HO CHI MINH UNIVERSITY OF TECHNOLOGY

ELECTRICAL ELECTRONIC ENGINEERING

Microprocessor

★ ★ ★

MINI PROJECT

<u>Control motor and LCD 16x2</u>

**INSTRUCTOR : ** Nguyen Tuan Hung (EE3414)

STUDENTS :

Le Tuan An - 2051079

Ha Gia Huy - 2051117

Tran Minh Tri - 2051022

April 11TH, 2023

CATALOG

1. Requirement:

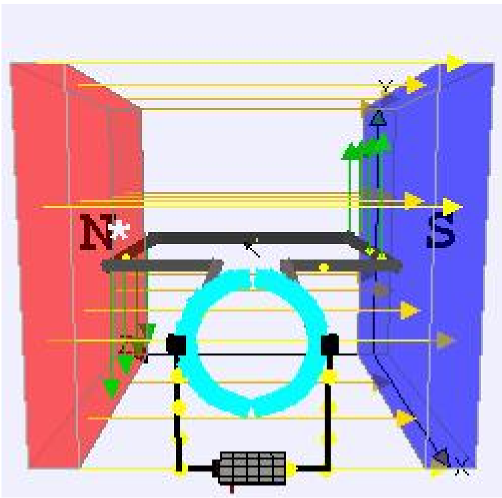
- Build a system using Nios II in kit DE10 to connect a LCD 16x2 and an H-bridge to control a motor. This system can do the following tasks:
  - When SW0 is ON, LCD blinks the sentence “Hello World !!!” in the middle of row 1 with frequency 1Hz. (Using timer)
  - When SW1 is ON, Nios II controls the motor by sending PWM pulses to the H-bridge. LCD displays the duty cycle and the frequency of PWM pulses.
  - When SW0 and SW1 are OFF, turn off the system.

- EXTENSION:
  - SW1, SW2, SW3 will control the speed of the DC motor based on the PWM Pulses be created by DE-10 kit nano.

2. How DC Motors Work

The simplest of all motors, DC motors turn when a DC voltage is applied across it. This kind of motor can be found in drones, power tools, and robots. A DC motor can change speend and direction depending on how much power is fed to it and in which direction.

The DC motor uses the uses a magnetic field generated by the by an electromagnet to turn the armature of a motor. The electromagnet is activated by applying voltage, so when the power is on, the magnetic field it generates will cause the armature (a coil of wire) to generate its own nagmetic field, these fields push eachother away and cause the armature to spin.



To get the motor to spin the other way, we need to reverse the applied voltage, meaning the flow of current through the motor will be reversed. Unfortunately switching the direction of current from a controller like the Omega is difficult. The processors use low current and voltages, plus they are usually disconnected from the motor to prevent inductive feedback distrupting their operation.

If only there was some kind of device that can help us control the power we supply to our DC motors…

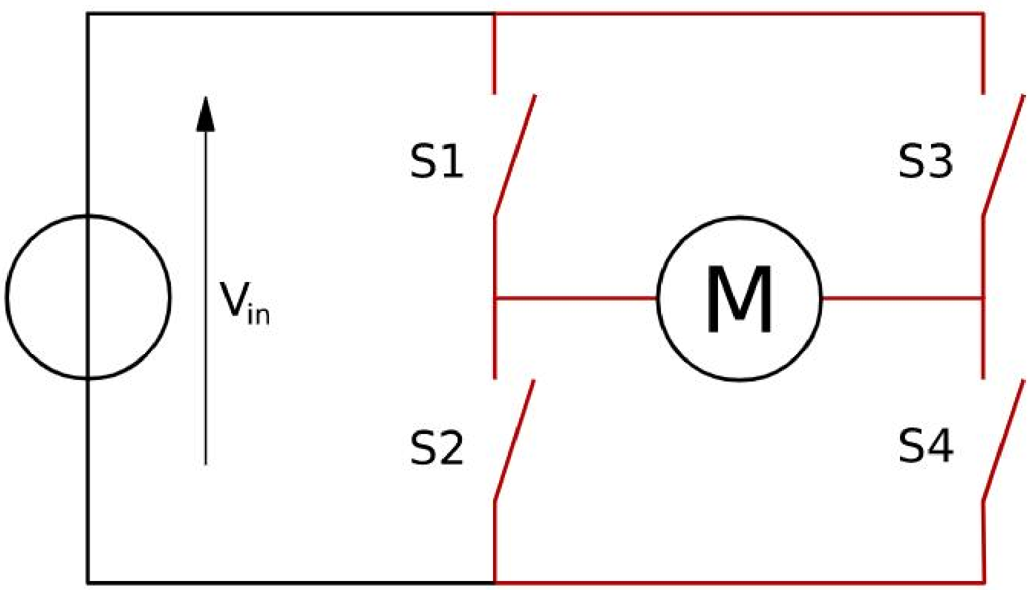
Note that applying current to both terminals can cause damage to the motor.

3. How H-Bridges Work

An H-Bridge is a circuit that allows voltages to be applied across a load in either direction. Electric current flows from the source to ground, and many components need to be oriented according to the direction of current to work as expected. An H-Bridge is a circuit built to change the direction of the voltage and thus the current flowing to a load.

In electrical terms, a **load** is any piece of a circuit that consumes electric energy to do things - heating, turning, lighting up, and so on.

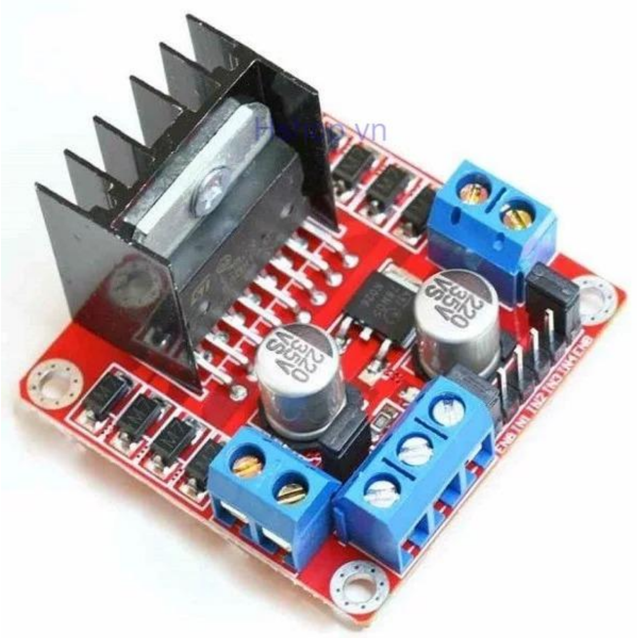
An H-Bridge is made up of four switches: two in series, and two in parallel, with the load placed in between the switches. In this configuration the circuit takes an “H” shape.

