Application Note AN2xxx

Easy Morse Coder

Authors: S. G. Sriharsha
H. N. Naveen
Associated Project: Yes
Associated Part Family: CY8C27xxx
PSoC Designer Version: 4.1
Associated Application Notes AN2124, AN2xxx

Abstract
Amateur radio operators simply called as HAM radio operators use morse code for communicating with other radio operators. Although morse code is not being used extensively in modern day communication, it is used during emergency messaging and by amateur radio operators during contests and general two way contacts.

This application note explains how an amateur radio operator can generate Morse code easily using a standard computer PS/2 keyboard, a PSoC device and few external components. This document is an extension of the application note AN2xxxx, “Interfacing PS/2 Keyboard using SPI Slave”.

Introduction
In 1836, Samuel Morse demonstrated the ability of a telegraph system to transmit information over wires. The information was sent as a series of electrical signals. Short signals are referred to as dits (represented as dots). Long signals are referred to as dahs (represented as dashes). With the advent of radio communications, an international version of Morse code became widely used.

In amateur radio "code" generally refers to the International Morse Code. Adopted world wide in the early 1900's for all commercial international communication, amateurs too adopted its use.

Since Morse relies on only an (on-off keyed) radio signal, it requires less complex equipment than other forms of radio communication, and it can be used in very high noise/low signal environments. It also requires less bandwidth than voice communications, typically 100-150 Hz.

Figure 1 below shows the typical Morse key used to send Morse code.

Figure 1. Manual Morse key

Morse Code Representation
There are two "symbols" used to represent letters, numbers and punctuation marks, called dots and dashes or dits and dahs. The length of the dit determines the speed at which the message is sent, and is used as the timing reference.

Figure 2. Below illustrates the exact timing durations of dots and dashes.

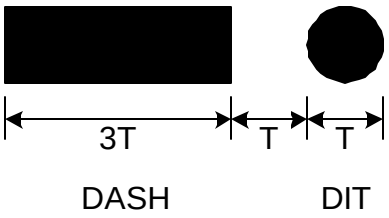


Figure 2. Morse code symbols

In full-speed Morse, a dah is conventionally 3 times as long as a dit shown in figure 2. Spacing between dits and dahs in a character is the length of one dit. Spacing between letters in a word is the length of a dah (3 dits). Spacing between words is 7 dits.

PSoC and PS/2 Keyboard

In this example the PSoC device is interfaced with the PS/2 keyboard. An LCD is connected to the PSoC in-order to display the key presses. In this example, only the Device-to-Host mode of communication is shown.

The PSoC device is configured for a SPI slave and a PWM8. The SPI slave is used for communicating with the keyboard and PWM8 is used for controlling the contrast of the LCD.

For complete information on interfacing a PSoC and a PS/2 keyboard, please refer to the Cypress application note “AN2XXX, Interfacing PS/2 Keyboard using SPI Slave”.

The schematic of the easy Morse coder example is provided as Figure.5.

PSOC User Modules

The figure 3 shows the PSoC user module placement of the keyboard example.

The following user modules are used in the easy Morse coder example:

- SPI Slave
- PWM8 - LCD Contrast
- PWM16 - Morse Code Output
- LCD Tool box

SPI Slave

The SPIS User Module is a Serial Peripheral Interconnect Slave. It performs full duplex synchronous 8-bit data transfers. SCLK phase, SCLK polarity, and LSB First can be specified to accommodate most SPI protocols.

The SPIS is used to communicate with the keyboard.

PWM8

PWM8 is an 8-bit pulse width modulator with programmable period and pulse width. The clock and enable can be selected from several sources.

The PWM8 controls the LCD contrast.

PWM16

PWM16 is an 16-bit pulse width modulator with programmable period and pulse width. The clock and enable can be selected from several sources.

The Morse code is generated using PWM16 user module.

LCD Tool box

The LCD Tool Box User Module is a set of library routines that writes text strings and formatted numbers to a common two- or four-line LCD module.

Port 1 of PSoC is used to drive the LCD in 4-bit interface mode to limit the number of I/O pins required.

