Serial Port Control and Debugging of the Target

Introduction

When developing small real-time DSP applications it is nice to be able to interact with the FM4 S6E2CC Pioneer, STM32F4 Discovery, or LPC4088 Quick Start, from the PC host. A variety of interfaces exist on each board, but the focus of this appendix is creating simple control interfaces using serial port communications between a PC host and the target board using a USB-to-TTL serial port adapter.

- A uniform solution across the first two platforms is complete
- The existence of three different communities contributing software for each platform, more work/time remains to complete this task for the NXP LPC4088

FM4 S6E2CC Pioneer Kit

- The FM4 Pioneer has a very extensive set of peripherals, giving many options
- The simple option chosen here is to use the virtual serial port which is part of the CMSIS-DAP interface used for program-
ming and debugging code

• On the host side a serial port (RS232) interface made with a terminal program, such as the *Serial Port Viewer and Terminal* or some other program such as the GUI slider app

  *FM4_GUI_slider.exe* or perhaps Python using the *pyserial* package or MATLAB via its serial port library

• To get to this point you need to first make sure Keil and the latest FM4 Packs are installed, then additionally install¹:
  – The *CMSIS-DAP driver* package which also installs the virtual serial port driver
  – Then install the *Serial Port Viewer*

¹.http://www.cypress.com/documentation/development-kitsboards/fm4-176l-s6e2cc-eth-arm-cortex-m4-mcu-starter-kit-ethernet-and
To talk to the virtual serial port using C-code on the FM4 Pioneer, we use a high level library discussed in sample code provided at the FM4 Pioneer Kit Web Site (above footnote):

<table>
<thead>
<tr>
<th>Software Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Labs</td>
</tr>
<tr>
<td>Master S6E2CC Peripheral Driver Lib</td>
</tr>
<tr>
<td>PCB Gerber files</td>
</tr>
<tr>
<td>CD Contents</td>
</tr>
<tr>
<td>Board Test</td>
</tr>
<tr>
<td>Demonstrate LwIP on FM4 on FM4-176L-S6E2CC</td>
</tr>
</tbody>
</table>

From ZIP extract S6E2CC_PDL v0.2.ZIP and drill into the UART folder

There are many examples for the entire FM4 that explore all the peripherals

Here the focus is the UART library that uses the multifunction serial port (MFS) on port 0 or MFS0

– MFS0 talks to the virtual serial port

The there is a low-level and high-level interface

Here we choose the high-level interface referred to as mfs_unuse_int
Serial Communication and GUI Control

- When developing real-time DSP code it is at times nice to be able to make live interactive parameter adjustments while your application is running.
- To meet this need a simple serial port GUI app from PC through the virtual serial port on FM4 Pioneer board was written.
- The GUI program is hosted on a Windows PC, but the tool used to develop it, Xojo¹, is cross platform, so an app can be compiled to run under Mac OS or Linux (32bit); at present only a Windows app is in the ZIP package for this example.

¹. http://www.xojo.com/

---

The Lab1_UART demo project uses FM4.P_vals[n], n=0,5 to define variables on the left and right channels of a line in/line out audio stream, at fs = 48 ksp. Note P_vals[] is a float array field of the struct FM4.

---

Six parameter slider app giving real-time control of the FM4 Pioneer Kit.
The parameters configuration dialog, shown below, allows parameter attributes to be set, including label.

