Kinetis SDK Demo Applications User's Guide

Freescale Semiconductor, Inc.

KSDKDEMOUG
1.0.0-beta
Mar 2014

Contents

Contents

Contents

Contents

Contents

Chapter 1
ADC Period Sample Demo

A demo application demonstrates ADC period sample triggered by PIT.

Modules

- ADC Period Sample Introduction
  Introduction of the ADC Period Sample demo application.
- ADC Period Sample User Guide
  User guide on how to customize this application for different configurations.

1.1 Detailed Description

1.2 ADC Period Sample Introduction

Introduction of the ADC Period Sample demo application.

1.2.0.1 ADC Period Sample Introduction

Overview

This is an ADC period sample demo application which demonstrates the ADC hardware trigger function by PIT timer. It samples the input digital signal from ADCx_DPx pin and records each sample point with the appropriate amplitude. After one period sample is complete, it prints out the rough shape of the signal wave on the debug console like a primitive oscilloscope.

Supported Hardware

- TWR-K64F120M
- FRDM-K64F120M

Default Debug Console

- UART instance: OpenSDA CDC
- Baudrate: 115200bps
- 8 data bits
- No parity
- 1 stop bit

1.3 ADC Period Sample User Guide

User guide on how to customize this application for different configurations.

1.3.0.2 ADC Period Sample User Guide

Getting Started

1. To run this demo, a sine wave signal input is required. Generally, you can use *Function Generator* to get such signal. The input signal should meet the following requirement:
   - sine wave signal
   - High amplitude: 0 V to 3.3 V
   - Low amplitude: 0 V to 3.3 V
   - Frequency: fixed to 50 Hz; If you want to change the input signal frequency configurations in the demo code, see the next chapter.
2. Connect the input signal to the following pin:
   - TWR-K64F120M: pin 17 in J4 as ADC0_DP0
   - FRDM-K64F120M: pin 1 in J26 as ADC0_DP1
3. Open the UART console on PC.
4. Download and run the program on the target.
5. The input signal wave form, the high/low amplitude is displayed on the console.
6. If using the GCC built out binary, please use JLINK connecting JTAG port to do debug. Also the debug UART must be connected out from Pin1 J10 for Rx, and Pin2 J15 for Tx (e.g TWR-K60-F120M)

Default configurations

ADC Configurations

- Clock source: bus clock
- VRef: VREFH/L 3.3 V
- Resolution: 16 bit single-ended
- Channel: ADC0_DP0/DP1
- Hardware Trigger: Period Interrupt Timer0

Sample frequency

The default sample rate is 50 Hz $*$ 100, which enables the demo application to get 100 samples per period. If you want to change the sample rate, see the next chapter.

HOWTO Customization

This demo application is customizable to show different kinds of input signal wave.

Configure the number of samples

Printing of the signal wave shape depends on the console size. A console can be 100x40. The best effect of printing is to align the number of samples to the console column numbers and convert the amplitude range to the [0, row - 1] range. Configuring the number of samples means configuring the console column size:

```
#define MAX_CONSOLE_COL 100 // the console max column size
#define NR_SAMPLES MAX_CONSOLE_COL // number of samples in one period
```

Configure the signal frequency

Change the following macro to configure the desired frequency in HZ units.

```
#define FREQ_INPUT 50  // 50HZ
```

Configure the ADC instance

Change the ADC_INST macro to configure the ADC instance you want to use.

```
#define ADC_INST 0  // ADC0 instance
```

Change the channel configurations in init_adc() third parameter.

```
adc_hal_enable(instance, 0, kAdcChannel0, false);
```

Configure the debug console baudrate.

```
#define DEBUG_UART_BAUD 115200 // debug uart baudrate
```

Chapter 2
DSPI eDMA Demo

This demo application is a DSPI & eDMA demonstration program.

Modules

- DSPI eDMA Demo Introduction
  Introduction and the function/definition comments of this demo.
- DSPI eDMA Demo User Guide
  User guide on how to use this demo.

2.1 Detailed Description

2.2 DSPI eDMA Demo Introduction

Introduction and the function/definition comments of this demo.

2.2.0.3 DSPI eDMA Demo Introduction

Overview

This application demonstrates how to configure DSPI transfers using eDMA, in both send and receive.

Supported platforms

Currently, the DSPI eDMA project is supported on these platforms:

- TWR-K64F120M
- FRDM-K64F120M

2.3 DSPI eDMA Demo User Guide

User guide on how to use this demo.

2.3.0.4 DSPI eDMA Demo User Guide

Getting started

1. Hardware settings

DSPI eDMA Demo program requires the following hardware:

- TWR-K64F120M, FRDMK64F120M

2. Download program

- Download the program to the target board.
- Connect a USB cable to the PC host and open a serial terminal at 115200 baud, 8-bits no parity, 1 stop bit.
- When successfully connected, the program sends "Press spacebar to begin." to the terminal. Press the space key to continue. Communication Interface Settings:
  - TWR-K64F120M(OpenSDA) UART1, TX: PTC3 RX: PTC4.
  - FRDM-K64F120M(OpenSDA) UART0, TX: PTB16 RX: PTB17.

SPI Connections:

TWR-K64F120M

Master | Slave

Signal | MCU | Board | Signal | MCU | Board

SS | PTD0 | (J11A-B46)| SS | PTB10 | (J11A-B9) SCK | PTD1 | (J11A-B48)| SCK | PTB11 | (J11A-B7) MstOut | PTD2 | (J11A-B45)| SlvIn | PTB17 | (J11A-B11) MstIn | PTD3 | (J11A-B44)| SlvOut | PTB16 | (J11A-B12)

FRDM-K64F

Master | Slave

Signal | MCU | Board | Signal | MCU | Board

SS | PTD0 | (J2-6) | SS | PTD4 | (J6-4) SCK | PTD1 | (J2-12) | SCK | PTD5 | (J6-5) MstOut | PTD2 | (J2-8) | SlvIn | PTD7 | (J6-7) MstIn | PTD3 | (J2-10) | SlvOut | PTD6 | (J6-6)

3. Run the program

The following will appear in the terminal window:

```
DSPI eDMA Demo

DSPI Transfers using eDMA

Configuration:
```

Setting	Value
Terminal Baud:	115200 bps
TDSPI Bit rate:	5000 bps
TDSPI Master:	DSPI0
TDSPI Slave:	DSPI1

```
Connection on TWR-K64F120M:
```

Master	Slave
SS PTD0 (J11A-B46)	PTB10 (J11A-B9)
SCK PTD1 (J11A-B48)	PTB11 (J11A-B7)
MstOut PTD2 (J11A-B45)	PTB17 (J11A-B11) SlvIn
MstIn PTD3 (J11A-B44)	PTB16 (J11A-B12) SlvOut

```
Connection on FRDM-K64F120M:
```

Master	Slave
SS PTD0 (J2-6)	PTD4 (J6-4)
SCK PTD1 (J2-12)	PTD5 (J6-5)
MstOut PTD2 (J2-8)	PTD7 (J6-7) SlvIn
MstIn PTD3 (J2-10)	PTD6 (J6-6) SlvOut

This indicates the speed of the SPI transfers, and what pins to connect and how to connect them.

Key Functions

1.

void dspi_edma_master_setup(uint8_t instance, uint32_t baudRateHz, uint8_t transferSizeBits)

Takes in the DSPI module instance, the desired data rate of DSPI transfers, and the frame size of the data transfer. Initializes master instance of DSPI.

Parameters

instance	DSPI module instance
baudRateHz	Pass in the desired baud rate of DSPI transfers in Hz.
transferSizeBits	Pass data frame size (8 or 16 bit)

2.

void dspi_edma_slave_setup(uint8_t instance, uint8_t transferSizeBits)

Takes in the DSPI module instance, and the frame size of the data transfer. Initializes slave instance of DSPI.

Parameters

instance	DSPI module instance
transferSizeBits	Pass data frame size (8 or 16 bit)

3.

dspi_status_t data_source(uint8_t ∗ sourceWord, uint32_t instance)

Callback function for DSPI slave. Used to transmit data from slave. In this application it generates slave data out, the data out is a count.

Parameters

sourceWord	8-bit data variable to be passed to slave PUSHR register.
instance	Instance of the DSPI module.

4.

void on_error(dspi_status_t error, uint32_t instance)

Callback function for DSPI slave. Used to handle errors. In this application is sets the error flag to signal the end of the demonstration.

Parameters

error	Uses dspi_status_t value given by driver interrupt handler.
instance	Instance of the DSPI module.

5.

void setup_edma_loop(edma_loop_setup_t ∗loopSetup, uart_state_t ∗uart)

Configures an eDMA transfer loop using a loopSetup structure, and if a valid uart_state_t is passed will print out the TCD for that loop.

Parameters

loopSetup	Data structure defined by user, containing all the parameters for the eDMA loop.
uart	Pointer to a valid uart_state_t for debug printing.

