Appendix A
Software

I. Assembly Code For The Base Unit

Module #1: Setup

```
; The following "$" commands must be included in every module

$NOMOD51                      ; Omit assembler predefined registers.
$INCLUDE(reg515.inc)          ; Include 515/535 microcontroller definitions.

NAME  SETUP                   ; Optional parameter; if no name is provided,
                              ; the filename will be used by default.

EXTRN CODE    (MAIN_LOOP,SERINIT)
EXTRN CODE    (LCDINIT,KPD)   ; Makes subroutine MAIN_LOOP in the
                              ; main.a51 module available to module
                              ; setup.a51 .

;*********************************************************************************

ST_ADDR equ     8000h         ; Set program starting address at 8000h.

CSEG  AT      ST_ADDR         ; Places beginning of code at in a fixed memory
                              ; location specified by ST_ADDR = 8000h.
                              ; This is referred to as an
                              ; "absolute code segment", and cannot be relocated.
BEGIN:
      LJMP    START           ; Jump to start of program.

ST_SEG  SEGMENT CODE          ; Reserve RAM space for 80535 initialization
                              ; code segment, ST_SEG. Again, since this is a
                              ; "generic segment", it is relocatable.

      RSEG    ST_SEG          ; Places the code segement containing
                              ; START at this point in assembled code.
                              ; The selected segment remains "active" until
                              ; a different segment is specified.

      USING   0               ; Indicates to the assembler that register
                              ; bank 0 will be used, but does not
                              ; actually select register bank 0.

; Place code for initializations specific to the fundamental operation of the
; EMAC MicroPac 80535 microcontroller board here.

START:
      CLR     PSW.4           ; Selects register bank 0.
      CLR     PSW.3           ; PSW bits 3 and 4 dictate register bank.

      MOV     SP,#60h         ; Initialize stack pointer to 60h. Note that

        ; the stack pointer could be initialized
        ; to any value between 20h and 7Fh. However, the programmer must
        ; ensure (1) stack has enough space to expand adequately, and
        ; (2) does not overwrite user data.

      MOV     IEN0,#0         ; Disable all interrupts.

      SETB    P5.5            ; Activates the external reset line.
      CLR     P5.5            ; De-activates the external reset line.

      SETB    P5.0            ; Make A16 of 128K RAM; system can use only
                              ; the high 128K of the RAM space.
```

```
                                 ; Note that A16 MUST BE SET, with no exceptions.

      CLR     P5.2               ; Disable EEPROM by setting the EEPROM
                                 ; clock to its low level.

      CLR     P5.1               ; Enable memory mapped input/output (MMIO)
                                 ; to enable the keypad and LCD panel.

; Add other initialization code here specific to the operation of interrupts,
; keypad, LCD, serial port, A/D, etc..

       SETB    EAL              ; Enable all interrupts
       CALL    SERINIT
       CALL    LCDINIT          ; initialize LCD display
                                ; this is possible because subroutine LCDINIT
                                ; has been made public and has been defined
                                ; as external code in module lcd.a51
       ;CALL   KPD
       MOV     R0,#2Fh         ; Clear a block of RAM (for example only).

CLR_RAM:
       MOV     @R0,#0
       DEC     R0
       CJNE    R0,#1Fh,CLR_RAM


       LJMP    MAIN_LOOP      ; Transfer control to MAIN_LOOP code in
                              ; module main.a51.

       END
```

Module #2: Main

```
; main.a51 (Module #2)
;
; The following "$" commands must be included in every module

$NOMOD51                      ; Omit assembler predefined registers.
$INCLUDE(reg515.inc)          ; Include 515/535 microcontroller definitions.

NAME   MAIN                   ; Optional parameter; if no name is provided,
                              ; the filename will be used by default.

PUBLIC MAIN_LOOP              ; Lets other modules access this section of code
                              ; from "public domain" utilizing the EXTRN command.

EXTRN  CODE    (LCDOUT, DISPMENU, KPD, ALERT, ALERTMENU)
;EXTRN CODE    (LOAD, SAVE)
                              ; Makes subroutines LCDINIT and LCDOUT in the
                              ; lcd.a51 module available to module main.a51 .

;*********************************************************************************

MAIN   SEGMENT CODE           ; Reserve RAM space for the generic code
                              ; segment, MAIN. The name segment name is referred
                              ; to by the following RSEG directive.


      RSEG    MAIN            ; Selects the MAIN code segement, and makes it
                              ; "active"  at this point in assembled code.
                              ; The selected segment remains "active" until
                              ; a different segment is specified.

       USING   0              ; Indicates to the assembler that register
                              ; bank 0 will be used, but does not actually
                              ; select register bank 0 .

; The module main.a51 should be used primarily to call subroutines in the
; various modules in the project, as opposed to incorporating detailed functions
; within module main.a51 (i.e., main.a51 should be relatively short).
```

```
MAIN_LOOP:

      CALL     DISPMENU
      CALL     KPD

      CJNE     A,#31h,SV
      CALL     ALERTMENU      ;ALERT

SV:
      CJNE     A,#32h,LD
      ;CALL    SAVE
LD:
       ;CALL   LOAD

      ;CALL LCDOUT            ; The call to LCDOUT is possible because
                              ; subroutine LCDOUT has been made public, and
                              ; has been defined as external code in
                              ; module lcd.a51 .

LOOP:
       JMP     LOOP           ; Infinite loop.

       END
```

