Project Final Report:

X-10 Protocol & Power Line Communication

By:

Juan Falquez

CSci 297 – Embedded Systems

The George Washington University

Spring 2009

1 Project Abstract

People have always strived to create a comfortable environment for themselves, and as technology evolves this has become particularly easier. The idea of “Intelligent Homes” is precisely one of such cases: a home that requires minimum interaction from people in order to perform some basic actions, and also, that these actions could be executed automatically or even remotely if the need arises. This home automation concept, also known as smart homes or domotics, have had a great proliferation thanks to inexpensive equipments and protocols like X-10.

Since X-10 modules are so popular now, the question we would like to answer is: what if we use a special module to transmit computer data in the same way X-10 transmits commands? It is the objective of this project to build an X-10 interface capable of transmitting this computer data. Thus, we could effectively connect two computers in a house via the power lines without the need of re-wiring. There are some performance issues, however, and we will also discuss this at the end of this project.

Note:
For this project, we assume that the reader is already familiar with how X-10 works. For more information on X-10 functionality, please read the “X-10 Protocol and Power line Communication” report.

2 Status

The project works as intended, but the transmission rates are extremely low making it only viable if the data we wish to transmit is not time-dependant. Although it was expected to achieve approximately 20 bits per second, in reality around 9 bits per second were achieved due to the way these particular X-10 modules worked. So this project is ideal when transmitting small data fragments or if a centralized X-10 “command and control” server is desired. It can be even used as an X-10 packet sniffer off the power line.

3 Specification

The project consists of 3 components: a TW523 X-10 module which bridges the power line with the microcontroller, a ZNEO microcontroller which translates X-10 to RS232 and vice-versa, and a computer connected via the serial port.