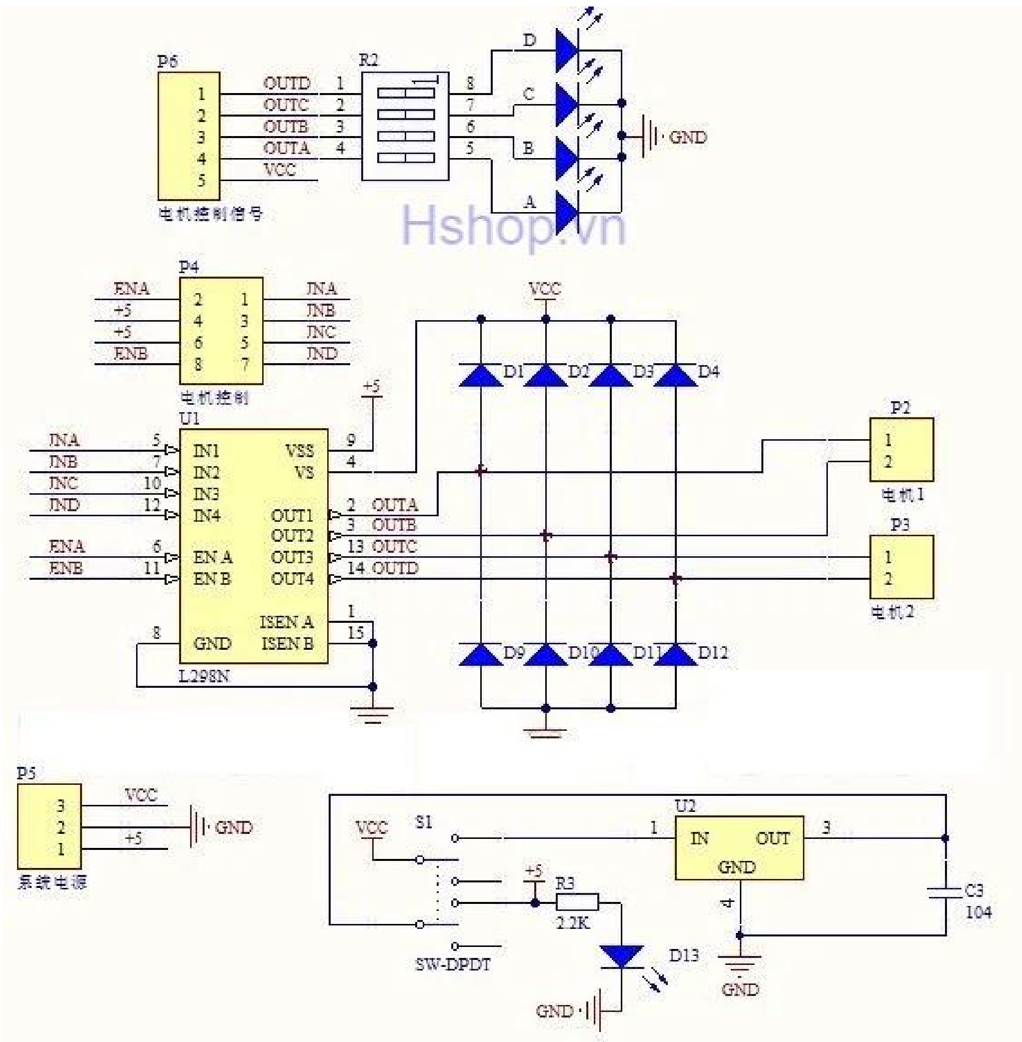


In order to change the direction of the voltage supplied, the H-Bridge controls the switches that deliver power to the load (S1). Looking at the diagram, if we close S1 and S4 while leaving the rest open, the voltage will be applied from left to right across the motor. If S2 and S3 are closed instead and the others open, the voltage will be applied from right to left.

This configuration has potential to create a short circuit, so most H-Bridges do not allow direct control of these switches.

4. L298 H-Bridge module (H shop)



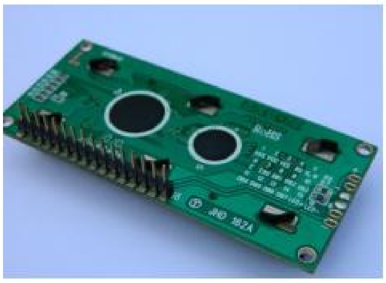


5. Parallel Interfacing: Interfacing LCD Display

5.1 Objective

Frequently, a microcontroller program must interact with the outside world using input and output devices that communicate directly with a human being. One of the most common devices attached to a microcontroller is an LCD display. Some of the most common LCDs are 16x2 and 20x2 displays. This means 16 characters per line by 2 lines and 20 characters per line by 2 lines, respectively.

(a) Front View



(b) Back View

5.2 LCD Controller Pins

Signal	No. of Lines	I/O	Device Interfaced with	Function
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/$\overline{W}$	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation.

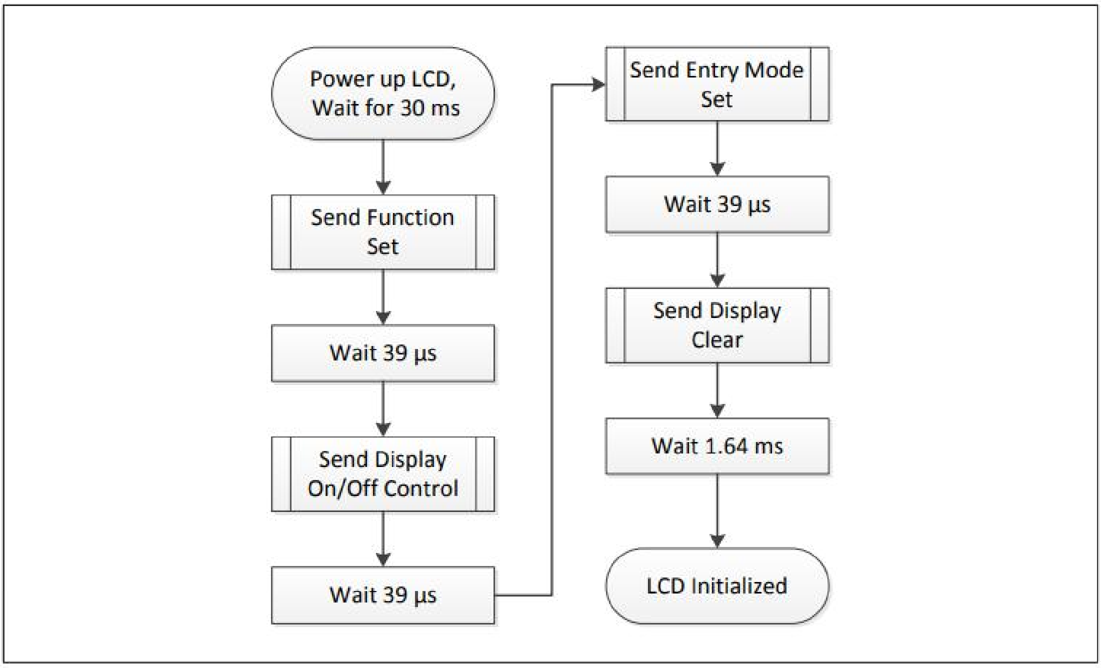
5.3 LCD Controller Commands

Instruction	Hexadecimal	Decimal
Function Set: 8-bit, 1 Line, 5x7 Dots	0x30	48
Function Set: 8-bit, 2 Line, 5x7 Dots	0x38	56
Function Set: 4-bit, 1 Line, 5x7 Dots	0x20	32
Function Set: 4-bit, 2 Line, 5x7 Dots	0x28	40
Entry Mode	0x06	6
Display off Cursor off (clearing display without clearing DDRAM content)	0x08	8
Display on Cursor on	0x0E	14
Display on Cursor off	0x0C	12
Display on Cursor blinking	0x0F	15
Shift entire display left	0x18	24
Shift entire display right	0x1C	30
Move cursor left by one character	0x10	16
Move cursor right by one character	0x14	20
Clear Display (also clear DDRAM content)	0x01	1
Set DDRAM address or coursor position on display	0x80 + A	128 + A
Set CGRAM address or set pointer to CGRAM location	0x40 + A	64 + A

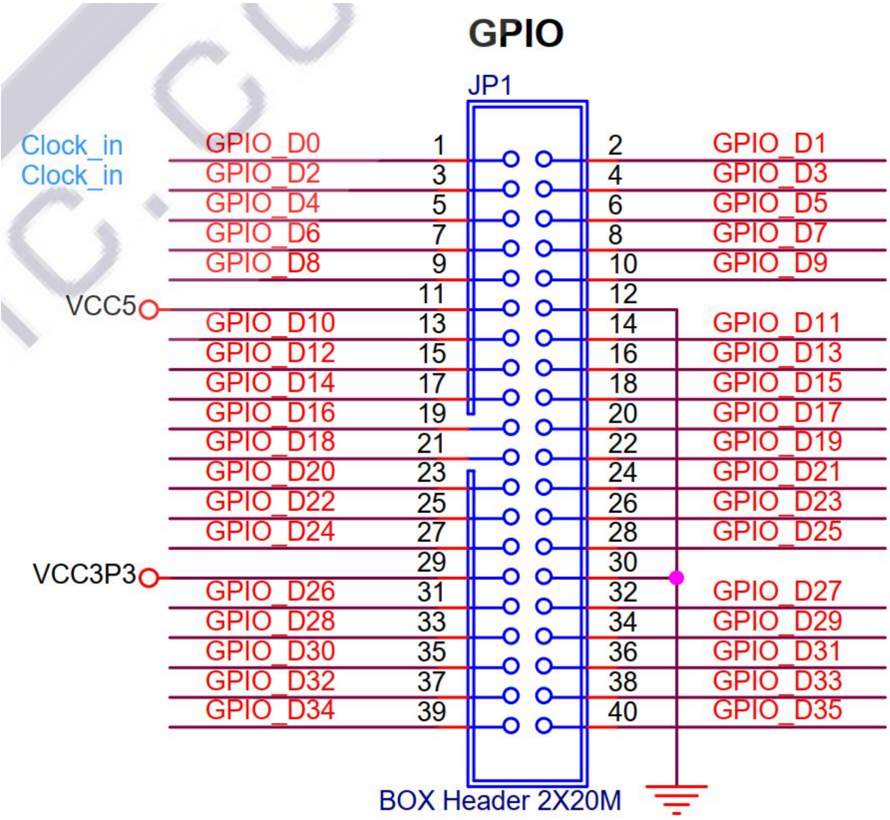
5.4 LCD Initialization

Before displaying characters on the LCD display, it must be configured first. To configure an LCD display, four command words must be sent to LCD. The commands are:

Function set=>display on/off control=>entry mode set=>display Clear



6. GPIO configuration of DE10 standard kit (DE10 standard datasheet)



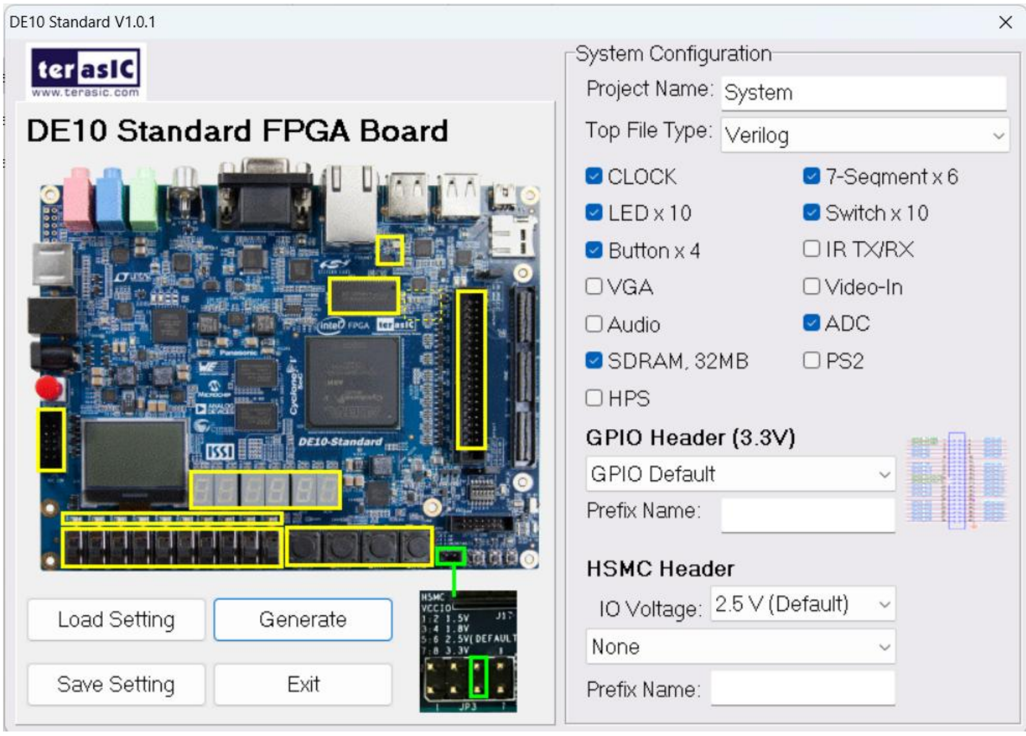
❖ **Table of configuration**

PIN OF DE10 KIT	PIN OF COMPONENT	COMPONENT
GPIO_D1	INA	L298
GPIO_D2	INB	
GPIO_D28	D0	LCD16x02
GPIO_D30	D1	
GPIO_D32	D2	
GPIO_D34	D3	
GPIO_D29	D4	
GPIO_D31	D5	
GPIO_D33	D6	
GPIO_D35	D7	
GPIO_D23	RS	
GPIO_D24	RW	
GPIO_D25	E	

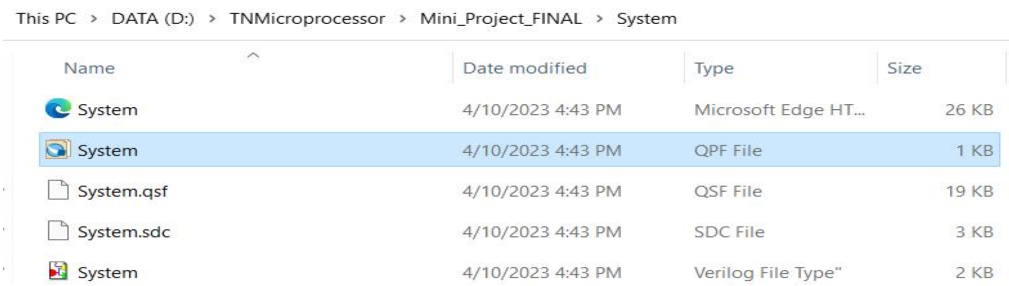
7. Quartus

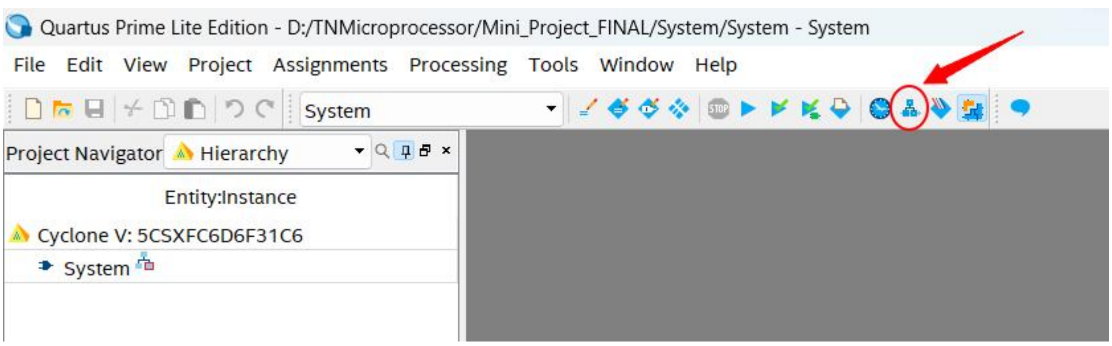
7.1 System Builder

➢ **Step 1:** Open System Builder tool and choose necessary peripherals. Save project in a directory path which MUST NOT contain space.



➢ **Step 2:** Open project using Quartus 18.1, and choose icon to open Platform Designer. At tab IP Catalog, search IP and add to our system.

