Amrita Vishwa Vidyapeetham

Amrita School of Engineering, Bengaluru

Department of Electronics and Communication Engineering

15ECE382/Microcontroller Lab

Design Project Report

Submitted by

BL.EN.U4ECE18138 - Sathyasri S

BL.EN.U4ECE18139 – Satti Kaushik Reddy

BL.EN.U4ECE18140 – Shariq Akhtar P P

Faculty Incharge

Ms. Jayashree M. Oli	Dr. Parmasivam C.
Assistant Professor , Electronics and Communication Engg.	Assistant Professor , Electonics and Communication Engg.

AIM: Develop an ARM7 TDMI instructions-based Assembly Language Program sub-routine to compute complex real value division.

Software Used : Keil µVision

Program:

```
AREA HELLO,CODE,READONLY
        ENTRY
        LDR R0,=0X40000000 ; starting address
        LDR R1,=0X4000001C ; storing the results
        LDMIA R0!,{R2-R5}   ; load a & b from a+ib in R2 & R3, and c & d from c+id in R4 & R5
        MUL R6,R2,R4        ; product - AC
        MUL R7,R3,R5        ; product - BD
        MUL R8,R2,R5        ; product - DA
        MUL R9,R3,R4        ; product - BC
        MUL R10,R4,R4       ; product - CC
        MUL R11,R5,R5       ; product - DD
        ADD R6,R6,R7        ; sum of real parts of the numerator - ac+bd
        SUB R8,R9,R8        ; difference of img parts of the numerator - bc-da
        ADD R9,R10,R11      ; sum of the denominator terms
        STMIA R1!,{R6,R8,R9}; store the real,img part of numerator & denominator in memory
        MOV R3,#02          ; to compute division for both real and img terms
        LDR R0,=0X4000001C ; Base register address
        LDR R12,=0X4000002C ; result
        LDR R2,[R0]         ;real term dividend value to R2
        LDR R4,[R0]         ;also store it in R4 for the case where fractional part is involved
        LDR R1,[R0,#08]!    ;divisor value to R1
        LDR R0,=0X40000048 ;load flag address in memory
INI
        MOV R11,#01
        CMP R2,#00
        MVNMI R2,R2         ;if dividend is negative, perform 1's complement for R2 and R4
        MVNMI R4,R4
        ADDMI R2,#1         ;and add 1 to R2 and R4 to get the twos complement in that case
        ADDMI R4,#1
        STRMI R11,[R0]      ;store flag=01 if dividend is negative
        CMP R1,#00
        MVNMI R1,R1         ;if divisor is negative, perform 1's complement for R1
        ADDMI R1,#1         ;add 1 to get 2's complement
        STRMI R11,[R0,#04]  ;store flag=01 if divisor is negative
        MOV R7,#00          ;reset integer part of quotient
        MOV R10,#00         ;reset fractional part of quotient
        MOV R5,#00          ;reset a total number of digits counter
        MOV R6,#00          ;Quotient
        MOV R8,#0x0A        ;R8=10
        TST R1,R1
        BEQ ZER             ;if divisor is 0, branch to ZER
DIV
        SUBS R2,R2,R1       ;Division happens through multiple subtractions
        ADDCS R6,R6,#01     ;To keep the carry count (to get quotient)
        BCS DIV             ;loop if carry is set
```

```
        ADDS R2,R2,R1           ;This is to correct the remainder by adding R2 with R1
        BNE DEC                 ;if remainder is not 0 branch to DEC
OUT
        MOV R11,R5              ;R11=R5
        MOV R1,#0x0A            ;r1=10
POW
        CMP R11,#00             ;check if r5=0
        MULNE R2,R8,R1          ;R2=10*R1 (to get 10^R5)
        MOVNE R1,R2             ;R1=R2
        SUBNE R11,R11,#01
        BNE POW
        CMP R5,#00
        MOVEQ R7,R6             ;if r5=0 then move r6 to r7
        CMP R7,R6               ;if quotient and integer part of quotient are same
        MULNE R9,R7,R8
        SUBNE R10,R6,R9         ;fractional part of quotient=R6-(R7*(10^R5))
        CMP R7,#00              ;if integer part of quotient is zero (dividend<divisor)
        MOVEQ R10,R6            ;fractional part of quotient=r6
        LDR R1,[R0],#04         ;load negative flag corresponding to dividend from memory
        CMP R1,#01
        MVNEQ R7,R7             ;if flag=01 perform 1's complement
        ADDEQ R7,#1             ;to get 2's complement, to reversing sign
        LDR R1,[R0]             ;load negative flag corresponding to divisor
        CMP R1,#01
        MVNEQ R7,R7             ;if flag=01 perform 1's complement
        ADDEQ R7,#1             ;to get 2's complement, reversing sign
EQU     STMIA R12!,{R5,R7,R10} ;store number of digits,integer part of quotient,fractional part of
                                ; quotient respectively
        SUB R3,R3,#01
        CMP R3,#01              ; so that img term calculation also takes place
        BNE UP
        LDR R0,=0X40000020 ; giving the img term as the new address
        LDR R2,[R0]             ;load img dividend onto R2
        LDR R4,[R0],#04         ;load it onto R4 as well
        LDR R1,[R0]             ;load divisor onto R1
        LDR R0,=0X40000050 ;load flag address in memory for img part
        B INI
DEC
        CMP R5,#00
        MOVEQ R7,R6             ;move integer part of quotient to r7 when r5 is 0
        MUL R2,R4,R8            ;multiply dividend by 10 and move result to R2
        MOV R4,R2               ;move the new dividend to R4
        ADD R5,R5,#01           ;increment digits counter
        CMP R5,#04              ;(upto 3 decimal accuracy)
        MOVNE R6,#00            ;reset quotient to 0 if div loop has to iterate again
        BNE DIV                 ;run division loop again if R5-3 is not
        B OUT                   ;if R5-3=0 stop dividing and proceed to display output
ZER
        ADDEQ R7,#0XFFFFFFFF;if divisor=0 store this as R7
        B EQU
UP  B UP
```

