FINAL PROJECT

EE/CS 120B

Priscilla Vuong (860-94-3311)

Email: pvuon002@ucr.edu

Partner: Michael Ferro

Email: mferr008@ucr.edu

Date: March 22, 2013

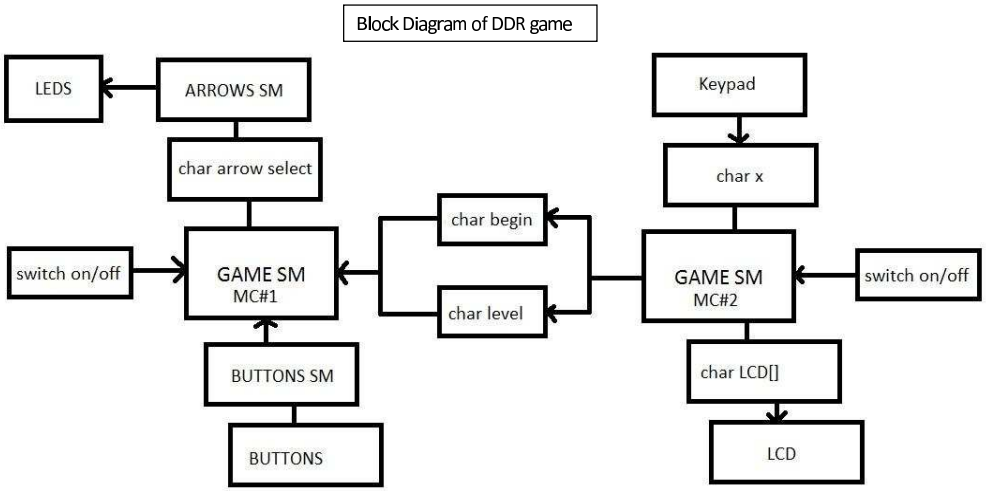
Title: DDR Game

"We, Priscilla Vuong and Michael Ferro, attest that this document and the work described herein is entirely our own work except where explicitly indicated otherwise."

Objective:

The course, "Introduction to Embedded Systems", includes a final project which involves using all knowledge gained from labs and lectures to create a game consisted of an AVR microcontroller and periphery.

High level description:



The game DDR works like the original DDR game found in arcades. In the original game, arrows fall with music playing in the background while players step on arrows to match the falling arrows. The game is modified to be played with buttons matching the arrows as they reach the bottom of the LED matrix.

The game starts when the on/off switch is turned on, which is connected to the power supply. This will turn on the LCD, which will display a greeting and ask the player to select a mode. The player then selects from two modes from the keypad: Easy or Hard. After a selection, the LCD will display the level to show that the press was read. The LCD will then display the amount of lives the player has, and the game shortly begins right after.

In the easy mode, the arrows fall at a slower rate allowing the player to easily win; the hard mode will have the arrows fall at a slightly quicker rate. The player must hit the corresponding buttons as they fall to the bottom. Two separate LED lights display a correct or incorrect hit. A correct hit is represented by an orange LED, while an incorrect hit is represented by a pink LED. A total of ten arrows fall for both modes. At the end, a win is represented by both the orange and pink LEDs lit as well as a happy face displayed on the LED matrix. A loss is represented by a sad face displayed on the matrix. The game is restarted by turning the game off then back on with the switch.

The block diagram displays the layout of the game. The state machines used for the project are included.

User Guide

Procedure:

The player turns on the game using the switch located below the LED matrix. After that, any components connected to microcontroller 1 remains off and only microcontroller 2 begins. The LCD connected to microcontroller 2 displays a greeting and will prompt the player to select a mode. The player will select a mode using the keypad where the buttons are as labeled, “Easy” and “Hard”. After a selection, the LCD will display the mode that the player has chosen and proceed to tell the player that he/she will have five lives. Slightly after this message, the player should direct his/her attention to the LED matrix, since the arrows will begin to generate and descend. The player will then proceed to attempt to coordinate their button presses with the falling arrows. The player will win as long as they can get through 10 falling arrows without losing all their lives. An error will light up a pink LED, and a correct press will light up an orange LED. Upon a success, both pink and orange LEDs will be lit, and a happy face will be displayed on the LED matrix. The opposite happens upon a loss.

Input:

The input for this game is the number pad for a mode selection as well as four buttons for the arrow presses. The input for power is the switch, which will turn on/off power for the whole game. When the game is turned on, any presses from the buttons connected to microcontroller one will not be an interference.

Output:

The output for this game is the LCD screen and the LED matrix. The LCD screen displays a greeting, instructions, and mode selection. The LED matrix displays a sequence of random arrows.

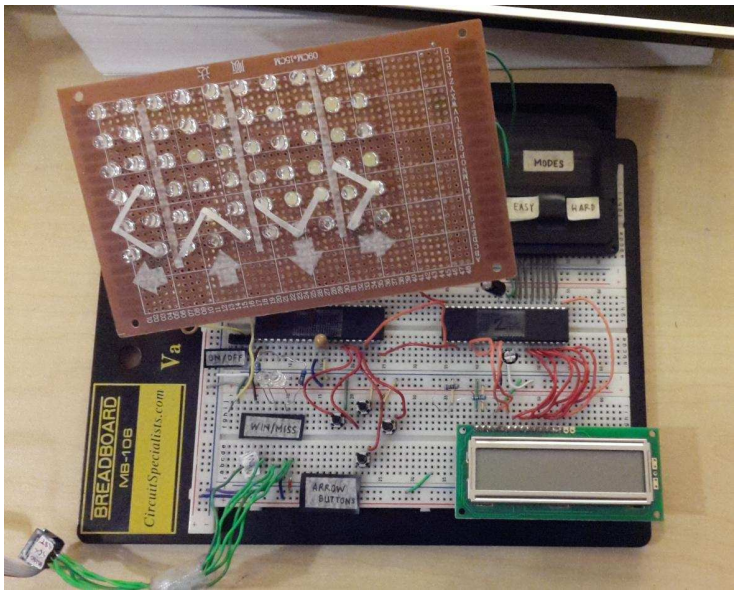
Technical Design

Microcontroller #1

	PORTA		PORTB		PORTC		PORTD	
0	OUTPUT	MATRIX			OUTPUT	MATRIX	INPUT	BUTTONS
1	OUTPUT	MATRIX			OUTPUT	MATRIX	INPUT	BUTTONS
2	OUTPUT	MATRIX			OUTPUT	MATRIX	INPUT	BUTTONS
3	OUTPUT	MATRIX	OUTPUT	LED	OUTPUT	MATRIX	INPUT	BUTTONS
4	OUTPUT	MATRIX	OUTPUT	LED	OUTPUT	MATRIX		
5	OUTPUT	MATRIX			OUTPUT	MATRIX		
6	OUTPUT	MATRIX			OUTPUT	MATRIX		
7	OUTPUT	MATRIX			OUTPUT	MATRIX		

Microcontroller #2

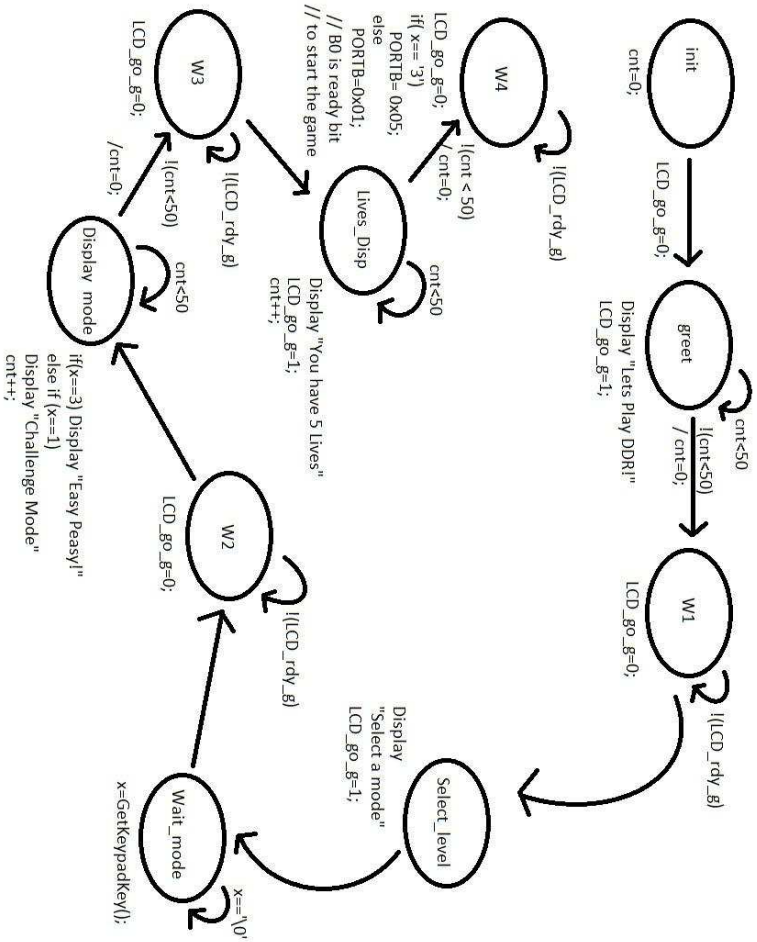
	PORTA		PORTB		PORTC		PORTD	
0			OUTPUT	READY BIT	INPUT	KEYPAD	OUTPUT	LCD
1					INPUT	KEYPAD	OUTPUT	LCD
2			OUTPUT	LEVEL BIT	INPUT	KEYPAD	OUTPUT	LCD
3					INPUT	KEYPAD	OUTPUT	LCD
4			OUTPUT	LCD control	INPUT	KEYPAD	OUTPUT	LCD
5			OUTPUT	LCD control	INPUT	KEYPAD	OUTPUT	LCD
6					INPUT	KEYPAD	OUTPUT	LCD
7					INPUT	KEYPAD	OUTPUT	LCD



