# Clemson University Department of Electrical and Computer Engineering

# ECE 372
# Microcomputer Interfacing Laboratory

## LAB MANUAL

Dr. John Gowdy
Dr. William E. Reid

By
Apoorva Kapadia
Andrew Neff
Jamie Wight

Special Thanks to
Jerry James

# ECE 372 – Microcomputer Interfacing Laboratory
# Lab Manual
## Table of Contents

## Format of the Lab Report

## Laboratory X

## Title

**NAME:** Give your name.

**CLASS:** ECE 372 Section ___

**DATE:** Indicate date the lab was performed.

**OBJECTIVE:** Clearly state, in your own words, the objective of performing the lab.

**EQUIPMENT USED:** Indicate the equipment used to perform the experiment as well as the chip numbers of any additional microchips used.

**PROCEDURE:** Provide a concise summary of the procedure used in the lab. Include any modifications you might have made to the experiment.

**DESCRIPTION OF CODE:** This section should verbally discuss how the C-Code accomplishes the task outlined in the Objective. All functions should have their purpose and arguments explained here.

**DATA:** Provide a record of the data obtained during the experiment. The data should be presented in a clear manner, preferably using tables.

**OBSERVATIONS AND DISCUSSIONS:** The student should state their observations after having performed the experiment. Sources of problems or errors should be noted here, along with possible solutions or corrections.

**QUESTIONS:** This section should have answers to relevant questions pertaining to the laboratory experiment performed.

**CONCLUSIONS:** The student should present conclusions which may be logically deduced from their data and observations.

*"This report is accurate to the best of my knowledge and is a true representation of my laboratory results."*

**Signed.**

**C- CODE:** Comprehensively commented C-code should be attached to the end of the report.

# Sample Report
## 8250 EXPERIMENT (Serial I/O)

**NAME:** John Doe

**DATE:** March 3, 1993

**OBJECTIVE:** The objective of this laboratory is to design and implement an interface to an 8250 serial input/output chip with a base address of 100H. A C language program is to be written to test this interface. An optional objective is to have the interfaced 8250 communicate with COM1 port of the PC via a null modem cable using the test software.

**EQUIPMENT:** PB-88/4 Breadboarding Design Station
GAL20V8A PAL for Address Decoding
8250 Serial I/O Chip
1488 Quad Line Driver Chip
1489 Quad Line Receiver Chip
74LSXX Logic Gates
Gateway PC
Borland Turbo C/C++
Null Modem Cable

**PROCEDURE:** The interface and C language test software were prepared according the pre-lab instructions that are stated in Appendix A. The external 8250 and COM1 are connected with a null modem cable.

**OBSERVATIONS AND DISCUSSIONS:** The 8250 UART (Universal Asynchronous Receive Transmit) chip was interfaced to the computer with a base address of 100H using the PB-88/4 breadboard. As shown by the schematic diagram in Figure 1, the GAL 20V8A was used to decode the base address and select the 8250 using the active low select line whenever an address match was found. (The remaining two active high chip select lines were tied to +5 Volts.) The data pins of the 8250 were connected directly to the ISA data bus lines. The /BAUDOUT pin is connect to the RCLK pin to synchronize the transmitter and receiver clocks. The 1488 and 1489 respectively convert the serial output (SOUT) and serial input (SIN) from TTL to RS-232C logic levels. A 7493 counter chip divides the clock signal on the breadboard by 4 to supply the clock signal to the 8250. Also, the /IORD and /IOWR control signals are respectively connected to the /RD and /WR of 8250. The lowest 3 bits of address bus are connected to the 3 address lines to provide function selection. Signals such as /ADS, MR, and RD are tied low to disable these function.

Figure 2 shows a listing of the test program. The test program programmed both the externally interfaced 8250 and the 8250 that comprises COM1 at a base address of 3f8h. A null modem cable connects the serial output line of one chip with the serial input line of the other chip so that data can be exchanged between 8250 chips.

After initializing the line control register (LCR) to 8 data bits, 2 stop bits, and no parity and the baud rate to 7200 Baud, the test program has 3 different options. Figure 3 gives an overview of the data flow for each of these options. The first option operates each chip in loopback mode by setting bit 4 in the modem control register (MCR) when the loopback mode is entered. This mode internally connects the serial output line to the serial input lines so that all transmitted characters are received. The second option allows for continuous transmission of a character from one 8250 to the other. The continuous function begins by making sure that both 8250s are out of loopback mode, and by clearing the Divisor Latch Access Bit (DLAB bit – bit 7 of the Line Control Register) so that both 8250s may be used for transmitting and receiving. That after a character is entered via the key board, it is continuously transmitted via the null modem from the internal 8250 to the external 8250 as the results are displayed. Then the character is transmitted in the opposite direction. This is useful for displaying the waveform on the oscilloscope and for determining when the bit stream being transmitted begins and ends.

The last option again takes both 8250s out of loopback mode. An integer is input from the keyboard, displayed, incremented, and transmitted from the internal to the external 8250. The received character is transmitted from the external 8250 to the internal 8250. The program first initializes both 8250s to 8 data bits, 2 stop bits, and no parity. The baud rate of both chips is initialized to 7200 Baud with a divisor of 10H for the internal 8250 and a divisor of 12H for the external 8250. The user selects a valid option and the appropriate procedure is called to execute this function.

**CONCLUSIONS:** Upon testing the interface, everything worked correctly except for one problem. During the execution of the loopback and increment options, the data being transmitted appeared on some chips always to be one step behind that being entered. For example, in loopback, if an ‘R’ were entered, then junk would be transmitted. If the ‘R’ was followed by an ‘m’, then the ‘R’ would be transmitted, and then the ’m’ would be transmitted after the next character was entered and so on. Similar results occurred with the increment option. The source of the problem could never be identified. A procedure to flush the receiver buffer before the test was added but did not correct the problem. This experiment designed and implemented an interface to an 8250 UART. Besides demonstrating the concept of serial I/O by displaying the transmitted waveforms on an oscilloscope, this lab also demonstrated RS-232 to TTL logic level conversion with the 1488 line driver and 1489 line receiver. The software design and development that was required to implement the test program provided a value understanding of the operation of the 8250 chip.

**SIGNATURE:** This report is accurate to the best of my knowledge and is a true representation of my laboratory results.

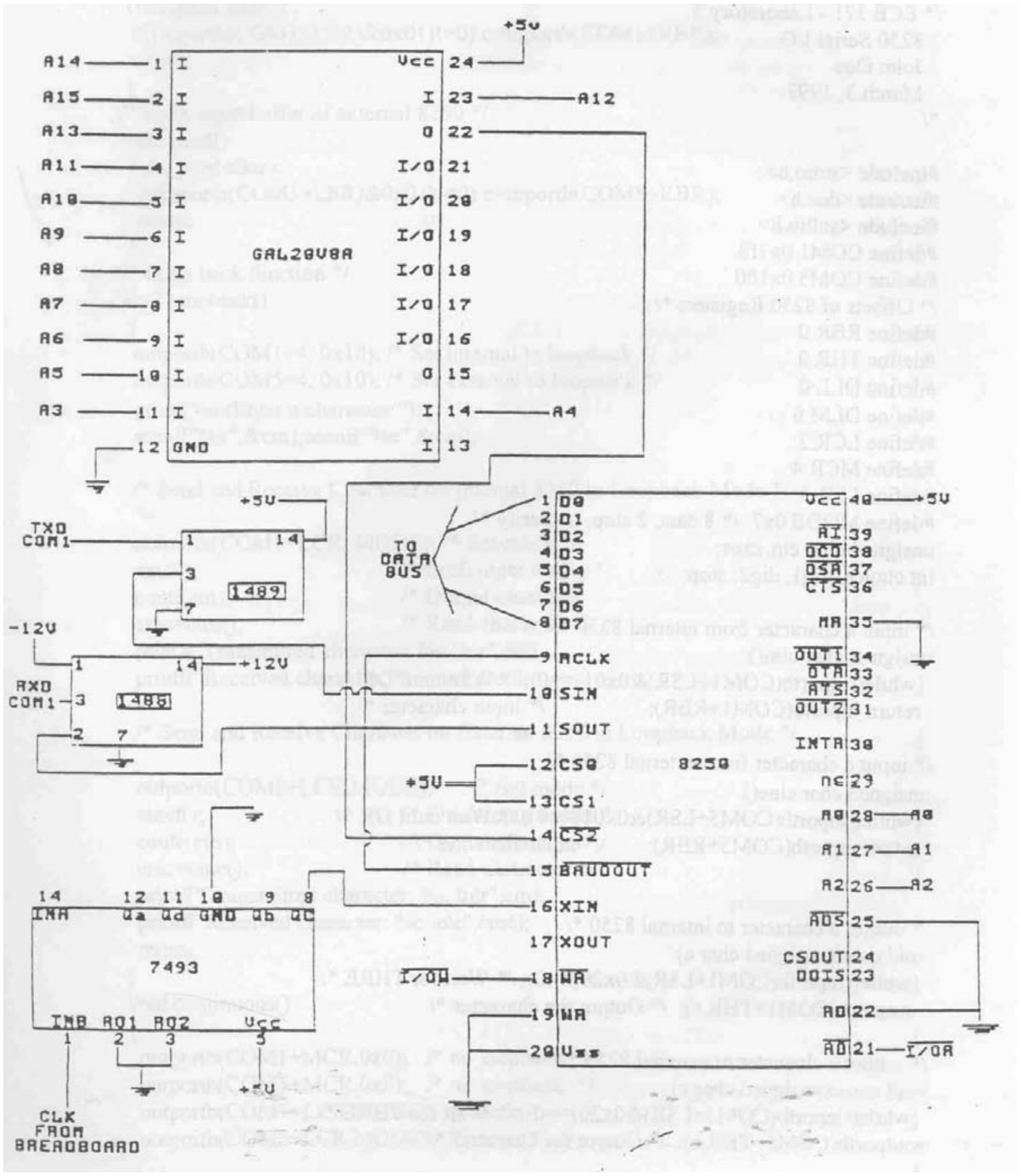


**Figure 1**
Final Schematic for 8250 Experiment

Figure 2 – Test Program for Laboratory 5

```
/* ECE 372 - Laboratory 5
8250 Serial I/O
John Doe
March 3, 1993 */
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#define COM1 0x3f8
#define COM5 0x2f8
/* Offsets of 8250 Registers */
#define RBR 0
#define THR 0
#define DLL 0
#define DLM 0
#define LCR 3
#define MCR 4
#define LSR 5
#define MODE 0x7 /* 8 data, 2 stop, no parity */
unsigned char cin, crec;
int choice, dig1, dig2, stop;
/* input a character from internal 8250 */
unsigned char cini()
{while((inportb(COM1+LSR)&0x01)==0); /* Wait until DR */
return inportb(COM1+RBR); /* input character */
}

/* input a character from external 8250 */
unsigned char cine()
{while((inportb(COM5+LSR)&0x01)==0); /* Wait until DR */
return inportb(COM5+RBR); /* input character */
}

/* output a character to internal 8250 */
void couti(unsigned char c)
{while((inportb(COM1+LSR)&0x20)==0); /* Wait for THRE */
outportb(COM1+THR,c); /* Output the character */
}

/* output a character to external 8250 */
void coute(unsigned char c)
{while((inportb(COM5+LSR)&0x20)==0); /* Wait for THRE */
outportb(COM5+THR,c); /* Output the character */
}

/* flush input buffer of internal 8250 */
void cinif()
{unsigned char c;
if((inportb(COM1+LSR)&0x01)!=0) c=inportb(COM1+RBR);
return;
}
/* flush input buffer of external 8250 */
void cinef()
{unsigned char c;
if((inportb(COM5+LSR)&0x01)!=0) c=inportb(COM5+RBR);
return;}
```

```
/* Loop back function */
void loopback()
{
      outportb(COM1+4, 0x10); /* Set internal to loopback */
      outportb(COM5+4, 0x10); /* Set external to loopback */
      printf("\n\rEnter a character:");
      scanf("%c",&cin);scanf("%c",&cin);
      /* Send and Receive Character on Internal 8250 in Loopback Mode */
      outportb(COM1+LCR, MODE); /* Set mode */
      cinif(); /* flush input buffer */
      couti(cin); /* Output character
      crec=cini(); /* Read character */
      printf("Transmitted character: %c \n\r",cin);
      printf("Received character: %c \n\r",crec);
      /* Send and Receive Character on External 8250 in Loopback Mode */
      outportb(COM5+LCR,MODE); /* Set mode */
      cinef(); /* Flush input buffer */
      coute(cin); /* Output character */
      crec=cine(); /* Read Character */
      printf("Transmitted character: %c \n\r",cin);
      printf("Received character: %c \n\r",crec);
      return;
}
void continuous()
{
      outportb(COM1+MCR,0x0); /* no loopback */
      outportb(COM5+MCR,0x0); /* no loopback */
      outportb(COM1+LCR,MODE); /* mode */
      outportb(COM5+LCR,MODE); /* mode */
      printf("\n\rEnter a character to be continously transmitted:");
      scanf("%c",&cin);scanf("%c",&cin);
      printf("Transmit from Internal to External \n\r");
      printf("Input Character Receive Character\n\r");
      for(stop=1; stop<2000; stop++)
      { couti(cin); /* Output character on Internal */
      crec=cine(); /* Input character on External */
      printf(" %c %c\n\r",cin, crec);
      }
      printf("TRansmit from External to Internal \n\r");
      printf("Input Character Receive Character\n\r");
      while (!kbhit())
      { coute(cin); /* Output character on External */
      crec=cini(); /* Input character on Internal */
      printf(" %c %c\n\r",cin,crec);
}
return;
}

void increment ()
{
      outportb(COM1+MCR, 0x0); /* Internal no loopback */
      outportb(COM5+MCR, 0x0); /* External no loopback */
      outportb(COM1+LCR, MODE); /* mode of internal */
      outportb(COM5+LCR, MODE); /* mode of external */
      printf("\n\rEnter a digit to be incremented:");
      scanf("%d",&dig1); scanf("%d"&dig1);
```

```
	dig2=dig1;
	printf("\n\rValue entered: %d\n\r",dig1);
	dig1=dig1+1;
	couti(dig1); /* Transmit on Internal */
	dig1=cine(); /* Receive on External */
	printf("INCREMENTED, TRANSMITTED FROM INTERNAL TO EXTERNAL \n\r");
	printf("Value received: %d\n\r",dig1);
	dig2=dig2+1;
	coute(dig2); /* Transmit on External */
	dig2=cini(); /* Receive on Internal */
	printf("INCREMENTED, TRANSMITTED FROM EXTERNAL TO INTERNAL \n\r");
	printf("Value received: %d\n\r",dig2);
	return;
}

void main()
{
	/* Initialization Baud Rate */
	outportb(COM1+LCR, 0x87); /* Set internal DLAB */
	outportb(COM5+LCR, 0x87); /* Set external DLAB */
	outportb(COM1+DLM,0); /* High divisor internal */
	outportb(COM1+DLL,0x10); /* Low divisor internal */
	outportb(COM5+DLM,0); /* High divisor external */
	/* should be 0x12 */
	outportb(COM5+DLL,0x10); /* Low divisor external */
	choice=0;
	while (choice != 4)
	{	printf("1. Loopback Mode \n\r");
		printf("2. Continuous Transmission \n\r");
		printf("3. Enter value and increment\n\r");
		printf("4. Exit \n\r");
		printf("Enter choice: ");
		scanf("%d",&choice);
		while(!((choice==1)||(choice==2)|| (choice==3)||(choice==4)))
		{	printf("\n\rPlease re-enter:\n\r");
			scanf("%d",&choice);
		}
		if (choice==1) loopback();
		else if (choice==2) continuous();
		else if (choice==3) increment();
		else return;
	}
}
```

