1 downto 0);

signal array_reg: array_type := (others => (others => '0'));
signal profile_w,profile_r:std_logic_vector(3 downto 0);


begin


write_proc: process(clk,wr,wdata,wadd,radd) begin
if(rising_edge(clk)) then
    if(wr='1') then
        array_reg(to_integer(unsigned(wadd(7 downto 0))))(to_integer(unsigned(profile_w))) <=
wdata; --bit wise acc
    else
         if(radd(7 downto 0)< 32) then
         rdata <= array_reg(to_integer(unsigned(radd(7 downto
0))))(to_integer(unsigned(profile_r))) or array_reg(0)(to_integer(unsigned(profile_r)));

         elsif(radd(7 downto 0) < 64) then
         rdata <= array_reg(to_integer(unsigned(radd(7 downto
0))))(to_integer(unsigned(profile_r))) or array_reg(32)(to_integer(unsigned(profile_r)));

         elsif(radd(7 downto 0) < 96) then
         rdata <= array_reg(to_integer(unsigned(radd(7 downto
0))))(to_integer(unsigned(profile_r))) or array_reg(64)(to_integer(unsigned(profile_r)));

         elsif(radd(7 downto 0) < 128) then
         rdata <= array_reg(to_integer(unsigned(radd(7 downto
0))))(to_integer(unsigned(profile_r))) or array_reg(96)(to_integer(unsigned(profile_r)));

         elsif(radd(7 downto 0) < 160) then
         rdata <= array_reg(to_integer(unsigned(radd(7 downto
0))))(to_integer(unsigned(profile_r))) or array_reg(128)(to_integer(unsigned(profile_r)));

         elsif(radd(7 downto 0) < 192) then
         rdata <= array_reg(to_integer(unsigned(radd(7 downto
0))))(to_integer(unsigned(profile_r))) or array_reg(160)(to_integer(unsigned(profile_r)));

         elsif(radd(7 downto 0) < 224) then
         rdata <= array_reg(to_integer(unsigned(radd(7 downto
0))))(to_integer(unsigned(profile_r))) or array_reg(192)(to_integer(unsigned(profile_r)));

         elsif(radd(7 downto 0) < 256) then
         rdata <= array_reg(to_integer(unsigned(radd(7 downto
0))))(to_integer(unsigned(profile_r))) or array_reg(224)(to_integer(unsigned(profile_r)));


         else rdata <='Z'; end if;
        end if;
end if;
end process;


profile_w <= Wadd(11 downto 8);
profile_r <= radd(11 downto 8);

end architecture;
```

Control Unit

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control is

port(
clk, ctr_rst : in std_logic;

--Reset Pins
PC_rst,ins_rst,timer_rst, in_rdr_reset,mtep_rst,Obuf_rst: out std_logic;
Ibuf_rst,mean_rst: out std_logic;
spd_add_load: out std_logic;
profile_ld: out std_logic;

--Other control signal pin
ROM_rd,PC_en,PC_ld,disen_read_block,timer_ld,flg_wr : out std_logic;
flg_wdata,spd_wr,cal_en,Mean_load,Obuf_load: out std_logic;
Ibuf_load,mtep_load,ins_ld: out std_logic;

flg_mem_radd : out std_logic_vector(7 downto 0);
sen_req_itm : out std_logic_vector(1 downto 0);
sen_req_idx : out std_logic_vector(4 downto 0);

--External Flags
Timer_fg, com_fg,flg_rd : in std_logic;

--Instrction
ins : in std_logic_vector(3 downto 0);
sen_req_itm_in: in std_logic_vector(5 downto 0)


);
end entity control;


architecture control_arch of control is

type state_type is (Initial,Ideal,Fetch_one,Fetch_two,Decode,Add2p,Point,Setval,
control,Findmean ,compare,JIC,waitt,waitt3,
waitt2,findmean7,Findmean6,findmean5,findmean4,findmean3,findmean22,findmean2,control
_two,setval_two,Setval_three,point_two,add2p_three,Add2p_two);

signal Current_state,Next_state : state_type;
--signal ins_reg : std_logic_vector(3 downto 0);
signal itm: std_logic_vector(1 downto 0);
signal idx: std_logic_vector(4 downto 0);
begin

--ins_reg <= ins;


clk_proc: process(clk,ctr_rst) begin

if(ctr_rst = '1') then
```

```
    Current_state <= Initial;
elsif(rising_edge(clk)) then
    Current_state <= Next_state;
    end if;
end process;


Control_proc: process(Current_state,ins,Timer_fg, com_fg )
begin
case (Current_state) is

    When Initial =>
        PC_rst <='1';
        ins_rst <='1';
        timer_rst <='1';
        in_rdr_reset <='1';
        mtep_rst <='1';
        Obuf_rst <='1';
        Ibuf_rst <='1';
        mean_rst <='1';
        spd_add_load <='0';
        Next_state <= Ideal;

    When Ideal =>
        PC_rst <='0';
        ins_rst <='0';
        timer_rst <='0';
        in_rdr_reset <='0';
        mtep_rst <='0';
        Obuf_rst <='0';
        Ibuf_rst <='0';
        mean_rst <='0';

        Next_state <= Fetch_one;

    When Fetch_one =>
        ROM_rd <= '1';
        ins_ld <= '1';
        Mean_load <= '0';
        mtep_load <= '0';
        PC_ld <= '0';
        PC_en <= '0';

        Next_state <= Fetch_two;

    When Fetch_two =>
        Next_state <= Decode;
    When Decode =>
        ROM_rd <='0';
        ins_ld<='0';
--      ins_reg <= ins;

    --  if(ins_reg(3)='0') then Next_state<=Initial;

        if(ins = "1000") then Next_state <= Add2p;
        elsif(ins = "1001") then Next_state <= Point;
        elsif(ins= "1010") then Next_state <= Setval;
        elsif(ins="1011") then Next_state<= control;
        elsif(ins = "1100") then Next_state<=Findmean;
        elsif(ins = "1101") then Next_state <= compare;
        elsif(ins = "1110") then Next_state <= JIC;
        elsif(ins = "1111") then Next_state <= waitt;
        else Next_state <= Initial;
        end if;

    ----+++++++++++++Instructions States++++++++++---------------------
```

```
-------Add2p---------------------->>>>>>>>>>>>>>>>>

When Add2p =>
    disen_read_block <= '1';
    flg_wdata <= '1';
    flg_wr <= '1';
    Next_state <= Add2p_two;

When Add2p_two =>
    PC_en <= '1';
    Next_state <= Add2p_three;

when Add2p_three =>
    PC_en <= '0';
    flg_wdata <= '0';
    flg_wr <= '0';
    disen_read_block <= '0';
    Next_state <= Fetch_one;

--------------Point and setval------------->>>>>>>>>>>>>>>>>>

when point =>
    spd_add_load <= '1';
    Next_state <= point_two;
    PC_en <= '1';
    Next_state <= point_two;

when point_two =>
    spd_add_load <= '0';
    PC_en <= '0';
    Next_state <= Fetch_one;

when Setval =>
    spd_wr <= '1';
    Next_state <= Setval_two;
    PC_en <= '1';

when Setval_two =>
    PC_en <= '0';
    Next_state <= Setval_three;

when Setval_three =>
    spd_wr <= '0';
    Next_state <= Fetch_one;

--------------Control------->>>>>>>>>>>>>>>>

when control =>
    disen_read_block <= '1';
    PC_en <= '1';
    Next_state <= control_two;

when control_two=>
    PC_en <= '0';
    disen_read_block <= '0';
    Next_state <= Fetch_one;


-----------Find Mean ------------>>>>>>>>*

when Findmean =>
    itm <= sen_req_itm_in(1 downto 0);
    disen_read_block <= '1';
    flg_wr <= '0';

    idx <= "00000";
```

```
        PC_en <= '1';
        Next_state <= Findmean2;

    when Findmean2 =>
        cal_en<='0';
        PC_en <= '0';

        if(idx = "11111") then
            disen_read_block <= '0';
            Next_state <= Fetch_one;
            Mean_load <= '1';
        else
            flg_mem_radd <= '1' & itm & idx;
            Next_state <= findmean22;
        end if;

    When Findmean22 =>
        Next_state <= Findmean3;

    When Findmean3 =>

        if(flg_rd = '1') then
            sen_req_itm <= itm;
            sen_req_idx <= idx;
            Next_state <= Findmean4;
        else
            idx <= idx +1;
            Next_state <= Findmean2;
        end if;

    When Findmean4 =>
        Next_state <= findmean5;

    When Findmean5 =>
        Ibuf_load <= '1';
        Next_state <= Findmean6;

    When findmean6 =>
        Next_state <= Findmean7;

    when findmean7 =>
        Next_state <= Findmean2;
        cal_en <= '1';
        idx <= idx+1;

    ----------------Compare------------------

    When compare =>
        mtep_load <= '1';
        Next_state <= Fetch_one;
        PC_en <= '1';

    when JIC =>

        if(com_fg = '1') then
            PC_ld <= '1';
            Next_state <= Fetch_one;
        else
            PC_en <= '1';
            Next_state <= Fetch_one;
        end if;

    when Waitt =>
        timer_ld <= '1';
        Next_state <= waitt2;
    when Waitt2 =>
       Next_state <= waitt3;

    when waitt3 =>
        timer_ld <= '0';
        if(Timer_fg = '1') then
            Next_state <= Fetch_one;
            PC_en <= '1';
        end if;


    When others=>
        Next_state <= Initial;

    end case;

    end process control_proc;



end architecture Control_arch;
```