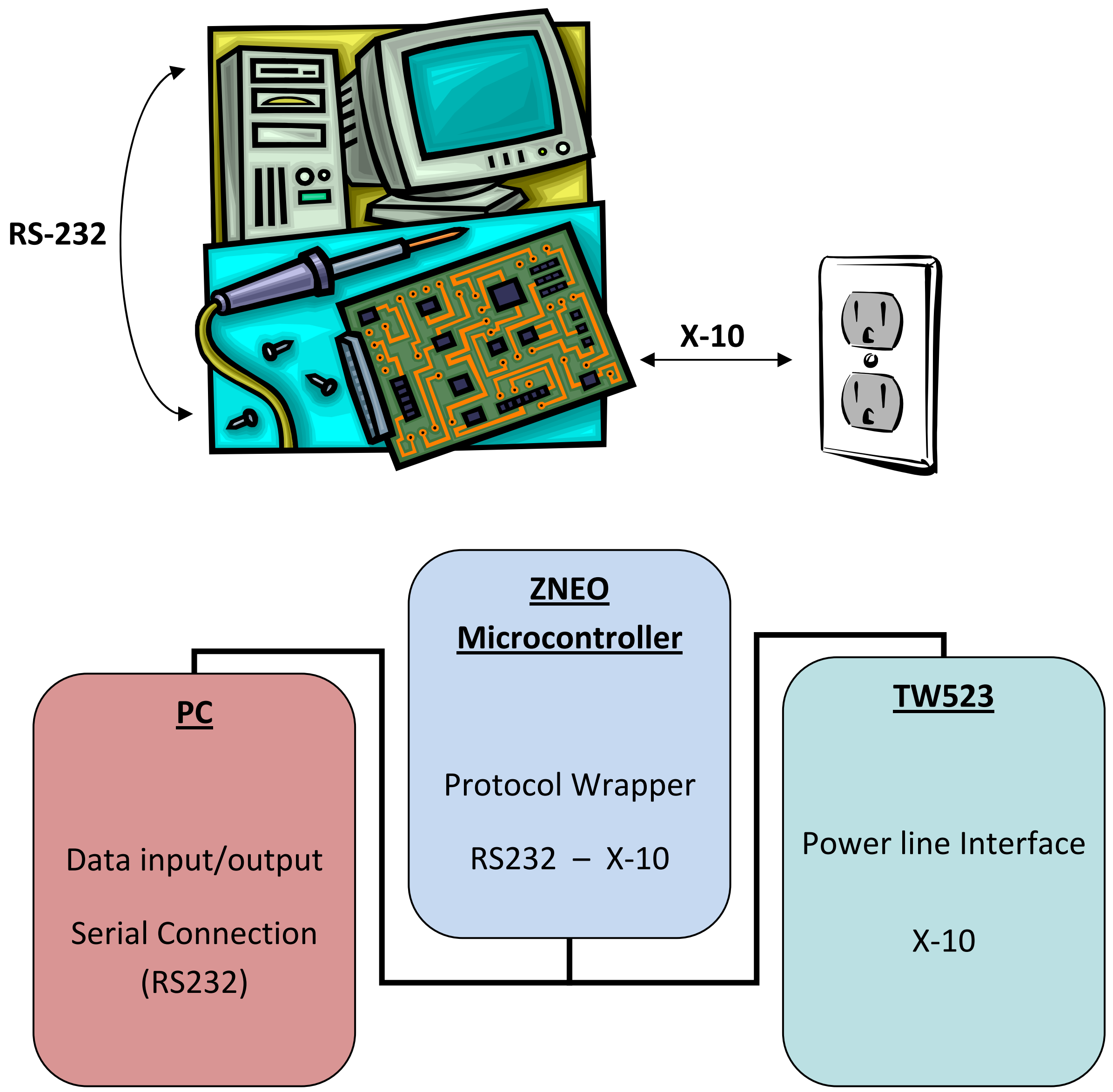

Figure 1: *General system overview.*

4 Implementation & Construction

4.1 Hardware Components

4.1.1 Computer

The computer used is the normal general purpose computer we use every day. The computer must have a serial port which is used to connect to the ZNEO. Hyperterminal or any other type of terminal software is used to send and view data.

4.1.2 TW523

The TW523 is a two-way power line interface that allows you to send and receive X-10. The module is optically isolated, so there is no potential of damaging the microcontroller. It provides 4 pins that can be plugged using an RJ9 jack. A photo of the module with the corresponding block diagram can be seen below:

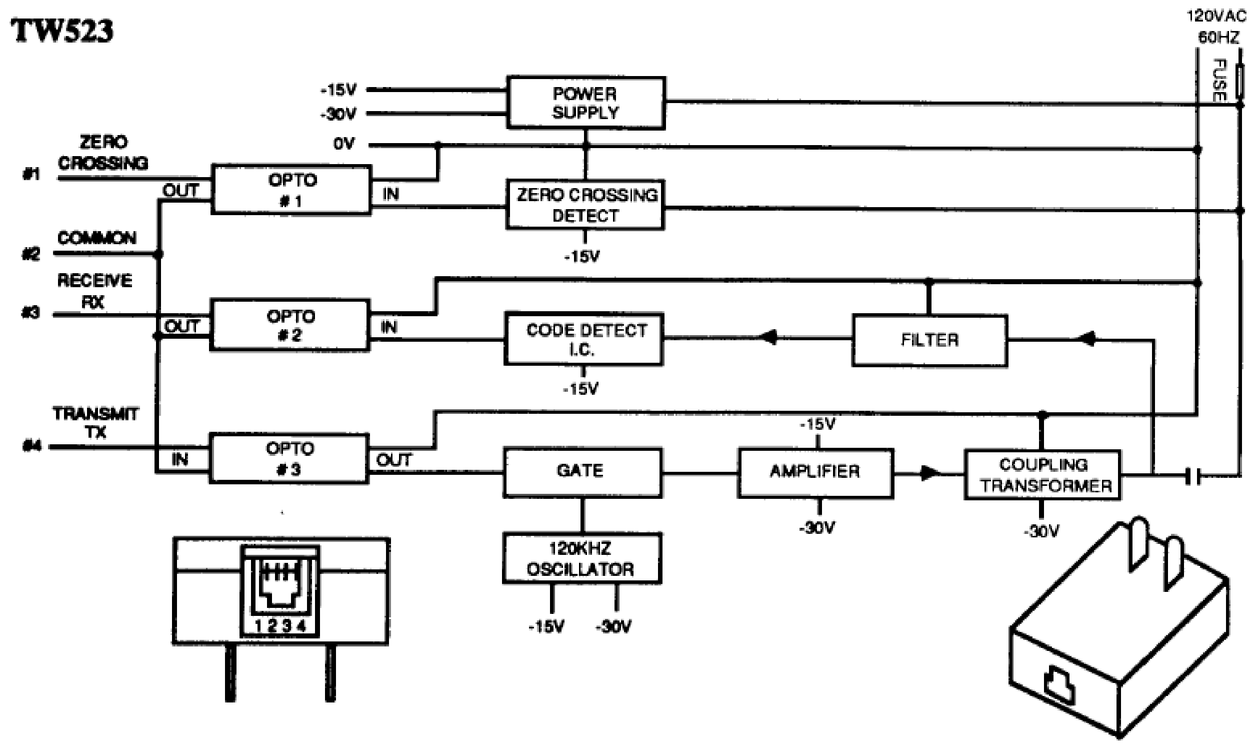

Figure 2: *TW523 modules and block diagram.*

As we can see, the pins are used for transmitting and receiving data as well as detecting the zero-crossing point in the power line. Table 1 shows the pin summary of the TW523:

Pin Number	Description
1	Zero-crossing Detect
2	Ground/Common
3	Data Output to ZNEO
4	Data Input from ZNEO

Table 1: *TW523 Pin Summary*

The TW523 outputs (the zero-crossing detect and the data output) are open collector transistors. Therefore, a pullup resistor is required to generate a logical level. Below is the schematic to connect the TW523 to the ZNEO.

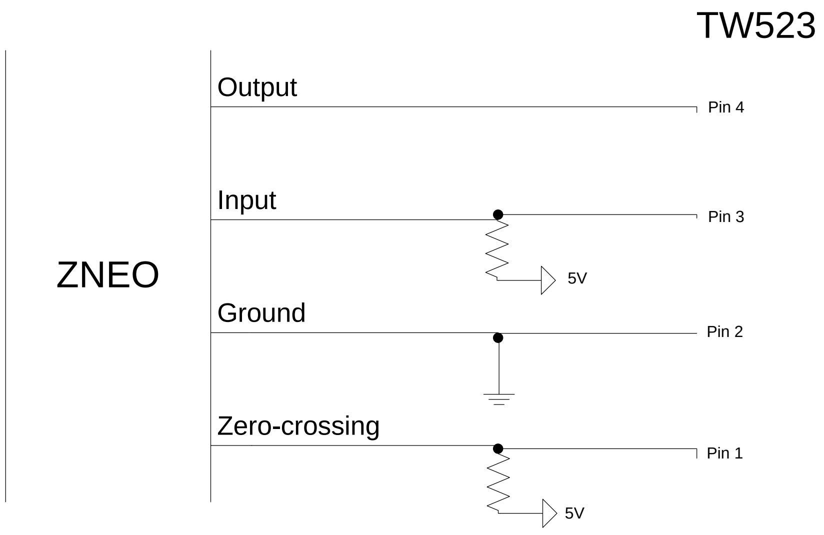

Figure 3: *TW523 with pullups diagram.*

4.1.3 ZNEO Kit

The ZNEO 16 bit microcontroller from Zilog is used as a bridging module between RS232 and X-10. Input data is received from the computer via the serial port using the microcontroller’s UART. A typical serial cable is connected between the computer and the ZNEO.

For the TW523, the pins are connected to specific GPIOs. Table 2 shows which pins are connected to which GPIOs.

TW523 Pin	ZNEO Pin
Zero-crossing Detect	PC0
Ground/Common	Ground
Data Output to ZNEO	PE0
Data Input from ZNEO	PD0

Table 2: *TW523 and ZNEO Pin Mapping*

As seen in section 4.1.2, the TW523 output pins must be pulled up so a direct connection with the ZNEO is not possible. The only exception is the data output which can be connected directly to the TW523's data input pin with no prior conditioning.

4.2 Software Components

The only main software component resides on the ZNEO. Zilog's IDE is used to compile and program the ZNEO board. All the pins are used as GPIOs, PE0 as data ouput and PD0 as data input. PC0 is the zero-crossing detect pin, and interrupts on both edges is enabled in this pin. PC0 is the heartbeat of the whole system, since the X-10 module cannot transmit or receive data but in specific windows. As such, timing is critical for the system to work properly. Figure 4 shows a timing diagram for a signal transmission.

