CMPE423 Embedded Systems Design

Seventh Application

Objective: Design Application with LCD and FSM.

Introduction:

An LCD module is a display device that may display ASCII characters on two-lines by 16 character. It is interfaced to microcontroller ports by 8-bit or 4-bit data and three control lines. You can find further information on LCD modules in CMPE423 Course Notes.

Interfacing of an LCD module

The interfacing of the LCD Module has two alternatives. An 8-bit interfacing is possible to transfer each character-code in one clock cycle. This transfer mode has simple and straightforward signal timing specification, and used in most applications if there is plenty of I/O pins (requires minimum 13 pins) available for this purpose. When there is a shortage of I/O pins, the display is interfaced in nibble-mode, where the transfer of each character-code requires two clock cycles.

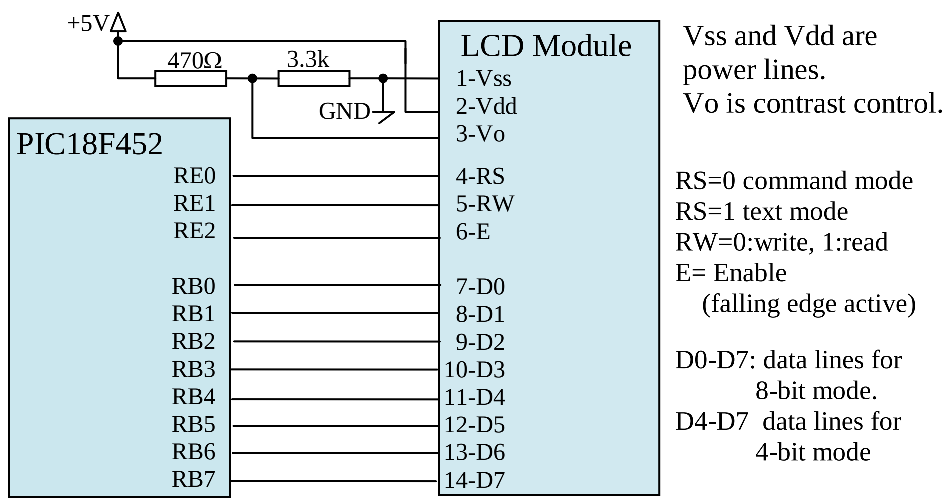
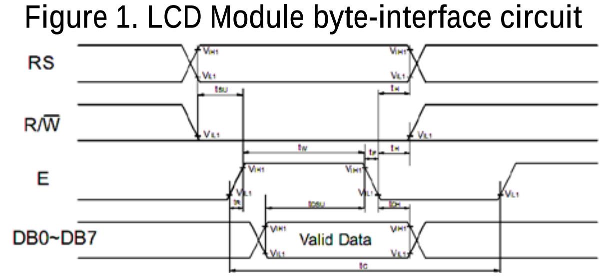


Figure 1. LCD Module byte-interface circuit



t_C>40ns (minimum time between successive writings) t_{DSU}>80ns (valid data prior to falling edge of E)
t_{SU}>40ns (setup time from R/W or RS to E) t_W>230ns (write cycle enable time)
t_H>10ns (write hold time after falling edge of E)

Figure 2 LCD Module write cycle interface timing specifications

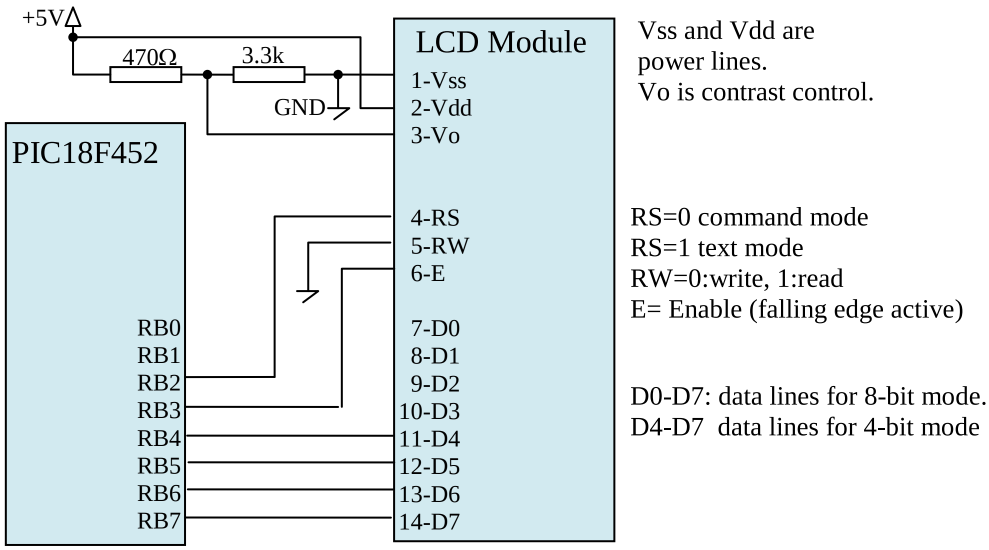


Figure 3 LCD Module Nibble-Mode Interfacing Circuit

Initialization

The initialization of a LCD Module requires 80ms reset period followed by a sequence of nibbles to be sent to the interface port. The control codes are sent after RS has been reset, and printable codes are sent after RS has been set to one.