By including all components without control unit, created no_ctr.vhd model. Processor consists of both no_ctr and control unit.

------------- NO CONTROL UNIT--------------------

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity proc_no_ctr is

generic(
w_12 :integer:=12;
w_13 : integer:=13;
w_8  : integer:=8;
w_11 : integer:=11;
w_16 : integer:=16
);

 Port (
 clk: in std_logic;

 --Contrl unit signals------------------------------/////*****
 PC_rst,PC_ld, ins_rst,ins_ld,timer_rst,timer_ld,flg_wr,flg_wdata,PC_en : in
std_logic;
 spd_wr,in_rdr_reset,cal_en,mean_rst,mean_load,mtep_rst,disen_read_block : in
std_logic;
 mtep_load,obuf_rst,obuf_load,ibuf_rst,ibuf_load,rom_rd : in std_logic;
 spd_add_load: in std_logic;
 sen_req_itm_in:out std_logic_vector(5 downto 0);

 ins : out std_logic_vector(3 downto 0);
 timer_fg,com_fg,flg_rd : out std_logic;
 sen_req_itm: in std_logic_vector(1 downto 0);
 sen_req_idx: in std_logic_vector(4 downto 0);
 flg_mem_radd: in std_logic_vector(7 downto 0);

--Main memory in out ---------------------------////******
Rom_read: out std_logic;
Rom_r_add: out std_logic_vector(11 downto 0);
ROM_d_out: in std_logic_vector(15 downto 0);

--Inout pins --------------------------------//////******
actuator_rd_item: in std_logic_vector(1 downto 0);
actuator_rd_index: in std_logic_vector(4 downto 0);
Sensor_rqst_address: out std_logic_vector(6 downto 0);
Actuator_output_value: out std_logic_vector(11 downto 0);
Sensor_Input_value: in std_logic_vector(7 downto 0)
 );
end proc_no_ctr;

architecture proc_no_ctr_arch of proc_no_ctr is


signal ins_reg_Q,radd_s,output_D,spd_read_data: std_logic_vector(11 downto 0);
signal sum_D,Sum_Q: std_logic_vector(12 downto 0);
signal expect_s,count_input:std_logic_vector(7 downto 0);
signal Flg_RData: std_logic;
signal count: std_logic_vector(4 downto 0);
signal Wadd_signal,Radd_signal : std_logic_vector(10 downto 0);

begin

PC: entity work.PC(PC_arch) --checked
generic map(width => w_12)
port map(clk=>clk,PC_en=>PC_en,reset =>
PC_rst,load=>PC_ld,Oaddr=>ROM_r_add,Laddr=>INS_REG_Q);

Ins_reg: entity work.Reg(Reg_arch) --checked
generic map(width => w_12)
port map(clk=>clk,reset=>ins_rst,load=>ins_ld,D=>ROM_d_out(11 downto
0),Q=>ins_reg_Q);