END

[1] – It does not display the output correctly if the fractional part has leading zeros . For e.g. if the value is 1.00389 then the fractional part is 00389 – as there are leading zeros then it causes a problem . We have to infer from R5 about this then .

Output :

Condition A : When the output is without any decimal points

$$\frac{A+iB}{C+iD} = \frac{2+i4}{1+i1} = 3 + \mathrm{i}$$

Register Contents :

Before Execution		After Execution	
Register	**Value**	**Register**	**Values**
R0	0X00000000	**R0**	**0X40000024**
R1	0X00000000	**R1**	**0X00000005**
R2	0X00000000	**R2**	0X00000000
R3	0X00000000	**R3**	0X00000000
R4	0X00000000	**R4**	**0X00000001**
R5	0X00000000	**R5**	**0X00000002**
R6	0X00000000	**R6**	0X00000000
R7	0X00000000	**R7**	**0X00000008**
R8	0X00000000	**R8**	0X00000000
R9	0X00000000	**R9**	**0X00000005**
R10	0X00000000	**R10**	**0X00000001**
R11	0X00000000	**R11**	**0X00000004**
R12	0X00000000	**R12**	**0X4000003C**
R13(SP)	0X00000000	**R13(SP)**	0X00000000
R14(LR)	0X00000000	**R14(LR)**	0X00000000
R15(PC)	0X00000000	**R15(PC)**	**0X000000a4**
CPSR	0X000000d3	**CPSR**	**0X800000d3**
N	0	**N**	**1**
Z	0	**Z**	0
C	0	**C**	0
V	0	**V**	0
I	1	**I**	1
F	1	**F**	1
T	0	**T**	0
M	0X13	**M**	0X13

Memory Location Contents :

Before Execution		After Execution	
Memory Address	**Values**	**Memory Address**	**Values**
0X40000000	**02 00 00 00**	**0X40000000**	**02 00 00 00**
0X40000004	**04 00 00 00**	**0X40000004**	**04 00 00 00**
0X40000008	**01 00 00 00**	**0X40000008**	**01 00 00 00**
0X4000000C	**01 00 00 00**	**0X4000000C**	**01 00 00 00**
0X40000010	00 00 00 00	**0X40000010**	00 00 00 00
0X40000014	00 00 00 00	**0X40000014**	00 00 00 00

