**05EP707- SYSTEM DESIGN LAB**

**DEPARTMENT OF ELECTRICAL ENGINEERING**

**VISION**

To develop the Department into a “Centre of Excellence” with a perspective to provide quality education and skill-based training with state-of-the-art technologies to the students, thereby enabling them to become achievers and contributors to the industry, society and nation together with a sense of commitment to the profession.

**MISSION**

M1: To impart quality education in tune with emerging technological developments in the field of Electrical and Electronics Engineering.

M2: To provide practical hands-on-training with a view to understand the theoretical concepts and latest technological developments.

M3: To produce employable and self-employable graduates.

M4: To nurture the personality traits among the students in different dimensions emphasizing the ethical values and to address the diversified societal needs of the Nation

M5: To create futuristic ambience with the state-of-the-art facilities for pursuing research.

**PROGRAM EDUCATIONAL OBJECTIVES**

PEO1: Envisage a solid foundation in Basic Sciences, Electrical and Electronics Engineering for a successful career and Life-long Learning in the fields of having Societal Implications.

PEO2: Design and implement effective solutions for complex Electrical and Electronics Engineering problems using modern tools and techniques.

PEO3: Establish Professionalism, Good Communication skills and ethical attitude in multi-disciplinary team work.

PEO4: Apply creative thinking and critical reasoning skills in collaborative research.

 PEO5: Contribute to the economical growth of the country by creating job opportunities through entrepreneurship.

**ANNAMALAI  UNIVERSITY**

**DEPARTMENT OF ELECTRICAL ENGINEERING**

**05EP707- SYSTEM DESIGN LAB**

**List of Experiments**

1. Study of Microcontroller – 89C51
2. Application using 89C51 Microcontroller – Frequency Measurement
3. Study of Microcontroller – PIC 16F877
4. Application of PIC Microcontroller – 16F877
5. Application of ARM7 Processor System – LPC2148
6. Boolean Equation using 89C51 Microcontroller
7. Stepper Motor Control using 89C51 Microcontroller
8. Seven Segment LED Display using 89C51 Microcontroller
9. Analog to Digital Conversion Using Pic Microcontroller – 16f877
10. Study of Digital Signal Processor – TMS320C50

**EX. NO. 1**

**DATE:**

**STUDY OF MICROCONTROLLER – 89C51**

**AIM**

To study the architecture details of 89C51 microcontroller and to perform simple arithmetic operations using the 89C5 1 microcontroller kit.

**INTRODUCTION**

The 89C51 microcontroller is a derivative of the 8051 microcontroller family. The Architecture of 8051 has almost all the blocks available in 89C5 1 except the flash memory. 89C5 1 is a low-power, high-performance CMOS 8-bit microcontroller. It has 64 K of on chip flash, three 16-bit timer/ event counters, a serial port, four general purpose parallel input/output ports and interrupt control logic with 7 sources of interrupts.

**ARCHITECTURE DETAILS**

The 89C5 1 architecture details are same as the 8051 microcontroller, so here the additional features are explained.

**1. Low power modes**

* Stop clock mode
* Idle mode
* Power down mode

**2. The additional timer-TIMER 2**

**3. Special function registers**

The additional SFR available in 89C5 1 is listed below

* CCAPOH-Module 0 Capture High
* CCAP1H-Module 1 Capture High
* CCAP2H-Module 2 Capture High
* CCAP3H-Module 3 Capture High
* CCAP4H-Module 4 Capture High
* CCAPOL-Module 0 Capture Low
* CCAP1L-Module 1 Capture Low
* CCAP2L-Module 2Capture Low
* CCAP3L-Modulc 3 Capture Low
* CCAP4L-Module 4Capture Low
* CCAPMO-Module 0 mode
* CCAPM1-Module 1 mode
* CCAPM2-Module 2 mode
* CCAPM3-Module 3 mode
* CAPM4-Module 4 mode
* CCON -PCA counter control
* CH- PCA Counter High
* CL- PCA Counter Low
* CMOD- PCA Counter Mode
* AUXR-Auxiliary
* AUXRI-Auxiliary
* IPH- Interrupt Priority High
* RCAP2H- Timer2 Capture High
* RCAP2L- Timer2 Capture Low
* SADDR- Slave Address
* SADEN- Slave Address Mask
* T2CON- Timer2 Control
* T2MOD- Timer2 Mode Control
* TH2- Timer High 2
* TL2- Timer Low 2
* WDTRST- watchdog Timer Reset

**4. INTERRUPT SOURCES**

89C51 has a7 source four level interrupt structure. There are 3 SFRs associated with the four level interrupt. They are 1E, IP and 1PH registers. The IPH (Interrupt Priority High) register makes the four level interrupt structure possible.

**5. PROGRAMMABLE COUNTER ARRAY (PCA)**

PCA on 89C5 1 is a special 16- bit timer that has five 16 —bit capture! compare modules. Each module can be programmed to operate in one of four modes.

* Rising and! or falling edge capture
* Software timer
* High- speed output
* Pulse Width Modulator

**6. FLASH EPROM MEMORY**

89C51 contains 64 K bytes of FLASH program memory. This memory is organized as 5 separate blocks. The first 2 blocks are 8 KB in size, filling the programming memory space from address 0000 through 3FFF hex. The final 3 blocks are 16 KB in size and occupy addresses from 4000 through FFFF hex. The flash can be read and written as bytes. The chip erase operation will erase the entire program memory. The block erase function can erase any flash block.



**PROCEDURE**

1. Double click the Ride IDE icon.
2. Enter project menu, select the type 80C51 and by using browse key, open a new folder and give the name.
3. Enter the file menu, create a new file and type the program and save as file name.
4. Click project and select the add node V source /application to add the file to the folder. Then select the hex file and open the file.
5. Click project and select build all.

**FOR DOWNLOADING (set the switch 1 - download switch in the down position and press**

**reset button)**

1. Double click the Win ISP icon.
2. Select Chip - P89C51RD2
3. Select Port - COM 1
4. Select Frequency - 12 MHZ
5. Click the Read button (It will display Boot Vector Read Ok)
6. Click Erase block button and select 0 and 8K block and press Erase button.

 **Don’t touch the Full chip erase button**

1. Click Load file and select the hex file to open (It will display the message file load OK)
2. Click the program part [it will display flash programming successfully]

Click WINX TALK (to see the result)

1. Select port – COM1 and baud rate – 9600 and lick open port
2. Select talk window ? Put the download switch in up position and press rest button, then enter data to get the result.

**PROGRAM 1 – ADDITION OF TWO 8-BIT DATA**

#include <reg5l.h>

 #include <stdio.h>

mt a,b,c, i;

void InitUart(void)

{

 TMOD = 0x20; //9600 baud rate

 TH1 =OxFD;

 TI = 1;

 TR1 = 1;

}

int lcm(int m, int n)

{

 for(i= 1 ;;i++)

 {

 if(i%rn==0&&i%n==0)

 rethrn i;

 }

}

void main()

{

 InitUart();

 printf(tt\.n\r Enter the First Data\t”);

 scanf(”%d”,&a);

 printf(”\n\r A=%d”,a);

 printf(”\n\r Enter the Second Data\t”);

 scanf(”%d’t,&b);

printf(”\n\r B=%d”,b);

c = a + b;

printf(\n\r Result = %d”,c);

while(I);

**PROGRAM 2 – SUBTRACTION OF TWO 8-BIT DATA**

#include <reg5l.h>

#include <stdio.h>

mt a,b,c, i;

void InitUart(void)

{

 TMOD = 0x20; //9600 baud rate

 TH1 =. OxFD; V

 TI = 1; V

 V TR1 = 1;

}

int lcm(int m, int n)

{

 for(i=1;;i++)

 {

 if(i%m==0&&i%n==0)

 return I;

 }

}

void main()

{

 InitUartQ;

 printf(”\n\r Enter the First Data\t”);

 scanf( %d”,&a);

 printf(’t\n\r A=%d11,a);

 printf(\n\r Enter the Second Data\t”);

**SUBTRACTION OF TWO 8-BIT DATA**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.**  | **DATA 1** | **DATA 2** | **RESULT**  |
| **1.** |  |  |  |
| **2.** |  |  |  |

 scanf(”%d’,&b);

 printfC’\n\r B=%d’,b);

 c = a b:

 printf(”\n\r Result = %d”,c);

 while(1);

}

**PROGRAM 3 – MULTIPLICATION OF TWO 8-BIT DATA**

#include <reg5l.h>

#include <stdio.h>

inta,b,c,i;

void InitUart(void)

{

 TMOD 0x20; //9600 baud rate

 TH1 = OxFD;

 TI = 1;

 TR1 = 1;

}

int lcm(int m,intn)

}

 for(i—\_k;;i++)

 {

 if(i%m= =0&&i%n= =0)

 retuñ i;

 }

}

void main()