Timer: entity work.Timer(Timer_arch) --checked
generic map(width=>w_12)
port map(clk=>clk,reset=>timer_rst,load=>Timer_ld,Ltime=>ins_reg_Q,Z=>timer_fg);

Mean_reg: entity work.Reg(Reg_arch)  --checked
generic map(width => w_13)
port map(clk=>clk,reset=>mean_rst,load=>mean_load,D=>sum_D,Q=>Sum_Q);

Mean_Temp_Reg: entity work.Reg(Reg_arch) --checked
generic map(width => w_8)
port map(clk=>clk,reset=>mtep_rst,load=>mtep_load,Q=>expect_s,D=>ins_reg_Q(11 downto
4));

Output_Buffer: entity work.Reg(Reg_arch) --checked
generic map(width => w_12)
port
map(clk=>clk,reset=>obuf_rst,load=>obuf_load,D=>output_D,Q=>Actuator_output_value);

Input_Buffer: entity work.Reg(Reg_arch)--checked
generic map(width => w_8)
port
map(clk=>clk,reset=>ibuf_rst,load=>ibuf_load,D=>Sensor_Input_value,Q=>count_input);

tri_buf: entity work.tri_buf(tri_buf_arch) --checked
port map(data_in=>spd_read_data, en=>Flg_RData, tri_out=>output_D);

compare: entity work.compare(compare_arch) --checked
port map(Expect=>expect_s, Sum=>Sum_Q, count=>count ,com_flag=>com_fg);

Input_Reader: entity work.Input_rdr(Input_rdr_arch) --checked
```

```
port map(Input=>count_input,
Cal_en=>cal_en,reset=>in_rdr_reset,count=>Count,sum=>Sum_D);

Speed_mem: entity work.Speed_reg(speed_reg_arch)--checked
Generic map (Dwidth=>W_12, Awidth=>w_11)

port map(clk=>clk,wr=>spd_wr,Wdata=>ins_reg_Q
,Wadd_T=>Wadd_signal,Radd=>Radd_signal,rdata=>spd_read_data,spd_add_load=>spd_add_loa
d);

Read_addr_gen: entity work.Read_addr_gen  --checked
port
map(Item=>actuator_rd_item,flg_mem_radd=>flg_mem_radd,Idx=>actuator_rd_index,add_out=
>radd_s,profile=>ins_reg_Q(11 downto 8),disable=>disen_read_block);


Flag_mem: entity work.flg_mem
generic map(Dwidth=>w_16,Awidth=>w_12)
port
map(clk=>clk,wr=>flg_wr,wdata=>flg_wdata,wadd=>ins_reg_Q,radd=>radd_s,rdata=>Flg_RDat
a);

Wadd_signal <= (ins_reg_Q(11 downto 8) & ins_reg_Q(6 downto 0));
Radd_signal <= (radd_s(11 downto 8) & radd_s(6 downto 0));
Rom_read <= rom_rd;
flg_rd <= Flg_RData;
Sensor_rqst_address <= sen_req_itm & sen_req_idx;
ins <= ROM_d_out(15 downto 12);
--radd_s <= ins_reg_Q(11 downto 8) & flg_mem_radd;
sen_req_itm_in <= ins_reg_Q(5 downto 0);