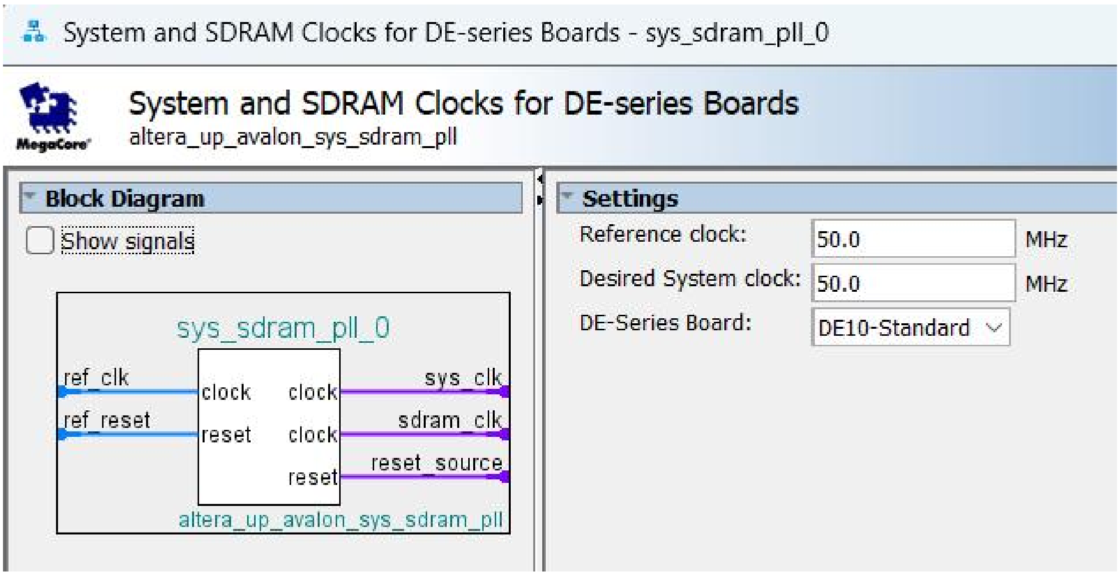




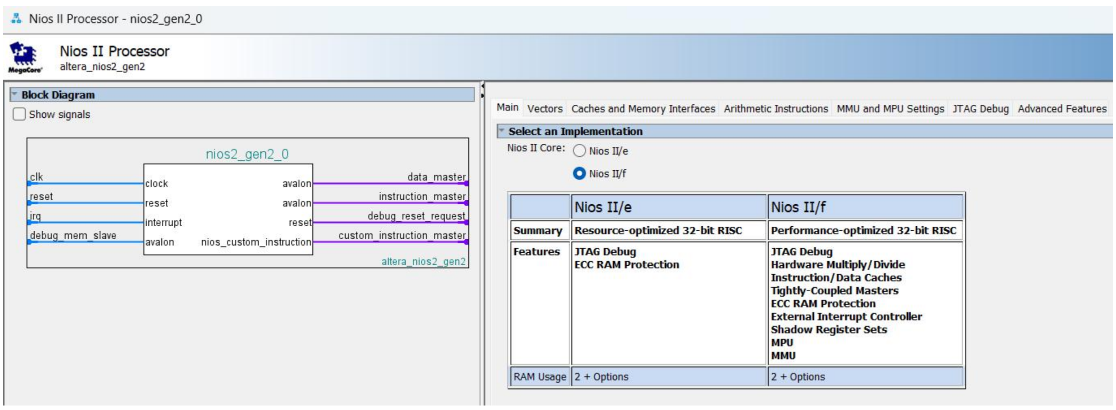
7.2 Platform Designer

7.2.1 Using following components for system:

- System and SDRAM Clocks for DE-series Boards
- Nios II Processor
- On-chip Memory (RAM or ROM) Intel FPGA IP
- SDRAM Controller Intel FPGA IP
- System ID Peripheral Intel FPGA IP
- JTAG UART Intel FPGA IP
- Two Interval Timer Intel FPGA IP (Timer 0 and Timer 1)
- Three blocks PIO (Parallel I/O) Intel FPGA IP

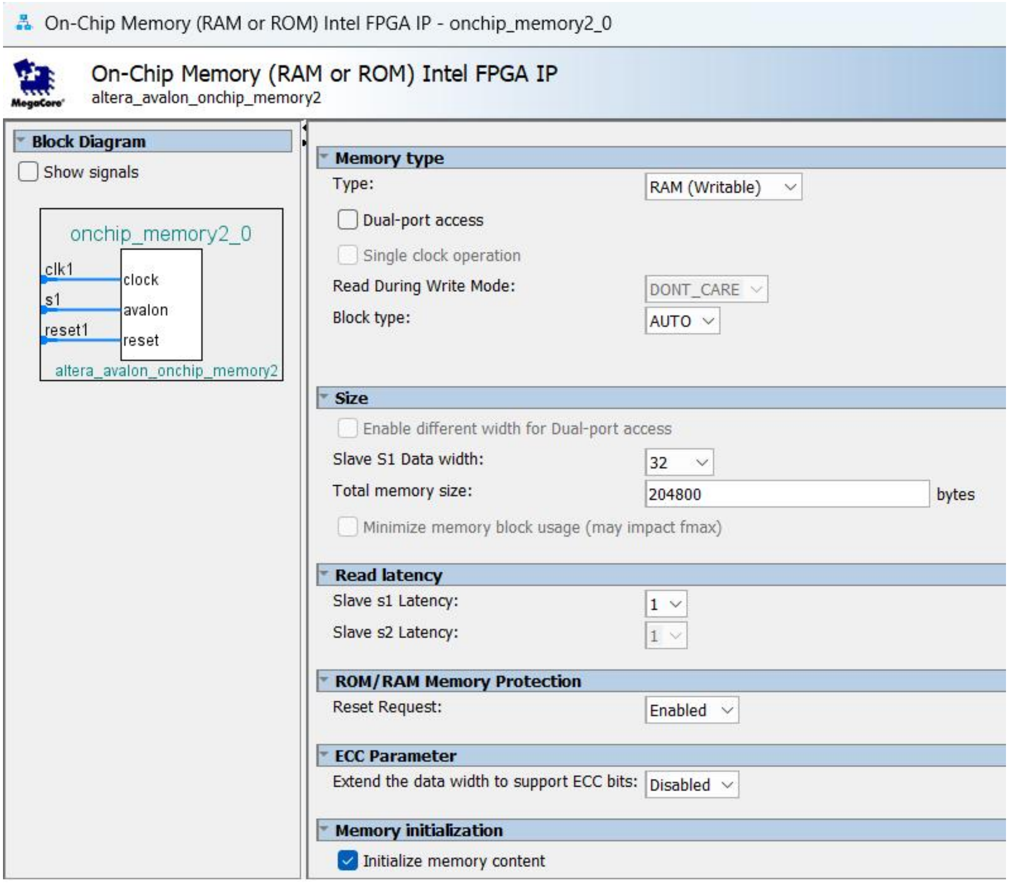


System and SDRAM Clocks for DE-series Boards

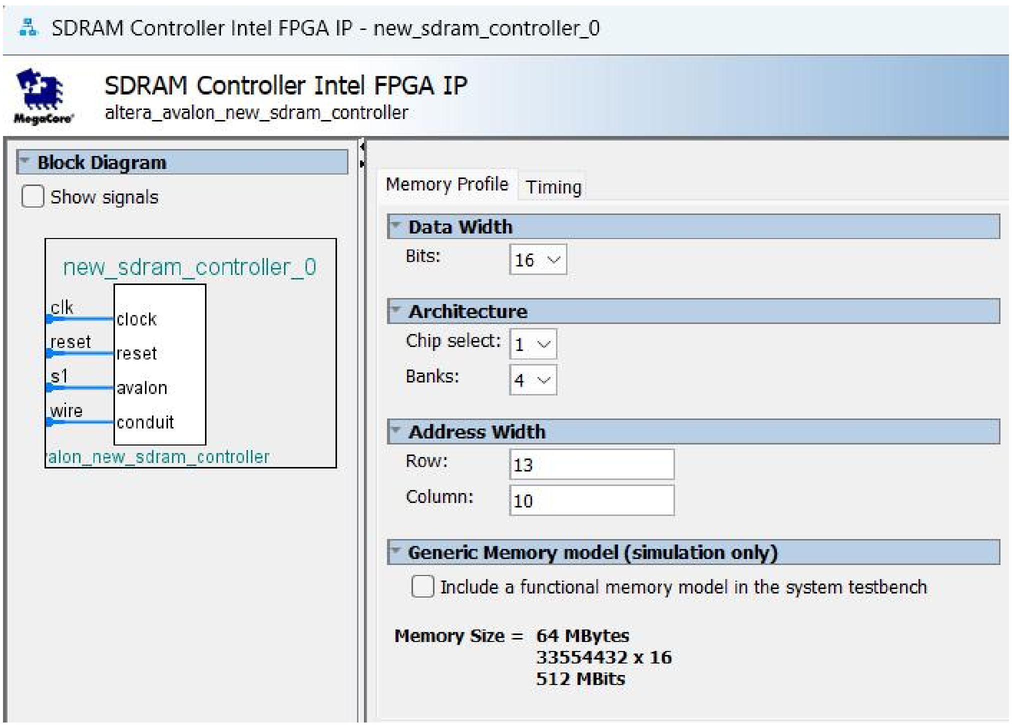


	Nios II/e	Nios II/f
Summary	Resource-optimized 32-bit RISC	Performance-optimized 32-bit RISC
Features	JTAG Debug ECC RAM Protection	JTAG Debug Hardware Multiply/Divide Instruction/Data Caches Tightly-Coupled Masters ECC RAM Protection External Interrupt Controller Shadow Register Sets MPU MMU
RAM Usage	2 + Options	2 + Options