Module #3: LCD Output Code

```
$NOMOD51                      ; omit assembler micro definitions
$Include(reg515.inc)          ; define 515 micro

Name   LCD


PUBLIC  LCDOUT, LCDINIT
EXTRN CODE (KPD)

LCD_DRV SEGMENT CODE

       RSEG    LCD_DRV        ; switch to this code segment

               USING   0      ; use register_bank 0

;*************************************************************************************

; definitions

escflag equ    psw.5          ; LCD equate
lcdcmd         equ     28h    ; value for P2 to select lcd command port

initdata:

      db    38h,08,01,06,0eh,80h,0

LCDOUT:
       MOV     R2,A           ; SAVE CHAR IN R2
       MOV     P2,#LCDCMD     ; POINT TO COMMAND PORT
       jnb     ESCflag,lcdnt5 ; skip if no ESC
       clr     escflag
       sjmp    reg0out        ; write directly to lcd reg 0

lcdnt5:
       ANL     A,#11100000B   ; SEE IF ANY OF UPPER 3 BITS SET
       JNZ     REG1OUT        ; IF YES, PRINT IT
       MOV     A,R2           ; RESTORE CHAR
       ANL     A,#11111000B   ; SEE IF CHAR IS < 7
       JZ      REG1OUT        ; IF LESS, A=0 SO PRINT USER DEF CHAR 0-7

       MOV     A,R2           ; SEE IF CONTROL CHAR
       CJNE    A,#0DH,LCNT1   ; IF NOT CR, SKIP
```

```
        MOVX    A,@R1           ; READ COMMAND PORT TO FIND CURSOR POS
        SETB    ACC.7           ; SET BIT 7 FOR DDRAM ADDR
        ANL     A,#11100000B    ; MOVE TO LEFT (ONLY VALID ON 2 LINE DISPL)
        MOV     R2,A
        SJMP    REG0OUT

LCNT1:
        CJNE    A,#0AH,LCNT2    ; IF NOT LF, SKIP
        MOVX    A,@R1           ; READ COMMAND PORT TO FIND CURSOR POS
        CPL     ACC.6           ; SWITCH LINE (ONLY VALID ON 2 LINE DISPL)
        SETB    ACC.7           ; SET BIT 7 FOR DDRAM ADDR
        MOV     R2,A
        SJMP    REG0OUT

LCNT2:
        CJNE    A,#1BH,LCNT3    ; IF NOT ESC, SKIP
        setb    ESCflag         ; indicate ESC received
        JMP     LCDEXIT

LCNT3:
        CJNE    A,#1AH,LCNT4    ; EXIT IF NOT CLEAR SCREEN
        MOV     R2,#1           ; CLEAR COMMAND
        SJMP    REG0OUT
                                ; OUTPUT THE CHAR IN R2 TO REG 1
REG1OUT:
        MOVX    A,@R1           ; READ LCD COMMAND PORT
        JB      ACC.7,REG1OUT   ; LOOP IF BUSY FLAG SET
        INC     P2              ; POINT TO LCD DATA PORT
        MOV     A,R2            ; RESTORE CHAR
        MOVX    @R1,A           ; OUTPUT IT

LCNT4:

        JMP   LCDEXIT


                                ; OUTPUT THE CHAR IN R2 TO REG 0
REG0OUT:
        MOVX    A,@R1           ; READ LCD COMMAND PORT
        JB      ACC.7,REG0OUT   ; LOOP IF BUSY FLAG SET
        MOV     A,R2            ; RESTORE CHAR
        MOVX    @R1,A           ; OUTPUT IT
        JMP     LCDEXIT

;
; LCDINIT: Init the LCD
;
LCDINIT:
        clr     ESCflag         ; indicate no esc found
        MOV     P2,#LCDCMD      ; POINT TO COMMAND PORT
        LCALL   DLAYA           ; 5MS DELAY
        LCALL   DLAYA           ; 5MS DELAY
        LCALL   DLAYA           ; 5MS DELAY
        LCALL   DLAYA           ; 5MS DELAY
        MOV     A,#30H
        MOVX    @R1,A           ; OUT TO LCD COMMAND PORT
        LCALL   DLAYA           ; 5MS DELAY
        MOVX    @R1,A           ; OUT TO LCD COMMAND PORT
        LCALL   DLAYA           ; 5MS DELAY
        MOVX    @R1,A           ; OUT TO LCD COMMAND PORT

        MOV     DPTR,#INITDATA  ; POINT TO INIT DATA
                                ; the last command should take no more than 40 uS.
        mov     b,#80           ; for timeout of 80*3 * (12/clock)

LCDINIT2:
        movx    a,@r1           ; read lcd command port
        jnb     acc.7,LCDINIT1  ; exit if not busy
        djnz    b,LCDINIT2      ; loop till timeout
        sjmp    lcdexit         ; exit if timeout
```

```
LCDINIT1:
        MOVX    A,@R1           ; READ LCD COMMAND PORT
        JB      ACC.7,LCDINIT1 ; LOOP IF BUSY FLAG SET
        CLR     A
        MOVC    A,@A+DPTR       ; GET BYTE FROM INIT TABLE
        JZ      LCDEXIT         ; EXIT IF 0
        INC     DPTR            ; POINT TO NEXT BYTE
        MOVX    @R1,A           ; OUTPUT BYTE
        SJMP    LCDINIT1        ; LOOP

LCDEXIT:
        RET




;
; MISCELLANEOUS DELAYS added to keep the LCD from scrolling
; when the buttons are held down

DLAYA:
        PUSH    ACC
        MOV     A,#100
        AJMP    DLAYA2

DLAYB:
        PUSH    ACC
        MOV     A,#128
        AJMP    DLAYA2

DLAYC:
        PUSH    ACC
        MOV     A,#255
        AJMP    DLAYA2

dlayd:
        PUSH    ACC
        MOV     A,#8


DLAYA2:
        PUSH    ACC
        MOV     A,#0FFH
DLAYA1:
        MOV     A,#0FFH
        DJNZ    ACC,$          ; LEVEL 3 LOOP
        POP     ACC
        DJNZ    ACC,DLAYA2      ; LEVEL 1 LOOP

        POP     ACC
        RET

        END
```