0X40000018	00 00 00 00	**0X40000018**	00 00 00 00
0X4000001C	00 00 00 00	**0X4000001C**	**06 00 00 00**
0X40000020	00 00 00 00	**0X40000020**	**02 00 00 00**
0X40000024	00 00 00 00	**0X40000024**	**02 00 00 00**
0X40000028	00 00 00 00	**0X40000028**	00 00 00 00
0X4000002C	00 00 00 00	**0X4000002C**	**00 00 00 00**
0X40000030	00 00 00 00	**0X40000030**	**03 00 00 00**
0X40000034	00 00 00 00	**0X40000034**	**00 00 00 00**
0X40000038	00 00 00 00	**0X40000038**	**00 00 00 00**
0X4000003C	00 00 00 00	**0X4000003C**	**01 00 00 00**
0X40000040	00 00 00 00	**0X40000040**	**00 00 00 00**
0X40000044	00 00 00 00	**0X40000044**	00 00 00 00
0X40000048	00 00 00 00	**0X40000048**	**00 00 00 00**
0X4000004C	00 00 00 00	**0X4000004C**	**00 00 00 00**
0X40000050	00 00 00 00	**0X40000050**	**00 00 00 00**
0X40000054	00 00 00 00	**0X40000054**	**00 00 00 00**

Condition B : When the output contains only the real term and no imaginary term

$$\frac{A+iB}{C+iD} = \frac{2+i4}{1+i2} = 2$$

Register Contents :

Before Execution		After Execution	
Register	**Value**	**Register**	**Values**
R0	0X00000000	**R0**	**0X40000054**
R1	0X00000000	**R1**	0X00000000
R2	0X00000000	**R2**	0X00000000
R3	0X00000000	**R3**	0X00000000
R4	0X00000000	**R4**	0X00000000
R5	0X00000000	**R5**	0X00000000
R6	0X00000000	**R6**	0X00000000
R7	0X00000000	**R7**	0X00000000
R8	0X00000000	**R8**	**0X0000000a**
R9	0X00000000	**R9**	**0X00000005**
R10	0X00000000	**R10**	0X00000000
R11	0X00000000	**R11**	0X00000000
R12	0X00000000	**R12**	**0X40000044**
R13(SP)	0X00000000	**R13(SP)**	0X00000000
R14(LR)	0X00000000	**R14(LR)**	0X00000000
R15(PC)	0X00000000	**R15(PC)**	**0X00000158**
CPSR	0X000000d3	**CPSR**	**0X800000d3**
N	0	**N**	**1**
Z	0	**Z**	0
C	0	**C**	0
V	0	**V**	0
I	1	**I**	1
F	1	**F**	1
T	0	**T**	0
M	0X13	**M**	0X13

Memory Location Contents :

Before Execution		After Execution	
Memory Address	**Values**	**Memory Address**	**Values**
0X40000000	**02 00 00 00**	**0X40000000**	**02 00 00 00**
0X40000004	**04 00 00 00**	**0X40000004**	**04 00 00 00**
0X40000008	**01 00 00 00**	**0X40000008**	**01 00 00 00**
0X4000000C	**02 00 00 00**	**0X4000000C**	**02 00 00 00**
0X40000010	00 00 00 00	**0X40000010**	00 00 00 00
0X40000014	00 00 00 00	**0X40000014**	00 00 00 00
0X40000018	00 00 00 00	**0X40000018**	00 00 00 00
0X4000001C	00 00 00 00	**0X4000001C**	**0A 00 00 00**
0X40000020	00 00 00 00	**0X40000020**	**00 00 00 00**
0X40000024	00 00 00 00	**0X40000024**	**05 00 00 00**
0X40000028	00 00 00 00	**0X40000028**	00 00 00 00
0X4000002C	00 00 00 00	**0X4000002C**	**00 00 00 00**
0X40000030	00 00 00 00	**0X40000030**	**02 00 00 00**
0X40000034	00 00 00 00	**0X40000034**	**00 00 00 00**
0X40000038	00 00 00 00	**0X40000038**	**00 00 00 00**
0X4000003C	00 00 00 00	**0X4000003C**	**00 00 00 00**
0X40000040	00 00 00 00	**0X40000040**	**00 00 00 00**
0X40000044	00 00 00 00	**0X40000044**	00 00 00 00
0X40000048	00 00 00 00	**0X40000048**	**00 00 00 00**
0X4000004C	00 00 00 00	**0X4000004C**	**00 00 00 00**
0X40000050	00 00 00 00	**0X40000050**	**00 00 00 00**
0X40000054	00 00 00 00	**0X40000054**	**00 00 00 00**