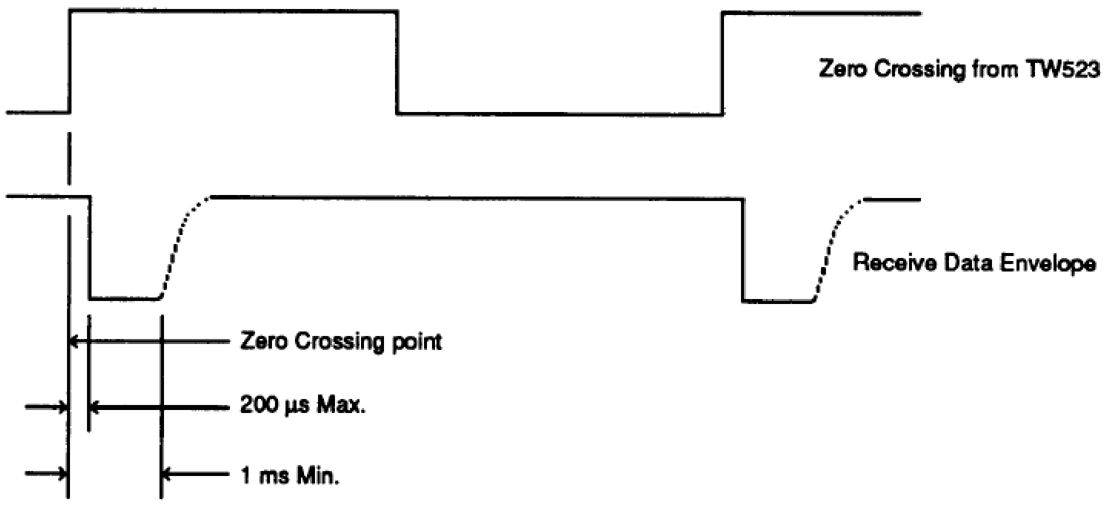

Figure 4: *Timing diagram.*

The zero-crossing pin gives us a 60Hz square signal, corresponding to the frequency of the typical AC wave. The system will only perform actions whenever an interrupt is received. There are 3 main states that the system can be in: idle, transmitting or receiving. Receiving is the highest priority, which means that if the system is in the receiving state and someone wishes to transmit the operation will be delayed until the X-10 frame is completely received. If the system is in the receiving state, it will delay 300 microseconds (as per the X-10 protocol) before sensing the line and retrieving a high (1) or a low (0).

If the system is transmitting, however, it will have to send the signal immediately after the zero-crossing is detected and up to 1 millisecond before latching the signal back to low. Finally, if the system is idle it means that it is free to either transmit or receive data, or obtain new data from the serial port.

The data being transmitted by the system consists of ASCII characters. Since the X-10 protocol contains 4 bits for the house code and 5 for the unit/command code, we have 9 bits overall that we can use to send information. ASCII characters contain 8 bits, which leaves 1 bit free if we wish to send one character per X-10 frame. Once the characters are received through Hyperterminal via the serial cable, the ZNEO will proceed to wrap the data into a new X-10 frame. For example, if a letter 'A' is received the ZNEO decodes it as 0100 0001. It will then append the start pattern and format the data as X-10 (using complement form). In this particular case, the final data transmitted would be:

11 10	+	01 10 01 01 01 01 01 10	+	01
Start Header		***X-10 Letter 'A'***		***Padded bit***

Each pair of bits corresponds to one full power cycle. Thus, it takes 11 cycles to transmit one complete X-10 frame to the TW523.

5 Retrospective

As we can see earlier, it takes 11 cycles to transmit a full X-10 frame to the TW523. This means it would take approximately 0.2 second to transmit the whole frame (or in this case, 1 ASCII character) at the frequency of the power line (60Hz). However, this is not the case. It takes 11 cycles for the frame to reach the TW523, but the module then needs to transmit this through the power line again and not once but twice (as the X-10 protocol requires). Finally, once the TW523 from the other end receive the frame twice, it will take another 11 cycles to fully transmit the character to the ZNEO. So it takes 4 times as long! Approximately 0.8 seconds are required to transmit a single character between the ZNEOs.

This problem lies mostly on the way the TW523 works. It does not allow us to manipulate signals directly over the power line, but rather it buffers first our X-10 and checks to see if it is valid. If it is, it will then transmit it. There is a project called “Modified TW523” which precisely aims to override this functionality, and allow direct manipulation of the signals over the power line. With this modified module, we could implement our own protocols without the need of using X-10.

This last is a very important idea, since currently the system could interfere with the normal functionality of X-10 if used in a house that has X-10 modules. For instance, if you send the letter ‘f’ through the system, a module with house code ‘A’ and unit code ‘1’ will think we are talking to it. There are ways to avoid this, like specifying a house and unit code for the other terminal we wish to communicate with. But in doing so, the throughput would decrease by half since it would take 2 frames instead of 1 to transmit a single character: the first frame contains the address of the terminal, and the second is the actual character. However, this again could be avoided if we had a way to manipulate directly the power line signals and then implement our own custom protocol instead of relying on X-10.