{

 InitUart();

 printf(u\n\r Enter the First Data\t);

 scanf(”%d”,&a);

 printf(”\n\r A=%d”,a);

 printf(”\n\r Enter the Second Data\t”);

 scanf(”%dt,&b);

 printf(”\n\r B=%d”,b);

 c = a \* b;

 printf(”\n\r Result %d”,c);

 while(1);

**MULTIPLICATION OF TWO 8-BIT DATA**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.**  | **DATA 1** | **DATA 2** | **RESULT**  |
| **1.** |  |  |  |
| **2.** |  |  |  |

**PROGRAM 4 –DIVISION OF TWO 8-BIT DATA**

#include <reg5l.h>

#include <stdio.h>

int a,b,c,d, i;

void InitUart(void)

{

 TMOD 0x20; //9600 baud rate

 THI = OxFD;

 TI = 1;

 TR1 1;

}

ilt lcrn(int m,int n)

{

 for(i1 ;;i++)

 {

 if(i%m==0&&i%n=0)

 return i;

 }

}

void main()

{

 InitUartO;

 printf(”\n\r Enter the First Data\t”);

 scanf(”%dt’,&a);

 printf(”\n\r A=%d,a);

 printf(”\n\r Enter the Second Data\t”);

**DIVISION OF TWO 8-BIT DATA**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.**  | **DATA 1** | **DATA 2** | **RESULT**  |
| **1.** |  |  |  |
| **2.** |  |  |  |

scanf(’ %d,&b);

 printfQ’\n\r B%d”,b);

 c= a/b;

 printf(”\n\r Quotient = %d”,c);

 d=a%b;

 printf(”\n\t Remainder = %d”,d);

 while(1);

}

**RESULTS**

The simple arithmetic programs (such as addition, subtraction, multiplication and division of two 8- bit numbers) are executed and the results are verified.

**EX.NO: 2**

**DATE:**

**APPLICATION USING 89C51 MICROCONTROLLER – FREQUENCY MEASUREMENT**

**ARCHITECTURE DETAILS**

The 89C51 architecture details are same as the 8051 microcontroller, so here the additional features are explained.

**1. Low power modes**

* Stop clock mode
* Idle mode
* Power down mode

**2. The additional timer-timer 2**

**3. Special function registers**

The additional SFR available in 89C51 is listed below

* CCAP0H-Module 0 Capture High
* CCAP1H-Module 1 Capture High
* CCAP2H-Module 2 Capture High
* CCAP3H-Module 3 Capture High
* CCAP4H-Module 4 Capture High
* CCAP0L-Module 0 Capture Low
* CCAP1L-Module 1 Capture Low
* CCAP2L-Module 2Capture Low
* CCAP3L-Module 3 Capture Low
* CCAP4L-Module 4Capture Low
* CCAPM0-Module 0 mode
* CCAPM1-Module 1 mode
* CCAPM2-Module 2 mode
* CCAPM3-Module 3 mode
* CCAPM4-Module 4 mode
* CCON -PCA counter control
* CH- PCA Counter High
* CL- PCA Counter Low
* CMOD- PCA Counter Mode
* AUXR-Auxiliary
* AUXR1-Auxiliary1
* IPH- Interrupt Priority High
* RCAP2H- Timer2 Capture High
* RCAP2L- Timer2 Capture Low
* SADDR- Slave Address
* SADEN- Slave Address Mask
* T2CON- Timer2 Control
* T2MOD- Timer2 Mode Control
* TH2- Timer High 2
* TL2- Timer Low 2
* WDTRST- Watchdog Timer Reset

**4. INTERRUPT SOURCES**

89C51 has a 7 source four level interrupt structure. There are 3 SFRs associated with the four level interrupt. They are IE, IP and IPH registers. The IPH (Interrupt Priority High) register makes the four level interrupt structure possible.

**5. PROGRAMMABLE COUNTER ARRAY (PCA)**

PCA on 89C51 is a special 16- bit timer that has five 16 –bit capture/ compare modules. Each module can be programmed to operate in one of four modes.

**ARCHITECTURE OF 89C51 MICROCONTROLLER**



* Rising and/ or falling edge capture
* Software timer
* High- speed output
* Pulse Width Modulator

**6. FLASH EPROM MEMORY**

89C51 contains 64 K bytes of FLASH program memory. This memory is organized as 5 separate blocks. The first 2 blocks are 8 KB in size, filling the programming memory space from address 0000 through 3FFF hex. The final 3 blocks are 16 KB in size and occupy addresses from 4000 through FFFF hex. The flash can be read and written as bytes. The chip erase operation will erase the entire program memory. The block erase function can erase any flash block.

Enter the file menu - create a new file and type the program and save as file name .c

**APPLICATION I- FREQUENCY MEASUREMENT**

**AIM**

To develop and execute a program in Embedded C to measure the frequency of given input for a particular time using 89C51 microcontroller.

**THEORY**

The frequency of given input is measured by counting the number of pulses for a fixed amount of time. Timer 0 is initialized for 1ms delay and also PCA timer interrupt is initialized. PCA (Programmable Counter Array) is a special 16-bit timer. When PCA interrupt comes, timer0 is turned on and starts counting the number of rising edges in the given input. When timer 0 interrupt comes, both the PCA timer and timer 0 are turned off.

**PROCEDURE**

1. Double click the **Ride IDE** icon

2. Enter project menu, select the type 80C51 and by using browse key, open a new folder and give the name.

3. Enter the file menu, create a new file and type the program and save as file name .c

4. Click project and select the add node source /application to add the file to the folder < select the hex file and open.

5. Click project and select build all.

**FOR DOWNLOADING (set the switch 1 - download switch in the down position and press reset button)**

1. Double click the **Win ISP** icon.

2. Select Chip - p89c51RD2

 Select Port - COM 1

 Select Frequency - 12 MHZ

 3. Click the Read button (It will display Boot Vector Read Ok)

 4. Click Erase block button and select 0 and 8K block and press Erase button.

**Don’t touch the Full chip erase button**

5. Click Load file and select the hex file to open (It will display the message file load OK)

6. Click the program part ( It will display flash programming successful).

Click **WINX TALK** (to see the result)

1. Select port – COM1 and baud rate – 19200 and click open port.

2. Select talk window > Put the download switch in up position and press reset button, then enter data to get the result

**PROGRAM**

#include<reg51f.h>

#include<stdio.h>

#include<math.h>

//Given Frequency 1 KHz using Function Generator -Maximum Frequency 36KHz

int flag =0;

int cap0=0,cap1=0,cap2=0,cap3=0,res=0,res1=0,cap4,cap5,j;

unsigned long int fr =500000;

unsigned long int val, fr\_val;

void PCA\_init()

{

 CMOD = 0X00; //initialize PCA Timer

 CH = 0X00;

 CL = 0X00;

 CCAPM0 = 0X21; //rising edge

 EC = 1;

 EA = 1;

 flag = 0;

}

void InitUart(void)

{

 TMOD = 0x20; // 9600 baud rate

TH1 = 0xFD;

 TI = 1;

 TR1 = 1;

}

main()

{

 InitUart();

 PCA\_init();

 while(1)

 {

val = ((cap3<<0x08));

val = val +(cap2\*1.085+10);

fr\_val =fr/val;

printf(“\n\r Freq=%d”, fr\_val);

}

}

void PCA\_INTERRUPT(void) interrupt 6

{

 CCF0 = 0;

 if(flag == 1 )

 {

 CR = 0;

 cap2 = CL;

 cap3 = CH;

 flag = 0;

 CCAPM0 = 0x21;

 }

else

{

flag =1;

CCAPM0 = 0x11;

CL = 0;

CH = 0;

CR = 1;

}

}

|  |  |  |
| --- | --- | --- |
| S.NO. | SET FREQUENCY VALUE(in KHZ) | MEASURED FREQUENCY VALUE(in KHZ) |
|  |  |  |
|  |  |  |
|  |  |  |

**EX.NO: 3**

**DATE:**

**STUDY OF MICROCONTROLLER – PIC 16F877**

**AIM**

To study the features of the programmable interface circuit chip PIC 16F877.

To develop and execute programs for arithmetic operations in PIC 16F877 using embedded C.

**HARDWARE DESCRIPTION**

PIC 16f877is a 20/48 PIN 8 bit CMOS flash microcontroller. Its operating frequency is 20 MHZ.

**FEATURES**

* + 8k\*14 words of FLASH program memory
	+ 368\*8 bytes of data memory (RAM)
	+ 256\*8 bytes of EEPROM data memory
	+ +5v, 600mA power supply.
	+ 10 bit 8 channel A/D converter.
	+ 10 bit 2 channel PWM outputs.

**Memory Architecture**

Program memory and data memory have separate buses so that concurrent access can occur.

**Data Memory Organisation**

The data memory is portioned into multiple banks which contain general purpose register and special function registers. The lower locations of each bank are reserved for special function registers. Each bank extends up to 7F bits. RPO and RP1 are used to select the memory bank.