6.

void disable_edma_loop(edma_loop_setup_t ∗loopSetup, uart_state_t ∗uart)

Disables user configured eDMA transfer loop. Also, will free memory allocated by eDMA transfer loop. If a valid uart_state_t is passed will print out the TCD for that loop.

Parameters

loopSetup	Data structure defined by user, containing all the parameters for the eDMA loop.
uart	Pointer to a valid uart_state_t for debug printing.

7.

void ∗mem_align(size_t ptrSize, uint32_t alignment)

Function to perform aligned data memory allocation. Useful when mem_align is not available.

Parameters

ptrSize	size_t variable to pass size of memory to be allocated
alignment	uint32_t variable to pass byte size to align data with

Returns

pointer to aligned & allocated memory

8.

void free_align(void ∗ptr)

Function to free memory allocated by mem_align

Parameters

ptr	pointer that has been allocated with mem_align

Chapter 3
Flash Demo

Flash Demo.

Modules

- Flash Demo Introduction
  Demo application which uses flash Standard Software Drivers (SSD) to erase and program flash regions, and perform swap.
- Flash Demo User Guide
  The flash_demo application can run from either internal RAM or Flash and uses the Standard Software Drivers (SSD) to erase and program different regions of flash, and run swap if supported on the device.

3.1 Detailed Description

3.2 Flash Demo Introduction

Demo application which uses flash Standard Software Drivers (SSD) to erase and program flash regions, and perform swap.

3.2.0.5 FLASH DEMO Introduction

Overview

The Flash demo project will show examples of how to erase, program, and performs swap (if available) on the flash module. The functionality is provided with the KSDK peripheral drivers and Standard Software Drivers (SSD) for Flash. Targets exist for both Flash and SRAM memory space. The features include:

1. Full Flash erase and programming support
2. Flash Erase by block or sector, including margin read options
3. Programming region defined by user
4. Flash verify and checksum support
5. Flash Swap (if supported on device)

Supported platforms

Currently, the Flash Demo project is supported on these platforms:

- TWR-K64F120M
- FRDM-K64F120M

3.3 Flash Demo User Guide

The flash_demo application can run from either internal RAM or Flash and uses the Standard Software Drivers (SSD) to erase and program different regions of flash, and run swap if supported on the device.

3.3.0.6 Flash Demo User Guide

Getting started

The Flash Demo example code walks through different examples of how to erase and program flash contents and use the swap feature if it is supported on the device.

Default Debugger Connection

```
1. TWR-K64F120M (on board OpenSDA)
2. FRDM-K64F120M (on board OpenSDA)
```

Commands/Directions

1. Select Flash_Debug target. Both Flash_Debug and Ram_Debug targets are available.
   Flash_Debug supports Swap feature for devices that support Swap (K64F).
2. Connect one end of the USB cable to a PC host and the other end to the OpenSDA connector on the board.
3. Open Terminal program such as TeraTerm, Putty, or Hyperterminal.
4. Configure the Terminal program to select the OpenSDA COMx port for the board, using 115200 8N1: 115200 baud, 8 data bits, No parity, 1 Stop bit.
5. Connect to the board with the debugger (download & debug), run the program, and view the Terminal messages for flash operations being performed.
6. For devices that support Swap, the Flash_Debug target will copy (program) the application that is running from the lower block to the upper block, and then swap commands will be issued.
7. Flash memory blocks will be swapped at the next reset. Disconnect debug session and hit the reset button on the board.
8. Also note, during swap, memory locations 0x7F000 & 0xFF000 are swapped and displayed on the terminal showing how the memory map changes.
9. For devices that do not support swap, simply view the terminal messages for flash operations that are occurring for the demo.
10. Optionally, run the Ram_Debug target, which supports additional commands like FlashEraseAllBlock() which are not possible when executing code from flash memory space.
11. Terminal will display "Flash Demo Complete!" when finished.
    Note: GCC Builds only support "Flash Debug" target

Chapter 4
FlexCAN Demo

FlexCAN Demo.

Modules

- FlexCAN Demo Introduction
  Introduction and the function/definition comments of this demo.
- FlexCAN Demo User Guide
  User guide on how to use this demo.

4.1 Detailed Description

4.2 FlexCAN Demo Introduction

Introduction and the function/definition comments of this demo.

4.2.0.7 FlexCAN Demo Introduction

Overview

UART to FlexCAN project is a simple demonstration program based on KSDK peripheral drivers. It provides a way to demonstrate the performance of FlexCAN and UART peripheral drivers and use asynchronous send and receive functions to create a bridge between UART and FlexCAN. The features include:

1. UART send and receive support
2. Timeout supported by PIT0
3. FlexCAN transfer and receive configure support
4. FlexCAN send and receive support

Supported platforms

So far this project has been supported on platforms listed as below:

```
TWR-K64F120M
```

Function Description

These are the five functions which are defined in main.c

The prototypes for these functions are:

```
void init_hardware(void);

int printf(const char *format,...);

int scanf(const char *format,...);

void PIT0_IRQHandler(void);

void main(void);
```

See the main.c file for more details.

4.3 FlexCAN Demo User Guide

User guide on how to use this demo.

4.3.0.8 FlexCAN Demo User Guide

Getting started

There are two boards which should be used in this demo to perform peer to peer communication.

Default Debugger Connection

- TWR-K64F120M(on board OpenSDA)

Terminal Configuration

- TWR-K64F120M: UART1 PTC3/PTC4

Pin configurations:

- TWR-K64F120M: CAN0_TX: J11A-B42 (PTB18) , CAN0_RX: J11A-B41 (PTB19)

Commands/Directions

1. Download the program to target board, which should be installed in a tower system with TWR-SER.
2. Connect the USB cable to a PC host and open a serial terminal at 115200 baud, 8-bits no parity, 1 stop bit.
3. Short J5(1-2) J5(3-4) J5(5-6) J5(7-8) to enable CAN connection.
4. Connect the two TWR-SERs through the CAN port(J7).
5. When successfully connected, reset the TWR-MCU boards to run the demo and select the node ID for each board by typing A/a or B/b followed Enter.
6. You may now transfer characters by using serial terminals. For more details, see the video in for more details.

Chapter 5
Flextimer PWM Demo

This demo Flextimer PWM application.

Modules

- Flextimer PWM Introduction
  Introduction and the function/configuration of Flextimer PWM.
- Flextimer PWM User Guide
  User Guide on how to use the Flextimer PWM applications.

5.1 Detailed Description

5.2 Flextimer PWM Introduction

Introduction and the function/configuration of Flextimer PWM.

5.2.0.9 FTM PWM Introduction

Overview

This application demonstrates the FTM edge-aligned PWM function.

This application uses one TWR-K64F120M board.

For TWR-K64F120M board: Output PWM from FTM3 Channel 1, which is connected to PTE6 (LED D5 on board). The LED D5 on TWR-K64F120M board increases/decreases lightness according to PWM pulse width changes.

For FRDM-K64F120M board: Output PWM from FTM0 Channel 0 on PTC1, PTC1 (J1 Pin5) is connected by external wire with PTE26 (J2 Pin1 LED Green). The LED Green on FRDM-K64F120M board increases/decreases lightness according to PWM pulse width changes.

Supported platforms

Currently, the ftm_pwm project is supported on the following platforms:

- TWR-K64F120M
- FRDM-K64F120M

5.3 Flextimer PWM User Guide

User Guide on how to use the Flextimer PWM applications.

5.3.0.10 FTM PWM User Guide

Getting started

1. Power on.
2. Download and run ftm_pwm code to board.
3. The LED on board increases/decreases lightness according to PWM pulse width changes.

Chapter 6
GPIO I2C Demo

This demo application uses GPIO to simulate an I2C to access mma8451.

Modules

- GPIO I2C Introduction
  Introduction and the function/configuration of GPIO I2C functions.
- GPIO I2C User Guide
  User guide on how to use GPIO I2C.

6.1 Detailed Description

6.2 GPIO I2C Introduction

Introduction and the function/configuration of GPIO I2C functions.

6.2.0.11 GPIO I2C Introduction

Overview

This demo application is a simulator. It uses GPIO to simulate I2C protocol and thus to control mma8451 chip on board. The UART console shows values of X, Y, and Z direction, obtained from mma8451.

Supported Hardware

TWR-K64F120M (OpenSDA as debug UART) FRDM-K64F120M (OpenSDA as debug UART)

Default Debug Console

- UART instance: OpenSDA CDC
- Baudrate: 115200bps

UART Output

MMA8451:

For MMA8451, UART gives output values of X, Y, and Z direction, obtained from mma8451. For FXOS8700CQ, UART gives output values of X, Y, Z direction and magnetism.

```
== Accelerometer Test ==

Reading acceleration data...
x=4, y=1, z=1
```

FXOS8700CQ:

For FXOS8700CQ, UART gives output values of X, Y, Z direction and magnetism.

```
== Accelerometer Test ==

Jan  8 2014    15:12:40
Initializing FXOS...
Resetting FXOS...
Reading acceleration data...
xAcc = -120, yAcc = -120, zAcc = 4095, xMag = -1, yMag = -1, zMag = -1
```

6.3 GPIO I2C User Guide

User guide on how to use GPIO I2C.

