Razik Ahmed and Stephen Hall
Harshad Desai
Lab 8

A) Objectives

1. Overview
   1.1. Objective: to layout an embedded system
   1.2. Roles and Responsibilities: Razik will do the software and Stephen will do the hardware. The clients are people that are bored of board games.
2. Function Description
   2.1. Functionality: to play Battleship
   2.2. Performance: current will be determined using the bench power supply which will help decide the battery
   2.3. Usability: Their will be two different modules each with a C32 microcontroller, 128 x 64 pixel graphical LCD screen, six buttons for user input, and 6 LEDs to signify a hit. The two modules will be connected via a serial cable.
3. Deliverables
   3.1. Reports: The reports for Labs 8 and 11 will be written
   3.2. Outcomes: Objectives, hardware and software design, and measurement data for Lab 8 and Lab 11

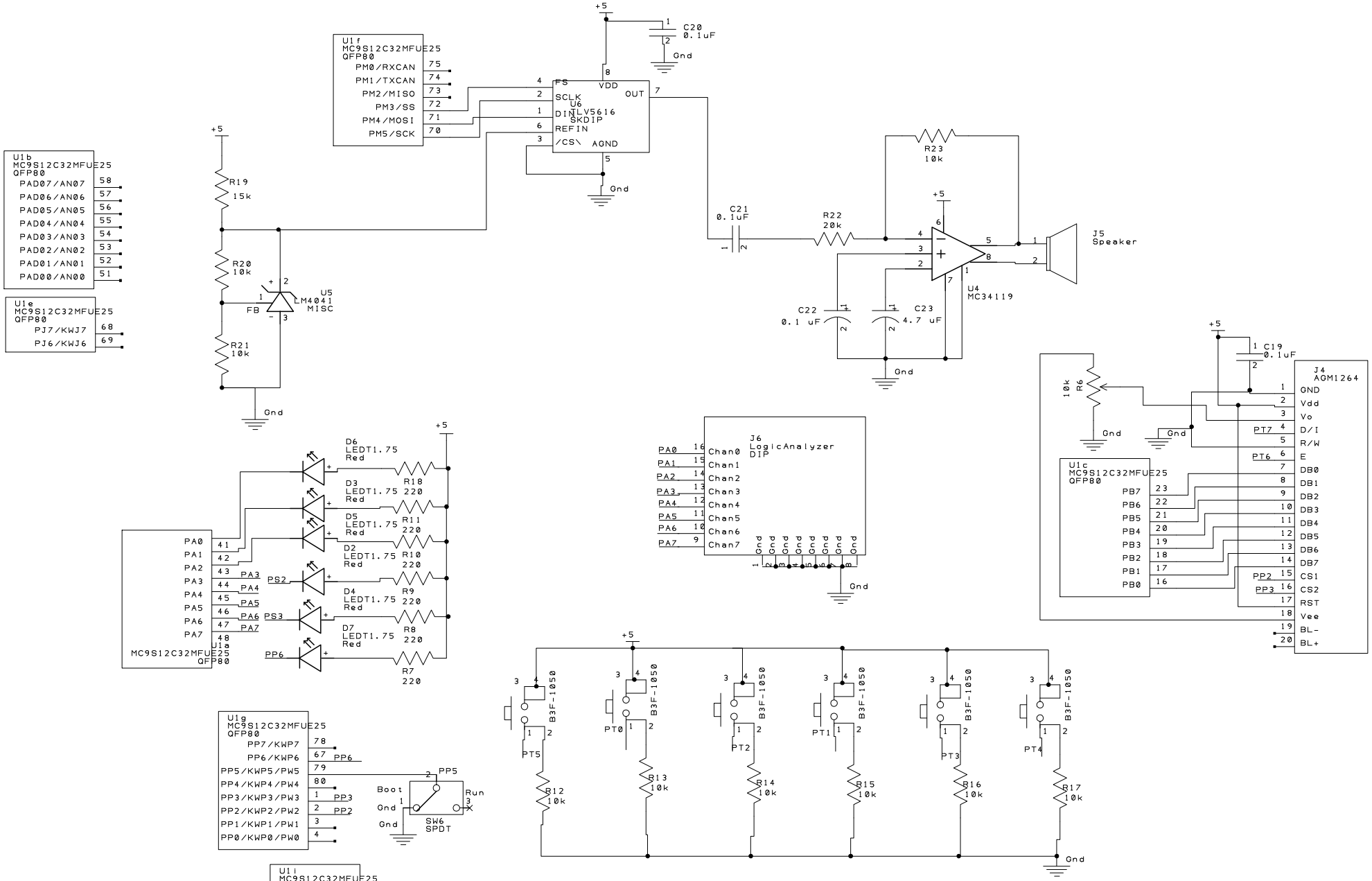
B) Hardware Design – Pages 2 - 7

C) Software Design – Requirements document already in Objectives

D) Measurement Data

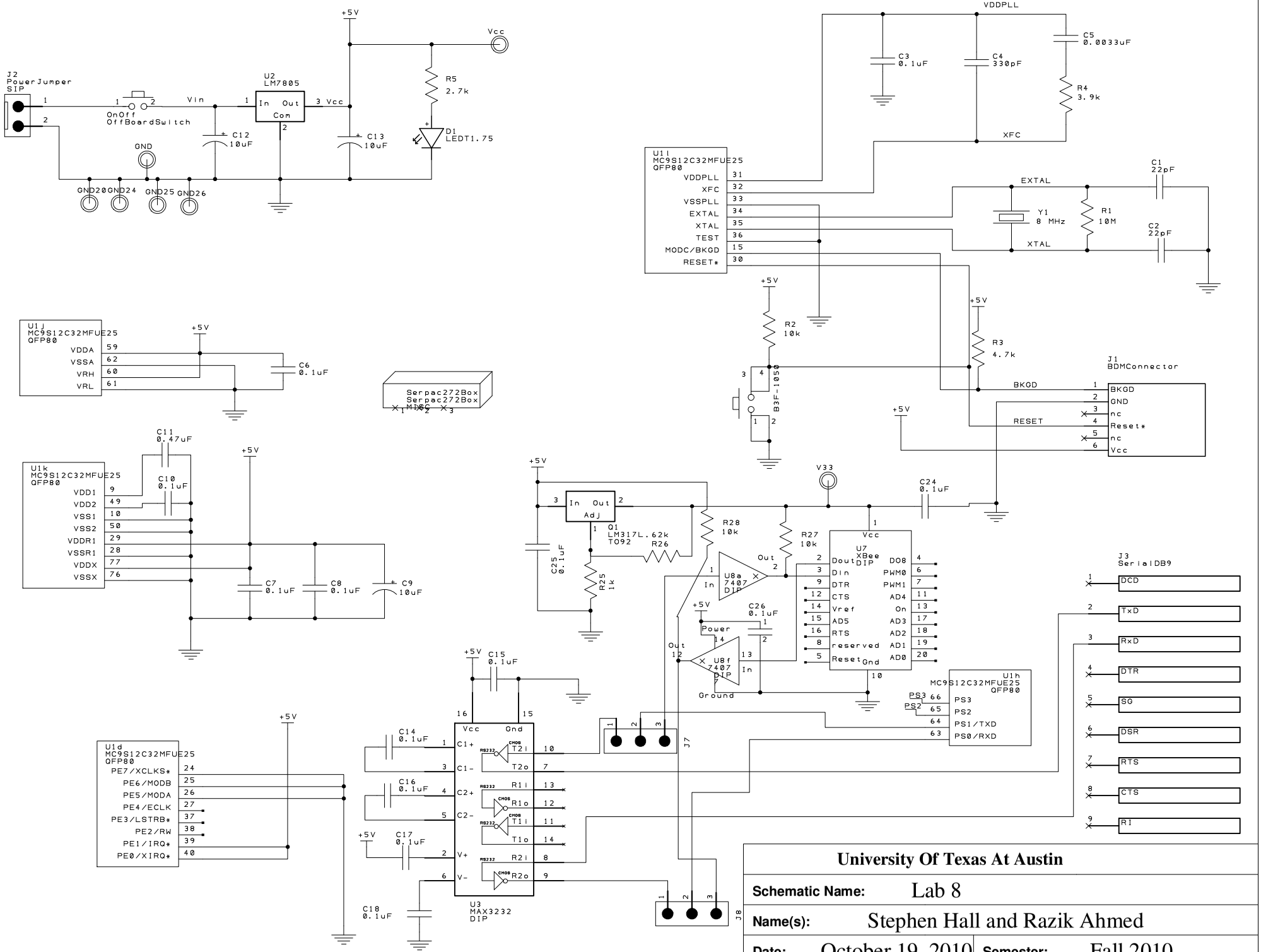
a. ~50mA measured current, ZigBee and sound shouldn't increase current much since they will be rare compared to the rest of the program. The current will actually be less once sleeping has been implemented
b. Estimated cost: $125.84, Bill of Materials: page 8