Figure 3 – Flow of Data

Loopback Test

KB => Int. Xmtr. => Int. Rcvr. => Display
KB => Ext. Xmtr. => Ext. Rcvr. => Display

Increment Test

KB => Int. Xmtr. => Ext. Rcvr. => Display
KB => Ext. Xmtr. => Int. Rcvr. => Display

Continuous Test

Do 2000 Times

KB => Int. Xmtr. => Ext. Rcvr. => Display

Do until Keyboard Hit

KB => Ext. Xmtr. => Int. Rcvr. => Display

# INTRODUCTION TO C

## FORMAT OF A C PROGRAM

```
int main()
{
	variable declarations; /* Comments are embedded between these sequences */
	program statements; // This is also a comment
}
```

Variable declarations are in the form:

*type* **name**;

*type* can be *int* if the variable is holding an integer value, *char* if it is a character variable, or *float* if this is a floating-variable. Unlike Basic and Fortran all variables in C must be declared explicitly. **Note that floats take a very long time to calculate on the Freescale MC9S12 chip, and should not be used.**

Statements can be simple or compound. All simple statements are terminated by a semicolon. Compound statements are one or more simple statements surrounded with braces and are treated as a single statement by the compiler.

**SAMPLE PROGRAM**

```
int main()
{
int a,b,sum;
a=15;
b=20;
sum=a+b;
sendchar((sum/100)+0x30); /* prints the hundreds digit */
sendchar((sum/100)+0x30); /* prints the tens digit */
sendchar((sum/100)+0x30); /* prints the ones digit */
//these three commands will print out the sum.
}
```

**INPUT/OUTPUT**

The function used for output is the sendchar statement. An example of it's use is: sendchar('5');
This print statement in the above sample program will produce the output 5
There is another output command called sendstring, this can send more than one char:

```
sendstring("Hello!\n");
```

This print statement in the above sample program will produce the output

```
Hello!
```

The “\n” is the next line character. Similarly \t is the tab character and \r is the return character.

The function getchar() is used to input a char and returns its value. An example is a=getchar();

This will read a single character and assign this to the variable a.

**ARITHMETIC OPERATORS**

| | |
|---|---|
| + | addition |
| - | subtraction |
| * | multiplication |
| / | division |
| % | modulo 12%5=2 |
| ++ | increment |
| -- | decrement |

**RELATIONAL OPERATORS**

| | |
|---|---|
| == | equal to. Note that assignment is = |
| <= | less than or equal to |
| >= | greater than or equal to |
| != | not equal to |
| < | less than |
| > | greater than |

**LOGICAL OPERATORS**

| | |
|---|---|
| ! | not |
| && | logical and |
| \|\| | logical or. |

**BITWISE OPERATORS**

| | |
|---|---|
| & | bitwise and |
| \| | bitwise or |
| ^ | bitwise xor |
| ~ | bitwise complement |

Bitwise operators are used for masking operations.

**CONDITIONAL EXECUTION**
**The if statement:**

If the condition is logically true, then the statement that follows is executed. The if statement syntax is:

if (condition) statement;

if (condition) {statment1; …, statement;}

**Example:**

```
if(month==2)
{
	if((year%4)==0) /* leap year */
	{
		sendstring("Leap year \n");
		sendstring("max days in this month: 29\n");
	}
	if((year%4)!=0) /* not a leap year */
	{
		sendstring("max days in this month: 28\n");
	}
	sendstring("February \n");
}
```

**The if-else construct:**

```
if(condition) statement1;
else statement2;
```

**Example:**

```
if(month==2)
{
	 if(year%4==0)
	{
		sendstring("Leap year \n");
		maxday=29;
	}
	else
	{
		sendstring("Not a leap year\n");
		maxday = 28;
	}

	sendstring("Febrary\n");
}
```

**if-else-if construct:**

```
if(a is in first range) action1;
else if(a is in second range) action2;
else if(a is in third range) action3;
else default action;
```

**Switch-case statement:**

This is used when an expressions value is to be checked against several values. If a match takes place, then the appropriate action is taken. The switch case syntax is as follows:

```
Switch(expression)
{
        case constant;
        statement;
        statement;
        break;

        case constant;
        statement;
        break;

        default;
        statements;
}
```

The expression’s value is checked against each of the specified cases and when a match occurs, the statements following the case are executed. When a break statement is encountered, control proceeds to the end of the switch-case statement. The statements following the default case are executed if none of the other cases are matched.

**Example**

```
switch(file_type)
{
        case ‘d’:
        sendstring(“File type is ‘data’\n”);
        break;

        case ‘c’:
        sendstring (“File type is ‘commands’\n”);
        break;

        default:
        sendstring (“Unknown file type \n”);
}
```

**The goto statement:**

The goto statement needs a label, declared as:
label:
The goto statement is formally defined as
goto label;

The use of the goto statement is usually discouraged and represents bad programming.

**ITERATION**
**The while statement**

```
while (condition) statement;
```

Note that there should be no semicolon following the closing parenthesis, otherwise the compiler will assume the loop body consists of a single null statements. This will result in an infinite loop because the value of the condition will not change within the body of the loop.

**Example**

```
i=0;
while (i<10)
{
        sendstring(“The value of i is”);
        sendchar(i+0x30); //print the ascii code of a number
        i++;
}
```

**The for statement**

```
for(expression1; condition; expression2) statement;
```

**Example**
```
for(i=0; i<10; i++)
{
        sendstring(“The value of i is”);
        sendchar(i+0x30); //print the ascii code of a number
}
```

**FUNCTIONS**

In addition to the standard library functions, users can write their own functions.
Functions are defined as follows:

```
type_of_return_value function_name(argument_type argument_name) definition_of_arguments;
{
        definition of local variables;
        executable statements;
        return(value);
}
```

For example, a function to sum two numbers and return the sum to the main
program is written as follows:

```
int main()
{
             int sum, a, b;
            sum = addem(a,b);
}

int addem(int arg1, int arg2)
{
            int tot;
            tot = arg1+arg2;
            return tot;
}
```

The function addem is not declared in the calling function (*main()*) because if the return value is an integer function does not have to be declared. If the return value was a floating point number or the function returns nothing (declared as void), then the function has to be declared in the calling function.

**POINTERS**

In the C language, each variable and character string is stored in memory and has an address that describes its location. Pointers are variables that contain addresses and are declared as follows:

*type *variable_name;*

For example, a pointer to a character variable is declared as follows:
*char *cptr;*

where the pointers name is cptr and the value is a pointer to a character. To obtain a pointer to a variable, you can use the “&” operator.

```
int z,*iptr;
iptr = &z; /* iptr is a pointer to z */
```

**Example**

```
main()
{
            char x, *iptr;
            iptr=&x;
            sendstring(“Enter a character:”);
            *iptr=getchar();
            sendstring(“The value entered is ”);
            sendchar(x);
}
```

Note that in the above example, we can find the value of x through either of the following expressions, x, *iptr

**ARRAYS**

Single and multidimensional arrays are available in the C language and are used to group sets of like objects. A single dimensional array is declared as follows:

```
type array_name[n];
```

In the C language, subscripting begins with zero. Arrays are stored internally as a contiguous set of certain data types. Two dimensional arrays are declared as *array_name[m][n]* and accessed as *array_name[3][19]*.

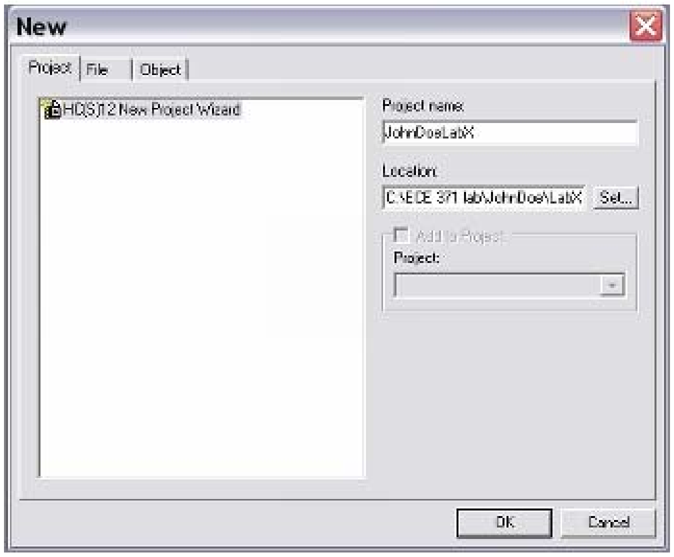
If the array name is used by itself as an expression, the value of the expression is the array's starting address.

```
main()
{
        char carray[5];
        int loopvar;
        sendstring(“Enter a 5 characters \n”);

        for(loopvar=0; loopvar<5; loopvar++)
        {
                carray[loopvar]=getchar();
        }
        sendstring(“The array contains: ”);
        sendstring(carray);
}
```

## Starting Up the Codewarrior IDE

- To launch **CodeWarrior**, double-click on the CodeWarrior IDE icon on the desktop.
- Create a new project:
  - From the IDE main menu bar, select File -> New. A new window will appear as in Figure 1.
  - Select HC(S)12 New Project Wizard.
  - Type the name you want to give your project and select the location you want to project to be situated.
  - Click **OK** to move ahead to the next window.



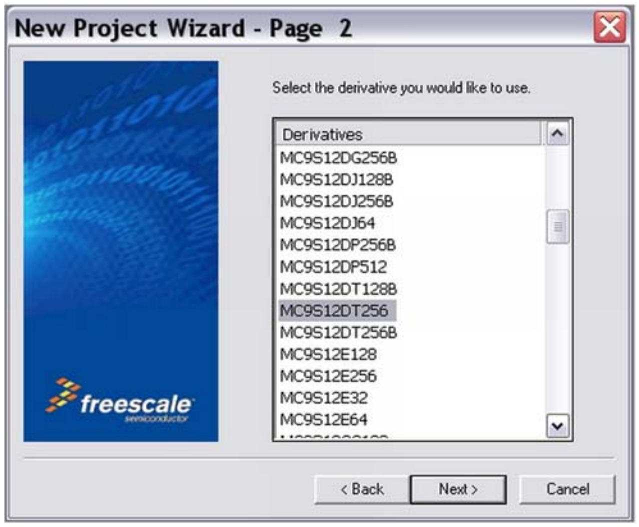
Project name:
JohnDoeLabX

Location:
C:\ECE 371 lab\JohnDoe\LabX

Project:

Figure 1: New Project Window.

- Select the appropriate microprocessor derivative. In ECE 372, we will be using MC9S12DT256 as shown in Figure 2. Click **Next** to move ahead.

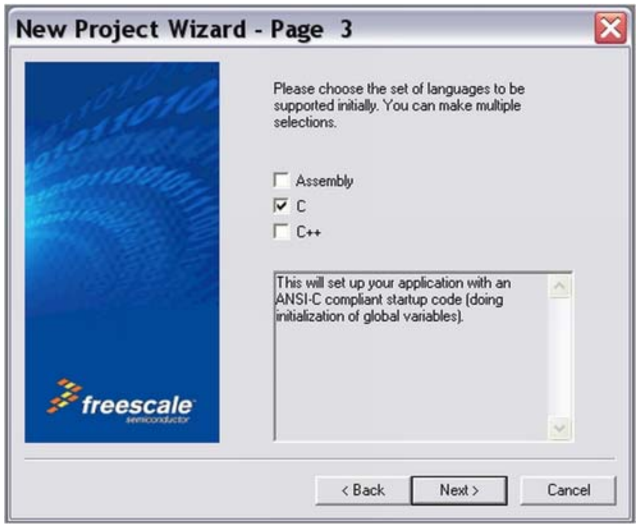


Select the derivative you would like to use.

| Derivatives |
|---|
| MC9S12DG256B |
| MC9S12DJ128B |
| MC9S12DJ256B |
| MC9S12DJ64 |
| MC9S12DP256B |
| MC9S12DP512 |
| MC9S12DT128B |
| MC9S12DT256 |
| MC9S12DT256B |
| MC9S12E128 |
| MC9S12E256 |
| MC9S12E32 |
| MC9S12E64 |

Figure 2: Derivative Microprocessor.

- Select the programming language to be used as shown in Figure 3. Note that we will be using C.


New Project Wizard - Page 3

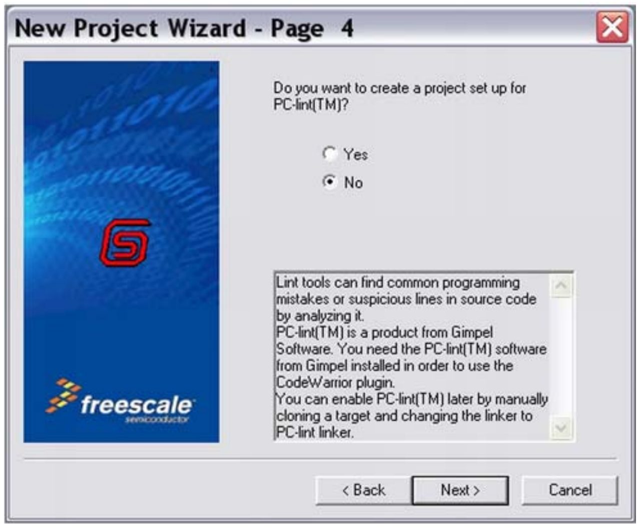
Please choose the set of languages to be supported initially. You can make multiple selections.