Nibble-Interface mode initialization procedure

-After the power-up or reset, wait 0.1 second for proper reset of LCD module.

-Make RS low to send following data as control codes.

-Send the following nibbles with valid data at the negative edge of E.

Nibbles	Function
`0x0,0x2,`	=> sets nibble-interface mode
`0x2,0x8,`	=> sets two-line 4-bit interface mode
`0x0,0x1,`	=> clears display
`0x0,0xC,`	=> turns off cursor, turns on display
`0x0,0x6.`	=> Increment cursor automatically

Byte-Interface mode initialization procedure

-After the power-up or reset, wait 0.1 second before the LCD initialization.

-Make RS low to send initialization control codes.

-Send the following bytes (at the negative edge of E).

Bytes	Function
0x03,	=> initialization reset
0x38,	=> sets two-line 8-bit interface mode
0x01,	=> clears display. It requires 4ms time.
0x0C,	=> turns off cursor, turns on display
0x06.	=> Increment cursor automatically

Control Characters for LCD:

0xFF: PrintLCD switches between text-mode and command mode (It starts in text mode)

0x00 ends the string C	**0x01** clears the display
0x02 initializes and moves cursor at home	**0x03** initializes, and moves cursor at home
0x04 moves cursor to left after a char.	**0x05** shifts line to left after a char
0x06 moves cursor to right after a char.	**0x07** shifts line to right after a char.
0x08 turns display off	**0x0C** turns display on
0x0E starts displaying cursor	**0x0F** starts cursor to blink on
0x10 shifts cursor one position to left	**0x14** shifts cursor one position to right
0x18 shifts display one position to left	**0x1C** shifts display one position to right
0x20 start 4-bit data, single-line display	**0x28** starts 4-bit data, 2-line display
0x30 start 8-bit data, single-line display	**0x38** starts 8-bit data, 2-line display

Once the module is initialized, the positioning of the cursor is achieved using the following cursor positioning codes:

0x80	0x81	0x82	0x83	0x84	0x85	0x86	0x87
0xC0	0xC1	0xC2	0xC3	0xC4	0xC5	0xC6	0xC7

i.e., for the first line, third position the code is **0x82**, and the fifth character position on the second line is addressed by **0xC4**.

The printable character set of the module is:

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
2-		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	***	]	^	-
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{	|	}	→	←

Coding for LCD Module 4-bit interfacing.

Using an LCD module in an application is quite simple if you have the necessary codes already tested in your code library. In our case, we have the following declarations of variables and functions for the LCD module.

The global bits `LCD4..LCD7`, `LCDE`, `LCDR` declare io pins for data, RS, and E lines. `LCD4T`..`LCD7T`, `LCDRT`, and `LCDET` declare tris bits of data, RS and E lines. `LCDNibble()` sends a single nibble conforming to write timing specifications.

`PrintLCD()` writes the pointed string to LCD display module starting in text mode. Character `0xFF` toggles between text and control modes. Any character less than `0x04` and greater than `0x7F` is sent to module as control characters, which provides to deliver clear (**0x01**), data-width selection (**0x02** and **0x03**) and cursor position codes (**0x80**...**0xDF**) in control mode without needing to toggle mode to control and text. It provides 4ms wait period after the characters which are less than or equal to **0x04.**

***c=strcpy(*a,*b)** provides a copy of string **b** starting from **a**, and **c** returns the pointer of last empty location.

***c=i2a(k,*a)** provides filling an integer value **k** starting from **a**, and **c** returns the pointer of last empty location.

LCDw2u(i) provides **2**x**i** microsecond wait for the timing of the data transfer to the module. i=0 corresponds to highest possible delay time, which is about 512 microseconds. It is convenient to trimm this function to exactly 2 μs and write all delays in terms of this function.

```
//Global variables for LCD
// Data and tris bits, each bit may be defined at any io pin.
bit LCD4 @PORTB.4, LCD5 @PORTB.5, LCD6 @PORTB.6, LCD7 @PORTB.7,
    LCD4T @TRISB.4, LCD5T @TRISB.5, LCD6T @TRISB.6, LCD7T @TRISB.7;
bit     LCDR  @PORTB.2, LCDRT @TRISB.2, // Control RS and its tris
        LCDE  @PORTB.3, LCDET @TRISB.3, // Control E and its tris
        LCDS, LCDC ; // single control char, and control toggle flags

// Procedures related to LCD
char strcpy( char *a, const char *b){
    char t; do{ t=*b; *a=t; ++a; ++b;}while(t); return a; }

char i2a(int k:16, char *a){
  char *b=a;
  if(k<0){ k=-k; *a='-'; ++a;}
  if(k>9999) { *a='0'; while(k>9999){++*a; k-=10000;} ++a;}
  if(k>999)  { *a='0'; while(k>999) {++*a; k-=1000 ;} ++a;}
  if(k>99)   { *a='0'; while(k>99)  {++*a; k-=100  ;} ++a;}
  if(k>9)    { *a='0'; while(k>9)   {++*a; k-=10   ;} ++a;}
  *a='0'+k; ++a; return a; }

void LCDw2u(char W){ // use single nop() for 4MHz Xtal
   WREG =W; do{ nop(); }while( --WREG ); }

void LCDNibble(char Ch){
  LCD4T=0; LCD5T=0; LCD6T=0; LCD7T=0;
  LCDRT=0;  LCDET=0; LCDE = 0;
  if(LCDC) LCDR=0; else LCDR = 1;
  if(Ch.4) LCD4=1; else LCD4=0; if(Ch.5) LCD5=1; else LCD5=0;
  if(Ch.6) LCD6=1; else LCD6=0; if(Ch.7) LCD7=1; else LCD7=0;
  LCDE=1; LCDw2u(2); LCDE=0;
  LCD4=0;nop();LCD5=0;nop();LCD6=0;nop();LCD7=0; LCDw2u(5); }

void PrintLCD(const char *Ch){
  char WC,WP=0; LCDS=0 ; LCDC=0 ; LCDE=0 ;
  do{ WC=Ch[WP]; WP++;
    if(WC){ LCDC=0;
      if(WC==0xFF) LCDS ^=1 ;
      else { LCDC=WC.7||LCDS; if(WC<4) LCDC=1;
        LCDNibble( WC & 0xF0);
        if(LCDS && WC==0x28){//set mode takes 3ms time
          char T=12;do{LCDw2u(0);}while(--T);}
        LCDNibble( swap(WC) & 0xF0 );
        if(WC<4){char T=12; do{ LCDw2u(0);}while(--T);}
        LCDw2u(20); } }
    }while(WC); }
```