6.3.0.12 GPIO I2C User Guide

File Description

- i2c_sim.h: API header of i2c functions.
- i2c_sim.c: API implementation of i2c functions.
- fsl_mma8451.h: API header of MMA8451 functions.
- fsl_mma8451.h: API implementation of MMA8451 functions.
- mma8451_main.c: Main function that shows the values that read from MMA8451 driver.
- fxos8700cq.c: Main function that will use fxos8700cq library APIs to show sensor data.

Build

In K64 Freedom board demo, fxos8700cq is a library in boards/common which you need to build first.

HOWTO Customization

There are parameters that can be configured to meet another request.

Pin Selection

Pin selection on board:

Function	TWR-Kinetis	FRDM-Kinetis
SCL	PTC10	PTC11
SDA	PTE24	PTE25

GPIO_SCL_PORT and GPIO_SDA_PORT are used as SCL and SDA port index in I2C driver.

GPIO_SCL_PIN and GPIO_SDA_PIN are used as SCL and SDA PIN index in I2C driver.

These pins are used to connect GPIO as I2C to MMA8451 or FXOS8700CQ I2C interface PINs.

These four macros can be configured in i2c_sim.h.

```
#define GPIO_SCL_PORT   BOARD_I2C_GPIO_SCL_PORT
#define GPIO_SDA_PORT   BOARD_I2C_GPIO_SDA_PORT
#define GPIO_SCL_PIN    BOARD_I2C_GPIO_SCL_PIN
#define GPIO_SDA_PIN    BOARD_I2C_GPIO_SDA_PIN
```

Baudrate Configuration

To get the correct baudrate, use an oscilloscope to get correct waveform for a baudrate.

A delay is used in this demo application for a correct baudrate.

```
#define I2C_DELAY \
do \
{ \
    int32_t i; \
for (i = 0; i < 500; i++) \
    { \
    __asm("nop"); \
    } \
} while (0)
```

For a higher baudrate, reduce the delay time and loop count.

For a lower baudrate, increase the delay time and loop count.

Additionally, you must also consider code execution time.

Thus, it is recommended that you use an oscilloscope to get the correct waveform for a baudrate.

There is no delay in code execution in this demo application, therefore, this might be the highest baudrate.

Please read I2C protocal and MMA8451 chip details for more information.

Chapter 7
GPIO UART Demo

This part describes the GPIO UART demo application. The GPIO_UART application simply implements a bit-banged UART terminal application.

Modules

- GPIO UART Introduction
  This part provides brief getting started introductory information.
- GPIO UART User Guide
  This part provides more detailed information on using this demo application.

7.1 Detailed Description

7.2 GPIO UART Introduction

This chapter provides brief getting started introductory information.

7.2.0.13 GPIO UART Introduction

Overview

This application is a simple bit-banged UART implementation featuring the KSDK. It prints a banner to the connected serial terminal and echoes characters that are typed into the connected terminal.

Supported platforms

This application supports the following platforms.

- TWR-K64F120M
- FRDM-K64F120M

7.3 GPIO UART User Guide

This chapter provides more detailed information on using this demo application.

7.3.0.14 GPIO UART User Guide

Getting started

The GPIO UART project is designed to work with the tower system or in a stand alone setting. If you are using your Tower or Freedom board in a stand alone setup, follow the OpenSDA directions. Otherwise, if you are using the Tower system and a serial card, follow the Serial Card instructions.

OpenSDA

Download the program to target board. Connect a USB cable to PC host and open a serial terminal configured as described in "Terminal Configuration". When successfully connected, press the reset button on your development board. The following banner should be displayed on the terminal:

- Beginning GPIO UART Emulation Demo ∗
- Input any character to have it echoed to ∗
- the terminal ∗

Now you may type characters into the terminal and what you type will be echoed back to the terminal.

Serial Card

Connect a Serial card to your PC (you may need a serial to USB converter for this) and open a serial terminal configured as described in "Terminal Configuration". When successfully connected, download the program to target board. The following banner should be displayed on the terminal:

- Beginning GPIO UART Emulation Demo ∗
- Input any character to have it echoed to ∗
- the terminal ∗

Now you may type characters into the terminal and what you type will be echoed back to the terminal.

Terminal Configuration

The GPIO UART project is, by default, configured to the following terminal configuration:

- 19200 baud rate
- 8 data bits
- No parity
- One stop bit
- No flow control

Communication Interface settings

If you need to change the default port pins, perform this step:

- Change the kGpioUartDemoTX and kGpioUartDemoRX to the desired pin in the _gpio_pins enumeration table in the gpio_pins.h file.

Pin configurations:

No extra pins are needed for this project. Only the pins described in Communication Interface Settings are required for this demo application.

Commands/Directions

Once the application begins, it will simply print a banner to the terminal, followed by two new line characters and a carriage return. Then it will wait for user input into the terminal. When characters are typed into the connected terminal, the application will receive the character, and then echo the received character back to the connected terminal.

Chapter 8
Hello World Demo

This demo application simply prints "Hello World" to the terminal and then echoes characters input to the terminal.

Modules

- Hello World Introduction
  Introduction and the function/configuration of the hello_world demo application.
- Hello World User Guide
  User guide on how to use the hello_world application.

8.1 Detailed Description

8.2 Hello World Introduction

Introduction and the function/configuration of the hello_world demo application.

8.2.0.15 Hello World Introduction

Overview

The Hello World project is a simple demonstration program that utilizes the KSDK software. It prints "Hello World" to the terminal using the UART drivers of the KSDK. The intent of this program is to provide an example of how to use the UART and a simple debugging (golden) project.

Supported platforms

Currently, the Hello World project is supported on these platforms:

- TWR-K64F120M
- FRDM-K64F120M

8.3 Hello World User Guide

User guide on how to use the hello_world application.

8.3.0.16 Hello World User Guide

Getting started

The Hello World project is designed to work with the Tower System or in a stand alone setting. If you are using the Tower System or a Freedom board in a stand alone setup, follow the OpenSDA directions. Otherwise, if you are using the Tower System and a serial card, follow the Serial Card instructions.

OpenSDA

Download the program to a target board. Connect the USB cable to the PC host and open a serial terminal configured as described in the "Terminal Configuration". When successfully connected, press the reset button on your development board. "Hello World!" should be displayed on the terminal. You may type characters into the terminal. The typed characters are echoed back to the terminal.

Serial Card

Connect a Serial card to a PC (you may need a serial to USB converter for this) and open a serial terminal configured as described in the "Terminal Configuration". When successfully connected, download the program to target board. "Hello World!" should be displayed on the terminal. You may type characters into the terminal and the typed characters are echoed back to the terminal.

Terminal Configuration

The Hello World project is, by default, configured to this terminal configuration:

- 115200 baud rate
- 8 data bits
- No parity
- One stop bit
- No flow control

Communication Interface Settings:

The Hello World is configured by default to use these port pins for the platforms:

- TWR-K64F120M(OpenSDA) UART1 TX: PTC4 RX: PTC3.

- FRDM-K64F120M(OpenSDA) UART0 TX: PTB17 RX: PTB16

If you need to change the default port pins, modify the configure_uart1_pin_mux function to enable the desired UART port pins.

Chapter 9
I2C Communication Demo

This demo I2C communication application.

Modules

- I2C Communication Introduction
  Introduction and the function/configuration of the I2C master/slave communication.
- I2C Communication User Guide
  User Guide on how to use the I2C communication applications.

9.1 Detailed Description

9.2 I2C Communication Introduction

Introduction and the function/configuration of the I2C master/slave communication.

9.2.0.17 I2C Communication Introduction

Overview

The I2C communication application demonstrates I2C data communication with sub address function and low power wakeup by I2C slave address matching.

At first I2C slave enters low power wait mode, and one LED on I2C slave board is on to indicate the MCU is in sleep and no code is running.

Then I2C slave is waken up by I2C address matching interrupt when I2C master sends address, and the LED on I2C slave board is toggled during data communication.

After power on, I2C master starts to read data from I2C slave data buffer, and print out on the screen showing below:

I2C Master reads buffer values from I2C Slave sub address:

Slave Sub Address	Character
[0]	I
[1]	2
[2]	C
[3]	-
[4]	C
[5]	O
[6]	M
[7]	M

User can change I2C slave sub address content by inputing new character in I2C master command line:

"

Please input Slave sub address and the new character.

Slave Sub Address: 5

Input New Character: 5

"

Then the slave sub address and the new character are transmitted to I2C slave by I2C sub address field and data field.

Received the data, I2C slave changes its sub address content according to the sub address and data received.

Currently, the i2c_comm project is supported on the following platforms:

- TWR-K64F120M
- FRDM-K64F120M

The I2C master and slave's serial terminal is configured for the following settings:

- 115200
- 8 data bits
- No parity
- 1 stop bit

9.3 I2C Communication User Guide

User Guide on how to use the I2C communication applications.