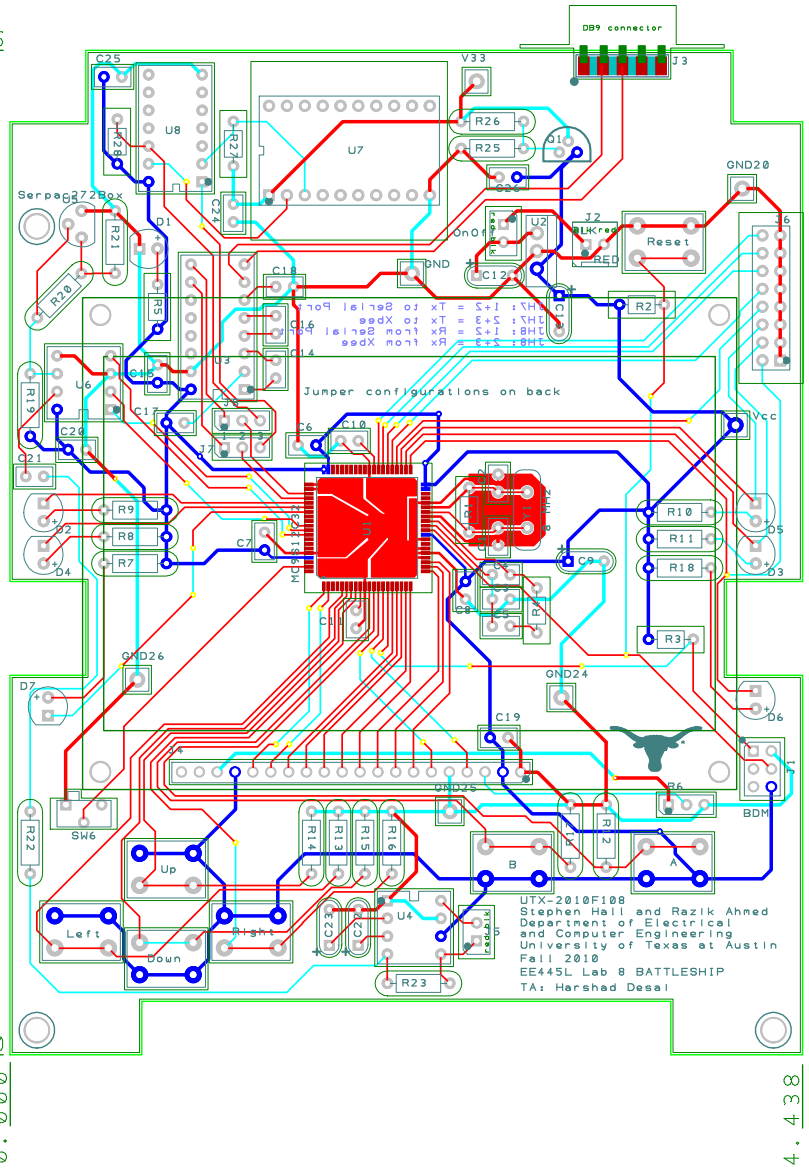
E) Analysis and Discussion – None



University Of Texas At Austin	
Schematic Name:	Lab 8 Battleship
Name(s):	Stephen Hall and Razik Ahmed
Date:	October 19, 2010 **Semester:** Fall 2010

A1


J2
PowerJumper
SIP
OnOff
OffBoardSwitch
Vin
U2
LM7805
In Out
Com
3 Vcc
+5V
Vcc
R5
2.7k
D1
LEDT1.75
C12
10uF
C13
10uF
GND
GND20
GND24
GND25
GND26
U1j
MC9S12C32MFUE25
QFP80
VDDA 59
VSSA 62
VRH 60
VRL 61
C6
0.1uF
Serpac272Box
Serpac272Box
U1k
MC9S12C32MFUE25
QFP80
VDD1 9
VDD2 49
VSS1 10
VSS2 50
VDDR1 29
VSSR1 28
VDDX 77
VSSX 76
C11
0.47uF
C10
0.1uF
C7
0.1uF
C8
0.1uF
C9
10uF
U1d
MC9S12C32MFUE25
QFP80
PE7/XCLKS* 24
PE6/MODB 25
PE5/MODA 26
PE4/ECLK 27
PE3/LSTRB* 37
PE2/RW 38
PE1/IRQ* 39
PE0/XIRQ* 40
C14
0.1uF
C16
0.1uF
C17
0.1uF
C18
0.1uF
C15
0.1uF
Vcc Gnd
C1+
C1-
C2+
C2-
V+
V-
T2i
T2o
R1i
R1o
T1i
T1o
R2i
R2o
U3
MAX3232
DIP
J7
J8
VDDPLL
C3
0.1uF
C4
330pF
C5
0.0033uF
R4
3.9k
XFC
U1i
MC9S12C32MFUE25
QFP80
VDDPLL 31
XFC 32
VSSPLL 33
EXTAL 34
XTAL 35
TEST 36
MODC/BKGD 15
RESET* 30
EXTAL
XTAL
Y1
8 MHz
R1
10M
C1
22pF
C2
22pF
R2
10k
R3
4.7k
B3F-1050
BKGD
RESET
J1
BDMConnector
BKGD
GND
nc
Reset*
nc
Vcc
V33
C24
0.1uF
Q1
LM317L.62k
TO92
R26
R28
10k
R27
10k
R25
1k
C25
0.1uF
U8a
7407
DIP
U8f
7407
DIP
C26
0.1uF
Power
Ground
U7
XBee
DIP
Dout
Din
DTR
CTS
Vref
AD5
RTS
reserved
Reset
DO8
PWM0
PWM1
AD4
On
AD3
AD2
AD1
AD0
Gnd
U1h
MC9S12C32MFUE25
QFP80
PS3 66
PS2 65
PS1/TXD 64
PS0/RXD 63
J3
SerialDB9
DCD
TxD
RxD
DTR
SG
DSR
RTS
CTS
RI
University Of Texas At Austin
Schematic Name: Lab 8
Name(s): Stephen Hall and Razik Ahmed
Date: October 19, 2010
Semester: Fall 2010
A2



