/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ECE 362 - Experiment 8 - Fall 2014

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Completed by: < Dominic Celiano >

< 0617-C >

< Lab Division 9 >

Academic Honesty Statement: In entering my name above, I hereby certify

that I am the individual who created this HC(S)12 source file and that I

have not copied the work of any other student (past or present) while

completing it. I understand that if I fail to honor this agreement, I will

receive a grade of ZERO and be subject to possible disciplinary action.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The objective of this experiment is to implement a reaction time assessment

tool that measures, with millisecond accuracy, response to a visual

stimulus -- here, both a YELLOW LED and the message "Go Team!" displayed on

the LCD screen. The TIM module will be used to generate periodic

interrupts every 1.000 ms, to serve as the time base for the reaction measurement.

The RTI module will provide a periodic interrupt at a 2.048 ms rate to serve as

a time base for sampling the pushbuttons and incrementing the variable "random"

(used to provide a random delay for starting a reaction time test). The SPI

will be used to shift out data to an 8-bit SIPO shift register. The shift

register will perform the serial to parallel data conversion for the LCD.

The following design kit resources will be used:

- left LED (PT1): indicates test stopped (ready to start reaction time test)

- right LED (PT0): indicates a reaction time test is in progress

- left pushbutton (PAD7): starts reaction time test

- right pushbutton (PAD6): stops reaction time test (turns off right LED

and turns left LED back on, and displays test results)

- LCD: displays status and result messages

- Shift Register: performs SPI -> parallel conversion for LCD interface

When the right pushbutton is pressed, the reaction time is displayed

(refreshed in place) on the first line of the LCD as "RT = NNN ms"

followed by an appropriate message on the second line

e.g., 'Ready to start!' upon reset, 'Way to go HAH!!' if a really

fast reaction time is recorded, etc.). The GREEN LED should be turned on

for a reaction time less than 250 milliseconds and the RED LED should be

turned on for a reaction time greater than 1 second.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include <hidef.h> /\* common defines and macros \*/

#include "derivative.h" /\* derivative-specific definitions \*/

#include <mc9s12c32.h>

/\* All funtions after main should be initialized here \*/

char inchar(void);

void outchar(char x);

void tdisp();

void shiftout(char x);

void lcdwait(void);

void send\_byte(char x);

void send\_i(char x);

void chgline(char x);

void print\_c(char x);

void pmsglcd(char[]);

/\* Variable declarations \*/

char go\_team = 0; // "go team" flag (used to start reaction timer)

char leftpb = 0; // left pushbutton flag

char rightpb = 0; // right pushbutton flag

char prevpb = 0; // previous pushbutton state

char runstp = 0; // run/stop flag

int random = 0; // random variable (2 bytes)

int react = 0; // reaction time (3 packed BCD digits)

char digit = 0; //digit to output to the LCD

int i, j, k = 0; //counters

/\* ASCII character definitions \*/

#define CR 0x0D // ASCII return character

/\* LCD COMMUNICATION BIT MASKS \*/

#define RS 0x04 // RS pin mask (PTT[2])

#define RW 0x08 // R/W pin mask (PTT[3])

#define LCDCLK 0x10 // LCD EN/CLK pin mask (PTT[4])

/\* LCD INSTRUCTION CHARACTERS \*/

#define LCDON 0x0F // LCD initialization command

#define LCDCLR 0x01 // LCD clear display command

#define TWOLINE 0x38 // LCD 2-line enable command

#define CURMOV 0xFE // LCD cursor move instruction

#define LINE1 0x80 // LCD line 1 cursor position

#define LINE2 0xC0 // LCD line 2 cursor position

/\* LED BIT MASKS \*/

#define GREEN 0x20

#define RED 0x40

#define YELLOW 0x80

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Initializations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void initializations(void) {

