THIS SPEC IS OBSOLETE

Spec No: 002-04393

Spec Title: AN204393 - FM0+ Starter Kit 32-Bit Microcontroller

Replaced by: 002-05536

FM0+ Starter Kit 32-Bit Microcontroller

Target Products: **S6E1B8 Series**

This application note describes the starter kit SK-FM0-100L-S6E1B8 and how to use it.

Contents

1 Introduction

1.1 Document Purpose

This application note describes the starter kit SK-FM0-100L-S6E1B8 and how to use it.

1.2 Definitions, Acronyms, and Abbreviations

UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
LED	Light Emitting Diode
LDO	Low-Drop-Out linear voltage regulator
I/F	Interface
I2S	Integrated Interchip Sound
I/O	Input and Output
PDL	Peripheral Driver Library

1.3 Document Overview

The rest of document is organized like this:

Section 2 explains Overview and Features

Section 3 explains Software Preparation and Installation.

Section 4 explains Board Description and Connection.

Section 5 explains Firmware Architecture.

Section 6 explains Module Labs Operation.

Section 7 explains Additional Information

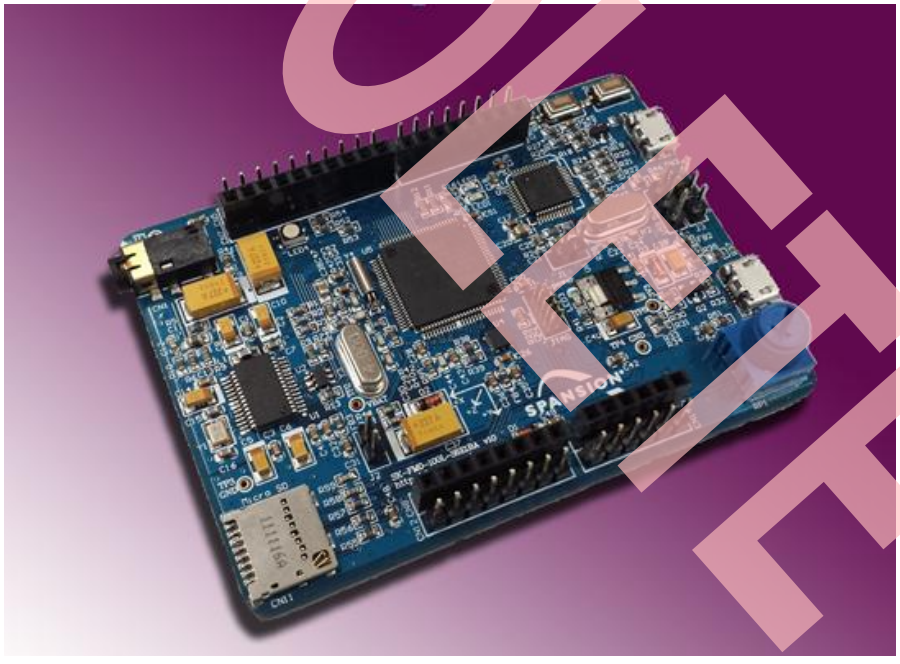
2 Overview and Features

2.1 Board Overview

The SK-FM0-100L-S6E1B8 is a Starter Kit for Cypress S6E1B8 Series microcontrollers.

The S6E1B8 Series is highly integrated 32-bit microcontrollers designed for embedded controllers aiming at low power consumption and low cost. This series has the ARM Cortex-M0+ Processor with on-chip Flash memory and SRAM, and consists of peripheral functions such as various timers, ADCs and communication interfaces (UART, CSIO, I^2C, LIN, I2S, and USB).

Figure 1. Board Overview

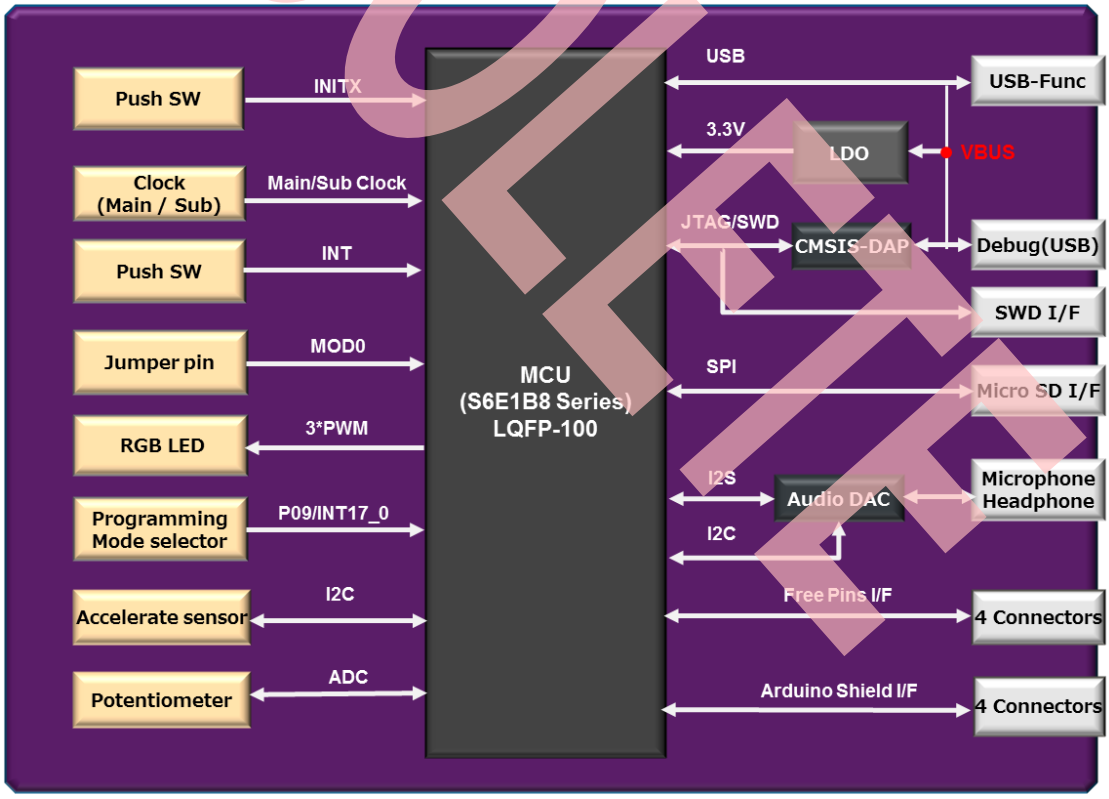


2.2 Board Features

- Cypress FM0+ S6E1B8 MCU
- CMSIS DAP interface
- JTAG interface
- Free Pins interface
- Arduino Shield interface
- USB Device (Micro type-B)
- SD card interface
- Acceleration Sensor
- Potentiometer
- Microphone/ Headphone interface
- RGB LED
- 2 x Push-button

2.3 System Block Diagram

Figure 2. System Block Diagram

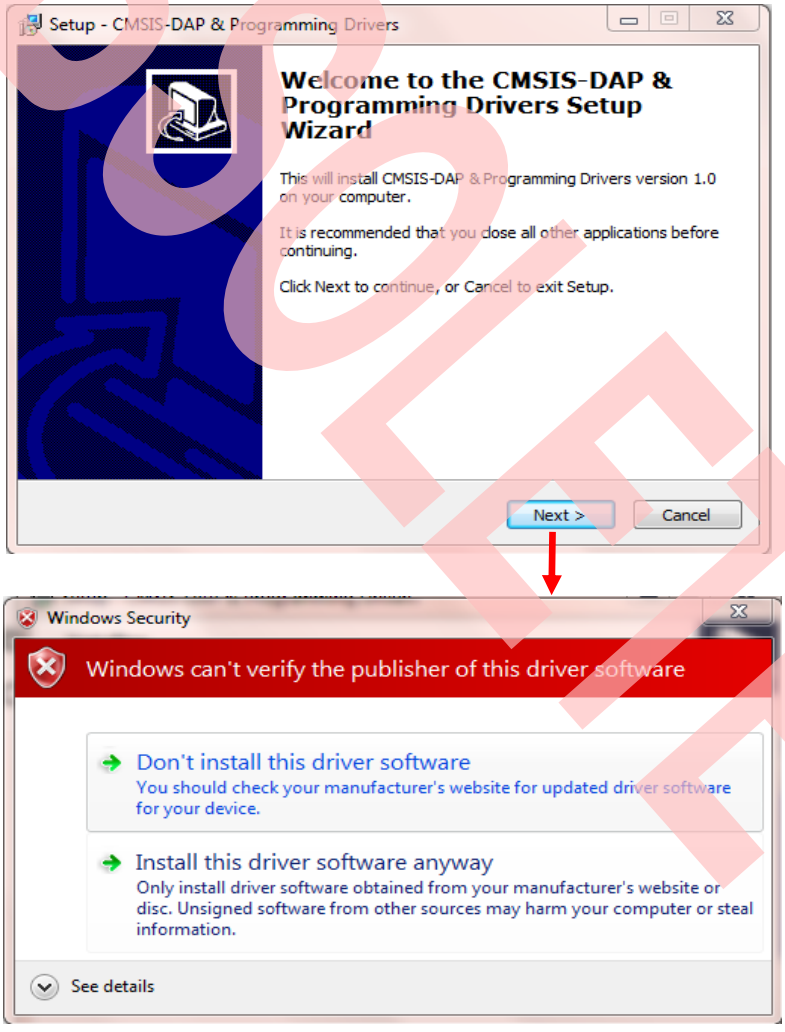


3 Software Preparation and Installation

3.1 CMSIS-DAP Driver Installation

- Please download the .zip file SK-FM0-100L-S6E1B8.vxyz.
- Open subfolder: \tools\ cmsisdap_fw_update.
- Double-click the “setup.exe” to install the driver.
- Click “**Next**” to continue.
- Select “**Install this driver software anyway**” (twice for two driver installations).
- Click the “**Finish**” button when the installation is completed.

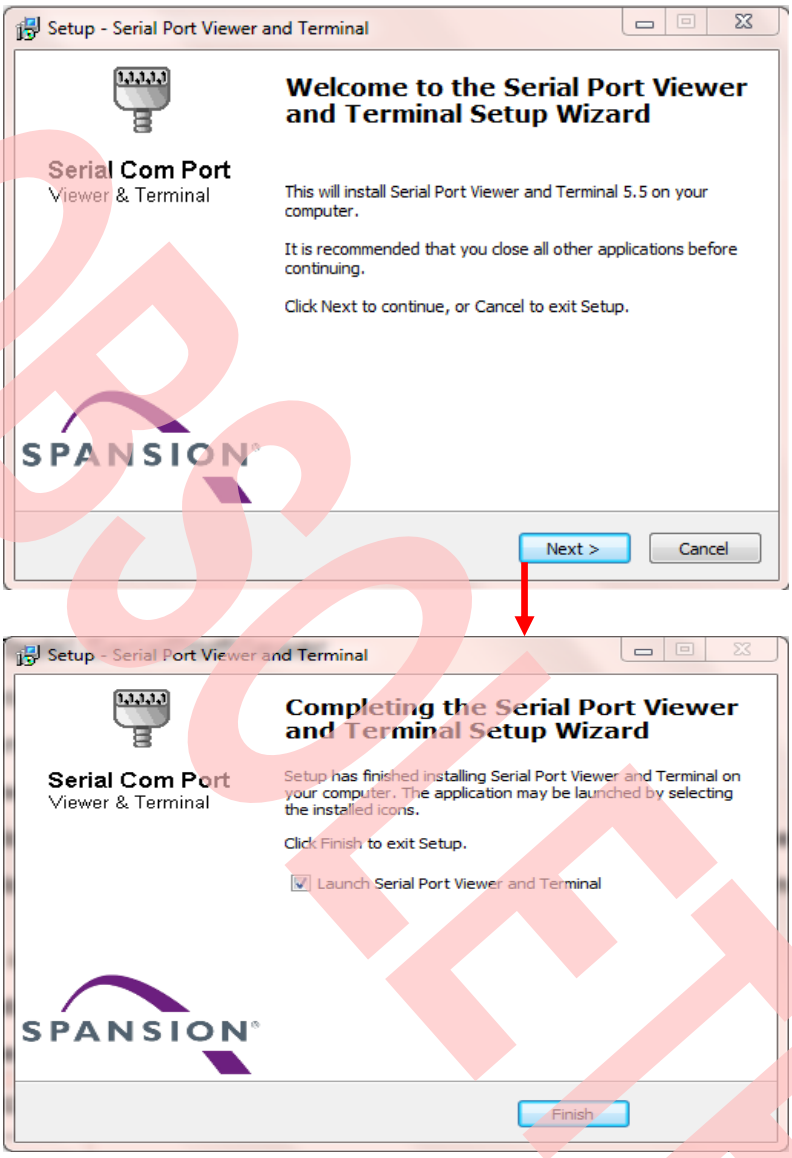
Figure 3. CMSIS-DAP Driver Installation



3.2 Virtual Communication Port Installation

- Open subfolder: Tools\ SerialPortViewer.
- Double-click “SerialPortViewerAndTerminalV5.5.exe” to install this software.
- Click “**Next**” to continue.
- Read the license agreement, and select “**I accept the agreement**”.
- Select a destination for the application or leave the default location, and click “**Next**”.
- Select a destination for the application shortcut or leave it in default location, and click “**Next**”.
- Select additional icon options and click “**Next**”.
- Select “**Install**” to start the installation.
- Check the box next to Launch Cypress Serial Port Viewer and Terminal and click “**Finish**”.

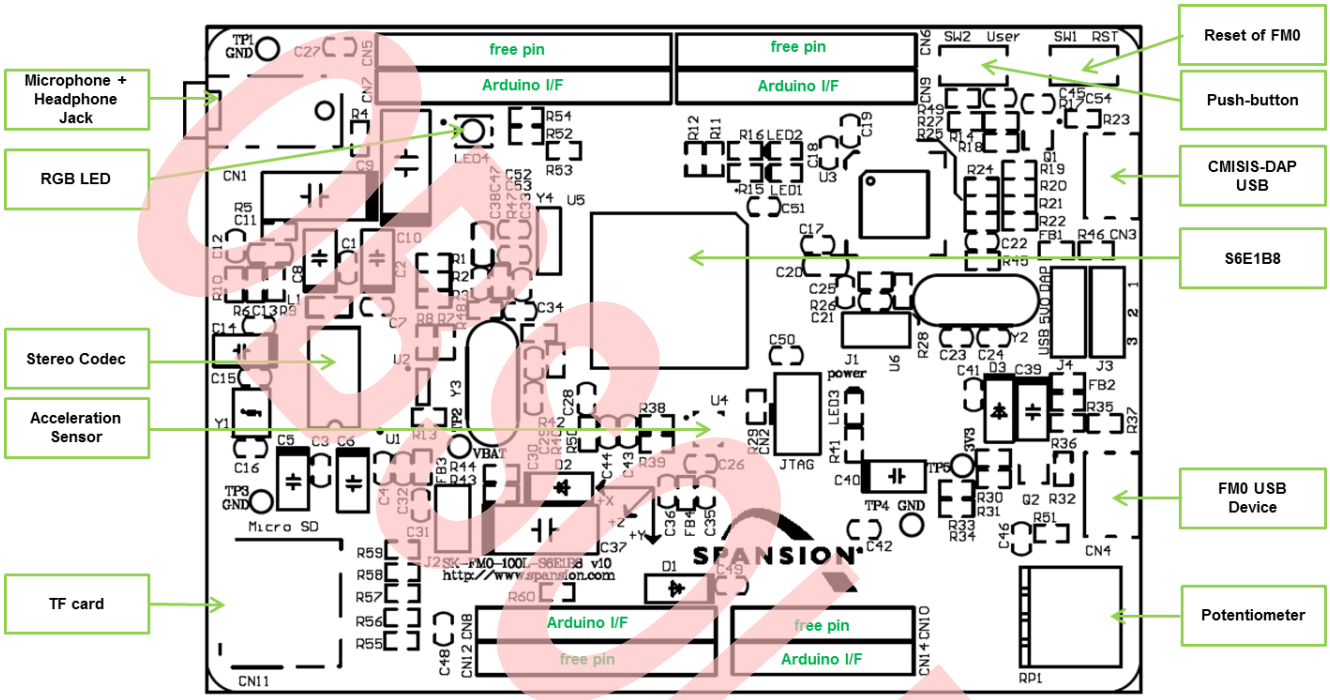
Figure 4. Virtual Communication Port Installation