9.3.0.18 I2C Communication User Guide

Getting started

The I2C communication application is designed to work with two boards.

1. Connect I2C clock, Data, GND between the two boards.
2. Power on the I2C slave board first, download and run the I2C slave code, wait until the LED turns on
3. Power on the I2C master board. download and run the I2C master code, the LED is toggled to indicate I2C communication.
4. See LED changes on both I2C master and slave boards.
5. Check I2C master serial terminal output and input the new data.

I2C Connections

Connection on TWR-K64F120M:

	Master	Slave
I2C1_SCL	PTC10 Primary A75	I2C1_SCL PTC10 Primary A75
I2C1_SDA	PTC11 Primary A60	I2C1_SDA PTC11 Primary A60
GND	Primary A65	GND Primary A65

Connection on FRDM-K64F120M:

	Master	Slave
I2C0_SCL	PTE24 J2 Pin18	I2C0_SCL PTE24 J2 Pin18
I2C0_SDA	PTE25 J2 Pin20	I2C0_SDA PTE25 J2 Pin20
GND	J2 Pin14	GND J2 Pin14

Chapter 10
I2C DAC Demo

This demo application simulates an I2C DAC.

Modules

- I2C DAC Introduction
  Introduction and the function/configuration of the I2C DAC slave and master serial terminals.
- I2C DAC User Guide
  User guide on how to use the I2C DAC master and slave applications and how to copy it.

10.1 Detailed Description

10.2 I2C DAC Introduction

Introduction and the function/configuration of the I2C DAC slave and master serial terminals.

10.2.0.19 I2C DAC Introduction

Overview

The I2C DAC project is a simple simulation program that utilizes the I2C drivers contained in the KSDK software.

The slave receives commands allowing the user to configure a buffer, latch the DAC output, reconfigure the slave address, or put the slave into sleep mode.

A master application is also included allowing the user to send to the slave any command that the slave supports.

Supported platforms

Currently, the I2C DAC project is supported on the following platforms:

- TWR-K64F120M
- FRDM-K64F120M

Serial terminal configurations

The master and slave serial terminal should be configured with the following settings:

- 115200 baud
- 8 data bits
- No parity
- 1 stop bit

10.3 I2C DAC User Guide

User guide on how to use the I2C DAC master and slave applications and how to copy it.

10.3.0.20 I2C DAC User Guide

Getting started

The I2C DAC project includes a master project and a slave project. These projects are designed to be programmed on two different development boards with the master sending commands to the slave.

Begin by programming the master application onto the desired development board and then program the slave application onto the desired slave board.

I2C Connections

The I2C protocol is a two-wire interface that consists of a clock line (SCL) and a data line (SDA). The master and slave SCL and SDA lines must be connected together (master SCL to slave SCL and master SDA to slave SDA) for proper demo operation. For this demo application, please make the necessary connections using the following table as a reference.

Platform Pin Pin Pin

Signal CPU Name Board Name

TWR-K64F120M SCL PTC10 A75 SDA PTC11 A60 GND GND A2

FRDM-K64F120M SCL PTE24 J2-18 SDA PTE25 J2-20 GND GND J25-14

Supported slave commands

The slave accepts four commands from the master. Any command sent to the slave that is not one of the four accepted commands is ignored. The slave prints a message to the terminal informing the user that an invalid command was received and reset the receive buffer. The supported slave commands are listed here:

- 1: This command allows the user to input three new RGB values to the simulated DAC buffer.
- 2: This command latches the simulated DAC buffer to the output.
- 3: This command should follow the General Call address only. This command forces the slave to sample the address pins and re-initialize the slave address without resetting the device.
- 4: This command puts the slave into VLPS mode.

Slave command structure

The commands sent to the slave follow this structure:

[Slave Address] [Command] [Data (Optional)]

Only command #1 accepts additional data. This command accepts three 8-bit data values after the command byte. The slave stores these bytes in the simulated DAC buffer as the R-value, G-value, and B-value in that order.

Using the master to send commands to the slave.

Upon startup, the master prints this menu to the serial terminal:

Select from the following menu:

- 1: Set new RGB value ∗
- 2: Output RGB value ∗
- 3: Latch new address without resetting ∗
- 4: Set DAC to sleep mode ∗

Once the menu has been printed, enter the number of the desired command. The slave should respond accordingly.

If you enter a command that is not listed on the menu, the master prints the following message to the terminal:

Invalid command entered!

The master will then wait for user input.

Chapter 11
I2C Demo with RTOS

This demo is a I2C commnication application running on different RTOS.

Modules

- I2C Demo with RTOS Introduction
  Introduction and the configuration of the I2C master/slave demo with RTOS support.
- I2C Demo with RTOS User Guide
  User Guide on how to use the I2C demo with RTOS.

11.1 Detailed Description

11.2 I2C Demo with RTOS Introduction

Introduction and the configuration of the I2C master/slave demo with RTOS support.

11.2.0.21 I2C Demo with RTOS Introduction

Overview

This I2C application demonstrates how KSDK peripheral drivers work on different RTOS. The application acts as both I2C Master and Slave device on different I2C bus, e.g. I2C Master on I2C0 bus, I2C Slave on I2C1 bus. It can be running on one single board or on two different boards. When connecting the two I2C buses on one board, the Master sends commands from I2C0 bus to the slave on I2C1 bus. When connecting the I2C0 bus to the other board's I2C1 bus, the application running on the first board acts as Master, and sends commands to the other board which acts as slave. That means the first board can send commands and get responses from the other board by I2C bus. The basic function of this demo is:

1. Read the Kinetis chip UID(low 32bit) from the slave board
2. Read the Kinetis chip internal temperature from the slave board
3. Control the RED/GREEN/BLUE color LED on/off on slave board

The application creates three different tasks to handle events concurrently:

1. Master task: responsible for user interface interaction, running as a I2C Master and acting as a S-HELL. It accepts user's command to read the basic chip UID, chip temperature and control the on board LED, and power mode on slave side.
2. Slave task: responsible for receiving the command from I2C Master and returning the result to Master.
3. ADC sample task: responsible for getting the chip temperature in a period.
4. For BM version, Master and Slave task are separated into two projects.

Supported RTOS

- Freescale MQX™ RTOS
- FreeRTOS
- uC/OS-II
- uC/OS-III
- Bare-Metal (no RTOS)

Supported platforms

Currently, this demo supports the following platforms:

- FRDM-K64F120M
- TWR-K64F120M

Serial terminal configuration

The debug serial terminal is configured for the following settings:

- 115200
- 8 data bits
- No parity
- 1 stop bit

11.3 I2C Demo with RTOS User Guide

User Guide on how to use the I2C demo with RTOS.

11.3.0.22 I2C Demo with RTOS User Guide

Getting started

The I2C RTOS application is designed to work on one single board or two different boards(BM version only support two boards).

Build with different RTOS support

Before running this application, you should build it with the correct RTOS you want to run. For IAR, the projects to support different RTOS is distinguished by the workspace file name: i2c_rtos_<rtos>.eww, e.g. i2c_rtos_ucosii.eww is uC/OS-II version of this application. Open the right workspace, first build the platform_<rtos>_lib project, then build the application project. The binary i2c_rtos_<rtos>.out will be generated.

Hardware Connections for I2C bus

FRDM-K64F120M

- Single board

Master	Slave
I2C0_SCL PTE24 (Pin 18 on J2)	I2C1_SCL PTC10 (Pin 10 on J26)
I2C0_SDA PTE25 (Pin 20 on J2)	I2C1_SDA PTC11 (Pin 12 on J26)

- Two boards

Master (board #1)	Slave (board #2)
I2C0_SCL PTE24 (Pin 18 on J2)	I2C1_SCL PTC10 (Pin 10 on J26)
I2C0_SDA PTE25 (Pin 20 on J2)	I2C1_SDA PTC11 (Pin 12 on J26)
GND (Pin 14 on J2)	GND (Pin 14 on J2)

TWR-K64F120M

- Single board

Master	Slave
I2C0_SCL PTD8 (Pin A7 on J11A)	I2C1_SCL PTC10 (Pin A75 on J11A)

Master	Slave
I2C0_SDA PTD9 (Pin A8 on J11A)	I2C1_SDA PTC11 (Pin B71 on J11A)

- Two boards

Master (board #1)	Slave (board #2)
I2C0_SCL PTD8 (Pin A7 on J11A)	I2C1_SCL PTC10 (Pin A75 on J11A)
I2C0_SDA PTD9 (Pin A8 on J11A)	I2C1_SDA PTC11 (Pin B71 on J11A)
GND (Pin A65 on J11A)	GND (Pin A65 on J11A)

Run the application

1. Connect I2C Clock, Data and GND signal as described above.
2. Power on the board (boards), download the application into flash and run.
3. Open the PC console to connect to the Master board.
4. When successfully connected, you can input help to get the supported command list:

```
Available Commands:
help     - print command description/usage
led     (1)- Light on/off the RGB LED
readtemp(2)- Get the temperature of the client
readid  (3)- Read the client unique id
```

LED Command

- Light the LED on/off on the slave board.
  - Usage: led R/G/B on/off
  - Example: Turn the RED LED on

    ```
    >>led R on
    ```

Temperature Get Command

- Get the chip temperature on the slave board.
  - Usage: readtemp
  - Example:

    ```
    >>readtemp
    Client Chip Temperature:37C
    ```

Read chip UID Command

- Get the slave chip UID
  - Usage: readid
  - Example:

    ```
    >>readid
    Client ID:0x10013
    ```

Chapter 12
Low Power ADC Demo

This demo application simply turn on a red LED if temperature rise above average or turn on a blue one if temperature drops below average.