end proc_no_ctr_arch;
```

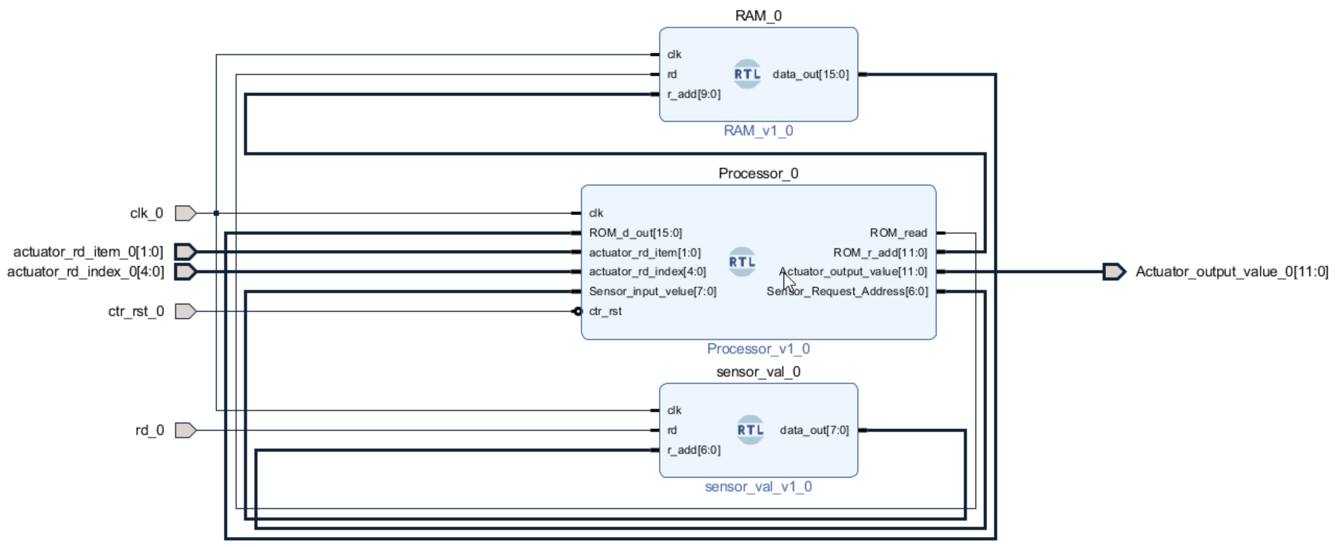
-------TOP MODULE – PROCESSROR--------

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Processor is
    generic(
        w_12 :integer:=12;
        w_13 : integer:=13;
        w_8  : integer:=8;
        w_11 : integer:=11;
        w_16 : integer:=16
    );

 Port (

 clk : in std_logic;

 --Memory Control pin
 ROM_read : out std_logic; --ok
 ROM_r_add : out std_logic_vector(11 downto 0); --ok
 ROM_d_out : in std_logic_vector(15 downto 0); --ok

 --IO pin
 actuator_rd_item : in std_logic_vector(1 downto 0); --ok
 actuator_rd_index : in std_logic_vector(4 downto 0); --ok
 Sensor_input_velue : in std_logic_vector(7 downto 0); --ok

 Actuator_output_value : out std_logic_vector(11 downto 0); --ok
 Sensor_Request_Address : out std_logic_vector(6 downto 0);  --ok

 --Reset control unit
 ctr_rst : in std_logic



 );
end Processor;

architecture Processor_Arch of Processor is
--Internal control paths

signal PC_rst,PC_ld, ins_rst,ins_ld,timer_rst,timer_ld,flg_wr,flg_wdata,PC_en :
std_logic;
signal spd_wr,in_rdr_reset,cal_en,mean_rst,mean_load,mtep_rst,disen_read_block :
std_logic;
signal mtep_load,obuf_rst,obuf_load,ibuf_rst,ibuf_load,rom_rd,spd_add_load :
std_logic;

signal ins : std_logic_vector(3 downto 0);
signal timer_fg,com_fg,flg_rd : std_logic;
signal sen_req_itm: std_logic_vector(1 downto 0);
signal sen_req_idx: std_logic_vector(4 downto 0);
signal flg_mem_radd: std_logic_vector(7 downto 0);
signal sen_req_itm_in: std_logic_vector(5 downto 0);


begin

No_CTR_sys: entity work.proc_no_ctr(proc_no_ctr_arch)
generic map(w_12=>w_12,w_13=>w_13,w_8=>w_8,w_11=>w_11,w_16=>w_16)
port map(
--Control unit signals
clk=>clk,PC_rst => PC_rst,PC_ld => PC_ld, ins_rst => ins_rst,ins_ld => ins_ld,
timer_rst => timer_rst,timer_ld =>
timer_ld,flg_wr=>flg_wr,flg_wdata=>flg_wdata,PC_en=>PC_en,
spd_wr=>spd_wr,in_rdr_reset=>in_rdr_reset,cal_en=>cal_en,mean_rst=>mean_rst,
mean_load=>mean_load,mtep_rst=>mtep_rst,disen_read_block=>disen_read_block,
mtep_load=>mtep_load,obuf_rst=>obuf_rst,obuf_load=>obuf_load,ibuf_rst=>ibuf_rst,
ibuf_load=>ibuf_load,rom_rd=>rom_rd,spd_add_load=>spd_add_load,sen_req_itm_in=>sen_re
q_itm_in,

ins=>ins,
timer_fg=>timer_fg,com_fg=>com_fg,flg_rd=>flg_rd,
sen_req_itm=>sen_req_itm,
sen_req_idx=>sen_req_idx,
flg_mem_radd => flg_mem_radd,

--Main Memory in out
Rom_read => ROM_read,
Rom_r_add => ROM_r_add,
ROM_d_out => ROM_d_out,

--Input Pins
actuator_rd_item => actuator_rd_item,
actuator_rd_index => actuator_rd_index,
Sensor_rqst_address => Sensor_Request_Address,
Actuator_output_value => Actuator_output_value,
Sensor_Input_value => Sensor_input_velue


);

CONTROL_U: entity work.control(control_arch)
port map(


--Reset Pins and clocks
clk=>clk, ctr_rst=>ctr_rst,PC_rst=>PC_rst,ins_rst=>ins_rst,timer_rst=>timer_rst,
in_rdr_reset=>in_rdr_reset,mtep_rst=>mtep_rst,Obuf_rst=>Obuf_rst,
Ibuf_rst=>Ibuf_rst,mean_rst=>mean_rst,spd_add_load=>spd_add_load,sen_req_itm_in=>sen_
req_itm_in,


--Other control signal pin
ROM_rd=>ROM_rd,PC_en=>PC_en,PC_ld=>PC_ld,disen_read_block=>disen_read_block,
timer_ld=>timer_ld,flg_wr=>flg_wr,
flg_wdata=>flg_wdata,flg_rd=>flg_rd,spd_wr=>spd_wr,cal_en=>cal_en,
Mean_load=>Mean_load,Obuf_load=>Obuf_load,
Ibuf_load=>Ibuf_load,mtep_load=>mtep_load,ins_ld=>ins_ld,

flg_mem_radd=>flg_mem_radd,
sen_req_itm=>sen_req_itm,
sen_req_idx=>sen_req_idx,

--External Flags
Timer_fg=>Timer_fg, com_fg=>com_fg,

ins=>ins


);


end Processor_Arch;
```

5 TEST BENCHES FOR THE PROCESSOR

5.1 Simulation of internal functions

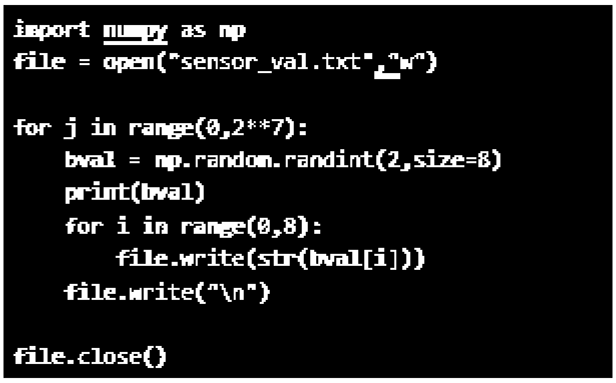
To simulate the processor functionality and make Timing diagrams, create a Test setup as follows.