Breadboard fully wired

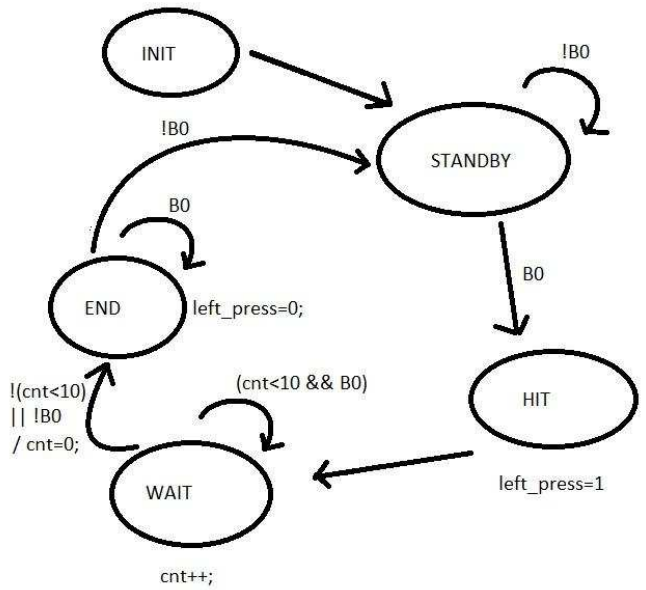
SYNCH SM NAME	DESCRIPTION
LEFT	-DECREMENTS Y COORDINATES, -CHECKS FOR BUTTON PRESSES -OUTPUTS LIGHTS BASSED ON CHECK -RESETS Y COORDINATES AT THE END -SETS FLAG FOR NEXT ACTION
RIGHT	SAME AS ABOVE
UP	SAME AS ABOVE
DOWN	SAME AS ABOVE
LI_TICK	LCD INITIALIZATION & DISPLAY
LT_TICK	-DISPLAYS LCD MESSAGES -WAITS FOR A KEYPAD PRESS -SENDS READY AND LEVEL BIT
CHECK_LEFT	-WAITS UNTIL D0 IS PRESSED -SETS THE GLOBAL VARIABLE LEFT_PRESS TO 1 -PREVENTS WRONG MULTIPLE/CONCURRENT PRESSES -PREVENTS HOLDING THE BUTTON
CHECK_RIGHT	-WAITS UNTIL D3 IS PRESSED -SETS THE GLOBAL VARIABLE RIGHT_PRESS TO 1 -PREVENTS WRONG MULTIPLE/CONCURRENT PRESSES -PREVENTS HOLDING THE BUTTON
CHECK_UP	-WAITS UNTIL D1 IS PRESSED -SETS THE GLOBAL VARIABLE UP_PRESS TO 1 -PREVENTS WRONG MULTIPLE/CONCURRENT PRESSES -PREVENTS HOLDING THE BUTTON
CHECK_DOWN	-WAITS UNTIL D2 IS PRESSED -SETS THE GLOBAL VARIABLE DOWN_PRESS TO 1 -PREVENTS WRONG MULTIPLE/CONCURRENT PRESSES -PREVENTS HOLDING THE BUTTON

LCD SM

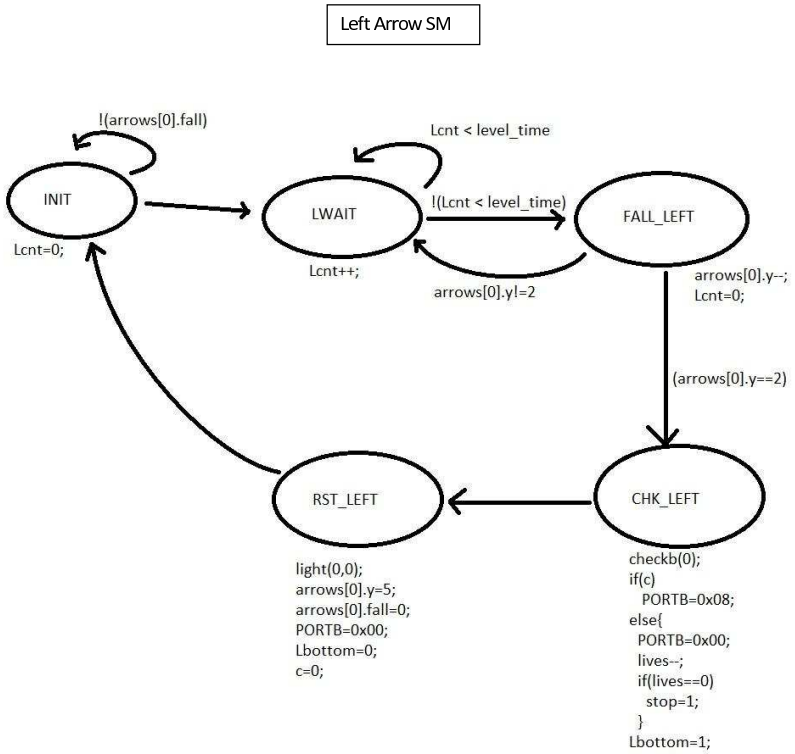


The **LCD SM** displays a greeting message then waits for the LCD ready bit to be ready. It then continues on to display messages and waits for a ready bit. It then retrieves a key press using the keypad function provided. Depending on the key press, it displays a certain message. The last state sends a ready bit and a level/mode selection to microcontroller #1.

Check_Left SM



This is the **check_left SM.** It waits for the corresponding button (ex. For Left, it is B0). When the button is pressed, the global variable left_press is set to 1. This allows for checking in the checking function of whether or not the button press is hit when the arrow is at the bottom or (y==2). The wait state counts to 10 to determine if the player is holding down the button. If the button is held, the variable will stay 1 for up to 10. It will clear the variable at the End state. The check_up, check_right,check_down SMs are equivalent to the one shown. The only differences are in the PORTs that the buttons are connected to.



This is the **left arrow state machine.** It decrements the y coordinate based on the speed it is given, in the global variable, "level_time", which is set depending on the mode the player chooses. Since the arrows are drawn based on the point of their arrow points, only the original (x,y) point is needed and only the y point needs to be decremented. In the chk_left state, a function checkb is called with a parameter of 0,1, 2, or 3 is passed in. 0 represents left, 1 represents up, 2 represents down, and 3 represents right. The variable 'c' is returned with 1 or 0 based on if the press was made when y==2. Lastly, the global variable "Lbottom" is set a A life is decremented accordingly if the press was not made at y==2. The up, down, and right state machines work the same except for the different indexes they are represented by.

Results and Performance Evaluation

Testing:

Since the main point of my game is to have the arrows descend on the LED matrix and for the button presses to be well coordinated with the arrows, those two aspects were testing repeatedly. The calls made to the arrow state machines were tested to determine which order of the calls would be accurate. A wait state had to be added into each arrow state machine because of the short system period of 10 ms. The function, draw(), which used the function, light(), to light up the arrows based on their points, had to be within the while(!TimerFlag) loop, to ensure a solid lit up LED. Based on testing, the system period had to be at most 5 ms to ensure a solid lit up LED arrow. For this reason, putting it inside of the TimerFlag loop allows it to be run as fast as possible without having too short of a system period.

For the arrows to fall, each arrow state machine was called. Testing the arrows had shown that a reset state was necessary in all the arrow state machines. Without the reset state, the arrow would not be able to be recalled in a sequence. In addition, a bottom variable also plays an important part, since arrows are not chosen again until the particular arrow has reached the bottom of the matrix.

The button presses in coordination to the arrows are also a main part in the game. Throughout many tests, a player would be able to hold all four buttons or any button combinations containing the correct button, and it would be registered as a correct press. For example, if a up arrow descends, and a player holds down the up and left arrow, it would still be registered as a correct hit. Since it was not a correct hit, creating a separate button check state machine for each button was necessary. These state machines would then set the individual global variables made for the presses if the correct button was hit.

Lastly, there were also tests for the game to only start after the LCD has displayed the last message, which informs the player of the number of lives he/she has. The LCD state machine, LT_TICK(), would send two bits into PORTD (D4 and D6) which would allow the game to start as well as set a speed for the falling of arrows.

There was an issue where the bits were not correctly being read. This was tested for a number of hours before inputting a while loop inside of the main while loop fixed the issue. In addition, a void function was put in the header file of the arrows to set “level_time”.

Bugs

There are not many bugs after the numerous amounts of tests ran on the game. There are only two known bugs left over. One bug includes the orange and pink LEDs staying lit for some presses and only flashing or not appearing for some button presses. Checking the code and state machines, I was not able to determine the cause of this issue. The second bug is that the button presses for up, down, and right are still registered as correct if the player held the correct button down, even though the arrow has not reached the bottom. This is supposed to be registered as incorrect, since holding the arrow down would create a very easy game environment for a win. The state machine to check the button presses only works 100% correctly for the left button press and not for the other three button presses. However, this is slightly unnoticeable because it will not allow the player to select a wrong button or press down two or more buttons at once.