Modules

- Low Power ADC Introduction
  Introduction and the function/configuration of the LowPower_ADC demo application.
- Low Power ADC User Guide
  User guide on how to use the LowPower_ADC application.

12.1 Detailed Description

12.2 Low Power ADC Introduction

Introduction and the function/configuration of the LowPower_ADC demo application.

12.3 Low Power ADC User Guide

User guide on how to use the LowPower_ADC application.

12.3.0.23 Low Power ADC User Guide

Getting started

The Low Power ADC project is designed to work with the Tower System or in a stand alone setting.

1) Set your target board in a place where temperature is constant. 2) Download the program to a target board. 3) Wait until the green LED light turns on. 4) Increment or decrement temperature to see the changes.

Caution

The use of liquids and some gases can cause a short circuit. Placing the board in contact with very hot or cold materials can damage it. Do not touch the microcontroller or any other part of the board with your tongue.

Overview

The Low Power ADC project is a simple demonstration program that uses the KSDK software. The microcontroller is set to a very low power stop (VLPS) mode, and every 500 ms an interrupt wakes up the ADC module and takes the current temperature of the microcontroller. While the temperature remains between boundaries, both LEDs are off. If the temperature is higher than average, a red LED comes on. If it is lower, a blue LED comes on. The intent of this program is to provide a simply example of how ADC works during a VLPS mode and to provide a simple debugging, "golden", project.

Supported platforms

Currently, the Low Power ADC project is supported on these platforms:

- TWR-K64F120M
- FRDM-K64F120M

Chapter 13
Low Power Demo

A demo application demonstrates different low power modes.

Modules

- Lowe Power Demo Introduction
  Introduction of the low power demo application.
- Low Power Demo User Guide
  User guide on how to execute this application and console setup etc.

13.1 Detailed Description

13.2 Lowe Power Demo Introduction

Introduction of the low power demo application.

13.2.0.24 Low Power Demo Introduction

Overview

This is a low power demo application which demonstate different low power modes supported in the Kinetis SoCs - those are WAIT, STOP, VLPR, VLPW, VLPS, LLS, VLLS. There is sequence to be followed to enter/exit each mode, and functionality availabile in different power modes are different. Application demonstrates multiple mode entry/exit cases as per the functionality availability.

Supported Hardware

- FRDM-K64F120M (OpenSDA/TWR-SER as debug UART, SW1 and SW3 as wakeup sources)
- TWR-K64F120M (OpenSDA/TWR-SER as debug UART, SW2 and SW3 as wakeup sources)

Default Debug Console

- UART instance: OpenSDA CDC/RS-232 serial cable connected to TWR-SER
- Baudrate: 19200bps

Execution instruction

There are many demo cases listed out of reset in the serial console. Upon selection, further actions are either requested or execution o/p is printed on console. The user needs to follow them in order to get the desired output.

13.3 Low Power Demo User Guide

User guide on how to execute this application and console setup etc.

13.3.0.25 Low Power Demo User Guide

Getting Started

1. Open the UART console on PC. Set the baudrate to 19200 with 8 bit data, no parity, 1 stop bit, and no flow control.
2. Download and run the program on the target. Follow instructions shown during execution of different demo cases.

Note

After the program is downloaded to the target and executed, by default the debug port is disabled. To re-enable the debug port, user should follow the correct option shown in console. This might be useful to connect a debugguer or download the demo application again to the target.

To execute the low power demo use cases, user should disable the debugger (that is the default setting of the application). The debugger can be turned back on if debugging is required.

Chapter 14
RTC Demo

RTC demo.

Modules

- RTC Demo Introduction
  Introduction and the function/definition comments of this demo.
- RTC Demo User Guide
  User guide on how to use this demo.

14.1 Detailed Description

14.2 RTC Demo Introduction

Introduction and the function/definition comments of this demo.

14.2.0.26 RTC Demo Introduction

Overview

The RTC Demo project is a simple demonstration program based on KSDK RTC peripheral driver. It uses the UART driver to show a clock like application and display hours, minutes, and seconds. The alarm feature will be added subsequently.

Supported platforms

Currently, the RTC demo is supported on the following platforms:

- TWR-K64F120M
- FRDM-K64F120M

Serial terminal configuration

The serial terminal is configured for the following settings:

- 115200
- 8 data bits
- No parity
- 1 stop bit

14.3 RTC Demo User Guide

User guide on how to use this demo.

14.3.0.27 RTC Demo User Guide

Getting started

1. Open terminal tool, such as Tera Term, and connect to port by using OpenSDA.
2. Flash the code and run.
3. When the code is running, the terminal shows a clock with hours and minutes and an extra field for seconds. It updates each second.
4. The alarm feature will be added subsequently.

Chapter 15
RTC Function Demo

This demo application demonstrates RTC basic functions.

Modules

- RTC Function Demo Introduction
  Introduction of the RTC Functions demo application.
- RTC Function Demo User Guide
  User guide on how to use RTC Function Demo application.

15.1 Detailed Description

15.2 RTC Function Demo Introduction

Introduction of the RTC Functions demo application.

15.2.0.28 RTC Function Demo Introduction

Overview

This is an RTC demo application which demonstrates the important features of the RTC Module. It supports the following features:

- Get the current date time with Year, Month, Day, Hour, Minute, and Second.
- Set the current date time.
- Set the alarm based on current time. Application prints a notification when the alarm has expired.
- Get the alarm configurations.
- Get the accurate current time in 30 us resolution.
- Configure the compensation with cycles and intervals.
- Get the compensation configurations.

Supported Hardware

- TWR-K64F120M
- FRDM-K64F120M

Default Debug Console

- UART instance: OpenSDA CDC
- Baudrate: 115200 bps

15.3 RTC Function Demo User Guide

User guide on how to use RTC Function Demo application.

15.3.0.29 RTC Function Demo User Guide

Getting started

1. Download the program to a target broad.
2. Connect the USB cable to the PC host and open a serial terminal at 115200 baud, 8-bits no parity, 1 stop bit.
3. When successfully connected, you can input help to get the supported command list:

```
Available Commands:
help    (0)- print command description/usage
alarm   (1)- Set an alarm
date    (2)- Get and set the datetime
time    (3)- Get precise time in nanosec
comp    (4)- Compensate demo
```

Alarm Command

- Get current armed alarm.
  - Usage: alarm
  - Example:

    ```
    RTC Demo> alarm
    No alarm armed
    ```

- Set an alarm
  - Usage: alarm <sec>
  - Example: Set the alarm 10 sec later.

    ```
    RTC Demo> alarm 10
    Current alarm: 15:46:13
    ```

Date Command

- Get date time
  - Usage: date
  - Example:

    ```
    RTC Demo> date
    Current datetime: 2013-12-12 16:46:39
    ```

- Set date time
  - Usage: date set <YYYY-MM-DD> <hh:mm:ss>
  - Example: Set datetime to 12/12/2013 16:46:39

    ```
    RTC Demo> date set 2013-12-12 16:46:39
    ```

Time Command

- Get current time in us resolution
  - Usage: time
  - Example:

```
RTC Demo> time
Current time: 15:46:03 267ms 870us
```

Compensation Command

- Get current compensation configurations
  - Usage: comp
  - Example:

```
RTC Demo> comp
Compensation, cycles: 32768, interval: 4
```

- Config compensation cycles and interval
  - Usage: comp <cycles> <interval>
    - Cycles: the number of 32.768 kHz clock cycles in each second. Range from 32641 Hz to 32896 Hz.
    - Interval: the compensation interval from 1 to 256 seconds to control how frequently the TCR should adjust the number of 32.768 kHz cycles in each second. The value written should be one less than the number of seconds. Range from 0 to 255.
  - Example:

```
RTC Demo> comp 32780 4
```

The compensation result can be checked from the RTC_CLKOUT external output pin which outputs a 1 Hz signal with the compensated result. The RTC_CLKOUT pin can be found:

- TWR-K64F120M: pin2 on J31
- FRDM-K64F120M: pin1 on J2

Chapter 16
SAI Modulator

This demo application is an audio waveform modulation demonstration program.