Module #4: Main Menu

```
;The following "$" commands must be included in every module

$NOMOD51                        ; Omit assembler predefined registers.
$INCLUDE(reg515.inc)            ; Include 515/535 microcontroller definitions.

NAME    TOPMENU                 ; Optional parameter; if no name is provided,
                                ; the filename will be used by default.

PUBLIC  DISPMENU                ; Lets other modules access this section of code
                                ; from "public domain" utilizing the EXTRN command.

                                ; Makes subroutines LCDINIT and LCDOUT in the
                                ; lcd.a51 module available to module main.a51.

EXTRN CODE (LCDOUT)

;********************************************************************************

TOPMENU   SEGMENT CODE          ; Reserve RAM space for the generic code
                                ; segment, TOPMENU. The name segment name is referred
                                ; to by the following RSEG directive.


    RSEG   TOPMENU              ; Selects the MAIN code segement, and makes it
                                ; "active"  at this point in assembled code.
                                ; The selected segment remains "active" until
                                ; a different segment is specified.

    USING   0                   ; Indicates to the assembler that register
                                ; bank 0 will be used, but does not actually
                                ; select register bank 0 .

; The module main.a51 should be used primarily to call subroutines in the
; various modules in the project, as opposed to incorporating detailed functions
; within module main.a51 (i.e., main.a51 should be relatively short).

DISPMENU:

    ;PUT MAIN MENU CODE HERE FOR DISPLAYING ALERT, SAVE, LOAD

        MOV     DPTR,#LINE1     ; initialize pointer

DISPLOOP:

        CLR     A
        MOVC    A,@A+DPTR
        JZ      NEXTLINE
        CALL    LCDOUT
        INC     DPTR
        SJMP    DISPLOOP

LOOPEXIT:
        RET

NEXTLINE:
        MOV     DPTR,#LINE2
        SJMP    DISPLOOP

LINE1:
        db      "(1) Alert  (2) Save        (3) Load     ",0
LINE2:
        RET

    END
```

Module #5: Keypad Code

```
;**************************************************************************
;
; KEYPAD subroutine: waits for key pressed and returns it in ACC.
; (MODULE #5)
;
;**************************************************************************
$NOMOD51                        ; omit assembler micro definitions
$Include(reg515.inc)            ; define 515 micro

Name    KEYPAD
PUBLIC  KPD

EXTRN CODE (LCDOUT)
KEYPAD  SEGMENT CODE
```

```
        RSEG    KEYPAD          ; switch to this code segment

        USING   0               ; use register_bank 0


; Dempsey Note:
; This code was provided by EMAC
; It is not an efficient way to use keypad
; Normally must do other main code processing
;
; local definitions

KEYSEL  EQU      38H            ; KEYPAD PORT


KPD:
        JNB     IE1,KPD         ; LOOP TILL KEY PRESSED
        CLR     IE1             ; clear for next transition

        PUSH    DPH
        PUSH    DPL             ; SAVE DPTR
        MOV     DPTR,#KEYTABL   ; POINT TO TRANSLATE TABLE
        MOV     P2,#KEYSEL      ; POINT TO KEYPAD PORT
        MOVX    A,@R1           ; GET KEY FROM PORT
        ANL     A,#00011111B    ; ONLY 5 BITS
        MOVC    A,@A+DPTR       ; TRANSLATE TO KEY FROM TABLE (ASCII)
        POP     DPL
        POP     DPH
        RET

KEYTABL: DB '123C456D789EA0BF'

        END
```

Module #6: Alert Mode Code

```
; ALERT MENU (MODULE #6)

$NOMOD51                        ; omit assembler micro definitions
$Include(reg515.inc)            ; define 515 micro

Name    ALERT_MENU

PUBLIC ALERTMENU

EXTRN CODE (KPD,LCDOUT,ALERT,LCDINIT,MAIN_LOOP)

ALRT    SEGMENT CODE

       RSEG     ALRT            ; switch to this code segment

        USING   0               ; use register_bank 0

;*****************************************************************************************
******
ALERTMENU:

        MOV     A,#1AH          ;CLEARS LCD SCREEN
        CALL    LCDOUT          ;
        MOV     B,#4
        CLR     A
        MOV     DPTR,#ASCII     ;MOVES DATA TABLE INTO DPTR
        MOVC    A,@A+DPTR       ;MOVES DPTR INTO ACC
        MOV     R0,#15          ;MOVES 15 INTO REGISTER 0

ALERTMENU2:

        JZ      LOOPEXIT
        CALL    LCDOUT
        INC     DPTR            ;INCREMENTS THE MEMORY LOCATION IN DPTR TO
                                ;THE NEXT LETTER TO DISPLAY
        MOVX    A,@DPTR

ALERT_LOOP:

        DJNZ    R0,ALERTMENU2  ;LOOPS UNTIL R0 IS 0 IN ORDER TO DISPLAY ALL THE
                                ;TEXT SAVED IN ACC

        MOV     @R1,A           ;STORE ID# IN R0
        CALL    KPD

        DJNZ    B,KPD_LOOP      ;WHEN THE LAST ITEM IS REACHED (B=0)
        CJNE    A,#45H,RETURN  ;DOUBLE CHECK KEYPAD FOR E BEFORE DISPLAYING
                                ;EACH ITEM AGAIN
        SJMP    ENTER           ;OTHERWISE WAIT FOR KEY

RETURN:

        CJNE    A,#42H,ALERTMENU       ;DOUBLE CHECKS THE KEYPAD BEFORE RETURNING
        SJMP    BACK                    ;TO THE TOP OF THE LIST (B = BACK TO MAIN MENU)

        ;WAIT FOR KEYPAD ENTRY, ENTER OR NEXT ITEM
        ;IF ENTER: ADD 1 TO A (PUSH A FIRST TO REOPEN SAVED VALUE)
        ;IF NEXT: ADD 2 TO A TO MOVE ONTO THE NEXT ITEM NAME

KPD_LOOP:

        CJNE    A,#42H,COMMANDS        ;B = BACK TO MAIN MENU
        SJMP    BACK

BACK:
                                ;IF B IS PRESSED, THE LCD WILL
        MOV     A,#1AH          ;CLEAR THE SCREEN AND
        CALL    LCDOUT          ;
        JMP     MAIN_LOOP       ;DISPLAY THE MAIN MENU AGAIN

COMMANDS:

        CJNE    A,#46H,ENTER   ;F = NEXT
        SJMP    NEXT

ENTER:

        CJNE    A,#45H,WAIT    ;E = ENTER

        MOV     A,#0AH          ;SKIPS TO THE NEXT LINE ON THE LCD
        CALL    LCDOUT
        MOV     DPTR,#TRANS    ;MOVES "TRANSMITTING..." INTO DPTR
        MOV     A,#0DH          ;MOVES CURSOR TO BEGINNING OF LINE 2
        CALL    LCDOUT

TRANSMITTING:                   ;THIS CODE DISPLAYS "Transmitting..."
                                ;UNDER THE ITEM NAME WHEN THE ALERT
        CLR     A               ;BUTTON IS PRESSED
        MOVC    A,@A+DPTR
        JZ      TRANSEXIT
        CALL    LCDOUT
        INC     DPTR
        CALL    TRANSMITTING   ;LOOP UNTIL ALL LETTERS ARE DISPLAYED

        CALL    ALERT
        MOV     A,#1AH          ;CLEAR LCD
        CALL    LCDOUT          ;

        JMP     MAIN_LOOP       ;AFTER TRANSMISSION, THE LCD WILL
                                ;DIPLAY THE MAIN MENU AGAIN
TRANSEXIT:
        RET

NEXT:
        MOV     A,#1AH          ;CLEARS LCD FOR NEXT
        CALL    LCDOUT          ;ITEM TO BE DISPLAYED
        INC     DPTR            ;MOVES THE DPTR TO THE NEXT LINE ON THE ASCII
                                ;TABLE
        MOVX    A,@DPTR         ;STORES THE NEXT LINE IN ACC
        MOV     R0,#15
        JNZ     ALERTMENU2
        SJMP    ALERTMENU

WAIT:

        CALL    KPD
        JMP     KPD_LOOP

LOOPEXIT:

        RET

ASCII:

    ;PUT ASCII TABLE HERE
        ;first item name(in ASCII),ID#
        ;DB     Ph,Hh,Oh,Nh,Eh,sph,sph,sph,sph,sph,sph,sph,sph,sph,sph,IDh
        ;second item
        ;db       ,      ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,ID2h


    db 50h,48h,4Fh,4Eh,45h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,33h    ;PHONE
    db 52h,45h,4Dh,4Fh,54h,45h,20h,20h,20h,20h,20h,20h,20h,20h,20h,55h    ;REMOTE
    db 4Bh,45h,59h,53h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,0FFh   ;KEYS
    db 48h,45h,41h,44h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,39h       ;HEAD

TRANS:

        DB      "Transmitting....",0          ;Displays "Transmitting...." when alert
                                                   ;button is pressed
    END
```