**Status Register**

The status register contains the status of ALU, the RESET status and bank select bits.

**Port A and TRIS A Register**

 PORT A is a 6 bit wide, bidirectional port, TRIS A is the corresponding data direction register. Setting TRIS A bit (=1) will make the corresponding port A pin as input and clearing it will make corresponding port A pin as output.

**Port B and TRIS B Register**

This is an 8 bit wide, bidirectional port and its corresponding data direction register is TRIS B. Three pins of PORT B are multiplexed with LOW voltage programming function.

**Port C and TRIS C Register**

PORT C is an 8 bit wide, bidirectional port and its corresponding data direction register is TRIS C. PORT C is multiplexed with several peripheral functions. PORT C pins have Schmitt Trigger input buffers.

**Port D and TRIS D Register**

PORT D is an 8 bit wide port with Schmitt trigger input buffers. Each pin is individually configurable as an input pin or output pin. PORT D can also be configured as an 8 bit wide microprocessor port (i.e) parallel slave port.

**Port E and TRIS E Register**

PORT E has three pins which are individually configurable as inputs and outputs. These have Schmitt trigger input buffers.

**Timer 0 Module**

TIMER 0 module (timer/Counter) has following features.

 8 bit timer or counter

 Readable/Writable

 8 bit software programmable prescaler

 Internal or external clock select

 Interrupt on overflow from FF to 00h.

Edge select for external clocks.

**Timer 1 Module**

The timer 1 module is a 16bit timer/counter consisting of two 8 bit registers which are readable and writable. This also has an internal RESET input.

**Timer 2 Module**

This is 8 bit timer with a pre-scalar and a post scalar. It can be used as the PWM time base for PWM mode of CCP module. This is readable as well as writable and it is cleared on any device RESET.

**Capture/Compare/PWM Modules**

Each capture /compare/PWM module contains a 16 bit register which can operate as

 16 bit capture register

 16 bit compare register

PWM master / slave duty cycle register.

**Addressable Usart Module**

This is one of the two serial I/O modules USART can be configured as a full duplex asynchronous system that can communicate with peripheral devices such as CRT terminals and PC’s or it can be configured as a half duplex system that can communicate with peripheral devices such as A/D, D/A integrated circuits.

**Analog to Digital Converter Module**

This has 5 inputs for a 28 pin device and 8 inputs for 40 pin device.

 **Instruction Set**

Each PIC 16F877 instruction is a 14 bit word divided into an opcode which specifies the instruction type & one or more operands which further specify operation of instruction.

The operations based upon instruction type used are classified as

 Byte oriented operations

 Bit oriented operations

 Literal or control operations.

**ARCHITECTURE OF PIC MICROCONTROLLER 16F877**



**PROCEDURE**

**MPLAB IDE** is a software program that runs on PC to develop applications for microchip controllers. All the programs are executed using MPLAB IDE.

1. Double click on the MPLAB IDE icon installed on the desktop. MPLAB IDE

screen will appear on the Monitor.

2. Go to configure menu & choose the PIC16F877 device.

3. Go to menu & select CCSC compile in the active tool suite box.

CCSC compiler should be visible.

4. Click on the same menu to see it location.

5. Create a project using project wizard.

6. Choose project > project wizard

7. Click on next > to advance

8. Create a new file for the application and save it.

9. Add the newly created file to the project by using add files option available in source file.

10. Compile the source code using compile > project wizard.

Once if the compilation is successful, build message will appear on the screen.

**PIC ISP** is windows serial downloader which supports hex format only. For programming executions, a ting switch is provided

1. Put the switch in programming mode.

2. Open PIC-ISP & choose the hex file to be downloaded.

3. After selecting the file click on the download. ( After the download is completed, the message box will show the “Download Successful” message)

4. After this the switch position is changed to execute mode & then reset CPU.

The code will be executed.

**PROGRAM 1- ADDITION**

#include<16f877.h>

#use delay (clock = 20000000)

#use rs232 (baud = 9600, xmit = PIN-C6, rcv = PIN-C7)

Signed long int i, j,k;

byte gethex 1( )

{

char digit;

digit = getch ( );

putchar (digit);

if (digit< = ‘9’);

return (digit-‘0’);

else

return ((to upper digit – ‘A’) +10);

}

byte gethex ( )

{

int lo, hi:

hi = gethex 1 ( );

lo = gethex 1 ( );

if (lo = = 0 x dd)

return (hi);

else

return (hi \*16 +lo);

}

void main ( )

{

while(1)

{

print f(“/n/r enter first number”);

i = gethex ( );

print f(“/n/r enter second number”);

j = gethex ( );

k =i + j;

print f(“/n/r resultant value = %4LX”, k);

}

}

**ADDITION**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **DATA 1** | **DATA 2** | **RESULT** |
| **1.** |  |  |  |
| **2.** |  |  |  |

**PROGRAM 2- SUBTRACTION**

#include<16f877.h>

#use delay (clock = 20000000)

#use rs232 (baud = 9600, xmit = PIN-C6, rcv = PIN-C7)

Signed long int i, j,k;

byte gethex 1( )

{

char digit;

digit = getch ( );

putchar (digit);

if (digit< = ‘9’);

return (digit-‘0’);

else

return ((to upper digit – ‘A’) +10);

}

byte gethex ( )

{

int lo, hi:

hi = gethex 1 ( );

lo = gethex 1 ( );

if (lo = = 0 x dd)

return (hi);

else

return (hi \*16 +lo);

}

void main ( )

{

while(1)

{

print f(“/n/r enter first number”);

i = gethex ( );

print f(“/n/r enter second number”);

j = gethex ( );

k =i - j;

print f(“/n/r resultant value = %4LX”, k);

}

}

**SUBTRACTION**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **DATA 1** | **DATA 2** | **RESULT** |
| **1.** |  |  |  |
| **2.** |  |  |  |

**PROGRAM 3- MULTIPLICATION**

#include<16f877.h>

#use delay (clock = 20000000)

#use rs232 (baud = 9600, xmit = PIN-C6, rcv = PIN-C7)

Signed long int i, j,k;

byte gethex 1( )

{

char digit;

digit = getch ( );

putchar (digit);

if (digit< = ‘9’);

return (digit-‘0’);

else

return ((to upper digit – ‘A’) +10);

}

byte gethex ( )

{

int lo, hi:

hi = gethex 1 ( );

lo = gethex 1 ( );

if (lo = = 0 x dd)

return (hi);

else

return (hi \*16 +lo);

}

void main ( )

{

while(1)

{

print f(“/n/r enter first number”);

i = gethex ( );

print f(“/n/r enter second number”);

j = gethex ( );

k =i \* j;

print f(“/n/r resultant value = %4LX”, k);

}

}

 **MULTIPLICATION**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **DATA 1** | **DATA 2** | **RESULT** |
| **1.** |  |  |  |
| **2.** |  |  |  |

**PROGRAM 4 - DIVISION**

#include<16f877.h>

#use delay (clock = 20000000)

#use rs232 (baud = 9600, xmit = PIN-C6, rcv = PIN-C7)

Signed long int i, j,k,l;

byte gethex 1( )

{

char digit;

digit = getch ( );

putchar (digit);

if (digit< = ‘9’);

return (digit-‘0’);

else

return ((to upper digit – ‘A’) +10);

}

byte gethex ( )

{

int lo, hi:

hi = gethex 1 ( );

lo = gethex 1 ( );

if (lo = = 0 x dd)

return (hi);

else

return (hi \*16 +lo);

}

 void main ( )

{

while(1)

{

print f(“/n/r enter first number”);

i = gethex ( );

print f(“/n/r enter second number”);

j = gethex ( );

k = i / j;

l = i%j;

print f(“/n/r quotient value = %4LX”, k);

print f(“/n/r remainder value = %4LX”, l);

}

}

**DIVISION**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **DATA 1** | **DATA 2** | **RESULT** |
| **1.** |  |  |  |
| **2.** |  |  |  |

**RESULT**

 The addition, subtraction, multiplication and division programs are executed using PIC 16F877 and the results are verified.

**EX.NO: 4**

**DATE:**

**APPLICATION OF PIC MICROCONTROLLER – 16F877**

**AIM**

To study the following applications of PIC 16F877

1. Seven segment LED display

2. Generation of PWM output

**1) SEVEN SEGMENT LED DISPLAY**

 To study this application, VIPAC-05, 4 digit multiplexed display board is used.

**Seven segment LED displays are preferred because**

1. LCD’s does not emit their own light

2. LED’s are less expensive

3. It is easier to interface them with microcontrollers.

**Types of seven segment display**

1. Common anode display

2. Common cathode display

* For a common anode display, a low is applied to a segment to twin it ON.
* For a common cathode display, a high is applied to a segment to twin it ON.
* In this experiment, common cathode display is used.
* The type of display used is static display because current is passed through the display at all the times, current limiting resistors in series with each segment.
* When the TTL-Low is applied to the input of the seven segment display, it emits one particular display.