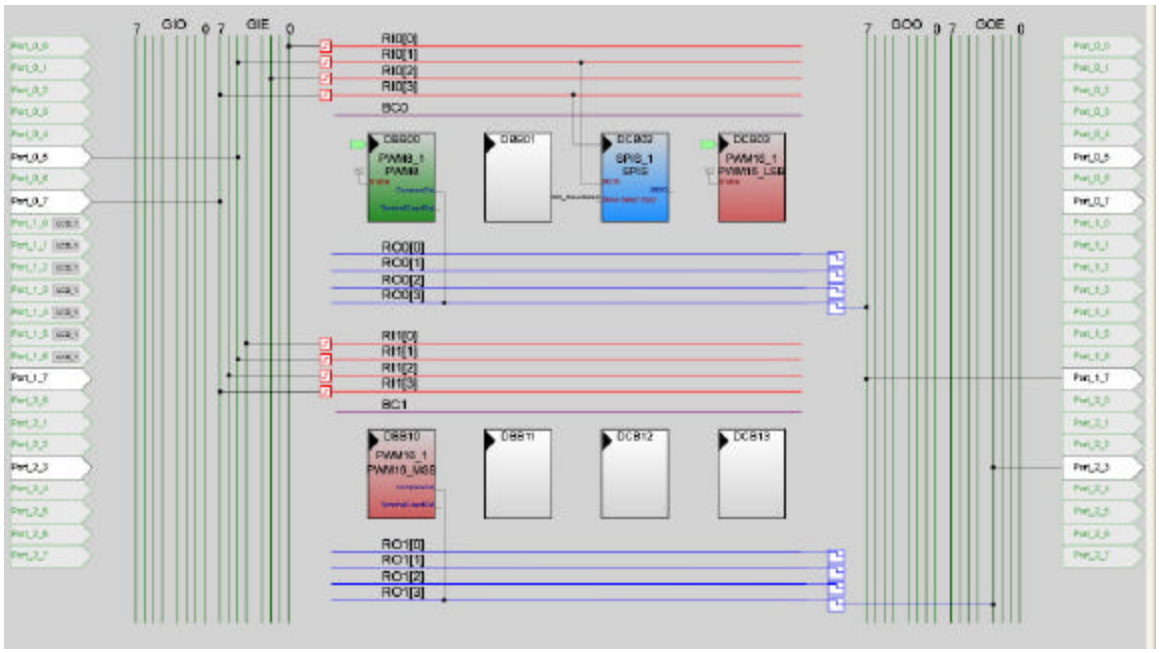


Figure 3. PSoC User Module Placement

Algorithm:-

Algorithm for the easy Morse coder project is given below:

This has 2 functions

- Key board
- Morse transmission

Keyboard

1. Initialize all modules and set Morse transmission speed default.
2. Wait for any key stroke from key board.
3. Recognize key.
4. If it is Function keys, Set morse transmission speed or change contrast of LCD by changing the pulse width of PWM or Transmit Morse code if array has some character.
5. Else if it is Character store it an array and display the same on LCD.
6. Back to step 2.

Morse Transmission

1. Preferred key to transmit is Return key (Enter key).
2. Disable keyboard and Enable Global interrupt.
3. Get Morse code and No. of bits to transmit.
4. Check LSB is '0' or '1', If '0' keep port 2[3] high for 'T' sec. If '1' than high for '3T' sec (Value of 'T' depends of speed setting).
5. Right shift Morse code to transmit next bit, if any.
6. Clear array and Display blank.
7. Repeat steps from 1 to 5.

Flow chart

Figure 4 shows the flow chart of the easy Morse coder project.