4 Board Description and Connection

4.1 Hardware Description

Figure 5. Components Layout



4.2 Jumpers Description

Jumper description is shown as below:

Table 1. Jumper Description

Number	Name	Functions
J1	CMSIS-DAP MD0	Open: normal operation
		Closed: serial programming mode
J2	FM0+ MD0	Open: normal operation
		Closed: serial programming mode
J3	UART or USB programming mode selection	(1-2) UART
		(2-3) USB
J4	5V power supply source selection (please select only one power source at the same time)	(1-2): DAP power supply (CN2)
		(2-3): USB power supply (CN4)

4.3 Connectors Description

CN1 to CN11 descriptions are:

Table 2. Connectors Description

Number	Name	Functions
1	CN1	Microphone + Headphone jack
2	CN2	JTAG I/F (10pin)
3	CN3	USB Device for CMSIS-DAP
4	CN4	USB Device for FM0+ MCU
5	CN5/CN6/CN12/CN14	Pin header of free pins
6	CN7/CN8/CN9/CN10	Pin header of Arduino I/F
7	CN11	Micro SD card socket

4.4 Hardware Connection and IAR Configuration

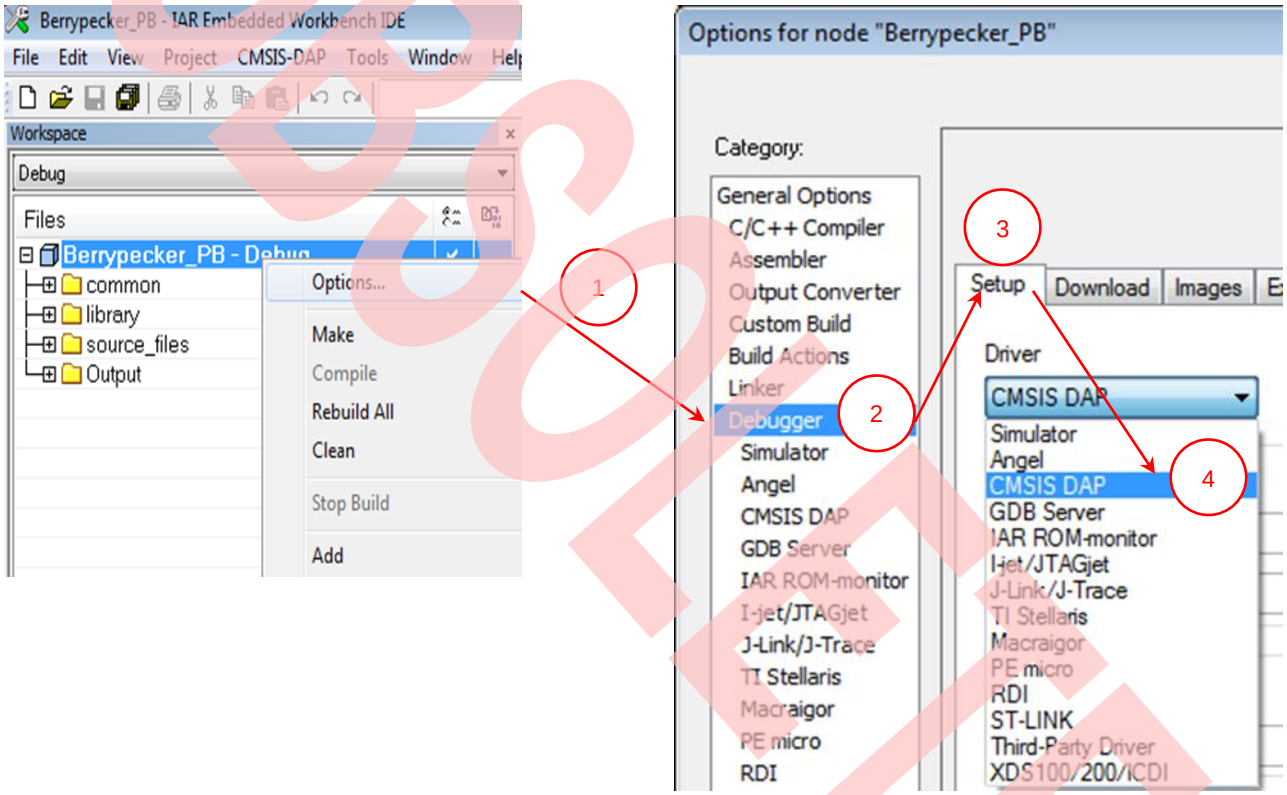
4.4.1 Power Supplies Selection

Because module lab will use CMSIS-DAP, close 1-2 of J4 jumper to select CMSIS-DAP USB power supply.

4.4.2 Select the CMSIS-DAP within IAR

- Open the project with IAR
- Right click on the project
- Select "**Options**"
- Select "**Debugger**"
- Click "**Setup**"
- Select "**CMSIS-DAP**"

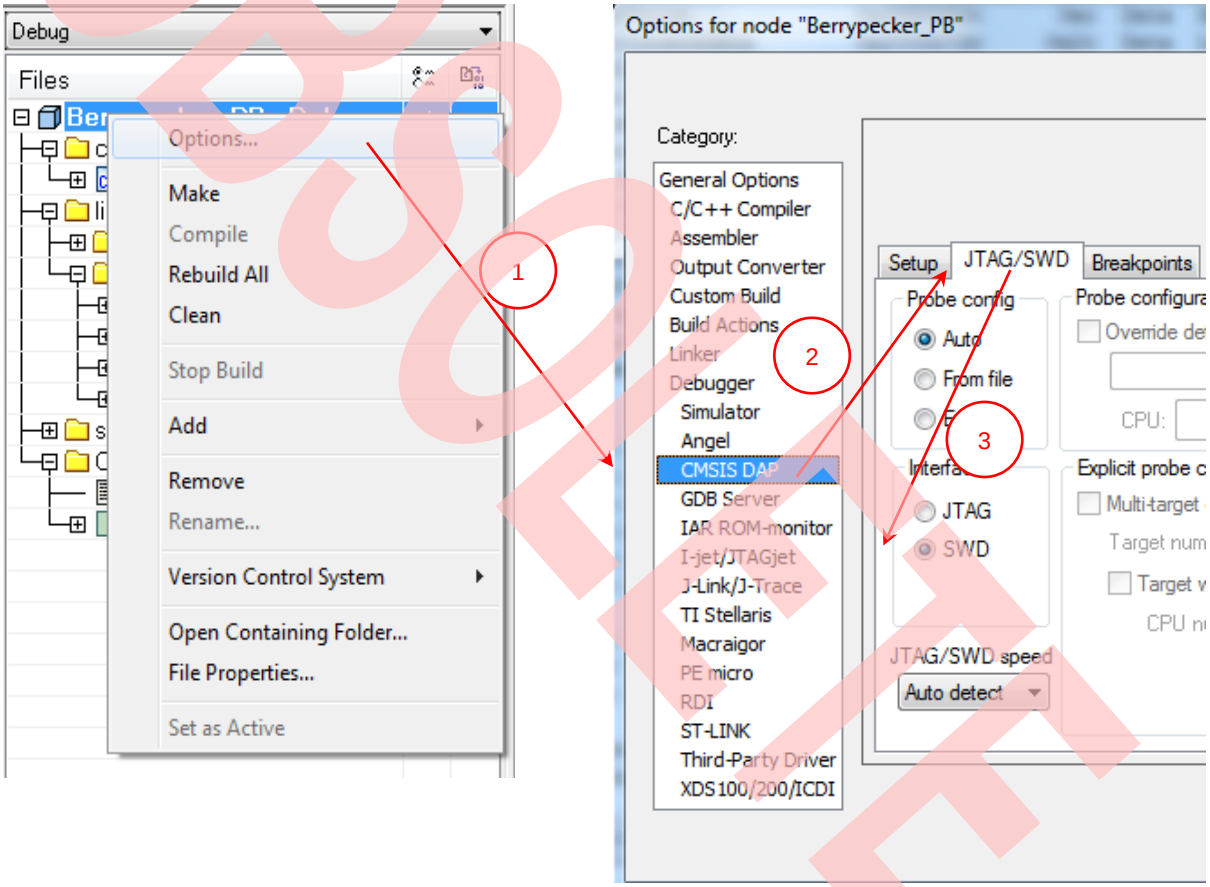
Figure 6. Select the CMSIS-DAP



4.4.3 Select SWD Interface for CMSIS-DAP

- Open the project with IAR
- Right click on the project
- Select "**Options**"
- Click "**CMSIS-DAP**"
- Click "**JTAG/SWD**"
- Click "**SWD**"

Figure 7. Select SWD Interface



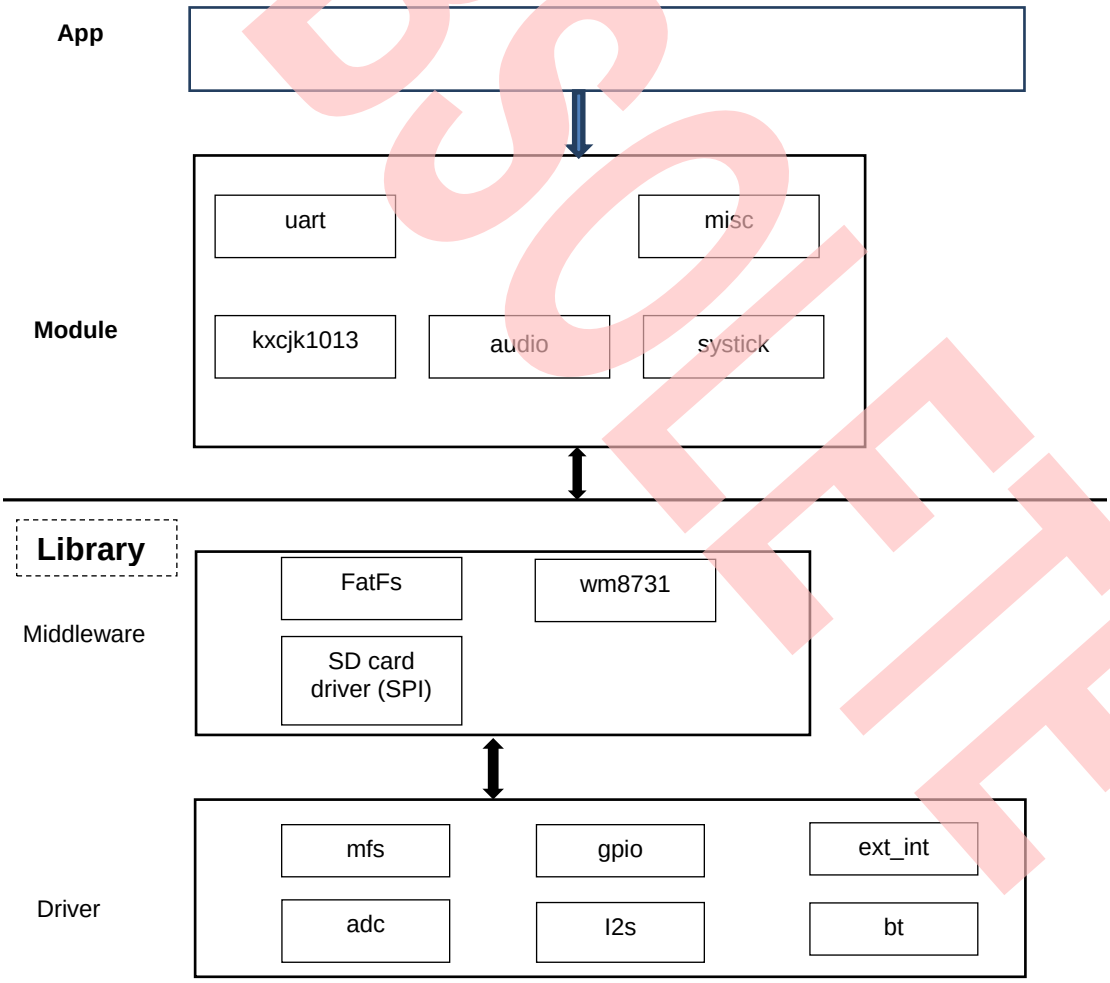
5 Firmware Architecture

5.1 Firmware Architecture Diagram

The FW subsystem architecture is shown as the Figure 8. There are 3 layers in subsystem:

1. **Library Layer:** This layer is a constant area. User must not change it. It provides all units' driving functions, such as GPIO driving function, timer driving function, UART driving function, etc. This layer is separated as 2 sub-layers: driver level and middleware level. For driver level, it directly operates the registers of the MCU. For middleware level, this layer implements adopted code (SD lib/fatFS/audio codec).
2. **Module Layer:** This layer is the highest layer of the system; it integrates the low level function and serves the system as the API (i2c, uart, audio, misc).
3. **App Layer:** The main function of the system is achieved by this layer

Figure 8. Firmware Architecture Diagram



6 Module Labs Operation

6.1 Control the RGB LED using Base Timer

6.1.1 Lab Objective

1. Learning the base principle and PWM.
2. Master the BT driver function about FM0+ driver

6.1.2 Lab Content

Use three PWM channels to light on or off RGB LED.

6.1.3 Preliminary Knowledge

1. Master the basic process of coding and debugging the program in the IAR integration development environment.
2. Understand FM0+ Peripheral Library framework

6.1.4 Preparation of Lab Equipment and Tools

1. Hardware
    a. Mini USB cable
    b. SK-FM0-100L-S6E1B8 board
    c. PC
2. Software
    a. IAR

6.1.5 Lab Principle and Instructions

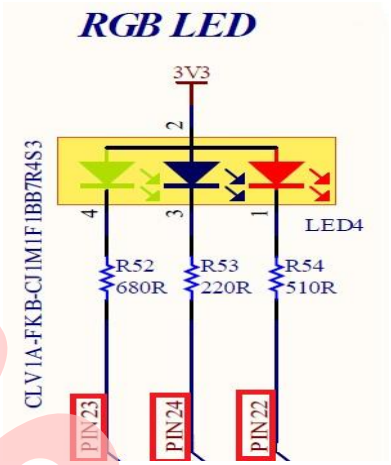
1. The function of the base timer can be set to 16-bit PWM timer. When triggered, the 16-bit PWM timer starts decrementing from the cycle set value. First, it outputs a LOW level pulse. When the 16-bit down counter matches the value set in the PWM Duty Set Register, the output inverts to the HIGH level. Then, the output inverts again to the LOW level when a counter underflow occurs. This can generate waveforms with any cycle and duty. (More details to refer to FM0+ Family Peripheral Manual Timer Part). So set PWM duty to control the total current into LED
2. The Figure 9 shows that PIN22, PIN23 and PIN24 are selected to control RGB LED.

    The PIN22 has TIOA3_1 function at the output pin of BT ch.3 TIOA.

    The PIN23 has TIOA4_1 function at the output pin of BT ch.4 TIOA.

    The PIN24 has TIOA5_1 function at the output pin of BT ch.5 TIOA.