Conclusion

The game sounds extremely simple to do, but more bugs occurred than expected. Deciding a method to light the LED matrix took longer than expected as well. My initial method turned out to be incorrect as that method would not allow me to light up more than one arrow at a time. It also would not have had the arrows descend as smoothly. Conjuring up arrow state machines was not too difficult after figuring out the way the LED matrix will be accessed. Bugs for the button presses was a major issue, which prevented a more creative game of having multiple arrows descend at the same time. Secondly, there was an issue in accessing the keypad presses. The solution was to uncheck jtag, which I had forgotten to do with my second microcontroller. This caused a delay in turning in the project on time because submitting a project with many bugs and two missing components was not acceptable. Most of the bugs were fixed and an extra level was added during the extension time upon fixing the keypad issue. In addition to the extra level added, a small display part was added at the end of the game depending on whether the player wins or loses.

YouTube Link: http://youtu.be/6lMlgBGXHl0

```
#ifndef _ALL_ARROWS_H_
#define _ALL_ARROWS_H_

#include <check_left.h>
#include <check_up.h>
#include <check_down.h>
#include <check_right.h>

void check_level(){
        if(mksnd)
        {
                level_time=30;

        }
        else
        {
                level_time= 65;

        }
}

enum Lstates {Linit,Lwait,fall_left,chk_left,rst_left} Lstate;
unsigned char Lbottom=0; //if reaches bottom =1; ready to re-fall
unsigned long Lcnt;

void checkb ( unsigned char in){
        if(in==0 )
        {
                if(arrows[in].y==2 && left_press && !up_press && !down_press &&
!right_press)
                        c=1;
                else
                        c=0;
        }

        else if(in==1)
        {
                if(arrows[in].y==2 && up_press && !left_press && !down_press &&
!right_press)
                        c=1;
                else
                        c=0;
        }

        else if(in==2)
        {
                if(arrows[in].y==1 && !up_press && !left_press && down_press &&
!right_press)
                        c=1;
                else
                        c=0;
        }

        else if(in==3)
        {
                if(arrows[in].y==2 && !up_press && !left_press && !down_press &&
 right_press)
                        c=1;
                else
                        c=0;
        }
        else if(in==4)
        {
                if(arrows[in].y==2 && up_press && left_press && !down_press && r
ight_press)
                        c=1;
                else
                        c=0;
        }
}//end of checkb

void left()
{
```



```
        check_level();
        switch(Lstate){
                case Linit:
                        if(!(arrows[0].fall) )
                                Lstate= Linit;
                        else
                                Lstate= Lwait;
                        break;

                case fall_left:
                        if(arrows[0].y==2 )
                                Lstate= chk_left;
                        else
                                Lstate= Lwait;
                        break;

                case Lwait:
                        if( (Lcnt<level_time) )
                                Lstate= Lwait;
                        else if ( !(Lcnt<level_time) /*&& (arrows[0].y!=2)*/ ){
                                Lstate= fall_left;
                                Lcnt=0;
                        }
                        break;

                case chk_left:
                                Lstate= rst_left;
                        break;

                case rst_left:
                        Lstate= Linit;
                        break;

                default:
                        break;
        } // end of left transitions

        switch(Lstate){
                case Linit:
                        break;
                case fall_left:
                        arrows[0].y--;
                        Lcnt=0;
                        break;

                case Lwait:
                        Lcnt++;
                        break;

                case chk_left:
                        checkb(0);
                        if(c){
                                PORTB=0x08;

                        }
                        else{
                                PORTB=0x10;
                                lives--;
                                if(lives==0){
                                        stop=1;
                                }
                        }
                        Lbottom=1;
                        break;

                case rst_left:
                        light(0,0);
                        arrows[0].y=5;
                        arrows[0].fall=0;
                        PORTB=0x00;
                        Lbottom=0;
                        Lcnt=0;
                        c=0; // reset c
```

```
                        break;
        }// end of left actions
}

enum Ustates{Uinit,Uwait,fall_up,chk_up,rst_up} Ustate;
unsigned char Ubottom=0;//if reaches bottom =1; ready to re-fall
unsigned long Ucnt;

void up()
{
        check_level();
        switch(Ustate){
                case Uinit:
                if(!(arrows[1].fall) )
                        Ustate= Uinit;
                else
                        Ustate= Uwait;
                break;

                case fall_up:
                if(arrows[1].y==2)
                Ustate= chk_up;
                else
                Ustate= Uwait;
                break;

                case Uwait:
                if( (Ucnt<level_time) )
                Ustate= Uwait;
                else if ( !(Ucnt<level_time) )
                Ustate= fall_up;
                break;

                case chk_up:
                Ustate= rst_up;
                break;
                case rst_up:
                Ustate= Uinit;
                break;
                default:
                break;
        } // end of up transitions

        switch(Ustate){
                case Uinit:
                        PORTB=0x00;
                case fall_up:
                        arrows[1].y--;
                        Ucnt=0;
                        break;
                case Uwait:
                        Ucnt++;
                        break;
                case chk_up:
                        checkb(1);
                        if(c)
                                PORTB=0x08;
                        else{
                                PORTB=0x10;
                                lives--;
                                if(lives==0){
                                        stop=1;

                                }
                        }
                        Ubottom=1;
                        break;
                case rst_up:
                        light(0,0);
                        arrows[1].y=6;
                        arrows[1].fall=0;
                        PORTB=0x00;
                        Ubottom=0;
```



```
                        c=0;
                        break;

        }// end of up actions
}

enum Dstates{Dinit,Dwait,fall_down,chk_down,rst_down} Dstate;
unsigned char Dbottom=0; //if reaches bottom =1; ready to re-fall
unsigned long Dcnt;

void down()
{
        check_level();
        switch(Dstate)
        {
                case Dinit:
                if(!(arrows[2].fall) )
                Dstate= Dinit;
                else
                Dstate= Dwait;
                break;

                case fall_down:
                if(arrows[2].y==1)
                Dstate= chk_down;
                else
                Dstate= Dwait;
                break;

                case Dwait:
                if( (Dcnt<level_time) )
                Dstate= Dwait;
                else if ( !(Dcnt<level_time) )
                Dstate= fall_down;
                break;

                case chk_down:
                Dstate= rst_down;
                break;
                case rst_down:
                Dstate= Dinit;
                break;
                default:
                break;
        } // end of down transitions

        switch(Dstate)
        {
                case fall_down:
                        arrows[2].y--;
                        Dcnt=0;
                        break;
                case Dwait:
                        Dcnt++;
                        break;
                case chk_down:
                        checkb(2);
                        if(c){
                                PORTB=0x08;

                        }
                        else{
                                PORTB=0x10;
                                lives--;
                                if(lives==0){
                                        stop=1;
                                }
                        }
                        Dbottom=1;
                        break;
                case rst_down:
                        light(0,0);
                        arrows[2].y=5;
```

```
                        arrows[2].fall=0;
                        PORTB=0x00;
                        Dbottom=0;
                        c=0;
                        break;

        } // end of down actions
}

enum Rstates{Rinit,Rwait,fall_right,chk_right,rst_right} Rstate;
unsigned char Rbottom=0; //if reaches bottom =1; ready to re-fall
unsigned long Rcnt;

void right()
{
        check_level();
        switch(Rstate)
        {
                case Rinit:
                        if(!(arrows[3].fall) )
                                Rstate= Rinit;
                        else
                                Rstate= Rwait;
                        break;

                case fall_right:
                        if(arrows[3].y==2)
                                Rstate= chk_right;
                        else
                                Rstate= Rwait;
                        break;

                case Rwait:
                        if( (Rcnt<level_time) )
                                Rstate= Rwait;
                        else if ( !(Rcnt<level_time) )
                                Rstate= fall_right;
                        break;

                case chk_right:
                        Rstate= rst_right;
                        break;

                case rst_right:
                        Rstate= Rinit;
                        break;
                default:
                break;
        } //end of right transitions

        switch(Rstate)
        {
                case fall_right:
                        arrows[3].y--;
                        Rcnt=0;
                        break;

                case Rwait:
                        Rcnt++;
                        break;

                case chk_right:
                        checkb(3);
                        if(c)
                                PORTB=0x08;
                        else{
                                PORTB=0x10;
                                lives--;
                                if(lives==0){
                                        stop=1;
                                }
                        }
                        Rbottom=1;
```



```
                        break;

                case rst_right:
                        light(0,0);
                        arrows[3].y=5;
                        arrows[3].fall=0;
                        PORTB=0x00;
                        Rbottom=0;
                        c=0;
                        break;

        } //end of right actions
}




#endif
```

```
1