- In the case of the ARM LiB intro lab, Lab1, the project file structure still contains **drivers** and **src** folders.
- The drivers folder contains 10 additional files for the UART:
Within Keil the project the `drivers` folder contains four more files, while the `src` folder remains the same relative to say the `loop_itr` project configuration of Lab1

Files added to the drivers project folder of a UART-based project

As for integrating the UART and GUI slider into your source code, the process is simple

You have to:

- `#include FM4_slider_interface.h`
- Instantiate a C struct `FM4_GUI_slider`
- Initialize the structure
- Update the structure in the main while loop

For example in `loop_intr.c` of Lab1 we now have:

```c
// loop_intr.c

#include "audio.h"
#include "FM4_slider_interface.h"

// Create (instantiate) GUI slider data structure
struct FM4_slider_struct FM4_GUI;

// Analog I/O working variables
volatile int16_t audio_chR=0;    //16 bits audio data channel right
```
volatile int16_t audio_chL=0;   //16 bits audio data channel left

void I2S_HANDLER(void) {   /****** I2S Interruption Handler *****/

    //gpio_toggle(TEST_PIN);
gpio_set(TEST_PIN, HIGH); //Pin P10

    audio_IN = i2s_rx();
audio_chL = (audio_IN & 0x0000FFFF);       //Separate 16 bits channel left
    audio_chR = ((audio_IN >>16)& 0x0000FFFF); //Separate 16 bits channel right

    //Process
    //delay_us(5);
audio_chL = (int16_t) (audio_chL*FM4_GUI.P_vals[0]); // slider gain
    audio_chR = (int16_t) (audio_chR*FM4_GUI.P_vals[1]); // slider gain

    //Put the two channels together again
    audio_OUT = ((audio_chR<<16 & 0xFFFF0000)) + (audio_chL & 0x0000FFFF);//
i2s_tx(audio_OUT);

    gpio_set(TEST_PIN, LOW);
}

int main(void)
{
    gpio_set_mode(TEST_PIN,Output);

    // Initialize the slider interface by setting the baud rate (460800
    // or 921600) and initial float values for each of the 6 slider
    // parameters:
    init_slider_interface(&FM4_GUI,460800, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0);

    // Send a string to the terminal
    write_uart0("Hello FM4 World!\r\n");

    //audio_init ( hz48000, mic_in, intr, I2S_HANDLER);
audio_init ( hz48000, line_in, intr, I2S_HANDLER);

    while(1){

        // Update slider parameters
        update_slider_parameters(&FM4_GUI);

    }
}

• Check the output on the Soundcard scope and manipulate the
  P0 and P1 sliders (music playing through)
Debugging Using the Serial Port with sprintf()

- You can also send formatted strings to the PC terminal, such as the float buffer[] holding output from sine_lut_buf_intr.c

```c
// sine_lut_buf_intr.c

#include "audio.h"
#include "FM4_slider_interface.h"

volatile int16_t audio_chR=0;
volatile int16_t audio_chL=0;

#define LOOP_SIZE 8
#define BUFFER_SIZE 100

int16_t sine_table[LOOP_SIZE] = {0, 7071, 10000, 7071, 0, -7071, -10000, -7071};

static int sine_ptr = 0;

float32_t buffer[BUFFER_SIZE];
float32_t temp;
```
static int buf_ptr = 0;

// Create GUI slider data structure
struct FM4_slider_struct FM4_GUI;

void I2S_HANDLER(void) {
    /* I2S Interruption Handler*/
    audio_IN = i2s_rx();
    audio_chL = (audio_IN & 0x0000FFFF);
    audio_chR = ((audio_IN >> 16) & 0x0000FFFF);

    temp = sine_table[sine_ptr] * FM4_GUI.P_vals[0];
    // audio_chL = sine_table[sine_ptr];
    audio_chL = (int16_t) temp;
    audio_chR = sine_table[sine_ptr];
    buffer[buf_ptr] = sine_table[sine_ptr];
    sine_ptr = (sine_ptr + 1) % LOOP_SIZE;
    buf_ptr = (buf_ptr + 1) % BUFFER_SIZE;

    audio_OUT = ((audio_chR << 16 & 0xFFFF0000) + (audio_chL & 0x0000FFFF));
    i2s_tx(audio_OUT);
}

int main(void) {
    char message[50];
    uint8_t buff_print_flag = TRUE;
    uint16_t i;