Modules

- SAI Modulator Introduction
  Introduction and the function/definition comments of this demo.
- SAI Modulator User Guide
  User guide on how to use this demo.

16.1 Detailed Description

16.2 SAI Modulator Introduction

Introduction and the function/definition comments of this demo.

16.2.0.30 SAI MODULATOR Introduction

Overview

The SAI Modulator project is an audio waveform modulation demonstration program that utilizes the K-SDK software. It performs modulation of WAV file data using the CMSIS-DSP FFT libraries and plays sound through a TWR-AUDIO-SGTL board using the I2S & I2C drivers of the KSDK. It also features a terminal menu for selecting the different types of waveform modulation, the individual note (WAV data), and the duration of play.

Supported platforms

Currently, the SAI Modulator project is supported on these platforms:

- TWR-K64F120M

16.3 SAI Modulator User Guide

User guide on how to use this demo.

16.3.0.31 SAI MODULATOR User Guide

Play PCM audio data from the WAV format array.

Getting started

1. Hardware settings

SAI_Modulator program requires the following hardware:

- TWR-K64F120M
- TWR-ELEV
- TWR-SGTL

2. Download program

- Download the program to the target board.
- Connect a USB cable to the PC host and open a serial terminal at 115200 baud, 8-bits no parity, 1 stop bit.
- When successfully connected, the program sends "Press spacebar to start demo" to the terminal. Then press space key to continue. Communication Interface Settings:
  - TWR-K64F120M(OpenSDA) UART1 TX: PC3 RX: PC4.

3. Run the program

If a serial terminal echo inputs normally,

```
Audio Modulator Demo!

Press spacebar to start demo.
```

Use headphones connected to the TWR-SGTL's HEADSET socket and press the spacebar in the terminal window.

```
Demo begin...
```

It plays a C major arpeggio and prints out a menu in the terminal window.

```
Select note to play:
    1. C4
    2. D4
    3. E4
    4. F4
    5. G4
```

```
    6. A4
    7. B4
    8. C5
->
```

Enter the number that corresponds to the note you wish to hear and a duration menu appears.

```
Select note duration:
    1. 200ms
    2. 100ms
    3. 50ms
->
```

Default duration of the notes in FLASH is 200 ms; Select the number that corresponds with the duration of the note you want. Then, a modulation type menu is printed in the terminal window.

```
Select note modulation:
    1. none
    2. Square
    3. Saw
    4. Triangle
->
```

Select the type of waveform you wish to modulate the pure tone to. If a non-pure tone is selected, the processing message appears.

```
Processing...
```

Followed by the completed message.

```
Completed!
```

At this point, you hear the note with the desired characteristics played through the headphone jack of the TWR-AUDIO-SGTL board.

Key Functions

1. play_wav

audio_status_t play_wav(uint32_t *pcmBuffer, uint8_t divider);

Parameters

pcmBuffer	Pointer to a data array containing WAV formatted audio data.
divider	Integer to divide PCM data length to adjust the audio sample duration.

Returns

status_t Return kStatus_Success if function completed successfully; return kStatusFail if function failed.

2. play_mod_wav

audio_status_t play_mod_wav(uint16_t ∗pcmBuffer, uint16_t ∗modPointer, float32_t ∗fftData, float32_t ∗fftResult, uint8_t divider, uint8_t modulation, uint32_t srcSizeBytes, uint32_t sampleSize);

Play modulated PCM audio data from a WAV format array.

Parameters

pcmBuffer	Pointer to a data array containing the WAV formatted audio data.
modBuffer	Pointer to a data array to store the modulated PCM data.
fftData	Pointer to a data array for storing the Fast Fourier Transform data.
fftResult	Point to a data array for storing real frequency bins from FFT.
divider	Integer to divide PCM data length to adjust an audio sample duration.
modulation	Constant representing type of waveform achieve from modulation.
srcSizeBytes	Size of the WAV format array being processed.
sampleSize	Size of a sample for FFT (each sample is 16-bit integer from the PCM data).

Returns

status_t Return kStatus_Success if function completed successfully; return kStatusFail if function failed.

3. mod_wav_data

float32_t mod_wav_data(uint16_t ∗pcmBuffer, uint16_t ∗modBuffer, float32_t ∗fftData, float32_t ∗fft-Result, uint16_t startIndex, uint32_t sampleSize, uint8_t modType);

Find fundamental frequency of the PCM data in a WAV format array.

Parameters

pcmBuffer	Pointer to a data array containing the WAV formatted audio data.
modBuffer	Pointer to a data array to store the modulated PCM data.
fftData	Pointer to a data array for storing the Fast Fourier Transform data.
fftResult	Point to a data array for storing real frequency bins from FFT.
startIndex	Starting index of array to get the PCM data.
sampleSize	Size of a sample for FFT (each sample is 16-bit integer from PCM data).
modType	Constant representing type of waveform achieve from modulation.

Returns

float32_t Returns fundamental frequency in Hz.

Chapter 17
SAI Play Sound

Play Windows PCM wave file through SAI bus.

Modules

- SAI Play Sound Introduction
  Introduction and the function/definition comments of this demo.
- SAI Play Sound User Guide
  User guide on how to use this demo.

17.1 Detailed Description

17.2 SAI Play Sound Introduction

Introduction and the function/definition comments of this demo.

17.2.0.32 SAI Play Sound Introduction

Overview

The SAI_PLAYSOUND demo project is a simple demonstration program based on KSDK SAI peripheral driver. It demonstrates the performance and features of the SAI peripheral driver. For this demo, SAI module is the controller and Freescale Low power Stereo Codec SGTL5000 is the codec. SAI_PLAYSOUND demo features include:

1. Playing audio through TWR-SGTL and volume adjust
2. Support for a different type of Windows PCM wave file format
3. Use of a shell system to demonstrate features
4. Decoding and displaying wave file information on the shell

Function Description

These are the three functions set in playsound.h

The prototypes for the three functions are:

```
void PLAYSOUND_Init(void);

status_t PLAYSOUND_GetWaveFileInfo(uint8_t* pBuffer, wave_file_t *pWave);

status_t PLAYSOUND_Play(APP_GET_DATA_FUNC fpAppGetData);
```

See the playsound.h file for more information about the functions.

17.3 SAI Play Sound User Guide

User guide on how to use this demo.

17.3.0.33 SAI Play Sound User Guide

Getting started

1. Hardware settings

SAI_PlaySound program needs this hardware:

- TWR-K64F120M
- TWR-ELEV
- TWR-SGTL

2. Download program

- Download the program to the target board.
- Connect a USB cable to the PC host and open a serial terminal at 115200 baud, 8-bits no parity, 1 stop bit.
- When successfully connected, the program sends the welcome information to the terminal. Communication Interface Settings:
  - TWR-K64F120M(OpenSDA) UART1 TX: PC3 RX: PC4.

3. Run the program

If a serial terminal echo inputs normally, use headphones connected to the TWR-SGTL's HEADSET socket and type "play" on the terminal. It plays a short audio clip.

Select different format of wave file

To change the format of a wave data source, change the include file located in main.c.

```
// Windows PCM wave data singed 16 bit, sample rate:44.1KHZ, stereo
#include "PCM_S16B_44100_2C.h"
```

All available wave files are:

- PCM_S16B_11025_2C.h
- PCM_S16B_44100_1C.h
- PCM_S16B_44100_2C.h
- PCM_S32B_44100_2C.h

Note

1. SGTL5000 cannot support 8-bit waveform data.
2. Wave data source header files are generated by any "BIN to C" tools. They are available on the Internet.

Add your own audio clip

1. Convert your audio file into the Windows PCM wave file format.

Note

The wave file should be less than 500 KB provided the internal flash is used as memory media.

2. Write call back function "AppGetData"

(1) Function "PLAYSOUND_Play"

Function "PLAYSOUND_Play" transfers wave data into the codec. The entire wave data does not have to be loaded into memory.

Function prototype:

```
status_t PLAYSOUND_Play(APP_GET_DATA_FUNC fpAppGetData);
```

(2) Prototype of "GetData" function

```
uint32_t AppGetDataCallBack(uint32_t NumBytesReq,  uint8_t **ppData, uint32_t offest)
```

(3) Description of "GetData" function

The function returns the number of requested bytes. The maximum number of bytes is NumBytesReq.

(4) Example

The following code excerpt shows how to use the function:

```
uint32_t AppGetDataCallBack(uint32_t NumBytesReq,  uint8_t **ppData, uint32_t offest)
{
    uint32_t NumBytesRead = 0;
    // Check if NumBytesReq is allowed for your data buffer size.
    ... /* TBD */
    // set the data pointer to your memory devices buffer.
    ... /* TBD */
    // return how many bytes we can get, return value should equal or less then NumBytesReq.
    ... /* TBD */
}
```

Note: sgtl5000 only supports 16bit, 20bit, 24bit and 32bit. Note: 20bit and 24 bit need to do some special operations in application, current application now do not support this. Note: Sgtl5000 only supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.