Condition C : When the output contains both real and imaginary terms with fractional parts

$$\frac{A+iB}{C+iD} = \frac{9+i6}{1+i1} = 7.5 - 1.5i$$

Register Contents :

Before Execution		After Execution	
Register	**Value**	**Register**	**Values**
R0	0X00000000	**R0**	**0X40000054**
R1	0X00000000	**R1**	**0X00000000**
R2	0X00000000	**R2**	**0X00000064**
R3	0X00000000	**R3**	**0X00000000**
R4	0X00000000	**R4**	**0X0000001e**
R5	0X00000000	**R5**	**0X00000001**
R6	0X00000000	**R6**	**0X0000000f**
R7	0X00000000	**R7**	**0Xffffffff**
R8	0X00000000	**R8**	**0X0000000a**
R9	0X00000000	**R9**	**0X0000000a**
R10	0X00000000	**R10**	**0X00000005**
R11	0X00000000	**R11**	0X00000000
R12	0X00000000	**R12**	**0X40000044**
R13(SP)	0X00000000	**R13(SP)**	0X00000000
R14(LR)	0X00000000	**R14(LR)**	0X00000000
R15(PC)	0X00000000	**R15(PC)**	**0X00000158**

CPSR	0X000000d3	CPSR	0X800000d3
N	0	**N**	**1**
Z	0	**Z**	0
C	0	**C**	0
V	0	**V**	0
I	1	**I**	1
F	1	**F**	1
T	0	**T**	0
M	0X13	**M**	0X13

Memory Location Contents :

Before Execution		After Execution	
Memory Address	**Values**	**Memory Address**	**Values**
0X40000000	**02 00 00 00**	**0X40000000**	**09 00 00 00**
0X40000004	**04 00 00 00**	**0X40000004**	**06 00 00 00**
0X40000008	**01 00 00 00**	**0X40000008**	**01 00 00 00**
0X4000000C	**02 00 00 00**	**0X4000000C**	**01 00 00 00**
0X40000010	00 00 00 00	**0X40000010**	00 00 00 00
0X40000014	00 00 00 00	**0X40000014**	00 00 00 00
0X40000018	00 00 00 00	**0X40000018**	00 00 00 00
0X4000001C	00 00 00 00	**0X4000001C**	**0F 00 00 00**
0X40000020	00 00 00 00	**0X40000020**	**FD FF FF FF**
0X40000024	00 00 00 00	**0X40000024**	**02 00 00 00**
0X40000028	00 00 00 00	**0X40000028**	00 00 00 00
0X4000002C	00 00 00 00	**0X4000002C**	**01 00 00 00**
0X40000030	00 00 00 00	**0X40000030**	**07 00 00 00**
0X40000034	00 00 00 00	**0X40000034**	**05 00 00 00**
0X40000038	00 00 00 00	**0X40000038**	**01 00 00 00**
0X4000003C	00 00 00 00	**0X4000003C**	**FF FF FF FF**
0X40000040	00 00 00 00	**0X40000040**	**05 00 00 00**
0X40000044	00 00 00 00	**0X40000044**	00 00 00 00
0X40000048	00 00 00 00	**0X40000048**	**00 00 00 00**
0X4000004C	00 00 00 00	**0X4000004C**	**00 00 00 00**
0X40000050	00 00 00 00	**0X40000050**	**01 00 00 00**
0X40000054	00 00 00 00	**0X40000054**	**00 00 00 00**

Those 3 rows say that – the real value is 7 which has a one decimal point with the value 5 . So the number is 7.5

Similarly here – the real value is -1 which has one decimal point with the value 5 . So the number is -1.5

Result: Division of two complex numbers using Assembly Language program was done successfully and the output was verified.

AIM: Develop C-program to compute complex real value division targeting LPC2148 devices and display the results using the LCD.