Figure 9. RGB LED Circuit



6.1.6 Lab Steps

1. Create new project based on FM0+ Peripheral Library.
2. Edit file pdl_user. to activate PDL resource and interrupt.

Figure 10. Activate Requested BT Channels

```
// Base Timers
#define PDL_PERIPHERAL_ENABLE_BT3          PDL_ON
#define PDL_PERIPHERAL_ENABLE_BT4          PDL_ON
#define PDL_PERIPHERAL_ENABLE_BT5          PDL_ON
// GPIO header inclusion
#define PDL_PERIPHERAL_ENABLE_GPIO         PDL_ON
```

Figure 11. Activate Requested BT Channels Interrupts

```
// Base Timers
#define PDL_INTERRUPT_ENABLE_BT3           PDL_ON
#define PDL_INTERRUPT_ENABLE_BT4           PDL_ON
#define PDL_INTERRUPT_ENABLE_BT5           PDL_ON
```

3. Create new file misc.c and misc.h; add the new files to the project
4. Define macros about the requested BT channels and BT I/O (misc.h)

Figure 12. Define Requested BT Channels Macros

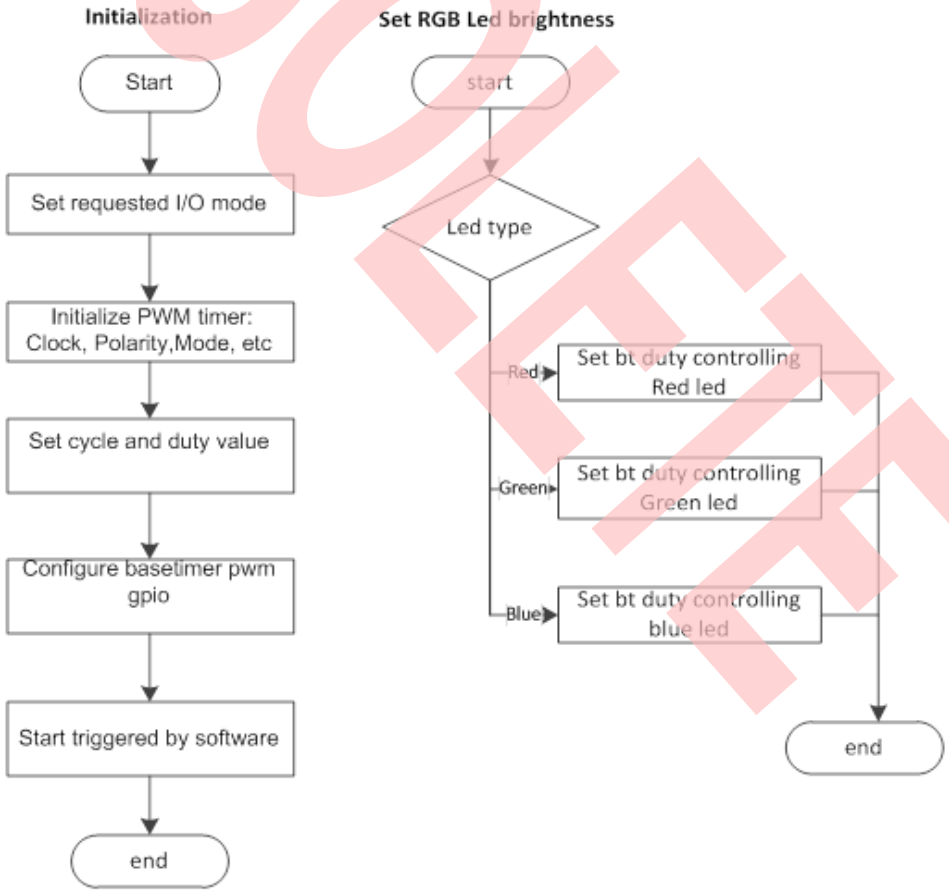
```
/* base timer - PWM channel configuration */
#define BT_LED_R        (BT3)
#define BT_LED_G        (BT4)
#define BT_LED_B        (BT5)
```

Figure 13. Define Requested BT Channels I/O Macros

```
/* base timer IO reallocation */
#define Bt_LedR_ConfigIO()   {SetPinFunc_TIOA3_1_OUT();}
#define Bt_LedG_ConfigIO()   {SetPinFunc_TIOA4_1_OUT();}
#define Bt_LedB_ConfigIO()   {SetPinFunc_TIOA5_1_OUT();}
```

5. Code BT initialization function and setting RGB LED function (misc.c). The flow chart is as below

Figure 14. BT Initialization and Setting RGB LED Flow



6. Create files sys_tick.c and sys_tick.h; add the new files to the project.
7. Code system tick initialization and interrupt function; code checking for timeout occurrence function (sys_tick.c).

Figure 15. BT Initialization and Setting RGB Led Flow

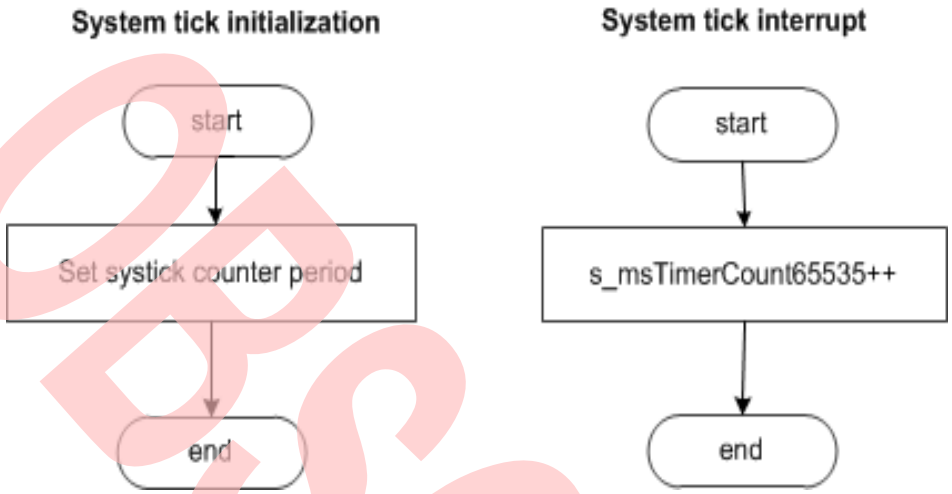
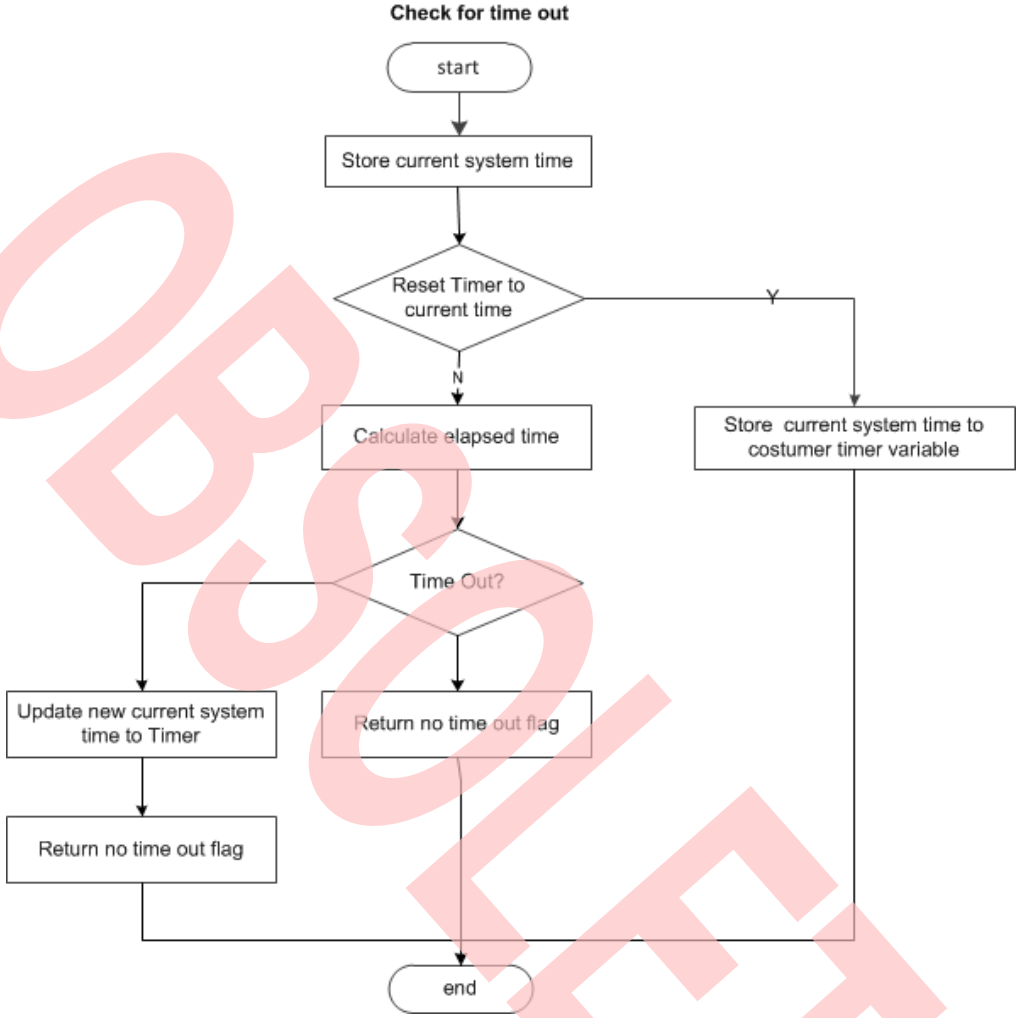


Figure 16. System Tick Timeout Function Flow



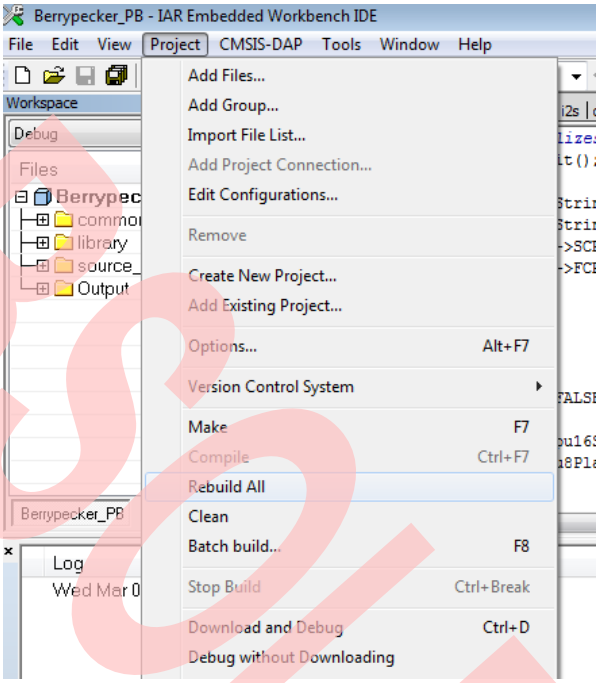
8. Create file main.c
   a. Initializes the System tick counter.
   b. Initializes the base timer for RGB LED driving.
   c. In the infinite loop
      i. RGB LED all light off when time out count equals 0.
      ii. Only red LED lights on when time out count equals 1.
      iii. Only green LED lights on when time out count equals 2.
      iv. Only blue LED lights on when time out count equals 3.
      v. Reset time out count when time out count is larger than 3.

Figure 17. Main Control Code

```
int32_t main(void)
{
   uint8_t   u8LedState = 0x00;
   uint16_t  RgbTimer;
   /* Initial system tick */
   SysTick_Init(100);
   /* Initial RGB */
   RgbLed_Init();
   TimerMax65535ms(&RgbTimer, 0x0);
   while (1)
   {
      switch(u8LedState)
      {
         case 0u:
         RgbLed_SetLumi(Led_Red, 100);    // LED R off
         RgbLed_SetLumi(Led_Green, 100);  // LED G off
         RgbLed_SetLumi(Led_Blue, 100);   // LED B off
         break;

         case 1u:
         RgbLed_SetLumi(Led_Red, 80);     // LED  R on
         RgbLed_SetLumi(Led_Green, 100);  // LED G off
         RgbLed_SetLumi(Led_Blue, 100);   // LED B off
         break;

         case 2u:
         RgbLed_SetLumi(Led_Red, 100);    // LED R off
         RgbLed_SetLumi(Led_Green, 80);  // LED G on
         RgbLed_SetLumi(Led_Blue, 100);  // LED B off
         break;

         case 3u:
         RgbLed_SetLumi(Led_Red, 100);     // LED R off
         RgbLed_SetLumi(Led_Green, 100);  // LED G off
         RgbLed_SetLumi(Led_Blue, 80);   // LED B on
         break;

         default:
            break;
      }
      if ( TIME_OUT_FLAG == TimerMax65535ms(&RgbTimer,100))
      {
         u8LedState++;
         if(u8LedState > 3)
           u8LedState = 0;
         TimerMax65535ms(&RgbTimer, 0x0);
      }
   }
}
```

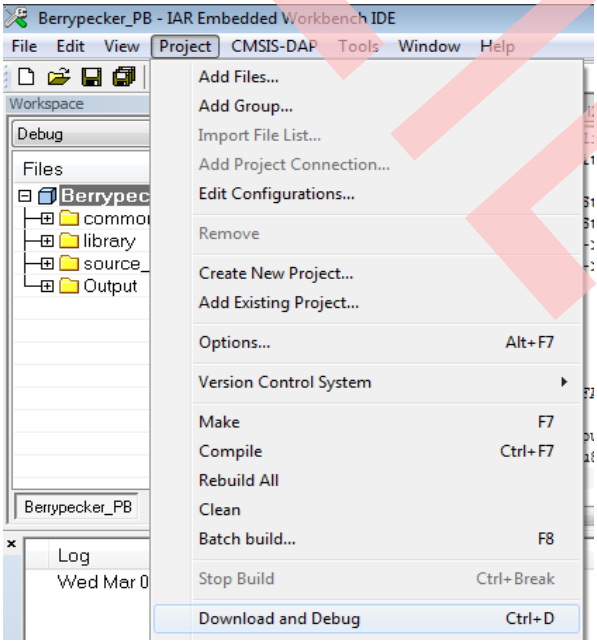
9. Connect CMSIS-DAP interface with PC to debug
10. Rebuild all.
    a. Click “**Project**”
    b. Select “**Rebuild All**”

Figure 18. Rebuild Project



11. Download and debug
    a. Click “Project”
    b. Select “Download and Debug”

Figure 19. Download and Debug



12. Watch RGB LED.
    a. RGB LED lights off.
    b. After some time, only red LED lights on.
    c. After some time, only green LED lights on.
    d. After some time, only blue LED lights on.
    e. After some time, turn to a

6.2 UART Receive/Send Data with CMSIS-DAP

6.2.1 Lab Objective

1. Understand multi-function serial interface module.
2. Study UART communication and program
3. Master the UART driver function about FM0+ PDL.

6.2.2 Lab Content

Receive the character from Cypress Serial Port Viewer and Terminal; and send to display on Terminal.

6.2.3 Preliminary Knowledge

1. Master the basic process of coding and debugging the program in the IAR integration development environment.
2. Understand FM0+ Peripheral Library framework.
3. Understand the serial bus.

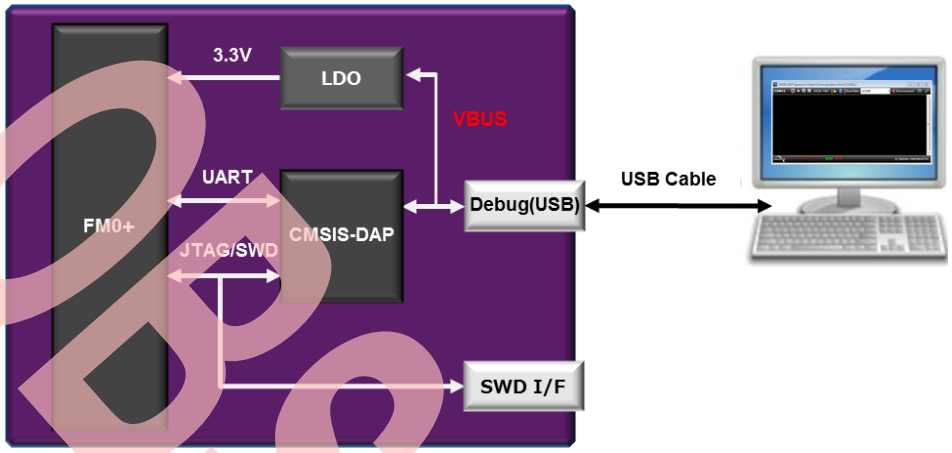
6.2.4 Preparation of Lab Equipment and Tools

1. Hardware
    a. Mini USB cable
    b. SK-FM0-100L-S6E1B8 board
    c. PC
2. Software
    a. IAR
    b. Cypress Serial Port Viewer and Terminal

6.2.5 Lab Principle and Instructions

1. The Figure 20 shows that CMSIS-DAP module can receive/send characters from/to PC and then forwards data to FM0+ with UART communication

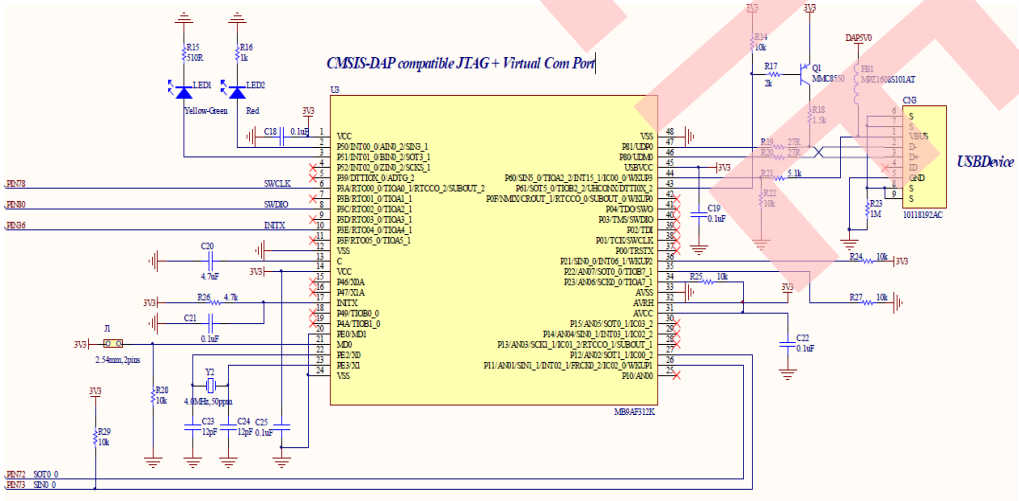
Figure 20. CMSIS-DAP Block



2. The multi-function serial interface has the following characteristics
   a. UART0 (Asynchronous normal serial interface)
   b. UART1 (Asynchronous multi-processor serial interface)
   c. CSIO (Clock synchronous serial interface) (SPI can be supported)
   d. LIN(LIN bus interface)
   e. I2C (I2C bus interface)

   Firstly, set multi-function serial interface mode to switch UART mode.
3. UART is a general-purpose serial data communications interface for asynchronous communications (start/stop synchronization) with external devices. It supports a bi-directional communications function (normal mode) and a master/slave type communications function (multi-processor mode: both master and slave modes supported). It also has transmitted/received FIFO (More details to refer to FM0+ Family Peripheral Manual Communication Macro Part).
4. The Figure 21 shows that SOT0_0 and SIN0_0 are selected to communicate with CMSIS-DAP.