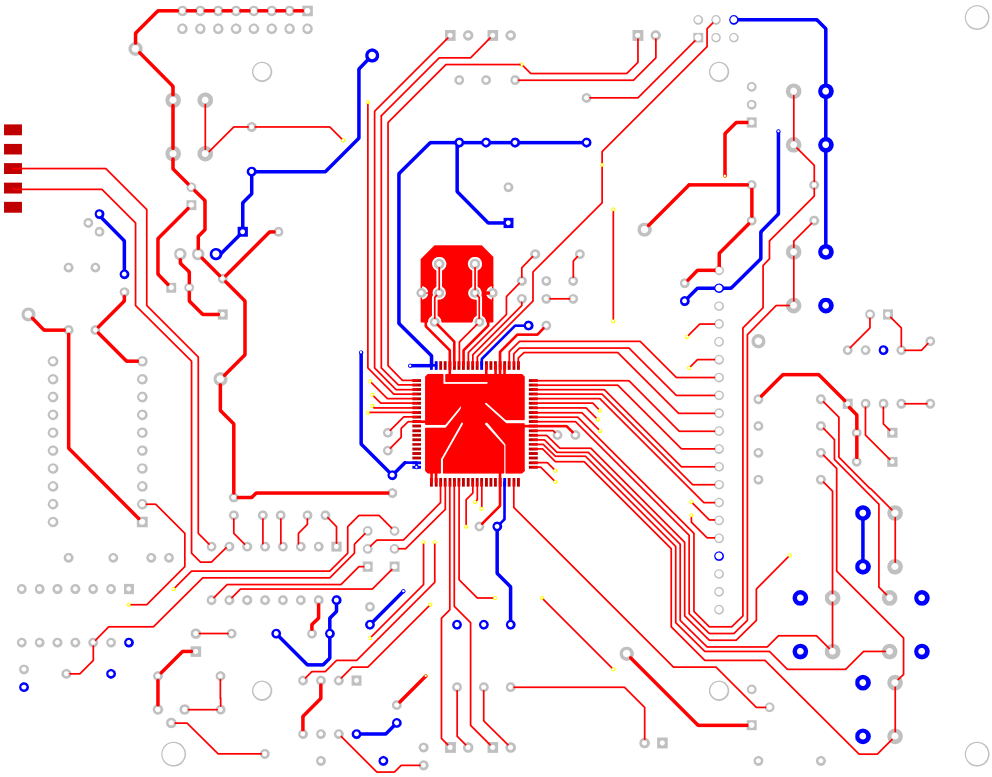
Fabrication Notes

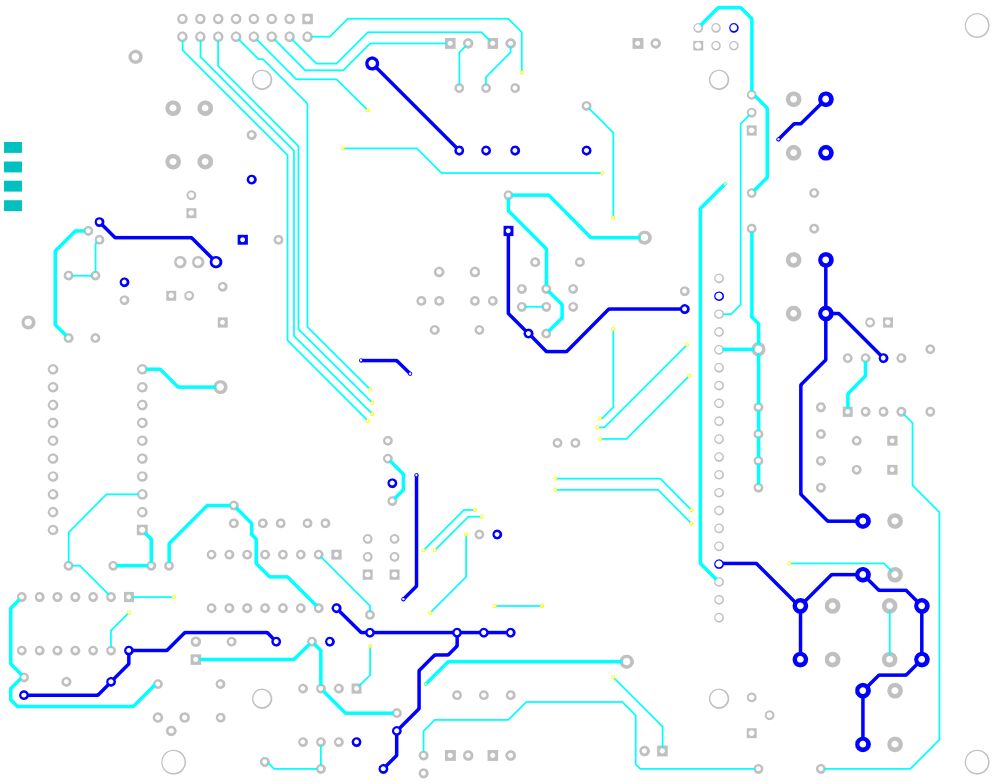
1. Fabricate PCB in accordance with IPC-6012, Class 2: per IPC-6011.
2. Layer Stackup
    1. Top Silkscreen
    2. Top Soldermask
    3. Top Copper, plate to 1oz/ft^2
    4. FR-4 or equivalent
    5. Bottom Copper, plate to 1oz/ft^2
    6. Bottom Soldermask
    7. Bottom Silkscreen
3. Finish Thickness should be 0.062"
4. No internal routing
5. All holes shall be plated

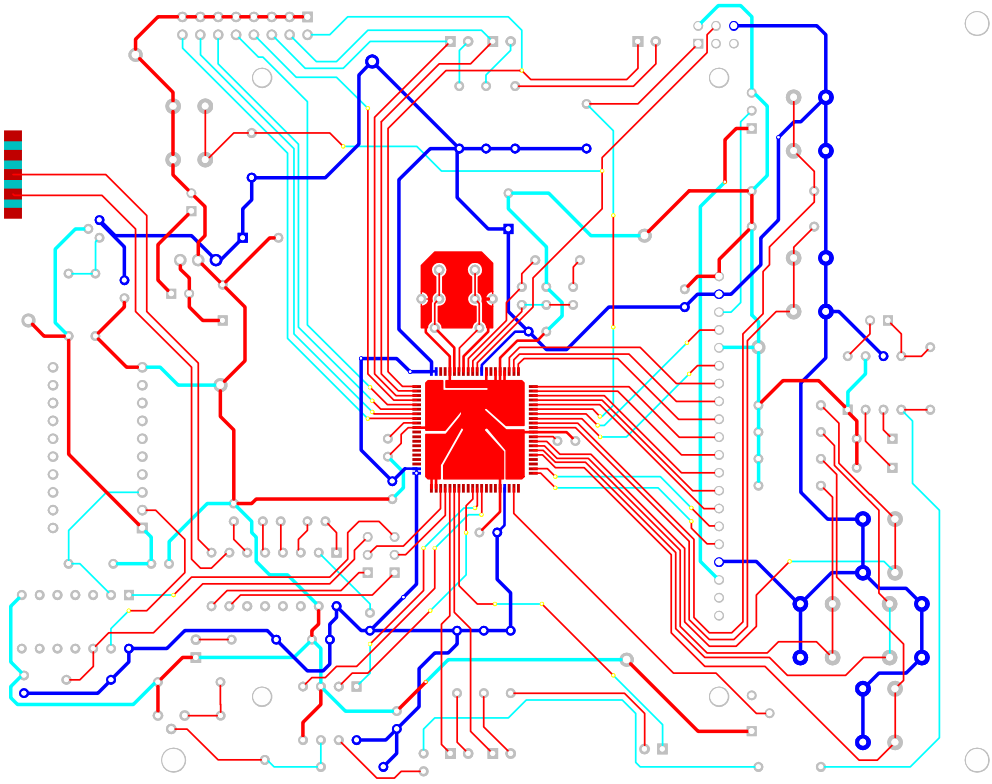
Battleship		
PART NO.: UTX-2010F108	REV: A	DATE: 11-04-2010
	GERBER: UTX-2010F108	
DESIGNER: Stephen Hall, Razik Ahmed	Checked by TA: Harshad	