**Disadvantage of Static Display**

1. Power consumption is more where several displays are used

2. Separate decoder is used for each seven segment display.

**Multiplexed displays:**

 To solve the disadvantages of static display, information is sent out to all of the digits on a common bus, but only one display is turned on at a time, at which the information is displayed. So it is possible to display information at different digits by simultaneously switching the data as well as the digit location is a sequential order. This refreshing process should be done continuously so that digits will appear to be lit at the same time. Multiplexing gives large power saving.

**Circuit Implementation:**

* VIPAC-05 board has 4 digits and each digit of the display is selected by the ccpin of the seven segment display.
* The digits are selected by the port wire of PIC microcontroller through PNP transistor.
* The segment of each digit is selected by outputting the data through port line of PIC microcontroller; data’s are strengthened by PNP transistors which are used to select the segments (A, B, C, D, E, F, G, and DP).
* To display a digit, the corresponding bits of the segments are quoted to zero and the equivalent data is sent from the PIC microcontroller port lines.
* The port lines of the microcontroller are terminated at 50 pin, FRC connector PI.
* In the display selection, the digits are displayed by the port line RDO to RD7.
* The hexadecimal data corresponding to the segments which have to glow for displaying a character is outputted.
* Logic ‘0’ in the above said format will twin “ON” that particulars segment. Logic”1” will keep the segment “OFF”.
* The data for turning ON the display is through the port lines RB0 to RB3.

**DATA FOR DISPLAYING “ABCD”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DATA** | **Dp** | **g** | **f** | **e** | **d** | **c** | **b** | **a** | **HEX****VALUE** |
| **A** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **88** |
| **B** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **80** |
| **C** | **1** | **1** | **0** | **0** | **0** | **1** | **1** | **0** | **C6** |
| **D** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **C0** |

**PROGRAM: SEVEN SEGMENT DISPLAY**

#include<16f877.h>

void delay();

void delay();

void main()

{

int a;

while(TRUE)

{

output\_b(0xfe);

output\_d(0x88);

delay();

output\_b(0xfd);

output\_d(0x80);

delay();

output\_b(0xfb);

output\_d(0xc6);

delay();

output\_b(0xef);

output\_d(0xc0);

delay();

}

}

void delay()

{

int i,j,k;

for(i=0;i<0x50;i++)

{

for(j=0;j<0xff;j++)

{

}

}

}

**2) GENERATION OF PWM OUTPUT**

**General PWM Output**

 In PWM mode, the CC PX produces up to a 10 bit resolution PWM output. Since CCPX pin is multiplexed with the PORT C data latch, the TRISC<2> bit must be cleared to make the CCPI pin on output.

**PWM Period**

* A PWM output has a time base (Period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period.
* The PWM period is specified by writing to the PR2 register.

The PWM period can be calculated using the following formula.

PWM period = [(PR2+1)]\*4\*TDSC\*[TMR2 prescale value]

**PWM Duty Cycle**

The PWM duty cycle is specified by writing to the CCPRIL register and to the CCPICON bits. Up to 10 bit resolution is possible. The CCPRIL contains 8 MSB’s and CCPICON contains the two LSB’s. The 10 bit value is represented by CCPR1L \* CCP160N.

**Setup for PWM Operation**

The following steps are to be taken for configuring the CCP module for PWM operation.

1. Set the PWM period by writing to the PR2 register.

2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON bits.

3. Make the CCP1 pin an output by clearing the TRISC12>bit.

4. Set the TMR2 prescale value and enable timer 2 by writing to T2CON.

5. Configure CCP1 module for PWM operation.

 **PWM GENERATION**

|  |  |  |
| --- | --- | --- |
| DUTY CYCLE | DEC VALUE | HEX VALUE |
| 25% | 64 | 40H |
| 50% | 128 | 7FH |
| 75% | 192 | C0H |
| 95% | 243 | F3H |

**PROGRAM: PWM GENERATION**

#include<16f877.h>

#use delay (clock=20000000)

#use rs232 (baud=9600, xmit=PIN\_c6, rcv=PIN\_c7)

signed int val;

void main()

{

setup\_ccp1(ccp\_PWM);

setup\_timer\_2(t2\_div\_by\_4,255,1);

while(true)

{

set\_PWM1\_duty (0x40);

}

}

**PROCEDURE**

Execute the programs by using the same procedure that had been used for the execution of arithmetic operation programs using PIC 16F877 microcontroller.

**RESULT**

By using the PIC microcontroller, the various applications such as seven segment LED display, analog to digital conversion and PWM output generation were studied.

**EX.NO: 5**

**DATE :**

**APPLICATION OF ARM7 PROCESSOR SYSTEM- LPC2148**

**AIM**

To develop a program to convert the given analog signal into a digital signal using on–chip ADC interface of LPC2148 development board of ARM7 processor.

**APPARATUS REQUIRED**

ARM7 LPC 2148 Development Kit -1 NO

 Personal Computer -1 NO

**THEORY**

The ARM7 processor is a general purpose 32- bit microprocessor, which offers high performance and very low power consumption. The ARM architecture is based on the Reduced Instruction Set Computer (RISC) principles and it generally runs on a 32-bit instruction set. But it is also capable of running a 16-bit instruction set, known as “THUMB” which helps it achieve a greater code density and enhanced power saving. While all of the register-to-register data processing instructions are single-cycle, other instructions such as data transfer instructions, are multi-cycle. Due to the inherent simplicity of the design and low gate count, ARM 7 is the industry leader in low-power processing on a watts per MIP basis. The ARM core has a built-in JTAG debug port and on-chip “embedded ICE” that allows programs to be downloaded and fully debugged in-system.

In order to keep the ARM 7 both simple and cost-effective, the code and data regions are accessed via a single data bus. Thus while the ARM 7 is capable of single-cycle execution of all data processing instructions, data transfer instructions may take several cycles since they will require at least two accesses onto the bus (one for the instruction one for the data). In order to improve performance, a three stage pipeline is used that allows multiple instructions to be processed simultaneously.

**ARCHITECTURE ARM7 PROCESSOR- LPC2148**

****

The three stages are FETCH, DECODE and EXECUTE. The hardware of each stage is designed to be independent, so that upto three instructions can be processed simultaneously. The pipeline is most effective in speeding up sequential code execution.

 ARM processors are mainly used for instrumentation and control applications. For the measurement and control of physical quantities such as temperature, speed, displacement etc. transducers are used, which give electrical voltage proportional to physical quantities. The electrical voltage which is obtained as an output of a transducer is an analog (continuous) quantity. It must be converted into digital quantity by an A/D converter before it is applied to the processor.

The ARM 2148 development board has two potentiometers for working with A/D converter. All potentiometer outputs are in the range of 0v to 3.3v. Each potentiometer can be connected on two different analog inputs. Jumpers group enables the connection between potentiometer and LPC2148 analog inputs.

ARM processor takes analog signal from its input pin and translates it into a digital value. Basically any analog signal that fits the acceptable range (0v to 3.3v) of LPC2148 can be converted into its equivalent digital signal.

**PROGRAMMING**

It involves three stages,

1. Developing the embedded code C for the particular project in the computer.
2. Generating the hex code for the developed embedded C coding.
3. Downloading the hex code in ARM processor for execution.

**PROCEDURE**

1) Double click IAR embedded work bench icon. A new window is opened.

2) In this window, choose project >create new project.

3) Check, whether Tool chain is set to ARM and click OK button.

4) Select E drive. Now create a new folder and give the folder name, file name and click save button to save the file. In the workspace it displays “Filename \_ Debug”.

5) Now right click “Filename –Debug” in the workspace window and choose options menu and do the following.

 (i) In general options, select Core ATM7TDMI and Device NXPLPC2148.

 (ii)In output converter, select (tick mark) General additional output then choose the output format as Intel extended and select (tick mark) Override default.

 (iii) In Linker, select (tick mark) Override default and clear the existing path. Now use the browse button to create the path as follows.

C drive>Program files>IAR Systems> Embedded work bench 5.4 kickstart> arm> examples> NXP>LPC214X> IAR-LPC-214X>ADC>Configuration>LPC2148\_flash.icf.

Click Ok button.

6) Choose file menu and select “New”. This will create a new window. Now type the Embedded C Program.

7) Choose File>Save as, give the file name as “FILENAME.C” and save the file.

8) In the work space window, right click the project file to add a source file directly to the project.

9) Choose File>Save workspace as and save the workspace.

10) Choose project> Rebuild all.

**Procedure for Downloading**

1. Double click “Launch LPC210X\_ISP.exe” Icon.
2. Select the port i.e COM 1 and press the Read Device ID. Now place the toggle switch in download mode (down position) and press the reset button. This will display Read Part ID Successfully.
3. Use brose button to select the Hex file for downloading as follows.