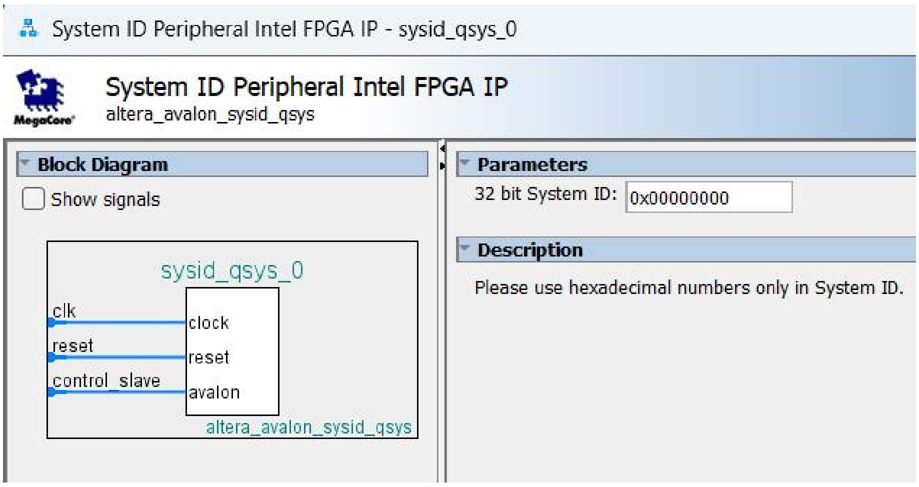
Nios II Processor



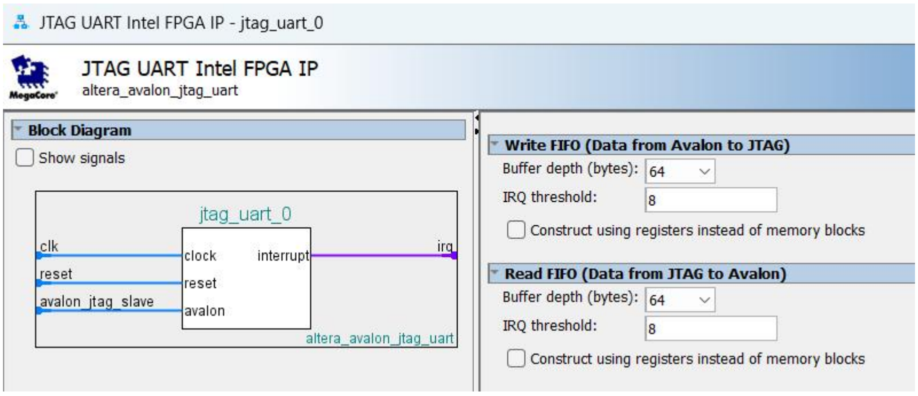
On-chip Memory (RAM or ROM) Intel FPGA IP