Design specifications

The who-is-fast game prototype shall have two buttons: **A** and **B**, two LED indicators: Blink-Alive-LED (**BA-LED**) and Game-LED (**G-LED**), and an **LCD** display module to write text messages.

BA-LED blinks for 200ms at every 1s periodically to indicate the microcontroller of the game is functional.

At the **standby state**, if two players push buttons **A** and **B**, and both of them released, then the processor writes on LCD “ready” and it waits for a random 1 to 5 seconds period.

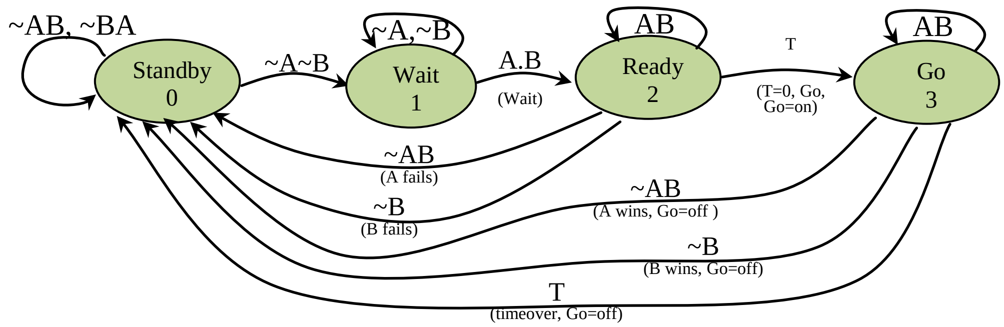
In the **ready state,** if button A is pushed, then **c**ontroller writes "**A failed**.". Similarly if **B** is pushed, controller writes "**B failed**.", and jumps to **standby**.

After the random period, controller turns **G-LED** on, and it jumps to **go-state**.

In the **go-state**, if player **A** pushes earlier than player **B**, controller writes "**A wins**." and jumps to **standby** after turning **G-LED off**. Similarly, if **B** is earlier than **A**, then it writes "**B wins**" and, it jumps to **standby** after turning **G-LED off**. If none pushes the buttons in one second, then controller writes "**timeover**" and jumps to **standby** mode.

Construction of the FSM chart for the design:

For **A** and **B,** let **1** indicates button is **released**, and **0** means button is **pushed**. FSM chart shall indicate each state by a circle (node), and the condition of transitions from a state to another state by an arrow. The design specification explains the game by three states: **standby**, **ready**, **go**. However, the standby state changes to ready after and intermediate state if only both buttons are pushed, and wait there until both switches are released. Let us call this intermediate state **wait**. We enumerate these states by **standby**=0, **wait**=1, **ready**=2, and **go**=3. The transition from a state to another state depends on inputs such as buttons **A** and **B**, and timeover **T**, which we plan to increment at every millisecond by one. At the beginning of the **ready** state we can set **T**=0, and **Tdc1ms** to random milliseconds **TR**, decrement **Tdc1ms** at every mainloop, and set **T**=1 when **Tdc1ms** becomes zero. Similarly, when entering to **go** state, we shall set T=0, and **Tdc1ms** to 1000, and let it be decremented by one at every mainloop. When Tdc1ms becomes zero, we may set T=1, and let the finite state machine pass to the standby state.