Here, RAM initialized with this program:

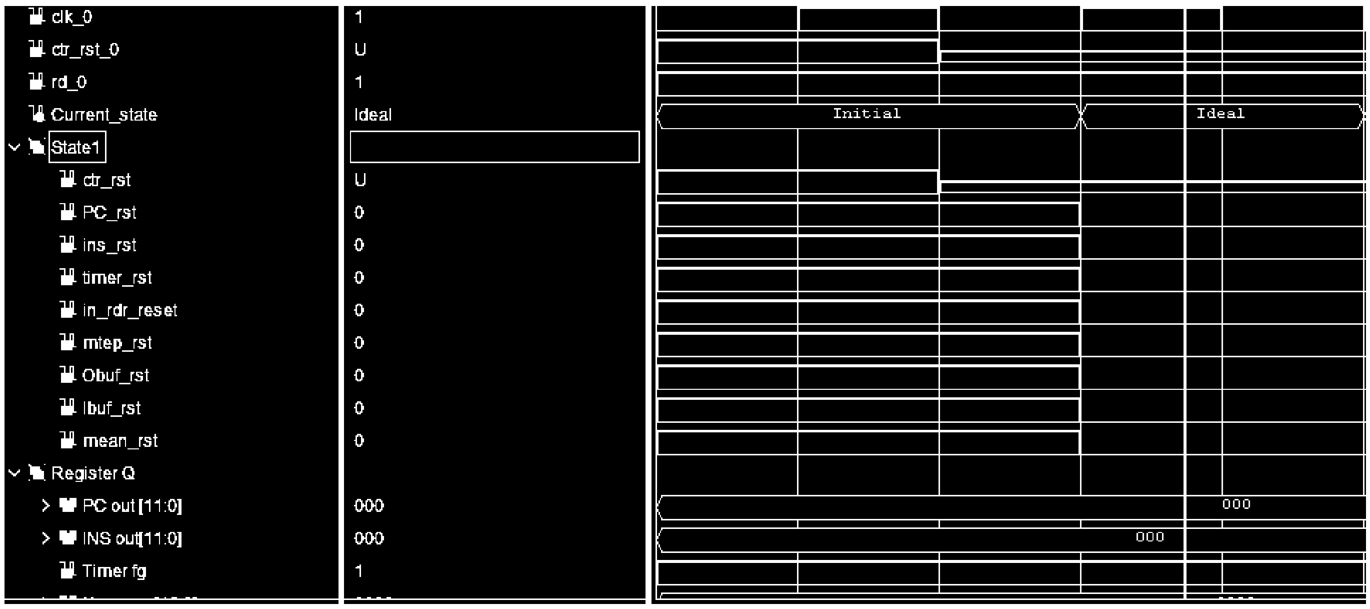
```
1000 0001 000 00010 -- ADD2P 1 0 3 ;Add motor 2 to profile 1

1000 0001 000 00011 -- ADD2P 1 0 4 ;Add motor 3 to profile 1
1000 0001 100 00000 -- ADD2P 1 0 0 ;Add all temp sensor to profile 1
1001 0001 000 00010 -- Point 1 0 2 ;Point to motor2 profile1
1010 001111111111   -- Setval 511  ; Set value 511 to located point
1001 0001 000 00011 -- Point 1 0 3 ;Point to motor3 profile 1
1010 001001110001   -- Setval 625  ; Set value 625 to located point
1011 000000000001   -- Control 1   ; Start profile 1
1111 000000000111   -- Waitt 7     ; wait 7 clocks
1100 000100000000   -- Findmean 1,0 ;Find mean val of all temp sensor
1101 000011001000   -- compare 200 ; find the mean >200
1110 000000001001   -- JIC 9       ; if mean > given vale, jump to 8
0000 000000000000   -- Reset       ; reset
```

To demonstrate the Sensor values, use another memory element. It act as a sensor driver and provide sensor read value. To verification purpose, initialized this RAM randomly using below python snippet.

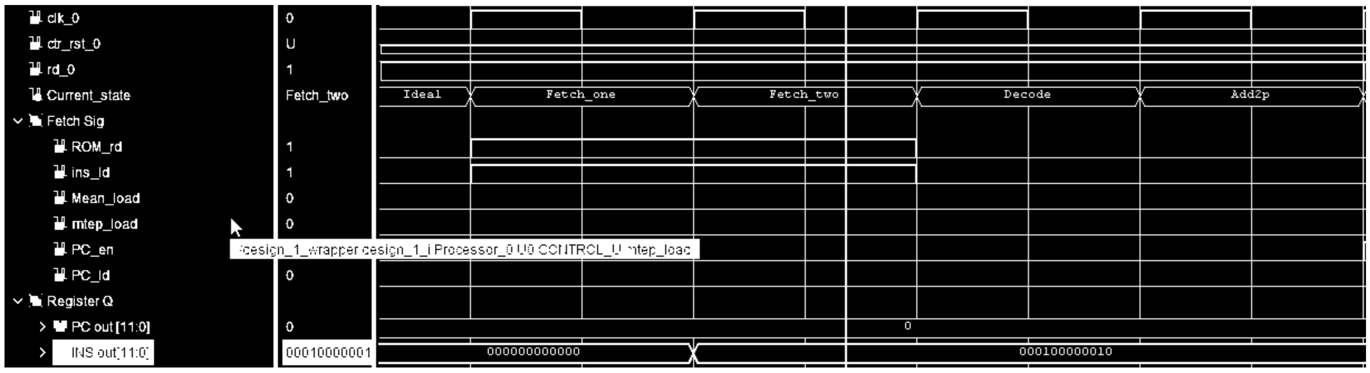


```
import numpy as np
file = open("sensor_val.txt","w")

for j in range(0,2**7):
    bval = np.random.randint(2,size=8)
    print(bval)
    for i in range(0,8):
        file.write(str(bval[i]))
    file.write("\n")

file.close()
```

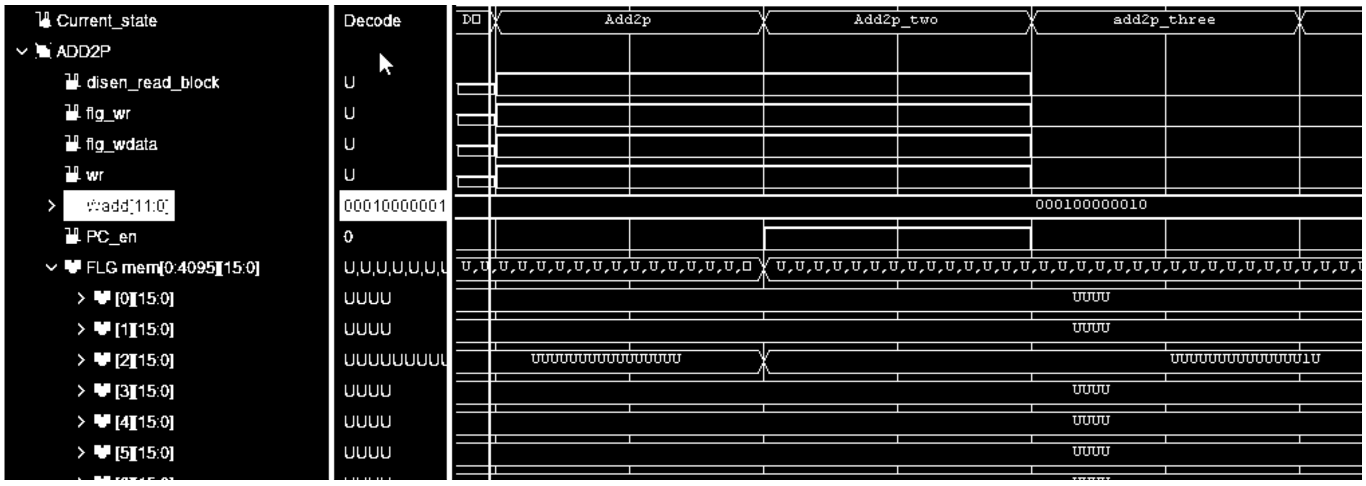
1. Initially control unit set all registers to reset state(0 state).