Chapter 18
Shell Test Demo

A demo application demonstrates shell system in apps/utilities folder.

Modules

- Shell Test Demo Introduction
  Introduction of the Shell Test Demo application.
- Shell Test Demo User Guide
  User guide on how to customize this application for different configurations.

Data Structures

- struct shell_io_install_t
- struct cmd_tbl_t
  Access command structure. More...

Enumerations

- enum command_ret_t {
  CMD_RET_SUCCESS,
  CMD_RET_FAILURE,
  CMD_RET_USAGE }

Functions

- uint8_t shell_register_function (const cmd_tbl_t *pAddress)
  register a user function
- void shell_register_function_array (const cmd_tbl_t *pAddress, uint8_t num)
  register function array
- uint8_t shell_unregister_function (char *name)
  unregister function
- int shell_printf (const char *format,...)
  use vsprintf for format.
- void shell_beep (void)
  beep consult
- const cmd_tbl_t * shell_find_command (const char *cmd)
  find command form command struct's name
- uint8_t shell_io_install (shell_io_install_t *IOInstallStruct)
  install shell io function.
- const cmd_tbl_t ** shell_get_cmd_tbl (void)
  get global cmdstruct table address
- char ** shell_get_hist_data_list (uint8_t *num, uint8_t *cur_index)
- void shell_main_loop (char *prompt)

Variables

- uint8_t(∗ shell_io_install_t::sh_getc)(void)
- void(∗ shell_io_install_t::sh_putc)(uint8_t ch)
- char ∗ cmd_tbl_t::name
- uint8_t cmd_tbl_t::maxargs
- uint8_t cmd_tbl_t::repeatable
- int(∗ cmd_tbl_t::cmd)(int argc, char ∗const argv[ ])
- char ∗ cmd_tbl_t::usage
- char ∗ cmd_tbl_t::help
- int(∗ cmd_tbl_t::complete)(int argc, char ∗const argv[ ], char last_char, int maxv, char ∗cmdv[ ])

18.1 Detailed Description

18.2 Data Structure Documentation

18.2.1 struct shell_io_install_t

Data Fields

- uint8_t(∗ sh_getc)(void)
- void(∗ sh_putc)(uint8_t ch)

18.2.2 struct cmd_tbl_t

Data Fields

- char ∗ name
- int32_t maxargs
- int32_t(∗ cmd)(int32_t, char ∗[ ])
- uint8_t maxargs
- uint8_t repeatable
- int(∗ cmd)(int argc, char ∗const argv[ ])
- char ∗ usage
- char ∗ help
- int(∗ complete)(int argc, char ∗const argv[ ], char last_char, int maxv, char ∗cmdv[ ])

18.2.2.0.0.1 Field Documentation

18.2.2.0.0.1.1 int32_t(∗ cmd_tbl_t::cmd)(int32_t, char ∗[])

18.2.2.0.0.1.2 int32_t cmd_tbl_t::maxargs

18.3 Enumeration Type Documentation

18.3.1 enum command_ret_t

Enumerator

CMD_RET_SUCCESS
CMD_RET_FAILURE

CMD_RET_USAGE

18.4 Function Documentation

18.4.1 void shell_beep (void)

18.4.2 const cmd_tbl_t* shell_find_command (const char * *cmd*)

18.4.3 const cmd_tbl_t** shell_get_cmd_tbl (void)

18.4.4 char** shell_get_hist_data_list (uint8_t * *num*, uint8_t * *cur_index*)

18.4.5 uint8_t shell_io_install (shell_io_install_t * *IOInstallStruct*)

18.4.6 void shell_main_loop (char * *prompt*)

18.4.7 int shell_printf (const char * *format*, ...)

18.4.8 uint8_t shell_register_function (const cmd_tbl_t * *pAddress*)

18.4.9 void shell_register_function_array (const cmd_tbl_t * *pAddress*, uint8_t *num*)

18.4.10 uint8_t shell_unregister_function (char * *name*)

18.5 Variable Documentation

18.5.1 int(∗ cmd_tbl_t::cmd)(int argc, char ∗const argv[])

18.5.2 int(∗ cmd_tbl_t::complete)(int argc, char ∗const argv[], char last_char, int maxv, char ∗cmdv[])

18.5.3 char∗ cmd_tbl_t::help

18.5.4 uint8_t cmd_tbl_t::maxargs

18.5.5 char ∗ cmd_tbl_t::name

18.5.6 uint8_t cmd_tbl_t::repeatable

18.5.7 uint8_t(∗ shell_io_install_t::sh_getc)(void)

18.5.8 void(∗ shell_io_install_t::sh_putc)(uint8_t ch)

18.5.9 char∗ cmd_tbl_t::usage

18.6 Shell Test Demo Introduction

Introduction of the Shell Test Demo application.

18.6.0.1 Shell Test Demo Introduction

Overview

This is a shell test demo application which demonstrates the use of shell system in apps/utilities folder. It uses the KSDK's UART drivers to transmit and receive data to shell. Shell features include:

1. Basic Linux xterm esc sequences support(arrow key, etc.).
2. Some command function templates, such as help and history.
3. Configurable history support.
4. Configurable auto complete support.

Supported Hardware

- TWR-K64F120M
- FRDM-K64F120M

Default Debug Console

- UART instance: OpenSDA CDC
- Baudrate: 115200bps
- 8 data bits
- No parity
- 1 stop bit

18.7 Shell Test Demo User Guide

User guide on how to customize this application for different configurations.

18.7.0.2 Shell Test Demo User Guide

Getting Started

To get started, perform following steps:

1. Download the program to a target board.
2. Connect the USB cable to a PC host and open a serial terminal at 115200 baud, 8-bits no parity, 1 stop bit.
3. Reset the board, after that, shell welcome information will be displayed on terminal. you can type "help" for more information. Communication Interface Settings:

- TWR-K64F120M(OpenSDA) UART1 TX: PC3 RX: PC4.
- FRDM-K64F120M(OpenSDA) UART0 TX: PC17 RX: PC16.

HOWTO Customization

You can customize shell system and register your command to shell. Those macro located in shell_config.-h.

Functional configuration

to enable shell auto complete support, use the following:

```
#define SHELL_CONFIG_AUTO_COMPLETE /* config if use auto complete */
```

to enable shell history support, use the following:

```
#define SHELL_CONFIG_USE_STDIO /* config if use auto complete */
```

use this macro to configure if stdout will be connected to shell input and output.

```
#define SHELL_CONFIG_USE_STDIO /* config if use standard output */
```

Configure consult and history record buffer size

Change the following macro to configure characters for read line input. generally HIST_SIZE should be equal to SHELL_CB_SIZE

```
#define SHELL_CB_SIZE                (128)
#define HIST_SIZE                    SHELL_CB_SIZE
```

Configure max numbers of commands and command arguments

Change the following macro to configure max arguments for all commands.

```
#define SHELL_MAX_ARGS               (8)
```

Change the following macro to configure max function number

```
#define SHELL_MAX_FUNCTION_NUM       (64)
```

Add command

To register your own function to the mini shell, perform those steps:

1. Define a command function table and fill in the table with register information.
2. Two ways to register your command to shell:

use shell_register_function(const cmd_tbl_t * pAddress) to register single commands.

use shell_register_function_array(const cmd_tbl_t * pAddress, uint8_t num) to register mutilple commands.

command function table struct is located in shell.h. as follow:

```
typedef struct
{
    char        *name;      /* command Name         */
    uint8_t     maxargs;    /* maximum number of arguments  */
    uint8_t     repeatable; /* autorepeat allowed?      */
    int     (*cmd)(int argc, char * const argv[]);  /* Implementation function  */
    char        *usage;     /* Usage message    (short) */
    char        *help;      /* Help  message    (long)  */
    int     (*complete)(int argc, char * const argv[], char last_char, int maxv, char *cmdv[]);     /* do
       auto completion on the arguments */
}cmd_tbl_t;
```

Example for register command struct

```
static const cmd_tbl_t CommandFun_Test =
{
    .name = "test",
    .maxargs = 5,
    .repeatable = 1,
    .cmd = DoTest,
    .usage = "help - app test function",
    .complete = NULL,
    .help = "long help - app test function",
};
```

Chapter 19
SPI Flash Demo

SPI Flash access application.

Modules

- SPI Flash Demo Introduction
  Introduction and the function/configuration of SPI Flash functions.
- SPI Flash Demo User Guide
  User guide on how to use SPI Flash Demo.

19.1 Detailed Description

19.2 SPI Flash Demo Introduction

Introduction and the function/configuration of SPI Flash functions.

19.2.0.3 SPI Flash Demo Introduction

Overview

This demo application provides a command line console and a series of commands to access the SPI Flash. These commands can be used either to read, write, or erase the SPI flash.

