APPENDIX A

PIN DETAILS OF THE PIC24FJ256 CONTROLLER

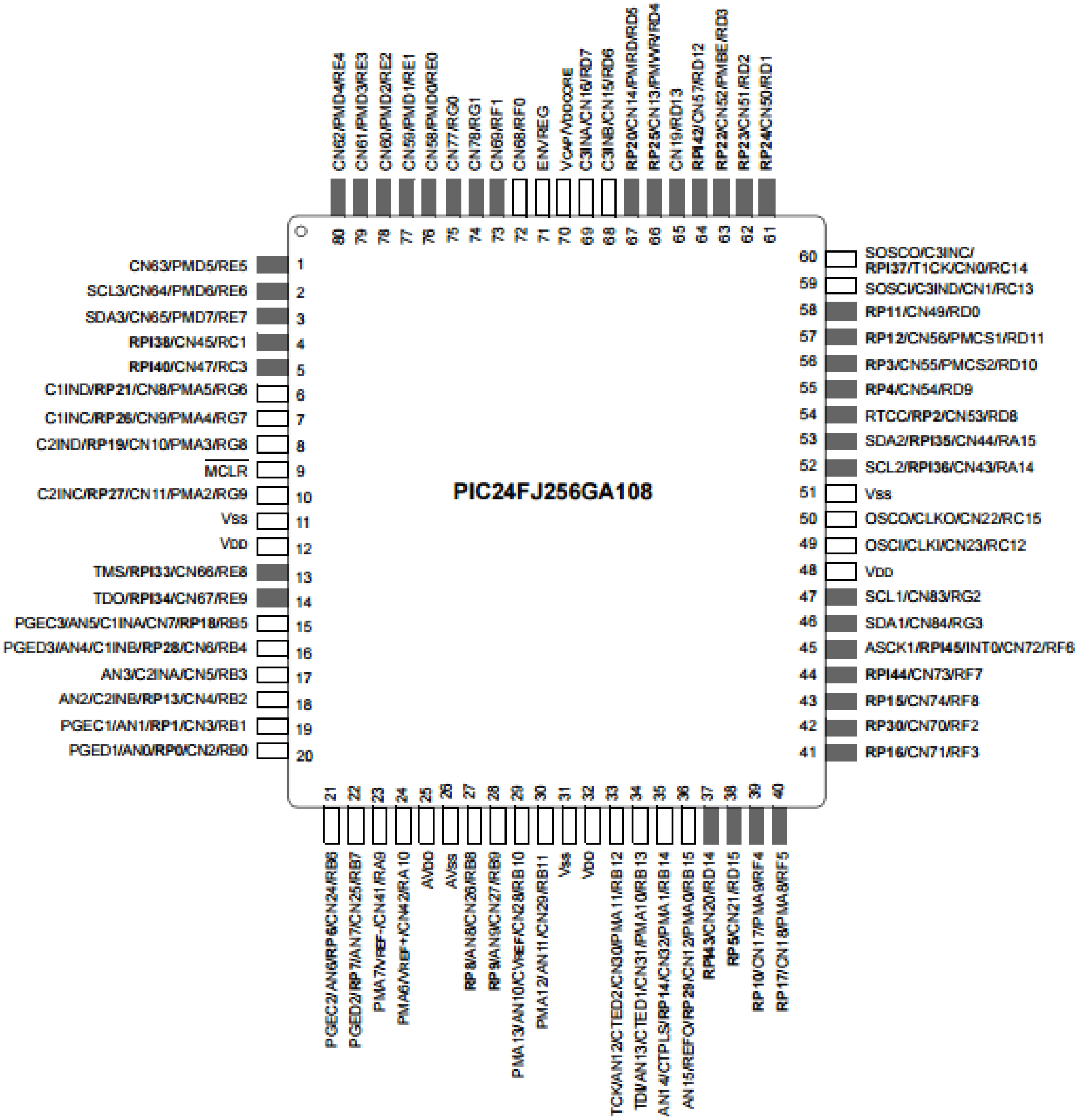

Fig 43:PIC24FJ256 Microcontroller

The PIC24FJ256GA108 is a 16 bit microcontroller and is available in a wide variety of pin configurations, ranging right from a 64 pin lay out to a 100 pin layout. For this project an 80 pin layout, which is most common, is being used. It operates on a 3.3V logic which implies that a high signal is issued to/from these controllers at a DC voltage level of about 3.3V. However, this can vary between 2.9 to 3.5V. The controller has a maximum current sink/source capacity of 18mA and hence can be used for actuating small relays directly. An external crystal of 10MHz is used, however instructions can be clocked at 40MHz using the concept of PLL as mentioned earlier. The analog to digital converter has a resolution of 12 bits and hence is quite accurate in measuring parameters.