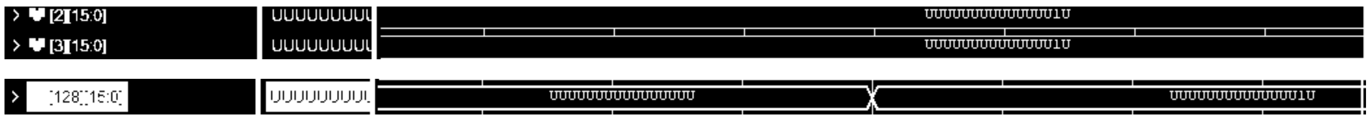
2. Fetch & Decode



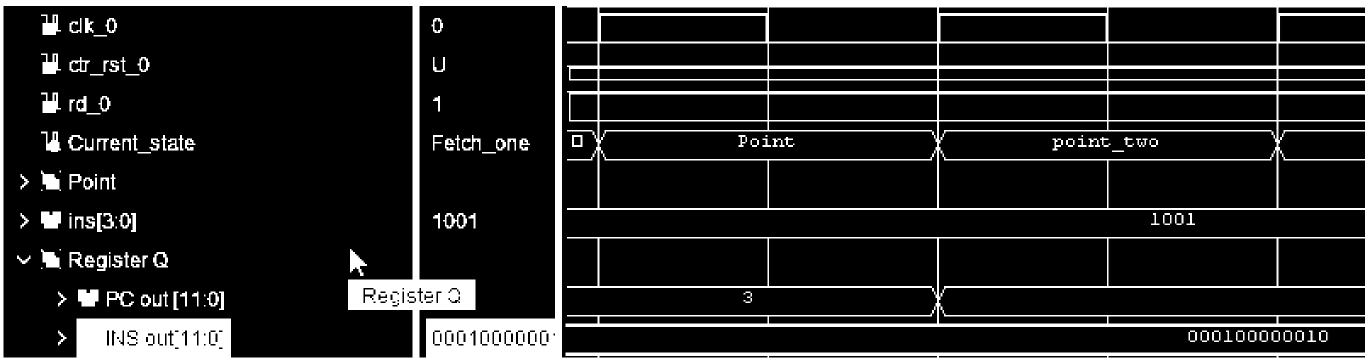
3. Add2P

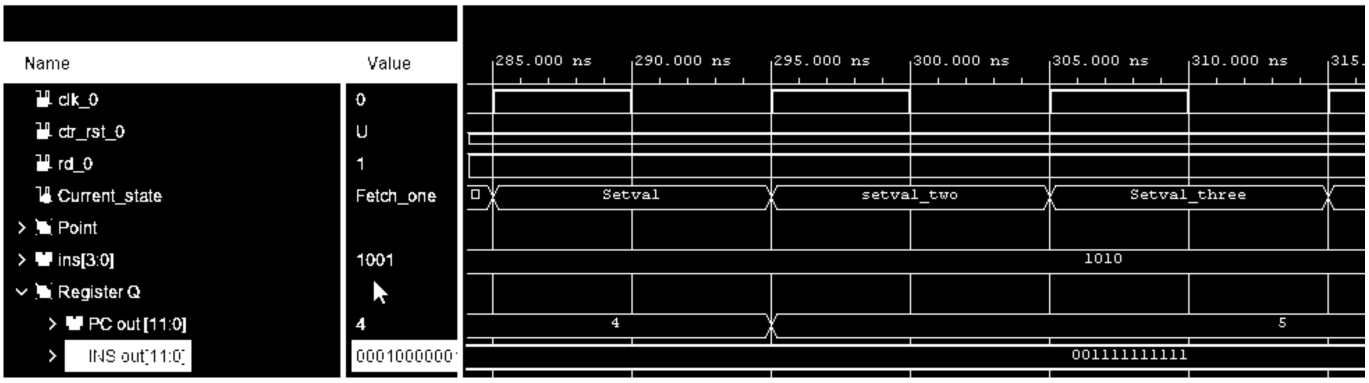


After add motor 2,3 and all temp sensors, the flag memory consists as this,

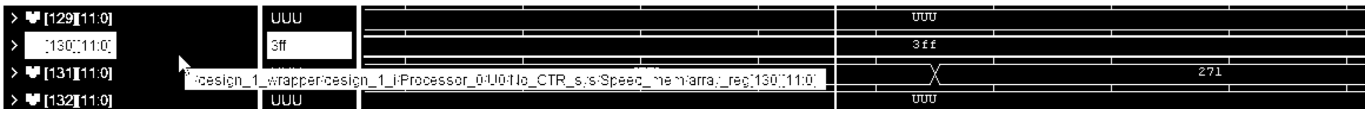

[2][15:0] UUUUUUUUU UUUUUUUUUUUUUU1U
[3][15:0] UUUUUUUUU UUUUUUUUUUUUUU1U
[128][15:0] UUUUUUUUU UUUUUUUUUUUUUUUU UUUUUUUUUUUUUU1U