REV [redacted] 2010

Battleship
P/N: UTX-2010F108 REV A
DESIGNED AT THE UNIVERSITY OF TEXAS AT AUSTIN







Qty Note	REF DES Note		Bill of Materials	EE345L Fall 2010	October 15, 2010		EE345L pays for the PCB			
Quantity	REF DES	Type	Description	Manufacturer	Mfg. P/N	Distributor	P/N	Unit cost	Cost	PCB Artist
1		ASM	Black case, 8" x 8" x 4"			BGMicro	CAS1007	$1.00	$1.00	CAS1007
11		CAP	Ceramic, X7R, 20%, 0.1uF			Jameco	544921	$0.14	$1.54	Ceramic0.2
1		CAP	Ceramic, Z5U, -20/+80%, 0.47 uF	Kemet	C320C474M5U5TA	Digikey	399-4309-ND	$0.40	$0.40	Ceramic
1		CAP	Ceramic, Z5U, -20/+80%, 0.0033 uF	Kemet	C320C332M5U5TA			$0.40	$0.40	Ceramic
2		CAP	Ceramic Z5U, -20/+80%, 22 pF			Jameco	15405	$0.03	$0.06	Ceramic
1		CAP	Ceramic Z5U, -20/+80%, 330 pF			Jameco	15410	$0.03	$0.03	Ceramic
3		CAP-Elect	Electrolytic 10uV 16V, 20%	Panasonic - ECG	ECE-A1CKA100	Digikey	P807-ND	$0.22	$0.66	Electrolytic
1		CAP-Elect	Electrolytic 4.7uV 25V, 20%	Panasonic - ECG	ECE-A1EKA4R7	Digikey	P812-ND	$0.14	$0.14	Electrolytic
1		CAP-Elect	Electrolytic 0.1uV 25V, 20%	Panasonic - ECG	ECE-A1EKA4R7	Digikey	P752-ND	$0.14	$0.14	Electrolytic
1		CON	2-pin for TechArts power plug			AllElectronics	CON-242	$0.70	$0.70	PowerJumper
1		CON	2-pin header			AllElectronics	SBH-2	$0.10	$0.10	Jumper2
1		CON	2-pin jumper			AllElectronics	SBC-2	$0.27	$0.27	goes with SBH
1		CON	Test point, black	Keystone Electronics	5001	Digikey	5001K-ND	$0.29	$0.29	testpoint
1		CON	Test point, red	Keystone Electronics	5000	Digikey	5000K-ND	$0.29	$0.29	testpoint
1		CON	DB9 serial connector, female, board mount			Jameco	15771	$0.65	$0.65	SerialDB9
1		CPU	MC9S12C32MFUE25, 80-pin QFP	Freescale	MC9S12C128MFUE			$5.00	$5.00	MC9S12C128
1		CRYS	8 MHz crystal,50ppm,HC49/U			Jameco	14728	$0.59	$0.59	XTAL
1		IC	78M05 500MA 5V TO-220	National	LM78M05	Jameco	192233	$0.19	$0.19	LM7805CT
1		IC	RS232 driver	STMicroelectronics	ST232CN	Jameco	2001171	$0.58	$0.58	MAX3232
1		IC	3-Terminal 1A Positive Voltage Regulator	Fairchild	LM7805	Digikey	rLM7805CT-ND	$0.65	$0.65	LM7805CT
1		JACK	DC Power MALE 2.1mm			Jameco	101179	$0.55	$0.55	PowerJack
1		LED	Green 2mA 5mm diffused	Avago Technologies	HLMP-4740	Digikey	516-1327-ND	$0.29	$0.29	LEDT1.75
1		PCB	PCB plus shipping	Advanced Circuits		Advanced Circuits		$49.06	$49.06	
1		RES	Carbon 1/6W, 5%, 2.7K	Yageo	CFR-12JB-2K7	Digikey	2.7KEBK-ND	$0.02	$0.02	0.125Wresisto
1		RES	Carbon 1/6W, 5%, 3.9K	Yageo	CFR-12JB-3K9	Digikey	3.9KEBK-ND	$0.02	$0.02	0.125Wresisto
1		RES	Carbon 1/6W, 5%, 4.7K	Yageo	CFR-12JB-4K7	Digikey	4.7KEBK-ND	$0.02	$0.02	0.125Wresisto
1		RES	Carbon 1/6W, 5%, 10K	Yageo	CFR-12JB-10K	Digikey	10KEBK-ND	$0.02	$0.02	0.125Wresisto
1		RES	Carbon 1/6W, 5%, 10M	Yageo	CFR-12JB-10M	Digikey	10MEBK-ND	$0.02	$0.02	0.125Wresisto
6		SW	B3F tactile push button switch	Omron Electronics	B3F-1052	Digikey	SW405-ND	$0.17	$1.02	B3F-1050
1		SW	On/off power switch			BGMicro	SWT1010	$0.85	$0.85	BOXMOUNT
4		BAT	AA, 1.25V NMH, 2500mAh			BGMicro	BAT1102	$2.49	$9.96	
6		RES	Carbon 1/4W, 5%, 10K	Yageo	CFR-12JB-10K	Digikey	10KEBK-ND	$0.02	$0.14	0.25Wresistor
1		LCD	128x64 Graphic LCD	AZ Displays	AGM1264F	BGMicro	LCD1030	$9.95	$9.95	LCD1030
1		XBEE	XBee 1mW Zigbee Module	Digi			WRL-08664	$22.95	$22.95	
1		XBEE	XBee Module Breakout Board	Digi			BOB-08276	$2.95	$2.95	
2		XBEE	Two 2mm 10 pin XBee Sockets	Digi			PRT-08272	$1.00	$2.00	
1		XBEE	0.1" 40 pin Break Away Headers	Digi			PRT-00116	$2.50	$2.50	
1		IC	MC34119, AUDIO LOW PWR 8-PDIP	Freescale	MC34119	Jameco	316865	$2.25	$2.25	MC34119
1		IC	TLV5616 12-bit DAC	TI	TLV5616	TI	TLV5616CP	$6.66	$6.66	TLV5616
1		IC	LM4041CILPR shunt diode reference	TI	LM4041CILPR	TI	LM4041CILPR	$0.90	$0.90	LM4041
1		BOX	Serpac 272 Electronic Enclosure	Serpac	272	Serpac	272	$0.00	$0.00	Serpac272Box
									$125.84	

```
#include <hidef.h>      /* common defines and macros */
#include <mc9s12dp512.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp512"

#include "PLL.h"
#include "adc.h"
#include "LCDG.h"
#include "Timer.h"
#include "Game.h"
#include "switch.h"

unsigned short ADCsample;  // ADC sample, 0 to 1023
unsigned short Voltage;   // 0.01 volts, 0 to 500
unsigned short ADCcount;  // 0 to 2999
unsigned short StartTime; // in seconds
void main(void) {
  PLL_Init();
  LCD_Init();   // TCNT at 1.5 MHz
  ADC_Init();   // Activate ADC
  DDRP |= 0xA0; // heartbeats, PP7 every 3000, PP5 at sampling rate
  PTP |= 0x80;
  EnableInterrupts;
  Key_Init();
  Game_Init();


  //enableOC6(&whee, 60000, 25, 5);

  for(;;) {

  }
}
```

```
#define EMPTY         0
#define SHIPEND_UP    1
#define SHIPEND_DOWN  2
#define SHIPEND_LEFT  3
#define SHIPEND_RIGHT 4
#define SHIP_VERT     5
#define SHIP_HORIZ    6
#define HIT           7
#define MISS          8

#define WELCOME       0
#define PLACING_SHIPS 1

#define UP    0
#define DOWN  1
#define LEFT  2
#define RIGHT 3

void Game_Init(void);
void Game_Update(void);


void Game_DPad(unsigned char direction);
void Game_A(void);
void Game_B(void);
```

```
#include <mc9s12dp512.h>     /* derivative information */
#include "game.h"
#include "LCDG.h"
#include "switch.h"

#define DEBOUNCE_DELAY 30000

#define SINGLE_PLAYER 0
#define MULTI_PLAYER 1

#define VERTICAL 0
#define HORIZONTAL 1

typedef struct {
  unsigned int x:4;
  unsigned int y:4;
  unsigned int orientation:1;
  unsigned int size:3;
} ShipType;

typedef struct {
  unsigned int x:4;
  unsigned int y:4;
  unsigned int type:1;
} AttackType;

struct {
  unsigned int x:4;
  unsigned int y:4;
} cursor;

static int state;

static int buttonFlag;

static ShipType ships[5] = {
  {0, 0, VERTICAL, 2},
  {0, 0, VERTICAL, 3},
  {0, 0, VERTICAL, 3},
  {0, 0, VERTICAL, 4},
  {0, 0, VERTICAL, 5}
};

static int numShips;

static AttackType enemyAttacks[100];
static int numEnemyAttacks;

static AttackType playerAttacks[100];
static int numPlayerAttacks;

void incState(void) {
  switch(state) {
    case WELCOME:
      numShips = 1;
      state = PLACING_SHIPS;
      break;
  }
  Game_Update();
}

void Game_Init(void) {
  state = WELCOME;
  numShips = 0;
  numEnemyAttacks = 0;
  numPlayerAttacks = 0;
  cursor.x = 0;
  cursor.y =0;
  Game_Update();
}

void Game_Update(void) {
  int i, j;

  if(state == WELCOME) {

    LCD_Clear(0);
    LCD_GoTo(4, 1);
```

```
    LCD_OutString("Welcome to Battleship");

    enableOC6(&incState, 62500, 9, 1);
  }
  else if (state == PLACING_SHIPS) {
    static unsigned  char field[10][10];
    LCD_Clear(0);

    for(i=0; i<10; i++) {
      for(j=0; j<10; j++) {
        field[i][j] = EMPTY;
      }
    }

    for(i=0; i<numShips; i++) {
      ShipType ship = ships[i];
      if(ship.orientation == HORIZONTAL) {
        field[ship.x][ship.y] = SHIPEND_LEFT;
        for(j=1; j<ship.size-1; j++) {
          field[ship.x][ship.y+j] = SHIP_HORIZ;
        }
        field[ship.x][ship.y+ship.size-1] = SHIPEND_RIGHT;
      }
      else {
        field[ship.x][ship.y] = SHIPEND_UP;
        for(j=1; j<ship.size-1; j++) {
          field[ship.x+j][ship.y] = SHIP_VERT;
        }
        field[ship.x+ship.size-1][ship.y] = SHIPEND_DOWN;
      }
    }

    for(i=0; i<numEnemyAttacks; i++) {
      AttackType attack = enemyAttacks[i];
      field[attack.x][attack.y] = attack.type;