Module #7: Serial Port Output Code

```
; The following "$" commands must be included in every module
$NOMOD51                        ; Omit assembler predefined registers.
$INCLUDE(reg515.inc)            ; Include 515/535 microcontroller definitions.

NAME    SERIAL                  ; Optional parameter; if no name is provided,
                                ; the filename will be used by default.

PUBLIC  SERINIT, SEROUT, ALERT; Lets other modules access this section of code
                                ; from "public domain" utilizing the EXTRN command.

EXTRN   CODE    (LCDOUT)
                                ; Makes subroutines LCDINIT and LCDOUT in the
                                ; lcd.a51 module available to module main.a51 .

MR1BDAT EQU     00010011B
MR2BDAT EQU     00000111B       ;Set stop bit length = 1
                                ;Put registers in memory spaces


ACR     EQU 04H                 ;Auxiliary Control Register

MR1B    EQU     08H             ;Mode Register B (1-receiver 2-transmitter)
SRB     EQU     09H             ;Channel B Status Register
CSRB    EQU     09H             ;Clock Select Register B
CRB     EQU     0AH             ;Channel A Command Register
THRB    EQU     0BH             ;Tx holding register
```

```
;**********************************************************************************

SERIAL  SEGMENT CODE          ; Reserve RAM space for the generic code
                              ; segment, MAIN. The name segment name is referred
                              ; to by the following RSEG directive.


      RSEG    SERIAL          ; Selects the MAIN code segement, and makes it
                              ; "active"  at this point in assembled code.
                              ; The selected segment remains "active" until
                              ; a different segment is specified.

      USING   0               ; Indicates to the assembler that register
                              ; bank 0 will be used, but does not actually
                              ; select register bank 0 .

SERINIT:

       MOV     A,#01010000B   ;Do from this command, down to 00010000

COM_B_RESET:

       MOV     P2,#CRB
       MOVX    @R1,A
       ADD     A,#-16         ;Subtracts 1 from the upper nibble; loop until = 0000
                              ;0101=Reset channel A interrupt
                              ;0100=Reset error status. Clears channel A received break,
                              ;     parity error, and overrun error bits.
                              ;0011=Reset transmitter.
                              ;0010=Reset receiver.
                              ;0001=Reset MR pointer.  Points MR pointer to MR1.
                              ;0000=No command, exit loop.
       JNZ     COM_B_RESET    ;If the first 4 bits don't equal 0000 jump back to
COM_B_RESET.

COM_B_SETUP:

       MOV     P2,#MR1B       ;Points Mode Register 1B to Port 2
       MOV     A,#MR1BDAT     ;Initializes MR1B receiver first in order to
       MOVX    @R1,A          ;initialize MR2B next for transmission.

       MOV     A,#MR2BDAT     ;Stores mode register parameters in acc
       MOVX    @R1,A          ;Move MR2BDAT into MR2B

       MOV     P2,#ACR        ;
       MOV     A,#80H         ;Points 80H into ACR in Port 2
       MOVX    @R1,A          ;Baud Rate Generator Set Select = 1

       MOV     P2,#CSRB       ;Set BAUD rate to 1.8kHz
       MOV     A,#01000100B
       MOVX    @R1,A

       MOV     A,#00000101B   ;Points data bits for CRB into Acc
       MOV     P2,#CRB        ;Points CRB into Port 2
       MOVX    @R1,A          ;Points data bits into CRB at Port 2
       RET                    ;Enables the COM B - transmitter and reciever

ALERT:

       MOV     B,#20          ;WILL TRANSMIT THE NUMBER OF TIMES OF THE
                              ;NUMBER STORED IN B

SEROUT:

       MOV     A,@R1
       ;POP    ACC            ;PUTS THE ID# INTO THE ACC
       ;MOV    A,#37H         ;Test data to be sent to the transmitter
```

```
SEROUTB:

        MOV     P2,#SRB
        PUSH    ACC             ; SAVE CHAR for later use

SOUTB1:

        MOVX    A,@R1           ;Point external Port 2 to Acc
        JNB     ACC.2,SOUTB1    ;Loop until SRB-bit 2 (TXrdy) is ready to transmit
        POP     ACC
        MOV     P2,#THRB        ;Send out the serial bit stored
        MOVX    @R1,A

        DJNZ    B,SEROUT
        RET

END
```