E drive> Folder name> debug> exe> Filename.hex.

1. Select Upload to Flash button.

This will display File Upload successfully complected.

Now close the LPC 2000 flash utility window.

To get the result

1) Double click “WINXTALK” Icon.

2) Select the communication port – COM1.

3) Set the baud rate as 19200 and click open port.

4) Select language >ASCII.

5) Change the toggle switch to execution mode (up position) and press reset button.

6) Vary the pot and set the analog values.

7) Note down the output digital values and check with the calculated values.

**Maximum reference voltage =**

|  |  |  |
| --- | --- | --- |
| **Analog voltage****(set value)** | **Digital voltage** | **Analog voltage****(calculated value)** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Specimen Calculation:**

**PROGRAM**

//On Chip ADC CHANNEL 0 INTERFACE

#include<NXP/iolpc2148.h>

#include<stdio.h>

#define DESIRED\_BAUDRATE 19200

#define CRYSTAL\_FREQUENCY\_IN\_HZ 12000000

#define PCLK CRYSTAL\_FREQUENCY\_IN\_HZ

#define DIVISOR(PCLK/(16\*DESIRED\_BAUDRATE))

int putchar(int a)

{

 while(!(U0LSR&0x20));

 return(U0THR=a);

}

void ADC\_init()

{

 AD0CR\_bit.SEL=2; //enable AD0.1 only

 AD0CR\_bit.CLKDIV=12000000/4500000;

 AD0CR\_bit.BURST=1; // put A/D into continuous convert mode

 AD0CR\_bit.CLKS=0; //11clocks/10 bit accuracy

 AD0CR\_bit.PDN=1; //power up the unit

 AD0CR\_bit.START=0x0001; //start 1st cnvrsn immediately

}

Void Arm\_Uart0\_Init() //UART Config.

{

 U0LCR=0x83; //U0LCR:UART0 Line Control Register.

 //0x83: enable Divisor Latch access,set 8-bit

 //word length.

 //1 stop bit, no parity, disable break

 //transmission.

VPBDIV=0x01; //VPBDIV : VPB bus clock divider

 //0x01:PCLK=processor clock

U0DLL=DIVISOR&0xFF; //U0DLL:UART0 Divisor Latch(LSB).

U0DLM=DIVISOR>>8; //U0DLM:UART0 Divisor Latch(MSB).

U0LCR=0x03; //U0LCR:UART0 Line Control Register

 //0x03: same as above,but disable Divisor

 //Latch accesss and same time U0THR(Transmitting

 //register writing)holding the data.

U0FCR=0x05; //U0FCR:UART0 FIFO Control Register

 //0x05.Clear Tx FIFO and enable Rx and Tx //FIFOs

}

//...............................................................................................................................

//ADC INTERFACE

//P0.28 - ADC0.1

//ADC VALUE Displayed on UART0

//........................................................................................................................................

Void main()

{

unsigned int ADC result;

PINSEL0=0x00000005; //UART0 PIN SELECTION

PINSEL1=0x01000000; //ADC0.1 CHANNEL SELECT

Arm\_Uart0\_Init();

ADC\_init();

while(1)

{

while(AD0DR\_bit.DONE==0); //wait until conversion done

ADC result=(AD0DR>>6)&0x3ff; // conversion data holds AD0DR0[6] to AD0DR0[15]

printf(“\n\r ADC VALUE------>%x”,ADC result); //send ADC Value to serial window

}}

**RESULT**

 The embedded C code developed for A/D conversion has been verified using LPC2148 development board of ARM7 processor.

**EX. NO. 6**

**DATE:**

**BOOLEAN EQUATION USING 89C51 MICROCONTROLLER**

**AIM**

To develop and execute a program in Embedded C to solve a given equation using Boolean algebra.

**THEORY**

To solve a equation Y= AB + CD using Boolean algebra.



The inputs A, B, C & D are given using switches and the output Y is displayed in LED.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.NO.** | **INPUT A** | **INPUT B** | **INPUT C** | **INPUT D** | **OUTPUT Y** |
| **1** | **0** | **0** | **0** | **0** | **0** |
| **2** | **0** | **0** | **0** | **1** | **0** |
| **3** | **0** | **0** | **1** | **0** | **0** |
| **4** | **0** | **0** | **1** | **1** | **1** |
| **5** | **0** | **1** | **0** | **0** | **0** |
| **6** | **0** | **1** | **0** | **1** | **0** |
| **7** | **0** | **1** | **1** | **0** | **0** |
| **8** | **0** | **1** | **1** | **1** | **1** |
| **9** | **1** | **0** | **0** | **0** | **0** |
| **10** | **1** | **0** | **0** | **1** | **0** |
| **11** | **1** | **0** | **1** | **0** | **0** |
| **12** | **1** | **0** | **1** | **1** | **1** |
| **13** | **1** | **1** | **0** | **0** | **1** |
| **14** | **1** | **1** | **0** | **1** | **1** |
| **15** | **1** | **1** | **1** | **0** | **1** |
| **16** | **1** | **1** | **1** | **1** | **1** |

 **PROGRAM**

#include <reg51.h>

#include <stdio.h>

// Boolean Equation Y = AB + CD

sbit A\_1 = P1^0;

sbit B\_1 = P1^1;

sbit C\_1 = P1^2;

sbit D\_1 = P1^3;

sbit y1 =P2^0;

sbit y2 =P2^1;

sbit y3 =P2^2;

sbit y4 =P2^3;

sbit y5 =P2^4;

sbit y6 =P2^5;

sbit y7 =P2^7;

int val,val1,val2,val3,val4,val5,out1,out2,out3,output;

main()

{

 while(1)

 {

 P2 = 0x00;

 val = A\_1;

 y1 = val;

 val1 = B\_1;

 y2 = val1;

 val2 = A\_1 & B\_1;

 y3 = val2;

 val3 = C\_1;

 y4 = val3;

 val4 = D\_1;

 y5 = val4;

 val5 = C\_1 & D\_1;

 y6 = val5;

 output = val2 | val5;

 y7 = output;

 printf("\n\r Output %x",output);

 }

}

**RESULT**

By using 89C51 microcontroller, the programs for frequency measurement and solving the equation in Boolean algebra were executed and results were verified.

**EX.NO: 7**

**DATE :**

**STEPPER MOTOR CONTROL USING 89C51 MICROCONTROLLER**

To write and execute an Embedded C program for stepper motor control using 89C51 microcontroller kit.

**THEORY**

 A Stepper motor is a device which converts digits pulses to precise angular movements. The rotory motion occurs in a stepwise manner from one equilibrium position to the next. Stepper motors are widely used in simple position control systems in the open and closed loop mode. It is used for variety of applications such as computer peripherals (printers, disk driver etc.) and in the areas of process control machine tools, medicine, numerically controlled machines and robotics.

 A Stepper motor could be either of the reluctance type or of the permanent magnet type (PM). A PM stepper motor consists of multi-phase stator and two part permanent magnet rotor. The VR stepper motor has un-magnetised rotor. PM stepper motor is the most commonly used type. The basic two phase stepper motor consists of two pairs of stator poles. Each of the four poles has its own winding. The excitation of any one winding generates a north pole (N), a south pole(S) gets induced at the diametrically opposite side.

As shown in the Fig.1, the four pole structure is continuous with the stator frame and the magnetic field passes through the cylindrical stator annular ring. The rotor magnetic system has two end faces. The left face is permanently magnetised as South Pole and the right face as north Pole. The South pole structure and the North Pole structure possess similar pole faces.

The North Pole structure is twisted with respect to the South Pole structure, so that South Pole comes precisely between two north poles. The North Pole structure is offset with respect to the South Pole structure by one pole pitch. The cross sectional view is shown in Fig. 2. In an arrangement where there are four stator poles and three pairs of rotor poles, there exists 12 possible stable positions in which a south pole of the rotor can lock with a north pole of the stator.

There are three different schemes available for "stepping" a stepper motor. They are:

(a)Wave scheme

(b) 2-phase scheme and

(c) Half stepping or mixed scheme



 **Fig. 1 Stepper motor-Four pole structure**



**Fig. 2 Stepper motor-** **Cross sectional view**

**2-phase scheme**

 In this scheme, any two adjacent stator windings are energised. There are two magnetic fields active in quadrature and none of the rotor pole faces can be in direct alignment with the stator poles. A partial but symmetric alignment of the rotor poles is of course possible. Typical equilibrium conditions of the rotor when the windings on two successive stator poles are excited is illustrated in Fig.3. In step (a), A1 and B1 are energised. The pole-face S1 tries to align itself with the axis of A1 (N) and the pole face S2 with B1(N). The north pole N3 of the rotor finds itself in the neutral zone between A1(N) and B1(N). S1 and S2 of the rotor, position themselves symmetrically with respect to the two stator north pole. Next, when B1 and A2 are energised, S2 tends to align with B1(N) and S3 with A2(N). Of course, again under equilibrium conditions, only partial alignment is possible and N1 finds itself in the neutral region, midway between B1(N) and A2(N) [Step (b)]. In step (c), A2 and B2 are on. S3 and S1 tend to align with A2(N) and B2(N), respectively, with N2 in the neutral zone. Step (d) illustrates the case when A1 and B2 are on. The switching sequence for the 2-phase scheme is given below

The stepping action is caused by sequential switching of the power supply to 2 phases of motor having double winding with centre taps. The angle of each step for the motor can be calculated as

Step angle in degrees = $\frac{360^{0}}{N\_{s }\*N\_{r}}$

Where,

 $ N\_{s }$is the No. of the stacks.