Figure 21. CMSIS-DAP Compatible JTAG + Virtual Com Port Circuit



6.2.6 Lab Step

1. Open the project based on the last lab.
2. Edit file pdl_user.h to activate PDL resource and interrupt

Figure 22. Activate Requested MFS Channel

```
// Multi Function Serial Interfaces
#define PDL_PERIPHERAL_ENABLE_MFS0          PDL_ON
```

Figure 23. Activate Requested MFS Channel Interrupts

```
// Multi Function Serial Interfaces
#define PDL_INTERRUPT_ENABLE_MFS0           PDL_ON
```

3. Create new file uart.c and uart.h; add the new files to the project.
4. Define macros about the requested UART I/O and channel(uart.c)

Figure 24. Define Requested UART Channel I/O Macros

```
#define InitUartIo()    {SetPinFunc_SIN0_0();SetPinFunc_SOT0_0();}
```

5. Code UART initialization function and Receive/Transmit function (uart.c). The flow chart is as below.

Figure 25. UART Initialization and Receive Interrupt Flow

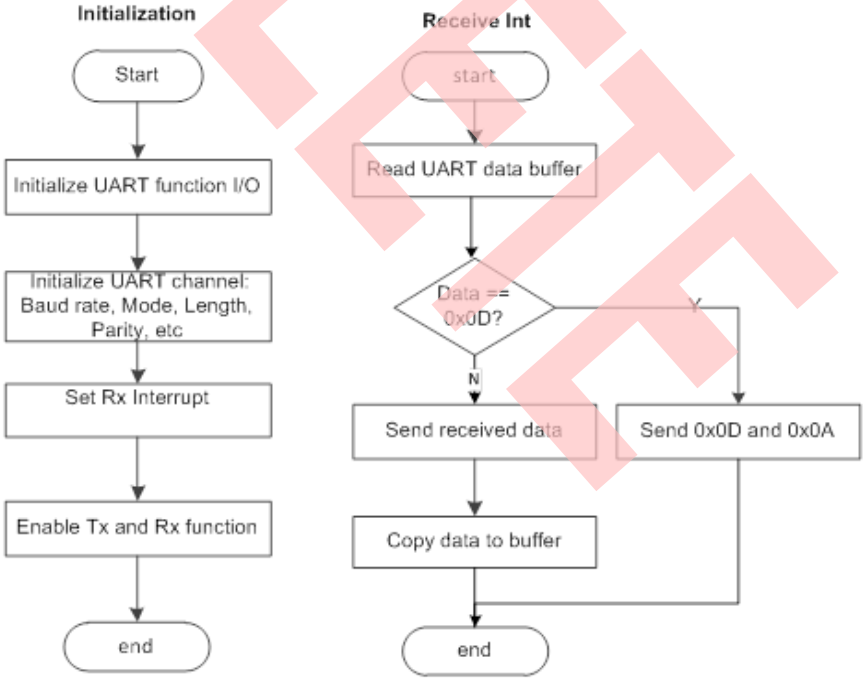
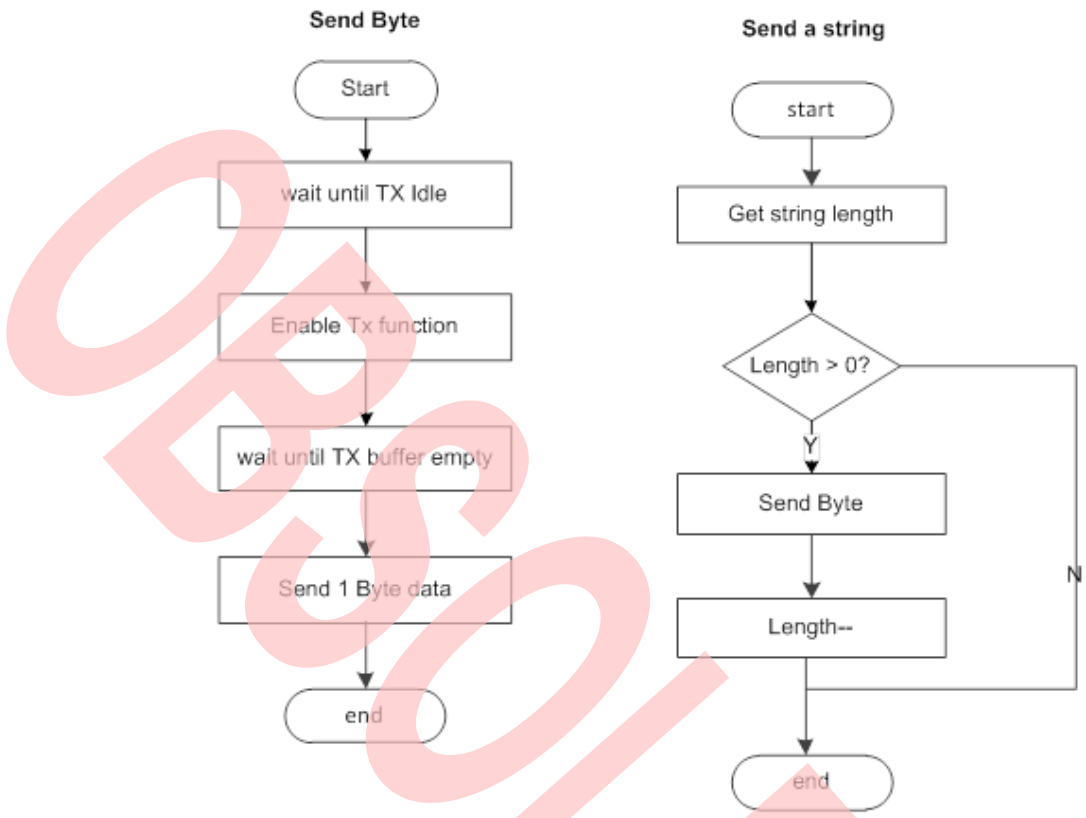


Figure 26. UART Send Byte and String Flow



13. Create new file main.c to replace the old one.

a. Initializes Uart.

b. In the infinite loop

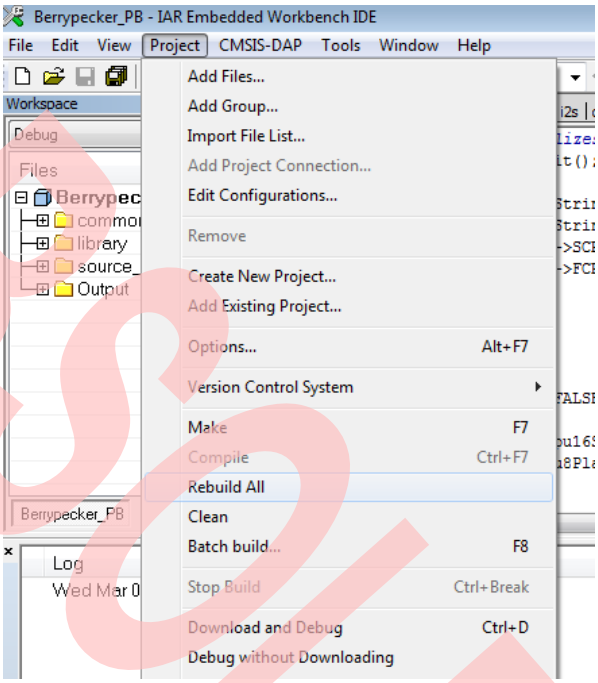
i. If UART receiving counter overflow, reset counter.

Figure 27. Main Control Code

```
int32_t main(void)
{
   /* Initial uart function*/
   UartInit();

   while (1)
   {
    /* If UART receiving counter overflow, reset counter.*/
       if ( stcUartRxBuf.RxCnt >= (UARTMAXBUFFRTSIZE-1))
       {
          stcUartRxBuf.RxCnt = 0;
       }
   }
}
```

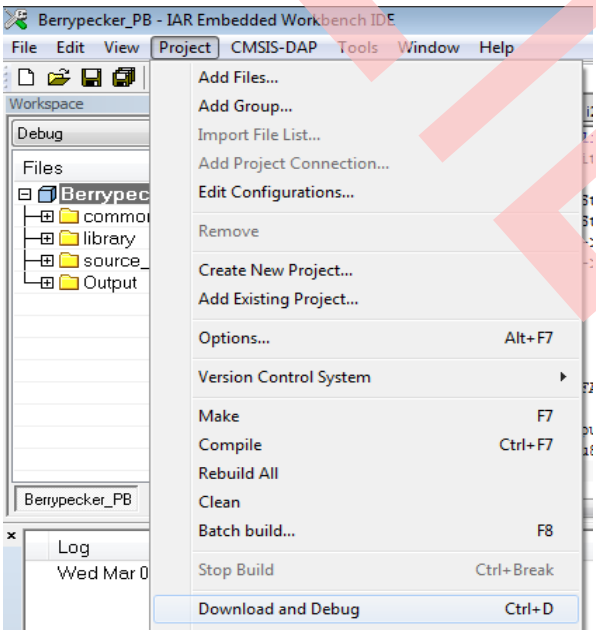
14. Connect CMSIS-DAP interface with PC to debug
15. Rebuild all.
    a. Click “**Project**”
    b. Select “**Rebuild All**”

Figure 28. Rebuild Project



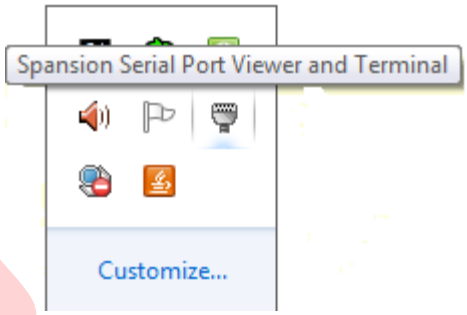
16. Download and debug
    a. Click “**Project**”
    b. Select “**Download and Debug**

Figure 29. Download and Debug



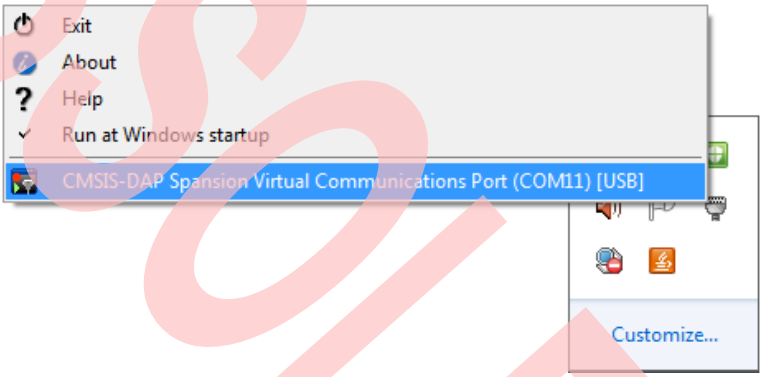
17. Open Cypress Serial Port Viewer and Terminal through icon in task menu or desktop shortcut

Figure 30. Cypress Serial Port Viewer and Terminal



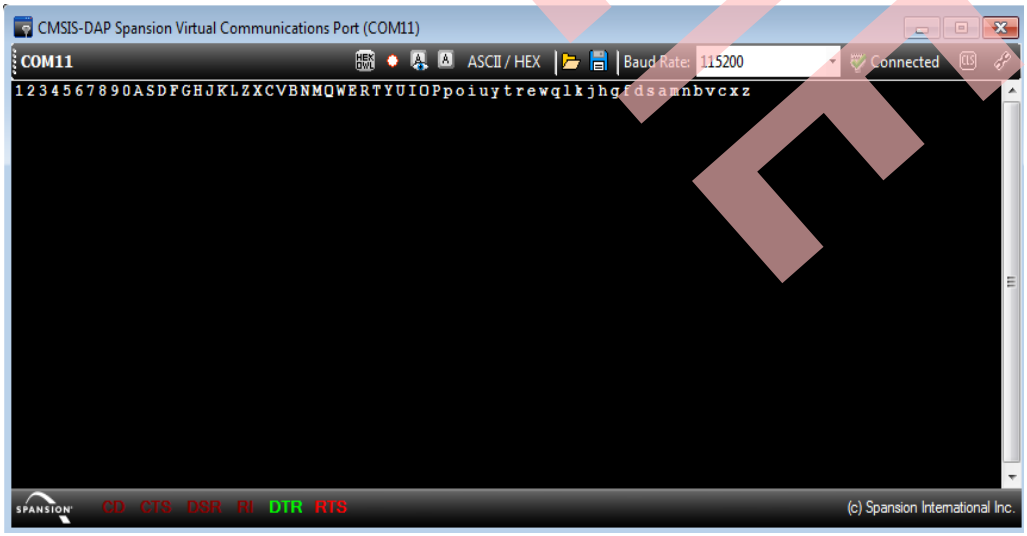
18. Select the virtual COM of CMSIS-DAP

Figure 31. CMSIS-DAP Cypress Virtual Communications Port



19. Select the baud rate “115200”, and Click the “disconnect” above the green arrow to connect
20. Click Cypress Serial Port Viewer and Terminal; press any key of PC Key Board, and check it

Figure 32. Cypress Serial Port Viewer and Terminal Display



6.3 Measure the Potentiometer Analog Voltage

6.3.1 Lab Objective

1. Familiar with FM0+ 12-bit A/D controller and the corresponding register.
2. Master the ADC driver function about FM0+ PDL.

6.3.2 Lab Content

Use ADC to measure potentiometer analog input voltage and then send sample value to display on Cypress Serial Port Viewer and Terminal.

6.3.3 Preliminary Knowledge

1. Master the basic process of coding and debugging the program in the IAR integration development environment.
2. Understand FM0+ Peripheral Library framework.
3. Familiar with A/D interface and principle.