    // Initialize the slider interface by setting the baud rate (460800
    // or 921600) and initial float values for each of the 6 slider
    // parameters:
    init_slider_interface(&FM4_GUI, 460800, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0);
    // Initialize the Wolfson audio ADC/DAC interface
    audio_init (hz8000, mic_in, intr, I2S_HANDLER);

    while(1) {

        // Update slider parameters
        update_slider_parameters(&FM4_GUI);

        if ((buf_ptr == BUFFER_SIZE - 1) && (buff_print_flag))
        {
            for (i = 0; i < BUFFER_SIZE; i++)
            {
                sprintf(message, "%.2f\r\n", buffer[i]);
                write_uart0(message);
            }
            buff_print_flag = FALSE;
        }
    }
}
Appendix A • Serial Port Control and Debugging of the Target

- Running the above file and observing the serial terminal:

![Serial terminal screenshot]

- The slider control GUI and UART interface files
  - FM4_slider_interface.h

```c
// FM4_slider_interface.h
/*
   Header code module to support the FM4 S6E2CC GUI slider interface app

   Six slider variables are stored in the global float array P_vals[6]. The initial
   values given to this array in the main module are the values used when a
   real-time DSP application first launches on the FM4 S6E2CC board.

   The main component in making this interface work is a UART running through
   the virtual comm port provided along side the CMSIS-DAP debug interface on
   the board.

   Copyright (C) 2016 Mark Wickert

   The Spansion (now part of Cypress) software is acknowledged below.
*/
```

---

ECE 5655/4655 Real-Time DSP
* BY DOWNLOADING, INSTALLING OR USING THIS SOFTWARE, YOU AGREE TO BE BOUND
* BY ALL THE TERMS AND CONDITIONS OF THIS AGREEMENT.
*
* This software contains source code for use with Spansion
* components. This software is licensed by Spansion to be adapted only
* for use in systems utilizing Spansion components. Spansion shall not be
* responsible for misuse or illegal use of this software for devices not
* supported herein. Spansion is providing this software "AS IS" and will
* not be responsible for issues arising from incorrect user implementation
* of the software.
*
* SPANSION MAKES NO WARRANTY, EXPRESS OR IMPLIED, ARISING BY LAW OR OTHERWISE,
* REGARDING THE SOFTWARE (INCLUDING ANY ACOOMPANYING WRITTEN MATERIALS),
* ITS PERFORMANCE OR SUITABILITY FOR YOUR INTENDED USE, INCLUDING,
* WITHOUT LIMITATION, THE IMPLIED WARRANTY OF MERCHANTABILITY, THE IMPLIED
* WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE, AND THE IMPLIED
* WARRANTY OF NONINFRINGEMENT.
* SPANSION SHAL HAVE NO LIABILITY (WHETHER IN CONTRACT, WARRANTY, TORT,
* NEGLIGENCE OR OTHERWISE) FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT
* LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION,
* LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING FROM USE OR
* INABILITY TO USE THE SOFTWARE, INCLUDING, WITHOUT LIMITATION, ANY DIRECT,
* INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES OR LOSS OF DATA,
* SAVINGS OR PROFITS,
* EVEN IF SPANSION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
* YOU ASSUME ALL RESPONSIBILITIES FOR SELECTION OF THE SOFTWARE TO ACHIEVE YOUR
* INTENDED RESULTS, AND FOR THE INSTALLATION OF, USE OF, AND RESULTS OBTAINED
* FROM, THE SOFTWARE.
*
* This software may be replicated in part or whole for the licensed use,
* with the restriction that this Disclaimer and Copyright notice must be
* included with each copy of this software, whether used in part or whole,
* at all times.
*/

/ ****************************************************************************
/* Include files                                                            */
/ ****************************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mfs_hl.h"

/ ****************************************************************************/
Appendix A • Serial Port Control and Debugging of the Target