    }

    LCD_DrawGrid(field);
  }

  /*

    else {
      for(i=0; i<numPlayerAttacks; i++) {
        AttackType attack = playerAttacks[i];
        field[attack.x][attack.y] = attack.type;
      }
    }
  */
}

int shipInBounds(int index) {
  ShipType ship = ships[index];

  if(ship.x < 0 || ship.x > 9 || ship.y < 0 || ship.y > 9 ||
    (ship.orientation == VERTICAL && ship.x + ship.size > 10) ||
    (ship.orientation == HORIZONTAL && ship.y + ship.size > 10)) {
      return 0;
  }

  return 1;
}

int validShipPos(int index) {
  ShipType ship = ships[index];
  int i;

  for(i=0; i<numShips; i++) {
    if(i != index) {
      if(ship.orientation == HORIZONTAL) {
        if(ships[i].orientation == HORIZONTAL) {
          if(ship.x == ships[i].x) {
            if(ship.y + ship.size > ships[i].y ||
               ship.y < ships[i].y + ships[i].size) {
              return 0;
            }
          }
        }
```

```
        else {
          if(ship.x >= ships[i].x &&
             ship.x < ships[i].x + ships[i].size &&
             ships[i].y >= ship.y &&
             ships[i].y < ship.y + ship.size) {
            return 0;
          }
        }
      }
      else {
        if(ships[i].orientation == HORIZONTAL) {
          if(ship.y >= ships[i].y &&
             ship.y < ships[i].y + ships[i].size &&
             ships[i].x >= ship.x &&
             ships[i].x < ship.x + ship.size) {
            return 0;
          }
        }
        else {
          if(ship.y == ships[i].y) {
            if(ship.x + ship.size > ships[i].x ||
               ship.x < ships[i].x + ships[i].size) {
              return 0;
            }
          }
        }
      }
    }
  }

  return 1;
}

void flag(void) {
  buttonFlag = 0;
}

void Game_DPad(unsigned char direction) {
  unsigned int tempX, tempY;
  if(!buttonFlag) {
    switch(state) {
      case PLACING_SHIPS:
        tempX = ships[numShips-1].x;
        tempY = ships[numShips-1].y;

        do {
          switch(direction) {
            case UP:
              ships[numShips-1].x--;
              break;
            case DOWN:
              ships[numShips-1].x++;
              break;
            case LEFT:
              ships[numShips-1].y--;
              break;
            case RIGHT:
              ships[numShips-1].y++;
              break;
          }
        }while(!validShipPos(numShips-1) && shipInBounds(numShips-1));

        if(validShipPos(numShips-1) && shipInBounds(numShips-1)) {
          Game_Update();
        }
        else {
          ships[numShips-1].x = tempX;
          ships[numShips-1].y = tempY;
        }

        buttonFlag = 1;
        enableOC6(&flag, DEBOUNCE_DELAY, 8, 1);
        break;
    }
  }
}

void Game_A(void) {
```

```
  if(!buttonFlag) {
    switch(state) {
      case PLACING_SHIPS:
      numShips++;
      Game_Update();
      buttonFlag = 1;
      enableOC6(&flag, DEBOUNCE_DELAY, 8, 1);
      break;
    }
  }
}

void Game_B(void) {
  if(!buttonFlag) {
    switch(state) {
      case PLACING_SHIPS:
        ships[numShips-1].orientation ^= 1;
        if(validShipPos(numShips-1) && shipInBounds(numShips-1)) {
          Game_Update();
        }
        else {
          ships[numShips-1].orientation ^= 1;
        }
        buttonFlag = 1;
        enableOC6(&flag, DEBOUNCE_DELAY, 8, 1);
        break;
    }
  }
}
```

```
//*****************LCDG.h************************************
// implementation of the driver for the AGM1264F MODULE
// Jonathan W. Valvano 11/21/09

//  This example accompanies the books
//   "Embedded Microcomputer Systems: Real Time Interfacing",
//         Engineering, copyright (c) 2006,
//   "Introduction to Embedded Microcomputer Systems:
//    Motorola 6811 and 6812 Simulation", Thomson, copyright (c) 2002

// Copyright 2009 by Jonathan W. Valvano, valvano@mail.utexas.edu
//    You may use, edit, run or distribute this file
//    as long as the above copyright notice remains

// Hardware:
// gnd    =  1- AGM1264F ground
// +5V    =  2- AGM1264F Vcc (with 0.1uF cap to ground)
// pot    =  3- AGM1264F Vo  (center pin of 10k pot)
// PP2    =  4- AGM1264F D/I (0 for command, 1 for data)
// gnd    =  5- AGM1264F R/W (blind cycle synchronization)
// PP3    =  6- AGM1264F E   (1 to latch in data/command)
// PH0    =  7- AGM1264F DB0
// PH1    =  8- AGM1264F DB1
// PH2    =  9- AGM1264F DB2
// PH3    = 10- AGM1264F DB3
// PH4    = 11- AGM1264F DB4
// PH5    = 12- AGM1264F DB5
// PH6    = 13- AGM1264F DB6
// PH7    = 14- AGM1264F DB7
// PP0    = 15- AGM1264F CS1 (controls left half of LCD)
// PP1    = 16- AGM1264F CS2 (controls right half of LCD)
// +5V    = 17- AGM1264F RES (reset)
// pot    = 18- ADM1264F Vee (-10V)
// 10k pot from pin 18 to ground, with center to pin 3
// references   http://www.azdisplays.com/prod/g1264f.php
// sample code  http://www.azdisplays.com/PDF/agm1264f_code.pdf
// data sheet   http://www.azdisplays.com/PDF/agm1264f.pdf

// BUG NOTICE 11/11/09 -Valvano
// When changing from right to left or from left to right
//   the first write with data=0 goes to two places
// One can reduce the effect of this bug by
// 1) Changing sides less often
// 2) Ignore autoincrement, and set column and page address each time
// 3) Blanking the screen then write 1's to the screen

//*****************************************************************

// to use it as an 8-line by 21-character display
// initialize it once using
//      LCD_Init
// clear screen with
//      LCD_Clear
// set cursor position using
//      LCD_GoTo
// place ASCII on the screen using
//      LCD_OutChar
//      LCD_OutString
//      LCD_OutDec
//      LCD_OutFix1
//      LCD_OutFix2
//      LCD_OutFix3

// ********* LCD_Init***********
// Initialize AGM1264F 128-bit by 64-bit graphics display
// activates TCNT at 1.5 MHz, assumes PLL active
// Input: none
// Output: none
// does not clear the display
void LCD_Init(void);


// ********* LCD_Clear***********
// Clear the entire 1024 byte (8192 bit) image on the
//    AGM1264F 128-bit by 64-bit graphics display
// Input: value to write into all bytes of display RAM
// Output: none
// e.g., LCD_Clear(0);  // makes all pixels off
```

```
void LCD_Clear(unsigned char data);

//------------------------LCD_GoTo------------------------
// Move cursor
// Input: line number is 1 to 8, column from 1 to 21
// Output: none
// errors: it will ignore legal addresses
void LCD_GoTo(int line, int column);


// ********* LCD_OutChar***********
// Output ASCII character on the
//    AGM1264F 128-bit by 64-bit graphics display
// Input: 7-bit ASCII to display
// Output: none
// letter must be between 32 and 127 inclusive
// execute LCD_GoTo to specify cursor location
void LCD_OutChar(unsigned char letter);


//---------------------LCD_OutString--------------
// Display String
// Input: pointer to NULL-terminationed ASCII string
// Output: none
void LCD_OutString(char *pt);

void LCD_DrawGrid(unsigned char field[10][10]);
```

```
//*****************LCDG.c************************************
// implementation of the driver for the AGM1264F MODULE
// Jonathan W. Valvano 11/20/09

//  This example accompanies the books