Finally, a minor improvement would be to use all bits in the X-10 frame instead of padding the last one. However, since the amount of data being transmitted is relatively small this would only increment throughput but little.

Attachments

Datasheets

- TW523 Datasheet: ftp://ftp.x10.com/pub/manuals/technicalnote.pdf
- Modified TW523 Project: http://home.comcast.net/~ncherry/mTW523/

Source Code

```
/***************************************************************************
 Author: Juan M. Falquez
 Class: CSCI-297 The George Washington University
 Final Project

 Other files:
      binary.h - Binary labels
      timer.h  - Timer configuration

 Notes:
      PE0 - Data INPUT (w/ 5v Pullup)
      PD0 - Data OUTPUT
      PC0 - Zero Crossing (w/ 5v Pullup)
***************************************************************************/


#include <zneo.h>
#include <stdio.h>
#include <sio.h>
#include "binary.h"
#include "timer.h"




/*
 *    TIMING ROUTINES
 */
void delay_ms(int ms);
void delay_us(int us);



/*
 *    DEFINES AND GLOBAL VARIABLES
 */
#define IDLE 0                   // The system is idle
#define TX 1                     // The system is transmitting
#define RX 2                     // The system is receiving

int state;                       // Variable that holds the state
int newmsg;                      // Flag that tells us if a new message arrived
```

```c
int ii,oi;                    // Input and Output auxilary variable
int ipos,fpos;                // Initial and Final position of our buffer
char buffer[80];              // Buffer
int ibuffer[30];              // Buffer to parse input codes
int obuffer[30];              // Buffer to parse output codes


// Auxilary routines
void pc0_isr(void);           // Routine triggered on every zero-crossing
void x10format(char x);       // Codes a character into the X10 format



/*
 * MAIN ROUTINE
 * The initial state of the system is IDLE. The routine simply alternates
 * between reading from serial, sending data if there is any, and checking
 * to see if there is any new data.
 */
void main(void) {
      char ic,oc;             // Input and Output characters aux variables

      SET_VECTOR(C0,pc0_isr); // Assign interrupt vector
      PCIMUX |= b00000001;    // Select PC port for interrupts
      IRQ2ENL |= b00000001;   // Enable interrupts on PC0

      PDDD &= b11111110;      // PD0 Data Direction OUT (Data Output)
      PEDD |= b00000001;      // PE0 Data Direction IN (Data Input)
      PCDD |= b00000001;      // PC0 Data Direction IN (Zero Crossing)

    EI();                     // Enable interrupts

    init_uart(_UART0, _DEFFREQ, _DEFBAUD);      // Initialize serial

      state = IDLE;           // Initial state set to idle
      ipos = 0;               // Initial positions of buffer is 0
      fpos = 0;

      while(1) {
            // Check to see if there is some data coming from serial
        if (kbhit()) {
                  // If there is, store in buffer
            ic = getchar();
                  buffer[fpos] = ic;
                  printf("%c",ic);
                  fpos++;
                  // If a carriage return is seen,
                  // flag to transmit buffer contents
                  if (ic == '\n') {
                        ipos = 0;
                        x10format(buffer[ipos]);
                        state = TX;
                        buffer[fpos] = '\0';
                        printf("Sending ...");
                  }
        }
```

```
            // Check to see if some data arrived
            if (ii > 21) {
                  // Convert X10 into ASCII character
                  oc = (ibuffer[6] * 64) + (ibuffer[8] * 32) + (ibuffer[10] *
                        16) + (ibuffer[12] * 8) + (ibuffer[14] * 4) +
                        (ibuffer[16] * 2) + ibuffer[18];
                  printf("%c",oc);
                  ii = 0;
                  // If end of line is found, reset new message flag
                  if (oc == '\n') {
                        newmsg = 0;
                        printf("\n");
                  }
                  state = IDLE;
            }
    }
}


/*
 * ZERO CROSSING INTERRUPT ROUTINE
 * This routine is called everytime the zero crossing is detected
 * (either rising or falling edge).
 */
void interrupt pc0_isr(void) {

      // If we are transmitting ...
      if (state == TX) {
            if(oi < 28) {
            if(obuffer[oi] == 1)
                  PDOUT |= b00000001;
            else
                  PDOUT &= b11111110;