II. VHDL Code For The Remote Unit

```
-- Re_enable has been commented out because it is not a necessary signal for the code.
-- The purpose of re_enable was to have a second reset so that rst_n could be a main reset
-- and re_enable could be a user reset.

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;

entity shftreg is
port
        (
                clk,rin,enable  : in std_logic;                 -- 2KHz clk input
                rst_n           : in std_logic;
                TTL_out         : out unsigned(1 downto 0)
                --Q             : buffer std_logic_vector(8 downto 0);
        );
end shftreg;

architecture smy of shftreg is
signal IQ               : std_logic_vector(8 downto 0);
signal TTL_out_w        : unsigned(1 downto 0);
signal clk_w            : unsigned(3 downto 0);
signal sampler          : unsigned(3 downto 0);
signal test_start       : unsigned(3 downto 0);
signal bit_counter      : unsigned(3 downto 0);
signal latch_out        : std_logic;
signal retest_start     : std_logic;
signal sample           : std_logic;
signal load             : std_logic;
signal inc              : std_logic;
signal start_bit        : std_logic;
signal clr              : std_logic;
signal check_compare    : std_logic;
signal slow_clk         : std_logic;

begin
```

```
-- Clock Generation Circuitry

        process(rst_n,clk)
        begin
                if (rst_n = '0') then
                        clk_w <= (others => '0');

                elsif rising_edge(clk) then             -- produces a 16 times slower clock for the
                        clk_w <= clk_w + 1;             -- shift register, compare, TTL output code
                end if;
        end process;
        slow_clk <= clk_w(3);

-- Start Bit Detection

        process(rst_n,clr,clk)
        begin
                if rst_n = '0' then
                        start_bit <= '1';

                elsif rising_edge(clk) then
                        if (rin = '0') then             -- latches start_bit low when input drops low
                                start_bit <= '0';

                        elsif (clr = '0') then          -- resets latch when clr = 0
                                start_bit <= '1';

--                      elsif (re_enable = '0') then
--                              start_bit <= '1';
                        end if;
                end if;
        end process;

-- UART Circuitry

        process(clk, rst_n)
        begin
                if (rst_n = '0') then
                        test_start <= (others => '0');
                        sampler <= "0001";
                        sample   <= '0';
                        load     <= '1';
                        inc      <= '0';
                        bit_counter <= "0110";          -- set bit_counter to 5 to count all 10 bits
                        check_compare <= '0';
                        clr <= '1';
                        retest_start <= '1';

                elsif rising_edge(clk) then

                        clr <= '1';

--                      if (re_enable ='0') then
--                              test_start <= (others => '0');
--                              sampler <= "0001";
--                              sample   <= '0';
--                          load        <= '1';
--                          inc         <= '0';
--                          bit_counter <= "0110";
--                          check_compare <= '0';
--                          clr <= '1';
--                          retest_start <= '1';
--                  end if;

                    if start_bit = '0' then
                            if (retest_start = '1') then
                                    if (load = '1') then
                                            test_start <= "1000";
                                            load <= '0';

                                    elsif (test_start = "1111" and rin = '0') then
                                            sample  <= '1';
                                            retest_start <= '0';

                                    elsif (test_start = "1111" and rin /= '0') then
                                            --RESET the start bit latch above
                                            clr <= '0';
                                            load <= '1';

                                    else
                                            test_start <= test_start + 1;
                                    end if;
                            end if;

                            if (sample = '1') then              -- count 16 times then sample bit
                                    sampler <= sampler + 1;
                            end if;

                            if (sampler = "0100") then          -- limits check_compare clk period
                                    check_compare <= '0';
                            end if;

                            if (sampler = "1111") then          -- counts 9 bits
                                    bit_counter <= bit_counter + 1;
                            end if;

                            if (bit_counter = "1111" and rin = '1') then
                                                                -- all bits have been counted and
                                                                -- stop bit = 1
                                    check_compare <= '1';       -- allows for comparing in shift
                                                                -- register
                                    -- RESETS
                                    clr  <= '0';                -- resets the start-bit latch
                                    load <= '1';                -- resets test_start
                                    sample <= '0';              -- stops sampler counting
                                    retest_start <= '1';        -- activates the start-bit sampler
                                    bit_counter <= "0110";      -- resets bit_counter

                            elsif (bit_counter = "1111" and rin /= '1') then
                                                                -- all bits have been counted
                                                                -- and stop-bit not = 1
                                    -- RESETS                    -- same resets as above
```