2  // Permission to copy is granted provided that this header remains intact.
3  // This software is provided with no warranties.
4
5  ////////////////////////////////////////////////////////////////////////////////
6
7  #ifndef BIT_H
8  #define BIT_H
9
10 ////////////////////////////////////////////////////////////////////////////////
11 //Functionality - Sets bit on a PORTx
12 //Parameter: Takes in a uChar for a PORTx, the pin number and the binary value
13 //Returns: The new value of the PORTx
14 unsigned char SetBit(unsigned char pin, unsigned char number, unsigned char bin_
   value)
15 {
16         return (bin_value ? pin | (0x01 << number) : pin & ~(0x01 << number));
17 }
18
19 ////////////////////////////////////////////////////////////////////////////////
20 //Functionality - Gets bit from a PINx
21 //Parameter: Takes in a uChar for a PINx and the pin number
22 //Returns: The value of the PINx
23 unsigned char GetBit(unsigned char port, unsigned char number)
24 {
25         return ( port & (0x01 << number) );
26 }
27
28 #endif //BIT_H
```

```
#ifndef _CHECK_DOWN_H_
#define _CHECK_DOWN_H_
#include <bit.h>

enum DCStates{DCinit, DCstandby, DChit, DCwait, DCend} DCstate;
unsigned char DCcnt;

void down_pr(){
switch(DCstate){
        case DCinit:
                DCstate= DCstandby;
                break;

        case DCstandby:
                if( GetBit((~PIND),2) == 0) //not pressed
                        DCstate= DCstandby;
                else if( !(GetBit((~PIND),0)) && !(GetBit((~PIND),1)) && (GetBit
((~PIND),2)) && !(GetBit((~PIND),3)) )// if pressed go to DC hit
                        DCstate= DChit;
                break;

        case DChit:
                DCstate= DCwait;
                break;

        case DCwait:
                if(DCcnt < 10 && (GetBit((~PIND),2)) )
                // if still pressed then wait for 10 ms
                        DCstate= DCwait;
                else if ( !(DCcnt<10) || ( (GetBit((~PIND),2))== 0) )
                //if count is done OR button is let go
                        DCstate= DCend;
                break;

        case DCend:
                if((GetBit((~PIND),2)==1))
                        DCstate= DCend;
                else
                        DCstate= DCstandby;
                break;

        default:
                break;
} //end of transitions

switch(DCstate){
        case DCstandby:
                DCcnt=0;
                break;

        case DChit:
                down_press=1;
                break;

        case DCwait:
                DCcnt++;
                break;

        case DCend:
                down_press=0;
                break;

        default:
                break;
} // end of actions
}

#endif
```

```
#ifndef _CHECK_LEFT_H_
#define _CHECK_LEFT_H_
#include <bit.h>

enum LCStates{LCinit, LCstandby, LChit, LCwait, LCend} LCstate;
unsigned char LCcnt;

void left_pr(){
switch(LCstate){
        case LCinit:
                LCstate= LCstandby;
                break;

        case LCstandby:
                if( GetBit((~PIND),0) == 0 ) //not pressed
                        LCstate= LCstandby;
                        else if( (GetBit((~PIND),0)) && !(GetBit((~PIND),1)) &&
!(GetBit((~PIND),2)) && !(GetBit((~PIND),3)) )// if pressed go to LC hit
                        LCstate= LChit;
                break;

        case LChit:
                LCstate= LCwait;
                break;

        case LCwait:
                if(LCcnt < 10 && (GetBit((~PIND),0)==1) )
                // if still pressed then wait for 10 ms
                        LCstate= LCwait;
                else if ( !(LCcnt<10) ||  (GetBit((~PIND),0) == 0) )
                //if count is done OR button is let go
                        LCstate= LCend;
                break;

        case LCend:
                if((GetBit((~PIND),0)==1))
                        LCstate= LCend;
                else
                        LCstate= LCstandby;
                break;

        default:
                break;
} //end of transitions

switch(LCstate){
        case LCstandby:
                LCcnt=0;
                break;

        case LChit:
                left_press=1;
                break;

        case LCwait:
                LCcnt++;
                break;

        case LCend:
                left_press=0;
                break;

        default:
                break;
} // end of actions
}

#endif
```

```
#ifndef _CHECK_RIGHT_H_
#define _CHECK_RIGHT_H_
#include <bit.h>

enum RCStates{RCinit, RCstandby, RChit, RCwait, RCend} RCstate;
unsigned char RCcnt;

void right_pr()
{
        switch(RCstate){
                case RCinit:
                        RCstate= RCstandby;
                        break;

                case RCstandby:
                        if( GetBit((~PIND),3) == 0) //not pressed
                                RCstate= RCstandby;
                        else if( !(GetBit((~PIND),0)) && !(GetBit((~PIND),1)) &&
 !(GetBit((~PIND),2)) && (GetBit((~PIND),3)) )// if pressed go to RC hit
                                RCstate= RChit;
                        break;

                case RChit:
                        RCstate= RCwait;
                        break;

                case RCwait:
                        if(RCcnt < 10 && (GetBit((~PIND),3)) )
                        // if still pressed then wait for 10 ms
                                RCstate= RCwait;
                        else if ( !(RCcnt<10) || ( (GetBit((~PIND),3))== 0) )
                        //if count is done OR button is let go
                                RCstate= RCend;
                        break;

                case RCend:
                        if( (GetBit((~PIND),3)==1) )
                                RCstate= RCend;
                        else
                                RCstate= RCstandby;
                        break;

                default:
                        break;
        } //end of transitions

        switch(RCstate){
                case RCstandby:
                        RCcnt=0;
                        break;

                case RChit:
                        right_press=1;
                        break;

                case RCwait:
                        RCcnt++;
                        break;

                case RCend:
                        right_press=0;
                        break;

                default:
                        break;
        } // end of actions
}

#endif
```

```
#ifndef _CHECK_UP_H_
#define _CHECK_UP_H_
#include <bit.h>

enum UCStates{UCinit, UCstandby, UChit, UCwait, UCend} UCstate;
unsigned char UCcnt;

void up_pr(){
switch(UCstate){
        case UCinit:
                UCstate= UCstandby;
                break;

        case UCstandby:
                if( GetBit((~PIND),1) == 0 ) //not pressed
                        UCstate= UCstandby;
                        else if( !(GetBit((~PIND),0)) && (GetBit((~PIND),1)) &&
!(GetBit((~PIND),2)) && !(GetBit((~PIND),3)) )// if pressed go to LC hit
                        UCstate= UChit;
                break;

        case UChit:
                UCstate= UCwait;
                break;

        case UCwait:
                if( (UCcnt < 10) && (GetBit((~PIND),1)==1) )
                // if still pressed then wait for 10 ms
                        UCstate= UCwait;
                else if ( !(UCcnt<10) ||  (GetBit((~PIND),1) == 0) )
                //if count is done OR button is let go
                        UCstate= UCend;
                break;

        case UCend:
                if((GetBit((~PIND),1)==0))
                        UCstate= UCend;
                else
                        UCstate= UCstandby;
                break;

        default:
                break;
} //end of transitions

switch(UCstate){
        case UCstandby:
                UCcnt=0;
                up_press=0;
                break;

        case UChit:
                up_press=1;
                break;

        case UCwait:
                UCcnt++;
                break;

        case UCend:
                up_press=0;
                break;

        default:
                break;
} // end of actions
}

#endif
```

```
#ifndef __FALLING_ARROWS__
#define __FALLING_ARROWS__

//** THIS FILE DECREMENTS THE Y VALUE IN THE POINT OF THE ARROW ** //

//#include <light.h>
#include <check_button.h>

enum Fstates{Finit, fall_down, fall_others, check_press, reset_y} Fstate;


/*void falling(){*/

        switch(Fstate){
                case Finit:
                        if(arrows[indx].fall==0)
                                Fstate= Finit;
                        else if( (indx==2) && (arrows[indx].fall==1) )
                                Fstate= fall_down;
                        else if ( ( (indx==0) || (indx==1) || (indx==3) ) && (ar
rows[indx].fall==1) )
                                Fstate= fall_others;
                        break;

                case fall_down:
                        if(arrows[indx].y!=1 && arrows[indx].fall==1)
                                Fstate=fall_down;
                        else if (arrows[indx].y==1 && arrows[indx].fall==1)
                                Fstate= check_press;
                        break;

                case fall_others:
                        if(arrows[indx].y!=2 && arrows[indx].fall==1)
                                Fstate= fall_others;
                        else if (arrows[indx].y==2 && arrows[indx].fall==1)
                                Fstate= check_press;
                        break;

                case check_press:
                        Fstate= reset_y;
                        break;

                case reset_y:
                        break;
                default:
                        break;
        }

        switch(Fstate){
                case fall_down:
                        arrows[indx].y--;
                        break;

                case fall_others:
                        arrows[indx].y--;
                        break;

                case check_press:
                        if(check(arrows,indx))
                                PORTB=0x08;
                        else
                                PORTB=0x00;
                        break;

                case reset_y:
                        light(0,0);
                        if((indx==0)|| (indx==2) || (indx==3))
                                arrows[indx].y=5;
                        else if (indx==1)
                                arrows[indx].y=6;
                        arrows[indx].fall=0;
                        PORTB=0x00;
                        break;
```



```
        }
/*}*/


#endif
```

```
1
2   // Permission to copy is granted provided that this header remains intact.
3   // This software is provided with no warranties.
4
5   ////////////////////////////////////////////////////////////////////////////////
6
7   // Returns '\0' if no key pressed, else returns char '1', '2', ... '9', 'A', ...
8   // If multiple keys pressed, returns leftmost-topmost one
9   // Keypad must be connected to port C
10  // Keypad arrangement
11  //        Px4 Px5 Px6 Px7
12  //        col 1   2   3   4
13  //  row  _____________
14  //Px0 1 | 1 | 2 | 3 | A
15  //Px1 2 | 4 | 5 | 6 | B
16  //Px2 3 | 7 | 8 | 9 | C
17  //Px3 4 | * | 0 | # | D
18
19  #ifndef KEYPAD_H
20  #define KEYPAD_H
21
22  #include <bit.h>
23
24  // Keypad Setup Values
25  #define KEYPADPORT PORTC
26  #define KEYPADPIN  PINC
27  #define ROW1 0
28  #define ROW2 1
29  #define ROW3 2
30  #define ROW4 3
31  #define COL1 4
32  #define COL2 5
33  #define COL3 6
34  #define COL4 7
35
36  ////////////////////////////////////////////////////////////////////////////////
37  //Functionality - Gets input from a keypad via time-multiplexing