$ N\_{r }$is the No. of rotor teeth.

Step angle in degrees = $\frac{360^{0}}{2\*100}$ = $1.8^{0}$

Therefore,

No. of steps / revolutions = $\frac{360^{0} }{1.8}$= 200 steps

The firing pulses corresponding to data 0A, 06, 05, & 09 cause 4 steps movement. i.e., in each sequence it produces 4 steps.

Therefore,

No. of sequence requiring to produce one revolution = No. of steps / revolution

 No. of step / sequence

 = $\frac{200}{4}$ = 50 d =32 h

**Table for 2-phase switching scheme**

|  |  |
| --- | --- |
| STEP | Clockwise |
|  |  |  |  | A1 | A2 | B1 | B2 |  |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Firing angle |
| 1 | x | x | x | X | 1 | 0 | 0 | 1 | 09 |
| 2 | x | x | x | X | 0 | 1 | 0 | 1 | 05 |
| 3 | x | x | x | X | 0 | 1 | 1 | 0 | 06 |
| 4 | x | x | x | X | 1 | 0 | 1 | 0 | 0A |

**PROCEDURE**

1. Double click the **Ride IDE** icon

2. Enter project menu, select the type 80C51 and by using browse key, open a new folder and give the name.

3. Enter the file menu, create a new file and type the program and save as file name .c

4. Click project and select the add node source /application to add the file to the folder < select the hex file and open.

5. Click project and select build all.

**FOR DOWNLOADING (set the switch 1 - download switch in the down position and press reset button)**

1. Double click the **Win ISP** icon.

2. Select Chip - p89c51RD2

 Select Port - COM 1

 Select Frequency - 12 MHZ

 3. Click the Read button (It will display Boot Vector Read Ok)

 4. Click Erase block button and select 0 and 8K block and press Erase button.

**Don’t touch the Full chip erase button**

5. Click Load file and select the hex file to open (It will display the message file load OK)

6. Click the program part ( It will display flash programming successful).

Click **WINX TALK** (to see the result)

1. Select port – COM1 and baud rate – 19200 and click open port

2. Select talk window > Put the download switch in up position and press reset button, then enter data to get the result.

**step (a) step (b)**

**step (c) step (d)**

 **Fig. 3 2-phase drive scheme**

**PROGRAM**

#include<reg51.h>

#include<stdio.h>

xdata char \*motorsel;

int val;

void InitUart(void)

{

 TMOD = 0x20; // 9600 baud rate

 TH1 = 0xFD;

 TI = 1;

 TR1 = 1;

}

void delay()

{

 unsigned int i,j;

 for(i=0;i<=0xff;i++)

 for(j=0;j<=0xff;j++);

}

void main()

{

 InitUart();

 motorsel = 0xff11;

 while(1)

 {

 \*motorsel = 0x09;

 delay();

 \*motorsel = 0x05;

 delay();

 \*motorsel = 0x06;

 delay();

 \*motorsel = 0x0a;

 delay();

 }

}

**RESULT**

**EX. No: 8**

**DATE:**

**SEVEN SEGMENT LED DISPLAY USING 89C51 MICROCONTROLLER**

**AIM**

To develop a program to display the hexadecimal characters (0 to 9) using 89C51 microcontroller.

**APPARATUS REQUIRED**

89C51 Microcontroller Development Kit -1 NO

 Personal Computer -1 NO

**THEORY**

The seven segment LED display is a multiple display. It can display all hexadecimal characters. It is very popular among multiple displays as it has the smallest number of separately controlled light emitting diodes (LED).

In a seven segment display there are seven light emitting diodes. Each LED can be controlled separately. To display a digit or letter the desired segments are made ON. There are two types of seven segment display namely, common cathode type and common anode type.

In a common cathode type display, all the 7 cathodes of LED’s are tied together to the ground. When a +5Vdc is applied to any segment, the corresponding diode emits light. Thus applying logic ‘1’.i.e, positive logic to the desired segments, the desired letter or decimal number can be displayed.

In a common anode type display all the 7 anodes are tied together and connected to +5V supply. A particular segment will emit light when 0 logic is applied to it.

A program has been developed to display the decimal numbers 0 to 9 using a common cathode type and a delay subroutine has been included in the program.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Data*  | ***dp*** | ***G*** | ***f*** | ***e*** | ***d*** | ***c*** | ***b*** | ***a*** | ***HEX******VALUE*** |
| *0* | *0* | *0* | *1* | *1* | *1* | *1* | *1* | *1* | *3f* |
| *1* | *0* | *0* | *0* | *0* | *0* | *1* | *1* | *0* | *06* |
| *2* | *0* | *1* | *0* | *1* | *1* | *0* | *1* | *1* | *5b* |
| *3* | *0* | *1* | *0* | *0* | *1* | *1* | *1* | *1* | *4f* |
| *4* | *0* | *1* | *1* | *0* | *0* | *1* | *1* | *0* | *66* |
| *5* | *0* | *1* | *1* | *0* | *1* | *1* | *0* | *1* | *6d* |
| *6* | *0* | *1* | *1* | *1* | *1* | *1* | *0* | *1* | *7d* |
| *7* | *0* | *0* | *0* | *0* | *0* | *1* | *1* | *1* | *07* |
| *8* | *0* | *1* | *1* | *1* | *1* | *1* | *1* | *1* | *7f* |
| *9* | *0* | *1* | *1* | *0* | *0* | *1* | *1* | *1* | *67* |

**PROCEDURE**

1. Double click the **Ride IDE** icon

2. Enter project menu, select the type 80C51 and by using browse key, open a new folder and give the name.

3. Enter the file menu, create a new file and type the program and save as file name .c

4. Click project and select the add node source /application to add the file to the folder < select the hex file and open.

5. Click project and select build all.

**FOR DOWNLOADING (set the switch 1 - download switch in the down position and press reset button)**

1. Double click the **Win ISP** icon.

2. Select Chip - p89c51RD2

 Select Port - COM 1

 Select Frequency - 12 MHZ

 3. Click the Read button (It will display Boot Vector Read Ok)

 4. Click Erase block button and select 0 and 8K block and press Erase button.

**Don’t touch the Full chip erase button**

5. Click Load file and select the hex file to open (It will display the message file load OK)

6. Click the program part ( It will display flash programming successful).

**FOR SEEING RESULT (set the switch 1 - download switch in the UP position and press reset button)**

See the result in the seven segment LED display.

**PROGRAM**

#include<reg51.h>

xdata char \*segsel;

xdata char \*segdata;

int k;

unsigned char arr[12] = { 0Xc0,0XF9,0XA4,0Xb0,0X99,0X92,0X92,0XF8,0X80,0x7f } ;

void delay()

{

int i,j;

for(i=0;i<0xff;i++)

for(j=0;j<0xff;j++);

}

void main()

{

 segsel = 0xff12;

 segdata = 0xff21;

 while(1)

 {

 \*segsel = 0x01;

 \*segdata = 0xc0;

 delay();

 \*segsel = 0x01;

 \*segdata = 0xf9;

 delay();

 \*segsel = 0x01;

 \*segdata = 0xa4;

 delay();

 \*segsel = 0x01;

 \*segdata = 0xb0;

 delay();

 \*segsel = 0x01;

 \*segdata = 0x99;

 delay();

 \*segsel = 0x01;

 \*segdata = 0x92;

 delay();

 \*segsel = 0x01;

 \*segdata = 0xf8;

 delay();

 \*segsel = 0x01;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x01;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x01;

 \*segdata = 0;

 delay();

 \*segsel = 0x02;

 \*segdata = 0xc0;

 delay();

 \*segsel = 0x02;

 \*segdata = 0xf9;

 delay();

 \*segsel = 0x02;

 \*segdata = 0xa4;

 delay();

 \*segsel = 0x02;

 \*segdata = 0xb0;

 delay();

 \*segsel = 0x02;

 \*segdata = 0x99;

 delay();

 \*segsel = 0x02;

 \*segdata = 0x92;

 delay();

 \*segsel = 0x02;

 \*segdata = 0xf8;

 delay();

 \*segsel = 0x02;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x02;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x02;