            delay_ms(1);
            PDOUT &= b11111110;
            }
            oi++;
            // This part is to wait for the propagation of the X10 frame
            if(oi == 50) {
                  ipos++;
                  oi = 0;
                  // If initial and final position of buffer are the same,
                  // we finished transmitting!
                  if(ipos == fpos) {
                        ipos = 0;
                        fpos = 0;
                        state = IDLE;
                        printf(" done.\n\n");
                  } else {
                        x10format(buffer[ipos]);
                  }
            }
      } else {
      // If we are not transmitting, sense the line
            delay_us(300);
            if (state == IDLE) {
            // Check to see if new frame is coming ...
```

```
                    if (!(PEIN & b00000001)) {
                          if (newmsg == 0) {
                                newmsg = 1;
                                printf("Remote: ");
                          }
                          state = RX;
                          ibuffer[0] = 1;
                          ii=1;
                    }
              } else {
              // ... otherwise continue transmitting the previous frame.
                    if (PEIN & b00000001)
                          ibuffer[ii] = 0;
                    else
                          ibuffer[ii] = 1;
                    ii++;
              }
        }
  }



  /*
   * X10 FORMAT ROUTINE
   * This routine takes a character and converts it into the appropriate
   * X10 code.
   */
  void x10format(char x) {
        // Start code
        obuffer[0]=1;
        obuffer[1]=1;
        obuffer[2]=1;
        obuffer[3]=0;

        // First bit of ASCII is always 0
        obuffer[4]=0;
        obuffer[5]=1;

        if (x & b01000000) {
              obuffer[6]=1;
              obuffer[7]=0;
        } else {
              obuffer[6]=0;
              obuffer[7]=1;
        }

        if (x & b00100000) {
              obuffer[8]=1;
              obuffer[9]=0;
        } else {
              obuffer[8]=0;
              obuffer[9]=1;
        }
        if (x & b00010000) {
              obuffer[10]=1;
              obuffer[11]=0;
        } else {
              obuffer[10]=0;
              obuffer[11]=1;
        }
        if (x & b00001000) {
              obuffer[12]=1;
              obuffer[13]=0;
        } else {
              obuffer[12]=0;
              obuffer[13]=1;
        }
        if (x & b00000100) {
              obuffer[14]=1;
              obuffer[15]=0;
        } else {
              obuffer[14]=0;
              obuffer[15]=1;
        }
        if (x & b00000010) {
              obuffer[16]=1;
              obuffer[17]=0;
        } else {
              obuffer[16]=0;
              obuffer[17]=1;
        }
        if (x & b00000001) {
              obuffer[18]=1;
              obuffer[19]=0;
        } else {
              obuffer[18]=0;
              obuffer[19]=1;
        }

        // Last bit padded
        obuffer[20]=0;
        obuffer[21]=1;

        // Three "silent" cycles
        obuffer[22]=0;
        obuffer[23]=0;

        obuffer[24]=0;
        obuffer[25]=0;

        obuffer[26]=0;
        obuffer[27]=0;

  }




  /********************************************************************
                            TIMING  ROUTINES
   ********************************************************************/
```

```c
/*
 * DELAY ROUTINE
 * It uses timer 2 to create a one shot timer with a reload value of
 * calculated depending on the amount of mili/micro seconds required to delay.
 */
void delay_ms(int ms) {
    PADD &= 0xFE;
    PAOUT |= 0x01;

    T2CTL0 = 0;                     // Disable, no settings
    T2CTL1 = TIMER_PRESCALE_2;      // Prescale value
    T2HL = 1;                       // Initial counter value
    T2R = CLOCK/2/1000 * ms;        // Reload value
    T2CTL1 |= TIMER_ENABLE;         // Start timer
    while(T2CTL1 & 0x80) ;          // Wait for it to be done

    PAOUT &= 0xFE;
}

void delay_us(int us) {
    PADD &= 0xFE;
    PAOUT |= 0x01;

    T2CTL0 = 0;                     // Disable, no settings
    T2CTL1 = TIMER_PRESCALE_1;      // Prescale value
    T2HL = 1;                       // Initial counter value
    T2R = CLOCK/1/1000000 * us;     // Reload value
    T2CTL1 |= TIMER_ENABLE;         // Start timer
    while(T2CTL1 & 0x80) ;          // Wait for it to be done

    PAOUT &= 0xFE;
}
```