38  //Parameter: None
39  //Returns: A keypad button press else '\0'
40  unsigned char GetKeypadKey() {
41
42          // Check keys in col 1
43          KEYPADPORT = SetBit(0xFF,COL1,0); // Set Px4 to 0; others 1
44          asm("nop"); // add a delay to allow PORTx to stabilize before checking
45          if ( GetBit(~KEYPADPIN,ROW1) ) { return '1'; }
46          if ( GetBit(~KEYPADPIN,ROW2) ) { return '4'; }
47          if ( GetBit(~KEYPADPIN,ROW3) ) { return '7'; }
48          if ( GetBit(~KEYPADPIN,ROW4) ) { return '*'; }
49
50          // Check keys in col 2
51          KEYPADPORT = SetBit(0xFF,COL2,0); // Set Px5 to 0; others 1
52          asm("nop"); // add a delay to allow PORTx to stabilize before checking
53          if ( GetBit(~KEYPADPIN,ROW1) ) { return '2'; }
54          if ( GetBit(~KEYPADPIN,ROW2) ) { return '5'; }
55          if ( GetBit(~KEYPADPIN,ROW3) ) { return '8'; }
56          if ( GetBit(~KEYPADPIN,ROW4) ) { return '0'; }
57
58          // Check keys in col 3
59          KEYPADPORT = SetBit(0xFF,COL3,0); // Set Px6 to 0; others 1
60          asm("nop"); // add a delay to allow PORTx to stabilize before checking
61          if ( GetBit(~KEYPADPIN,ROW1) ) { return '3'; }
62          if ( GetBit(~KEYPADPIN,ROW2) ) { return '6'; }
63          if ( GetBit(~KEYPADPIN,ROW3) ) { return '9'; }
64          if ( GetBit(~KEYPADPIN,ROW4) ) { return '#'; }
65
66          // Check keys in col 4
67          KEYPADPORT = SetBit(0xFF,COL4,0); // Set Px7 to 0; others 1
68          asm("nop"); // add a delay to allow PORTx to stabilize before checking
69          if (GetBit(~KEYPADPIN,ROW1) ) { return 'A'; }
70          if (GetBit(~KEYPADPIN,ROW2) ) { return 'B'; }
71          if (GetBit(~KEYPADPIN,ROW3) ) { return 'C'; }
72          if (GetBit(~KEYPADPIN,ROW4) ) { return 'D'; }
73
```



```
74          return '\0';
75  }
76
77  #endif //KEYPAD_H
```

```
/*
** Permission to copy is granted provided that this header remains intact.
** This software is provided with no warranties.
*/

/*--------------------------------------------------------------------------*/

#ifndef LCD_8BIT_H
#define LCD_8BIT_H

#include <bit.h>

// Define LCD port assignments here so easier to change than if hardcoded below
#define LCD_DATA PORTD  // LCD 8-bit data bus
#define LCD_CTRL PORTB  // LCD needs 2-bits for control
#define LCD_RS   4              // LCD Reset pin
#define LCD_E    5              // LCD Enable pin

// Set by LCD interface synchSM, ready to display new string
unsigned char LCD_rdy_g = 0;
// Set by user synchSM wishing to display string in LCD_string_g
unsigned char LCD_go_g = 0;
// Filled by user synchSM, 16 chars plus end-of-string char
unsigned char LCD_string_g[16];
// Determine if the LCD will write a char (0) or a string (1)
unsigned char LCD_write_str = 1;
// Position to write a single character to the LCD Position (0~15)
unsigned char LCD_char_pos = 0;

void LCD_WriteCmdStart(unsigned char cmd) {
        LCD_CTRL = SetBit(LCD_CTRL,LCD_RS, 0);
        LCD_DATA = cmd;
        LCD_CTRL = SetBit(LCD_CTRL,LCD_E, 1);
}
void LCD_WriteCmdEnd() {
        LCD_CTRL = SetBit(LCD_CTRL,LCD_E, 0);
}
void LCD_WriteDataStart(unsigned char Data) {
        LCD_CTRL = SetBit(LCD_CTRL,LCD_RS,1);
        LCD_DATA = Data;
        LCD_CTRL = SetBit(LCD_CTRL,LCD_E, 1);
}
void LCD_WriteDataEnd() {
        LCD_CTRL = SetBit(LCD_CTRL,LCD_E, 0);
}
void LCD_Cursor(unsigned char column) {
        if ( column < 8 ) {
                LCD_WriteCmdStart(0x80+column);
        }
        else {
                LCD_WriteCmdStart(0xB8+column);
        }
}

enum LI_States { LI_Init1, LI_Init2, LI_Init3, LI_Init4, LI_Init5,
                LI_WaitDisplayString, LI_PositionCursor, LI_DisplayChar, LI_Wait
Go0 };

int LCDI_SMTick(int state) {
        static unsigned char i;
        switch(state) { // Transitions
        case -1:
                state = LI_Init1;
                break;
        case LI_Init1:
                state = LI_Init2;
                i=0;
                break;
        case LI_Init2:
                if (i<10) { // Wait 100 ms after power up
                        state = LI_Init2;
                }
                else {
```



```
                        state = LI_Init3;
                }
                break;
        case LI_Init3:
                state = LI_Init4;
                LCD_WriteCmdEnd();
                break;
        case LI_Init4:
                state = LI_Init5;
                LCD_WriteCmdEnd();
                break;
        case LI_Init5:
                state = LI_WaitDisplayString;
                LCD_WriteCmdEnd();
                break;
        case LI_WaitDisplayString:
                if (!LCD_go_g) {
                        state = LI_WaitDisplayString;
                }
                else if (LCD_go_g) {
                        LCD_rdy_g = 0;
                        state = LI_PositionCursor;
                        i=0;
                }
                break;
        case LI_PositionCursor:
                state = LI_DisplayChar;
                LCD_WriteCmdEnd();
                break;
        case LI_DisplayChar:
                if (i<16 && LCD_write_str) {
                        state = LI_PositionCursor;
                        LCD_WriteDataEnd();
                        i++;
                }
                else {
                        state = LI_WaitGo0;
                        LCD_WriteDataEnd();
                }
                break;
        case LI_WaitGo0:
                if (!LCD_go_g) {
                        state = LI_WaitDisplayString;
                }
                else if (LCD_go_g) {
                        state = LI_WaitGo0;
                }
                break;
        default:
                state = LI_Init1;
        } // Transitions

        switch(state) { // State actions
        case LI_Init1:
                LCD_rdy_g = 0;
                break;
        case LI_Init2:
                i++; // Waiting after power up
                break;
        case LI_Init3:
                LCD_WriteCmdStart(0x38);
                break;
        case LI_Init4:
                LCD_WriteCmdStart(0x0F);
                break;
        case LI_Init5:
                LCD_WriteCmdStart(0x01); // Clear
                break;
        case LI_WaitDisplayString:
                LCD_rdy_g = 1;
                break;
        case LI_PositionCursor:
                if ( LCD_write_str ) {
```

```
146                         LCD_Cursor(i);
147                 } else {
148                         LCD_Cursor(LCD_char_pos);
149                 }
150                 break;
151         case LI_DisplayChar:
152                 if ( LCD_write_str ) {
153                         LCD_WriteDataStart(LCD_string_g[i]);
154                 } else {
155                         LCD_WriteDataStart(LCD_string_g[0]);
156                 }
157                 break;
158         case LI_WaitGo0:
159                 break;
160         default:
161                 break;
162         } // State actions
163         return state;
164 }
165
166 #endif //LCD_8BIT_H
```

```
#ifndef _LIGHT_H_
#define _LIGHT_H_
#include <util/delay.h>

//*THIS FILE CONTAINS A STRUCT OF UNSIGNED CHARS X AND Y
// WHICH REPRESENTS A X,Y COORDINATE OF EACH LED LIGHT IN THE MATRIX

// THE FUNCTION LOAD_VALS ACCEPTS AN ARRAY, ARRAY INDEX, X AND Y
// AND LOADS ALL VALUES INTO THE ARRAY PASSED IN

// THE FUNCTION RST RESETS ALL ORIGINAL VALUES OF THE POINTS OF EACH ARROW

// THE FUNCTION LIGHT LIGHTS EACH LED SPECIFIED IN THE X AND Y COORDINATE

// THE FUNCTION DRAW DRAWS EACH ARROW ACCORDING TO THE POINT COORDINATE OF THE A
RROW
// IN THE DRAW FUNCTION, ANOTHER FUNCTION CHECK, CHECKS FOR CORRECTNESS OF
// BUTTON PRESS.

//*




void load_vals(coord a[],unsigned char loc, unsigned char x_loc, unsigned char y
_loc, unsigned char f)
{
        a[loc].x= x_loc;
        a[loc].y= y_loc;
        a[loc].fall= f;
}

void light(unsigned char x, unsigned char y){
        switch(x){
                case 0:
                        DDRA,DDRC = 0x00;
                        PORTA,PORTC = 0x00;
                        break;
                case 1:
                        switch(y){
                                case 1:                 //1, 1
                                        DDRA= 0x01;
                                        DDRC= 0x20;
                                        PORTA= 0x00;
                                        PORTC= 0x20;
                                        asm("nop");
                                        break;
                                case 2:                 //1, 2
                                        DDRA = 0x01;
                                        DDRC = 0x10;
                                        PORTA = 0x00;
                                        PORTC = 0x10;
                                        asm("nop");
                                        break;
                                case 3:                 //1, 3
                                        DDRA = 0x01;
                                        DDRC = 0x08;
                                        PORTC = 0x08;
                                        PORTA = 0x00;
                                        asm("nop");
                                        break;
                                case 4:                 //1, 4
                                        DDRA = 0x01;
                                        DDRC = 0x04;
                                        PORTC = 0x04;
                                        PORTA = 0x00;
                                        asm("nop");
                                        break;
                                case 5:                 //1, 5
                                        DDRA = 0x01;
                                        DDRC = 0x02;
                                        PORTA = 0x00;
                                        PORTC = 0x02;
```



```
                                        asm("nop");
                                        break;
                                case 6:                 //1, 6
                                        DDRA = 0x01;
                                        DDRC = 0x01;
                                        PORTA = 0x00;
                                        PORTC = 0x01;
                                        asm("nop");
                                        break;
                        }
                        break;
                case 2:
                        switch(y){
                                case 1:         //2, 1
                                DDRA = 0x02;
                                DDRC = 0x20;
                                PORTA = 0x00;
                                PORTC = 0x20;
                                asm("nop");
                                break;
                                case 2:         //2, 2
                                DDRA = 0x02;
                                DDRC = 0x10;
                                PORTA = 0x00;
                                PORTC = 0x10;
                                asm("nop");
                                break;
                                case 3:         //2, 3
                                DDRA = 0x02;
                                DDRC = 0x08;
                                PORTA = 0x00;
                                PORTC = 0x08;
                                asm("nop");
                                break;
                                case 4:         //2, 4
                                DDRA = 0x02;
                                DDRC = 0x04;
                                PORTA = 0x00;
                                PORTC = 0x04;