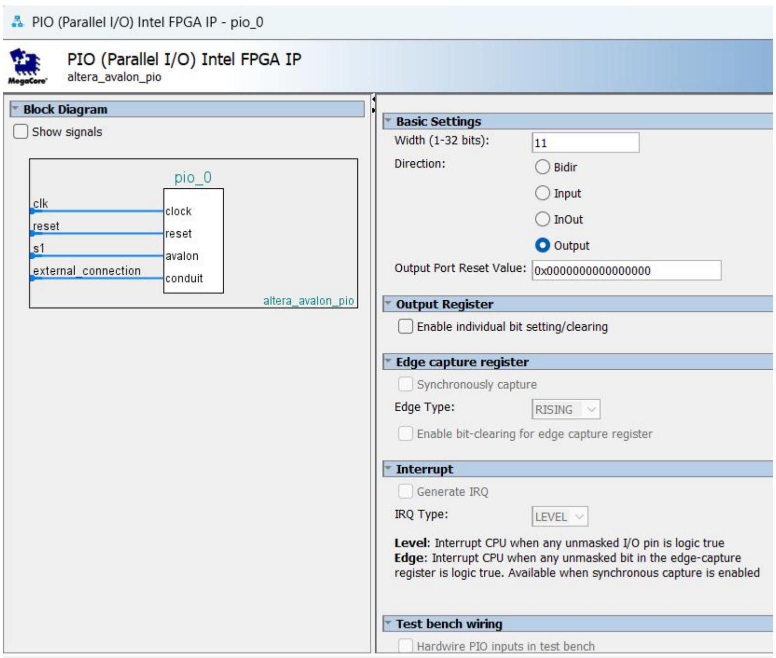
SDRAM Controller Intel FPGA IP



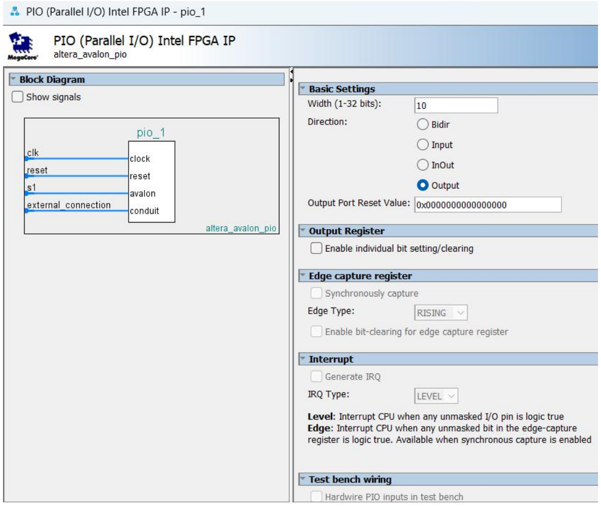
System ID Peripheral Intel FPGA IP



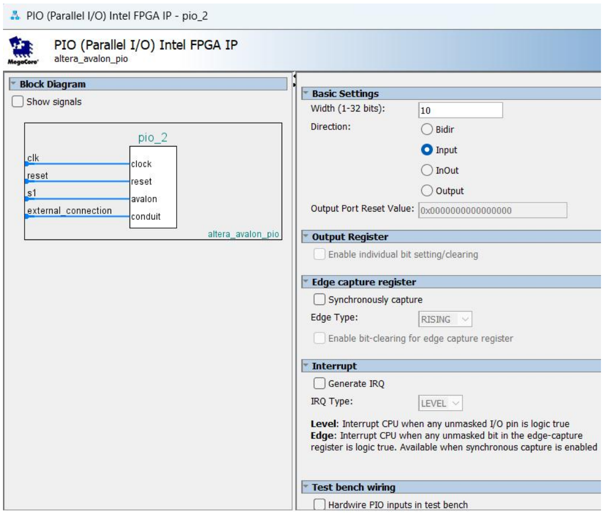
JTAG UART Intel FPGA IP



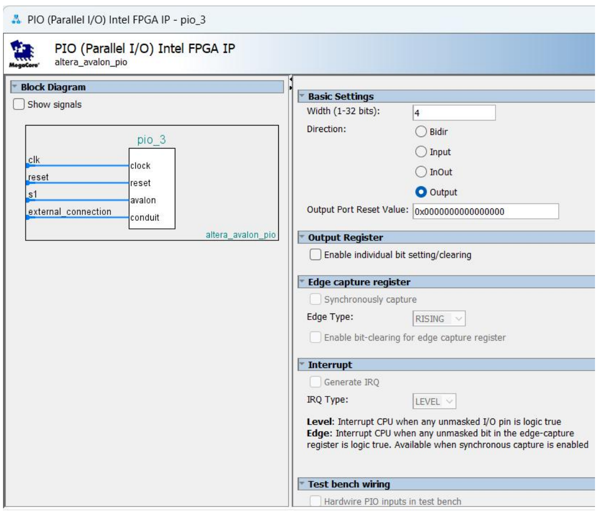
LCD



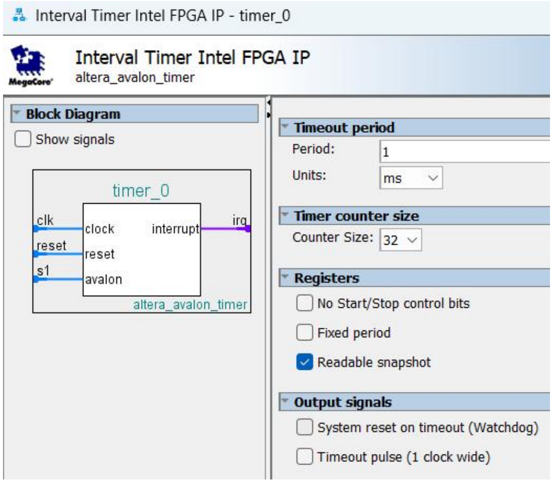
LED



SWITCH

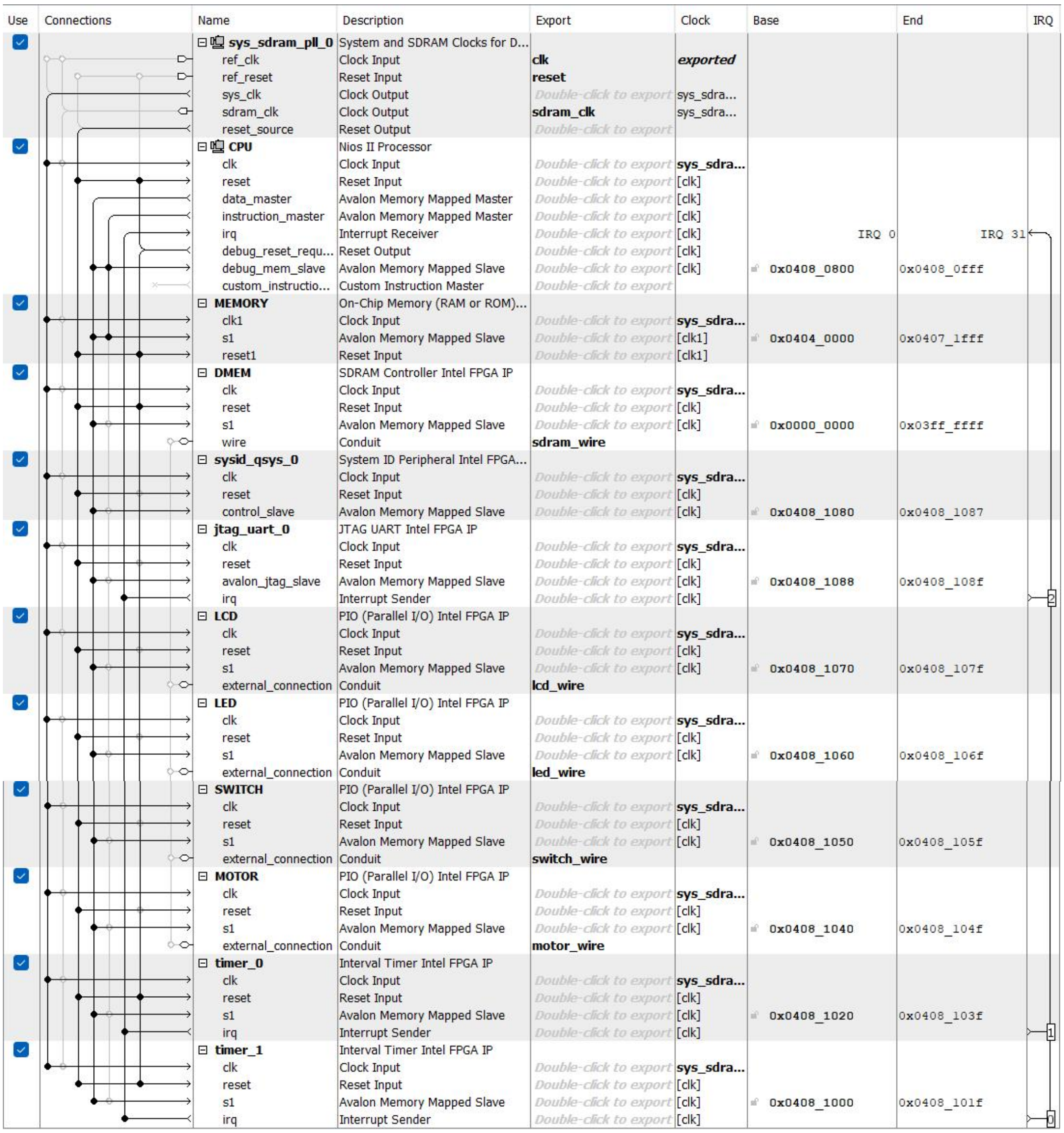


MOTOR



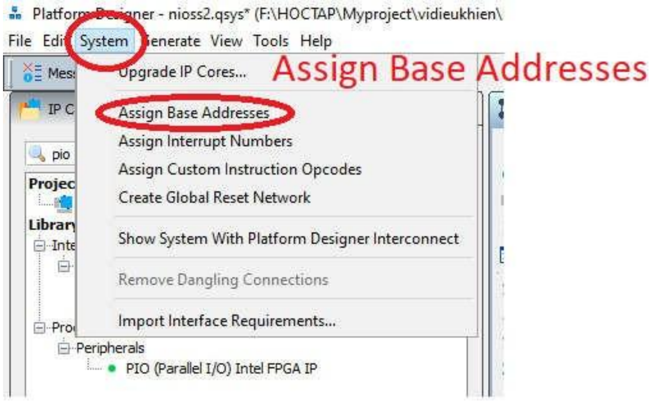
Interval Timer Intel FPGA IP

7.2.2 Connect IP as the picture:

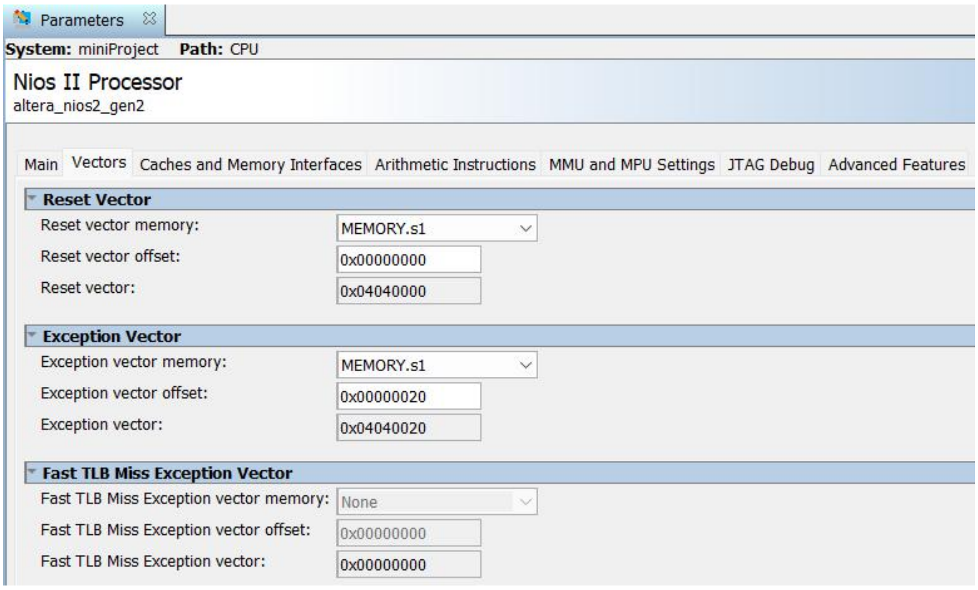


Name	Description	Export	Clock	Base	End	IRQ
sys_sdram_pll_0	System and SDRAM Clocks for D...					
ref_clk	Clock Input	clk	exported			
ref_reset	Reset Input	reset				
sys_clk	Clock Output	Double-click to export	sys_sdra...			
sdram_clk	Clock Output	sdram_clk	sys_sdra...			
reset_source	Reset Output	Double-click to export				
CPU	Nios II Processor					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
irq	Interrupt Receiver	Double-click to export	[clk]	IRQ 0	IRQ 31	
debug_reset_requ...	Reset Output	Double-click to export	[clk]			
debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_0800	0x0408_0fff	
custom_instructio...	Custom Instruction Master	Double-click to export				
MEMORY	On-Chip Memory (RAM or ROM)...					
clk1	Clock Input	Double-click to export	sys_sdra...			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0404_0000	0x0407_1fff	
reset1	Reset Input	Double-click to export	[clk1]			
DMEM	SDRAM Controller Intel FPGA IP					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x03ff_ffff	
wire	Conduit	sdram_wire				
sysid_qsys_0	System ID Peripheral Intel FPGA...					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_1080	0x0408_1087	
jtag_uart_0	JTAG UART Intel FPGA IP					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_1088	0x0408_108f	
irq	Interrupt Sender	Double-click to export	[clk]			2
LCD	PIO (Parallel I/O) Intel FPGA IP					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_1070	0x0408_107f	
external_connection	Conduit	lcd_wire				
LED	PIO (Parallel I/O) Intel FPGA IP					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_1060	0x0408_106f	
external_connection	Conduit	led_wire				
SWITCH	PIO (Parallel I/O) Intel FPGA IP					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_1050	0x0408_105f	
external_connection	Conduit	switch_wire				
MOTOR	PIO (Parallel I/O) Intel FPGA IP					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_1040	0x0408_104f	
external_connection	Conduit	motor_wire				
timer_0	Interval Timer Intel FPGA IP					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_1020	0x0408_103f	
irq	Interrupt Sender	Double-click to export	[clk]			1
timer_1	Interval Timer Intel FPGA IP					
clk	Clock Input	Double-click to export	sys_sdra...			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0408_1000	0x0408_101f	
irq	Interrupt Sender	Double-click to export	[clk]			0