6.3.4 Preparation of Lab Equipment and Tools

4. Hardware
   a. Mini USB cable
   b. SK-FM0-100L-S6E1B8 board
   c. PC
5. Software
   a. IAR
   b. Cypress Serial Port Viewer and Terminal

6.3.5 Lab Principle and Instructions

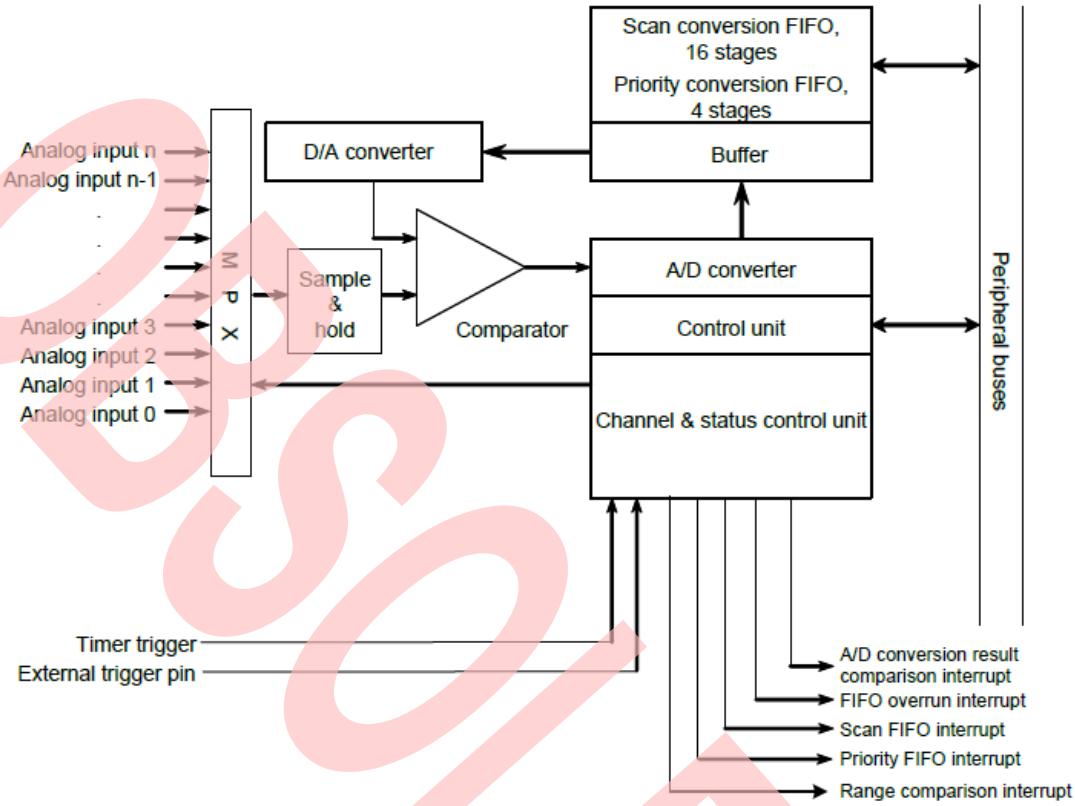
1. The 12-bit A/D converter is a function that converts analog input voltages into 12-bit digital values using a type of the RC Successive Approximation Register.

   Features of 12-bit A/D converter:

- 12-bit resolution
- Converter using a type of RC Successive Approximation Register with sample and hold circuits
- Two sampling times selectable for each input channel
- Two conversion mode: scan conversion and Priority conversion
- FIFO function
- Changeable A/D conversion data placement
- The A/D conversion result comparison function is available.
- Range comparison function
- DMA transfer triggered by an interrupt request

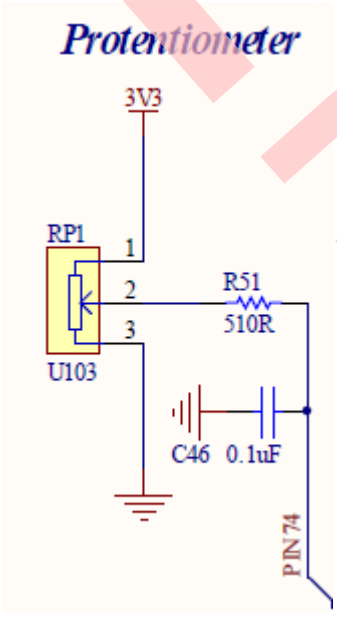
2. The Figure 33 shows 12-bit A/D converter block diagram

Figure 33. ADC Block Diagram



3. The Figure 34 shows that PIN74 is selected to measure voltage. This pin corresponds to AN19.

Figure 34. Potentiometer Circuit



6.3.6 Lab Step

1. Open the project based on the last lab.
2. Edit file pdl_user.h below to activate PDL resource and interrupt.

Figure 35. Activate Requested ADC Channels

```
// ADC
#define PDL_PERIPHERAL_ENABLE_ADC0          PDL_ON
```

Figure 36. Activate Requested ADC Channel Interrupts

```
// ADC
#define PDL_INTERRUPT_ENABLE_ADC0           PDL_ON
```

3. Define macros about the requested ADC I/O and channel(misc.h)

Figure 37. Define Requested ADC Channel I/O Macros

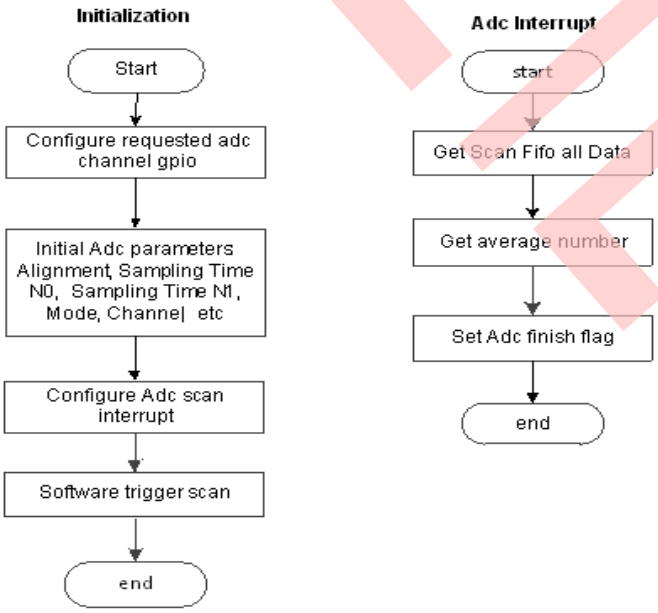
```
/* adc IO configuration for potentiometer volume control */
#define Adc_VolumeConfigIO() {SetPinFunc_AN19();}
```

Figure 38. Define Requested ADC Channel Macros

```
/* adc channel configuration for audio volume control */
#define VOLUME_ADC          (ADC0)
```

4. Code ADC initialization and interrupt function (misc.c and misc.h). The flow chart is as below

Figure 39. ADC Initialization and Interrupt Flow



5. Create new file main.c to replace the old one.
   a. Initializes the System tick counter.
   b. Initializes the ADC to measure voltage.
   c. Initializes the UART channel 0
   d. In the infinite loop
      i. Get ADC sample value
      ii. Send data by UART Channel 0
      iii. If UART buffer counter overflow, clear counter.

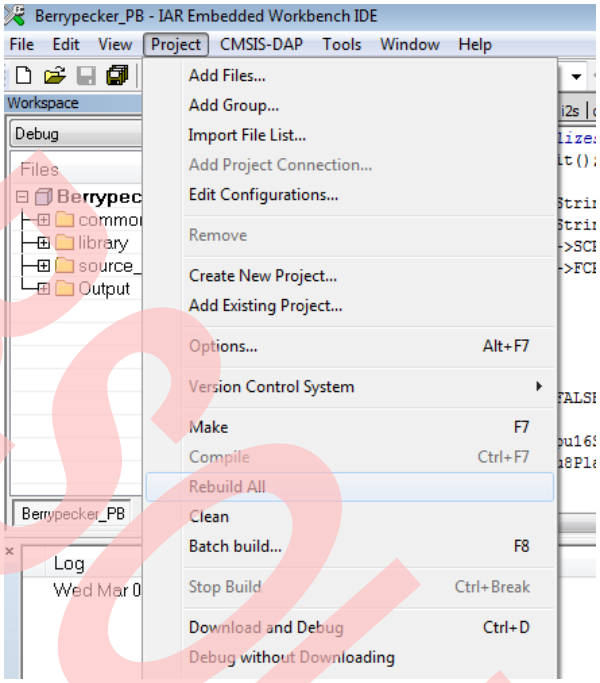
Figure 40. Main Control Code

```
int32_t main(void)
{
    uint16_t u16VolumeAdcTimer;
    uint16_t u16VolAdcData = 0;

    /* Initial system tick */
    SysTick_Init(100);
    /* Initial uart function*/
    UartInit();
    /* Initial adc function for potentiometer value capture */
    Volume_Adc_Init();

    TimerMax65535ms(&u16VolumeAdcTimer, 0x0);
    while (1)
    {
        if ( TIME_OUT_FLAG == TimerMax65535ms(&u16VolumeAdcTimer,100))
        {
            TimerMax65535ms(&u16VolumeAdcTimer, 0x00);
            u16VolAdcData = Volume_Adc_GetVal();
            UartWriteString("Adc Smaple value: ");
            UartSendAscii(3, u16VolAdcData);
            UartWriteString("\r\n\0");
        }
    }
}
```

6. Connect CMSIS-DAP interface with PC to debug

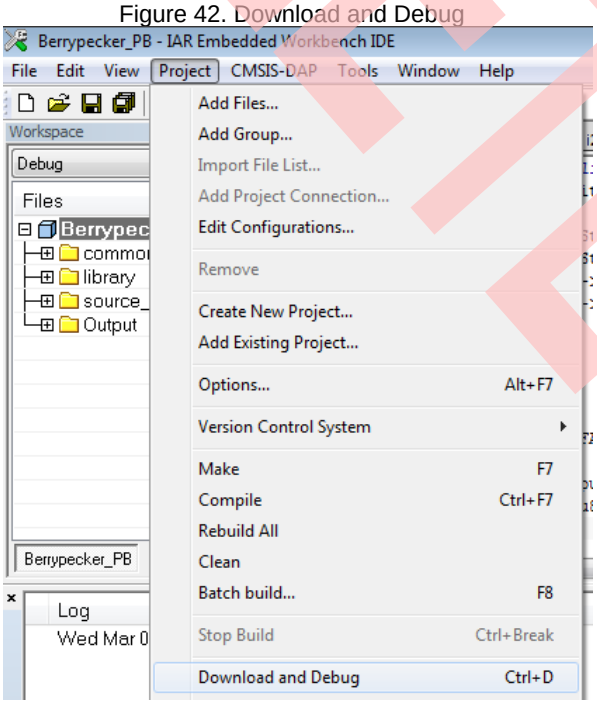
7. Rebuild all.
   a. Click “**Project**”
   b. Select “**Rebuild All**”

Figure 41. Rebuild Project



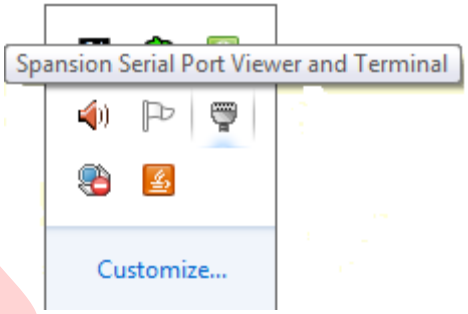
8. Download and debug
   a. Click “**Project**”
   b. Select “**Download and Debug**”

Figure 42. Download and Debug



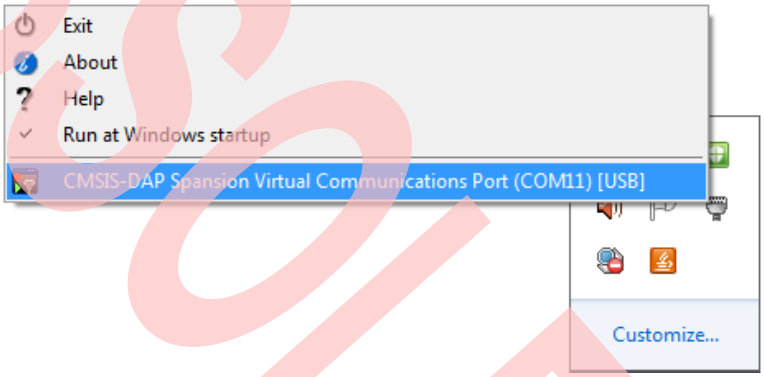
9. Open Cypress Serial Port Viewer and Terminal through icon in task menu or desktop shortcut

Figure 43. Cypress Serial Port Viewer and Terminal



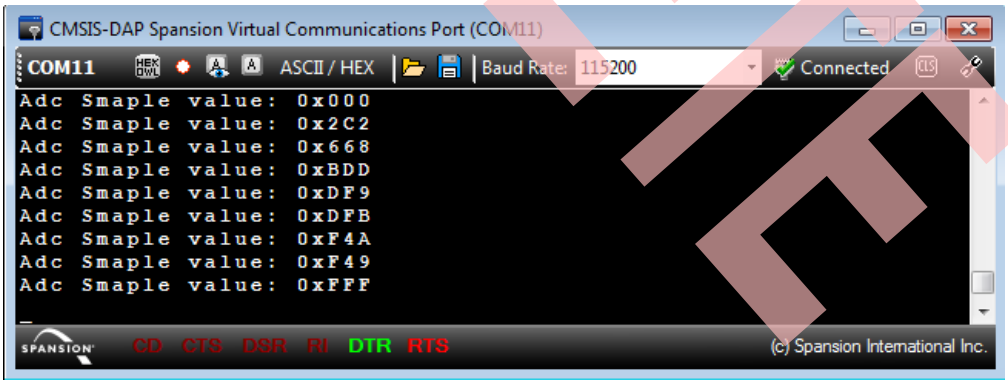
10. Select the virtual COM of CMSIS-DAP

Figure 44. CMSIS-DAP Cypress Virtual Communications Port



11. Select the baud rate “115200”, and Click the “**disconnect**” above the green arrow to connect

Figure 45. Virtual Communications Port Display



12. Check the output in Cypress Serial Port Viewer and Terminal; see if the value is equal to DC power display

6.4 Measure Acceleration with KXCJK-1013 Accelerometer

6.4.1 Lab Objective

1. Understand multi-function serial interface module.
2. Study I2C communication and program
3. Master the I2C driver function about FM0+ PDL.

6.4.2 Lab Content

Use KXCJK-1013 accelerometer to accelerate and then send value to display on Cypress Serial Port Viewer and Terminal.

6.4.3 Preliminary Knowledge

1. Master the basic process of coding and debugging the program in the IAR integration development environment.
2. Understand FM0+ Peripheral Library framework.
3. Understand the I2C bus protocol.

6.4.4 Preparation of Lab Equipment and Tools

1. Hardware
   a. Mini USB cable
   b. SK-FM0-100L-S6E1B8 board
   c. PC
2. Software
   a. IAR
   b. Cypress Serial Port Viewer and Terminal

6.4.5 Lab Principle and Instructions

1. The KXCJK is a tri-axis ±2g, ±4g or ±8g silicon micro-machined accelerometer. The sense element is fabricated using Kionix's proprietary plasma micromachining process technology. Acceleration sensing is based on the principle of a differential capacitance arising from acceleration-induced motion of the sense element, which further uses common mode cancellation to decrease errors from process variation, temperature, and environmental stress.

   The Kionix KXCJK digital accelerometer has the ability to communicate on the I2C digital serial interface bus. This allows for easy system integration by eliminating analog-to-digital converter requirements and by providing direct communication with system micro-controllers.

2. I2C is a two-wire serial interface that contains a Serial Clock (SCL) line and a Serial Data (SDA) line. SCL is a serial clock that is provided by the Master, but can be held low by any Slave device, putting the Master into a wait condition. SDA is a bi-directional line used to transmit and receive data to and from the interface. Data is transmitted MSB (Most Significant Bit) first in 8-bit per byte format, and the number of bytes transmitted per transfer is unlimited. The I2C bus is considered free when both lines are high. The I2C interface is compliant with high-speed mode, fast mode and standard mode I2C standards

3. KXCJK I2C Data Transfer Sequences

Figure 46. I2C Terms

Term	Definition
S	Start Condition
Sr	Repeated Start Condition
SAD	Slave Address
W	Write Bit
R	Read Bit
ACK	Acknowledge
NACK	Not Acknowledge
RA	Register Address
Data	Transmitted/Received Data
P	Stop Condition

Figure 47. KXCJK I2C Data Transfer Sequences

Sequence 1. The Master is writing one byte to the Slave.

Master	S	SAD + W		RA		DATA		P
Slave			ACK		ACK		ACK	

Sequence 2. The Master is writing multiple bytes to the Slave.