APPENDIX B

PIC - PROGRAM

The program for the discussed algorithm is written in the C language and is as given below

```
#include "p24HJ256GP206.h"

#define FCY 40000000UL

#include <libpic30.h>

#include <math.h>

//// External Oscillator

//_FOSCSEL(FNOSC_PRIPLL);                                            //
Primary (XT, HS, EC) Oscillator with PLL

//_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF  & POSCMD_XT);      // Clock Switching
and Fail Safe Clock Monitor is disabled..

//
                              // ..OSC2 Pin Function: OSC2 is Clock Output..

//
                              // ..Primary Oscillator Mode: XT Crystanl

//

//_FWDT(FWDTEN_OFF);                                    // Watchdog Timer
Enabled/disabled by user software..

//                                                                   // ..
(LPRC can be disabled by clearing SWDTEN bit in RCON register

////_FPOR(PWRTEN_OFF);                                               //
Turn off the power-up timers.

//_FGS(GCP_OFF);                                        // Disable Code
Protection

#define          LCD_RS_DIR              TRISFbits.TRISF6

#define          LCD_RW_DIR                        TRISGbits.TRISG3

#define          LCD_EN_DIR              TRISGbits.TRISG2
```

```
#define         LCD_DATA0_DIR           TRISCbits.TRISC13

#define         LCD_DATA1_DIR           TRISCbits.TRISC14

#define         LCD_DATA2_DIR           TRISFbits.TRISF0

#define         LCD_DATA3_DIR           TRISFbits.TRISF1

#define         LCD_DATA4_DIR           TRISGbits.TRISG1

#define         LCD_DATA5_DIR           TRISGbits.TRISG0

#define         LCD_DATA6_DIR           TRISGbits.TRISG12

#define         LCD_DATA7_DIR           TRISGbits.TRISG13

#define         LCD_RS_DATA             LATFbits.LATF6

#define         LCD_RW_DATA             LATGbits.LATG3

#define         LCD_EN_DATA             LATGbits.LATG2

#define         LCD_DATA0_DATA          LATCbits.LATC13

#define         LCD_DATA1_DATA          LATCbits.LATC14

#define         LCD_DATA2_DATA          LATFbits.LATF0

#define         LCD_DATA3_DATA          LATFbits.LATF1

#define         LCD_DATA4_DATA          LATGbits.LATG1

#define         LCD_DATA5_DATA          LATGbits.LATG0

#define         LCD_DATA6_DATA          LATGbits.LATG12

#define         LCD_DATA7_DATA          LATGbits.LATG13


#define         RELAY_GPIO1_DIR         TRISDbits.TRISD2

#define         RELAY_GPIO2_DIR         TRISDbits.TRISD3

#define         RELAY_GPIO3_DIR         TRISDbits.TRISD4

#define         RELAY_GPIO4_DIR         TRISDbits.TRISD5
```

```
#define             RELAY_GPIO1_DATA        LATDbits.LATD2
#define             RELAY_GPIO2_DATA        LATDbits.LATD3
#define             RELAY_GPIO3_DATA        LATDbits.LATD4
#define             RELAY_GPIO4_DATA        LATDbits.LATD5

void Init_Port(void);
void Init_Lcd(void);
void Lcd_String(unsigned char *data);
void Lcd_Data(unsigned char data);
void Delay_ms(intms);
void Lcd_Data1(unsigned char data);
float adcconv(int channel);
void _ISR _T1Interrupt(void);
void init_timer1(void);
void initAdc1(void);
void Init_Timer4(void);
void Lcd_Display(void);
void De_Init_Timer4(void);
unsigned int calc1 = 0,calc2 = 0;
float phase_shift_us = 0.0;
float Current = 0.0;
float Voltage = 0.0;
unsigned char ADCerror_flag=0;
float result[2];
float power_factor = 0.0;
```

```
int k = 0;

intfilter_count=0;

unsigned inttimer_count = 0;

void main()

{

   // Configure Oscillator to operate the device at 40Mhz

   // Fosc= Fin*M/(N1*N2), Fcy=Fosc/2

   // Fosc= 8M*40(2*2)=80Mhz for 8M input clock

                          PLLFBD=30;                                        // M=40

                          CLKDIVbits.PLLPOST=0;              // N1=2

                          CLKDIVbits.PLLPRE=0;               // N2=2

                          OSCTUN=0;                                         // Tune FRC oscillator, if
FRC is used

                          RCONbits.SWDTEN=0;           // Disable Watch Dog Timer