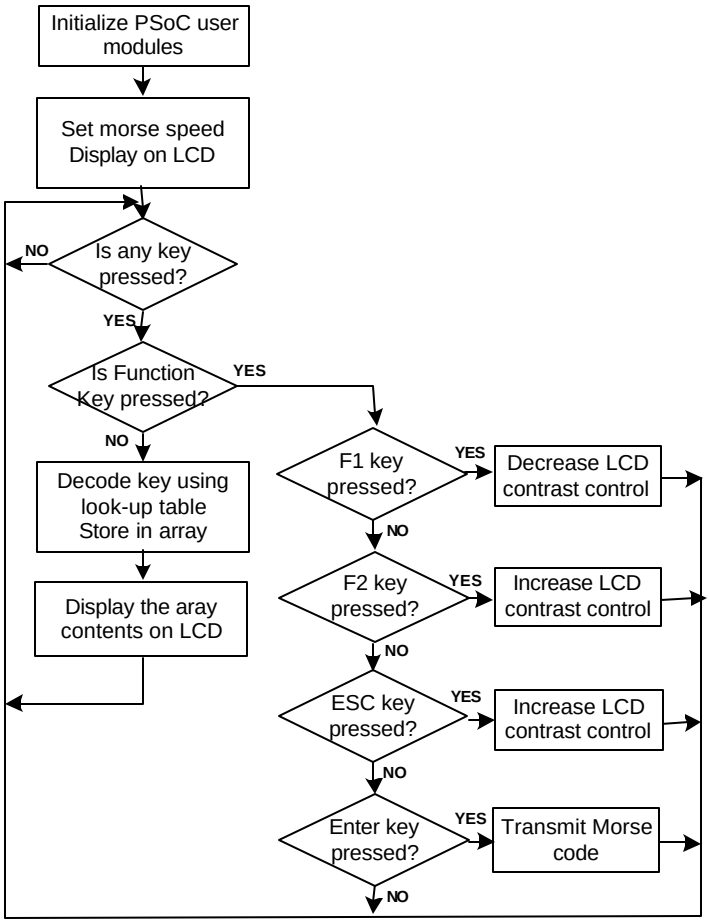
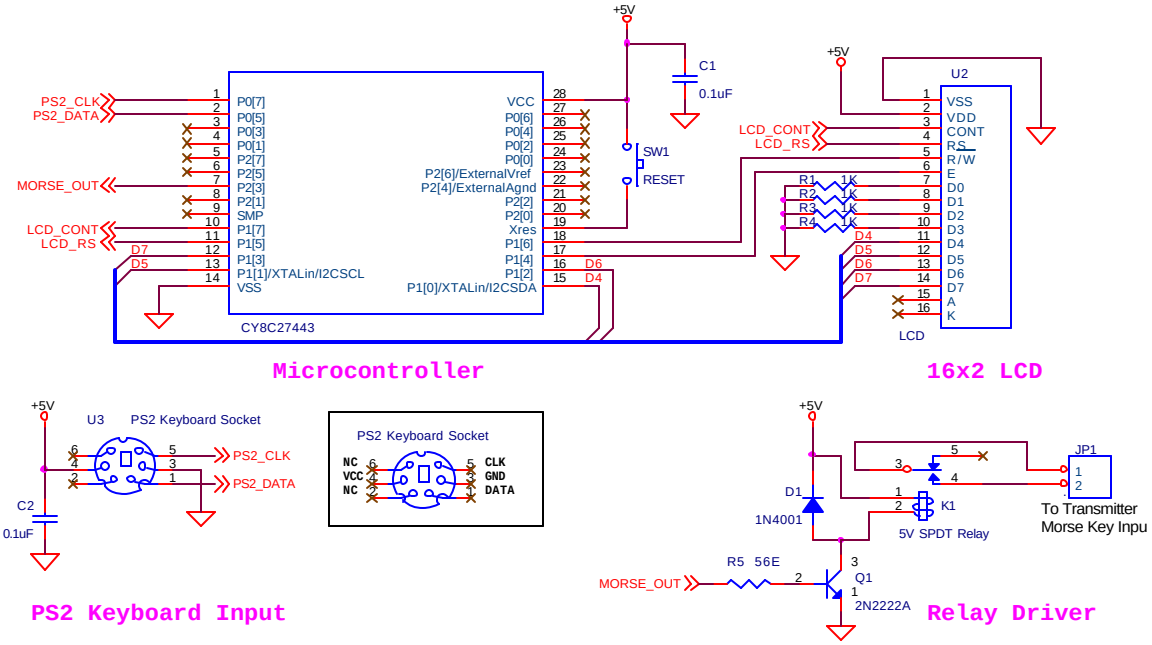


Figure 4. Flow chart

Schematics

The figure 5 shows the schematic of the Morse Coder. The PSoC port 1 is used to drive the LCD in 4-bit interface mode. The PS/2 keyboard is connected to the PSoC through pins P0[7] and P0[5]. In turn these pins are routed to the SPI Slave user module connections SCLK and SDA respectively of the PSoC. The output of PWM8 user module is connected to the LCD contrast control (Pin 3). The contrast of the LCD is controlled through the keyboard keys F1 and F2. The F1 key press results in decreasing the contrast of the LCD and F1 key press increases the contrast. The Morse code is generated using PWM16 user module. The morse code is output through PSoC device pin P2[3]. The generated Morse code is fed to an amateur radio transceiver Morse key input through a relay driver transistor (Q1) and a Relay (K1) circuit. Diode D1 is used to block the back EMF generated by the relay.