Master	S	SAD + W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Sequence 3. The Master is receiving one byte of data from the Slave.

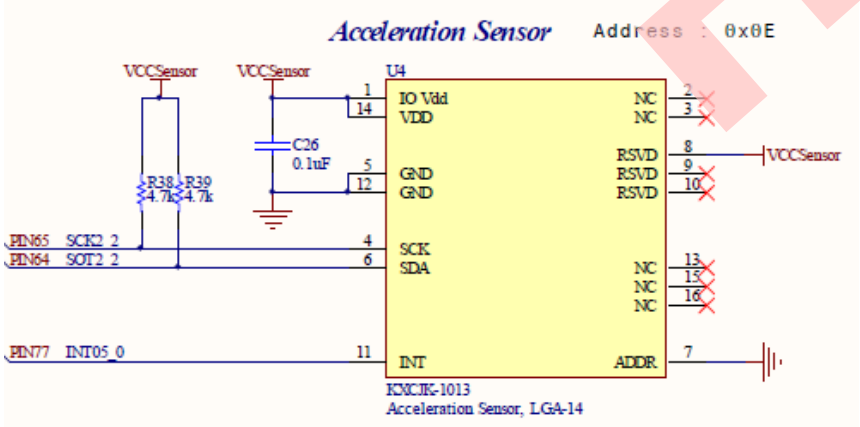
Master	S	SAD + W		RA		Sr	SAD + R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Sequence 4. The Master is receiving multiple bytes of data from the Slave.

Master	S	SAD + W		RA		Sr	SAD + R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

4. The Figure 6 41 shows that SOT2_2 and SCK2_2 are selected to communicate between FM0+ MCU and KXCJK-1013. The KXCJK's Slave Address is comprised of a programmable part and a fixed part, which allows for connection of multiple KXCJK's to the same I2C bus. The Slave Address associated with the KXCJK is 000111X, where the programmable bit, X, is determined by the assignment of ADDR (pin 7) to GND or IO_Vdd. So the Slave Address associated with the KXCJK is 0001110.

Figure 48. Acceleration Sensor Circuit



6.4.6 Lab Step

1. Open the project based on the last lab.
2. Edit file pdl_user.h to activate PDL resource and deactivate interrupt

Figure 49. Activate Requested I2C Channels

```
#define PDL_PERIPHERAL_ENABLE_MFS2          PDL_ON
```

Figure 50. Deactivate Requested I2C Channel Interrupts

```
#define PDL_INTERRUPT_ENABLE_MFS2           PDL_OFF
```

3. Create new files acc_i2c.c and acc_i2c.h; add the files to the project.
4. Define macros about the requested ADC I/O, channel and speed (acc_i2c.h).

Figure 51.Define Requested I2C Channel I/O Macros

```
#define Acc_ConfigI2cIO() {SetPinFunc_SOT2_2();SetPinFunc_SCK2_2();}
```

Figure 52. Define Requested I2C Channel Macros

```
/*  Mfs channel configuration */
#define ACC_I2C_CH          (I2C2)
```

```
/*  Mfs channel configuration */
#define ACC_I2C_CH          (I2C2)
```

5. Code I2C initialization function and Read/Write function (acc_i2c.c). The flow chart is as below.

Figure 54. Accelerator I2C Initialization Flow

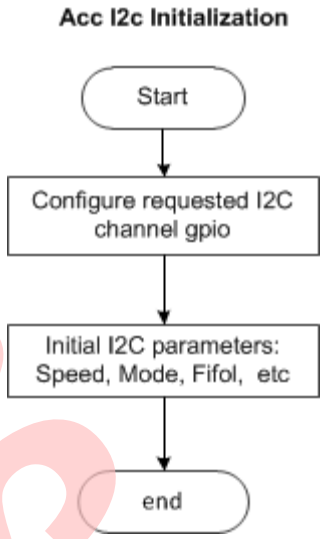


Figure 55. Accelerator Read Register Flow

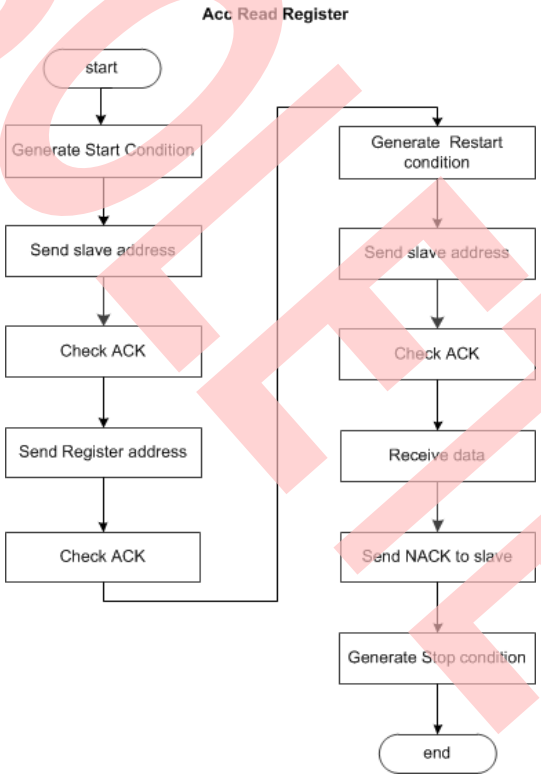


Figure 56.Accelerator Write Register Flow

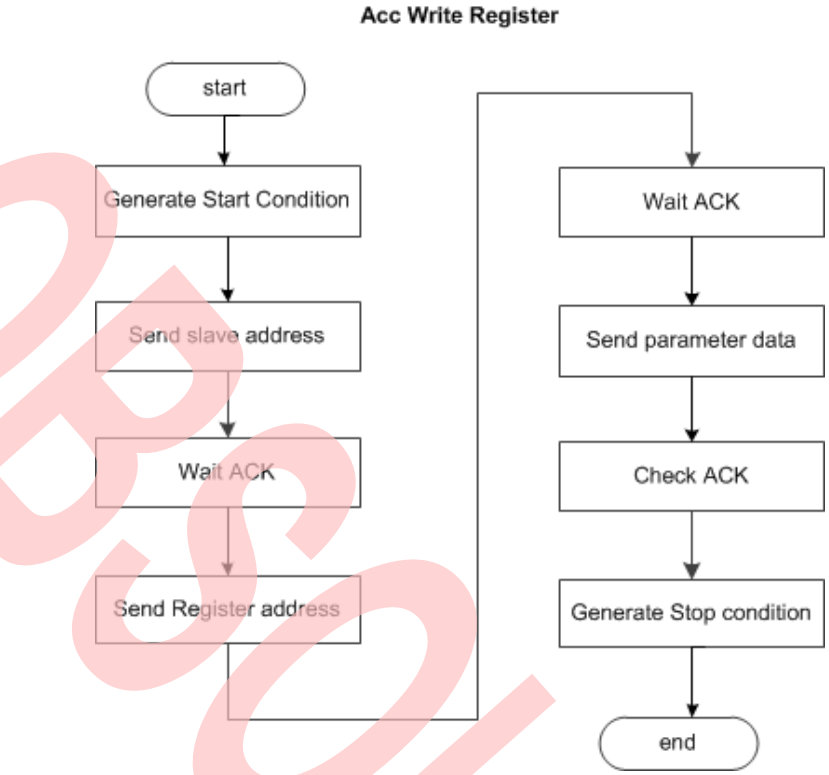
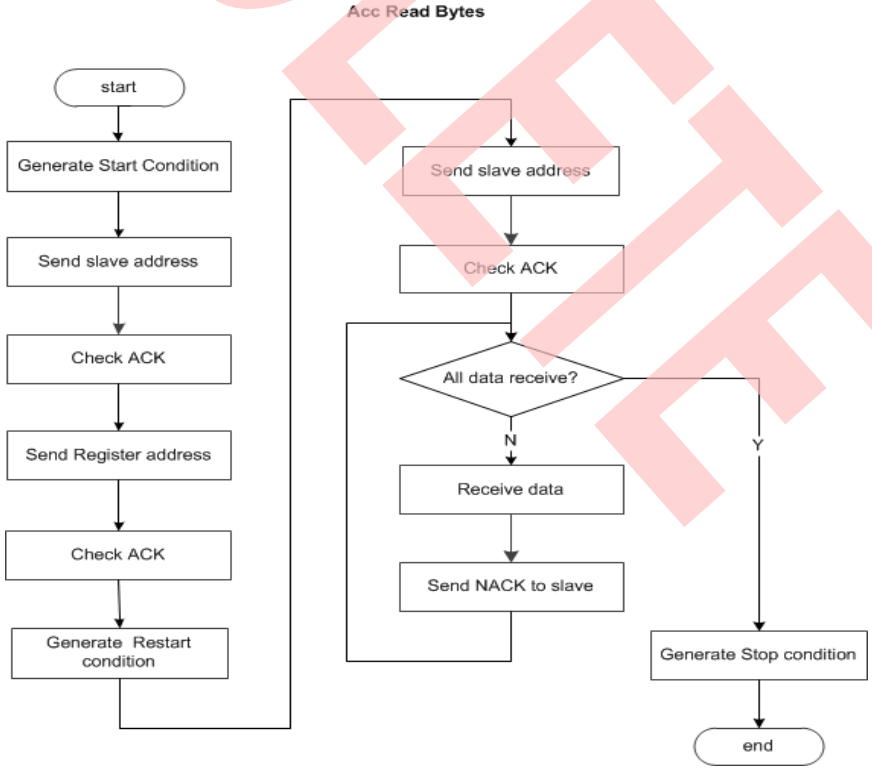
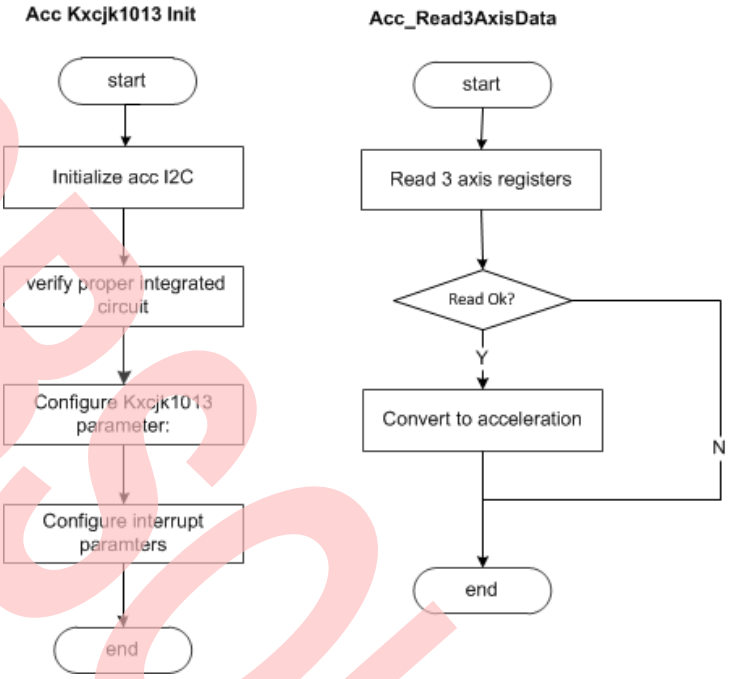


Figure 57. Accelerator Read Register Flow



6. Create new files acc_kxcjk1013.c and acc_kxcjk1013.h; add the files to the project.
7. Code acc_kxcjk1013 initialization function and Read registers function (acc_i2c.c). The flow chart is as below

Figure 58.Accelerator Initialization and Reading 3 Axis Registers Flow



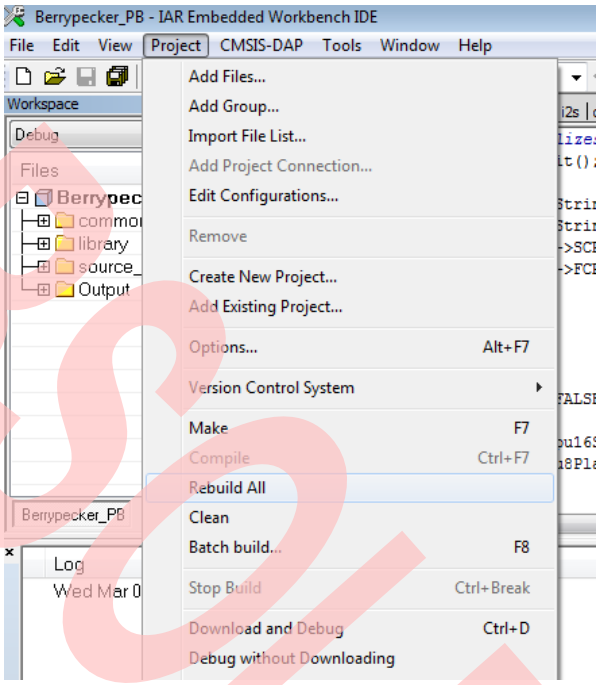
8. Create new file main.c to replace the old one.
   a. Initializes the System tick counter.
   b. Initializes the UART channel 0
   c. Initializes the Kxcjk1013 to measure acceleration.
   d. In the infinite loop
      i. Read Kxcjk1013 data
      ii. Convert data to acceleration.
      iii. Send to PC

Figure 59. Main Control Code

```
int32_t main(void)
{
    uint16_t AccSensorTimer;
    stc_acc_axis_value_t stcAccVal = {0};
    uint8_t u8Str[3][10] = {0};

    /* Initial system tick */
    SysTick_Init(100);
    /* Initial uart*/
    UartInit();
    /* Initial Kxcjk1013 */
    if(Ok != Acc_Kxcjk1013_Init())
    {
        while(1);
    }
    /*Reset time counter*/
    TimerMax65535ms(&AccSensorTimer, 0x0);
    while (1)
    {
        if ( TIME_OUT_FLAG == TimerMax65535ms(&AccSensorTimer,100))
        {
            TimerMax65535ms(&AccSensorTimer, 0x00);
            Acc_Read3AxisData(&stcAccVal);
            Float2Char(u8Str[0], stcAccVal.f32X, 1, 3);
            Float2Char(u8Str[1], stcAccVal.f32Y, 1, 3);
            Float2Char(u8Str[2], stcAccVal.f32Z, 1, 3);

            UartWriteString("     X axis is: ");
            UartWriteString(u8Str[0]);
            UartWriteString("     Y axis is: ");
            UartWriteString(u8Str[1]);
            UartWriteString("     Z axis is: ");
            UartWriteString(u8Str[2]);
            UartWriteString("\r\n\0");
        }
    }
}
```

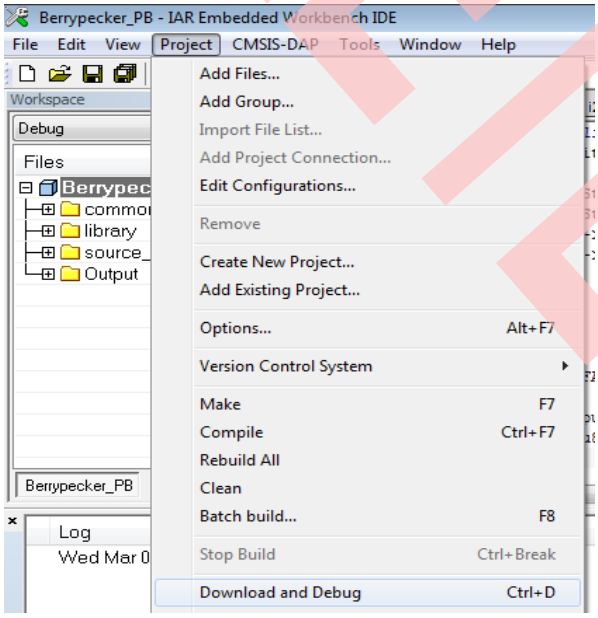
9. Connect CMSIS-DAP interface with PC to debug
10. Rebuild all.
    a. Click “**Project**”
    b. Select “**Rebuild All**”