//   "Embedded Microcomputer Systems: Real Time Interfacing",
//         Engineering, copyright (c) 2006,
//   "Introduction to Embedded Microcomputer Systems:
//    Motorola 6811 and 6812 Simulation", Thomson, copyright (c) 2002

// Copyright 2009 by Jonathan W. Valvano, valvano@mail.utexas.edu
//    You may use, edit, run or distribute this file
//    as long as the above copyright notice remains

// Hardware:
// gnd    =  1- AGM1264F ground
// +5V    =  2- AGM1264F Vcc (with 0.1uF cap to ground)
// pot    =  3- AGM1264F Vo  (center pin of 10k pot)
// PP2    =  4- AGM1264F D/I (0 for command, 1 for data)
// gnd    =  5- AGM1264F R/W (blind cycle synchronization)
// PP3    =  6- AGM1264F E   (1 to latch in data/command)
// PH0    =  7- AGM1264F DB0
// PH1    =  8- AGM1264F DB1
// PH2    =  9- AGM1264F DB2
// PH3    = 10- AGM1264F DB3
// PH4    = 11- AGM1264F DB4
// PH5    = 12- AGM1264F DB5
// PH6    = 13- AGM1264F DB6
// PH7    = 14- AGM1264F DB7
// PP0    = 15- AGM1264F CS1 (controls left half of LCD)
// PP1    = 16- AGM1264F CS2 (controls right half of LCD)
// +5V    = 17- AGM1264F RES (reset)
// pot    = 18- ADM1264F Vee (-10V)
// 10k pot from pin 18 to ground, with center to pin 3
// references   http://www.azdisplays.com/prod/g1264f.php
// sample code  http://www.azdisplays.com/PDF/agm1264f_code.pdf
// data sheet   http://www.azdisplays.com/PDF/agm1264f.pdf

// BUG NOTICE 11/11/09 -Valvano
// When changing from right to left or from left to right
//   the first write with data=0 goes to two places
// One can reduce the effect of this bug by
// 1) Changing sides less often
// 2) Ignore autoincrement, and set column and page address each time
// 3) Blanking the screen then write 1's to the screen
// GoTo bug fixed on 11/20/09

//*********************************************************************
#include <mc9s12dp512.h>     /* derivative information */
#include "LCDG.h"
#include "Timer.h"
#include "game.h"

#define E PTP_PTP3
#define DI PTP_PTP2
#define CS2 PTP_PTP1
#define CS1 PTP_PTP0
#define DATA PTH

// assuming TCNT is 1.5 MHz
#define T1usec 2
#define T4usec 6

static unsigned short OpenFlag=0;// 5 wide by 7 tall font

unsigned char Column1; // column position
unsigned char bLeft1;  // to be placed into CS1, in LCD_OutChar
unsigned char bRight1; // to be placed into CS2, in LCD_OutChar
unsigned char Page;
unsigned char bDown;    // true if want font shifted down

const unsigned char Font[96*5]={   // no numbers with bit7=1
  0,0,0,0,0,           // 32  space
  0,0,95,0,0,          // 33  !
  0,7,0,7,0,           // 34  "
  20,127,20,127,20,    // 35  #
  36,42,127,42,18,     // 36  $
  35,19,8,100,98,      // 37  %
```

```
  54,73,85,34,80,      // 38  &
  0,5,3,0,0,           // 39  quote
  0,28,34,65,0,        // 40  (
  0,65,34,28,0,        // 41  )
  20,8,62,8,20,        // 42  *
  8,8,62,8,8,          // 43  plus
  0,80,48,0,0,         // 44  ,
  8,8,8,8,8,           // 45  minus
  0,112,112,112,0,     // 46  .
  32,16,8,4,2,         // 47  /
  62,81,73,69,62,      // 48  0
  0,66,127,64,0,       // 49  1
  66,97,81,73,70,      // 50  2
  33,65,69,75,49,      // 51  3
  24,20,18,127,16,     // 52  4
  39,69,69,69,57,      // 53  5
  60,74,73,73,48,      // 54  6
  3,1,113,9,7,         // 55  7
  54,73,73,73,54,      // 56  8
  6,73,73,41,30,       // 57  9
  0,54,54,0,0,         // 58  :
  0,86,54,0,0,         // 59  ;
  8,20,34,65,0,        // 60  <
  20,20,20,20,20,      // 61  equals
  65,34,20,8,0,        // 62  >
  2,1,81,9,6,          // 63  ?
  50,73,121,65,62,     // 64  @
  126,17,17,17,126,    // 65  A
  127,73,73,73,54,     // 66  B
  62,65,65,65,34,      // 67  C
  127,65,65,65,62,     // 68  D
  127,73,73,73,65,     // 69  E
  127,9,9,9,1,         // 70  F
  62,65,73,73,122,     // 71  G
  127,8,8,8,127,       // 72  H
  65,65,127,65,65,     // 73  I
  32,64,65,63,1,       // 74  J
  127,8,20,34,65,      // 75  K
  127,64,64,64,64,     // 76  L
  127,2,12,2,127,      // 77  M
  127,6,24,96,127,     // 78  N
  62,65,65,65,62,      // 79  O
  127,9,9,9,6,         // 80  P
  62,65,81,33,94,      // 81  Q
  127,9,25,41,70,      // 82  R
  70,73,73,73,49,      // 83  S
  1,1,127,1,1,         // 84  T
  63,64,64,64,63,      // 85  U
  31,32,64,32,31,      // 86  V
  63,64,56,64,63,      // 87  W
  99,20,8,20,99,       // 88  X
  7,8,112,8,7,         // 89  Y
  97,81,73,69,67,      // 90  Z
  0,127,65,65,0,       // 91  [
  2,4,8,16,32,         // 92  back slash
  0,65,65,127,0,       // 93  ]
  4,2,1,2,4,           // 94  ^
  64,64,64,64,64,      // 95  _
  0,1,2,4,0,           // 96  quote
  32,84,84,84,120,     // 97  a
  127,72,68,68,56,     // 98  b
  56,68,68,68,32,      // 99  c
  56,68,68,72,127,     // 100  d
  56,84,84,84,24,      // 101  e
  8,126,9,1,2,         // 102  f
  8,84,84,84,60,       // 103  g
  127,8,4,4,120,       // 104  h
  0,72,125,64,0,       // 105  i
  32,64,68,61,0,       // 106  j
  127,16,40,68,0,      // 107  k
  0,65,127,64,0,       // 108  l
  124,4,24,4,120,      // 109  m
  124,8,4,4,120,       // 110  n
  56,68,68,68,56,      // 111  o
  124,20,20,20,8,      // 112  p
  12,18,18,20,126,     // 113  q
  124,8,4,4,8,         // 114  r
  72,84,84,84,36,      // 115  s
```

```
  4,63,68,64,32,       // 116  t
  60,64,64,32,124,     // 117  u
  28,32,64,32,28,      // 118  v
  60,64,48,64,60,      // 119  w
  68,40,16,40,68,      // 120  x
  12,80,80,80,60,      // 121  y
  68,100,84,76,68,     // 122  z
  0,65,54,8,0,         // 123  }
  0,0,127,0,0,         // 124  |
  0,8,54,65,0,         // 125  {
  8,4,8,16,8,          // 126  ~
  31,36,124,36,31      // 127  UT sign
};


// ********* lcdCmd***********
// Output command to AGM1264F 128-bit by 64-bit graphics display
// Inputs: 8-bit instruction
// Outputs: none
void lcdCmd(unsigned char instruction){
  // R/W=0, write mode  default, R/W=0 always
  // normally D/I will be left at D/I=1 for data
  DI = 0;        // D/I=0, COMMAND WRITE
  Timer_Wait(T1usec);
  E = 1;         // E pulse width > 450ns
  DATA = instruction;
  Timer_Wait(T1usec);
  E = 0;         // falling edge latch, setup time 200ns
  DI = 1;        // D/I=1 default state is data
  Timer_Wait(T4usec);
}

// ********* lcdData***********
// Output data to AGM1264F 128-bit by 64-bit graphics display
// Inputs: 8-bit data
// Outputs: none
void lcdData(unsigned char data){
  // R/W=0, write mode  default, R/W=0 always
  // normally D/I will be left at D/I=1 for data
  E = 1;         // E pulse width > 450ns
  DATA = data;
  Timer_Wait(T1usec);
  E = 0;         // falling edge latch, setup time 200ns
  Timer_Wait(T4usec);
}

// ********* LCD_Init***********
// Initialize AGM1264F 128-bit by 64-bit graphics display
// activates TCNT at 1.5 MHz, assumes PLL active
// Input: none
// Output: none
// does not clear the display
void LCD_Init(void){
  Timer_Init();   // TCNT at 1.5 MHz
  DDRH = 0xFF;    // PH7-PH0 outputs to DB7-DB0, PT3=E
  DDRP |= 0x0F;   // PP3-PP0 outputs to E,DI,CS1,CS2
  CS2 = 1;         // talk to both LCD controllers
  CS1 = 1;
  DI = 1;          // default mode is data
  E = 0;           // inactive