☐ Assembly
☑ C
☐ C++

This will set up your application with an ANSI-C compliant startup code (doing initialization of global variables).

Figure 3: Programming Language.

- On page 4 of the Wizard, select **No** as shown in Figure 4. You do not want your project to be configured in PC-Lint.


New Project Wizard - Page 4

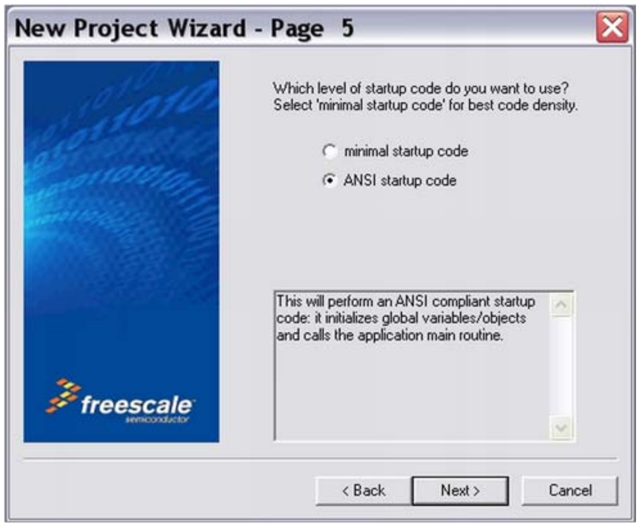
Do you want to create a project set up for PC-lint(TM)?

Yes [✓ No]

Lint tools can find common programming mistakes or suspicious lines in source code by analyzing it.
PC-lint(TM) is a product from Gimpel Software. You need the PC-lint(TM) software from Gimpel installed in order to use the CodeWarrior plugin.
You can enable PC-lint(TM) later by manually cloning a target and changing the linker to PC-lint linker.

Figure 4: PC Lint Option

- On page 5 of the Wizard, select **ANSI start up code** as shown in Figure 5.


New Project Wizard - Page 5

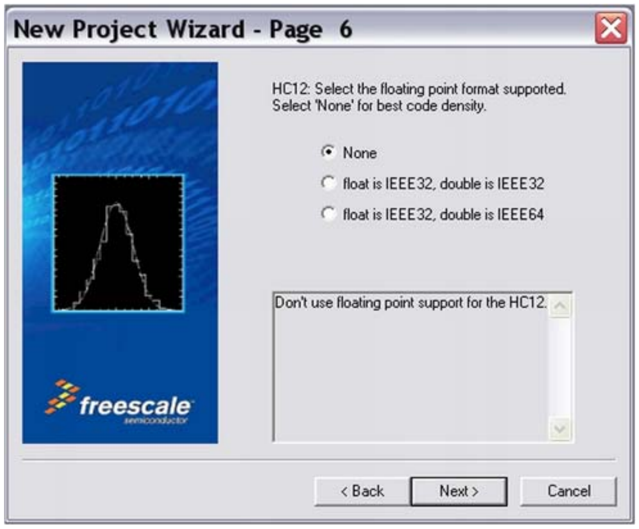
Which level of startup code do you want to use?
Select 'minimal startup code' for best code density.

( ) minimal startup code
(•) ANSI startup code

This will perform an ANSI compliant startup code: it initializes global variables/objects and calls the application main routine.

Figure 5: Startup Code.

- Select **None** for the floating point format, as the MCS12 take a very long time to compute floating point numbers, Figure 6.


New Project Wizard - Page 6

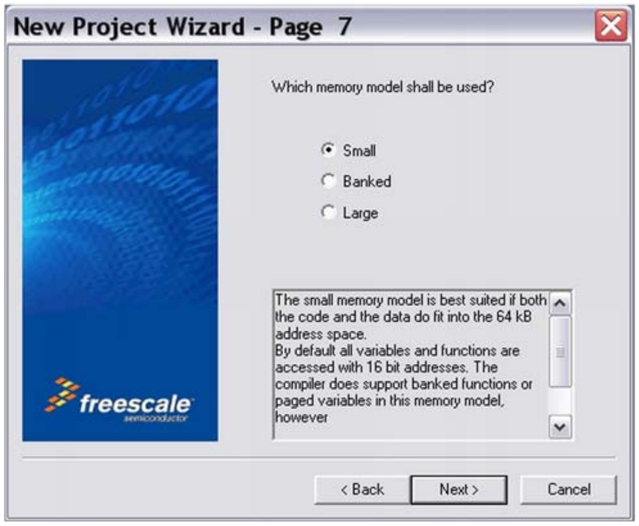
HC12: Select the floating point format supported.
Select 'None' for best code density.

(•) None
( ) float is IEEE32, double is IEEE32
( ) float is IEEE32, double is IEEE64

Don't use floating point support for the HC12.

Figure 6: Floating Point.

- Select the **Small** memory model to be used for the ECE 372 projects, Figure 7.

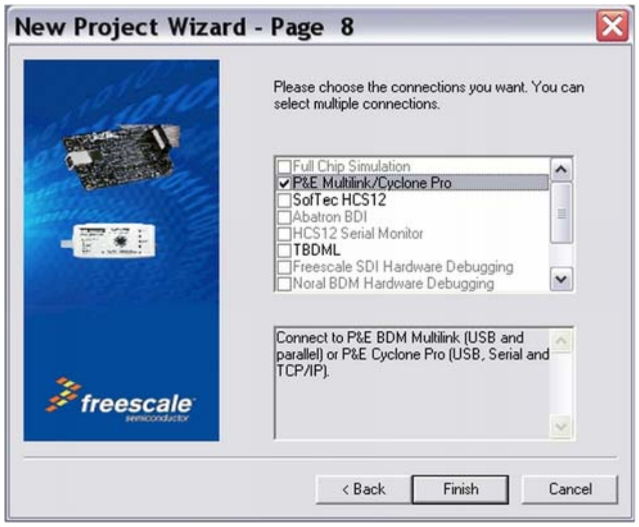

New Project Wizard - Page 7
Which memory model shall be used?
Small
Banked
Large
freescale semiconductor
< Back
Next >
Cancel

(•) Small
( ) Banked
( ) Large

The small memory model is best suited if both the code and the data do fit into the 64 kB address space.
By default all variables and functions are accessed with 16 bit addresses. The compiler does support banked functions or paged variables in this memory model, however

Figure 7: Memory Model.

- The next option allows you to choose the type of connections the project should be configured to support. Choose the **P & E Multilink/Cyclone Pro** option for your projects as shown in Figure 8. Click on **Finish** to end the Wizard.


New Project Wizard - Page 8
Please choose the connections you want. You can select multiple connections.
freescale semiconductor
< Back
Finish
Cancel

- [ ] Full Chip Simulation
- [✓] P&E Multilink/Cyclone Pro
- [ ] SofTec HCS12
- [ ] Abatron BDI
- [ ] HCS12 Serial Monitor
- [ ] TBDML
- [ ] Freescale SDI Hardware Debugging
- [ ] Noral BDM Hardware Debugging

Connect to P&E BDM Multilink (USB and parallel) or P&E Cyclone Pro (USB, Serial and TCP/IP).

Figure 8: Connection.

## National Instruments ELVIS

The National Instruments ELVIS system will be used in ECE 372. For more information on how the ECE Department uses the NI ELVIS, as well as a brief tutorial, please refer to the ECE department website at http://www.ece.clemson.edu and under “Resources > Course Lab Manuals” using the left-hand navigation links.

Additionally, ECE 372 will be using the Freescale MCU Project Board Student Learning Kit, which is essentially a Prototyping Board with a microcontroller interface. A brief explanation about this kit can be found at the ECE department website at http://www.ece.clemson.edu and under “Resources > Course Lab Manuals” using the left-hand navigation links.

The Microcontroller used is part of the Freescale HCS12 Family, model MC9S12DT256. The device data manual and user guide can be found at the ECE department website at http://www.ece.clemson.edu and under “Resources > Course Lab Manuals” using the left-hand navigation links.

Additional details can be found in the Appendix.

# Lab 1
# Introduction to the MC9S12DP256B

**OBJECTIVE**

Learn how to use the equipment and programs necessary in developing and executing any program for the Motorola chip. Furthermore, to learn the useful tools of debugging that will aid in this lab for the rest of the semester. The student will be led through a prewritten lab. They will learn the different ways of running and debugging a program using the Axiom software provided. This lab will output a count to the LED's, and show the student where in the code this happens. The student will then have to modify the code to reverse the output using the information provided in the Lab Manual and Appendices.

**EQUIPMENT**

NI ELVIS II
Freescale Project Board
Freescale Microprocessor Student Learning Kit

**PRE-LAB**

1) Read **Introduction to C** and become familiar with **Appendix A: Project Board Pin Layout**

**DURING LAB**

**1.** Turn on the NI ELVIS unit using the switch on the Project Board. (switch is located on the top right-hand corner of the unit)
**2.** Turn on the computer and log in using your Clemson username and password. There should be an icon on the bottom-left of the screen: CodeWarrior IDE. Start it up by double-clicking it.
**3.** From the CodeWarrior interface, open the project found in "C:\ece372\lab1".
**4.** From the navigation bar on the right, pull down the "sources" bar, and double-click on the file "main.c".
**5.** Read through the code and comments.
**6.** Press **"Ctrl+F7"** to compile the code. A comment should pop up on top of the code stating something like

   **Error C2801: ';' missing.**
   **main.c line 28.**

   Notice that line 28 containing the function "LCDPutString("Test\n") is missing a semi-colon. Add the semi-colon, and compile again. This time, no errors should occur.
**7.** Press the green "play" button found in the toolbar to start up the "True-Time Simulator and Debugger".
**8.** Press the green arrow button found on the toolbar to start running the code. Notice the LCD display on the right changing count, while the LEDs on the Freescale board represent the binary version of the count value.
**9.** Now modify the code to play the count backwards.

# Lab 2
# Reading and Writing Using RAM

**OBJECTIVE**
The student will learn how to read and write to internal RAM of the MC9S12DT256B Microprocessor chip. This will be accomplished by writing a program that writes a data pattern to RAM, and then reads it back onto the LCD screen. This can also be verified in the debugger memory map.

**EQUIPMENT**
Freescale PBMCUSLK Student Learning Kit

**PRE-LAB**
1) Become familiar with the memory map of the Motorola chip's internal and external memory configurations.

**DATA PATTERNS:**
Your TA for this lab will tell you which patterns to use for this lab a full week before you execute this lab:

**PROGRAM**
Every C program for this lab course must have at least three things. While most of these will be included while going through the wizard, you must ensure these commands are included in your code, without which, the program might not function as desired.

1. The first thing that every program must have is the includes for the ***hidef.h***, ***mc9s12dt256.h***, and ***pbs12dslk.h*** files. The includes should look like:

   ```
   #include <hidef.h>
   #include <mc9s12dt256.h>
   #include <pbs12dslk.h>
   ```

   **Note:** ***hidef.h*** is the header file in which macros and few definitions related to processor are done. For example the *EnableInterrupts* function used in the main function is defined in this file.
   ***mc9s12dt256.h*** is the header file in which device register definitions are set up.
   ***pbs12dslk.h*** initializes the project board hardware for direct access without having to write programs for specific port pins and bits.
   You are encouraged to read through the ***pbs12dslk.h*** file

2. The second thing to do is add the processor-specific compiler directive:

   ```
   #pragma LINK_INFO DERIVATIVE “mc9s12dt256”
   ```

   This command tells the compiler to link to processor-specific (in this case the Motorola 9s12dt256 chip) libraries and options before compiling.

3. The final check is for the main loop, which looks like,

```
void main(void){
  …
}
```

   You will notice that there is code present inside the main loop, which may or may not be required, depending on the lab parameters, so you should decide whether to delete the code or not on a case-by-case basis.

4. Write a C program that writes to the RAM. Write the data pattern the TA assigns you:

   *Your TA will tell you which range to use during this lab, not before.*

   **Hint #1:** The best way to handle an unknown range is to create two constants at the beginning of the program, a starting and ending address. Simply refer to these two constants when you write your program. When you get to lab, you can simply change those two constants, and recompile your program in lab now that you know the values.

   **Hint #2:** Write to memory using the _P command. You can write its definition as:

```
#define _P(ADDRESS)*(unsigned char volatile *)(ADDRESS)
```

5. Once the pattern is written, it should be read back from memory. Write a loop to read all the bytes written to in Step 4. Use the `LCDPutChar(…)` command to output to the LCD screen. Remember, that since the LCD screen is only 8 characters wide, you will have to refresh the display after 8 characters are written. To learn more about the LCD manipulation commands that are available, read the ***lcd.h*** and ***lcd.c*** files.

6. Also ensure that there is a delay after the printing of each character to ensure they can all be read by you as they are being printed out.

**DURING LAB**

In lab, write, compile and load the program after you enter in the addresses your Lab TA gives you. Load the program into the debugger and run it. In the debugger, go to the address range that you wrote to. Show the TA the RAM window and the output on the LCD screen.

# Lab 3
# Application of a Digital Latch

**OBJECTIVE**
Each student will interface 4 bits of input to 4 bits of output. The input consists of 4 DIP switches that will be fed into a 74LS374 flip-flop as shown in Figure 3.1. The switches will serve as the input signals that will be read by Port T on the MPU. Port T will then output to the 74LS374 chip and it will latch in either ground (logic 0) or high (logic 1). The output will be the 4 LED's using Port B.