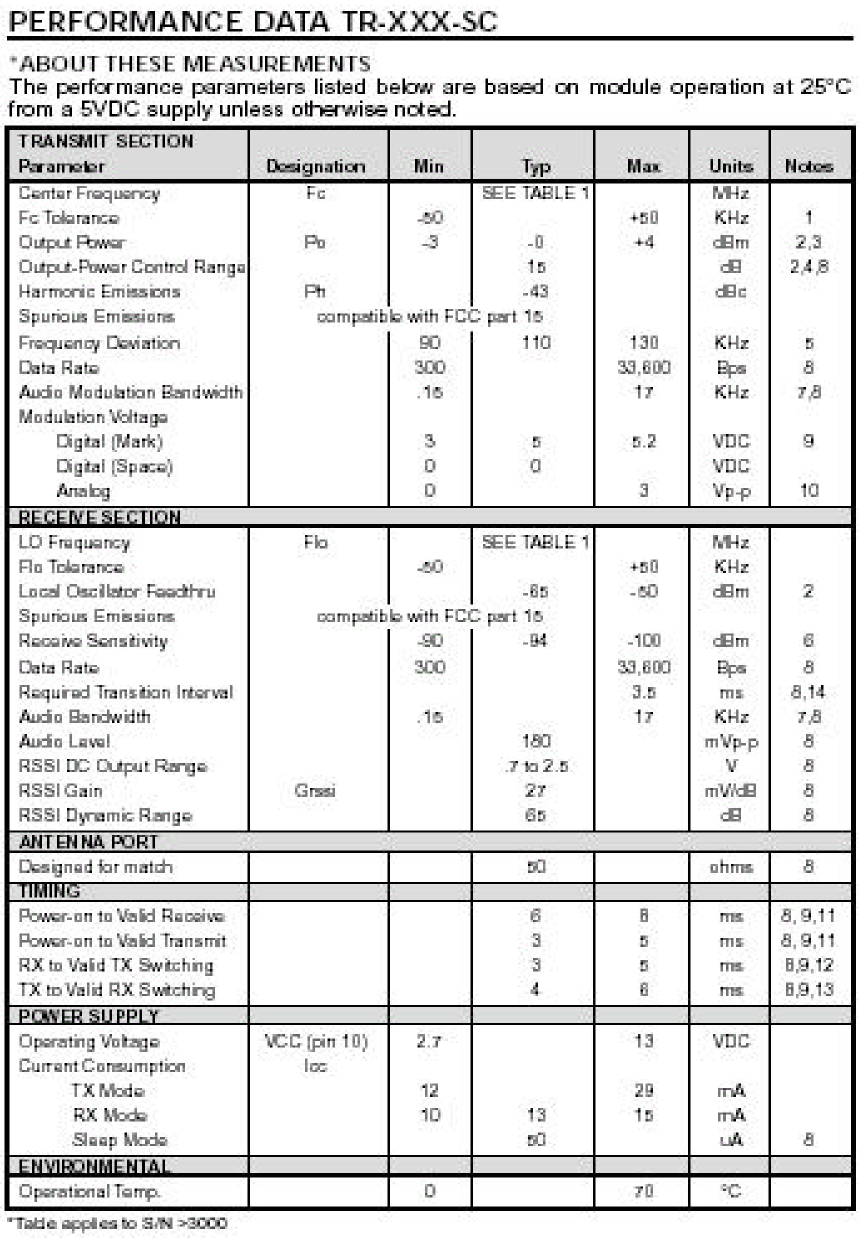
```
                                        clr  <= '0';
                                        load <= '1';
                                        sample <= '0';
                                        bit_counter <= "0110";
                                        retest_start <= '1';
                                end if;
                        end if;
                end if;
        end process;

-- Shift Register Circuitry

        process(slow_clk,rst_n)
        begin
        if rst_n = '0' then
                IQ <= (others => '0');
        elsif rising_edge(slow_clk) then
                        case enable is
                                when '0' => null;
                                when '1' => IQ <= IQ(7 downto 0) & rin;      -- shifts the bits through
                                                                             -- the register(MSB first)
                                when others => null;
                        end case;

--              if (re_enable = '0') then
--                      IQ <= (others => '0');                       -- resets register if re_enable = 0
--              end if;

        end if;
        Q <= IQ;

        end process;

-- Latch Circuitry\Compare Circuitry

        process(rst_n,slow_clk)
        begin
                if rst_n = '0' then
                        latch_out <= '0';

                elsif rising_edge(slow_clk) then
--                      if (re_enable = '0') then
--                              latch_out <= '0';              -- resets latch if re_enable = 0

                        if (IQ = "000000001" and check_compare = '1') then
                                                               -- compares input to preset ID code
                                latch_out <= '1';              -- holds TTL output high until reset by user
                        elsif (IQ = “000000001” and check_compare = ‘1’) then
                                latch_out <= ‘0’;              --turns audible tone off when user presses
                                                               --button on the base unit
                        end if;
                end if;
        end process;

-- TTL Output Generation Circuitry
```

```
        process(rst_n, TTL_out_w, clk) is
        begin
                if(rst_n = '0') then
                        TTL_out_w <= (others => '0');

                elsif rising_edge(clk) then

                        if(latch_out = '1') then
                                TTL_out_w <= TTL_out_w + 1;     -- creates TTL output wave to
                        end if;                                 -- speaker for tone generation
                end if;
        end process;

        TTL_out <= TTL_out_w;

end smy;
```

Appendix B

Data Sheets and Pin Assignments

Transceiver Data Sheet



PERFORMANCE DATA TR-XXX-SC

*ABOUT THESE MEASUREMENTS
The performance parameters listed below are based on module operation at 25°C from a 5VDC supply unless otherwise noted.

TRANSMIT SECTION Parameter	Designation	Min	Typ	Max	Units	Notes
Center Frequency	Fc		SEE TABLE 1		MHz	
Fc Tolerance		-50		+50	KHz	1
Output Power	Po	-3	-0	+4	dBm	2,3
Output-Power Control Range			15		dB	2,4,8
Harmonic Emissions	Ph		-43		dBc	
Spurious Emissions	compatible with FCC part 15					
Frequency Deviation		90	110	130	KHz	5
Data Rate		300		33,600	Bps	8
Audio Modulation Bandwidth		.15		17	KHz	7,8
Modulation Voltage						
Digital (Mark)		3	5	5.2	VDC	9
Digital (Space)		0	0		VDC	
Analog		0		3	Vp-p	10
RECEIVE SECTION						
LO Frequency	Flo		SEE TABLE 1		MHz	
Flo Tolerance		-50		+50	KHz	
Local Oscillator Feedthru			-65	-50	dBm	2
Spurious Emissions	compatible with FCC part 15					
Receive Sensitivity		-90	-94	-100	dBm	6
Data Rate		300		33,600	Bps	8
Required Transition Interval				3.5	ms	8,14
Audio Bandwidth		.15		17	KHz	7,8
Audio Level			180		mVp-p	8
RSSI DC Output Range			.7 to 2.5		V	8
RSSI Gain	Grssi		27		mV/dB	8
RSSI Dynamic Range			65		dB	8
ANTENNA PORT						
Designed for match			50		ohms	8
TIMING						
Power-on to Valid Receive			6	8	ms	8, 9,11
Power-on to Valid Transmit			3	5	ms	8, 9,11
RX to Valid TX Switching			3	5	ms	8,9,12
TX to Valid RX Switching			4	6	ms	8,9,13
POWER SUPPLY						
Operating Voltage	VCC (pin 10)	2.7		13	VDC	
Current Consumption	Icc					
TX Mode		12		29	mA	
RX Mode		10	13	15	mA	
Sleep Mode			50		uA	8
ENVIRONMENTAL						
Operational Temp.		0		70	°C	