Figure 60. Rebuild Project



11. Download and debug
    a. Click “Project”
    b. Select “Download and Debug”

Figure 61. Download and Debug

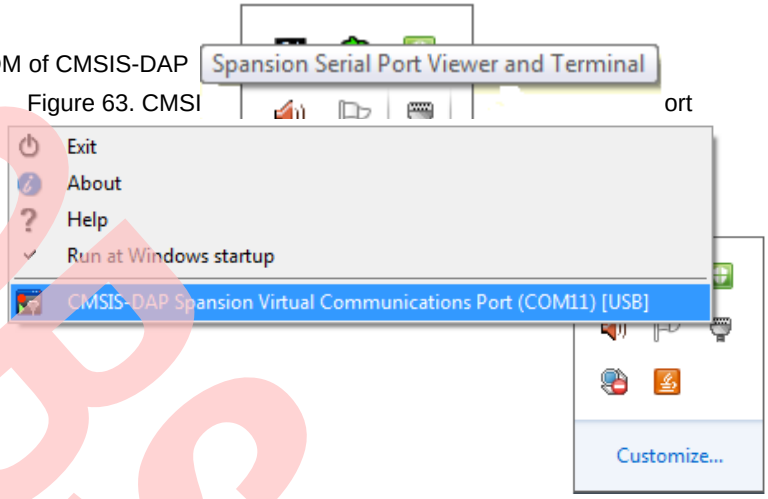


12. Open Cypress Serial Port Viewer and Terminal through icon in task menu or desktop shortcut

Figure 62. Cypress Serial Port Viewer and Terminal

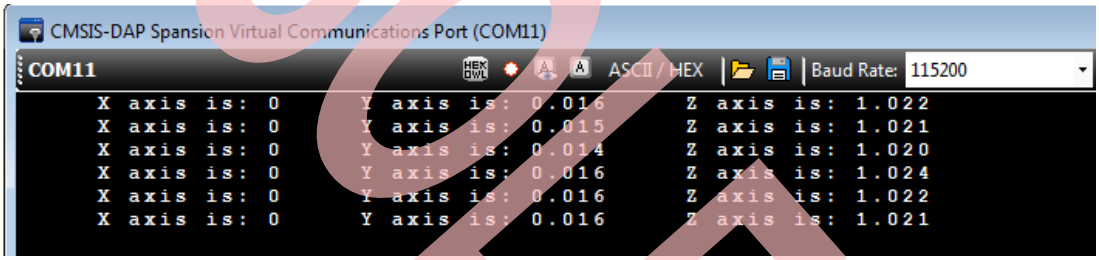
13. Select the virtual COM of CMSIS-DAP

Figure 63. CMSI ort



14. Select the baud rate “115200”, and Click the “disconnect” above the green arrow to connect

Figure 64. Virtual Communications Port Display



15. Check the output in Cypress Serial Port Viewer and Terminal; see if the value is right

6.5 Read/Write SD card

6.5.1 Lab Objective

1. Understand multi-function serial interface module.
2. Study SPI communication and program.
3. Master the I2C driver function about FM0+ PDL.
4. Understand SD card driver.
5. Understand FatFs function.

6.5.2 Lab Content

Create test.txt in SD memory card; write some characters into test.txt; and then compare.

6.5.3 Preliminary Knowledge

1. Master the basic process of coding and debugging the program in the IAR integration development environment.
2. Understand FM0+ Peripheral Library framework.
3. Understand the SPI bus protocol.
4. Understand SD card protocol (SPI mode).

6.5.4 Preparation of Lab Equipment and Tools

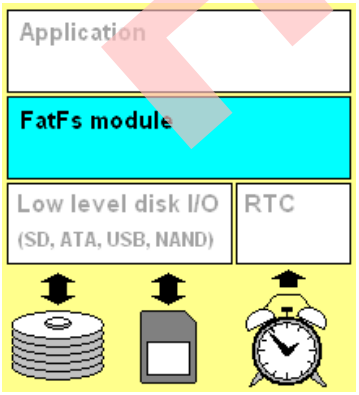
1. Hardware
   a. Mini USB cable
   b. SK-FM0-100L-S6E1B8 board
   c. PC
   d. SD Memory Card
2. Software
   a. IAR
   b. Cypress Serial Port Viewer and Terminal

6.5.5 Lab Principle and Instructions

1. FatFs is a generic FAT file system module for small embedded systems. The FatFs is written in compliance with ANSI C and completely separated from the disk I/O layer. Therefore it is independent of hardware architecture. Petit FatFs module is also available here.

Features:

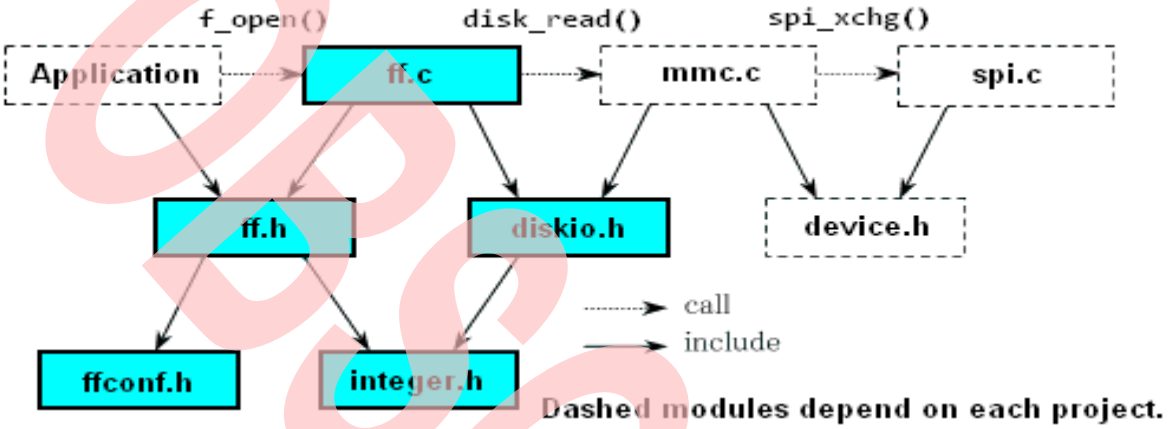
- Windows compatible FAT file system.
- Platform independent, Easy to port
- Very small footprint for code and work area
- Various configuration options:
- Multiple volumes (physical drives and partitions)
- Multiple ANSI/OEM code pages including DBCS.
- Long file name supported in ANSI/OEM or Unicode.



- RTOS support.
- Multiple sector sizes supported
- Read-only, minimized API, I/O buffer and etc

The picture shown below is a typical configuration of the embedded system with FatFS module.

Figure 65. Typical configuration of FatFS module

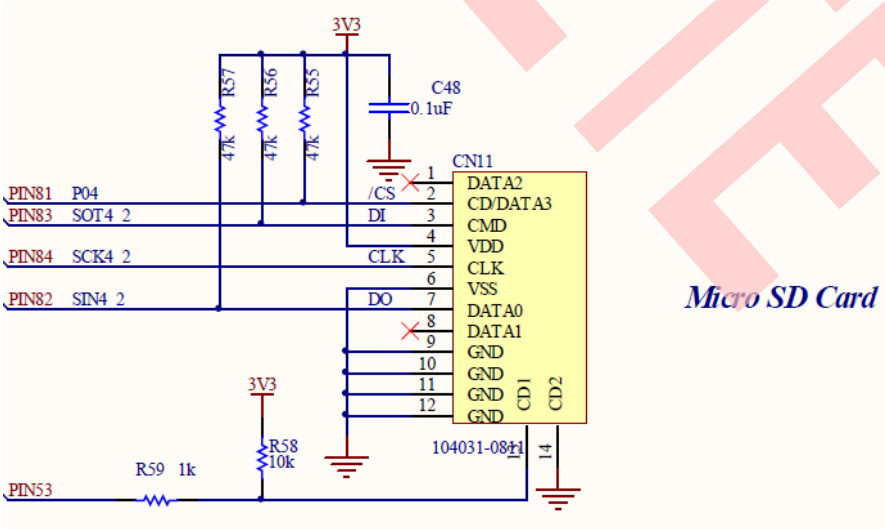


2. Disk I/O Interface:

   Since the FatFs module is completely separated from disk I/O layer, it requires following functions to access the physical media. When OS related feature is enabled, it will require process/memory functions in addition. However the low level disk I/O module is not a part of FatFs module, so that it must be provided by user.

3. As Berrypecker doesn't have the specified SDIO interface, choose the SPI mode to drive the SD card. It contains data read and write access on SD card. It will supply the user three functions, including SD card initialization, SD card read and SD card write.
4. The PDL middleware of FM0+ PDL implements the middleware SD card driver(SPI mode), so you only need to modify the requested SPI channel
5. The Figure 6 59 shows that SOT4_2, SIN4_2 and SCK2_2 are selected as SPI interface to communicate between FM0+ MCU and SD card. P04 is selected as CS signal

Figure 66. Micro SD Card Circuit



6.5.6 Lab Step

1. Insert SD memory card into CN11.
2. Open the project based on the last lab.
3. Edit file pdl_user.h to activate PDL resource and interrupt

Figure 67.Activate Requested SPI Channels

```
#define PDL_PERIPHERAL_ENABLE_MFS4          PDL_ON
```

Figure 68. Activate Requested SPI Channel Interrupts

```
#define PDL_INTERRUPT_ENABLE_MFS4           PDL_ON
```

4. Define macros about the requested SPI I/O, CS pin, and channel (sdcard_spi.h).

Figure 69. Define Requested SPI Channel I/O Macros

```
/*  Mfs pin reallocation */
#define Mfs_SdcSpiConfigIO() {bFM0P_GPIO_PFR0_P05 = 1;bFM0P_GPIO_ADE_AN20 = 0;\
                              FM0P_GPIO->EPFR08_f.SIN4S = 3;\
                              bFM0P_GPIO_PFR0_P06 = 1;bFM0P_GPIO_ADE_AN21 = 0;\
                              FM0P_GPIO->EPFR08_f.SOT4B = 3;\
                              bFM0P_GPIO_PFR0_P07 = 1;bFM0P_GPIO_ADE_AN22 = 0;\
                              FM0P_GPIO->EPFR08_f.SCK4B = 3;\
                              Gpio1pin_InitOut(GPIO1PIN_P04,Gpio1pin_initVal(1u));}
```

Figure 70. Define Requested SPI Channel Macros

```
/*  Mfs channel configuration */
#define SDC_SPI          CSIO4
```

Figure 71. Define Requested CS Pin Operation Macros

```
#define Mfs_SpiCSLow()    {Gpio1pin_Put( GPIO1PIN_P04, 0u);}
#define Mfs_SpiCSHigh()   {Gpio1pin_Put( GPIO1PIN_P04, 1u);}
```

5. Create new file main.c to replace the old one.
   a. Initializes the UART channel 0
   b. In the infinite loop
      i. If “Enter” key is pressed, the test starts.
      ii. Mount SD Card
      iii. Open file test.txt; if this file does not exist, create it
      iv. Write data (dec 0,1,2,3,4,5,6,7,8,9) into test.txt
      v. Read test.txt
      vi. Compare data

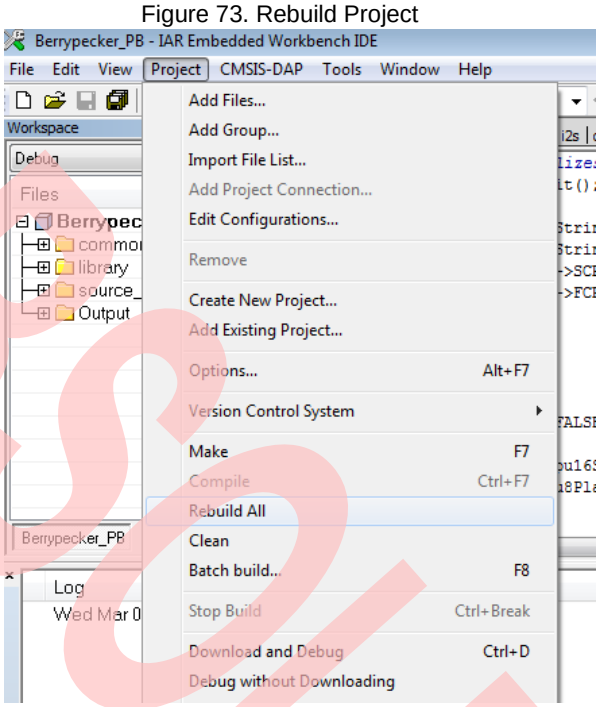
Figure 72. Main Control Code

```
int32_t main(void)
{
    uint8_t i ;
    UINT btw = 10;
    UINT bw = 0;
    BYTE u8WriteBuf[10] = {0,1,2,3,4,5,6,7,8,9};
    BYTE u8ReadBuf[10] =  {0};
    FATFS FatFs;
    FIL file;
    FRESULT fr;
    UINT u16ReadOutLen = 0;
    /* Initial uart*/
    UartInit();
    UartWriteString("*  Verify insert SD Memory Card Insert.      *\r\n");
    UartWriteString("*  Press Enter Key to start test.....       *\r\n");
    while (1)
    {
        if( FALSE == stcUartRxBuf.RxFlg)
        {
            continue;
        }

        /* Register work area to the default drive */
        fr = f_mount(&FatFs, "", 0);
        if (FR_OK != fr)
        {
            UartWriteString("*  Fail to mount Micro SD Card....       *\r\n");
            break;
        }

        /* Open a wav file */
        fr = f_open(&file, "0:test.txt", FA_CREATE_ALWAYS | FA_WRITE | FA_READ);
        if (FR_OK != fr)
        {
            UartWriteString("*  Fail to open Micro SD Card ....       *\r\n");
            f_mount(NULL, "", 0);
            break;
        }

        fr = f_write(&file, u8WriteBuf, btw, &bw);
        if (FR_OK != fr)
        {
            UartWriteString("*  Fail to write Micro SD Card ...       *\r\n");
            f_close(&file);
            f_mount(NULL, "", 0);
            break;
        }
```

```
        else
        {
            UartWriteString("*  Success to write Micro SD Card .      *\r\n");
            UartWriteString("*  Write Data:   ");
            for(i = 0; i < 10; i++)
            {
                UartWriteString(" ");
                UartSendByte(Dec2Ascii(u8WriteBuf[i]));
            }
            UartWriteString("       *\r\n");
        }
        fr = f_lseek(&file, 0);
        if (FR_OK != fr)
        {
            UartWriteString("*  Fail to lseek.....          *\r\n");
            f_close(&file);
            f_mount(NULL, "", 0);
            break;
        }
        /* Read data from wav file */
        fr = f_read (&file, &u8ReadBuf[0], 10, &u16ReadOutLen);
        if (FR_OK != fr)
        {
            f_close(&file);
            f_mount(NULL, "", 0);
            UartWriteString("*  Fail to read Micro SD Card ....      *\r\n");
            break;
        }
        /* Close the file */
        f_close(&file);
        f_mount(NULL, "", 0);
        if(Ok !=  memcmp(u8ReadBuf, u8WriteBuf, 10))
        {
            UartWriteString("*  Micro SD Card is error.....      *\r\n");
            break;
        }
        else
        {
            UartWriteString("*  Micro SD Card is OK.....         *\r\n");
            UartWriteString("*  Read Data:    ");
            for(i = 0; i < 10; i++)
            {
                UartWriteString(" ");
                UartSendByte(Dec2Ascii(u8ReadBuf[i]));
            }
            UartWriteString("       *\r\n");
            break;
        }
    }
}
```

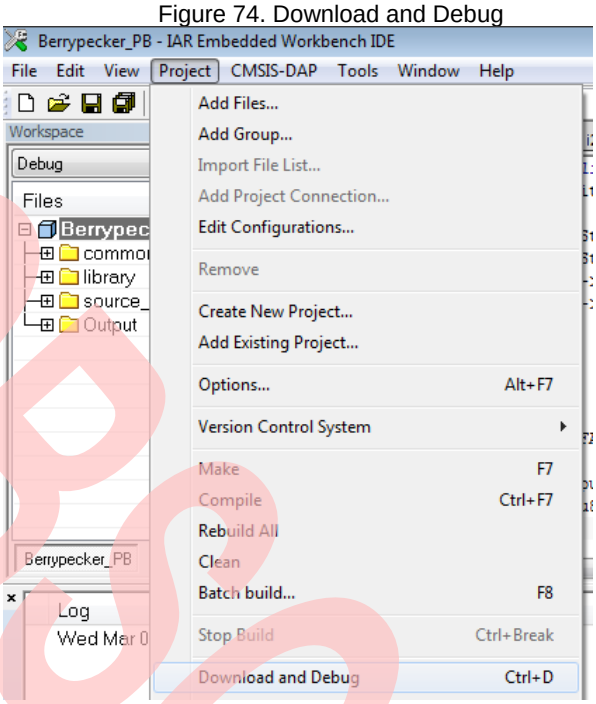
6. Connect CMSIS-DAP interface with PC to debug.
7. Rebuild all.
   a. Click “**Project**”
   b. Select “**Rebuild All**