                                asm("nop");
                                break;
                                case 5:         //2, 5
                                DDRA = 0x02;
                                DDRC = 0x02;
                                PORTA = 0x00;
                                PORTC = 0x02;
                                asm("nop");
                                break;
                                case 6:         //2, 6
                                DDRA = 0x02;
                                DDRC = 0x01;
                                PORTA = 0x00;
                                PORTC = 0x01;
                                asm("nop");
                                break;
                        }
                        break;
                case 3:
                        switch(y){
                                case 1:         //3, 1
                                DDRA = 0x04;
                                DDRC = 0x20;
                                PORTA = 0x00;
                                PORTC = 0x20;
                                asm("nop");
                                break;
                                case 2:         //3, 2
                                DDRA = 0x04;
                                DDRC = 0x10;
                                PORTA = 0x00;
                                PORTC = 0x10;
                                asm("nop");
                                break;
```

```
145                             case 3:         //3, 3
146                             DDRA = 0x04;
147                             DDRC = 0x08;
148                             PORTA = 0x00;
149                             PORTC = 0x08;
150                             asm("nop");
151                             break;
152                             case 4:         //3, 4
153                             DDRA = 0x04;
154                             DDRC = 0x04;
155                             PORTA = 0x00;
156                             PORTC = 0x04;
157                             asm("nop");
158                             break;
159                             case 5:         //3, 5
160                             DDRA = 0x04;
161                             DDRC = 0x02;
162                             PORTA = 0x00;
163                             PORTC = 0x02;
164                             asm("nop");
165                             break;
166                             case 6:         //3, 6
167                             DDRA = 0x04;
168                             DDRC = 0x01;
169                             PORTA = 0x00;
170                             PORTC = 0x01;
171                             asm("nop");
172                             break;
173                     }
174                     break;
175             case 4:
176                     switch(y){
177                             case 1:         //4, 1
178                             DDRA = 0x08;
179                             DDRC = 0x20;
180                             PORTA = 0x00;
181                             PORTC = 0x20;
182                             asm("nop");
183                             break;
184                             case 2:         //4, 2
185                             DDRA = 0x08;
186                             DDRC = 0x10;
187                             PORTA = 0x00;
188                             PORTC = 0x10;
189                             asm("nop");
190                             break;
191                             case 3:         //4, 3
192                             DDRA = 0x08;
193                             DDRC = 0x08;
194                             PORTA = 0x00;
195                             PORTC = 0x08;
196                             asm("nop");
197                             break;
198                             case 4:         //4, 4
199                             DDRA = 0x08;
200                             DDRC = 0x04;
201                             PORTA = 0x00;
202                             PORTC = 0x04;
203                             asm("nop");
204                             break;
205                             case 5:         //4, 5
206                             DDRA = 0x08;
207                             DDRC = 0x02;
208                             PORTA = 0x00;
209                             PORTC = 0x02;
210                             asm("nop");
211                             break;
212                             case 6:         //4, 6
213                             DDRA = 0x08;
214                             DDRC = 0x01;
215                             PORTA = 0x00;
216                             PORTC = 0x01;
217                             asm("nop");
```



```
218                             break;
219                     }
220                     break;
221             case 5:
222                     switch(y){
223                             case 1:         //5, 1
224                             DDRA = 0x10;
225                             DDRC = 0x20;
226                             PORTA = 0x00;
227                             PORTC = 0x20;
228                             asm("nop");
229                             break;
230                             case 2:         //5, 2
231                             DDRA = 0x10;
232                             DDRC = 0x10;
233                             PORTA = 0x00;
234                             PORTC = 0x10;
235                             asm("nop");
236                             break;
237                             case 3:         //5, 3
238                             DDRA = 0x10;
239                             DDRC = 0x08;
240                             PORTA = 0x00;
241                             PORTC = 0x08;
242                             asm("nop");
243                             break;
244                             case 4:         //5, 4
245                             DDRA = 0x10;
246                             DDRC = 0x04;
247                             PORTA = 0x00;
248                             PORTC = 0x04;
249                             asm("nop");
250                             break;
251                             case 5:         //5, 5
252                             DDRA = 0x10;
253                             DDRC = 0x02;
254                             PORTA = 0x00;
255                             PORTC = 0x02;
256                             asm("nop");
257                             break;
258                             case 6:         //5, 6
259                             DDRA = 0x10;
260                             DDRC = 0x01;
261                             PORTA = 0x00;
262                             PORTC = 0x01;
263                             asm("nop");
264                             break;
265                     }
266                     break;
267             case 6:
268                     switch(y){
269                             case 1:         //6, 1
270                             DDRA = 0x20;
271                             DDRC = 0x20;
272                             PORTA = 0x00;
273                             PORTC = 0x20;
274                             asm("nop");
275                             break;
276                             case 2:         //6, 2
277                             DDRA = 0x20;
278                             DDRC = 0x10;
279                             PORTA = 0x00;
280                             PORTC = 0x10;
281                             asm("nop");
282                             break;
283                             case 3:         //6, 3
284                             DDRA = 0x20;
285                             DDRC = 0x08;
286                             PORTA = 0x00;
287                             PORTC = 0x08;
288                             asm("nop");
289                             break;
290                             case 4:         //6, 4
```

```
                        DDRA = 0x20;
                        DDRC = 0x04;
                        PORTA = 0x00;
                        PORTC = 0x04;
                        asm("nop");
                        break;
                        case 5:         //6, 5
                        DDRA = 0x20;
                        DDRC = 0x02;
                        PORTA = 0x00;
                        PORTC = 0x02;
                        asm("nop");
                        break;
                        case 6:         //6, 6
                        DDRA = 0x20;
                        DDRC = 0x01;
                        PORTA = 0x00;
                        PORTC = 0x01;
                        asm("nop");
                        break;
                }
                break;
        case 7:
                switch(y){
                        case 1:         //7, 1
                        DDRA = 0x40;
                        DDRC = 0x20;
                        PORTA = 0x00;
                        PORTC = 0x20;
                        asm("nop");
                        break;
                        case 2:         //7, 2
                        DDRA = 0x40;
                        DDRC = 0x10;
                        PORTA = 0x00;
                        PORTC = 0x10;
                        asm("nop");
                        break;
                        case 3:         //7, 3
                        DDRA = 0x40;
                        DDRC = 0x08;
                        PORTA = 0x00;
                        PORTC = 0x08;
                        asm("nop");
                        break;
                        case 4:         //7, 4
                        DDRA = 0x40;
                        DDRC = 0x04;
                        PORTA = 0x00;
                        PORTC = 0x04;
                        asm("nop");
                        break;
                        case 5:         //7, 5
                        DDRA = 0x40;
                        DDRC = 0x02;
                        PORTA = 0x00;
                        PORTC = 0x02;
                        asm("nop");
                        break;
                        case 6:         //7, 6
                        DDRA = 0x40;
                        DDRC = 0x01;
                        PORTA = 0x00;
                        PORTC = 0x01;
                        asm("nop");
                        break;
                }
                break;
        case 8:
                switch(y){
                        case 1:         //8, 1
                        DDRA = 0x80;
                        DDRC = 0x20;
```



```
                        PORTA = 0x00;
                        PORTC = 0x20;
                        asm("nop");
                        break;
                        case 2:         //8, 2
                        DDRA = 0x80;
                        DDRC = 0x10;
                        PORTA = 0x00;
                        PORTC = 0x10;
                        asm("nop");
                        break;
                        case 3:         //8, 3
                        DDRA = 0x80;
                        DDRC = 0x08;
                        PORTA = 0x00;
                        PORTC = 0x08;
                        asm("nop");
                        break;
                        case 4:         //8, 4
                        DDRA = 0x80;
                        DDRC = 0x04;
                        PORTA = 0x00;
                        PORTC = 0x04;
                        asm("nop");
                        break;
                        case 5:         //8, 5
                        DDRA = 0x80;
                        DDRC = 0x02;
                        PORTA = 0x00;
                        PORTC = 0x02;
                        asm("nop");