4. Point and Setval



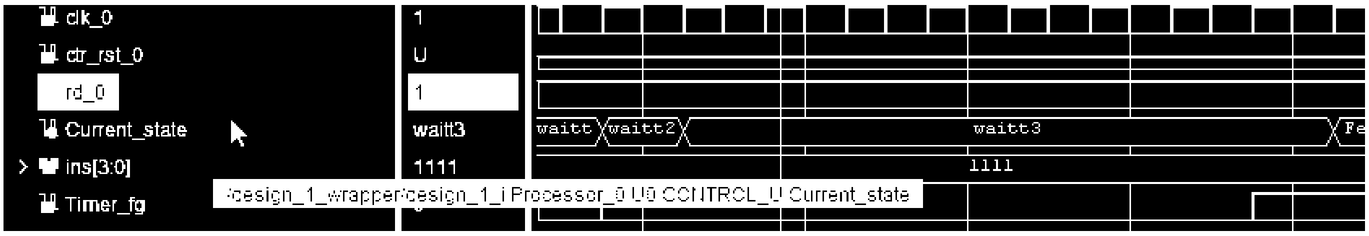


After the point and setval instructions, these are the speed memory content,



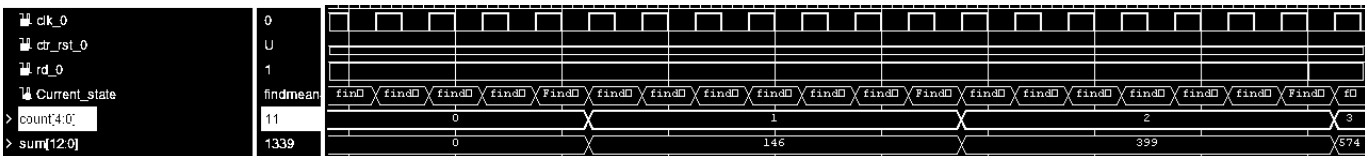
[129][11:0]
[130][11:0]
[131][11:0]
[132][11:0]
UUU
3ff
UUU
UUU
3ff
271
UUU
/design_1_wrapper/design_1_i/Processor_0/U0/No_CTR_s/s/Speed_mem/array_reg[130][11:0]

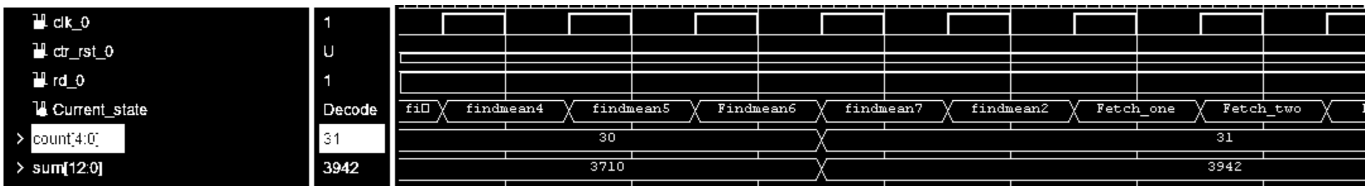
5. Waitt



6. Findmean

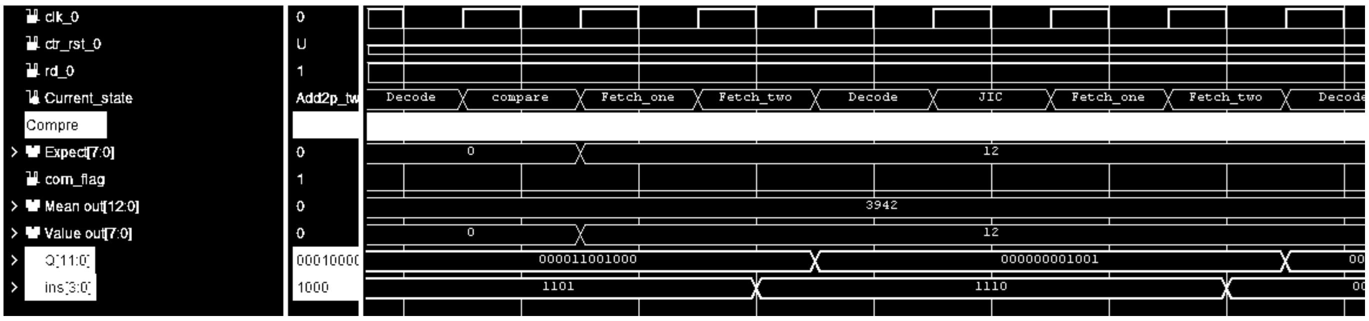
Findmean is the most computationally expansive operation in this processor. It summing up all the enabled sensor values and locate them in mean register.





7. Compare & JIC

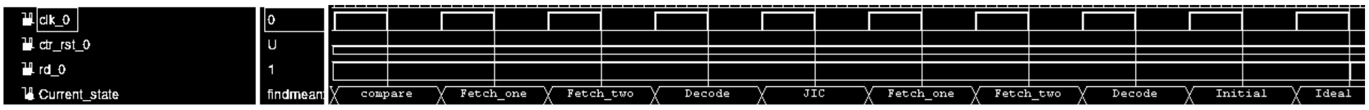
Check the mean value < Expected value. If true, go to given address. Else continue. In this example condition was true.



8. Reset

Reset to the Initial stage.

Reset processor to initial stage.