**EQUIPMENT**
Freescale PBMCUSLK Student Learning Kit
74LS374 Flip-Flop

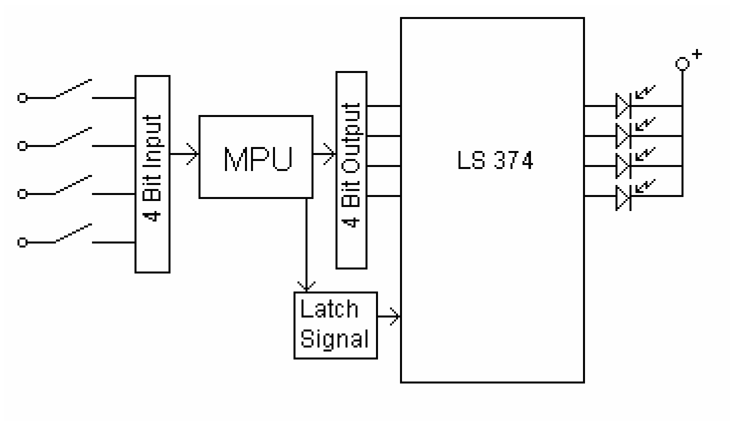


**Figure 3.1**

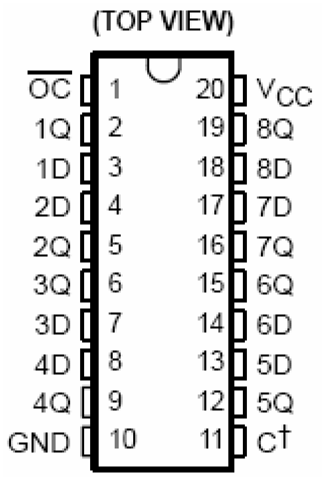
**PRE-LAB**

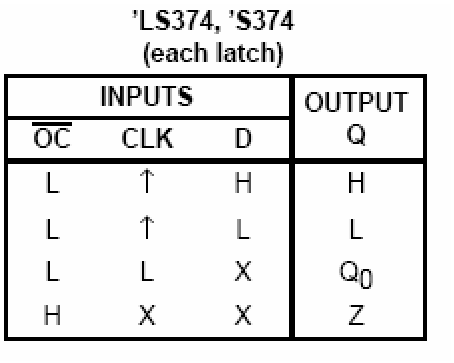
**1)** Consult the datasheet for the 74LS374 flip-flop to learn about the pins that will be used in this lab, and their possible configurations.

**2)** Design a circuit that will meet the following specifications:
• The 4 DIP switches are connected to the input port
• The Output Port is connected to a 74LS374 Data Flip-Flop device
Note: the Flip-Flop should always have its output control enabled (tied to GND)
• Connect the Latch Signal from the Motorola Chip to the input Clock pin of the flip-flop.

**3)** Develop the flow chart to satisfy steps 1 through 5 of the program section. Your TA will select one of the following options for you to use a full week before this lab:
___ Option A: Pins 0 through 3 of Port T will act as the input port and will be connected to the DIP switches.
-OR-
___ Option B: Pins 4 through 7 of Port T will act as the input port and will be connected to the DIP switches.

4) Draw the schematic of your circuit to be graded during lab (keep it neat) Label the exact pins that will be connected on the MPU and 74LS374. Use Figure 3.2 to wire the 74LS374. Figure 3.2 came from http://www.digikey.com/. Try going to DigiKey and typing in 74LS374 in the part number search. Click on one of the icons on the far right, and click on one of the Datasheets (NOT the DigiKey US Catalog sheets).




'LS374, 'S374
(each latch)

| INPUTS | | | OUTPUT |
|---|---|---|---|
| $\overline{OC}$ | CLK | D | Q |
| L | ↑ | H | H |
| L | ↑ | L | L |
| L | L | X | $Q_0$ |
| H | X | X | Z |

Figure 3.2

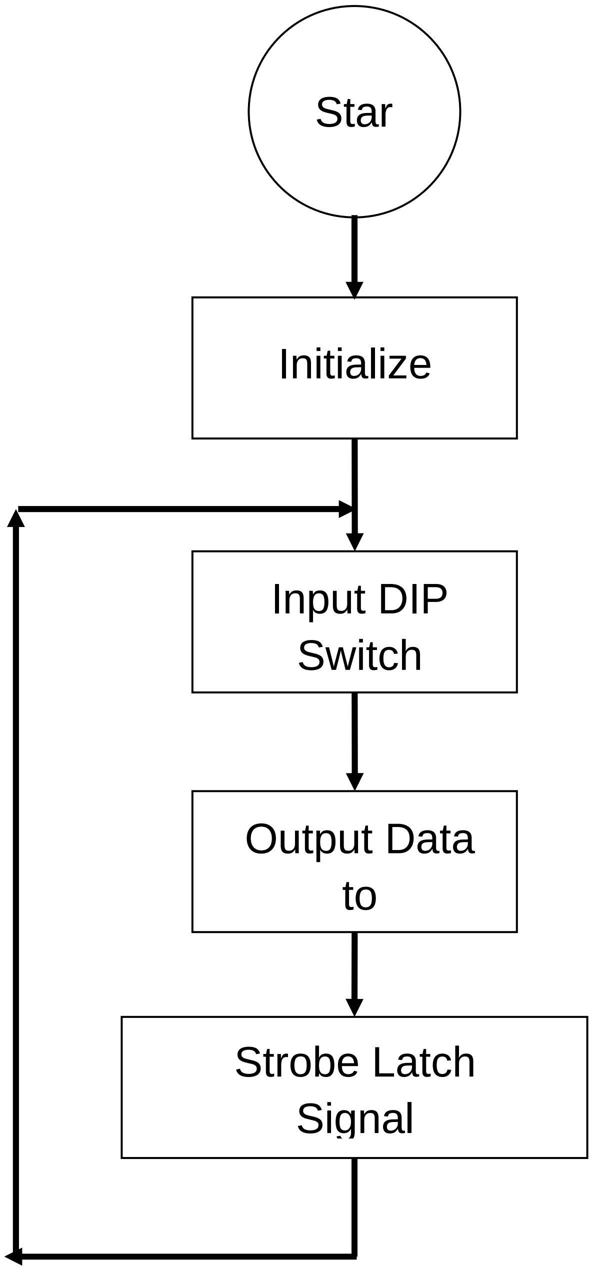
Latch Signal: Your TA will tell you which **ONE** pin to use as the Clock input for the Flip-Flop once this lab has started, not before:

| | | |
|---|---|---|
| __ Port J Pin 6 | __ Port J Pin 7 | __ Port M Pin 0 |
| __ Port M Pin 1 | __ Port M Pin 2 | __ Port M Pin 3 |
| __ Port M Pin 4 | __ Port M Pin 5 | __ Port P Pin 0 |
| __ Port P Pin 1 | __ Port P Pin 2 | __ Port P Pin 3 |

**PROGRAM**

1. Your program should set up the data directions for the input, output, and Latch Signal (clock) pins of the 74LS374.
2. When you press a key on the keyboard in the monitor (detect using getchar()) the DIP Switch settings should be read in on the input port.
3. The DIP Switch data should then be put on the output port. The Latch Signal pin should then strobe high, and then the data should be removed from the output port (set to all zeros). The data is removed to simulate a bus in an actual system. The data must be removed so that other devices can theoretically use the "bus".
4. The DIP switch value should be outputted to the monitor in binary followed by an end-of-line (\n).
5. If the return value of getchar() is 'i', make the program go into an infinite loop of steps 3-4. For every other return value of getchar(), execute steps 3-4 only once.

**FLOWCHART**



**DURING LAB**

1) Connect the specified circuit. Connect the Data lines directly to the 4 LEDs.
2) Run the program. Set the DIP switches to various values, and press any key in the monitor. Verify the circuit works and that the correct values are printed out.

# Lab 4
# Interrupts

## OBJECTIVE

This laboratory requires the student to write a program that uses interrupts to handle the input of a switch. The student will implement an interrupt by setting up an interrupt handler, setting and restoring interrupt vectors, and masking and unmasking interrupts. A NAND gate debouncer will be used to stabilize the input.

## EQUIPMENT

Axiom Processor and Project Board Set
7400 NAND Gate
DPST Switch
Two resistors (about 8.2KΩ) (Gray, Red, Red)

## REGISTERS OF INTEREST

DDRP, PERP, PPSP, PIEP, PIFP, PORTB, DDRB

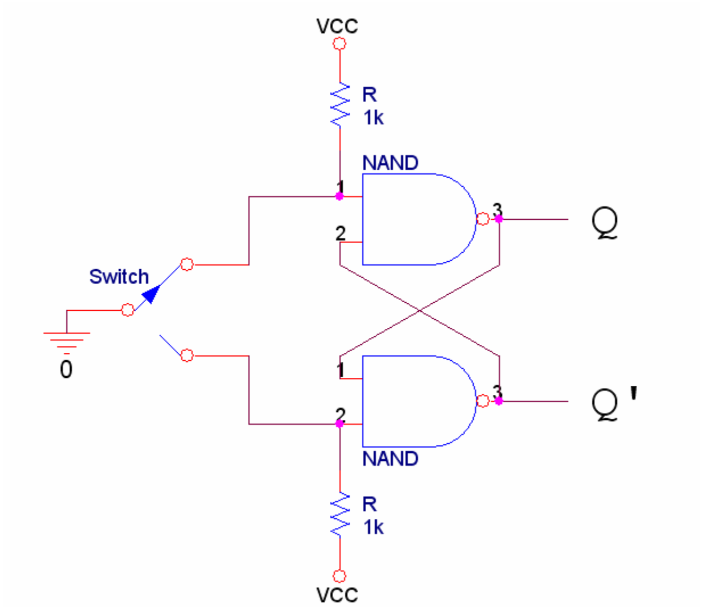


**Figure 4.1 Nand Gate Debouncer.**

## PRE-LAB

1) Design a circuit that will interface with two pins of Port P as interrupts. The NAND gate latch seen in Figure 4.1 will be used to prevent bouncing (a phenomenon common to any type of switch where the signal "bounces" rapidly between logic 1 and 0). Simple identify one pin as "PORT P ON" and the other as "PORT P OFF".

2) Set up the program to satisfy steps 1 through 7 of the program section. When writing a program without knowing which pins on port P will be used, it is best to create two constants. An example is one called "PTPON" and the other "PTPOFF". If one of the pins you used was Pin 3 of Port P, you can simply set that constant's value to '0x08'

when you get to lab. In your program, whenever you mask the pins, you refer to these constants.

**PROGRAM**

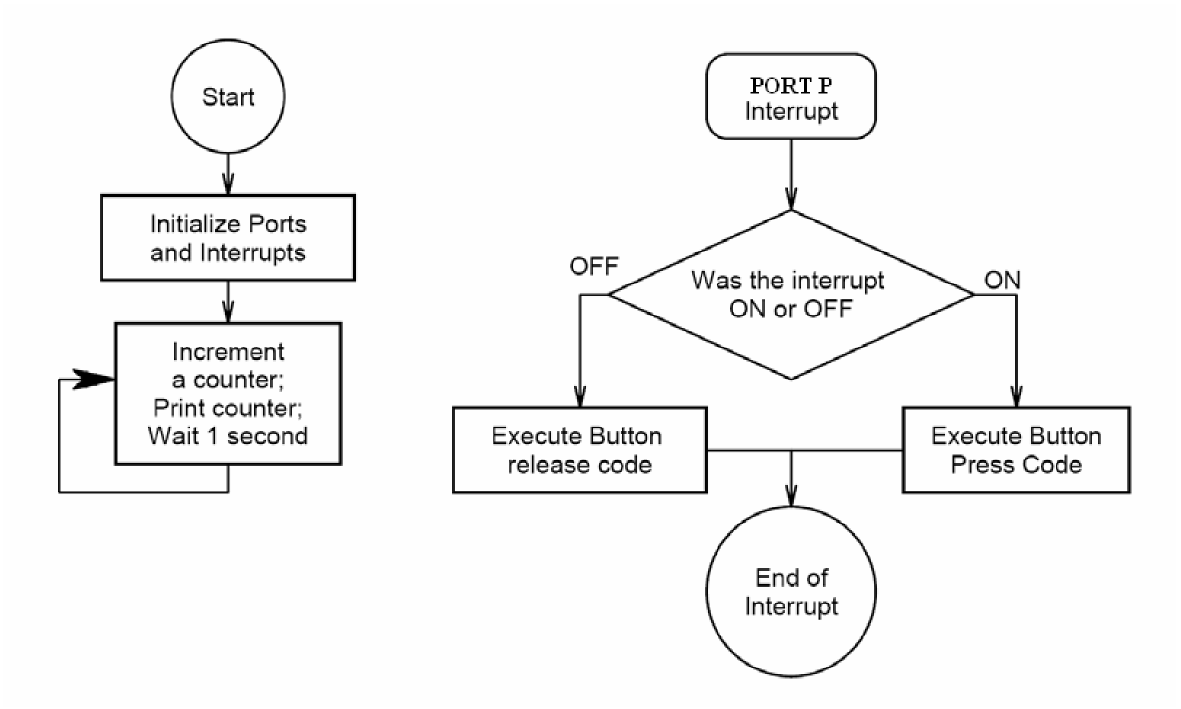
1. Pin ___ of Port P will be known as the ON pin. Pin ___ of Port P will be known as the OFF pin. The TA will tell you these values during this lab.
2. Your TA will select one option for you to use:
   ___ Option A: Configure the ON pin and OFF pin to be pulled down by the pulling resistors. Consequentially, an interrupt will occur on rising action.
   -OR-
   ___ Option B: Configure the ON pin and OFF pin to be pulled up by the pulling resistors. Consequentially, an interrupt will occur on falling action.
3. Write a program that will loop infinitely and follows the specifications below:
   - It will add one to a counter.

   a. Then use the `delay(1000);` command to add approximately a 1 second delay in between counts. (Since this is the first time using the delay command, please keep in mind that using the delay command makes BREAKing in the debugger hard. It may be best to RESET when you want to restart your program.)

   b. Print the last number of the count to the screen (0x30 + (count % 10))
4. You are to write the interrupt service routine for Port H. Detect if the button was pressed or released (by using the PIFH register). When you declare an interrupt service routine, you must create an ISR function that looks like:
   ```
   void interrupt <ISR Vector no.> InterruptFunction(void){
   }
   ```
   You have to find the Port ISR number from the Appendix.

5. In order for a port to interrupt properly, all the registers must be properly configured. Fin the details of the PORT P registers in the Port Integration Module of the datasheet and configure their options.
6. Write the interrupt service routines so that when the switch is flipped, the result is displayed on the LEDs, one for ON, and the other for OFF.
7. The ISR will be called when the button is pressed or released because there are two pins on PORT P set up to interrupt. The ISR must detect if the button was pressed or released (by reading and clearing the PIFH flags) and choose its course of action from there. In the interrupt service routine, add a `delay routine` command when the button is either pressed or released to show that counting has stopped.