state (S)	inputs (TAB)	next state (NS)	output Go-LED/LCD/Timer	Explanation
0	X00	1		standby, A,B both pushed
0	X01	0		standby, only A pushed
0	X10	0		standby, only B pushed
0	X11	0		standby, A,B released
1	X00	1		wait, both A,B pushed
1	X01	1		wait, only A pushed
1	X10	1		wait, only B pushed
1	X11	2	/ ready / 1..5 sec.	wait, A,B released
2	000	0	/ A fails /	ready, A,B both pushed
2	001	0	/ A fails /	ready, only A pushed
2	010	0	/ B fails /	ready, only B pushed
2	011	2		ready, A,B released
2	1XX	3	on / go / 1 sec.	ready, timeover
3	000	0	off / A wins /	go, A,B both pushed
3	001	0	off / A wins /	go, only A pushed
3	010	0	off / B wins /	go, only B pushed
3	011	3		go, A,B released
3	1XX	0	off / timeover /	go, timeover

Figure 4. FSM chart of the Who Is Fast game

For a simpler representation of the state transitions, we have shown timeover by **T**=1. Timeover signal may be obtained easily by using a 16-bit time down-counter, **Tdc1ms**, which down counts for the specified period of time.

The outputs may be implemented by assigning numbers to each action for LED, LCD, and timer individually, i.e., G=0 means no action, G=1 is turn LED on, G=2 means turn LED off; L=0 means do not write anything on LCD, L=1 is “ready”, L=2 is “A fails”, L=3 is “B fails”, L=4 is “go”, L=5 is “A wins”, L=6 is “B wins”, L=7 is “timeover”. Similarly, P=0 is no action for time-counter, P=1 is set it randomly between 1000 and 5000 milliseconds, P=2 is set it randomly to 1000 milliseconds.

A random number between 1000 and 6000 may be easily generated with a 16-bit variable tr, adding 1111 on tr at each mainloop, and subtracting 3000 from tr whenever it exceeds 5000.

Here is the simplified **state transition** and **output tables** of the FSM chart.

state (S)	inputs (TAB)	next state (NS)	G	L	T
0	X00	1	0	0	0
0	X01	0	0	0	0
0	X10	0	0	0	0
0	X11	0	0	0	0
1	X00	1	0	0	0
1	X01	1	0	0	0
1	X10	1	0	0	0
1	X11	2	0	1	1
2	000	0	0	2	0
2	001	0	0	2	0
2	010	0	0	3	0
2	011	2	0	0	0
2	1XX	3	1	4	2
3	000	0	2	5	0
3	001	0	2	5	0
3	010	0	2	6	0
3	011	3	0	0	0
3	1XX	0	2	7	0

Keys: **T**:Timeover, **A**/**B**: Buttons (0-pushed,1-released); B: **S**/**NS**: State/Nextstate(0-standby, 1-wait, 2-ready, 3-go); **G**: Go-LED(0-no action, 1-on, 2-off); **L**: LCD(0-no action, 1-“ready”, 2-“A fails”, 3-“B fails”, 4-“go”, 5-“A wins”, 6-“B wins”, 7-“timeover”).

Figure 5. Simplified FSM chart

Preliminaries for Implementation of Coding

1- **Timebase:** The code needs 1 ms timebase. Assuming that crystal frequency is 4MHz, the timer must be set to 16-bit, prescaler=1 (PSA=1), **Nc**=1000. Interrupt serviced timer can increment **Tint** at every 1ms. Main loop may update and test **Tdc1ms** whenever **Tint** is nonzero.

2- **The random time TR** must be between **1000** and **5000** ms. A random TR may be obtained by `TR+=1111; if(TR>5000) TR-=4000;` by turning a roulette at every main loop.

3- **Action for LCD** messages depends on output **L**
0-no action, 1-“ready”, 2-“A fails”, 3-“B fails”, 4-“go”,
5-“A wins”, 6-“B wins”, 7-“timeover”

4- Action for timeout T depends on output **P**.
0-no action, 1- `T=0;Tdc1ms=TR;` 2- `T=0;Tdc1ms=1000;` ,
and at every mainloop cycle execute `if(Tdc1ms<=0) T=1;`

5- **A and B** represent button status: 0- pushed, 1- released

6- **Game-LED** is controlled by **G. G**=**1** is **on**; **G**=2 is **off**, G=0 is no action.

7- **Index of the state transition table** is

```
IX= 8*S+4*T+2*A+B;
```

8- For this selection of **IX**, the **next state table** is

```
NS={1,0,0,0,  1,0,0,0,  1,1,1,2,  1,1,1,2,
    0,0,0,2,  3,3,3,3,  0,0,0,3,  0,0,0,0 }
```

The **output table for LED** action is

```
G ={0,0,0,0,  0,0,0,0,  0,0,0,1,  0,0,0,1,
    0,0,0,0,  1,1,1,1,  2,2,2,0,  2,2,2,2 }
```

The **output table for LCD** action is

```
L ={0,0,0,0,  0,0,0,0,  0,0,0,1,  0,0,0,1,
    2,2,3,0,  4,4,4,4,  5,5,6,0,  7,7,7,7 }
```

The **output table for timing** action is

```
P ={0,0,0,0,  0,0,0,0,  0,0,0,1,  0,0,0,1,
    0,0,0,0,  2,2,2,2,  0,0,0,0,  0,0,0,0, }
```

For almost all large system development processes, the shortest development time is achieved by divide and conquer concept. In this project, we will implement the timer interrrupts and blink-alive at the first stage of the development. Thereafter, we plan to test LCD module initialization, and as the third stage we plan to complete the implementation of FSM.