  Timer_Wait1ms(100); // let it warm up
  lcdCmd(0x3F);    // display=ON
  lcdCmd(0xB8);    // Page address (0 to 7) is 0
  lcdCmd(0x40);    // Column address (0 to 63) is 0
  lcdCmd(0xC0);    // Y=0 is at top
  OpenFlag = 1;    // device openopen
  Column1 = 0x41;  // column position
  bLeft1 = 1;
  bRight1 = 0;
  Page = 0xB8;
  bDown = 0;    // true if want font shifted down

}


// ********* LCD_Clear***********
// Clear the entire 1024 byte (8192 bit) image on the
//    AGM1264F 128-bit by 64-bit graphics display
```

```
// Input: value to write into all bytes of display RAM
// Output: none
// e.g., LCD_Clear(0);  // makes all pixels off
void LCD_Clear(unsigned char data){
unsigned char page;
  int i;
  if(OpenFlag == 0) return;
  for(page = 0xB8; page< 0xB8+8; page++){ // pages 0 to 7
    CS2 = 1;          // right enable
    CS1 = 0;
    lcdCmd(page);    // Page address (0 to 7)
    lcdCmd(0x40);    // Column = 0
    for(i=64; i>0; i--){
      lcdData(data);   // copy one byte to right side
    }
    CS2 = 0;
    CS1 = 1;         // left enable
    lcdCmd(page);    // Page address (0 to 7)
    lcdCmd(0x40);    // Column = 0
    for(i=64; i>0; i--){
      lcdData(data);   // copy one byte to left side
    }
  }
}

// page   is 0xB8 to 0xBF for pages 0 to 7
// column is 0x40 to 0x7F for columns 0 to 63
void OutByte(unsigned char page, unsigned char column,unsigned char data){
  lcdCmd(page);    // Page address (0 to 7)
  lcdCmd(column);  // Column = 0 to 63
  lcdData(data);   // data
}

int pixelOn(int type, int x, int y) {
  switch(type) {
    case SHIPEND_UP:
      if((x == 2 && y == 3) ||
         (x == 3 && y == 2) ||
         (x == 3 && y == 3) ||
         (x == 3 && y == 4) ||
         (x == 4 && y == 2) ||
         (x == 4 && y == 3) ||
         (x == 4 && y == 4) ||
         (x == 5 && y == 2) ||
         (x == 5 && y == 3) ||
         (x == 5 && y == 4)) {

        return 1;
      }
      break;
    case SHIPEND_DOWN:
      if((x == 1 && y == 2) ||
         (x == 1 && y == 3) ||
         (x == 1 && y == 4) ||
         (x == 2 && y == 2) ||
         (x == 2 && y == 3) ||
         (x == 2 && y == 4) ||
         (x == 3 && y == 2) ||
         (x == 3 && y == 3) ||
         (x == 3 && y == 4) ||
         (x == 4 && y == 3)) {

        return 1;
      }
      break;
    case SHIPEND_LEFT:
      if((x == 2 && y == 3) ||
         (x == 2 && y == 4) ||
         (x == 2 && y == 5) ||
         (x == 3 && y == 2) ||
         (x == 3 && y == 3) ||
         (x == 3 && y == 4) ||
         (x == 3 && y == 5) ||
         (x == 4 && y == 3) ||
         (x == 4 && y == 4) ||
         (x == 4 && y == 5)) {

        return 1;
```

```
    }
    break;
  case SHIPEND_RIGHT:
    if((x == 2 && y == 1) ||
       (x == 2 && y == 2) ||
       (x == 2 && y == 3) ||
       (x == 3 && y == 1) ||
       (x == 3 && y == 2) ||
       (x == 3 && y == 3) ||
       (x == 3 && y == 4) ||
       (x == 4 && y == 1) ||
       (x == 4 && y == 2) ||
       (x == 4 && y == 3)) {

      return 1;
    }
    break;
  case SHIP_VERT:
    if((x == 1 && y == 2) ||
       (x == 1 && y == 3) ||
       (x == 1 && y == 4) ||
       (x == 2 && y == 2) ||
       (x == 2 && y == 3) ||
       (x == 2 && y == 4) ||
       (x == 3 && y == 2) ||
       (x == 3 && y == 3) ||
       (x == 3 && y == 4) ||
       (x == 4 && y == 2) ||
       (x == 4 && y == 3) ||
       (x == 4 && y == 4) ||
       (x == 5 && y == 2) ||
       (x == 5 && y == 3) ||
       (x == 5 && y == 4)) {

      return 1;
    }
    break;
  case SHIP_HORIZ:
    if((x == 2 && y == 1) ||
       (x == 2 && y == 2) ||
       (x == 2 && y == 3) ||
       (x == 2 && y == 4) ||
       (x == 2 && y == 5) ||
       (x == 3 && y == 1) ||
       (x == 3 && y == 2) ||
       (x == 3 && y == 3) ||
       (x == 3 && y == 4) ||
       (x == 3 && y == 5) ||
       (x == 4 && y == 1) ||
       (x == 4 && y == 2) ||
       (x == 4 && y == 3) ||
       (x == 4 && y == 4) ||
       (x == 4 && y == 5)) {

      return 1;
    }
    break;
  case HIT:
    if((x == 1 && y == 3) ||
       (x == 2 && y == 2) ||
       (x == 2 && y == 3) ||
       (x == 2 && y == 4) ||
       (x == 3 && y == 1) ||
       (x == 3 && y == 2) ||
       (x == 3 && y == 4) ||
       (x == 3 && y == 5) ||
       (x == 4 && y == 2) ||
       (x == 4 && y == 3) ||
       (x == 4 && y == 4) ||
       (x == 5 && y == 3)) {

      return 1;
    }
    break;
  case MISS:
    if((x == 1 && y == 1) ||
       (x == 1 && y == 5) ||
       (x == 2 && y == 2) ||
```

```
          (x == 2 && y == 4) ||
          (x == 3 && y == 3) ||
          (x == 4 && y == 2) ||
          (x == 4 && y == 4) ||
          (x == 5 && y == 1) ||
          (x == 5 && y == 5)) {

        return 1;
      }
      break;
  }
  return 0;
}

void LCD_DrawGrid(unsigned char field[10][10]) {
  int i, j, k;

  //PTP |= 0x80;

  CS1 = 0;
  CS2 = 1;

  for(i=0; i<8; i++) {
    for(j=0; j<61; j++) {
      unsigned char pixels = 0;
      if(!(j%6)) {
        if(i<7) {
          pixels = 0xFF;
        }
        else {
          pixels = 0x1F;
        }
      }
      else {
        switch(i) {
          case 0:
          case 3:
          case 6:
            pixels = 0x41;
            break;
          case 1:
          case 4:
          case 7:
            pixels = 0x10;
            break;
          case 2:
          case 5:
            pixels = 0x04;
            break;
        }
        for(k=0; k<8 && (i<7 || k<4); k++) {
          unsigned char boxRow = ((i*8)+k)/6;
          unsigned char boxCol = j/6;

          unsigned char boxX = ((i*8)+k)%6;
          unsigned char boxY = j%6;

          if(pixelOn(field[boxRow][boxCol], boxX, boxY)) {
            pixels |= 1 << k;
          }
        }
      }
      OutByte(i + 0xB8, j + 0x40, pixels);
    }
  }
}

// ********* LCD_OutChar***********
// Output ASCII character on the
//    AGM1264F 128-bit by 64-bit graphics display
// Input: 7-bit ASCII to display
// Output: none
// letter must be between 32 and 127 inclusive
// execute LCD_GoTo to specify cursor location
void LCD_OutChar(unsigned char letter){
unsigned short i,cnt;
  if(OpenFlag == 0) return;
// page 0 is 0xB8, varies from 0xB7 to 0xBF
```

```
  if(letter<32) return;
  if(letter>127) return;
  i = 5*(letter-32); // index into font table
  CS2 = bRight1;   // right enable
  CS1 = bLeft1;    // left enable
  lcdCmd(Page);    // Page address 0 to 7
  lcdCmd(Column1); // Column = 0
  for(cnt=5; cnt>0; cnt--){
    if(bDown){
      lcdData(Font[i]<<1);  // copy one byte, shifted down
    } else{
      lcdData(Font[i]);  // copy one byte
    }
    i++;
    Column1++;
    if(bLeft1&&(Column1==0x80)){
      Column1 = 0x40;
      bLeft1 = 0;
      bRight1 = 1;     // switch to right side
      CS2 = bRight1;   // right enable
      CS1 = bLeft1;    // left enable
      lcdCmd(Page);    // Page address 0 to 7)
      lcdCmd(Column1); // Column = 0
    }
    if(bRight1&&(Column1==0x7F)){
      Column1 = 0x41;
      bLeft1 = 1;
      bRight1 = 0;     // switch to left side
      CS2 = bRight1;   // right enable
      CS1 = bLeft1;    // left enable
      lcdCmd(Page);    // Page address 0 to 7)