**FLOWCHART**


Start
Initialize Ports and Interrupts
Increment a counter; Print counter; Wait 1 second
PORT P Interrupt
Was the interrupt ON or OFF
OFF
ON
Execute Button release code
Execute Button Press Code
End of Interrupt

**DURING LAB**

1) Connect the specified circuit. Note, you will have to look up the pin layout on DigiKey again. Remember that a NAND gate is a 74LS00 chip.
2) Compile and load the program.
3) Run the program. Verify that when you press the button, the output device is on, and when you release the button, it goes off.
4) Connect the oscilloscope to the output of the NAND Gate Debouncer, and demonstrate that the system works.

# Lab 5
# Keypad Interfacing

**OBJECTIVE**

This laboratory requires the student to write a program that scans the inputs of a 4 by 2 matrix keypad. The student will scan the keypad by activating one output at a time (by writing a pattern to the 4 output pins) and reading the inputs pins to see if any of the switches are pressed. When a button press is detected, it will be decoded and outputted.

**EQUIPMENT**

Freescale PBMCUSLK Student Learning Kit.
4 x 4 matrix Keypad.

**PRE-LAB**

1) There are two parts to the 2x4 matrix. The Rows and the Columns. Let the [] rows, [] columns be known as the input, and the [ ]rows, [ ]columns be known as the output.

2) Write a C program to satisfy steps 1 through 3 of the program section.

**PROGRAM**

1. Set up and initialize Port T. Set the pulling resistors to pull the inputs
[] UP -or- [] DOWN.

2. To determine if a key is pressed, you will turn on the output pins and read the input pins. You will turn on each output, one at a time, by writing a pattern. The program should check the inputs of the keypad every time you write a pattern. This is how to scan the input to see if a key was pressed.

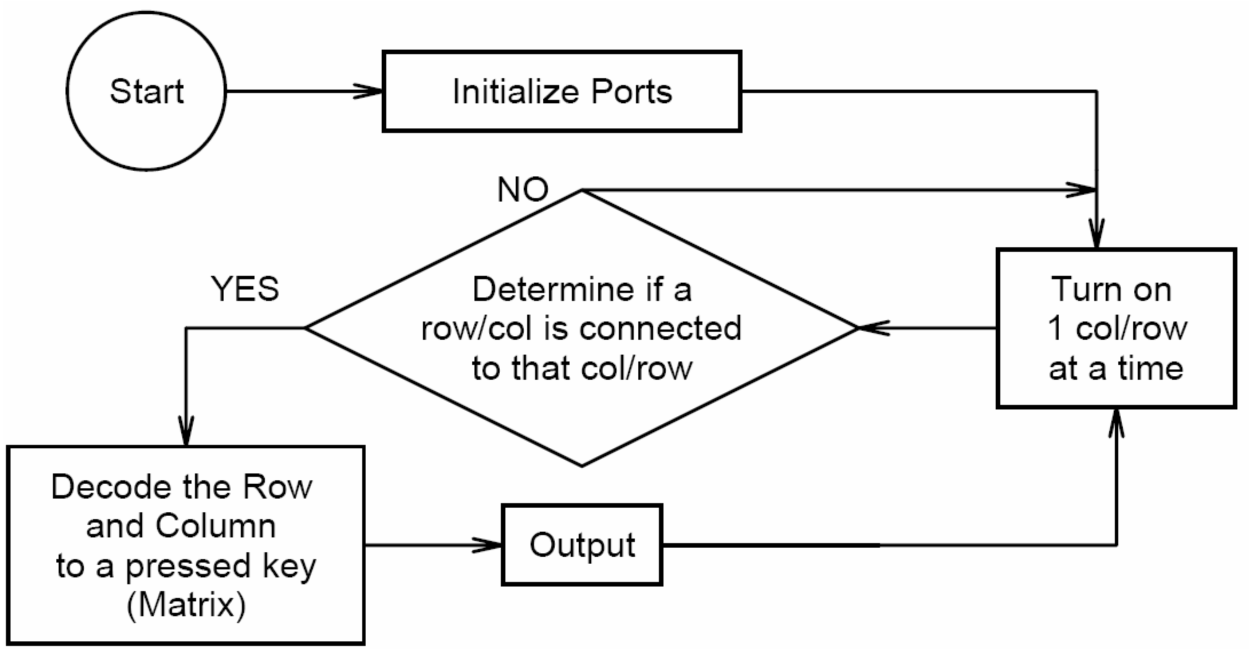
3. Once a key is detected, its value should be outputted to:
   a. __ B: in binary, output to the 4 LED's
   b. __ C: output to the LCD Display

**DURING LAB**

1) Compile and load the program.

2) Run the program. Verify that when you press the button, the output device is on and correctly displays which button was pressed.

**FLOWCHART**


Start
Initialize Ports
NO
YES
Determine if a row/col is connected to that col/row
Turn on 1 col/row at a time
Decode the Row and Column to a pressed key (Matrix)
Output

# Lab 6
# Serial Communication - SCI

## OBJECTIVE

The purpose of this lab is for the student to write a serial communication program that will run on the Axiom board and communicate with the PC. The serial port must be initialized, which includes setting up the baud rate, number of data bits, parity bit, and stop bits.

## EQUIPMENT

Freescale PBMCUSLK Student Learning Kit

## REGISTERS OF INTEREST

SCI0BDH/SCI0BDL, SCI0CR1, SCI0CR2, SCI0SR1, SC0DRL

## PRE-LAB

1) Become familiar with SCI Serial communications.

2) Write a C program to satisfy steps 1 through 3 of the program section.

## PROGRAM

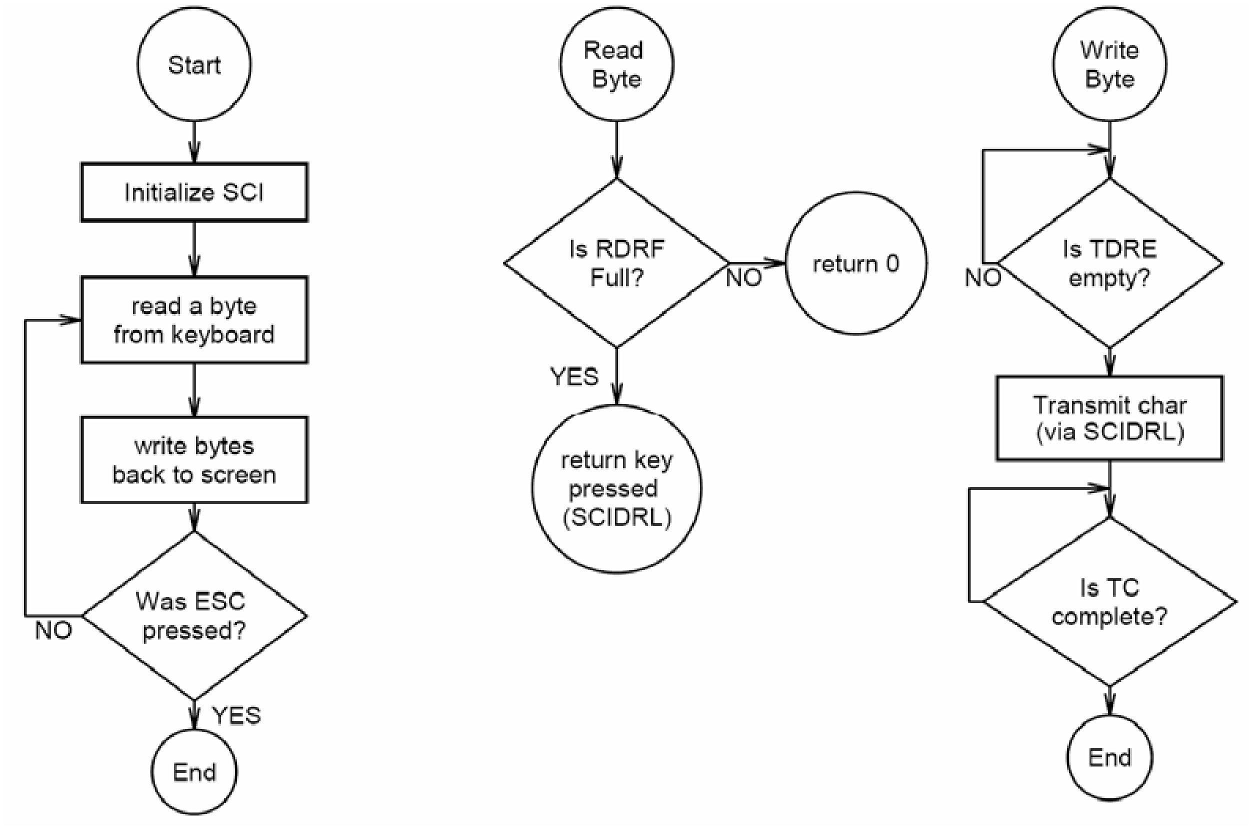
1. Initialize the SCI0 to be at 9600 Baud Rate (remembering the base frequency in the debugger is 2 MHz), 8 bits per character, 1 stop bit, and no parity. Use the Data Sheets for SCI to determine all the correct settings. These can be found on the Desktop of any ECE 372 Lab Machine (SCI.pdf in the data sheets folder) or go to http://www.freescale.com/. At the top right of the website, type in MC9S12DP256BCPV for the part number, and then click the part number from the list. Look for the SCI data sheet.

2. Make the program detect if a key has been pressed on the PC's keyboard and sent from the PC to the Axiom Board. When this happens, read in the char and do one of the following:
   A: When a letter is received, detect if it is lowercase or upper case. Then transmit (echo) back the opposite. So if a lower case key is pressed, you will echo uppercase. Special characters can be either:
   [ ]Ignored
   [ ]Replaced with a '*'
   [ ]Replaced with a ___ (a special character determined by the TA)

   B: Only detect numbers. When a number is typed, spell out that number.
   C: Only detect numbers. When a number is typed, output it in binary to the LEDs.
   D: Only detect numbers. When a number is typed, output it in binary to the LED Bar.
   E: Whenever a key is typed, the ASCII code is to be echoed back. This means that two characters will have to be sent, in hex.

3. If the ESC key (0x1B) is pressed, the program is to terminate.

**FLOWCHART**



**DURING LAB**

1) Compile and load the program.
2) Run the program. Verify that when you type, the correct output is shown.
3) In lab, your TA will tell you an additional Baud Rate to use. After verifying the 9600bps Baud Rate, modify your program to run at the new Baud Rate of [ ]300, [ ]1200, [ ]2400, [ ]4800, __________. (Only use Baud Rates below 9600).
4) Compile and Load your program again. Verify that it works at the new baud rate.

# Lab 7
# Rotary Pulse Generator

## OBJECTIVE

The purpose of this lab is for the student to write a program that will interface a rotary pulse generator (encoder). As the knob turns, the output of the encoder changes, as can be seen in Figure 7.1. Interrupt handlers will be written to detect when the levels change and adjust an internal counter.

## EQUIPMENT

Freescale PBMCUSLK Student Learning Kit
Rotary Pulse Generator
Two 8.2KΩ resistors (Gray, Red, Red)
Virtual Oscilloscope

## REGISTERS OF INTEREST

Port J and Port P data and their interrupt registers.

| Clockwise Rotation | | |
|---|---|---|
| Position | Output A | Output B |
| 1 | | |
| 2 | ● | |
| 3 | ● | ● |
| 4 | | ● |

● Indicates logic high; blank indicates logic low. Code repeats every 4 positions.

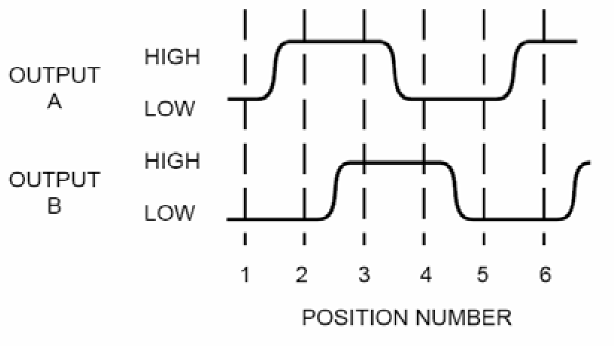


Figure 7.1

## PRE-LAB

**1)** Develop logic that will allow you to determine if the encoder is turning clockwise or counterclockwise. It should follow something along these lines:

**a)** Something just interrupted, was it Port P (Output A) or Port J (Output J)

   **b)** If it was Port P, was it a rising action (for example from state 1 to 2 in Figure 7.1) or a falling actions (for example from state 2 to 1).
   **c)** If Port P was a rising action, was Port J low? (Going from state 1 to 2) or was J high? (Going from state 4 to 3).

This should be done for all 4 possibilities. Refer to **PART 2** of the **program section** for recommended implementations of your logic.

**2)** Write a C program to satisfy steps 1 through 3 of the **program section**.

## PROGRAM

**1.** Initialize the **Port P Pin [ ]4 [ ]5 [ ]6 [ ]7** and **Port J Pin 0** to be the interruptible inputs from the encoder.

**2.** There are three suggested ways of handling the four output stages of the encoder. You can use one of the recommended implementations or come up with your own ISR.

**a)** Write **ONE ISR:**

When both Port J Pin 0 (PJ0) or Port P Pin 0 (PP0) interrupt, they execute this one ISR. Then the ISR will have to read both inputs and detect if the interrupt was PJ0 rising, PJ0 falling, PP0 rising or PP0 falling by looking at the interrupt flags. You can then determine clockwise or counterclockwise movement. Also, since you can only interrupt on ONE edge (falling or rising) you must switch the polarity of the inputs so they trigger on the opposite edge. In other words, if PJ0 just rose, it must fall next time, so set it to interrupt on falling action.

**b)** Write **TWO ISR's:**

Implement one ISR for PJ0 and another for PP0. This method of implementing the ISR's does not need to detect which port interrupted, which makes the code simpler. Continue to implement your ISR's following the recommendations in **method a)** above.

**c)** Write **FOUR ISR's:**

Implement an ISR for each of the four possible states of PJ0 and PP0. This way when PJ0 rises, it sets the polarity to detect falling action next, and changes the ISR to the "Port J FALLS" interrupt.

**3.** Start a counter at 0. As the encoder turns, increment and decrement this counter. Each time it changes, its value should be outputted to one of the following

__ A: The numbers 0-9 on the serial port (SCI)
__ B: The numbers 0 to 15 on the 4 colored LED's (in binary)

## DURING LAB

**1)** Connect the encoder as seen in **Figure 7.2**.
**2)** Compile and load the program.
**3)** Run the program. Verify that when you turn the encoder, the output changes as you would expect it to.
**4)** Attach the oscilloscope to the two outputs of the rotary pulse generator, and observe the pulses that are generated as you turn the knob. Do the waveforms match the logic you developed for this lab?