Software Used : Keil µVision and Proteus Design Suite 8.0

C program :

```
//PINSEL0 controls PORT0 pins P0.0 to P0.15, PINSEL1 controls PORT0 pins P0.16 to
P0.31 and PINSEL2 controls PORT1 pins P1.16 to P1.31.
//IOxSET - To set an output configured pin
//IOxCLR - To clear an output configured pin
//IOxPIN - To get logic value on a I/O pin
//IOxDIR - To select input /output function (by placing 0/1) for an I/O pin
//ldc is 16X2 i.e. can display 16 characters 2 lines at a time

#include<lpc21xx.h>
#include<string.h>
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
void delay(void);
void lcd(char,int);
void lcd_display(char s[]);
void waitforkeypress(void);
void enterchar(char s[]);
char keypad(void);
char k;                              //pressed keys are assigned to this variable
struct complex
{
  double real, img;
};
int main(void)
{
   double n1,n2,n3;
   double out_real,out_img,freal,fimg,ireal,iimg;
   struct complex s1,s2;
   char ch[]="1)a+ib 2)c+id";
   char ch2[]="Enter a:";
   char ch3[]="Enter b:";
   char ch4[]="Enter c:";
   char ch5[]="Enter d:";
   char ch6[]="Output:";
   char err[]="ERROR: 0/0 div!";
   char test='T';
   char a[6];
   char b[6];
   char c[6];
   char d[6];
```

```
char rreal[16];
char rimg[6];
PINSEL0=PINSEL2=0;              //configure pins as GPIO
IODIR0=0x000003ff;              //pins P0.0 to 9
IODIR1=0x00f80000;              //pins P1.19 to 23, P1.16 to 18 is set as 0 hence I/P pins
lcd(0x38,0);                    //0x38 is lcd command to initialise lcd in 8 bit mode
lcd(0x0f,0);                    //0x0f is command for - display on,cursor blinking
while(1)
{       a[0]='\0';              //to clear/empty the strings
        b[0]='\0';
        c[0]='\0';
        d[0]='\0';
        rreal[0]='\0';
        rimg[0]='\0';
        n1=0;                   //resetting values for current iteration
        n2=0;
        n3=0;
        out_real=0;
        out_img=0;
        lcd_display(ch);        //display string ch on lcd
        waitforkeypress();      //wait until key is pressed
        lcd(0x01,0);            //LCD cmd to clear screen
        lcd_display(ch2);       //display string ch2 on lcd
        lcd(0x0c0,0);           //lcd cmd to force cursor to 2nd line
        enterchar(a);           //enter characters for string a
        lcd(0x01,0);
        lcd_display(ch3);       //display string ch3 on lcd
        lcd(0x0c0,0);
        enterchar(b);           //enter characters for string b
        lcd(0x01,0);
        lcd_display(ch4);       //display string ch4 on lcd
        lcd(0x0c0,0);
        enterchar(c);           //enter characters for string c
        lcd(0x01,0);
        lcd_display(ch5);       //display string ch5 on lcd
        lcd(0x0c0,0);
        enterchar(d);           //enter characters for string d
        lcd(0x01,0);
        //code to check if string arrays are inititalised properly by printing input
        lcd_display(a);         //display string a on lcd
        lcd(0x0c0,0);
        lcd_display(b);         //display string b on lcd
        waitforkeypress();
        lcd(0x01,0);
        lcd_display(c);         //display string c on lcd
        lcd(0x0c0,0);
```

```
lcd_display(d);              //display string d on lcd
waitforkeypress();
lcd(0x01,0);
//code to divide 2 complex numbers-
s1.real=strtod(a,NULL);
//convert string a to double s1.real using strtod , 2nd arg, endptr=NULL
s1.img=strtod(b,NULL);
s2.real=strtod(c,NULL);
s2.img=strtod(d,NULL);
if (s2.real==0 && s2.img==0)
        lcd_display(err);//show error msg if c and d both are 0
else
{
        n1 = s1.real*s2.real + s1.img*s2.img;
        n2 = s1.img*s2.real - s1.real*s2.img;
        n3 = s2.real*s2.real + s2.img*s2.img;
        out_real=n1/n3;
        out_img=n2/n3;
        freal=modf(out_real,&ireal);
        //splits the integer part and fractional part of out_real into variables
ireal and freal respectively
        fimg=modf(out_img,&iimg);
        //splits the integer part and fractional part of out_img into variables
iimg and fimg respectively
        if(out_real<0 && out_img<0)//if both real and img terms are negative
        {
        sprintf(rreal,"-%d.%02u\0",(int)ireal,(int)fabs(freal*100));
        sprintf(rimg,"-%d.%02u\0",(int)iimg,(int)fabs(fimg*100));
        }
        else if(out_real<0)//if real term is negative
        {
        sprintf(rreal,"-%d.%02u\0",(int)ireal,(int)fabs(freal*100));
        sprintf(rimg,"%d.%02u\0",(int)iimg,(int)fabs(fimg*100));
        }
        else if(out_img<0)//if img term is negative
        {
        sprintf(rreal,"%d.%02u\0",(int)ireal,(int)fabs(freal*100));
        sprintf(rimg,"-%d.%02u\0",(int)iimg,(int)fabs(fimg*100));
        }
        Else            //if both real and img terms are positive
        {
        sprintf(rreal,"%d.%02u\0",(int)ireal,(int)fabs(freal*100));
        sprintf(rimg,"%d.%02u\0",(int)iimg,(int)fabs(fimg*100));
        }
        lcd(test,1);
        //print test character 'T' to check if code has executed uptil here
```