/\* Set the PLL speed (bus clock = 24 MHz) \*/

CLKSEL = CLKSEL & 0x80; // disengage PLL from system

PLLCTL = PLLCTL | 0x40; // turn on PLL

SYNR = 0x02; // set PLL multiplier

REFDV = 0; // set PLL divider

while (!(CRGFLG & 0x08)){ }

CLKSEL = CLKSEL | 0x80; // engage PLL

/\* Disable watchdog timer (COPCTL register) \*/

COPCTL = 0x40; //COP off, RTI and COP stopped in BDM-mode

/\* Initialize asynchronous serial port (SCI) for 9600 baud, no interrupts \*/

SCIBDH = 0x00; //set baud rate to 9600

SCIBDL = 0x9C; //24,000,000 / 16 / 156 = 9600 (approx)

SCICR1 = 0x00; //$9C = 156

SCICR2 = 0x0C; //initialize SCI for program-driven operation

DDRB = 0x10; //set PB4 for output mode

PORTB = 0x10; //assert DTR pin on COM port

/\* Add additional port pin initializations here \*/

PTT\_PTT0 = 0; //turn right LED off to begin

PTT\_PTT1 = 0; //turn left LED off to begin

/\* Initialize SPI for baud rate of 6 Mbs \*/

DDRM = 0xFF;

//DDRT = 0xFF;

SPIBR = 0x01;

SPICR1 = 0x50;

SPICR2 = 0x00; //should be set to output only mode

/\* Initialize digital I/O port pins \*/

ATDDIEN = 0xC0; //establish PAD7 and PAD6 (the pushbuttons) pins as digital inputs

DDRT = 0xFF; //establish all the PTT pins (PT0-PT7) as digital outputs

/\* Initialize the LCD \*/

PTT\_PTT4 = 1; //- pull LCDCLK high (idle)

PTT\_PTT3 = 0; //- pull R/W' low (write state)

send\_i(LCDON); //- turn on LCD (LCDON instruction)

send\_i(TWOLINE); //- enable two-line mode (TWOLINE instruction)

send\_i(LCDCLR); //- clear LCD (LCDCLR instruction)

lcdwait(); //- wait for 2ms so that the LCD can wake up

/\* Initialize RTI for 2.048 ms interrupt rate \*/

CRGINT = CRGINT | 0x80;

RTICTL = 0x1F; //RTI will occur every 2.048 ms (Modulus of 16, Pre-Scalar of 1 = x0011111)

/\* Initialize TIM Ch 7 (TC7) for periodic interrupts every 1.000 ms \*/

TSCR1 = 0x80; //- enable timer subsystem

TIOS = 0x80; //- set channel 7 for output compare

TSCR2 = 0x0C; //- set appropriate pre-scale factor and enable counter reset after OC7

TC7 = 1500; //- set up channel 7 to generate 1 ms interrupt rate

TIE\_C7I = 0; //- initially disable TIM Ch 7 interrupts

/\* Initialize ATD Registers (for extra credit) \*/

ATDCTL2 = 0x80;

ATDCTL3 = 0x10;

ATDCTL4 = 0x85;

ATDCTL5 = 0x10;

ATDCTL2\_AFFC = 1; //fast flag clear mode

/\* Other port initializations necessary for extra credit (but not concerning ATD): \*/

DDRAD = 0; //program port AD (i.e. analog pins) for input mode ??

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void main(void) {

DisableInterrupts;

initializations();

EnableInterrupts;

send\_i(LCDCLR);

pmsglcd("TEST");

for(;;) {

/\* write your code here \*/

if(leftpb){ //If the left pushbutton ("start reaction test") flag is set, then:

leftpb = 0; //- clear left pushbutton flag

runstp = 1; //- set the "run/stop" flag

send\_i(LCDCLR);

chgline(LINE1);

pmsglcd("Ready, Set..."); //- display message "Ready, Set..." on the first line of the LCD

PTT\_PTT1 = 0; //- turn off the left LED (PT1)