Supported Hardware

- TWR-K64F120M (OpenSDA as debug UART)
- FRDM-K64F120M (OpenSDA as debug UART)

Default Debug Console

- UART instance: OpenSDA CDC
- Baud rate: 115200 bps

UART Output

```
SF Test >probe 0:0 5000000
SF: Detected AT26DF081A with page size 4096, total 1048576
SF Test >help
SPI flash sub-system
probe [[bus:]cs] [hz] [mode]    - init flash device on given SPI bus
                  and chip select
read addr offset len     - read 'len' bytes starting at
          'offset' to memory at 'addr'
write addr offset len    - write 'len' bytes from memory
              at 'addr' to flash at 'offset'
update addr offset len  - erase and write 'len' bytes from memory
                  at 'addr' to flash at 'offset'
erase offset [+]len        - erase 'len' bytes from 'offset'
          '+len' round up 'len' to block size
help - print usage
SF Test >erase 0x0 0x100000
SF: Successfully erased 1048576 bytes @0x0
SF Test >
```

19.3 SPI Flash Demo User Guide

User guide on how to use SPI Flash Demo.

19.3.0.4 SPI Flash Demo User Guide

Build

SPI_Flash test program needs SPI flash library, so please build SPI flash library first.

File Description

- char_handler.h: Character handle functions used in the command_handler.c.
- cmd_handler.c: Functions to handle the command input console shell.
- cmd_handler.h: API header of the command handler.
- spi_flash_test.c: Main functions that create a shell on the console and let the user access the SPI flash.

Hardware environment

In this demo, SPI is used to access flash on a TWR-MEM board.

Put the development board and TWR-MEM board to a TWR-ELEV board. On a TWR-MEM board, SPI flash can be accessed via SPI0. CS can be selected with J14 on TWR-MEM board.

Connection on board:

Function	TWR-Kinetis	FRDM-Kinetis	TWR-MEM
SPI0_MISO	PTD0	J2-10	B44
SPI0_MOSI	PTD2	J2-8	B45
SPI0_SCLK	PTD1	J2-12	B48
SPI0_CS0	PTD3	J2-6	B46
SPI0_CS1	PTD4	J6-4	B47
GND	n/a	J2-14	B2

Chip selection(CS) can be controlled by J14 pin on TWR-MEM board. Normally, we can insert TWR--Kinetis and TWR-MEM board to TWR-ELEV board.

Please visit http://www.freescale.com for more information about TWR-ELEV and TWR-MEM boards.

Command explanation

Probe

probe [[bus:]cs] [hz] [mode]

This function is used to detect the SPI flash on a designated port. It can accept bus, CS, Hz, and mode as parameters.

- Bus is the SPI module, 0(spi0), 1(spi1).
- CS is chip select index.
- Hz is the operation baud rate frequency that is input to the SPI flash.
- Mode is the clock polarity and clock phase that the SPI flash uses for data communication.

mode	Clock Polarity	Clock Phase
0	Active High	First Edge
1	Active High	Second Edge
2	Active Low	First Edge
3	Active Low	Second Edge

The default value of mode is 0.

Example:

```
SF Test >probe 0:0 5000000
SF: Detected AT26DF081A with page size 4096, total 1048576
```

Help

help

The help command shows help for the supported functions.

Example:

```
SF Test >help
SPI flash sub-system
probe [[bus:]cs] [hz] [mode]    - init flash device on given SPI bus
                  and chip select
read addr offset len     - read 'len' bytes starting at
          'offset' to memory at 'addr'
write addr offset len    - write 'len' bytes from memory
         at 'addr' to flash at 'offset'
erase offset [+]len        - erase 'len' bytes from 'offset'
          '+len' round up 'len' to block size
```

Erase

erase <offset> [+]<len>

This command is used to erase the flash beginning from the designated offset and through the designated length. It can accept offset and len as parameters.

- Offset is the offset in the SPI flash that erase starts.
- Len is the length of data that is erased.
- "+" means round up. If "+" is specified, len is rounded up with the flash sector size.

Example:

```
SF Test >erase 0x0 0x100000
SF: Successfully erased 1048576 bytes @0x0
```

Write

write <addr> <offset> <len>

This command writes data in RAM to flash.

It can accept addr, offset, and len as parameters.

- Addr is the data address in RAM.
- Offset is the offset in the SPI flash that write starts.
- Len is the length of data.

Example:

```
SF Test >write 0x20000000 0x0 0x1000
SF: program success 4096 bytes @ 0x0
```

Read

read <addr> <offset> <len>

This command reads data from the flash. It can accept addr, offset, and len as parameters.

- Addr is the data address in RAM.
- Offset is the offset in the SPI flash that read starts.
- Len is the length of data.

Example:

```
SF Test >read 0x20000000 0x0 0x1000
SF: read success 4096 bytes @ 0x0
```

Update

update <addr> <offset> <len>

This function erases sector and writes data in RAM to flash.

It can accept addr, offset, and len as parameters.

- Addr is the data address in RAM.
- Offset is the offset in SPI flash that read starts.
- Len is the length of data. Ensure that at least 4K RAM can be allocated for the sector size. The update uses the 4K RAM to compare the write data to the data on the flash.

HOWTO Customization

See device/spi_flash for adding a new SPI flash driver.

You should check whether the 'spi_cs_activate' and 'spi_cs_deactivate' functions need to be added.

These two functions pull the CS pin up or down.

For most flashes that have timing requirements between CS pin and clock, it is better to use GPIO to control this pin in software.

Another benefit is that we can use CS GPIO pin to control start and stop of a transfer, instead of using a halt bit.

Example:

```
status_t spi_cs_activate(spi_slave_dev *spi)
{
    switch (spi->cs)
    {
    case 0:
        gpio_write_pin_output(kGpioSpi0Cs0, spi->ss_pol);
    break;
    case 1:
        gpio_write_pin_output(kGpioSpi0Cs1, spi->ss_pol);
    break;
    default:
        break;
    }

    return 0;
}

status_t spi_cs_deactivate(spi_slave_dev *spi)
{
    switch (spi->cs)
    {
    case 0:
        gpio_write_pin_output(kGpioSpi0Cs0, !(spi->ss_pol));
    break;
    case 1:
        gpio_write_pin_output(kGpioSpi0Cs1, !(spi->ss_pol));
    break;
    default:
        break;
    }

    return 0;
}
```

Chapter 20
Watchdog Timer Demo

A demo application demonstrates watchdog signal interrupt and reset when timeout.

Modules

- Watchdog Timer Demo Introduction
  Introduction of the Watchdog Timer demo application.
- Watchdog Timer Demo User Guide
  User guide on how to use this application.

20.1 Detailed Description

20.2 Watchdog Timer Demo Introduction

Introduction of the Watchdog Timer demo application.

20.2.0.5 Watchdog Timer Demo Introduction

Overview

The watchdog demo application demonstrates how the watchdog signals interrupt and reset. The overflow time for a watchdog timer is approximately 2 seconds.

Supported platforms

Currently, the watchdog demo is supported on the following platforms:

- TWR-K64F120M
- FRDM-K64F120M

Serial terminal configuration

The serial terminal is configured for the following settings:

- 115200
- 8 data bits
- No parity
- 1 stop bit

20.3 Watchdog Timer Demo User Guide

User guide on how to use this application.

20.3.0.6 Watchdog Timer Demo User Guide

Getting started

1. Power on.
   - Open terminal tools, such as Putty.
   - Download the code to the board.
2. When the code is running, the watchdog is enabled. The code continuously refreshes the watchdog to prevent the CPU reset.
3. When the LED starts blinking, the "Watchdog example running, Loop #: xx, press <SW> to start watchdog timeout..." message displays on the terminal. on the TWR-K64F120M board, the blink led is yellow LED, and on the FRDM-K64F120M board, the blink led is green LED.
4. When the SW key is pressed, on the TWR-K64F120M board, the yellow LED blinks rapidly signifying that the watchdog is about to expire, on the FRDM-K64F120M board, the red LED blinks rapidly signifying that the watchdog is about to expire(Because the FRDM-K64F120M board has a tri-color LED and at the very beginning the program light the green led. When the red one start to blink, red and green mix togther, the led color that you can see is yellow.).
5. When the watchdog signals a reset, the "Watchdog(COP) Reset" message and "Watchdog(cop) reset count: xx" message output to the terminal.

Note

1) On the TWR-K64F120M board, the pull-up resistor R49 is labeled with "DNP", so this register should be installed first. The SW key is SW1. 2) On the TWR-K64F120M board, the information which is output to the terminal may be incorrect and sometimes display "External Pin Reset" message. This occurs when the watchdog signals a reset to the system and the K20 also sends a pin reset to the K64 MCU through TXS0101. Sometimes, the TXS0101 can’t pull up the pin in time and the duration of the low level of this pin is long. When this occurs, the watchdog reset status is changed by the external pin reset and the message "External Pin Reset" is output to the terminal. 3) On the FRDM-K64F120M board, the pull-up resistor R1 is labeled with "DNP", so this register should be installed first. The SW key is SW2.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: KSDKDEMOUG
Rev. 1.0.0-beta
03/2014