/* Local pre-processor symbols/macros ('#define') */

#define SAMPLE_UART_TX_BUFFSIZE  (128)
#define SAMPLE_UART_RX_BUFFSIZE  (256)
#define SAMPLE_UART_RX_BUFF_FILL_LVL  (1) // read just one character

struct FM4_slider_struct
{
    float32_t P_vals[6];
    char P_rcvd[10];
    uint8_t P_idx;
    stc_mfs_hl_uart_config_t stcMfsHlUartCfg;
    uint8_t au8ReadBuf[2];
    uint16_t ul6ReadCnt;
    uint8_t H_found;
    uint8_t idx_P_rec;
};

void init_slider_interface(struct FM4_slider_struct *FM4_slider, uint32_t baud_rate, float32_t P_val0, float32_t P_val1, float32_t P_val2, float32_t P_val3, float32_t P_val4, float32_t P_val5);

void update_slider_parameters(struct FM4_slider_struct *FM4_slider);

void write_uart0(char* message);

– FM4_slider_interface.c

// FM4_slider_interface.c

/*
Code module to support the FM4 S6E2CC GUI slider interface app

Six slider variables are stored in the global float array P_vals[6].
The initial values given to this array in the main module are the values used
when a real-time DSP application first launches on the FM4 S6E2CC board.
The main component in making this interface work is a UART running through
the virtual comm port provided along side the CMSIS-DAP debug interface on
the board.
Copyright (C) 2016 Mark Wickert

The Spansion (now part of Cypress) software is acknowledged below.
*/

#include "FM4_slider_interface.h"

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

ECE 5655/4655 Real-Time DSP
static uint8_t au8UartTxBuf[SAMPLE_UART_TX_BUFFSIZE];
static uint8_t au8UartRxBuf[SAMPLE_UART_RX_BUFFSIZE];

/**

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

/* Function implementation - global ('extern') and local ('static') */

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

void init_slider_interface(struct FM4_slider_struct *FM4_slider,
Appendix A • Serial Port Control and Debugging of the Target

```c
uint32_t baud_rate,float32_t P_val0,
float32_t P_val1, float32_t P_val2,
float32_t P_val3, float32_t P_val4,
float32_t P_val5)
{

    //Variables for parameter slider communication
    FM4_slider->P_vals[0] = P_val0;
    FM4_slider->P_vals[1] = P_val1;
    FM4_slider->P_vals[2] = P_val2;
    FM4_slider->P_vals[3] = P_val3;
    FM4_slider->P_vals[4] = P_val4;
    FM4_slider->P_vals[5] = P_val5;
    FM4_slider->H_found = 0;

    // Baud rate : 115200 bps (230400, 460800, 921600)
    FM4_slider->stcMfsHlUartCfg.u32DataRate = baud_rate;

    // Bit direction : LSB first
    FM4_slider->stcMfsHlUartCfg.bBitDirection = FALSE;

    // Inverted serial data format : NRZ
    FM4_slider->stcMfsHlUartCfg.bSignalSystem = FALSE;

    // Inverted serial data format : NRZ
    FM4_slider->stcMfsHlUartCfg.bHwFlow = FALSE;

    // Inverted serial data format : NRZ
    FM4_slider->stcMfsHlUartCfg.pu8TxBuf = au8UartTxBuf;

    // Use the reception buffer in the driver (*2)
    FM4_slider->stcMfsHlUartCfg.pu8RxBuf = au8UartRxBuf;

    // Size of transmit buffer (for *1)  : 128bytes
    FM4_slider->stcMfsHlUartCfg.u16TxBufSize = SAMPLE_UART_TX_BUFFSIZE;

    // Size of reception buffer (for *2) : 256bytes
    FM4_slider->stcMfsHlUartCfg.u16RxBufSize = SAMPLE_UART_RX_BUFFSIZE;

    // Unread counts of reception buffer to call RX Callback function : 1byte
    FM4_slider->stcMfsHlUartCfg.u16RxCbBufFillLvl = SAMPLE_UART_RX_BUFF_FILL_LVL;

    // Uart mode : Normal mode
    FM4_slider->stcMfsHlUartCfg.u8UartMode = MfsUartNormal;

    // Parity    : none
    FM4_slider->stcMfsHlUartCfg.u8Parity = MfsParityNone;

    // Stop bit  : 1bit
    FM4_slider->stcMfsHlUartCfg.u8StopBit = MfsOneStopBit;

    // Data bits : 8bit
    FM4_slider->stcMfsHlUartCfg.u8CharLength = MfsEightBits;

    // FIFO      : not use
    FM4_slider->stcMfsHlUartCfg.u8FifoUsage = MfsHlUseNoFifo;

    // Callback function : if NULL ==> interrupt not used
    FM4_slider->stcMfsHlUartCfg.pfnRxCb = NULL;

    // Callback function : if NULL ==> interrupt not used
    FM4_slider->stcMfsHlUartCfg.pfnTxCb = NULL;

    //************ Configure UART Serial0 *******************
    // Disable Analog input (P21:SIN0_0/AN17, P22:SOT0_0/AN16)
    FM4_GPIO->ADE = 0;
```