Coding of timer interrupt test

```
// who is fast game
// RB7 is  for blinkalive LED
// TIMER0 generates 1ms interrupts.
// Fosc=4MHz, 1ms=1000cc., PS=1, Nc= 1000
#define NC 1000
char Tint,T1ms;

#pragma origin 0x08
void ISR(void){ //Int. Service Routine
  static char WX,SX,BX;WX=W; SX=STATUS;BX=BSR;
  if(TMR0IF){
    T0CON=0b10001000; // set Timer0 for 1ms
    TMR0H=-NC/256; TMR0L=-NC%256; TMR0IF=0;
    ++Tint;  }
  W=WX; STATUS=SX;BSR=BX;
  retint();}  // return from interrupt

void InitPorts(void){
  TRISB.0=0; // RB0 LED out
  TRISB.1=1; // RB1 button A
  TRISB.2=1; // RB2 button B
  TRISB.7=0; // RB7 BA LED
  TRISC =0; } // RC7..RC0  output

void InitInt(void){
  TMR0IE=1; TMR0IF=0; TMR0IP=0;
  IPEN=0; PEIE=1; GIE=1;}

void Blink(void){
  static uns16 BAcount;
  if(BAcount<200) PORTB.7=0; // BA LED on
  else PORTB.7=1; // BA LED off
  BAcount++; if(BAcount>1000) BAcount=0;}

void main(void){
  char PB;
  T1ms=0;
  InitPorts();
  InitInt(); TMR0IF=1;

  do{ // mainloop
```

```
    if(Tint){
       GIE=0;T1ms+=Tint;Tint=0;GIE=1;
       Blink();}
    }while(1);
  }
```

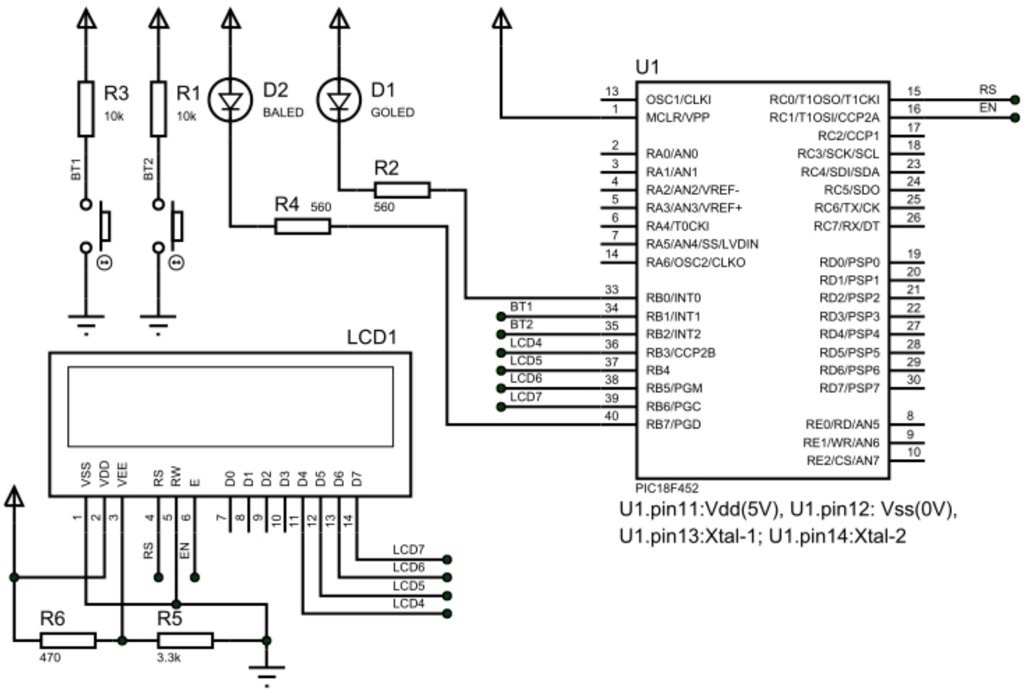


Figure 3. Circuit for LCD connections with PIC18F452.