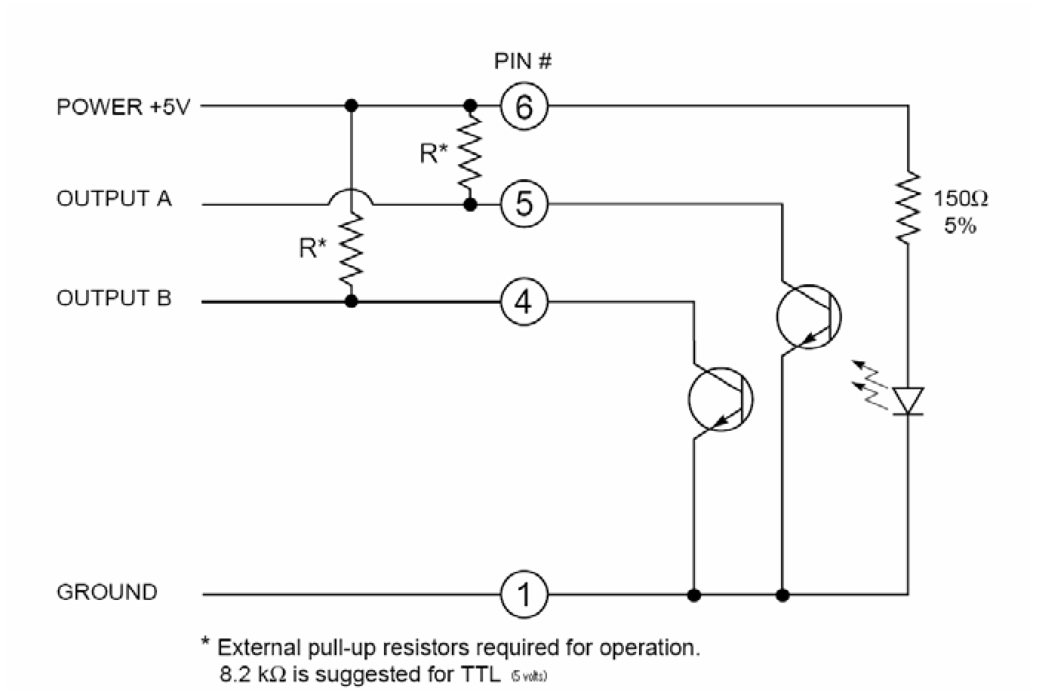

PIN #
POWER +5V
OUTPUT A
OUTPUT B
GROUND
6
5
4
1
R*
R*
150Ω
5%
* External pull-up resistors required for operation.
8.2 kΩ is suggested for TTL (5 volts)

**Figure 7.2**

## FLOWCHART

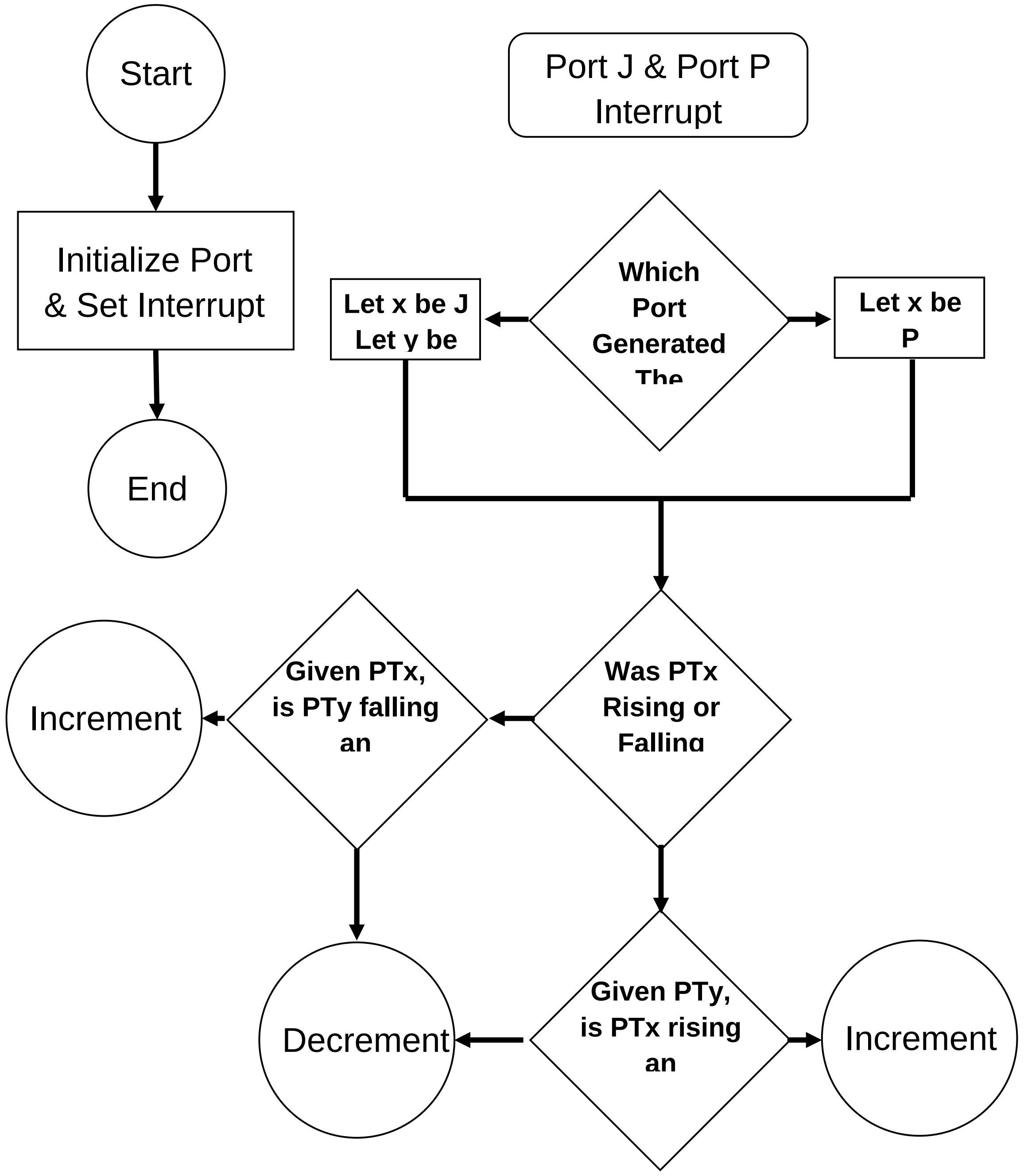

Start
Initialize Port & Set Interrupt
End
Port J & Port P Interrupt
Which Port Generated The
Let x be J Let y be
Let x be P
Was PTx Rising or Falling
Given PTx, is PTy falling an
Increment
Given PTy, is PTx rising an
Decrement
Increment

**Figure 7.3**

# Lab 8
# Clock Pulse Generator – Enhanced Capture Timer

**OBJECTIVE**
The purpose of this lab is for the student to write a serial communication program that will run on the Freescale board and communicate with a DAC.

**EQUIPMENT**
Freescale PBMCUSLK Student Learning Kit
Virtual Oscilloscope

**REGISTERS OF INTEREST**
ECT Registers: TIOS, TSCR1, TCTL1, TCNT (16 bits), TCx (16 bits) PORTT

**PRE-LAB**
**1)** Write the program to satisfy steps 1 through 3 of the **program section**.

**PROGRAM**
**1.** Initialize the Buzzer using Port T.
*Note: Don't forget to enable the main Counter. And if your TA does NOT specify using 1μ-second timing, don't do it.*

**2.** Develop code that will generate a square wave at the frequency listed in Table 8.1 for each of the notes. The square wave must be generated by loading the ECT with the appropriate values and will then be output to the speaker Using the ED1 Amp on Port J.

| **Notes** | **Frequency (Hz)** |
|---|---|
| C | 262 |
| C# | 277 |
| D | 294 |
| D# | 311 |
| E | 330 |
| F | 349 |
| F# | 369 |
| G | 392 |
| G# | 415 |
| A | 440 |
| A# | 466 |
| B | 495 |

**3.** Compose a song (OK, maybe just some random tones) using the notes coded in **PART 2**. Use the delay function to put some space between notes. Loop your program so the notes play repeatedly.

**FLOWCHART**

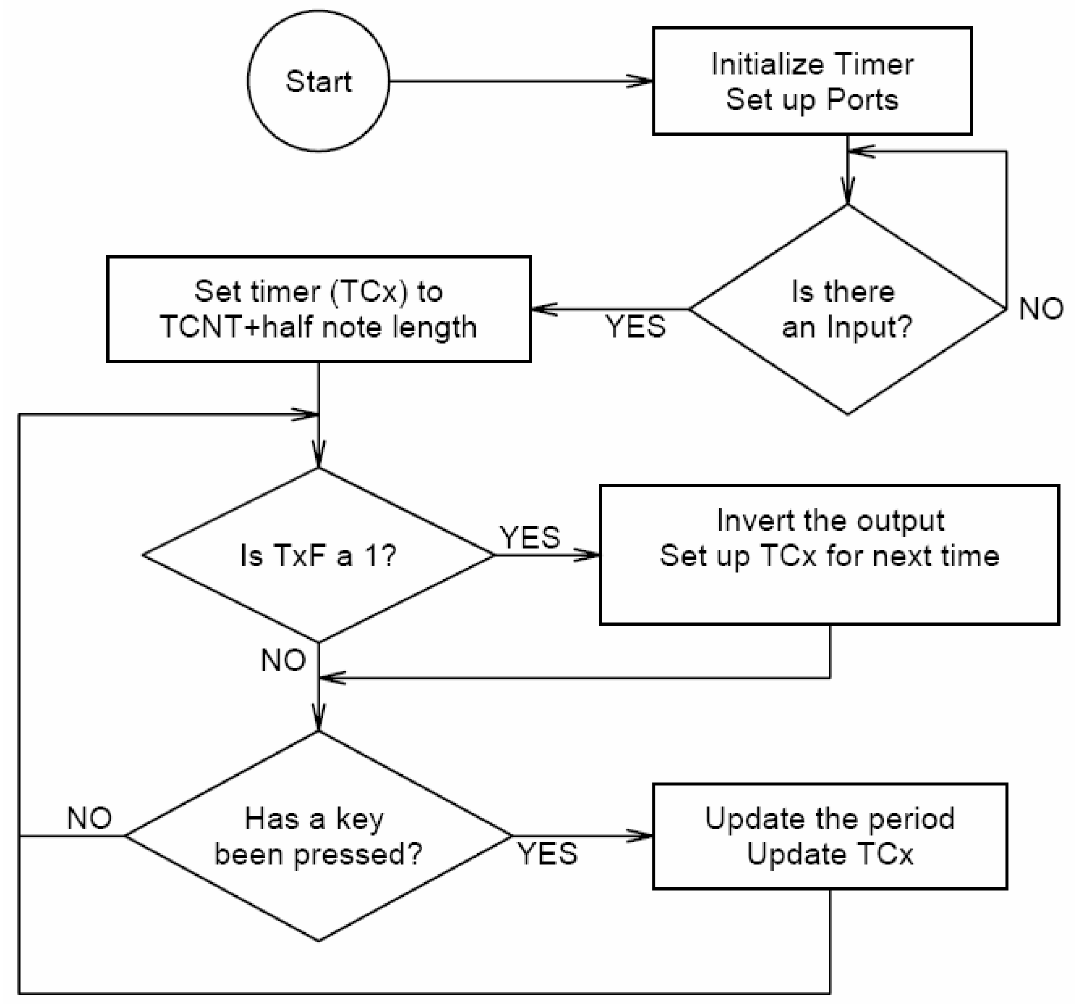


**Figure 8.1**

**DURING LAB**

1) Compile and load the program.

2) Run the program. Verify that the tones generated sound correct. Your TA will have a program that generates reference tones for verification of correct notes.

# Lab 9
# Serial Communication 2 - SPI

**OBJECTIVE**
The purpose of this lab is for the student to write a serial communication program that will run on the Freescale board and communicate with a DAC.

**EQUIPMENT**
Freescale PBMCUSLK Student Learning Kit
DAC 7512
Virtual Oscilloscope

**REGISTERS OF INTEREST**
SPI1CR1, SPI1CR2, SPI1SR, SPI1DR, SPI1BR, DDRM, PTM

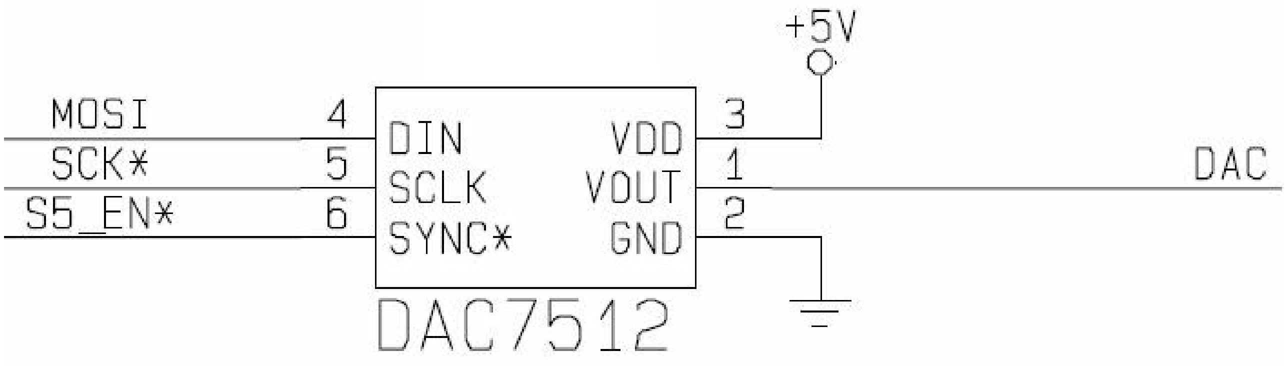


**Figure 9.1**

**PRE-LAB**
**1)** Become familiar with SPI Serial communications and the associated registers listed above.

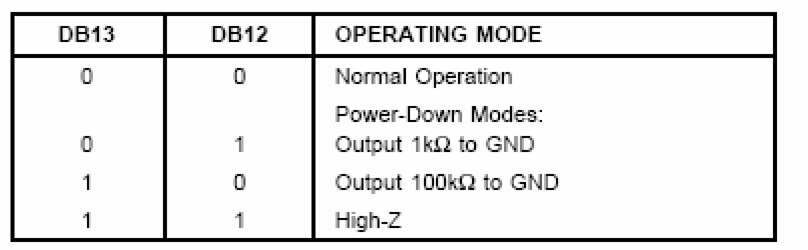
**2)** Go through the Project Board pin layout to determine the serial device to be used. Remember, the SPI channel is set by outputting to Port M bits 4 through 6. Port M directions must be initialized so that you can output these values.