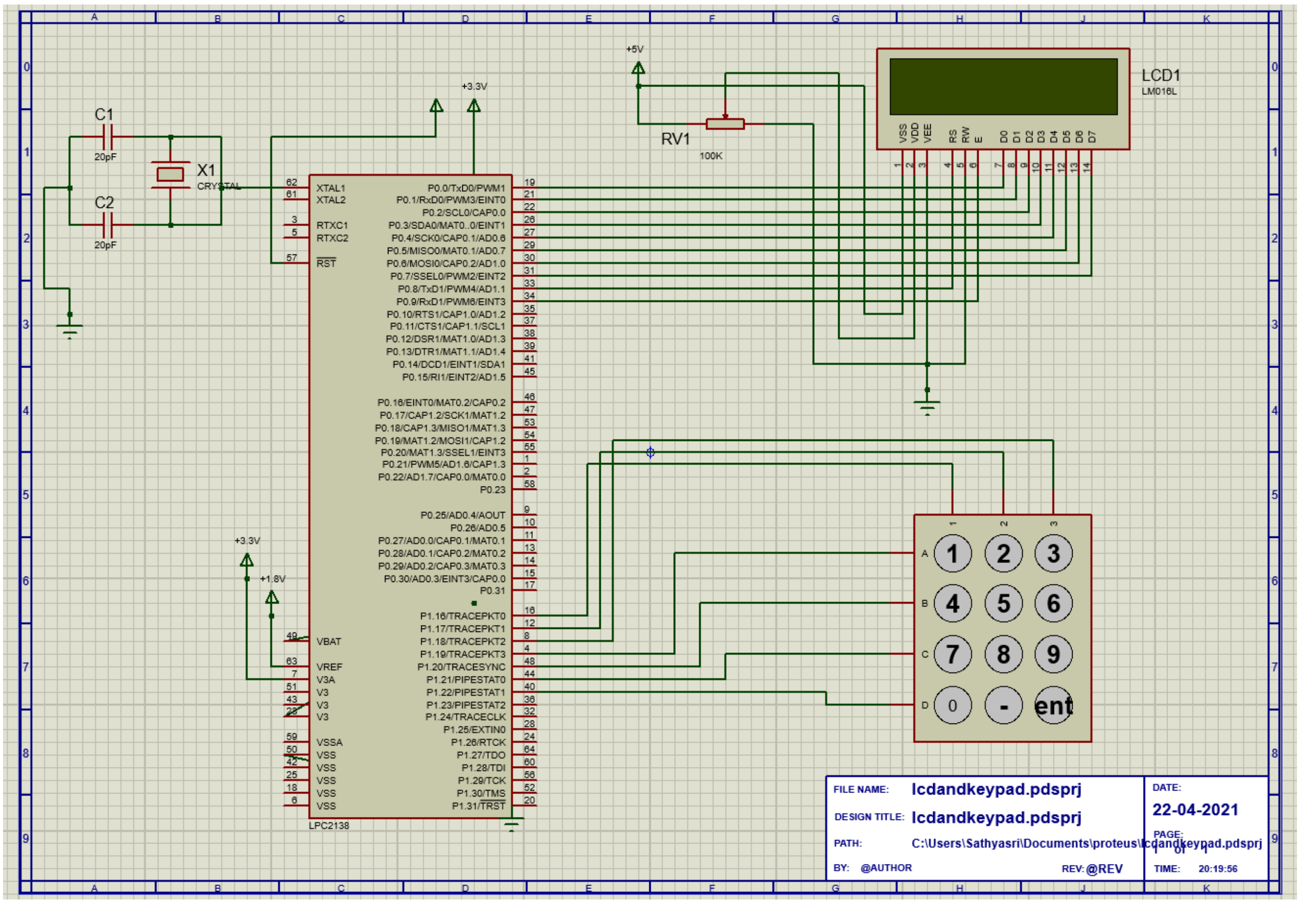
```
                        waitforkeypress();
                        lcd(0x01,0);
                        strcat(rreal,"+i(");
                        strcat(rreal,rimg);
                        strcat(rreal,")\0");
                        lcd_display(ch6);       //display string ch6 on lcd
                        lcd(0x0c0,0);
                        lcd_display(rreal);     //display string rreal on lcd (final result)
                }
                waitforkeypress();
                lcd(0x01,0);
  }
}
void lcd(char a,int b)          //LCD Subroutine
 {
   IOSET0=a<<0;
   IOSET0=b<<8;
//P0.8 is connected to Register select RS, when set to 1, displays data output, when set to 0,
treats input as command
   IOSET0=1<<9;                 //P0.9 is connected to Lcd Enable
   delay();
   IOCLR0=1<<9;
   IOCLR0=b<<8;
   IOCLR0=a<<0;
 }
void lcd_display(char s[])      //to display string on LCD
{
        int i=0;
        for(i=0;s[i]!='\0';i++)
                lcd(s[i],1);
}
void enterchar(char s[])        //to append characters entered through keypad into string
{
        int i=0;
        while(1)
    {
                k=keypad();             //Obtaining values from keypad
                if(k=='E')
                        {
                        s[i]='\0';
                        break;
                        }
                        s[i]=k;
                        i++;
                        lcd(k,1);
    }
```

```
}
char keypad(void)                          //Keypad Scan
 {
   while(1)
   {
     IOCLR1|=(1<<19);               //Making row1 LOW          (P1.19)
     IOSET1|=(1<<20)|(1<<21)|(1<<22)|(1<<23); //Making rest of the rows '1'
     if(!(IOPIN1&(1<<16)))              //Scan for key press (P1.16 is column 1)
// i guess if button is pressed then corresponding bit will be 0