---

A–14   ECE 5655/4655 Real-Time DSP
// Set UART Ch0_0 Port (SIN, SOT)
FM4_GPIO->PFR2 = FM4_GPIO->PFR2 | 0x0006;
FM4_GPIO->EPFR07 = FM4_GPIO->EPFR07 | 0x00000040;
//****************************************************************************

// Initialize the MFS ch.0 as UART
if (Ok != Mfs_Hl_Uart_Init(&MFS0, &FM4_slider->stcMfsHlUartCfg))
{
    while(1);
}

// Write a test string
//Mfs_Hl_Write(&MFS0, (uint8_t*) "Getting started\n", 17, FALSE, FALSE);

void update_slider_parameters(struct FM4_slider_struct *FM4_slider)
{
    // If data is received from UART,
    // Get character from internal buffer and
    // decode char string into slider float32_t
    // held in P_vals[] array
    // Receive data from UART asynchronously (Non-blocking)
    if (Ok != Mfs_Hl_Read(&MFS0, FM4_slider->au8ReadBuf,
        &FM4_slider->u16ReadCnt, 1, FALSE))
    {
        Mfs_Hl_Write(&MFS0, (uint8_t*) "Read Error\n", 12, FALSE, FALSE);
    }

    if ((FM4_slider->u16ReadCnt == 1) && (FM4_slider->au8ReadBuf[0] >= 0x2D)) {
        //gpio_set(TEST_PIN, HIGH); //Pin A3/P10
        if (FM4_slider->H_found == 0) {
            if (FM4_slider->au8ReadBuf[0] == 0x48) {  // 'H' <=> 0x48
                //gpio_set(A4, HIGH); // A4/P10
                FM4_slider->H_found = 1;
            }
        }
        else {
            // '0' <= 0x30, '9' <= 0x39, '.' <= 0x2E, '-' <= 0x2D
            if ((FM4_slider->au8ReadBuf[0] == 0x30 &&
                (FM4_slider->au8ReadBuf[0] <= 0x39) ||
                (FM4_slider->au8ReadBuf[0] == 0x2E) ||
                (FM4_slider->au8ReadBuf[0] == 0x2D)) {
                FM4_slider->P_rcvd[FM4_slider->idx_P_rec] =
                    (char) FM4_slider->au8ReadBuf[0];
                FM4_slider->idx_P_rec++;
            }
        }
    }
}
else if (FM4_slider->au8ReadBuf[0] == 0x3A) { // ':' <=> 0x3A
    FM4_slider->P_idx = (uint8_t) atoi(FM4_slider->P_rcvd);
    FM4_slider->idx_P_rec = 0;
    // clear received char array
    memset(FM4_slider->P_rcvd, 0, sizeof(FM4_slider->P_rcvd));
}
else if (FM4_slider->au8ReadBuf[0] == 0x54) { // 'T' <=> 0x54
    FM4_slider->P_vals[FM4_slider->P_idx] = (float32_t) atof(FM4_slider->P_rcvd);
    memset(FM4_slider->P_rcvd, 0, sizeof(FM4_slider->P_rcvd));
    FM4_slider->idx_P_rec = 0;
    FM4_slider->H_found = 0;
}
else {
    // memset(P_rcvd, 0, sizeof(P_rcvd));
    FM4_slider->H_found = 0;
}
}