Next we shall interface the LCD module.

```
// who is fast game, LCD interface
// RB7 is  for blinkalive LED
// TIMER0 generates 1ms interrupts.
// Fosc=4MHz, 1ms=1000cc., PS=1, Nc= 1000
#define NC 1000
char Tint,T1ms ;

//Global variables for LCD
// Data and tris bits, each bit may be defined at any io pin.
bit LCD4 @PORTB.3, LCD5 @PORTB.4, LCD6 @PORTB.5, LCD7 @PORTB.6,
    LCD4T @TRISB.3, LCD5T @TRISB.4, LCD6T @TRISB.5, LCD7T @TRISB.6;
bit     LCDR  @PORTC.0, LCDRT @TRISC.0, // Control RS and its tris
        LCDE  @PORTC.1, LCDET @TRISC.1, // Control E and its tris
        LCDS, LCDC ; // single control char, and control toggle flags


#pragma origin 0x08
void ISR(void){    // Int. Service Routine
  static char WX,SX,BX;WX=W; SX=STATUS;BX=BSR;
  if(TMR0IF){
    T0CON=0b10001000; // set Timer0 for 1ms
    TMR0H=-NC/256; TMR0L=-NC%256; ++Tint;
    TMR0IF=0; }
  W=WX; STATUS=SX;BSR=BX;
  retint();}      // return from interrupt

// Procedures related to LCD
void LCDw2u(char W){ // W x 3 microsecond for 4MHz.
```

```
   WREG =W; do{ }while( --WREG ); }

void LCDNibble(char Ch){
  LCD4T=0; LCD5T=0; LCD6T=0; LCD7T=0;
  LCDRT=0;  LCDET=0; LCDE = 0;
  if(LCDC) LCDR=0; else LCDR = 1;
  if(Ch.4) LCD4=1; else LCD4=0; if(Ch.5) LCD5=1; else LCD5=0;
  if(Ch.6) LCD6=1; else LCD6=0; if(Ch.7) LCD7=1; else LCD7=0;
  LCDE=1; LCDw2u(2); LCDE=0;
  LCD4=0;nop();LCD5=0;nop();LCD6=0;nop();LCD7=0; LCDw2u(5); }

void PrintLCD(const char *Ch){
  char WC,WP=0; LCDS=0 ; LCDC=0 ; LCDE=0 ;
  do{ WC=Ch[WP]; WP++;
    if(WC){ LCDC=0;
      if(WC==0xFF) LCDS ^=1 ;
      else { LCDC=WC.7||LCDS; if(WC<4) LCDC=1;
        LCDNibble( WC & 0xF0);
        if(LCDS && WC==0x28){//set mode takes 3ms time
          char T=12;do{LCDw2u(0);}while(--T);}
        LCDNibble( swap(WC) & 0xF0 );
        if(WC<4){char T=12; do{ LCDw2u(0);}while(--T);}
        LCDw2u(20); } }
    }while(WC); }

void InitPorts(void){
  TRISB.0=0;  // RB0 LED out
  TRISB.1=1;  // RB1 button A
  TRISB.2=1;  // RB2 button B
  TRISB.7=0;  // RB7 BA LED
  TRISC =0; } // RC7..RC0  output

void InitInt(void){
  TMR0IE=1; TMR0IF=0; TMR0IP=0;
  IPEN=0; PEIE=1; GIE=1;}

void Blink(void){
  static uns16 BAcount;
  if(BAcount<200) PORTB.7=0; // BA LED on
  else PORTB.7=1; // BA LED off
  BAcount++; if(BAcount>2000) BAcount=0;}

void main(void){
  char PB;
  T1ms=0;
  InitPorts();
  InitInt(); TMR0IF=1;
  // wait 100ms for LCD reset
  {char i=200; do{LCDw2u(250);}while(--i);}
  //PrintLCD("\xff\x02\x82\x28\x03\x0C\x06");
  PrintLCD("\xff\x02\x82\x28\x01\x03\x0C\x06");
  PrintLCD("\x80Ready. ");
  do{
        if(Tint){
                T1ms+=Tint; GIE=0;Tint=0;GIE=1;
          Blink();}
     }while(1);
  }
```