PTT\_PTT0 = 1; //- turn on the right LED (PT0)

}

if(runstp){ //If the "run/stop" flag is set, then:

if(!go\_team){ //- If the "go\_team" flag is NOT set (i.e. the challenge flag hasn't been thrown yet), then:

if(random == 0x0000){ // If "random" = $0000 - this makes the challenge flag be thrown at a random time

go\_team = 1; //- set the "go\_team" flag (indicating that the challenge flag has been thrown)

TCNT = 0; //- clear TCNT register (of TIM) - i.e. reset the timer to make sure it doesn't overflow

react = 0; //clear "react" variable (2 bytes)

TIE\_C7I = 1; //- enable TIM Ch7 interrupts

PTT\_PTT7 = 1; //- turn on YELLOW LED

chgline(LINE2);

pmsglcd("Go Team!"); //- display message "Go Team!" on the second line of the LCD

}

}

}

if(rightpb){ //If the right pushbutton ("stop reaction test") flag is set, then

rightpb = 0; //- clear right pushbutton flag

runstp = 0; //- clear the "run/stop" flag

go\_team = 0; //- clear the "go\_team" flag, consequently resetting the count

PTT\_PTT7 = 0; //- turn off yellow LED

TIE\_C7I = 0; //- disable TIM Ch 7 interrupts

tdisp(); //- call "tdisp" to display reaction time message

PTT\_PTT0 = 0; //- turn off right LED (PT0)

PTT\_PTT1 = 1; //- turn on left LED (PT1)

}

if(react == 999){ //If "react" >= 999 (the maximum 3-digit BCD value), we reached the max time, so:

runstp = 0; //- clear the "run/stop" flag

PTT\_PTT7 = 0; //- turn off yellow LED

PTT\_PTT6 = 1; //turn on red LED

TIE\_C7I = 0; //- disable TIM Ch 7 interrupts

send\_i(LCDCLR);

chgline(LINE1);

pmsglcd("Time = 999 ms"); //- display message "Time = 999 ms" on the first line of the LCD

chgline(LINE2);

pmsglcd("Too slow!"); //- display message "Too slow!" on the second line of the LCD

PTT\_PTT0 = 0; //- turn off right LED (PT0)

PTT\_PTT1 = 1; //- turn on left LED (PT1)

react++; //increment react to get us out of this

}

} /\* loop forever \*/

} /\* do not leave main (unless going to an isr...) \*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RTI interrupt service routine: RTI\_ISR

Initialized for 2.048 ms interrupt rate

Samples state of pushbuttons (PAD7 = left, PAD6 = right)

If change in state from "high" to "low" detected, set pushbutton flag

leftpb (for PAD7 H -> L), rightpb (for PAD6 H -> L)

Recall that pushbuttons are momentary contact closures to ground

Also, increments 2-byte variable "random" each time interrupt occurs

NOTE: Will need to truncate "random" to 12-bits to get a reasonable delay

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

interrupt 7 void RTI\_ISR(void)

{

CRGFLG = CRGFLG | 0x80; // clear RTI interrupt flag

// increment 'random', but only if the program is running (i.e. we are counting up the accumulated time)

if (runstp){

random++; //this is incremented to change the random time that will be generated

random = random & 0x0FFF;

}

if (prevpb && !PORTAD0\_PTAD7){ // if leftpb was pressed

leftpb = 1;

}

prevpb = PORTAD0\_PTAD7;

if (prevpb && !PORTAD0\_PTAD6){ // if rightpb was pressed

rightpb = 1;

}

prevpb = PORTAD0\_PTAD6;

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TIM Channel 7 interrupt service routine

Initialized for 1.00 ms interrupt rate

Increment (3-digit) BCD variable "react" by one

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

interrupt 15 void TIM\_ISR(void)

{

/\* We enter this ISR every 1.00 ms \*/

TFLG1 = TFLG1 | 0x80; // clear TIM CH 7 interrupt flag

if (runstp){ //only increment the time if the program is running/the time should be accumulating

react++; //increment the time the reaction is taking by 1ms

}

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

tdisp: Display "RT = NNN ms" on the first line of the LCD and display

an appropriate message on the second line depending on the

speed of the reaction.