**PROGRAM**
1. Initialize the SPI1 for unidirectional mode. Ensure that SPI1CR1, SPI1CR2, SPI1BR are set. Remember the base frequency is 4 MHz, thus set SPI1BR to go as fast as possible.
2. Using SPI protocol write the two byte word to the 12-bit DAC.
3. Start sending the 16-bits of data described below:

```
[xxCCMMMM LLLLLLLL]
FIRST BYTE LAST BYTE
<----16-bits of data--->
```

**xx** = two **DON'T CARE** bits
**CC** = two **CONTROL** bits (Modes of operation)
**MMMM** = four **MSB**
**LLLLLLLL** = eight **LSB**

4. Set the Control bits to run the DAC in Normal Operation by using the table in Figure 9.2.



| DB13 | DB12 | OPERATING MODE |
|---|---|---|
| 0 | 0 | Normal Operation |
| | | Power-Down Modes: |
| 0 | 1 | Output 1kΩ to GND |
| 1 | 0 | Output 100kΩ to GND |
| 1 | 1 | High-Z |

TABLE I. Modes of Operation for the DAC7512.

5. Make the program continuously output a square wave of period ___ milliseconds.

**FLOWCHART**

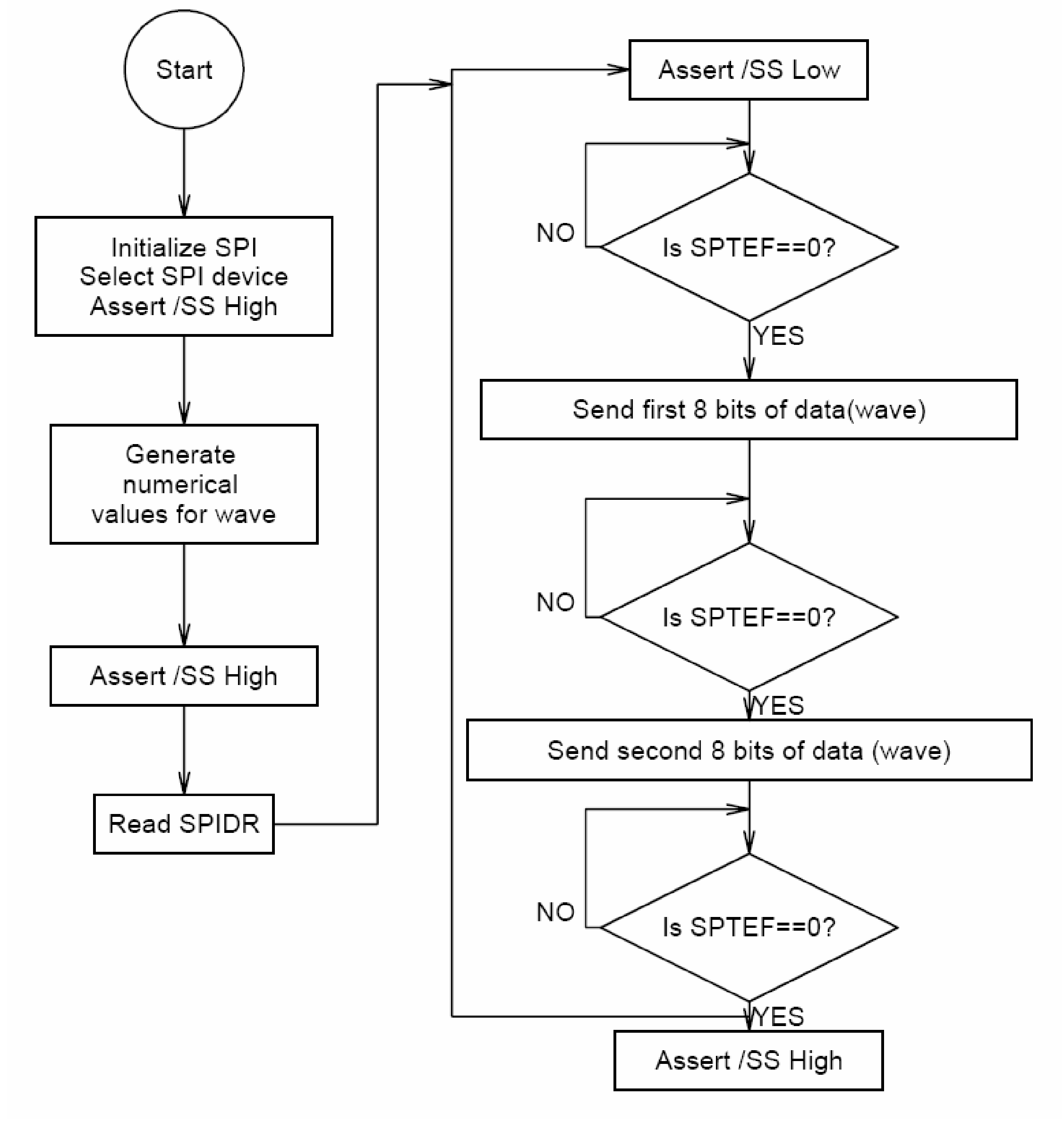


**Figure 9.2**

**DURING LAB**

**1)** Make sure you have the connections required for DAC serial communication set correctly (refer to the datasheet).

**2)** Compile and load the program.

**3)** Run the program. Verify the correct output by showing the signal on the oscilloscope. Use the oscilloscope to determine the period of the wave and sample width.

**POST-LAB QUESTIONS**

1) Was the observed frequency close to what you expected given the sample width you used?
2) If the frequency was a little lower than expected, what does that mean about the actual sample width?
3) What might add these time delays to the sample width?

# Lab 10
# A/D and D/A Conversion

**OBJECTIVE**

This laboratory requires the student to write a program will read in a signal from the Analog to Digital converter. This signal will then be manipulated and output to the Digital to Analog converter.

**EQUIPMENT**

Freescale PBMCUSLK Student Learning Kit
DAC 7512
Virtual Oscilloscope
Virtual Function Generator

**REGISTERS OF INTEREST**

ATD1CTL2, ATD1CTL3, ATD1CTL4, ATD1CTL5, ATD1DR0H, ATD1DR0L, ATD1STAT0

**BACKGROUND**

In ECE 330 you have covered Fourier Transforms and Fourier Series. The continuous world lives in the Fourier Transform domain. Simple low pass and high pass filters consist of a resistor and an inductor or capacitor. Mathematically, a low pass filter is like integrating. However when you have a discrete signal, you have to use the Fourier Series. The equivalent here is the average function.

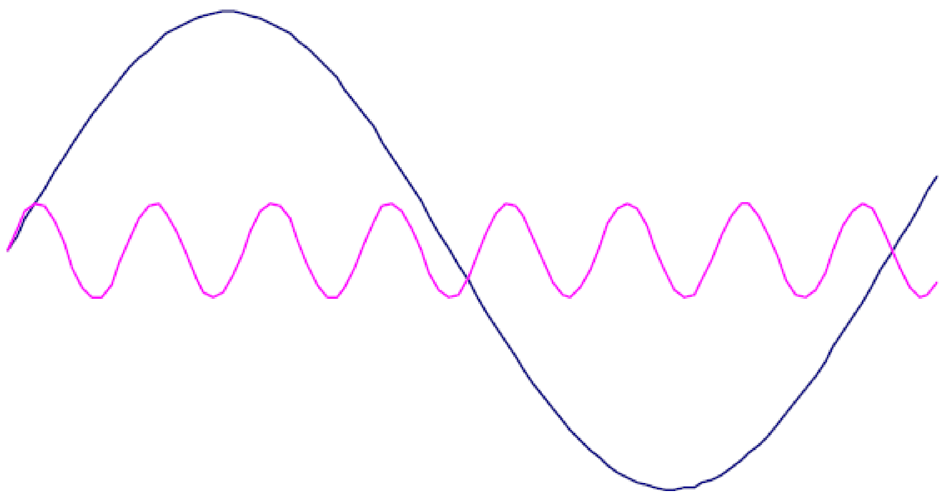


**Figure 10.1**

One way to think of this is to have a high frequency and low frequency way, as seen in Figure 10.1. There sum is Figure 10.2.

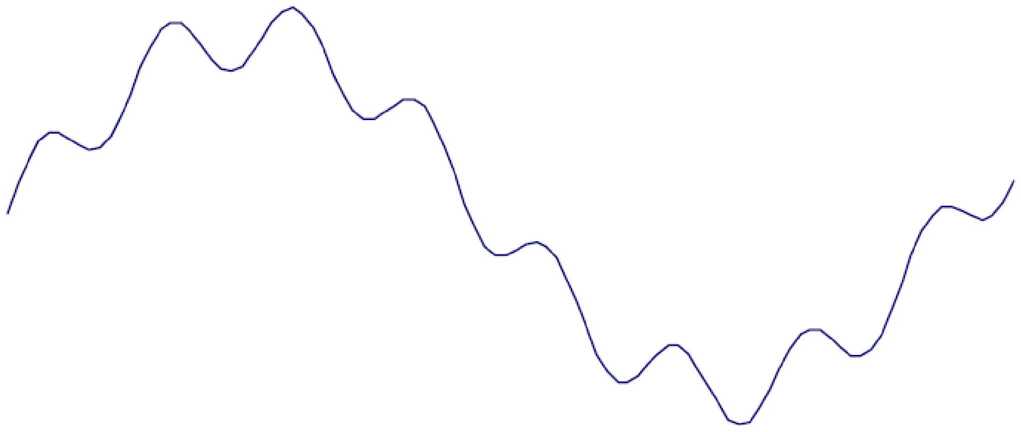


**Figure 10.2**

Using current technology such as an analog to digital converter, you can acquire a discrete version of this waveform that might look like Figure 10.3.

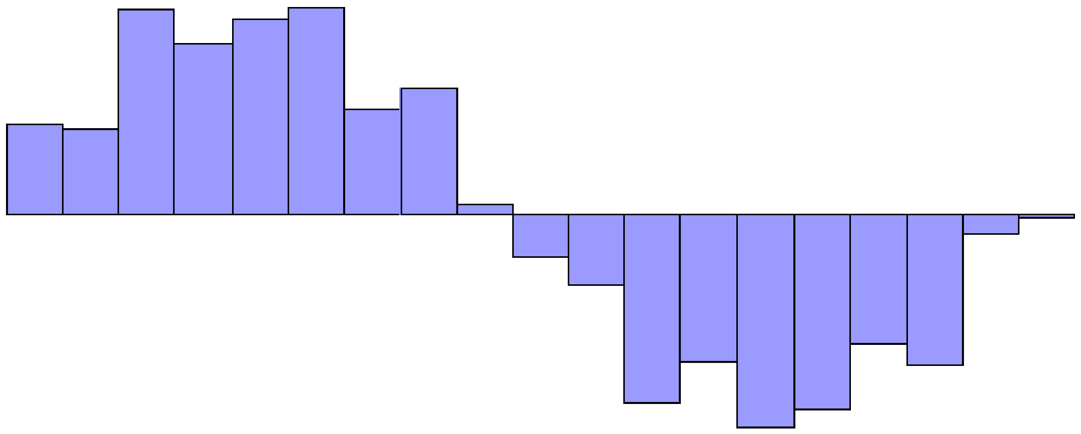


**Figure 10.3**

By averaging the discrete data, you will be smoothing it out, low pass filtering it. An example of this can be seen in Figure 10.4 where every 3 data points are averaged together. In other words, the first data point is the sum of the original 1, 2, 3; the second is 2, 3, 4; the third is 3, 4, 5; etc… As you can see, the higher frequency is removed leaving the lower frequency. The 3 data point average used here is the number that is referred to in step 2 of the **program** section.

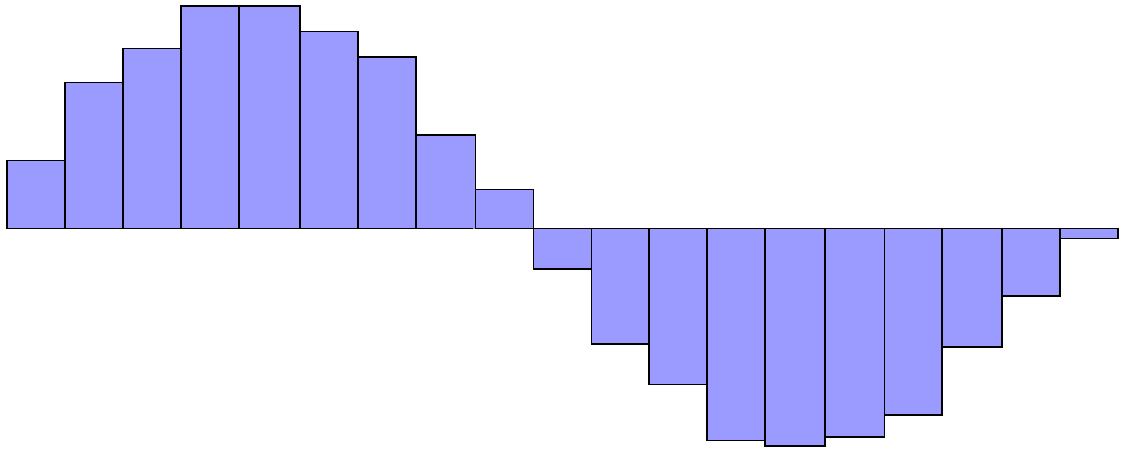


**Figure 10.4**

**PRE-LAB**

**1)** Write a C program to satisfy steps 1 through 3 of the program section.

**2)** Become familiar with the Fast Flag Clearing technique in the Analog to Digital converter. You MUST use Fast Flag Clearing for this lab due to an error within the chip itself

**PROGRAM**

1. Initialize the SPI1 for unidirectional mode. Ensure that SPI1CR1, SPI1CR2, SPI1BR are set. Remember the base frequency is 4 MHz, thus set SPI1BR to go as fast as possible.
2. Using SPI protocol write the two byte word to the 12-bit DAC.
3. Start sending the 16-bits of data described below:

```
       [xxCCMMMM LLLLLLLL]
       FIRST BYTE LAST BYTE
     <----16-bits of data--->
```

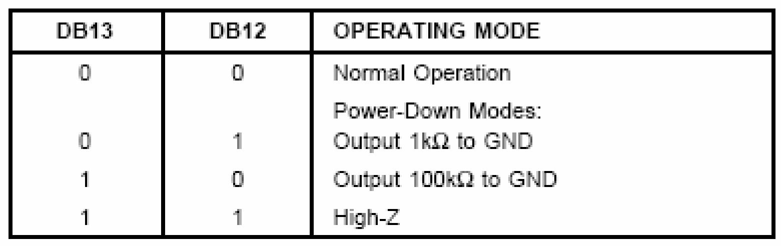
**`xx`** = two **DON'T CARE** bits

**`CC`** = two **CONTROL** bits (Modes of operation)

**`MMMM`** = four **MSB**

**`LLLLLLLL`** = eight **LSB**

4. Set the Control bits to run the DAC in Normal Operation by using the table in Figure 9.2.



| DB13 | DB12 | OPERATING MODE |
|---|---|---|
| 0 | 0 | Normal Operation |
| | | Power-Down Modes: |
| 0 | 1 | Output 1kΩ to GND |
| 1 | 0 | Output 100kΩ to GND |
| 1 | 1 | High-Z |

TABLE I. Modes of Operation for the DAC7512.