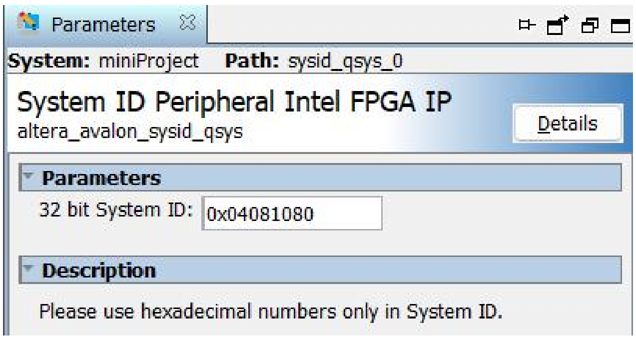
7.2.3 Assign Base Address as picture below



7.2.4 Change Reset Vector Memory and Exception Vector Memory



7.2.5 Change System ID in SytemID IP to base address of it



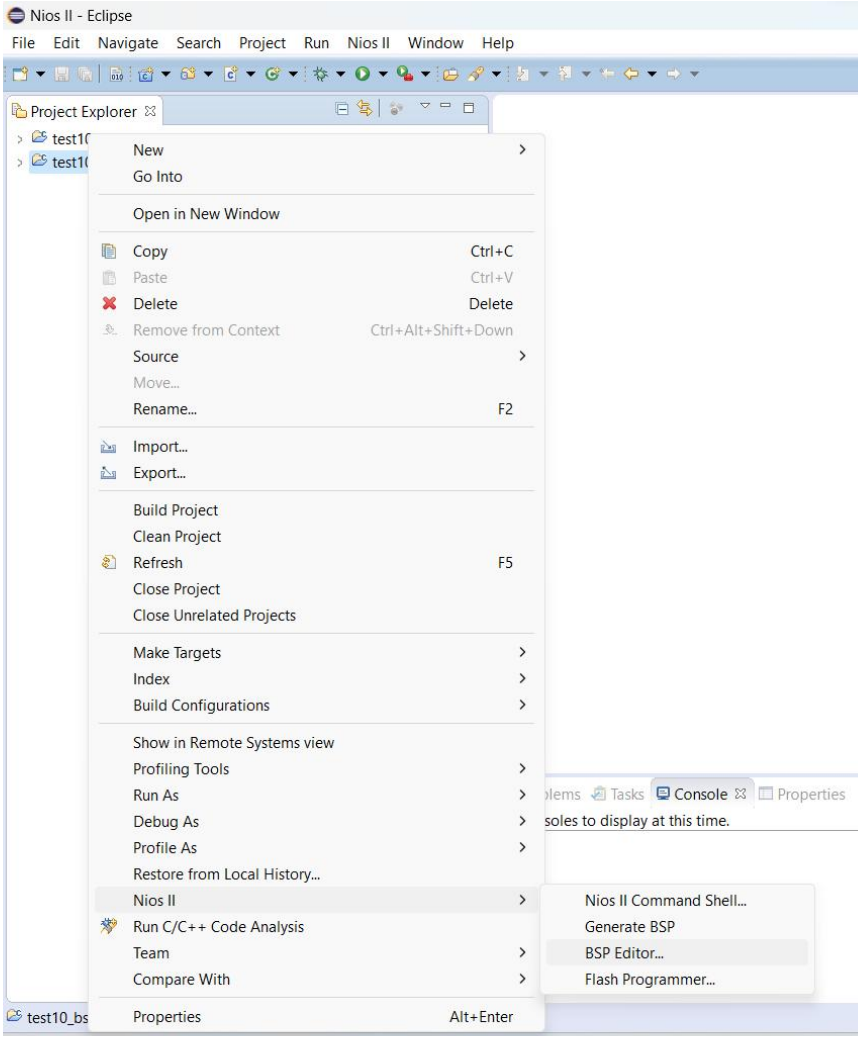
7.3 VERILOG code

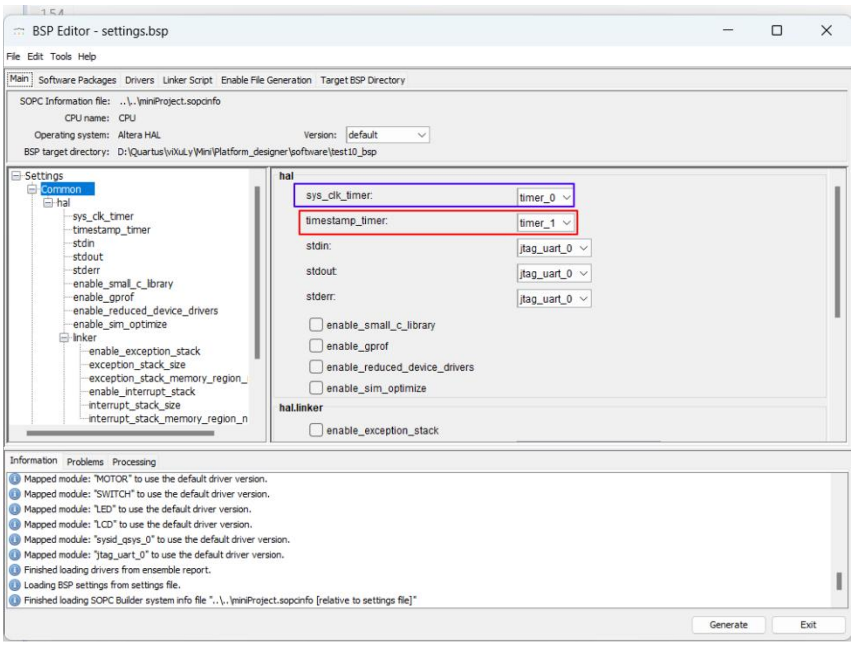
```
module mini(
	//////////// CLOCK //////////
	input		CLOCK2_50,
	input		CLOCK3_50,
	input		CLOCK4_50,
	input		CLOCK_50,

	//////////// KEY //////////
	input	[3:0]	KEY,
	//////////// SW //////////
	input	[9:0]	SW,
	//////////// LED //////////
	output	[9:0]	LEDR,

	//////////// Seg7 //////////
	output	[6:0]	HEX0,
	output	[6:0]	HEX1,
	output	[6:0]	HEX2,
	output	[6:0]	HEX3,
	output	[6:0]	HEX4,
	output	[6:0]	HEX5,

	//////////// SDRAM //////////
	output	[12:0]	DRAM_ADDR,
	output	[1:0]	DRAM_BA,
	output		DRAM_CAS_N,
	output		DRAM_CKE,
	output		DRAM_CLK,
	output		DRAM_CS_N,
	inout	[15:0]	DRAM_DQ,
	output		DRAM_LDQM,
	output		DRAM_RAS_N,
	output		DRAM_UDQM,
	output		DRAM_WE_N,

	//////////// ADC //////////
	output		ADC_CONVST,
	output		ADC_DIN,
	input		ADC_DOUT,
	output		ADC_SCLK,

	//////////// GPIO, GPIO connect to GPIO Default //////////
	inout	[35:0]	GPIO
);
```

```
//=======================================================
//  REG/WIRE declarations
//=======================================================
wire [1:0] DQM;
assign DRAM_LDQM = DQM[0];
assign DRAM_UDQM = DQM[1];


//=======================================================
//  Structural coding
//=======================================================
miniProject (
                .clk_clk(CLOCK_50),
                .lcd_wire_export({GPIO[23],GPIO[24],GPIO[25],
                                 GPIO[35],GPIO[33],GPIO[31],GPIO[29],
                                 GPIO[34],GPIO[32],GPIO[30],GPIO[28]}),
                .led_wire_export(LEDR),
                .motor_wire_export({GPIO[1],GPIO[2],GPIO[3],GPIO[4]}),
                .reset_reset(1'b0),
                .sdram_clk_clk(DRAM_CLK),
                .sdram_wire_addr(DRAM_ADDR),
                .sdram_wire_ba(DRAM_BA),
                .sdram_wire_cas_n(DRAM_CAS_N),
                .sdram_wire_cke(DRAM_CKE),
                .sdram_wire_cs_n(DRAM_CS_N),
                .sdram_wire_dq(DRAM_DQ),
                .sdram_wire_dqm(DQM),
                .sdram_wire_ras_n(DRAM_RAS_N),
                .sdram_wire_we_n(DRAM_WE_N),
                .switch_wire_export(SW)
        );


endmodule
```