Figure 73. Rebuild Project



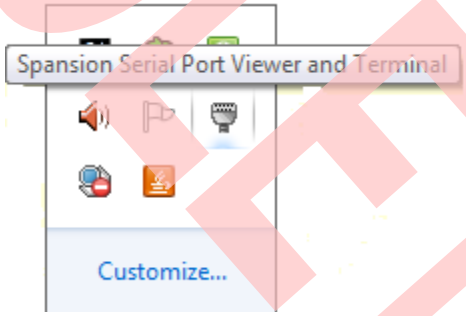
8. Download and debug
   a. Click “**Project**”
   b. Select “**Download and Debug**”

Figure 74. Download and Debug



9. Open Cypress Serial Port Viewer and Terminal through icon in task menu or desktop shortcut

Figure 75. Cypress Serial Port Viewer and Terminal

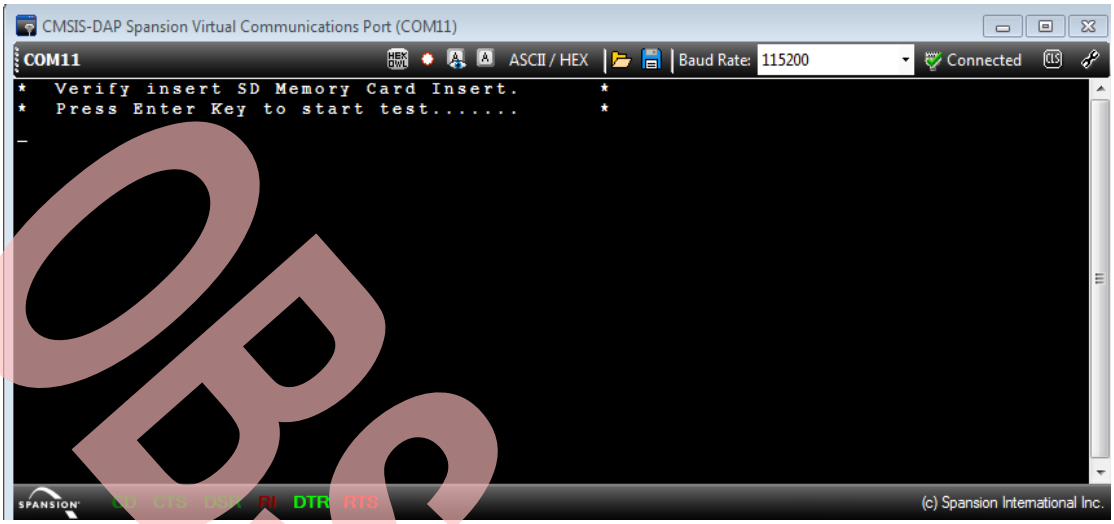


10. Select the virtual COM of CMSIS-DAP

Figure 76. Cypress Virtual Communications Port

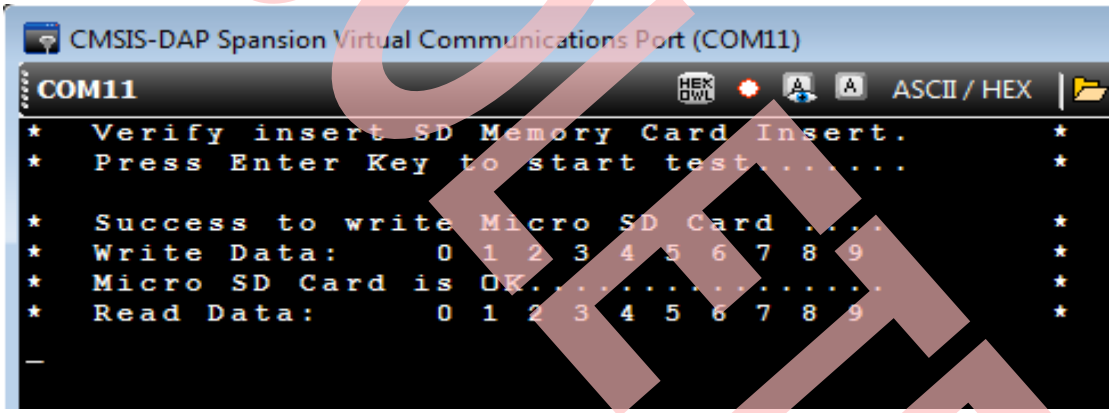
11. Select the baud rate “115200”, and Click the “disconnect” above the green arrow to connect

Figure 77. Virtual Communications Port Connection



12. Click Virtual Com and press “Enter” key.
13. Check the information displayed in Cypress Serial Port Viewer and Terminal.

Figure 78. Virtual Communications Port Display



6.6 Play Pixie Dust Sound Data

6.6.1 Lab Objective

1. Understand multi-function serial interface module.
2. Study I2C communication and program.
3. Master the I2S driver function about FM0+ PDL

6.6.2 Lab Content

Play Pixie Dust sound data which .was taken from android.googlesource.com pixiedust.wav and was converted into I2S data.

6.6.3 Preliminary Knowledge

1. Master the basic process of coding and debugging the program in the IAR integration development environment.
2. Understand FM0+ Peripheral Library framework.
3. Understand the I2S bus protocol.
4. Understand I2C protocol.

6.6.4 Preparation of Lab Equipment and Tools

1. Hardware
   a. Mini USB cable
   b. SK-FM0-100L-S6E1B8 board
   c. PC
   d. Headphone
2. Software
   a. IAR

6.6.5 Lab Principle and Instructions

1. I2S, known as Inter-IC Sound, Integrated Interchip Sound, or IIS, is an electrical serial bus interface standard used for connecting digital audio devices together. It is used to communicate PCM audio data between integrated circuits in an electronic device. The I2S bus separates clock and serial data signals, resulting in a lower jitter than is typical of communications systems that recover the clock from the data stream. Despite the name, it is unrelated to the bidirectional I²C bus.
2. Codec module is developed to transmit audio data and control the transmission flow. To achieve this purpose, I2S and I2C bus protocol must be supported. Audio data transmission will be powered by I2S (inter-IC sound) bus protocol. This bus has only to handle audio data, while the other signals are transferred separately. I2S is a 3-line serial bus and consists of a line for two-multiplexed data channels, a word select line and a clock line. The picture below is the simple system configurations of I2S bus. Both transmitter and receiver could be master mode. In other words, both transmitter and receiver can supply clock and word select signal.

Figure 79. I2S Bus – Transmitter is master

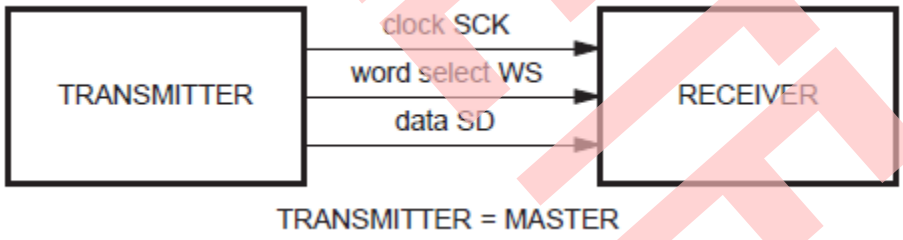


Figure 80. Receiver is master

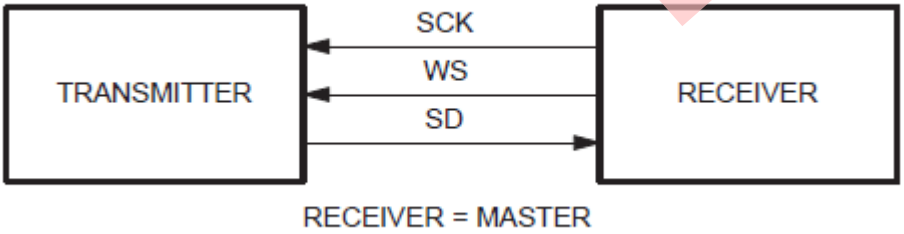
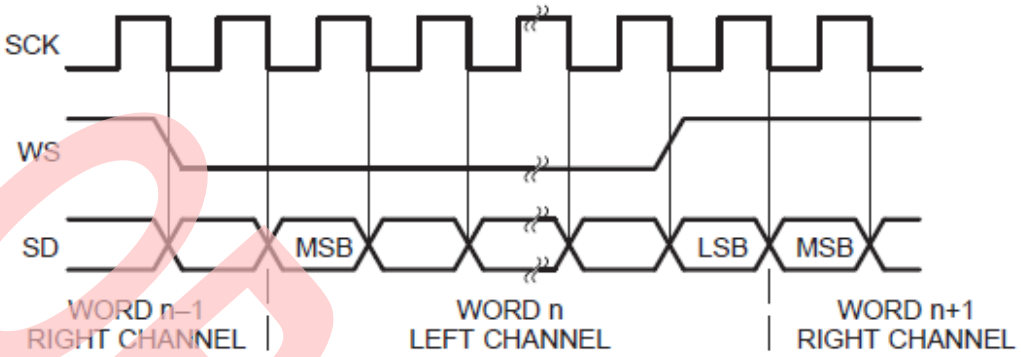
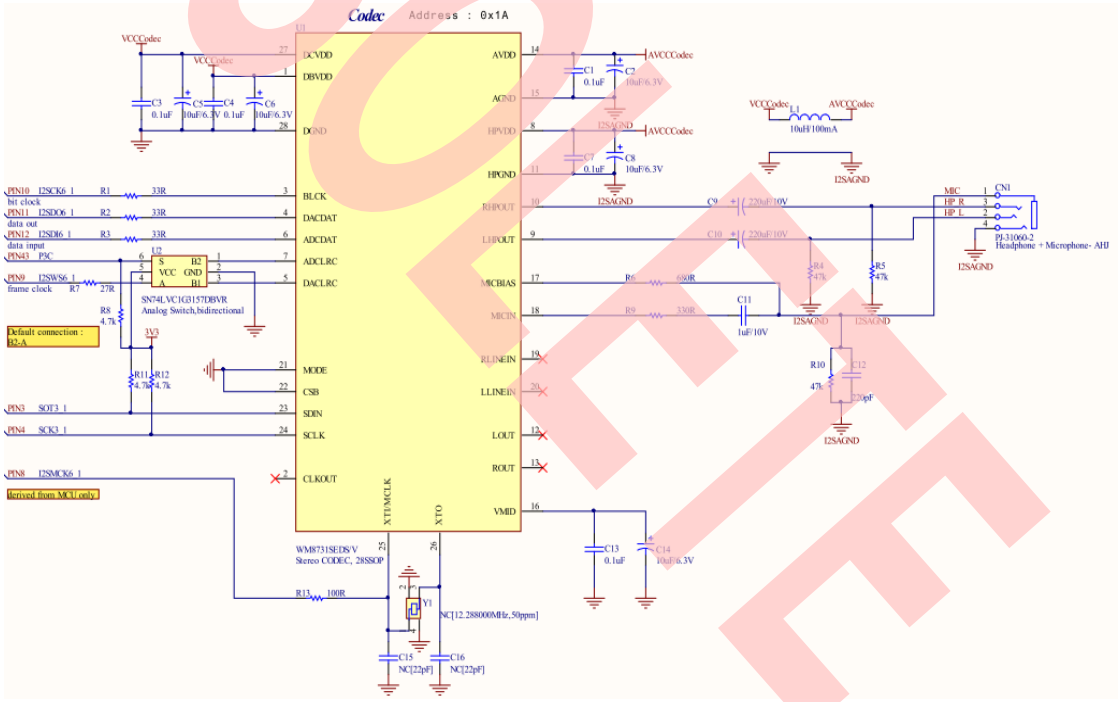


Figure 81. I2S basic interface timing



3. The PDL middleware of FM0+ PDL implements middleware codec_wm8731 driver, so you only need to modify the requested SPI channel and I2C channel.
4. The Figure 6 75 shows that I2SMCK6_1, I2SWS6_1, I2SDO6_1 and I2SCK6_1 are selected as I2S interface I/O; SOT3_1 and SCK3_1 are selected as I2C interface I/O to communicate between FM0+ MCU and wm8731.

Figure 82. Codec Circuit



6.6.6 Lab Step

1. Open the project based on the last lab.
2. Edit file pdl_user.h to activate PDL resource and interrupt

Figure 83. Activate Requested I2C and I2S Channels

```
#define PDL_PERIPHERAL_ENABLE_MFS3          PDL_ON
#define PDL_PERIPHERAL_ENABLE_I2S1          PDL_ON
```

Figure 84. Activate Requested I2C and I2S Channel Interrupts

```
#define PDL_INTERRUPT_ENABLE_MFS3           PDL_ON
#define PDL_INTERRUPT_ENABLE_I2S1           PDL_ON
```

3. Define macros about the requested I2C I/O, I2S I/O, and channel (codec_wm8731.h).

Figure 85. Define Requested I2C and I2S Channel I/O Macros

```
/*  Mfs pin reallocation */
#define    Mfs_CodecI2cConfigIO()   {SetPinFunc_SOT3_1();SetPinFunc_SCK3_1();}

/*  I2S pin reallocation */
#define    InitI2sIIo()   {SetPinFunc_I2SMCK6_1_IN();  SetPinFunc_I2SWS6_1();\
                SetPinFunc_I2SDO6_1();  SetPinFunc_I2SCK6_1();}
```

Figure 86. Define Requested I2C and I2S Channel Macros

```
#define WM8731_I2C_CH         (I2C3)
```

Figure 87. Define Requested I2C Speed and Slave Address Macros

```
/* I2C runnning baudrate */
#define WM8731_BAUDRATE       (400000u)
#define WM8731_ID             (0x1Au)      // Codec slave address
```

4. Create new file main.c to replace the old one.
   a. Initialize system tick
   b. Initialize the UART channel 0
   c. Initialize Codec WM8731
   d. In the infinite loop
      i. Check whether Pixie Dust sound data is played over.
      ii. If “Enter” key is pressed, the play stops

Figure 88. Main Control Code

```
int32_t main(void)
{
    /* Initial system tick */
    SysTick_Init(100);
    /* Initial UART */
    UartInit();
    /* Initializes Codec */
    WM8731_Init();

    UartWriteString("*  Codec WM8731..... Started          *\r\n");
    UartWriteString("* Fristly, connect CN1 to the headphone          *\r\n ");
    FM0P_I2S1->SCR_f.TXE = 1;
    FM0P_I2S1->FCR1_f.FTIE = 1;// Enable FIFO int

    while (1)
    {
        if (FALSE == u8PlayFlag)
        {
            pu16SoundData = &au16PixieDustSoundI2s[0];
            u8PlayFlag = TRUE;
        }

        if (TRUE == stcUartRxBuf.RxFlg)
        {
            FM0P_I2S1->SCR_f.TXE = 0;
            FM0P_I2S1->FCR1_f.FTIE = 0;
            u8PlayFlag = FALSE;
            stcUartRxBuf.RxFlg = FALSE;
            break;
        }
    }
}
```

5. Rebuild and run the program.
6. Insert Headphone to CN1
7. Listen and check whether playing is normal.

7 Additional Information

For more Information on Spansion semiconductor products, visit the following websites:

English version address:

http://www.spansion.com/Products/microcontrollers/

Chinese version address:

http://www.spansion.com/CN/Products/microcontrollers/

Please contact your local support team for any technical question

America: Spansion.Solutions@Spansion.com

China: mcu-ticket-cn@spansion.com

Europe: mcu-ticket-de@spansion.com

Japan: mcu-ticket-jp@spansion.com

Other: http://www.spansion.com/Support/SES/Pages/Ask-Spansion.aspx

8 Document History

Document Title: AN204393 - FM0+ Starter Kit 32-Bit Microcontroller

Document Number: 002-04393

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	–	JHHO	03/13/2015	Initial release
*A	5029377	JHHO	12/01/2015	Migrated Spansion Application Note S6E1B8_AN710-00007-1v0-E to Cypress format.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless
Spansion Products	spansion.com/products

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com