                          while(OSCCONbits.LOCK!=1) {};  // Wait for PLL to lock


   /******************************ADC **********************************/

                          TRISDbits.TRISD8  = 1;    // IC1

                          TRISDbits.TRISD9  = 1;    // IC2

                          init_timer1();

                          Init_Timer4();

                          initAdc1();

                          init_input_capture();

                          /******************************End of ADC
config************************/
```

```
Init_Port();

Init_Lcd();

Lcd_Command(0x01);

Lcd_String("APFC KVA^2");

while(timer_count<= 250)        {

}

De_Init_Timer4();

DelayNmSec(30000);

Lcd_Command(0x01);

Lcd_String("V");

Lcd_Command(0x86);

Lcd_String("I");

Lcd_Command(0x8C);

Lcd_String("PF");

Lcd_Command(0xC0);

while(1){

  ADC_Conv();

  Phase_Shift_Calc();

  Lcd_Display();

  if(power_factor< 0.85){

            filter_count++;

            if(filter_count> 5       ){

                      RELAY_GPIO1_DATA =1;

            }

            if(filter_count> 10){
```

```
                                        RELAY_GPIO2_DATA =1;

                                }

                                if(filter_count> 15){

                                        RELAY_GPIO3_DATA =1;

                                }

                                if(filter_count> 20){

                                        RELAY_GPIO4_DATA =1;

                                }

                        }else if(power_factor> 0.85){

                                filter_count=0;

                        }


                        Delay_ms(500);

                       }

}

void Init_Timer4(void)

{                      T4CON=0x8030;
                       PR4=0x0C35;                // 20 ms interrupt

                       IPC6bits.T4IP=0x01;               //priority 03
                       IFS1bits.T4IF=0;

                       IEC1bits.T4IE=1;                        //enable the timer interrupt


}

void De_Init_Timer4(void)

{

                       T4CON=0x0000;
```

```
                    IFS1bits.T4IF=0;
                    IEC1bits.T4IE=0;                    //enable the timer interrupt

}
void _ISR _T4Interrupt(void)
{
 TMR4 = 0x0000;
                    timer_count++;
                    if(timer_count>= 350)  {
                     timer_count = 0;
                    }
  IFS1bits.T4IF=0;              //clear the timer interrupt flag
}
void Init_Port(void){
                    //For control signals
                    LCD_RS_DIR  =0;
                    LCD_RW_DIR  =0;
                    LCD_EN_DIR  =0;
                    LCD_RS_DATA =0;
                    LCD_RW_DATA =0;
                    LCD_EN_DATA =0;
                    //for data
                    LCD_DATA0_DIR =0;
                    LCD_DATA1_DIR =0;
                    LCD_DATA2_DIR =0;
```

```
                LCD_DATA3_DIR =0;

                LCD_DATA4_DIR =0;

                LCD_DATA5_DIR =0;

                LCD_DATA6_DIR =0;

                LCD_DATA7_DIR =0;

                LCD_DATA0_DATA =0;

                LCD_DATA1_DATA =0;

                LCD_DATA2_DATA =0;

                LCD_DATA3_DATA =0;

                LCD_DATA4_DATA =0;

                LCD_DATA5_DATA =0;

                LCD_DATA6_DATA =0;

                LCD_DATA7_DATA =0;


                RELAY_GPIO1_DIR =0;

                RELAY_GPIO2_DIR =0;

                RELAY_GPIO3_DIR =0;

                RELAY_GPIO4_DIR =0;

                RELAY_GPIO1_DATA =0;

                RELAY_GPIO2_DATA =0;

               RELAY_GPIO3_DATA =0;

                RELAY_GPIO4_DATA =0;

}

void Init_Lcd(void){

                Lcd_Command(0x38);
```

```
			Delay_ms(5);
			Lcd_Command(0x0f);
			Delay_ms(5);
			Lcd_Command(0x80);
}
void Lcd_Command(unsigned char command){
			LCD_DATA0_DATA = command;
			LCD_DATA1_DATA = command >> 1;
			LCD_DATA2_DATA = command >> 2;