*Table applies to S/N >3000

Transceiver Pin Assignments

PIN DESCRIPTION

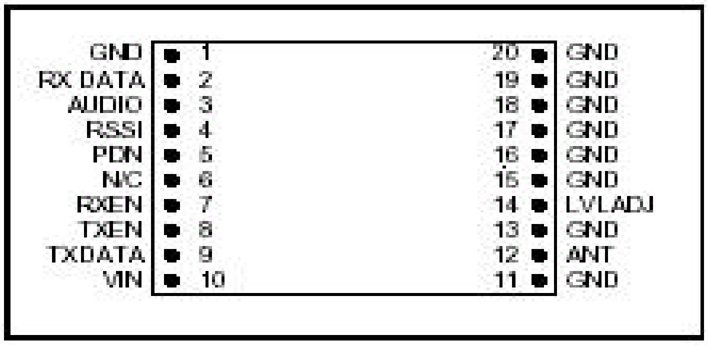


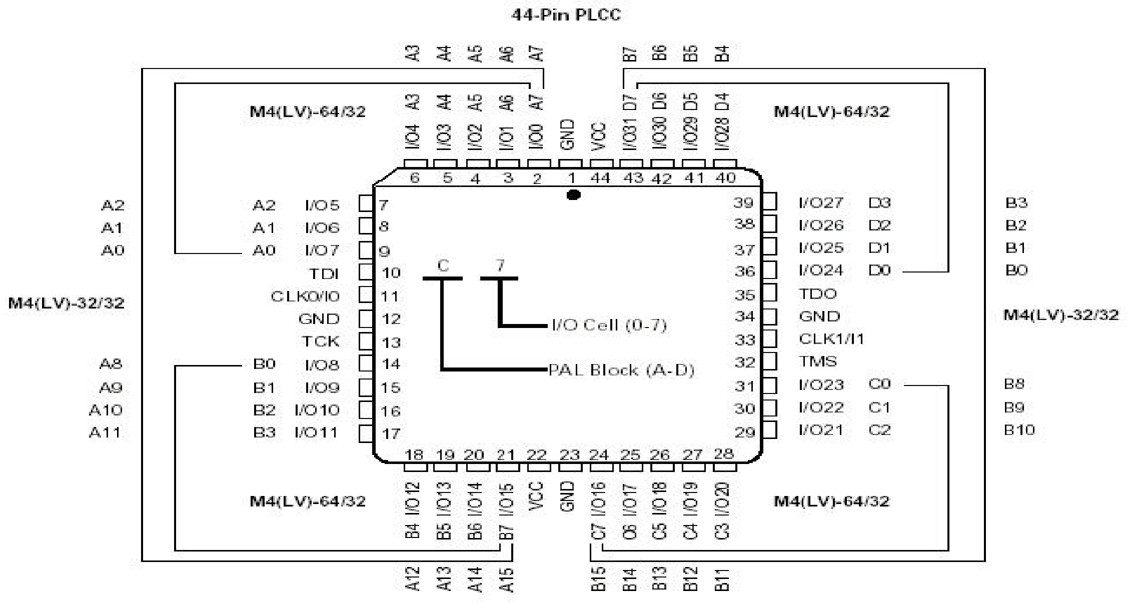
Figure 5: SC Series Pinouts (viewed looking down on top cover)

PIN#	Pin Title	Description
1,11,13 15-20	Ground	Module Grounds Tie to Common Groundplane
2	RXDATA	Recovered Data Output
3	AUDIO	Recovered Analog Output
4	RSSI	Received Signal Strength Indicator
5	PDN	Logic Low Powers Down The Transceiver
6	N/C	Not Implemented Do Not Connect
7	RXEN	Receiver Enable Pin Active High Pull Low When in TX
8	TXEN	Transmitter Enable Pin Active High Pull Low When in RX
9	TXDATA	Analog or Digital Content to be Transmitted
10	VIN	2.7-13VDC Supply
12	ANT	50Ω Antenna Port TX/RX Switch Inside Module
14	LVLADJ	Open for Maximum TX Power Insert Resistor to Reduce by up to 15dB

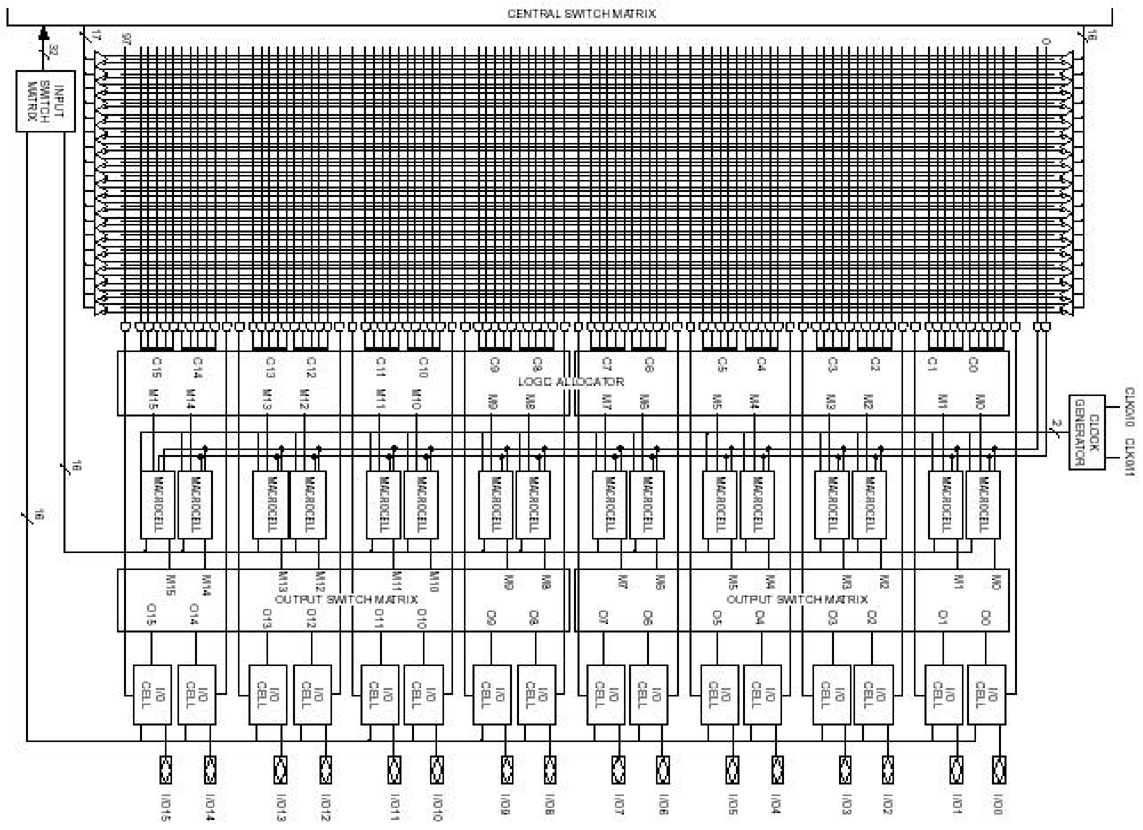
CPLD Pin Assignments

44-PIN PLCC CONNECTION DIAGRAM (M4(LV)-32/32 AND M4(LV)-64/32)