void write_uart0(char* message)
{
    // write a string to UART0
    Mfs_Hl_Write(&MFS0, (uint8_t*) message, strlen(message), FALSE, FALSE);
}
/
**************************************************************************
/* EOF (not truncated)                                                      */
**************************************************************************
/
STM32F4 Discovery

- The STM32F4 has limited device peripherals on the board itself, but great potential exists for interfacing through the GPIO pins
- The STM32F4 has a fairly active user community, with one such site being http://stm32f4-discovery.com/
• Thanks to Majerle Tilen, the primary contributor on the above site, for libraries that allow access to the USARTs, the user (blue) programmable button, and the four user LEDs
• In the remainder of this section code examples are provided for using these STM32F4 interfaces
• The hardware interface to the STM32F4 is shown below:

1. See for example the *Honey House* product at [http://www.amazon.com/gp/product/B00PQMUBQK/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1](http://www.amazon.com/gp/product/B00PQMUBQK/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1)
Serial Communication and GUI Control

• When developing real-time DSP code it is at times nice to be able to make *live* interactive parameter adjustments while your application is running

• To meet that need a simple serial port interface from PC to USART pins on the STM32F4 Discovery board was written

• The GUI program is hosted on a Windows PC, but the tool used to develop it, Xojo\(^1\), is cross platform, so an app can be compiled to run under Mac OS or Linux (32bit); at present only a Windows app is in the ZIP package for this example

\(1. \text{http://www.xojo.com/}\)
• The parameters configuration dialog:

- Button press decoding and the use of the four user LEDs to serve as state indicators. Here just button press is decoded. What else can you do?
- Enter the minimum, initial, maximum, and step size the slider parameter should range over. The range may be bipolar or negative.
- The number of slider steps from the attribute settings
- Select the number of decimal and fractional digits to display and send via the serial link
- Locks values for index being edited
- Saves all values to .ini file and applies values to the slider display

• A sample Keil MD5 project is contained in the ZIP package STM32F4_GUI_slider_project.zip
• The latest version of the GUI includes many enhancements
  – Increasing the number of parameters to 6
  – Allowing for app state retention via a .ini file
  – A configure panel to set the range and adjustment resolution of each parameter
  – TBD: Plotting of data returned from the STM32F4 to the GUI
• The good news is that the Xojo project is included in the ZIP
package, and you can add enhancements to the code with a free download of Xojo

– Without a license you can open projects, write code, and run code in the debug mode

– You cannot however build standalone executables

• The Keil project is based on the Lab0/Lab1 project with additional source code modules in the project

  ![Diagram of project structure]

  For USART
  New main module
  The Windows project folder

  For USART
  These two for button and LEDs

  For USART
  The first two are needed for the USART library

  The third file is needed for the button and LEDs

  The codec Wolfson library (all Lib need this)