                        break;
                        case 6:         //8, 6
                        DDRA = 0x80;
                        DDRC = 0x01;
                        PORTA = 0x00;
                        PORTC = 0x01;
                        asm("nop");
                        break;
                }
        break;
        case 9:
        switch(y){
                        case 1:         //8, 1
                        DDRC = 0xA0;
                        PORTC = 0x20;
                        asm("nop");
                        break;
                        case 2:         //8, 2
                        DDRC = 0xB0;
                        PORTC = 0x10;
                        asm("nop");
                        break;
                        case 3:         //8, 3
                        DDRC = 0x88;
                        PORTC = 0x08;
                        asm("nop");
                        break;
                        case 4:         //8, 4
                        DDRC = 0x84;
                        PORTC = 0x04;
                        asm("nop");
                        break;
                        case 5:         //8, 5
                        DDRC = 0x82;
                        PORTC = 0x02;
                        asm("nop");
                        break;
                        case 6:         //8, 6
                        DDRC = 0x81;
                        PORTC = 0x01;
                        asm("nop");
                        break;
```

```
437                         }
438                         break;
439                         case 10:
440                         switch(y){
441                                 case 1:         //8, 1
442                                 DDRC = 0x60;
443                                 PORTC = 0x20;
444                                 asm("nop");
445                                 break;
446                                 case 2:         //8, 2
447                                 DDRC = 0x50;
448                                 PORTC = 0x10;
449                                 asm("nop");
450                                 break;
451                                 case 3:         //8, 3
452                                 DDRC = 0x48;
453                                 PORTC = 0x08;
454                                 asm("nop");
455                                 break;
456                                 case 4:         //8, 4
457                                 DDRC = 0x44;
458                                 PORTC = 0x04;
459                                 asm("nop");
460                                 break;
461                                 case 5:         //8, 5
462                                 DDRC = 0x42;
463                                 PORTC = 0x02;
464                                 asm("nop");
465                                 break;
466                                 case 6:         //8, 6
467                                 DDRC = 0x41;
468                                 PORTC = 0x01;
469                                 asm("nop");
470                                 break;
471                         }
472                         break;
473         }
474         _delay_ms(5);
475 }
476
477 void draw(coord arr[], unsigned char d){
478         //unsigned char p= (~PIND) & 0x0F;
479
480         //LEFT
481                 if(d==0 && arr[d].fall){
482 //
483 //                      if(arr[0].y==2 && left_press && !up_press && !down_press
     && !right_press )
484 //                              PORTB=0x08;
485 //                      else if ( (arr[0].y==2 && (!left_press || up_press || do
    wn_press || right_press  ) ) ||
486 //                                                      (arr[0].y!=2 && (left_press || u
    p_press || down_press || right_press  )) )
487 //                              PORTB=0x10;
488
489                         light(arr[0].x, arr[0].y);
490                         light(arr[0].x+1, arr[0].y+1);
491                         light(arr[0].x+1, arr[0].y-1);
492
493                 }
494         //UP
495                 else if (d== 1 && arr[d].fall){
496                         //if(arr[0].y==2 && p)
497                         //      PORTB=0x08;
498                         light(arr[1].x, arr[1].y);
499                         light(arr[1].x-1, arr[1].y-1);
500                         light(arr[1].x+1, arr[1].y-1);
501                 }
502         //DOWN
503                 else if (d== 2 && arr[d].fall){
504                         //if(arr[0].y==1 && p)
505                         //      PORTB=0x08;
```



```
506                         light(arr[2].x, arr[2].y);
507                         light(arr[2].x-1, arr[2].y+1);
508                         light(arr[2].x+1, arr[2].y+1);
509                 }
510         //RIGHT
511                 else if (d == 3 && arr[d].fall){
512                         //if(arr[0].y==2 && p)
513                         //      PORTB=0x08;
514                         light(arr[3].x, arr[3].y);
515                         light(arr[3].x-1, arr[3].y+1);
516                         light(arr[3].x-1, arr[3].y-1);
517                 }
518 //              else if(d == 5 && arr[0].fall && arr[1].fall){
519 //                      light(arr[0].x, arr[0].y);
520 //                      light(arr[0].x+1, arr[0].y+1);
521 //                      light(arr[0].x+1, arr[0].y-1);
522 //
523 //                      light(arr[1].x, arr[1].y);
524 //                      light(arr[1].x-1, arr[1].y-1);
525 //                      light(arr[1].x+1, arr[1].y-1);
526 //              }
527
528 }
529
530
531
532 #endif
```

```
1
2  // Permission to copy is granted provided that this header remains intact.
3  // This software is provided with no warranties.
4
5  ////////////////////////////////////////////////////////////////////////////////
6
7  #ifndef SCHEDULER_H
8  #define SCHEDULER_H
9
10 ////////////////////////////////////////////////////////////////////////////////
11 //Functionality - finds the greatest common divisor of two values
12 //Parameter: Two long int's to find their GCD
13 //Returns: GCD else 0
14 unsigned long int findGCD(unsigned long int a, unsigned long int b)
15 {
16         unsigned long int c;
17         while(1){
18                 c = a % b;
19                 if( c == 0 ) { return b; }
20                 a = b;
21                 b = c;
22         }
23         return 0;
24 }
25 ////////////////////////////////////////////////////////////////////////////////
26 //Struct for Tasks represent a running process in our simple real-time operating
    system
27 typedef struct _task{
28         // Tasks should have members that include: state, period,
29         //a measurement of elapsed time, and a function pointer.
30         signed   char state;            //Task's current state
31         unsigned long period;           //Task period
32         unsigned long elapsedTime;      //Time elapsed since last task tick
33         int (*TickFct)(int);            //Task tick function
34 } task;
35
36 #endif //SCHEDULER_H
```

```
1
2   // Permission to copy is granted provided that this header remains intact.
3   // This software is provided with no warranties.
4
5   ////////////////////////////////////////////////////////////////////////////
6
7   #ifndef TIMER_H
8   #define TIMER_H
9
10  #include <avr/interrupt.h>
11
12  volatile unsigned char TimerFlag = 0; // TimerISR() sets this to 1. C programmer
     should clear to 0.
13
14  // Internal variables for mapping AVR's ISR to our cleaner TimerISR model.
15  unsigned long _avr_timer_M = 1; // Start count from here, down to 0. Default 1ms
16  unsigned long _avr_timer_cntcurr = 0; // Current internal count of 1ms ticks
17
18  // Set TimerISR() to tick every M ms
19  void TimerSet(unsigned long M) {
20          _avr_timer_M = M;
21          _avr_timer_cntcurr = _avr_timer_M;
22  }
23
24  void TimerOn() {
25          // AVR timer/counter controller register TCCR0
26          TCCR0   = 0x0B; // bit3bit6=10: CTC mode (clear timer on compare)
27                                          // bit2bit1bit0=011: prescaler /64
28                                          // 00001011: 0x0B
29                                          // SO, 8 MHz clock or 8,000,000 /64 = 12
    5,000 ticks/s
30                                          // Thus, TCNT0 register will count at 12
    5,000 ticks/s
31
32          // AVR output compare register OCR0.
33          OCR0    = 125;  // Timer interrupt will be generated when TCNT0==OCR0
34                                          // We want a 1 ms tick. 0.001 s * 125,00
    0 ticks/s = 125
35                                          // So when TCNT0 register equals 125,
36                                          // 1 ms has passed. Thus, we compare to
    125.
37                                          // AVR timer interrupt mask register
38
39          TIMSK   = 0x02; // bit1: OCIE0 -- enables compare match interrupt
40
41          //Initialize avr counter
42          TCNT0 = 0;
43
44          // TimerISR will be called every _avr_timer_cntcurr milliseconds
45          _avr_timer_cntcurr = _avr_timer_M;
46
47          //Enable global interrupts
48          SREG |= 0x80;   // 0x80: 1000000
49  }
50
51  void TimerOff() {
52          TCCR0   = 0x00; // bit2bit1bit0=000: timer off
53  }
54
55  void TimerISR() {
56          TimerFlag = 1;
57  }
58
59  // In our approach, the C programmer does not touch this ISR, but rather TimerIS
    R()
60  ISR(TIMER0_COMP_vect)
61  {
62          // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn se
    ttings)
63          _avr_timer_cntcurr--;                   // Count down to 0 rather than u
    p to TOP
64          if (_avr_timer_cntcurr == 0) {  // results in a more efficient compare
65                  TimerISR();                             // Call the ISR that the
```



```
    user uses
66                  _avr_timer_cntcurr = _avr_timer_M;
67          }
68  }
69
70  #endif //TIMER_H
```

```
#include <avr/io.h>

//--------------------------------GLOBALS-------------------//

typedef struct _coord{
        unsigned char x;
        unsigned char y;
        unsigned char fall;
} coord;

// GLOBAL ARRAY OF ARROWS
// (x,y) and falling bit
coord arrows[4];

unsigned char c=0;
unsigned char indx=0;
unsigned char msp=0; //wrong press bit, 1 if wrong 0 if correct
unsigned char lives=3;
unsigned char stop=0;
unsigned char begin=0;
unsigned char win=2;
unsigned char wval=0;
unsigned char left_press=0;
unsigned char up_press=0;
unsigned char down_press=0;
unsigned char right_press=0;
unsigned char mksnd=0;
unsigned char bval=0;
unsigned char level_time=0;


//----------------------HEADER FILES----------------------//
 #include <timer.h>
#include <light.h>
#include <all_arrows.h>
// #include <check_left.h>
// #include <check_up.h>
// #include <check_down.h>
// #include <check_right.h>

// //-------------------------------------------------------//

int main(void)
{
        DDRD= 0x00; PORTD= 0xFF;
        DDRB= 0xFF; PORTB= 0x00;



        //---------DECLARE INITIAL STATES-------------//

        //Bstate= Binit; //button press SM
        Lstate= Linit; //LEFT arrow falling SM
        Ustate= Uinit; //UP SM
        Dstate= Dinit; //DOWN SM
        Rstate= Rinit; //UP SM
        /*B3State= off;*/
        //Ranstate= Raninit;


        //----------Button Check States--------------//
        LCstate= LCinit;
        UCstate= UCinit;
        DCstate= DCinit;
        RCstate= RCinit;

        //---------------------------------------------

        //---------------------------------------------//


        //---------LOAD INITIAL ARROW POINT LOCATIONS-------------//