			LCD_DATA3_DATA = command >> 3;
			LCD_DATA4_DATA = command >> 4;
			LCD_DATA5_DATA = command >> 5;
			LCD_DATA6_DATA = command >> 6;
			LCD_DATA7_DATA = command >> 7;

			LCD_EN_DATA = 1;
			LCD_RS_DATA = 0;
			LCD_RW_DATA = 0;
			Delay_ms(1);
			LCD_EN_DATA = 0;
			Delay_ms(5);
}
void Lcd_Data(unsigned char data){
			//Move the data to the port
			LCD_DATA0_DATA = data;
```

```
                LCD_DATA1_DATA = data >> 1;
                LCD_DATA2_DATA = data >> 2;
                LCD_DATA3_DATA = data >> 3;
                LCD_DATA4_DATA = data >> 4;
                LCD_DATA5_DATA = data >> 5;
                LCD_DATA6_DATA = data >> 6;
                LCD_DATA7_DATA = data >> 7;
                LCD_EN_DATA = 1;
                LCD_RS_DATA = 1;
                LCD_RW_DATA = 0;
                Delay_ms(1);
                LCD_EN_DATA = 0;
}
void Lcd_String(unsigned char *data){
                int k=0;
                for(k=0;data[k] != 0;k++){
                  Lcd_Data(data[k]);
                }
}
void Delay_ms(intms){
__delay_ms(ms);
}
```

```
///////////////////////////////////Code for power monitoring///////////////////////////////////

float res = 0.0;

void Phase_Shift_Calc(void)  {


                    if(calc1 <= calc2)       {

                      phase_shift_us = ((calc2 - calc1)*0.025*64.0);

                    }else if (calc1 > calc2)  {

                      phase_shift_us = (((65535 - calc1) + calc2)*0.025*64.0);

                    }

                    res = ((phase_shift_us/5000.0)*90.0);

                    res = ((res*3.14)/180.0);

                    power_factor = cos(res);

}

void ADC_Conv(void)        {

                    result[0] =       adcconv(0);

                    result[1] =       adcconv(1);

                    //Current = ((((result[1]/4030.0)*3.3)/3.3)*20.0);

                    Current = (((result[1]/4030.0)*3.3)*10.0);

                    Voltage = ((result[0]/4030.0)*230.0);

}

void init_timer1(void)

{

                     T1CON=0x8030;

                     INTCON1bits.NSTDIS=0;

                     IPC0bits.T1IP=0x05;
```

```
                IFS0bits.T1IF=0;

                PR1=0x0064;

                TMR1=0x0000;

}

void _ISR _T1Interrupt(void)

{

                ADCerror_flag=1;

                TMR1=0x0000;

                IFS0bits.T1IF=0;

                IEC0bits.T1IE=0;

}

void initAdc1(void)

{

                AD1CON1bits.FORM = 0;          // Data Output Format: Signed
Fraction (Q15 format)

                AD1CON1bits.SSRC = 0;          // Sample Clock Source: GP Timer
starts conversion

                AD1CON1bits.ASAM = 0;          // ADC Sample Control: Sampling
begins immediately after conversion

                AD1CON1bits.AD12B = 1;         // 12-bit 1-channel operation

                AD1CON2bits.VCFG = 0;

                AD1CON3bits.ADRC=0;                    // ADC Clock is derived
from Systems Clock

                AD1CON3bits.SAMC=31;

                AD1CON3bits.ADCS = 63;         // ADC Conversion Clock
Tad=Tcy*(ADCS+1)= (1/40M)*64 = 1.6us (625Khz)

                                                       // ADC Conversion Time
```

```
for 12-bit Tc=14*Tad = 22.4us

                                                         // ADC Conversion Time
for 10-bit Tc=12*Tab = 19.2us

AD1CON4=0;                        // Allocate 1 words of buffer to each analog input

                                  // This register is not used in conversion order mode

                                  // This is required only in the scatter/gather mode

AD1CHS0bits.CH0SA=0;              // MUXA +ve input selection (AIN5) for CH0

AD1CHS0bits.CH0NA=0;              // MUXA -ve input selection (Vref-) for CH0

                  AD1PCFGL=0xFFFF;

                  AD1PCFGH=0xFFFF;

                  AD1PCFGLbits.PCFG0 = 0;     //AN0

                  AD1PCFGLbits.PCFG1 = 0;     //AN1

                  AD1CSSH = 0x0000;