  Added from Keil tools folder for USART support
The main module source code is given below:

```c
/**
 * USART For GUI Control of the STM32F4
 * via a Serial Connection to USART6 (tx:PC6, rx:PC7)
 *
 * The code libraries found at http://stm32f4-discovery.com/ are
 * used in this example.
 *
 * Mark Wickert, February 2015, Modified March 2015
 */
#include "defines.h"
#include "stm32f4xx.h"
#include "tm_stm32f4_usart.h"
#include "tm_stm32f4_delay.h"
#include "tm_stm32f4_disco.h"
#include "stm32_wm5102_init.h"

#define NUMBER_OF_FIELDS 6 //how many parameters to receive from GUI

//Globals for parameter sliders via serial port on USART6
float32_t P_vals[NUMBER_OF_FIELDS] = {1.0,1.0,1.0,1.0,1.0,1.0}; //init vals
int16_t P_idx;
int8_t H_found = 0;

void SPI2_IRQHandler()
{
    int16_t left_out_sample = 0;
    int16_t right_out_sample = 0;
    int16_t left_in_sample = 0;
    int16_t right_in_sample = 0;

    //GPIO_ToggleBits(GPIOA, GPIO_Pin_15);

    if (SPI_I2S_GetFlagStatus(I2Sx, I2S_FLAG_CHSIDE) == SET)
    {
        GPIO_SetBits(GPIOA, GPIO_Pin_15);
        left_in_sample = SPI_I2S_ReceiveData(I2Sx);
        left_out_sample = (int16_t)(P_vals[0]*left_in_sample);
        while (SPI_I2S_GetFlagStatus(I2Sxext, SPI_I2S_FLAG_TXE ) != SET){}
        SPI_I2S_SendData(I2Sxext, left_out_sample);
        GPIO_ResetBits(GPIOA, GPIO_Pin_15);
    }
    else
    {
        right_in_sample = SPI_I2S_ReceiveData(I2Sx);
        right_out_sample = (int16_t)(P_vals[1]*right_in_sample);
        while (SPI_I2S_GetFlagStatus(I2Sxext, SPI_I2S_FLAG_TXE ) != SET){}
        SPI_I2S_SendData(I2Sxext, right_out_sample);
    }
```
int main(void) {
    //Variables for parameter slider communication
    uint8_t c;
    char P_tx[20]; // tx echo chr array
    char P_rcvd[10]; // received chr array
    uint8_t i = 0;    //Initialize system
    SystemInit();
    /* Initialize LEDs on board */
    TM_DISCO_LedInit();
    /* Initialize button on board */
    TM_DISCO_ButtonInit();
    /* Initialize delay */
    TM_DELAY_Init();
    //Initialize USART6 at some baud rate, TX: PC6, RX: PC7
    // Timer error with TM_USART requires a recalc of the baud rate by 231/127
    TM_USART_Init(USART6, TM_USART_PinsPack_1, (231*230400)/127);
    // e.g., 9600 <=> 17461
    //Initialize the Wolfson codec
    //stm32_wm5102_init(FS_48000_HZ, WM5102_DMIC_IN, IO_METHOD_INTR);
    stm32_wm5102_init(FS_48000_HZ, WM5102_LINE_IN, IO_METHOD_INTR);
    while (1) {
        //Get character from internal buffer and
        //decode char string into slider float32_t
        //held in P_vals[] array
        c = TM_USART_Getc(USART6);
        if (c) {
            //Wait for header char 'H'
            if (H_found == 0) {
                if (c == 'H') {
                    H_found = 1;
                }
            } else {
                //TM_USART_Putc(USART6, c);
                if ((c >= '0' && c <= '9') || (c == '.') || (c == '-')) {
                    P_rcvd[i] = c;
                    i++;
                } else if (c == ':') {
                    P_idx = (int16_t) atoi(P_rcvd);
                    i = 0;
                    memset(P_rcvd, 0, sizeof(P_rcvd)); //clear received char array
                } else if (c == 'T') {
                    P_vals[P_idx] = (float32_t) atof(P_rcvd);
                    //For debug echo parameter value back to GUI
                    sprintf(P_tx, "%d:%1.3fT", P_idx, P_vals[P_idx]);
                }
            }
        }
    }
}
TM_USART_Puts(USART6, P_tx);
memset(P_rcvd, 0, sizeof(P_rcvd));
i = 0;
H_found = 0;
else {
    memset(P_rcvd, 0, sizeof(P_rcvd));
}
}
/* If button pressed */
if (TM_DISCO_ButtonPressed()) {
    /* Turn on leds */
    TM_DISCO_LedOn(LED_RED | LED_GREEN);
} else {
    /* Turn off leds */
    TM_DISCO_LedOff(LED_RED | LED_GREEN);
}