C code of FSM and its outputs

```
// who is fast game
// RB7 is  for blinkalive LED
// TIMER0 generates 1ms interrupts.
// Fosc=4MHz, 1ms=1000cc., PS=1, Nc= 1000
#define NC 1000
uns16 Tdc1ms, // time down counter
char Tint, T ; // for counting interrupts and timeover
// FSM related tables and variables
```

```
      TR;  // Random time for FSM
char  SS;  // state for FSM
const char
NS[]={1,0,0,0,  1,0,0,0,  1,1,1,2,  1,1,1,2,
      0,0,0,2,  3,3,3,3,  0,0,0,3,  0,0,0,0 },
GT[]={0,0,0,0,  0,0,0,0,  0,0,0,0,  0,0,0,0,
      0,0,0,0,  1,1,1,1,  2,2,2,0,  2,2,2,2 },
LT[]={0,0,0,0,  0,0,0,0,  0,0,0,1,  0,0,0,1,
      2,2,3,0,  4,4,4,4,  5,5,6,0,  7,7,7,7 },
TT[]={0,0,0,0,  0,0,0,0,  0,0,0,1,  0,0,0,1,
      0,0,0,0,  2,2,2,2,  0,0,0,0,  0,0,0,0 };
//Global variables for LCD
// Data and tris bits, each bit may be defined at any io pin.
bit LCD4 @PORTB.3, LCD5 @PORTB.4, LCD6 @PORTB.5, LCD7 @PORTB.6,
    LCD4T @TRISB.3, LCD5T @TRISB.4, LCD6T @TRISB.5, LCD7T @TRISB.6;
bit     LCDR  @PORTC.0, LCDRT @TRISC.0, // Control RS and its tris
        LCDE  @PORTC.1, LCDET @TRISC.1, // Control E and its tris
        LCDS, LCDC ; // single control char, and control toggle flags

#pragma origin 0x08
void ISR(void){ //Int. Service Routine
  static char WX,SX,BX;WX=W; SX=STATUS;BX=BSR;
  if(TMR0IF){
    T0CON=0b10001000; // set Timer0 for 1ms
    TMR0H=-NC/256; TMR0L=-NC%256; TMR0IF=0;
    ++Tint;  }
  W=WX; STATUS=SX;BSR=BX;
  retint();}  // return from interrupt

// Procedures related to LCD
char strcpy( char *a, const char *b){
    char t; do{ t=*b; *a=t; ++a; ++b;}while(t); return a; }

char i2a(int k:16, char *a){
  char *b=a;
  if(k<0){ k=-k; *a='-'; ++a;}
  if(k>9999) { *a='0'; while(k>9999){++*a; k-=10000;} ++a;}
  if(k>999)  { *a='0'; while(k>999) {++*a; k-=1000 ;} ++a;}
  if(k>99)   { *a='0'; while(k>99)  {++*a; k-=100  ;} ++a;}
  if(k>9)    { *a='0'; while(k>9)   {++*a; k-=10   ;} ++a;}
  *a='0'+k; ++a; return a; }

void LCDw2u(char W){ // use single nop() for 4MHz Xtal
   WREG =W; do{ nop(); }while( --WREG ); }

void LCDNibble(char Ch){
  LCD4T=0; LCD5T=0; LCD6T=0; LCD7T=0;
  LCDRT=0;  LCDET=0; LCDE = 0;
  if(LCDC) LCDR=0; else LCDR = 1;
  if(Ch.4) LCD4=1; else LCD4=0; if(Ch.5) LCD5=1; else LCD5=0;
  if(Ch.6) LCD6=1; else LCD6=0; if(Ch.7) LCD7=1; else LCD7=0;
  LCDE=1; LCDw2u(2); LCDE=0;
  LCD4=0;nop();LCD5=0;nop();LCD6=0;nop();LCD7=0; LCDw2u(5); }

void PrintLCD(const char *Ch){
  char WC,WP=0; LCDS=0 ; LCDC=0 ; LCDE=0 ;
  do{ WC=Ch[WP]; WP++;
    if(WC){ LCDC=0;
      if(WC==0xFF) LCDS ^=1 ;
      else { LCDC=WC.7||LCDS; if(WC<4) LCDC=1;
        LCDNibble( WC & 0xF0);
        if(LCDS && WC==0x28){//set mode takes 3ms time
          char T=12;do{LCDw2u(0);}while(--T);}
        LCDNibble( swap(WC) & 0xF0 );
        if(WC<4){char T=8; do{ LCDw2u(0);}while(--T);}
        LCDw2u(20); } }
    }while(WC); }

void FSM(void){
char PB, // portb to stabilize A, B, and outputs
     IX, // next state table index
     A, B,// button inputs
```

```
     To,Lo,Go; // for outputs
clrwdt();
PB=PORTB;
TR+=1111; if(TR>5000) TR-=4000; // Roulette for random number
if(PB.1) A=1; else A=0; if(PB.2) B=1; else B=0;
//index IX=8*SS+4*T+2*A+B
IX=8*SS; IX+=4*T; IX+=2*A; IX+=B;
// next state
SS=NS[IX];
// LCD outputs
Lo=LT[IX];
  if(Lo==1) PrintLCD("\x80 Ready  ");
  if(Lo==2) PrintLCD("\x80A fails ");
  if(Lo==3) PrintLCD("\x80B fails ");
  if(Lo==4) PrintLCD("\x80  Go    ");
  if(Lo==5) PrintLCD("\x80A wins  ");
  if(Lo==6) PrintLCD("\x80B wins  ");
  if(Lo==7) PrintLCD("\x80timeover");
Go=GT[IX]; // Go-LED
   if(Go==1) PB.0=0;   //on
   if(Go==2) PB.0=1;   //off
To=TT[IX]; // timeover counter
   if(To==1) {T=0; GIE=0; Tdc1ms=TR; GIE=1;}   // random
   if(To==2) {T=0; GIE=0; Tdc1ms=1000; GIE=0;} //1second
PORTB=PB;}

void InitPorts(void){
  TRISB.0=0; // RB0 LED out
  TRISB.1=1; // RB1 button A
  TRISB.2=1; // RB2 button B
  TRISB.7=0; // RB7 BA LED
  TRISC =0; } // RC7..RC0  output

void InitInt(void){
  TMR0IE=1; TMR0IF=0; TMR0IP=0;
  IPEN=0; PEIE=1; GIE=1;}

void Blink(void){
  static uns16 BAcount;
  if(BAcount<200) PORTB.7=0; // BA LED on
  else PORTB.7=1; // BA LED off
  BAcount++; if(BAcount>1000) BAcount=0;}


void main(void){
  SS=0;
  Tdc1ms=0; TR=0;
  InitPorts();
  InitInt(); TMR0IF=1;
  // wait 100ms for LCD reset
  {char i=200; do{LCDw2u(250);}while(--i);}
  PrintLCD("\xff\x02\x82\x28\x01\x03\x0C\x06");
  PrintLCD("\x80Hello  ");

  do{ // mainloop
    if(Tint){
       GIE=0;
       Tdc1ms-=Tint; Tint=0;
       if(Tdc1ms<=0) {T=1; Tdc1ms=0;}
       GIE=1;
       Blink();
       }
    FSM();
    }while(1);
  }
```