                  AD1CSSL = 0x0000;           // Channel Scan is disabled, default state

                  IFS0bits.AD1IF = 0;                // Clear the A/D interrupt flag bit

                  IEC0bits.AD1IE = 0;                // Do Not Enable A/D interrupt

                  AD1CON1bits.SAMP = 0;              // 12-bit 1-channel operation

                  AD1CON1bits.ADON = 1;              // Turn on the A/D converter

}

float adcconv(int channel)

{

                  int j;

                  AD1CHS0bits.CH0SA = channel;

                  AD1CON1bits.ADON = 1;              // on the ADC module..

                  TMR1=0x0000;
```

```
                    IFS0bits.T1IF=0;

                    IEC0bits.T1IE=1;             // enable timer interrupt to avoid hanging
of the code in the while loop

                    AD1CON1bits.DONE = 0;

                    AD1CON1bits.SAMP = 1;            // start sampling ...

                    DelayNmSec(100);             //  about 20us delay

                    AD1CON1bits.SAMP = 0;            // start Converting

                    while (AD1CON1bits.DONE != 1)      // wait till the done bit goes high.
(indicates a to d conversion is over)

                    {

                     if(ADCerror_flag==1)        // do nothing

                     {

                            ADCerror_flag=0;

                            AD1CON1bits.ADON = 0;

                            AD1CON1bits.DONE = 0;

                            break;

                     }

                    }

                    IEC0bits.T1IE=0;             // once it comes out of while loop
indicates adc is over ..so now disable the timer interrupt

                    AD1CON1bits.ADON = 0;

                    AD1CON1bits.DONE = 0;

                    return((float)ADC1BUF0);

}


void DelayNmSec(int time)
```

```
{
            int j;
            for(j=0;j<time;j++)
            {
              ;// delay routine
            }
}
void init_input_capture(void)
{
            T3CON=0x8020;
            TMR3=0;
            INTCON1bits.NSTDIS=0;
            IC1CON=0x0003;                    //ic1  //timer 3 captures the
event,interrupt on every capture,positive edge is detected
            IFS0bits.IC1IF=0;
            IEC0bits.IC1IE=1;
            IC2CON=0x0003;                    //ic2
            IFS0bits.IC2IF=0;
            IEC0bits.IC2IE=1;
}
void _ISR _IC1Interrupt(void)
{
            calc1=IC1BUF;
            IFS0bits.IC1IF=0;
   return;
```

```
}
void _ISR _IC2Interrupt(void)
{
                        calc2=IC2BUF;
                        IFS0bits.IC2IF=0;
   return;
}
void Lcd_Display(void){
                        char temp_str[10];
                        memset(temp_str,0,10);
                        Lcd_Command(0xC0);
                        Delay_ms(2);
                        sprintf(temp_str,"%.1f",Voltage);
                        strcat(temp_str," ");
                        Lcd_String(temp_str);
                        memset(temp_str,0,10);
                        Lcd_Command(0xC6);
                        Delay_ms(2);
                        sprintf(temp_str,"%.2f",Current);
                        strcat(temp_str," ");
                        Lcd_String(temp_str);
                        memset(temp_str,0,10);
                        Lcd_Command(0xCC);
                        Delay_ms(2);
                        if(power_factor> 0.0)    {
```

```
  sprintf(temp_str,"%.2f",power_factor);

  strcat(temp_str," ");

  Lcd_String(temp_str);

}

Lcd_Command(0xC0); }
```

APPENDIX C

REFERENCES

1. William Stevenson, *Power System Analysis and Stability*.

2. David. A. Bell, *Linear Integrated Circuits*

3. Vinaya Skanda, Microchip Technology Inc., ***Power Factor Correction in Power Conversion Applications***

4. Technical Data SA02607001E*, Power Factor Correction: A Guide for the Plant Engineer, November 2010*

5. Dogan Ibrahim, *PIC Microcontroller Projects in C*

7. ABB Technical Applications Paper 8*, Power Factor Correction & Harmonic Filtering*