      lcdCmd(Column1); // Column = 0
    }
  }
  lcdData(0);  // inter-character space copy one byte
  Column1++;
  if(bLeft1&&(Column1==0x80)){
    Column1 = 0x40;
    bLeft1 = 0;
    bRight1 = 1;     // switch to right side
    CS2 = bRight1;   // right enable
    CS1 = bLeft1;    // left enable
    lcdCmd(Page);    // Page address 0 to 7)
    lcdCmd(Column1); // Column = 0
  }
  if(bRight1&&(Column1==0x7F)){
    Column1 = 0x41;
    bLeft1 = 1;
    bRight1 = 0;     // switch to left side
    CS2 = bRight1;   // right enable
    CS1 = bLeft1;    // left enable
    lcdCmd(Page);    // Page address 0 to 7)
    lcdCmd(Column1); // Column = 0
  }
}



//---------------------LCD_OutString--------------
// Display String
// Input: pointer to NULL-terminationed ASCII string
// Output: none
void LCD_OutString(char *pt){
  if(OpenFlag==0){
    return;  // not open
  }
  while(*pt){
    LCD_OutChar((unsigned char)*pt);
    pt++;
  }
}

//------------------------LCD_GoTo-------------------------
// Move cursor
// Input: line number is 1 to 8, column from 1 to 21
// Output: none
// errors: it will ignore legal addresses
void LCD_GoTo(int line, int column){
```

```
  if(OpenFlag==0){
    return;  // not open
  }
  if((line<1) || (line>8)) return;
  if((column<1) || (column>21)) return;
  if(line<5){
    bDown = 0;              // normal position on lines 1,2,3,4
  } else{
    bDown = 0xFF;           // shifted down on lines 5,6,7,8
  }
  Page = 0xB8+line-1;       // 0xB8 to 0xBF
  if(column<12){
    Column1 = 59+6*column; // 0x41+6*(column-1);
    bLeft1 = 1;
    bRight1 = 0;            // on left side
  } else{
    Column1 = 6*column-5;  // 0x43+6*(column-12);
    bLeft1 = 0;
    bRight1 = 1;            // on right side
  }
}
```

```
void Key_Init(void);

void enableOC6(void (*function) (void), unsigned short delay, unsigned short delayCount, unsigned s ¬
hort count);
```

```
#include <mc9s12dp512.h>     /* derivative information */
#include "game.h"

#define DEBOUNCE_DELAY 10

// UP     PT5
// DOWN   PT4
// LEFT   PT3
// RIGHT  PT2
// A      PT1
// B      PT0

static void (*OC6Func) (void);
unsigned static short OC6Delay;
unsigned static short OC6DelayCount1;
unsigned static short OC6DelayCount2;
unsigned static short OC6Count;

void Key_Init(void){
asm sei            // make atomic
  DDRT &= ~0x3F;    // PT7,PT6 all rows are output
  PERT  =  0x3F;    // internal pullup on PT3,PT2
  TCTL3 =  0x05;
  TCTL4 =  0x55;   // falling edges IC3,IC2
  TIOS  =  0xC0;
  TIE   =  0x3F;     // Arm only IC3,IC2
asm cli
}

void enableOC6(void (*function) (void), unsigned short delay, unsigned short delayCount, unsigned s            ¬
hort count) {
  asm sei
  TIE |= 0x40;
  OC6Func = function;
  OC6Delay = delay;
  OC6DelayCount1 = delayCount;
  OC6DelayCount2 = delayCount;
  OC6Count = count;
  TC6 = TCNT + OC6Delay;
  asm cli
}

void interrupt 8 IC0Han(void) {
  Game_B();
  TFLG1 = 0x01;
}

void interrupt 9 IC1Han(void) {
  Game_A();
  TFLG1 = 0x02;
}

void interrupt 10 IC2Han(void) {
  Game_DPad(RIGHT);
  TFLG1 = 0x04;
}

void interrupt 11 IC3Han(void) {
  Game_DPad(DOWN);
  TFLG1 = 0x08;
}

void interrupt 12 IC4Han(void) {
  Game_DPad(LEFT);
  TFLG1 = 0x10;
}

void interrupt 13 IC5Han(void){
  Game_DPad(UP);
  TFLG1 = 0x20;
}

void interrupt 14 OC6Han(void){
  TFLG1 = 0x40;
  if(!OC6DelayCount2) {
    OC6DelayCount2 = OC6DelayCount1;
    (*OC6Func)();
```

```
      OC6Count--;
      if(!OC6Count) {
        TIE &= ~0x40;
      }
    }
    else {
      OC6DelayCount2--;
    }

    TC6 = TCNT + OC6Delay;
  }
```

```
// File **********Timer.h**********
// Timer wait routine, 9S12DP512
// assumes PLL is active and E clock is 24 MHz
// TCNT will become 1.5MHz
// Jonathan W. Valvano 1/27/09

//  This example accompanies the books
//   "Embedded Microcomputer Systems: Real Time Interfacing",
//         Thomson Engineering, copyright (c) 2006,
//   "Introduction to Embedded Microcomputer Systems:
//    Motorola 6811 and 6812 Simulation", Thomson, copyright (c) 2002

// Copyright 2007 by Jonathan W. Valvano, valvano@mail.utexas.edu
//    You may use, edit, run or distribute this file
//    as long as the above copyright notice remains



//---------------------Timer_Init---------------------
// activate TCNT at 1.5 MHz
// inputs: none
// outputs: none
void Timer_Init(void);


//---------Timer_Wait------------------
// fixed time delay
// inputs: time to wait in 667ns cycles
// outputs: none
void Timer_Wait(unsigned short delay);

//---------Timer_Wait1ms------------------
// fixed time delay
// inputs: time to wait in ms
// outputs: none
// 1500 cycles equals 1ms
void Timer_Wait1ms(unsigned short delay);

//---------Timer_Wait10ms------------------
// fixed time delay
// inputs: time to wait in 10ms
// outputs: none
// 15000 cycles equals 10ms
void Timer_Wait10ms(unsigned short delay);
```

```
// File **********Timer.C***********
// Timer wait routines, 9S12DP512
// assumes PLL is active and E clock is 24 MHz
// TCNT will become 1.5MHz
// Jonathan W. Valvano 1/27/09

//  This example accompanies the books
//   "Embedded Microcomputer Systems: Real Time Interfacing",
//         Thomson Engineering, copyright (c) 2006,
//   "Introduction to Embedded Microcomputer Systems:
//    Motorola 6811 and 6812 Simulation", Thomson, copyright (c) 2002

// Copyright 2009 by Jonathan W. Valvano, valvano@mail.utexas.edu
//    You may use, edit, run or distribute this file
//    as long as the above copyright notice remains

#include <mc9s12dp512.h>     /* derivative information */



//--------------------Timer_Init---------------------
// activate TCNT at 1.5 MHz, assumes 24 MHz E clock
// inputs: none
// outputs: none
void Timer_Init(void){
  asm sei         // make ritual atomic
  TSCR1 = 0x80;   // Enable TCNT, 24 MHz E clock
  TSCR2 = 0x04;   // divide by 16 TCNT prescale, TOI disarm
  PACTL = 0;      // timer prescale used for TCNT
/* Bottom three bits of TSCR2 (PR2,PR1,PR0) determine TCNT period
    divide  FastMode(24MHz)    Slow Mode (4MHz)
000   1     42ns  TOF  2.73ms  250ns TOF 16.384ms
001   2     84ns  TOF  5.46ms  500ns TOF 32.768ms
010   4    167ns  TOF  10.9ms    1us TOF 65.536ms
011   8    333ns  TOF  21.8ms   2us TOF 131.072ms
100  16    667ns  TOF  43.7ms   4us TOF 262.144ns
101  32   1.33us  TOF  87.4ms     8us TOF 524.288ms
110  64   2.67us  TOF 174.8ms   16us TOF 1.048576s
111 128   5.33us  TOF 349.5ms   32us TOF 2.097152s */
}


//---------Timer_Wait------------------
// fixed time delay
// inputs: time to wait in 667ns cycles
// outputs: none
void Timer_Wait(unsigned short delay){
unsigned short startTime;
  startTime = TCNT;
  while((TCNT-startTime) <= delay){}
}

//---------Timer_Wait1ms------------------
// fixed time delay
// inputs: time to wait in ms
// outputs: none
// 1500 cycles equals 1ms
void Timer_Wait1ms(unsigned short delay){
  for(;delay>0;delay--){
    Timer_Wait(1500);
  }
}

//---------Timer_Wait10ms------------------
// fixed time delay
// inputs: time to wait in 10ms
// outputs: none
// 15000 cycles equals 10ms
void Timer_Wait10ms(unsigned short delay){
  for(;delay>0;delay--){
    Timer_Wait(15000);
  }
}
```