Easy Morse Coder

Figure 5. Schematics

Source Code for Easy Morse Coder

File: main.c

```
//-------------------------------------------------------------------------------
// Project Name     : Easy Morse Coder
// Project ID       : EMC001
// Target Processor : CY8C27443
// Author           : SG Sriharsha & HN Naveen
//-------------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "lcd_1.h"
#include "pwm8_1.h"
#include "PWM16_1.h"
#include "SPIS_1.h"
#include "scancodes.h"    // keyboard and morse code lookup table

unsigned char interrupt_flag;   // Global variable

/* main function  */
void main()
{
    /* Varialable Declaration   */

     static unsigned int  dash_period, dot_period;
     static unsigned int  dash_width, dot_width;
     static unsigned char character_space, word_space;
     static unsigned char bData, DataToTx;
     static unsigned char key_up, NoOfBits, OneOrZeroBit;
     static unsigned char i;
     static unsigned char buffer_cnt=0;
     char wpm;
     char kb_data[] = "                ";/*16 white space to initialize array*/
     char *ptr_kb_data;
     char *ptr_wpm;
     ptr_kb_data = kb_data;
     ptr_wpm = &wpm;

/* Initialization of PSoC Blocks/Modules  */
    PWM8_1_DisableInt();
    PWM16_1_DisableInt();
    PWM8_1_Start();
    LCD_1_Init();
    LCD_1_Start();

    LCD_1_Position(0,1);
    LCD_1_PrCString("EMC");   /* Easy Morse Coder version display */
    *ptr_wpm = 0x13;
    dash_period = 12799;    /* Default WPM setting  */
    dash_width  = 9599;
    dot_period  = 6399;
    dot_width   = 3199;
    character_space = 16;
    word_space = 48;
    LCD_1_Position(0,7);
    LCD_1_PrCString("-WPM");
    LCD_1_Position(0,5);
    LCD_1_PrHexByte(wpm);

  /* Keyboard Board Interface using SPI slave */
  /* parity bit is ignored --- SPI block is only 8 bit */
```

```
  while(1)
   {
     SPIS_1_Stop();/*every time necessary to clear shift reg and data reg*/
     SPIS_1_ClearSS();  /* make sure in Slave mode  */
      SPIS_1_Start(SPIS_1_SPIS_MODE_2 | SPIS_1_SPIS_LSB_FIRST );   /* Mode 2  -
data capture on trailing edge */

     while(!(SPIS_1_bReadStatus() & SPIS_1_SPIS_SPI_COMPLETE));
     bData = SPIS_1_bReadRxData();     /* Data only - Parity bit is ignored */

     if(!key_up)     /* to avoid break code   */
     {
      switch(bData)
      {
        case 0xAA: key_up = 0;   break;/* Keyboard Basic Assurance Test - OK*/
        case 0xF0: key_up = 1; break;   /* Breack code */
        case 0x76:    /* ESC key to clear the display and morse code array */
        ptr_kb_data = kb_data;
        for(i=0; i<16; i++)
        {
         *ptr_kb_data = ' ';
         ptr_kb_data++;
        }
        ptr_kb_data = kb_data;
        buffer_cnt =0;
        break;

        case 0x05:   /* F1 - Decerese LCD Contrast */
        PWM8_1_WritePulseWidth((PWM8_1_bReadPulseWidth()+2)&0x1F);
        break;
        case 0x06:   /* F2 - Increase LCD Contrast */
        PWM8_1_WritePulseWidth((PWM8_1_bReadPulseWidth()-2)&0x1F);
        break;

        case 0x04: *ptr_wpm = 0x25; character_space = 8; word_space = 24;
        dash_period=6399; dash_width=4799; dot_period=3199; dot_width=1599;
        break;
        case 0x0C: *ptr_wpm = 0x13; character_space = 16; word_space = 48;
        dash_period=12799; dash_width=9599; dot_period=6399; dot_width=3199;
        break;
        case 0x03: *ptr_wpm = 0x8; character_space = 32; word_space = 94;
        dash_period=25599; dash_width=19199; dot_period=12799; dot_width=6399;
        break;
        case 0x0B: *ptr_wpm = 0x4; character_space = 47; word_space = 141;
        dash_period=38399; dash_width=28799; dot_period=19199; dot_width=9599;
        break;

          case 0x5A:   /* Morse transmission from Morse buffer */
          SPIS_1_Stop();
          LCD_1_Position(0, 12);
          LCD_1_PrCString("Ting");
          ptr_kb_data = kb_data;
          while(buffer_cnt != 0)
          {
             for(i=0; codes[i][1] != *ptr_kb_data && codes[i][1]; i++);
             DataToTx = codes[i][2];
             NoOfBits = codes[i][3];
             if(NoOfBits == 7)
             {
               for(i=0; i<word_space; i++)    /* space between two words  */
                   LCD_1_Delay50uTimes(255);
             }
             else
```

```
   {
      OneOrZeroBit = 0x01;
      PWM16_1_EnableInt();
      M8C_EnableGInt;
      for(i=0; i<NoOfBits; i++)
      {
        interrupt_flag = 0;
        if(DataToTx & OneOrZeroBit)
        {   /* Dash */
          PWM16_1_WritePeriod(dash_period);
          PWM16_1_WritePulseWidth(dash_width);  /* 75% duty cycle  */
          PWM16_1_Start();
          while(interrupt_flag == 0);
        }
        else
        {    /* Dot  */
          PWM16_1_WritePeriod(dot_period);
          PWM16_1_WritePulseWidth(dot_width);  /* 50% duty cycle  */
          PWM16_1_Start();
          while(interrupt_flag == 0);
        }
     PWM16_1_Stop();/*Stop PWM16 after transmitting every character*/
     OneOrZeroBit = OneOrZeroBit << 1;
      }
      for(i=0; i<character_space; i++) /* delay between character  */
          LCD_1_Delay50uTimes(255);
   }
   buffer_cnt--;
   ptr_kb_data++;
}   /* while */
LCD_1_Position(0, 12);
LCD_1_PrCString("Txed");
for(i=0; i<125; i++)
  LCD_1_Delay50uTimes(255); /* Display Transmitted for few seconds */
LCD_1_Position(0, 12);
LCD_1_PrCString("    ");
/* clear data in array  */
ptr_kb_data = kb_data;
for(i=0; i<16; i++)
{
  *ptr_kb_data = ' ';
  ptr_kb_data++;
}
ptr_kb_data = kb_data;
break;

case 0x66:   /* Backspace - delete last character from buffer */
if (buffer_cnt == 0)
    ptr_kb_data = kb_data;
else
{
  ptr_kb_data--;
  buffer_cnt--;
  *ptr_kb_data = ' ';
}
break;

default:
if(buffer_cnt != 16)
{
   for(i=0; codes[i][0] != bData && codes[i][0] ; i++);
       if(codes[i][0] == bData)
       {
```

```
                    *ptr_kb_data = codes[i][1];
                    ptr_kb_data++;
                    buffer_cnt++;
                  }  /* Data found  */
            }   /* if buffer_cnt = 16 than buffer_full   */
            break;
       } /* switch */
    }  /* if key */
    else
        key_up = 0;

    LCD_1_Position(1,0);
    LCD_1_PrString(kb_data);
    LCD_1_Position(0,5);
    LCD_1_PrHexByte(wpm);

  } /* while */

 } /* END of Main */

 // Interrupt service routine for morse transmission from PWM16_1
 #pragma interrupt_handler PWM16_1_ISR
 void PWM16_1_ISR()
 {
   interrupt_flag = 1;
   return;
 }
```