Also, this routine should set the green LED if the reaction

time was less than 250 ms.

NOTE: The messages should be less than 16 characters since

the LCD is a 2x16 character LCD.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void tdisp()

{

send\_i(LCDCLR);

//output "RT = NNN ms" on the first line

chgline(LINE1);

pmsglcd("RT = ");

digit = (react / 100) + 48;

print\_c(digit); //print hundred's digit

digit = (react / 10);

digit = (digit % 10) + 48;

print\_c(digit); //print ten's digit

digit = (react % 10) + 48;

print\_c(digit); //print one's digit

pmsglcd(" ms");

chgline(LINE2); //change to the second line

if (react >= 250 && react < 650){

pmsglcd("Not half bad!"); //output "Not bad!"

}

if (react >= 650){

pmsglcd("Try harder!"); //output "Try harder!"

}

if (react < 250){

PTT\_PTT5 = 1; //turn on the green LED if our reaction time was <250 ms

pmsglcd("Wow! Amazing!"); //output "Wow! Amazing!"

}

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

shiftout: Transmits the character x to external shift

register using the SPI. It should shift MSB first.

MISO = PM[4]

SCK = PM[5]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void shiftout(char x)

{

// read the SPTEF (transmit empty) bit, continue if bit is 1 (i.e. transmit data register is empty - if not, wait)

while(SPISR\_SPTEF != 1){

}

SPIDR = x; // write data to SPI data register

for(i = 0; i<30; i++){

asm{

nop;

}

}// wait for 30 cycles for SPI data to shift out

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

lcdwait: Delay for approx 2 ms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void lcdwait()

{

for(j = 0; j<1000; j++){ //since clock = 24MHz, 48000 cycles will take 2ms

}

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

send\_byte: writes character x to the LCD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void send\_byte(char x)

{

shiftout(x); // shift out character to the SPI module, which will then send out out to the GAL and then to the LCD

// pulse LCD clock line low->high->low? Why?

PTT\_PTT4 = 0;

lcdwait();

PTT\_PTT4 = 1;

PTT\_PTT4 = 0;

lcdwait(); // wait 2 ms for LCD to process data

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

send\_i: Sends instruction byte x to LCD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void send\_i(char x)

{

PTT\_PTT2 = 0; // set the register select line low (instruction data)

send\_byte(x); // send byte

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

chgline: Move LCD cursor to position x

NOTE: Cursor positions are encoded in the LINE1/LINE2 variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void chgline(char x)

{

send\_i(CURMOV); //tell the LCD we're about to move the cursor

send\_i(x); //tell the LCD where we're going to move the cursor to

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

print\_c: Print (single) character x on LCD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void print\_c(char x)

{

PTT\_PTT2 = 1; //set the register select to high, since we're sending a character

send\_byte(x);

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

pmsglcd: print character string str[] on LCD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void pmsglcd(char str[])

{

k = 0;

while(str[k] != '\0'){

print\_c(str[k]);

k++;

}

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Character I/O Library Routines for 9S12C32 (for debugging only)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name: inchar

Description: inputs ASCII character from SCI serial port and returns it

Example: char ch1 = inchar();

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

char inchar(void) {

/\* receives character from the terminal channel \*/

while (!(SCISR1 & 0x20)); /\* wait for input \*/

return SCIDRL;

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name: outchar

Description: outputs ASCII character x to SCI serial port

Example: outchar('x');

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void outchar(char x) {

/\* sends a character to the terminal channel \*/

while (!(SCISR1 & 0x80)); /\* wait for output buffer empty \*/

SCIDRL = x;

}