 \*segdata = 0;

 delay();

 \*segsel = 0x04;

 \*segdata = 0xc0;

 delay();

 \*segsel = 0x04;

 \*segdata = 0xf9;

 delay();

 \*segsel = 0x04;

 \*segdata = 0xa4;

 delay();

 \*segsel = 0x04;

 \*segdata = 0xb0;

 delay();

 \*segsel = 0x04;

 \*segdata = 0x99;

 delay();

 \*segsel = 0x04;

 \*segdata = 0x92;

 delay();

 \*segsel = 0x04;

 \*segdata = 0xf8;

 delay();

 \*segsel = 0x04;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x04;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x04;

 \*segdata = 0;

 delay();

 \*segsel = 0x08;

 \*segdata = 0xc0;

 delay();

 \*segsel = 0x08;

 \*segdata = 0xf9;

 delay();

 \*segsel = 0x08;

 \*segdata = 0xa4;

 delay();

 \*segsel = 0x08;

 \*segdata = 0xb0;

 delay();

 \*segsel = 0x08;

 \*segdata = 0x99;

 delay();

 \*segsel = 0x08;

 \*segdata = 0x92;

 delay();

 \*segsel = 0x08;

 \*segdata = 0xf8;

 delay();

 \*segsel = 0x08;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x08;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x08;

 \*segdata = 0;

 delay();

 \*segsel = 0x10;

 \*segdata = 0xc0;

 delay();

 \*segsel = 0x10;

 \*segdata = 0xf9;

 delay();

 \*segsel = 0x10;

 \*segdata = 0xa4;

 delay();

 \*segsel = 0x10;

 \*segdata = 0xb0;

 delay();

 \*segsel = 0x10;

 \*segdata = 0x99;

 delay();

 \*segsel = 0x10;

 \*segdata = 0x92;

 delay();

 \*segsel = 0x10;

 \*segdata = 0xf8;

 delay();

 \*segsel = 0x10;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x10;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x10;

 \*segdata = 0;

 delay();

 \*segsel = 0x20;

 \*segdata = 0xc0;

 delay();

 \*segsel = 0x20;

 \*segdata = 0xf9;

 delay();

 \*segsel = 0x20;

 \*segdata = 0xa4;

 delay();

 \*segsel = 0x20;

 \*segdata = 0xb0;

 delay();

 \*segsel = 0x20;

 \*segdata = 0x99;

 delay();

 \*segsel = 0x20;

 \*segdata = 0x92;

 delay();

 \*segsel = 0x20;

 \*segdata = 0xf8;

 delay();

 \*segsel = 0x20;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x20;

 \*segdata = 0x00;

 delay();

 \*segsel = 0x20;

 \*segdata = 0;

 delay();

 }

}

**RESULT**

**EX. NO. 9**

**DATE:**

**ANALOG TO DIGITAL CONVERSION USING PIC MICROCONTROLLER – 16F877**

**Aim**

To study and develop the program of Analog to Digital conversion using Pic Microcontroller.

**ANALOG TO DIGITAL CONVERSION:**

* The A/D conversion module has 8 inputs for a 40 pin module.
* The conversion will result in a 10 bit number. The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, VSS RA2 or RA3.
* The A/D module has four registers, these registers are
* A/D Result high register – ADRESH
* A/D Result Low register- ADRESL
* A/D Control register 0-ADCON0
* A/D Control register 1-ADCON1
* ADCON0 register controls the operation of A/D module. ADCON1 register configures the functions of the port pins. The port pin can be configured as analog inputs or as digital I/O.
* To do an A/D conversion, the following steps to be followed.

**1. Configure the A/D module**

**\*** Configures analog pins/voltage reference and digital I/O (ADCON1)

**\*** Select A/D input channel (ADCON0)

**\*** Select A/D conversion clock (ADCON0)

**2. Configure A/D intercept (if desired)**

**3. Wait the required acquisition time**

**4. Start conversion**

 \* Set GO / DONE bit (ADCON0)

**5. Wait for A/D conversion to complete by either**

 \*Polling for the GO/GONE bit to be cleared (interrupts disabled) OR.

 \*Waiting for the A/D intercept.

**6. Result A/D result register**

**Maximum reference voltage =**

|  |  |  |
| --- | --- | --- |
| **Analog voltage****(set value)** | **Digital voltage** | **Analog voltage****(calculated value)** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Specimen Calculation:**

**PROGRAM: ADC**

#include<16f877.h>

#device adc=10

#include<stdio.h>

#use delay(clock=20000000)

#use rs232(baud=19200, xmit=pin\_c6, rcv=pin\_C7)

int i;

void delay();

void main()

{

signed long int c;

setup\_adc\_ports(ALL\_ANALOG);

setup\_adc(ADC\_CLOCK\_INTERNAL);

set\_adc\_channel(0);

for(i=0;i<2;i++)

{

delay();

c=read\_adc();

printf(“\n\rAdc value:%4Lx”,c);

}

}

delay()

{

int j;

for(j=0;j<45;j++);

}

**Result :**

The embedded C code developed for A/D conversion has been verified using PIC Microcontroller

**EX. NO. 10**

**DATE:**

**STUDY OF DIGITAL SIGNAL PROCESSOR – TMS320C50**

**AIM**

1. To study the architecture details of digital signal processor –TMS320C50
2. To write and execute the programs in TMS320C50.

**DSP Kit:**

1. Extended precision addition of two 64 bit numbers.
2. Extended precision subtraction of two 64 bit numbers
3. Multiplication of two integer numbers.

**Digital Signal Processor:**

Digital Signal Processor can perform signal based task against digital data. The basic Digital Signal Processor diagram is given below.

ADC

DSP

DAC

Real World

Signal (Analog)

Real World

Signal (Analog)

**TMS320 Family:**

The TMS320 family consists of two types of single – chip DSP’s : 16 bit fixed point and 32 bit floating point. These DSP’s posses the operational flexibility of high speed controllers and the numerical capability of array processors. Combining these two qualities, the TM320 processors are inexpensive alternatives to custom fabricated VLSI and Multi chip bit slice processors.

* Very flexible instruction set.



* Inherent operational flexibility
* High speed performance
* Innovative, parallel architectural design
* Cost effectiveness

**TM320 Typical Applications:**

The TMS320 family of DSP’s offers better, more adaptable approaches to traditional signal processing problems, such as vo coding, filtering and error coding. Further more, the TMS320 family supports complex applications that often require multiple operations to be performed simultaneously.

**TMS320 C5X Overview:**

The C5X generation consist of the ‘C50’, ‘C51’, ‘C52’, ‘C53’, ‘C53S’, ‘C56’, ‘C57’ and ‘C57S’. DSPs which are fabricated by CMOS integrated – circuit technology. Their architectural design is based on the ‘C25’. The operational flexibility and speed of the ‘C5X’ are the result of combining an advanced Harvard architecture (which has separate memory for program memory and data memory), a CPU with application specific hardware logic, on chip peripherals, on chip memory, and a highly specialized instruction set. The ‘C5X is designed to execute up to 50 million instructions per second (MIPS). Spin – off devices that combines the ‘C5X CPU with customized on chip memory and peripherals configurations may be developed for special applications in the world wide electronics market.

The ‘C5X devices offer these advantages:

* Enhanced TM320 architectural design for increased performance and versatility.
* Modular architectural design for fast development of spin off devices.
* Advanced integrated circuit processing technology for increased performance and low power consumption.
* Source code compatibity with ‘C1X, ‘C2X and ‘C2XX DSPs for fast and easy performance upgrades.
* Enhanced instruction set for faster algorithms and for optimized high level language operation.
* Reduced power consumption and increased radiation hardness because of new static design technique.

**Bus Structure:**

Separate programe and data buses allow simultaneous access to program instructions and data, providing a high degree of parallelism. For example, while, which data is multiplied, a previous product can be loaded into, added to or subtracted from the accumulator and at the same time, a new address can be generated, such parallelism supports a powerful set of arithmetic, logic, an bit – manipulation operations that can all be performed in a single machine cycle. In addition, the ‘C5X includes the control mechanisms to manage interrupts, repeated operations and function calling.

The ‘C5X architecture has four major buses:

* Program bus (PB)
* Program address bus (PAB)
* Data read bus (DB)
* Data read address bus (DAB)

The PAB provides address to program memory space for both reads and writes. The PB also carries the instruction code and immediate operands from program memory space to the CPU. The DB interconnects various elements of the CPU to data memory space. The program and data been can work together to transfer data from on chip data memory and memory and internal or external program memory to the multiplier for single cycle multiple accumulate operations.

**Central Processing Unit (CPU):**

The ‘C5X CPU consists of these elements:

* Central arithmetic logic unit (CALU)
* Parallel logic unit (PLU)
* Auxiliary register arithmetic unit (ARAU)
* Memory mapped registers
* Program controller.