      {
       while(!(IOPIN1&(1<<16)));
       return '1';                          //Returning value to display
      }
     if(!(IOPIN1&(1<<17)))
      {
        while(!(IOPIN1&(1<<17)));
        return '2';
      }
     if(!(IOPIN1&(1<<18)))
      {
        while(!(IOPIN1&(1<<18)));
        return '3';
      }
     IOCLR1|=(1<<20);
     IOSET1|=(1<<21)|(1<<22)|(1<<19)|(1<<23);
     if(!(IOPIN1&(1<<16)))
{
       while(!(IOPIN1&(1<<16)));
       return '4';
     }
     if(!(IOPIN1&(1<<17)))
{
       while(!(IOPIN1&(1<<17)));
       return '5';
    }
     if(!(IOPIN1&(1<<18)))
{
       while(!(IOPIN1&(1<<18)));
       return '6';
    }
     IOCLR1|=(1<<21);
     IOSET1|=(1<<22)|(1<<20)|(1<<19)|(1<<23);
     if(!(IOPIN1&(1<<16)))
{
       while(!(IOPIN1&(1<<16)));
       return '7';
```

```
      }
      if(!(IOPIN1&(1<<17)))
{
        while(!(IOPIN1&(1<<17)));
        return '8';
    }
      if(!(IOPIN1&(1<<18)))
{
        while(!(IOPIN1&(1<<18)));
        return '9';
}

      IOCLR1|=(1<<22);
      IOSET1|=(1<<19)|(1<<20)|(1<<21)|(1<<23);
      if(!(IOPIN1&(1<<16)))
{
        while(!(IOPIN1&(1<<16)));
        return '0';
}
      if(!(IOPIN1&(1<<17)))
{
        while(!(IOPIN1&(1<<17)));
        return '-';
}
      if(!(IOPIN1&(1<<18)))
{
        while(!(IOPIN1&(1<<18)));
        return 'E';
}
 }
}
void waitforkeypress(void)
{
                  while(1)
            //wait for keypress to clear screen and display next line
     {
                  k=keypad();                    //Obtain any value from keypad
                          break;
     }
}
void delay(void)                            //Delay loop
{
  unsigned int i;
  for(i=0;i<=20000;i++);
}
```

Circuit Diagram :



Result: Division of two complex numbers using C program and linking it with the LPC2138 and LCD was done successfully and the output was verified.