File: scancodes.h

```
unsigned char codes[][4] = {
0x1C, 'A', 0x02, 2,
0x32, 'B', 0x01, 4,
0x21, 'C', 0x05, 4,
0x23, 'D', 0x01, 3,
0x24, 'E', 0x00, 1,
0x2B, 'F', 0x04, 4,
0x34, 'G', 0x03, 3,
0x33, 'H', 0x00, 4,
0x43, 'I', 0x00, 2,
0x3B, 'J', 0x0E, 4,
0x42, 'K', 0x05, 3,
0x4B, 'L', 0x02, 4,
0x3A, 'M', 0x03, 2,
0x31, 'N', 0x01, 2,
0x44, 'O', 0x07, 3,
0x4D, 'P', 0x06, 4,
0x15, 'Q', 0x0B, 4,
0x2D, 'R', 0x02, 3,
0x1B, 'S', 0x00, 3,
0x2C, 'T', 0x01, 1,
0x3C, 'U', 0x04, 3,
0x2A, 'V', 0x08, 4,
0x1D, 'W', 0x06, 3,
0x22, 'X', 0x09, 4,
0x35, 'Y', 0x0D, 4,
0x1A, 'Z', 0x03, 4,
0x45, '0', 0x1F, 5,
0x16, '1', 0x1E, 5,
0x1E, '2', 0x1C, 5,
0x26, '3', 0x18, 5,
0x25, '4', 0x10, 5,
```

```
0x2E, '5', 0x00, 5,
0x36, '6', 0x01, 5,
0x3D, '7', 0x03, 5,
0x3E, '8', 0x07, 5,
0x46, '9', 0x0F, 5,
0x29, ' ', 0x00, 7,     /* Space - 7 dits between Word */
0x49, '.', 0x2A, 6,     /* perod (full stop) */
0x41, ',', 0x33, 6,     /* comma */
0x55, '=', 0x11, 5,     /* double dash [=]  */
0x4E, '-', 0x21, 6,     /* Hyphen   */
0, 0, 0, 0
};
```