Top View



MACH 4 64/32 Memory Cell Space (Size Constraints for the MACH 4)



Product Data Sheet

Dimensions	**L X W X H**
Base Unit:	6” X 4” X 3”
Remote Unit	.5” X .5” X .125”

Number of remote units: 8

Power Supply	**Min**	**Typ**	**Max**	**Unit**
Operating Voltage				
Base Unit:	7		15	Vdc
Remote Unit:	2.7	3.3	13	Vdc
Current Consumption				
Base Unit:		45	85	mA
Remote Unit:	10	13	29	mA
Power:				
Base Unit:	.315		1.3	W
Remote Unit:	27	43	377	mW
Operational Temp:	0		70	^{o}C

The values on this data sheet were estimated due to the fact that nothing has actually been built and tested in the lab yet. The dimensions for the base unit were based off of the dimensions of a Micro Pac 8051 microcontroller board and an LCD screen. The remote unit dimensions were based off another remote unit device that was found during a patent search. The power supply ratings were based off of the microcontroller board for the base unit and the receiver for the remote unit.

Appendix C
Product Manufacturing Pricing

Remote unit:

Processor: AT tiny 12L-4sc: 8-pin surface mount/ 4MHz/ in-system programmable --- $1.46

Receiver: RXM-433-LC-S-ND Surface mount 433MHz receiver --- $9.85

Speaker: P9902 TR-ND: 8.5mm x 8.5mm/ 92dB/ surface mount/ 2.5KHz→2.7KHz range --- $2.234

Custom-made Casing: Estimated at $1.50

Antenna: part of board.

Battery: P189-ND Panasonic CR2032: 3V/ 220mAh/ 20mm --- $0.21675

Battery Holder: BA2032 SM-Bulk-ND: Surface mount coin 20mm battery holder --- $0.35

Audible Alert Off Button: P8006S Momentary switch --- $0.099

PCB: $.65 / sq in. = 1 x 1 in. = $0.65

HCP = $16.36
LCP = .1*TPC = $1.82
TPC = HCP/.9 = $18.18

Base Unit:

Processor: ATMEL AT 90S1200-4YC --- $2.05

LCD: Vacuum fluorescent display/ 2x20 lines --- $4.95

Keypad: $2.00

Custom-made Casing: Estimated at: $2.5 - $3

Transmitter: TXM-433-LC-ND surface mount 433MHz transmitter --- $4.90

Power Supply: Diamond 35-6-500D: 6V/ 500mA --- $1.53/per unit

Antenna: $1

PCB 2 x 2 in * $0.65 = $2.60

HCP = $21.53
LCP = .1*TPC = $2.39
TPC = HCP/.9 = $23.92

<u>Total cost of package: $96.64 (with 4 remote units)</u>

This pice is very high due to the expensive transmitter, receiver, LCD, and keypad. If the product were actually produced by a major company, an ASIC chip with a transceiver built in would be used. This would lower the price of each remote and base unit $10. A major company would also have better connections, so the LCD and keypad would be found at a much cheaper price. I estimate that the cost of the total product would be approximately $60 cheaper if a major production company were building it.

Appendix D
Other Works

Patent Number WO0217265:

A remote control locator system (10) that can be retro-fitted to any existing remote control device in a straightforward manner. The remote control locator system (10) comprises a sending unit (20) and a receiving unit (30, 130). The sending unit (20) includes a transmitter residing (28) in a sending unit housing (26) and an activation mechanism (25) coupled to the transmitter (28) to send a locator signal when the activation mechanism (25) is activated by a user. The receiving unit (30, 130) includes a receiver (46) residing in a receiving unit housing (38) to receive the locator signal and to emit an audible sound when the receiver (46) receives the locator signal.

Sharper Image Item Finder: $50

Key Ringer Item Finder: $30

Standards

Code of Federal Regulations Par 15-Title 47: Radio Frequency Demodulation.

UART standards for packing and unpacking serial bit streams.

Appendix D

Schedule of Tasks

January

Week 4:	Finish all assignments for EE 419 and 451.

February

Week 1:	Begin hardware design for remote the remote units and work on the web page .
Week 2:	Begin simulation of hardware, review microcontroller code, and work on the web page.
Week 3:	Debug and test simulations and review microcontroller code.
Week 4:	Finish all simulation and begin building in lab, review microcontroller code, and work on web page.

March

Week 1:	Build the hardware for the remote units and test.
Week 2:	Continue testing of hardware and reviewing microcontroller language.
Week 3:	Finish testing the remote units and finish review of microcontroller language.
Week 4:	Begin writing the microcontroller software and work on the web page.

April

Week 1:	Write main menu and LCD software.
Week 2:	Debug any problems with written software, and write, the modes different modes of operation software.
Week 3:	Debug all software and begin the implementation of the combination of the hardware with the software.
Week 4:	Test the software and hardware combination.

May

Week 1:	Write the final project report and the oral presentation and finish the web page.

Appendix E
References

1. Dempsey, Dr. Gary. EE 451 Lab Instructor. Illinois: Bradley, 2002.

2. Huggins, Dr. Brian. Senior Project Advisor. Illinois: Bradley, 2002-2003.

3. Key Ringer. www.keyringer.com

4. Lattice Semiconductor Corporation. www.latticesemiconductor.com, M4(LV) Data Sheets.

5. Philips Semiconductors. *SC26C92 Dual Universal Asynchronous Receiver/Transmitter (DUART) Data Sheet.* 1997.

6. Quatech. www.quatech.com, *Asynchronous Serial Communication Overview*.

7. Sánchez, José. Senior Project Advisor. Illinois: Bradley, 2003.

8. Sedra, Adel S., and Kenneth C. Smith. Microelectronic Circuits. New York: Oxford, 1998.

9. Sharper Image. www.sharperimage.com, Item Finder.