**On Chip Memory:**

The ‘C5X architecture contains a considerable amount of on chip memory to aid in system performance and integration.

* Program read only memory
* Data / Program single access RAM (DARAM)
* Data / Program Single access RAM (SARAM)

The ‘C5X has a total address range of 224K words X16 bits. The memory space is divided into four individually selectable memory segments: 64K – word program memory space 64-K word local data memory space, 64-K word input / output posts, and 32K – word global data memory space. For information on the memory organization.

**On Chip Peripherals**

All ‘C5X DSPs have the same CPU structure, however they have different on chip peripherals connected to their CPUs.

The ‘C50 DSP an chip peripherals available are:

* Clock generator
* Hardware timer
* Software – programmable wait state generators
* Parallel I/o pasts.
* Host post interface (HPI)
* Buffered serial post (BSP)
* Time division multiplexed (TDM) serial post
* User – mask able interrupts.

**Program 1:** To perform extended precision addition of two 64 bit numbers.

The two 64-bit numbers are added as shown below, where all the numbers XB, X2, XL, XO, Y3, Y2, Y1, Y0 are 16 bit numbers. TMS320C50 has provision to added two 32 bit no’s

 X3 X2 X1 X0

 + Y3 Y2 Y1 Y0

 W3 W2 W1 W0

For example the addition of 5432109876543210 and 0123456789012345 results in 555555FFFF555555, as shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X – operand  | X3 | X2 | X1 | X0 |
|  | 5432 | 1098 | 7654 | 3210 |
| Y – operand  | Y3 | Y2 | Y1 | Y0 |
|  | 0123 | 4567 | 8901 | 2345 |
| Result  | W3 | W2 | W1 | W0 |
|  | 5555 | 55FF | FF55 | 5555 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Address**  | **Label**  | **Opcode**  | **Memories**  | **Comments**  |
| C000 |  | BD, 00 | LDP # 100H | This instruction considers address values 0000, 0001, 0002 which is entered in the program as 8000, 8001, 8002 resp.  |
| C001 |  | 6A, 01 | LACC 0001, 10H | ACC = X100 |
| C002 |  | 62, 00 | ADDS 0000 | ACC = X1 X0 |
| C003 |  | 66, 04 | ADDS 0004 | ACC = X1 X0 + 00 Y0 |
| C004 |  | 65, 05 | ADD 0005, 10H | ACC = X1 X0 + Y1 Y0 |
| C005 |  | 90,08 | SACL 0008 | ACCL = W0, lower 16 bits of Accumulator is stored in the memory location 8008 |
| C006 |  | 98,0A | SACH 000A | ACCH – W1, higher 16 bits of accumulator is stored in the memory location 8009 |
| C007 |  | 6A, 03 | LACC 0003, 10H | ACC =X300 |
| C008 |  | 62, 02 | ADDS 0002 | ACC=X3 X2 + Carry  |
| C009 |  | 62, 06 | ADDS0006 | ACC=X3 X2 + 00Y2+ Carry  |
| C00A |  | 61, 07 | ADD 0007, 10H | ACC = X3 Y2 + Y3 Y2 + Carry  |
| C00B |  | 90, OA | SACL 000A | ACCL = W2, Low 16 bits of accumulator is stored in the memory location 800A |
| C00C |  | 98, 0B | SACH 000B | ACCH = W3, higher 16 bits of accumulator is stored in the memory location 800B |
| C00D  |  | 79, 80 | B | Halt  |
| CooE |  | C00C | END |  |

**Date:**

|  |  |
| --- | --- |
| **Memory address**  | **Data** |
| 8000 (X0) |  |
| 8001 (X1) |  |
| 8002 (X2) |  |
| 8003 (X3) |  |
| 8004 (Y0) |  |
| 8005 (Y1) |  |
| 8006 (Y2) |  |
| 8007 (Y3) |  |

**Result:**

|  |  |
| --- | --- |
| **Memory address**  | **Data** |
| 8008 (W0) |  |
| 8009 (W1) |  |
| 800A (W2) |  |
| 800B (W3) |  |

**Program 2:** To perform extended precision subtraction of two 64-bit numbers.

The two 64-bit numbers X and Y are subtracted to give the result was follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X – operand  | X3 | X2 | X1 | X0 |
|  | 2222 | 2222 | 2222 | 2222 |
| Y – operand  | Y3 | Y2 | Y1 | Y0 |
|  | 1111 | 1111 | 1111 | 1111 |
| Result  | W3 | W2 | W1 | W0 |
|  | 1111 | 1111 | 1111 | 1111 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Address**  | **Label**  | **Opcode**  | **Memories**  | **Comments**  |
| C000 |  | BD, 00 | LDP # 100H | This instruction considers address values 0000, 0001, 0002 which is entered in the program as 8000, 8001, 8002 resp.  |
| C001 |  | 6A, 01 | LACC 0001, 10H | ACC = X100 |
| C002 |  | 62, 00 | ADDS 0000 | ACC = X1 X0 |
| C003 |  | 66, 04 | ADDS 0004 | ACC = X1 X0 + 00 Y0 |
| C004 |  | 65, 05 | ADD 0005, 10H | ACC = X1 X0 + Y1 Y0 |
| C005 |  | 90,08 | SACL 0008 | ACCL = W0, lower 16 bits of Accumulator is stored in the memory location 8008 |
| C006 |  | 98,09 | SACH 000A | ACCH – W1, higher 16 bits of accumulator is stored in the memory location 8009 |
| C007 |  | 10, 02 | LACC 0003, 10H | ACC =00X2 |
| C008 |  | 64, 06 | ADDS 0002 | ACC= 00X2 – 00Y2 – Carry  |
| C009 |  | 65, 07 | ADDS0006 |  |
| C00A |  |  | ADD 0007, 10H | ACC = X3 Y2 + Y3 Y2 + Carry  |
| C00B |  | 90, 0A | SACL 000A | ACCL = W2, Low 16 bits of accumulator is stored in the memory location 800A |
| C00C |  | 98, 0B | SACH 000B | ACCH = W3, higher 16 bits of accumulator is stored in the memory location 800B |
| C00D  |  | 79, 80 | B | Halt  |
| C00E |  | C0, 0C | B |  |

**Date:**

|  |  |
| --- | --- |
| **Memory address**  | **Data** |
| 8000 (X0) |  |
| 8001 (X1) |  |
| 8002 (X2) |  |
| 8003 (X3) |  |
| 8004 (Y0) |  |
| 8005 (Y1) |  |
| 8006 (Y2) |  |
| 8007 (Y3) |  |

**Result:**

|  |  |
| --- | --- |
| **Memory address**  | **Data** |
| 8008 (W0) |  |
| 8009 (W1) |  |
| 800A (W2) |  |
| 800B (W3) |  |

**Program 3:**

To perform multiplication of two integer number.

The TMS320C50 hardware multiplier normally perform two’s complement 16 bit by 16 bit multiplies and produces a 32 bit result in a single processor cycle. To multiply two operands, one operand must be located into T- register. The second operand is moved by the multiply instruction to the multiplier, which then produces the product in the p-register. Before another multiply can be performed the content of the register should be moved to the accumulator or should be stored in data memory.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  **Address**  | **Label**  | **Opcode**  | **Memories**  | **Comments**  |
| C000 |  | BD, 00 | LDP # 100H | This instruction considers address values 0000, 0001, 0002 which is entered in the program as 8000, 8001, 8002 resp.  |
| C001 |  | BF, 80 | LACC #037AM | Loads the accumulator with the first operand ACC = 037A |
| C002 |  | 03, 7A |  |  |
| C003 |  | 90, 00 | SACL 000 | Stores the accumulator content (lower 16 bits) the memory location 8000 |
| C004 |  | BF, 80 | LACC # 012EH | Loads the accumulator with the second operand ACC = 012E |
| C005 |  | 01, 2E |  |  |
| C006 |  | 90, 01 | SACL0001 | Stores the accumulator content (lower 16 bits) to the memory location 8001 |
| C007 |  | 73, 00 | LT0000 | 7=037A, Loads the first operand fr the memory location 8000 to T register. |
| C008 |  | 54, 01 | MPY0001 | P = 037A\* 012E = 000419EC, multiplies the content of the memory location 8001 with T register and the result is stored in P register |
| C009 |  | BE, 03 | PAC | ACC = 000419EC, transfers the content of P register to accumulator  |
| C00A |  | 90, 02 | SACL0002 | Lower 16 bits of accumulator is stored in the memory location 8002 |
| C00B |  | 98, 03 | SACL0003 | Higher 16 bits of accumulator is stored in the memory location 8003  |
| C00C |  | 79, 80 | B | Halt  |
| C00D  |  | C00C |  |

**Result:**

|  |  |
| --- | --- |
| **Memory address**  | **Data** |
| 8002 |  |
| 8003 |  |

**Result:**

The above programs are executed and their results are verified using TMS320C50 DSP.