8. Nios II Software Build Tools for Eclipse

8.1 Setting Timer





8.2 Source code

```
/*#####################################################################
                          //Mini Project//

- Build a system using Nios II in kit DE10 to connect a LCD 16x2 and
an H-bridge to control a motor. This system can do the following
tasks:
  + When SW0 is ON, LCD blinks the sentence "Hello World !!!" in the
    Middle of row 1 with frequency 1Hz. (Using timer)
  + When SW1 is ON, Nios II controls the motor by sending PWM pulses
    to the H-bridge. LCD displays the duty cycle and the frequency
    of PWM pulses.
  + When SW0 and SW1 are OFF, turn off the system.

- EXTENSION:
  + SW1, SW2, SW3 will control the speed of the DC motor based on the
    PWM
    Pulses be created by DE-10 kit nano.

- FILE: miniProject

#####################################################################*/

#include <stdio.h>
#include <altera_avalon_pio_regs.h>
#include <alt_types.h>
#include <sys/alt_alarm.h>
#include <system.h>
#include <string.h>
#include <unistd.h>

/*#####################################################
                        |LCD 1602|
- Bits: 11
- Order: RS  RW  E  D7  D6  D5  D4  D3  D2  D1  D0
- 3 bit control (RS RW E):
      + 001: send command
      + 101: send data
      + EN (1->0): data was sent to LCD
#####################################################*/

/*--------------------------------------------------/
 Name:              lcd_write
 Description:  support lcd_cmd and lcd_data to
               send data to LCD controller
 -------------------------------------------------*/

void lcd_write(int data)
{
      // write data and command
      IOWR_ALTERA_AVALON_PIO_DATA(LCD_BASE, data | 0b00100000000);

      myusleep();
      // just set bit EN (1->0) to recognize data sent
      IOWR_ALTERA_AVALON_PIO_DATA(LCD_BASE, data & 0b11011111111);

      myusleep();
}
```

```c
/*------------------------------------------------/
 Name:            lcd_cmd
 Description: send command to LCD controller
 ------------------------------------------------*/

void lcd_cmd(char cmd)
{
      lcd_write(0b00100000000 + cmd);
}

/*------------------------------------------------/
 Name:            lcd_data
 Description: send data to LCD controller
 ------------------------------------------------*/

void lcd_data(char data) {
      lcd_write(0b10100000000 + data);
}

/*------------------------------------------------/
 Name:            lcd_init
 Description: Initialize LCD before showing text
 ------------------------------------------------*/

void lcd_init()
{
      lcd_write(0b00100111000);     // Set 2 line on LCD

      lcd_write(0b00100001100);     // Display On/Off control

      lcd_write(0b00100000110);     // Entry mode set

      lcd_write(0b00100000001);     // Clear screen
}

/*------------------------------------------------/
 Name:            lcd_printtext
 Description: print text or string on the screen
 ------------------------------------------------*/

void lcd_printtext(unsigned char string[])
{
      for (int i = 0; i < strlen(string); i++)
            lcd_data(string[i]);
}

/*------------------------------------------------/
 Name:            lcd_setcursor
 Description: set cursor position on display
 ------------------------------------------------*/

void lcd_setcursor(char row, char col)
{
      int row_char = 0;
      if (row == 1) row_char = 64;
      lcd_cmd(0b00010000000 + row_char + col);
}
```

```c
/*-----------------------------------------------------/
 Name:            variables
 Description: declare variables for using timer
 ----------------------------------------------------*/
unsigned long lcd_state=1;
unsigned long HIGH, LOW, wait_time, wait;
unsigned long now, PWM_mark, PWM_state;
unsigned long DC;

/*-----------------------------------------------------/
 Name:            string char
 Description: declare strings printing on LCD
 ----------------------------------------------------*/

unsigned char hello[]  = "Hello World !!!";
unsigned char empty[] = "                 ";
unsigned char paraPWM[] = "f: 1KHz DC:    %";

/*-----------------------------------------------------/
 Name:            myusleep
 Description: set delay time without affecting
                   other operations
 ----------------------------------------------------*/

void myusleep()
{
      wait = alt_timestamp();
      while (alt_timestamp() - wait < 5000) create_PWM();
}

/*#####################################################
                         |PWM|
####################################################*/

/*-----------------------------------------------------/
 Name:            update_PWM
 Description: Update on time and off time for PWM
              pulse depending on duty cycle change
 ----------------------------------------------------*/

void update_PWM()
{
/* Using DE-10 kit with frequency 50MHz --> 1 clock cycle corresponds
   20 nanosecond.
 * So, when applying required frequency 1kHz --> 1 required clock
   cycle corresponds 1 millisecond.
 * => 1 millisecond of 1 required clock cycle corresponds 50000 clock
      cycle on DE-10 kit.
 * => From above, the number of clock cycle using for duty cycle be
      determined.
*/
      HIGH = 50000*DC/100;
      LOW = 50000 - HIGH;
}
```

```
/*-----------------------------------------------/
 Name:             pwm_init
 Description: Initialize the default values and set
              the time start & time control for PWM
 ------------------------------------------------*/

void pwm_init()
{
      DC = 50;
      update_PWM();
      PWM_state = 0;
      wait_time = LOW;
      alt_timestamp_start();
      PWM_mark = alt_timestamp();
}

/*-----------------------------------------------/
 Name:             create_PWM
 Description: create PWM pulse to L298_H_bridge
 ------------------------------------------------*/

void create_PWM()
{
        now = alt_timestamp();
        if (now - PWM_mark  >= wait_time)
        {
              PWM_state = !PWM_state;

        if (PWM_state == 0) wait_time = LOW;
        else                wait_time = HIGH;

              PWM_mark = alt_timestamp();
        }
}

/*-----------------------------------------------/
 Name:             displace_PWM
 Description: display frequency and duty cycle
              controlling the motor on LCD
 ------------------------------------------------*/

void display_PWM()
{
      lcd_setcursor(1,0);
      lcd_printtext(paraPWM);
      lcd_setcursor(1,12);

      unsigned long num = DC;
      unsigned long a = num/100;     // Split to find and print
hundreds
      if (a==0) lcd_printtext(" ");
      else lcd_data(a + 0x30);
      lcd_setcursor(1,13);
      num = num - a*100;       // Update number to find and print next
      a = num/10;              // Split to find and print tens
      lcd_data(a+0x30);
      num = num - a*10;        // Update number to find and print next
      a = num/1;               // Split to find and print units
      lcd_data(a+0x30);
}
```

```
/*-------------------------------------------------/
 Name:              MAIN PROGRAM
 ------------------------------------------------*/

int main()
{
        pwm_init();
        lcd_init();

        while(1)
  {

/*-----------------------------------------------------------------/
Operation:                      SWITCH 0 IS ON
Description: LCD blinks the sentence "Hello World !!!" in the middle
             of row 1 with frequency 1Hz
-----------------------------------------------------------------*/

if ((IORD_ALTERA_AVALON_PIO_DATA(SWITCH_BASE)&1)== 0X01)
        {
              lcd_setcursor(0,1);
              lcd_printtext(hello);       // Print "Hello World!!!"
        }
else
        {
              lcd_state = 1;
              lcd_setcursor(0,0);
              lcd_printtext(empty);      // Clear 1st line if SW0 is
OFF
        }

/*-----------------------------------------------------------------/
Operation:        SWITCH 1 OR SWITCH 2 OR SWITCH 3 IS ON
Description: Create PWM pulses to control DC motor based on duty
             cycle with 100%, 50% and 25%.
-----------------------------------------------------------------*/

if (((IORD_ALTERA_AVALON_PIO_DATA(SWITCH_BASE) >> 1) & 1) == 1)
        {
              DC = 50;
              update_PWM();
              create_PWM();
              IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_BASE, PWM_state);
              display_PWM();
        }
else if (((IORD_ALTERA_AVALON_PIO_DATA(SWITCH_BASE) >> 2) & 1) == 1)
        {
              DC = 100;
              update_PWM();
              create_PWM();
              IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_BASE, PWM_state);
              display_PWM();
        }
else if (((IORD_ALTERA_AVALON_PIO_DATA(SWITCH_BASE) >> 3) & 1) == 1)
        {
              DC = 25;
              update_PWM();
              create_PWM();
              IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_BASE, PWM_state);
```

```
                    display_PWM();
            }
else
            {
                    IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_BASE, 0);
                    lcd_setcursor(1,0);
                     // Clear 1st line if SW(1||2||3) is OFF
                    lcd_printtext(empty);
            }
   }
}
```

9. Reference

- Datasheet of LCD 16x2
- An instruction of LCD interference
- L298_H_bridge datasheet

https://docs.onion.io/omega2-maker-kit/maker-kit-servo-h-bridge.html

https://www.engineersgarage.com/dc-motor-control-using-h-bridge/