Morse code generation logic

The "codes" array in scanecodes.h has 4 column
1st column --- scan code
2nd column --- character
3rd column --- morse code
4th column --- no of bits in morse code

Morse code example

Table 1. Morse Code Example

Character	Morse Code	Morse in Hex	No of bits
A	.-	0x02	2
E	.	0x00	1
I	..	0x00	2
O	---	0x07	3
U	..-	0x04	3
9	----.	0x1E	5
6	-....	0x10	5
0	-----	0x1F	5

'0' represents *dot*
'1' represents *dash*

References

[1] AN2XXX, "Interfacing a PS/2 Keyboard using SPI Slave", Cypress Microsystems.
[2] Web reference: http://en.wikipedia.org/wiki/Morse_code
[3] Web reference: http://www.naveenmysore.com/hamradio/morse.htm/
[4] Web reference: http://www.wrvmuseum.org/morsecode/morsecodehistory.htm
[5] Web reference: http://www.qsl.net/ka1ddb/hrterminology.html

Table 2. Morse Code Character Set

A	●–	Period [.]	●–●–●–	Ä	●–●–
B	–●●●	Comma [,]	––●●––	Á, Å, À, Â	●––●–
C	–●–●	Question mark [?]	●●––●●	Ç	–●–●●
D	–●●	Hyphen [-]	–●●●● –	É, È	●●–●●
E	●	Double dash [=]	–●●●–	È	●–●●–
F	●●–●	Colon [:]	––– ●●●	Ê	–●●–●
G	––●	Semicolon [;]	–●–●–●	Ö,Ó	––– ●
H	●●●●	Left parenthesis [(]	–●––●	Ñ	––●––
I	●●	Right parenthesis [)]	–●––●–	Ü	●●––
J	●–––	Fraction bar [/]	–●●–●	Ž	––●●–
K	–●–	Quotation marks ["]	●–●●–●		
L	●–●●	Dollar sign [$]	●●●–●●–		
M	––	Apostrophe [']	●––––●		
N	–●	Underline [_]	●●––●–		
O	–––	End of message or cross [+]	●–●–●		
P	●––●	Paragraph [¶]	●–●–●●		
Q	––●–	End of work [‘]	●●●–●–	0	–––––
R	●–●	Wait [|]	●–●●●	1	●––––
S	●●●	Understood [*]	●●●–●	2	●●–––
T	–	Starting signal [➔]	–●–●–	3	●●●––
U	●●–			4	●●●●–
V	●●●–			5	●●●●●
W	●––			6	–●●●●
X	–●●–			7	––●●●
Y	–●––			8	–––●●
Z	––●●			9	––––●

About the Authors

Name: S. G. Sriharsha
Title: DVD Front-End Application Engineer
Background: Sriharsha is a Bachelor graduate in Telecommunication. Working with ST Microelectronics Asia Pacific Pte Ltd, Singapore.
Contact: sriharshasg@gmail.com

Name: H. N. Naveen
Title: Senior Design Engineer
Background: Naveen is a Masters graduate in Industrial Electronics. He is working with Sasken Communication Technologies, Bangalore. He has experience in hardware modeling and designing microcontroller boards. His interests are product development and reference designs.
Contact: navmys@gmail.com or navmys@lycos.com
Web Site: http://www.naveenmysore.com

Cypress MicroSystems, Inc.
2700 162nd Street SW, Building D
Lynnwood, WA 98037
Phone: 800.669.0557
Fax: 425.787.4641
http://www.cypress.com/ / http://www.cypress.com/support/mysupport.cfm