6 SOCIAL IMPACT AND ENVIRONMENTAL ISSUES

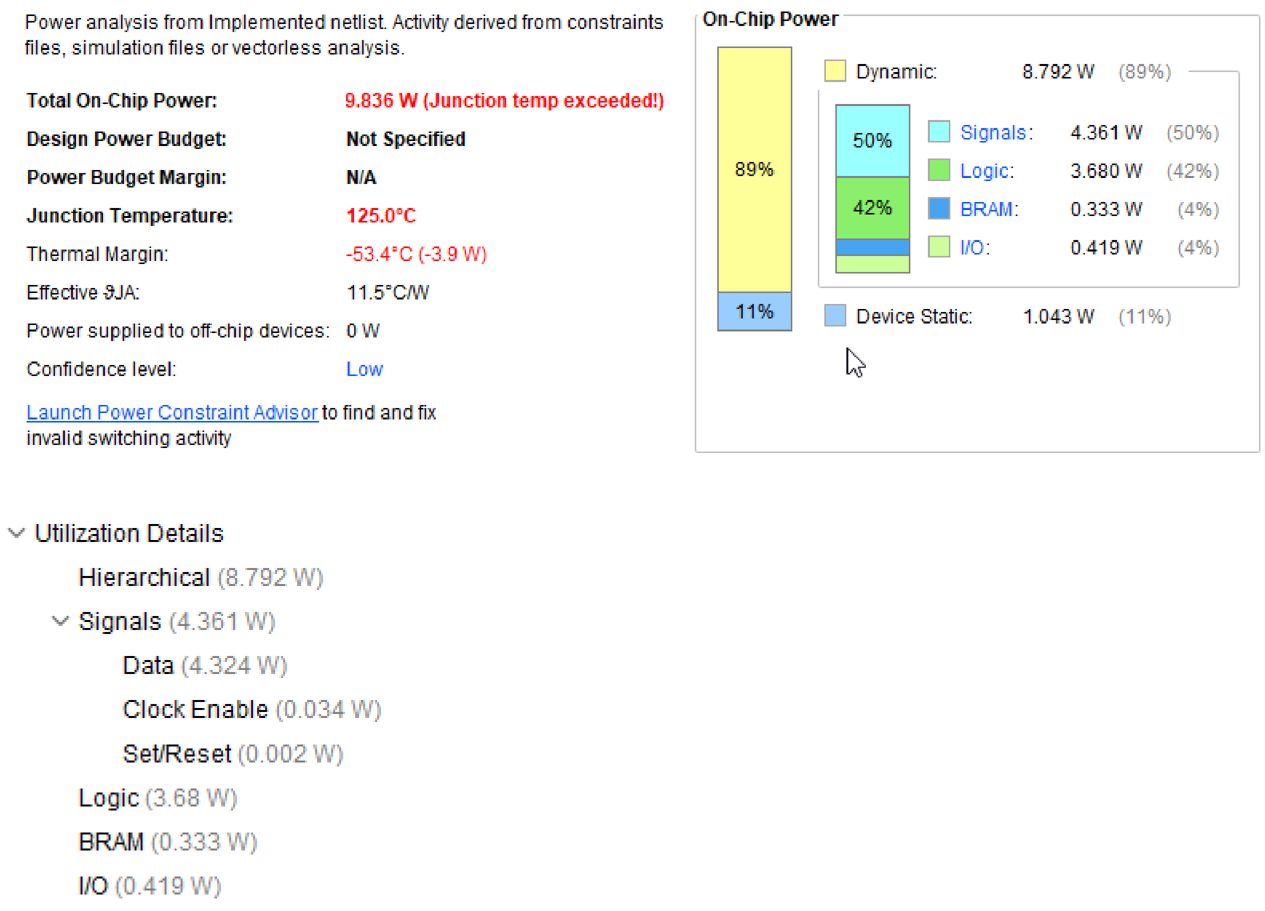
1. Improved safety: The processor could help reduce the risk of accidents and injuries in mixing processes, leading to a safer working environment for employees.
2. Increased efficiency: The processor could lead to more efficient mixing processes, potentially reducing the amount of time and resources required for production.
3. Job creation: The development and implementation of the processor could create job opportunities for individuals skilled in operating and maintaining the technology.

The environmental issues related to this processor design could include:

1. Energy consumption: The processor may require a significant amount of energy to operate, potentially contributing to increased energy consumption and greenhouse gas emissions.
2. Waste generation: The mixing process controlled by the processor may produce waste materials that need to be properly managed and disposed of to minimize environmental impact.
3. Resource depletion: The production and operation of the processor could contribute to the depletion of natural resources, such as metals and minerals used in its construction.

7 PERFORMANCE

7.1 Power Analysis



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**9.836 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-53.4°C (-3.9 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

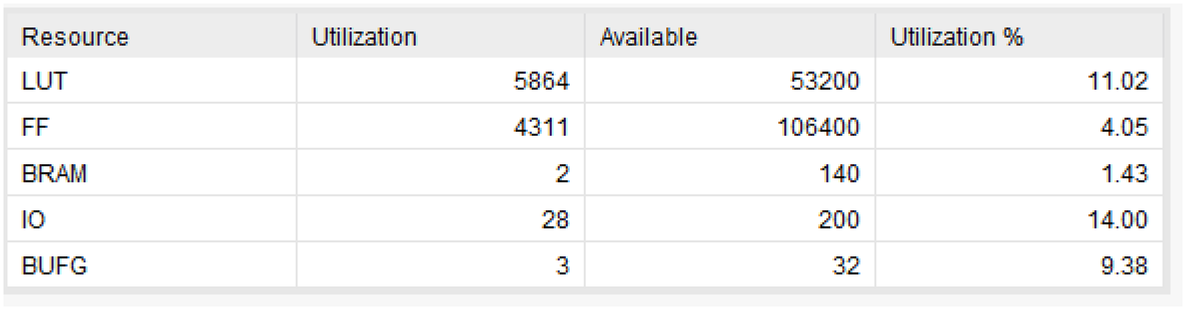
On-Chip Power

Dynamic:		8.792 W	(89%)
	Signals:	4.361 W	(50%)
	Logic:	3.680 W	(42%)
	BRAM:	0.333 W	(4%)
	I/O:	0.419 W	(4%)
Device Static:		1.043 W	(11%)

Utilization Details
- Hierarchical (8.792 W)
- Signals (4.361 W)
  - Data (4.324 W)
  - Clock Enable (0.034 W)
  - Set/Reset (0.002 W)
- Logic (3.68 W)
- BRAM (0.333 W)
- I/O (0.419 W)

8 COST OF DESIGN

1. **Material Cost**



Resource	Utilization	Available	Utilization %
LUT	5864	53200	11.02
FF	4311	106400	4.05
BRAM	2	140	1.43
IO	28	200	14.00
BUFG	3	32	9.38

2. **Research and development**: This includes the cost of conducting research, feasibility studies, and prototyping to develop the processor technology. (Nearly 1 week)
3. **Design Cost** – (Nearly 10 days)
4. **Labor**: The cost of hiring skilled engineers, designers, and technicians to work on the processor design.(1 person with two advisors)
5. **Testing and validation**: The cost of testing the processor to ensure it meets performance and safety standards, as well as obtaining necessary certifications.
6. **Overhead and administrative costs**: These include general operational costs such as office space, utilities, and administrative expenses.(University Laboratory)