```



```
        load_vals(arrows,0,1,5,0);
        load_vals(arrows,1,4,6,0);
        load_vals(arrows,2,7,5,0);
        load_vals(arrows,3,10,5,0);

        //-------------------------------------------------------//

        //---------DECLARE OTHER VARIABLES-------------//
        srand(1000);
        begin= GetBit(PIND,4); //gets a ready bit from microcont. 2
        indx=rand()%4;
        lives=5;
        stop=0;

        TimerSet(10);
        TimerOn();
        win=10;



        while(1)
        {


                while(GetBit((PIND),4) != 0)
                {
                        mksnd= ( GetBit((PIND),6) );

                        left_pr();
                        up_pr();
                        down_pr();
                        right_pr();

                if(!stop && GetBit(PIND,4) )
                {

                        arrows[indx].fall=1;

                        while(!TimerFlag)
                        {
                                draw(arrows,indx);
                        }
                        TimerFlag=0;

                        if(indx==0)
                        left();
                        else if(indx==1)
                        up();
                        else if(indx==2)
                        down();
                        else if(indx==3)
                        right();


                        if(indx==0 && Lbottom)
                        {
                                indx= rand()%4;
                                win--;
                        }
                        else if(indx==1 && Ubottom)
                        {
                                indx= 0;
                                win--;
                        }
                        else if(indx==2 && Dbottom)
                        {
                                indx= rand()%4;
                                win--;
                        }
                        else if(indx==3 && Rbottom)
                        {
                                indx= 1;
                                win--;
```

```
147                             }
148
149                             if(win==0)
150                             {
151                                     PORTB=0x18;
152                                     stop=1;
153
154                             }
155                             if(stop)
156                                     break;
157
158                     }//end of if
159
160             else{
161                             light(0,0);
162                             if(lives==0){
163                                     light(5,5);
164                                     light(7,5);
165                                     light(4,1);
166                                     light(5,2);
167                                     light(6,2);
168                                     light(7,2);
169                                     light(8,1);
170                             }
171                             else if(win==0 && lives!=0)
172                             {
173
174                                             light(5,5);
175                                             light(7,5);
176                                             light(3,4);
177                                             light(4,3);
178                                             light(4,4);
179                                             light(5,2);
180                                             light(6,2);
181                                             light(7,2);
182                                             light(8,3);
183                                             light(9,4);
184                             }
185             }
186             }// end of pind loop
187
188       }// end of while
189
190
191     } // end of main
```

```
#include <avr/io.h>
#include <bit.h>
#include <timer.h>

unsigned char level=1;
unsigned char mksnd=0;

#include <keypad.h>
#include <lcd_8bit_task.h>
#include <speaker.h>

//--------Find GCD function --------------------------------------------------
unsigned long int findGCD(unsigned long int a, unsigned long int b)
{
        unsigned long int c;
        while(1){
                c = a%b;
        if(c==0){return b;}
        a = b;
        b = c;
}
return 0;
}
//--------End find GCD function ----------------------------------------------

// Struct for Tasks represent a running process in our simple real-time operatin
g system.
typedef struct _task {
        /*Tasks should have members that include: state, period,
                a measurement of elapsed time, and a function pointer.*/
        signed char state; //Task's current state
        unsigned long int period; //Task period
        unsigned long int elapsedTime; //Time elapsed since last task tick
        int (*TickFct)(int); //Task tick function
} task;

enum key_sm {press};
 unsigned short x; //stores key press value
// unsigned short kp; //key press


// SynchSM for testing the LCD interface -- waits for button press, fills LCD wi
th repeated random num

enum LT_States { init, greet, w1, w2, w3, w4, select, w_mde, disp_mde, lives_dis
p,wait, };
unsigned char cnt=0;
unsigned char lives=5;
unsigned char Bval;


int LT_Tick(int state) {
        switch(state){
                case init:
                        state= greet;
                        LCD_go_g=0;
                        break;

                case greet:
                        if(!(cnt<50)){

                                state= w1;
                                cnt=0;
                        }
                        else
                                state= greet;
                        break;

                case w1:
                        if(!LCD_rdy_g)
                                state= w1;
                        else{
                                state= select;
```



```
//                              Bval= SetBit(Bval,0,1);
//                              PORTB=Bval;
                        }
                        break;
                case select:
                        state= w_mde;
                        break;

                case w_mde:
                        if(x== '\0')
                                state= w_mde;
                        else if (x!= '\0')
                                state= w2;
                        break;

                case w2:
                        if(!LCD_rdy_g)
                                state= w2;
                        else
                                state= disp_mde;
                        break;

                case disp_mde:
                        if(cnt < 50)
                                state= disp_mde;
                        else
                        {
                                state= w3;
                                cnt=0;
                        }
                        break;

                case w3:
                        if(!LCD_rdy_g)
                                state= w3;
                        else
                                state= lives_disp;
                        break;

                case lives_disp:
                        if(cnt<50)
                                state=lives_disp;
                        else if (!(cnt<50) )
                        {
                                state= w4;
                                cnt=0;
                        }
                        break;
                case w4:
                        if(!LCD_rdy_g)
                                state= w4;
                        else
                                state= wait;
                        break;

                case wait:
                        if(GetBit(PINA,1) && !GetBit(PINA,2)){
                                lives--;
                                state= lives_disp;
                        }
                        else
                                state= wait;
                        break;
        }   // transitions

        switch(state){
                case init:
                        Bval= SetBit(Bval,0,0);
                        PORTB=Bval;
                        lives=3;
                        break;
                case greet:
                        cnt++;
```

```
144                         strcpy(LCD_string_g,"Let's play DDR! ");
145                         LCD_go_g=1;
146                         break;
147                 case w1:
148                         LCD_go_g=0;
149                         break;
150                 case select:
151                         strcpy(LCD_string_g,"Select a mode. ");
152                         LCD_go_g=1;
153                         break;
154                 case w_mde:
155                         x= GetKeypadKey();
156                         break;
157                 case w2:
158                         LCD_go_g=0;
159                         break;
160                 case disp_mde:
161                         if(x == '3')
162                                 strcpy(LCD_string_g,"Easy Peasy!   ");
163                         else if(x== '1')
164                                 strcpy(LCD_string_g,"Challenge Mode. ");
165                         cnt++;
166                         LCD_go_g=1;
167                         break;
168                 case w3:
169                         LCD_go_g=0;
170                         break;
171                 case lives_disp:
172                         cnt++;
173                         if(lives==5)
174                         strcpy(LCD_string_g,"You have 5 Lives");
175                         LCD_go_g=1;
176                         break;
177                 case w4:
178                         LCD_go_g=0;
179                         //Bval= SetBit(Bval,0,1);
180                         if(x == '1')
181                                 /*Bval= SetBit(Bval,2,1);*/
182                                 PORTB=0x05;
183                         else
184                                 PORTB=0x01;
185
186                         //PORTB=Bval;
187                         /*mksnd=0;*/
188                         break;
189
190         }// state actions
191
192         return state;
193 }
194
195
196 int main(void)
197 {
198         DDRB=0xFF; PORTB=0x00;
199         DDRC = 0xF0; PORTC = 0x0F; // Keypad
200         DDRD= 0xFF; PORTD= 0x00; // LCD
201         Bval= PORTB;
202
203         static task task1,task2;
204         task *tasks[]= {&task1,&task2};
205         const unsigned short numtasks=sizeof(tasks)/sizeof(task*);
206
207         unsigned long int LT_calc = 10;
208         unsigned long int LI_calc = 10;
209
210
211         unsigned long int tmpGCD = 2;
212         tmpGCD = findGCD(LT_calc, LI_calc);
213 //      tmpGCD= findGCD(tmpGCD,P_calc);
214
215         unsigned long int GCD = tmpGCD;
216
```



```
217 //      unsigned long int LT_period = LT_calc/GCD;
218 //      unsigned long int LI_period = LI_calc/GCD;
219 //      unsigned long int P_period= P_calc/GCD;
220 //
221         task1.state= init;
222         task1.elapsedTime= LT_calc;
223         task1.period= LT_calc;
224         task1.TickFct= &LT_Tick;
225
226         task2.state= -1;
227         task2.elapsedTime= LI_calc;
228         task2.period= LI_calc;
229         task2.TickFct= &LCDI_SMTick;
230
231 //      task3.state= off;
232 //      task3.elapsedTime= P_calc;
233 //      task3.period= P_calc;
234 //      task3.TickFct= &psound;
235
236         TimerSet(GCD);
237         TimerOn();
238
239         unsigned char i=0;
240
241         while(1)
242         {
243                 for ( i = 0; i < numtasks; i++ ) {
244                         // Task is ready to ticks
245                         if ( tasks[i]->elapsedTime == tasks[i]->period ) {
246                                 // Setting next state for task
247                                 tasks[i]->state = tasks[i]->TickFct(tasks[i]->s
    tate);
248                                 // Reset the elapsed time for next tick.
249                                 tasks[i]->elapsedTime = 0;
250                         }
251                         tasks[i]->elapsedTime += 1;
252                 }
253                 while(!TimerFlag);
254                 TimerFlag = 0;
255         }
256 }
```