**5.** Make the program continuously output a square wave of period ___ milliseconds.

**FLOWCHART**

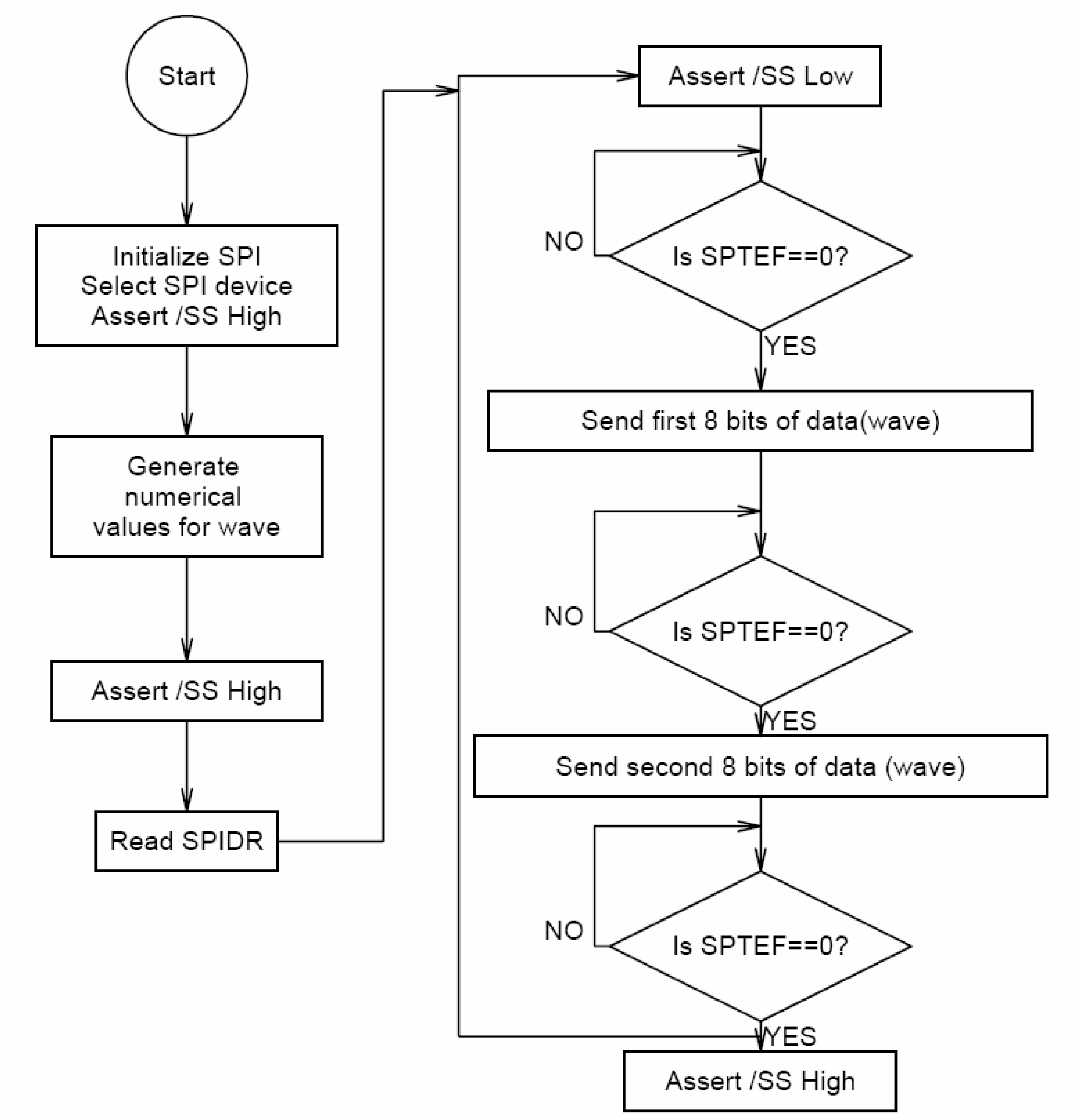


**Figure 9.2**

**DURING LAB**

**1)** Make sure you have the connections required for DAC serial communication set correctly (refer to the datasheet).

**2)** Compile and load the program.

**3)** Run the program. Verify the correct output by showing the signal on the oscilloscope. Use the oscilloscope to determine the period of the wave and sample width.

# Lab 11.a
# Pulse Width Modulation (PWM)

**OBJECTIVE**
In this lab, the student will interface a DC motor with a Pulse Width Modulator. The PWM from the Motorola chip will be fed through a TD62003 driver chip so enough current can be supplied to the motor to power it. The student will connect the DC motor to this driver chip and write a program to control the speed of the DC motor.

**EQUIPMENT**
Freescale PBMCUSLK Student Learning Kit
Current Driver Chip (TD62003)
Small DC Motor

**REGISTERS OF INTEREST**
PWME, PWMPOL, PWMCLK, PWMPRCLK, PWMCAE, PWMCTL, PWMSCLA, PWMPERx, PWMDTYx.

**PRE-LAB**

**1)** Write a C program that will satisfy steps 1 to 3 of the **program section**.

**PROGRAM**

**1.** Interface the **PWM** to use one of the follow channels
___ Channel 0 ___ Channel 1 ___ Channel 2 ___ Channel 3
___ Channel 4 ___ Channel 5 ___ Channel 6 ___ Channel 7

**2.** Make the program start the motor at the following duty:
[ ] 0% duty [ ] 50% duty [ ] 100% duty [ ] ___% duty

**3.** Your TA will choose one of the following inputs to vary the speed of the motor:
___ Interface the keypad so that when you press a key on the keypad, the PWM is set to that speed (such as 0%, 10%, 20%, etc…)
___ Interface the SW2 button so that when you press it the speed increases (don't forget to loop the speed from max speed back to 0)
___ Interface the SW2 button so that when you press it the speed decreases (don't forget to loop the speed from 0 back to max speed)
___ Interface with the serial program so that when you press a keyboard key while in Monitor, the motor spins at a specified speed:
___ = 0% ___ = 15% ___ = 30% ___ = 45%
___= 60% ___ = 75% ___ = 90% ___ = 100%

**DURING LAB**

**1)** Wire the motor and current driver and connect is to the specified PWM channel.

**2)** Connect the oscilloscope to the output of the PWM channel used.

**3)** Compile and load the program.

**4)** Run the program and demonstrate that the specified input device varies the speed of the motor.

**5)** Observe the waveform on the oscilloscope. Is this what you expected to see?

## APPENDIX A
## Freescale Board Connector J1

| | | | |
|---|---|---|---|
| $V_{aux}$ | 1 | 2 | PE1/IRQ* |
| GND | 3 | 4 | RESET* |
| PS1/TXD0 | 5 | 6 | MODC/BKGD |
| PS0/RXD0 | 7 | 8 | PP7/KWP7/PWM7/SCK2 |
| PP0/KWP0/PWM0/MISO1 | 9 | 10 | PAD07/AN07 |
| PP1/KWP1/PWM1/MOSI1 | 11 | 12 | PAD06/AN06 |
| PT0/IOC0 | 13 | 14 | PAD05/AN05 |
| PT1/IOC1 | 15 | 16 | PAD04/AN04 |
| PM4/RXCAN2/RXCAN0/RXCAN4/MOSI0 | 17 | 18 | PAD00/AN00 |
| PM2/RXCAN1/RXCAN0/MISO0 | 19 | 20 | PAD01/AN01 |
| PM5/TXCN2/TXCAN0/TXCAN4/SCK0 | 21 | 22 | PAD02/AN02 |
| PM3/TXCAN1/TXCAN0/SS0 | 23 | 24 | PAD03/AN03 |
| PA7/**ADDR15**/**DATA15** | 25 | 26 | PJ7/KWJ7/TXCAN4/SCL0 |
| PA6/**ADDR14**/**DATA14** | 27 | 28 | PJ6/KWJ6/RXCAN4/SDA0 |
| PA5/**ADDR13**/**DATA13** | 29 | 30 | PP2/KPP2/PWM2/SCK1 |
| PA4/**ADDR12**/**DATA12** | 31 | 32 | PP3/KWP3/PWM3/SS1* |
| PA3/**ADDR11**/**DATA11** | 33 | 34 | PP4/KWP4/PWM4/MISO2 |
| PA2/**ADDR10**/**DATA10** | 35 | 36 | PP5/KWP5/PWM5/MOSI2 |
| PA1/**ADDR9**/**DATA9** | 37 | 38 | PS2/RXD1 |
| PA0/**ADDR8**/**DATA8** | 39 | 40 | PS3/TXD1 |
| PB7/**ADDR7**/**DATA7** | 41 | 42 | PE0/XIRQ* |
| PB6/**ADDR6**/**DATA6** | 43 | 44 | PE2/**RW** |
| PB5/**ADDR5**/**DATA5** | 45 | 46 | PE3/**LSTRB*** |
| PB4/**ADDR4**/**DATA4** | 47 | 48 | PE4/**ECLK** |
| PB3/**ADDR3**/**DATA3** | 49 | 50 | PT2/IOC2 |
| PB2/**ADDR2**/**DATA2** | 51 | 52 | PT3/IOC3 |
| PB1/**ADDR1**/**DATA1** | 53 | 54 | PT4/IOC4 |
| PB0/**ADDR0**/**DATA0** | 55 | 56 | PT5/IOC5 |
| PM1/TXCAN0/**TXB** | 57 | 58 | PT6/IOC6 |
| PM0/RXCAN0/**RXB** | 59 | 60 | PT7/IOC7 |

# APPENDIX B
Interrupt Vector Assignments

| Priority | Vector Address | Interrupt Source | Local Enable | HPRIO Value To Elevate |
|---|---|---|---|---|
| - | $FF80:$FF89 | Reserved | - | - |
| 58 | $FF8A:FF8B | VREG LVI | LVIE | $8A |
| 57 | $FF8C:$FF8D | PWM Emergency Shutdown | PWMIE | $8C |
| 56 | $FF8E:$FF8F | Port P | PIEP | $8E |
| 33 | $FF90:$FFAF | Reserved | - | - |
| 39 | $FFB0:$FFB1 | CAN Transmit | TXEIE [2:0] | $B0 |
| 38 | $FFB2:$FFB3 | CAN Receive | RXFIE | $B2 |
| 37 | $FFB4:$FFB5 | CAN Errors | CSCIE,OVRIE | $B4 |
| 36 | $FFB6:$FFB7 | CAN Wake-up | WUPIE | $B6 |
| 35 | $FFB8:$FFB9 | Flash | CCIE, CBEIE | $B8 |
| - | $FFBA:$FFC3 | Reserved | - | - |
| 29 | $FFC4:$FFC5 | CRG Self Clock Mode | SCMIE | $C4 |
| 28 | $FFC6:$FFC7 | CRG PLL Lock | LOCKIE | $C6 |
| - | $FFC8:$FFCD | Reserved | - | - |
| 24 | $FFCE:$FFCF | Port J | PIEJ | $CE |
| - | $FFD0:$FFD1 | Reserved | - | - |
| 22 | $FFD2:$FFD3 | A/D Converter | ASCIE | $D2 |
| - | $FFD4:$FFD5 | Reserved | - | - |
| 20 | $FFD6:$FFD7 | SCI Serial System | TIE,TCIE,RIE,ILIE | $D6 |
| 19 | $FFD8:$FFD9 | SPI Serial Peripheral System | SPIE, SPTIE | $D8 |
| 18 | $FFDA:$FFDB | Pulse Acc. Input Edge | PAI | $DA |
| 17 | $FFDC:$FFDD | Pulse Acc. Overflow | PAOVI | $DC |
| 16 | $FFDE:$FFDF | Timer Overflow | TOI | $DE |
| 15 | $FFE0:$FFE1 | Timer Channel 7 | C7I | $E0 |
| 14 | $FFE2:$FFE3 | Timer Channel 6 | C6I | $E2 |
| 13 | $FFE4:$FFE5 | Timer Channel 5 | C5I | $E4 |
| 12 | $FFE6:$FFE7 | Timer Channel 4 | C4I | $E6 |
| 11 | $FFE8:$FFE9 | Timer Channel 3 | C3I | $E8 |
| 10 | $FFEA:$FFEB | Timer Channel 2 | C2I | $EA |
| 9 | $FFEC:$FFED | Timer Channel 1 | C1I | $EC |
| 8 | $FFEE:$FFEF | Timer Channel 0 | C0I | $EE |
| 7 | $FFF0:$FFF1 | Real-Time Interrupt | RTIE | $F0 |
| 6 | $FFF2:$FFF3 | IRQ_L Pin | IRQEN | $F2 |
| 5 | $FFF4:$FFF5 | XIRQ_L Pin | CCR | - |
| 4 | $FFF6:$FFF7 | SWI | - | - |
| 3 | $FFF8:$FFF9 | Unimplemented Instruction Trap | None | - |
| 2 | $FFFA:$FFFB | COP Failure Reset | CR2:CR1:CR0 | - |
| 1 | $FFFC:$FFFD | Clock Monitor Fail Reset | CME,SCME | - |
| 0 | $FFFE:$FFFF | External Reset | None | - |