- When the project runs you can dynamically change six float parameters, all with their own parameter ranges
  - **Note**: Only one parameter is transmitted at a time to keep from needlessly loading the processor
  - Furthermore, the echo from STM32 back to the PC app can be turned off to further reduce processor loading
  - The maximum baud rate that the USB serial device supports is 230400 bits/s; this is now the default setting
Debug Using PC Terminal

- Another application of the serial port interface is to send debug information, normally sent to `printf()`, to a terminal program such as `CoolTerm`\(^1\).
- Here I consider a simple Keil MDK5 project that has C calling assembly.
- The result of some simple calculations done in assemble are returned to the `main()` and then formatted as a text string with `sprintf()`.
- The formatted string is then sent via USART6 through the TTL to USB serial port dongle.
- The project is located in the ZIP package.
- The project tree is:

```
A simple asm project showing the use of the serial debug
```

- The main module code is given below:

\(^1\) http://freeware.the-meiers.org/
Simple C calling ASM Example

#include "defines.h"
#include "stm32f4xx.h"
#include "tm_stm32f4_usart.h"
#include "tm_stm32f4_delay.h"
#include "tm_stm32f4_disco.h"

extern int add_asm(int16_t x1, int16_t x2, int16_t x3, int16_t x4);
extern int mac_asm(int16_t x1, int16_t x2, int16_t x3);

MAIN function

int main(void){
    int16_t x = 0;
    char my_debug[20];
    //Initialize system
    SystemInit();
    // Initialize leds on board */
    TM_DISCO_LedInit();
    // Initialize button on board */
    TM_DISCO_ButtonInit();
    // Initialize delay */
    TM_DELAY_Init();
    //Initialize USART6 at some baud, TX: PC6, RX: PC7
    // Timer error with TM_USART requires a rescale of the baud rate
    // by 231/127
    TM_USART_Init(USART6, TM_USART_PinsPack_1, (231*38400)/127); // e.g., 9600 <= 17461
    x = add_asm(11,22,33,44);// call assembly function
    sprintf(my_debug, "ADD: The answer is %d\n", x);
    TM_USART_Puts(USART6, my_debug);
    x = mac_asm(23,18,-101);// call assembly function
    sprintf(my_debug, "MAC: The answer is %d\n", x);
    TM_USART_Puts(USART6, my_debug);
    while (1)
    {
    }
}

• The assembly module is next:

; A simple ASM function to add four numbers together
Appendix A • Serial Port Control and Debugging of the Target

; Mark Wickert, February 2015

PRESERVE8 ; Preserve 8 byte stack alignment
THUMB     ; indicate THUMB code is used
AREA |.text|, CODE, READONLY ; Start of the CODE area

EXPORT add_asm
add_asm FUNCTION
  ADDSR0, R0, R1
  ADDSR0, R0, R2
  ADDSR0, R0, R3
  BX     LR         ; Return result in R0
ENDFUNC

EXPORT mac_asm
mac_asm FUNCTION
  MLA     R0, R0, R1, R2
  BX      LR         ; Return result in R0
ENDFUNC
END    ; End of file

- Sample terminal output:

![Sample terminal output](image)
LPC4088 Quick Start

A pre-built USART library for doing serial communications with the LPC4088 Quick Start needs to be identified, or the library used with STM32F4 needs to be ported. This is an open task for someone to work on.
When using the mbed capability supplied with the LPC4088, a serial port library is already supplied. Furthermore, the mbed interface also provides a UART connection through the USB debug connection. In other words, the mbed side of the LPC4088 is good to go.

Additionally, if you add in the baseboard capabilities, you have even more options. A case in point are the two potentiometers on the baseboard. They can be brought through the on-chip ADC (pins P15, P16, P16, or P17) to provide a direct analog-like control interface. There is also a five position joystick.