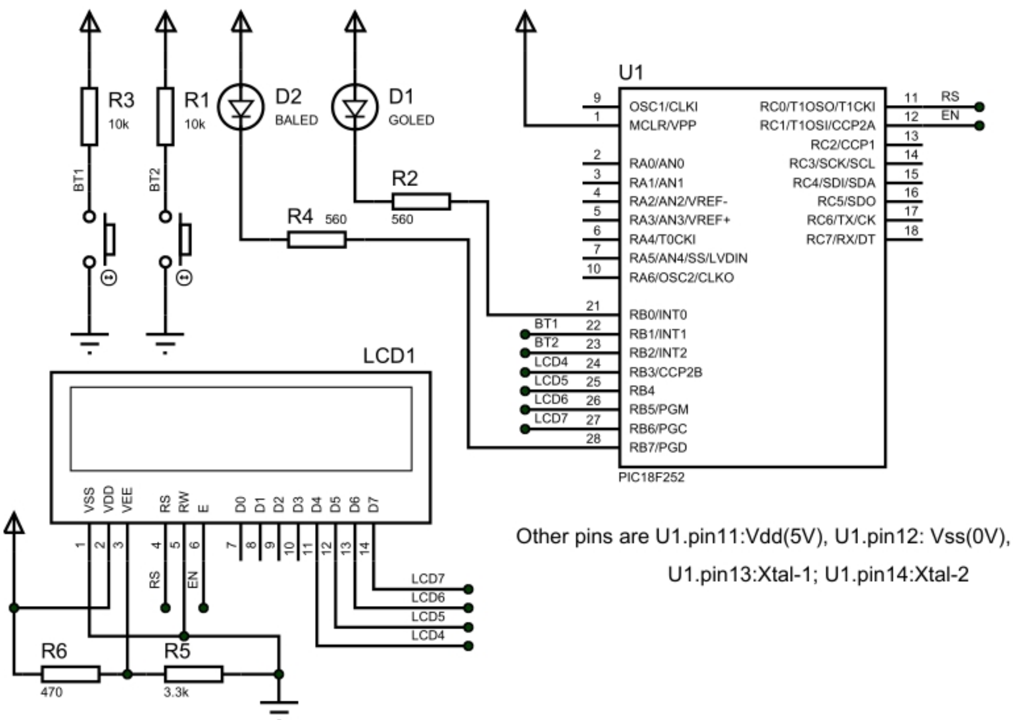


Fig 4 Circuit for Who-is-fast game with PIC18F252

After Lab Report

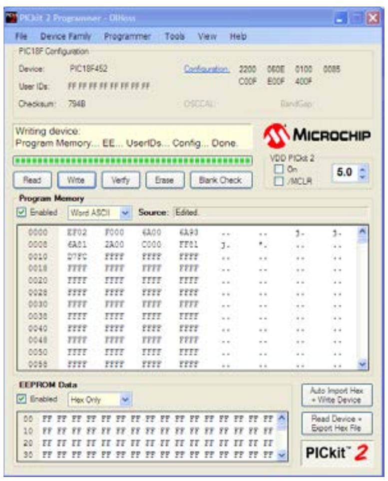
You shall design the individual project homework (Timer Project / Application-6) using FSM methods and support your design with test procedures.

Send you report together with a short conclusion on what you've learnt to CMPE423LAB@GMAIL.COM in two weeks after the project assignment.

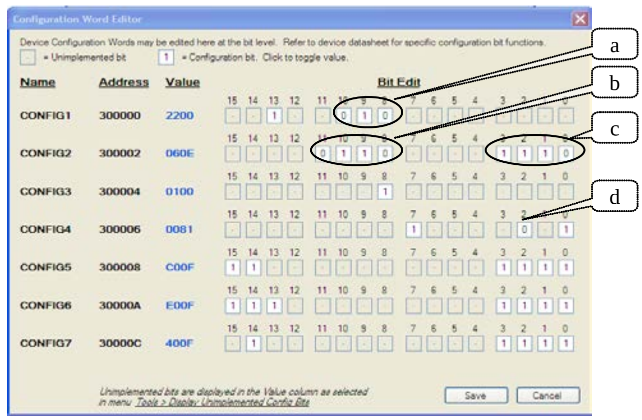
Programming the code into an IC device

Attach PIC-Kit-2 programmer to a USB port, and start PIC-Kit-2 application in Windows-XP or -7. The application shall open PIC-Kit-2 window on the display.

1- Select Device **Family>>PIC18**
2- Place the **PIC18F452 IC** to the ZIF (zero-insertion-force) IC-socket. Be sure the notch is pointing up (aligned to zif-arm).
3- Import the HEX file by **file>>import-hex** selecting it with import-browser. It will warn you that the configuration is not set properly.



PIC-Kit-2 programmer window



PIC-Kit-2 programmer PIC18F452 configuration settings. a- for HS crystal, b- for brownout reset at 4.2V and to enable power-on timer , c- to disable watchdogtimer, d- to disable Low-Voltage Programming (RB5 becomes available for i/o)

4- Set the configuration to have 2200 060E 0100 0081 on the first line. It corresponds to: HS-crystal (20MHz), Brown-out-reset at 4.5V, Watchdog-timer disabled, and Low-Voltage Programming disabled. These settings are critical to have proper operation.
5- Start programming by clicking on **write** button.
6- Take out the IC from the socket and insert it to the proto board.