**Freescale Semiconductor, Inc.**
Application Note

Document Number: AN5242
Rev. 0, 01/2016

# Emulating Dual SPI Using FlexIO

## 1. Introduction

This application note discusses one example of how to use FlexIO module to emulate the dual SPI of both master and slave mode at the same time.

FlexIO is a new on-chip peripheral available on some of the Kinetis series microcontrollers. It is highly configurable and capable of emulating a wide range of communication protocols, such as UART, $I^2C$, SPI, and $I^2S$.

The standalone peripheral module FlexIO is used as an additional peripheral module of the microcontroller and is not a replacement of the SPI peripheral. The key feature of this peripheral is that it enables the user to build their own peripheral directly.

This example creates a simple software demo based on KSDK HAL drivers for you to use FlexIO to emulate the dual SPI of both master and slave mode at the same time.

**Contents**

## 2. Overview of the FlexIO module

The FlexIO module has the following main hardware resources:

- Shifter
- Timer
- Pin

The amount of these resources for a given MCU can be read from the FLEXIO_PARAM register. For example, there are 4 shifters, 4 timers, and 8 pins in MKS22FN256.

The following diagram shows a high-level overview of the FlexIO module.

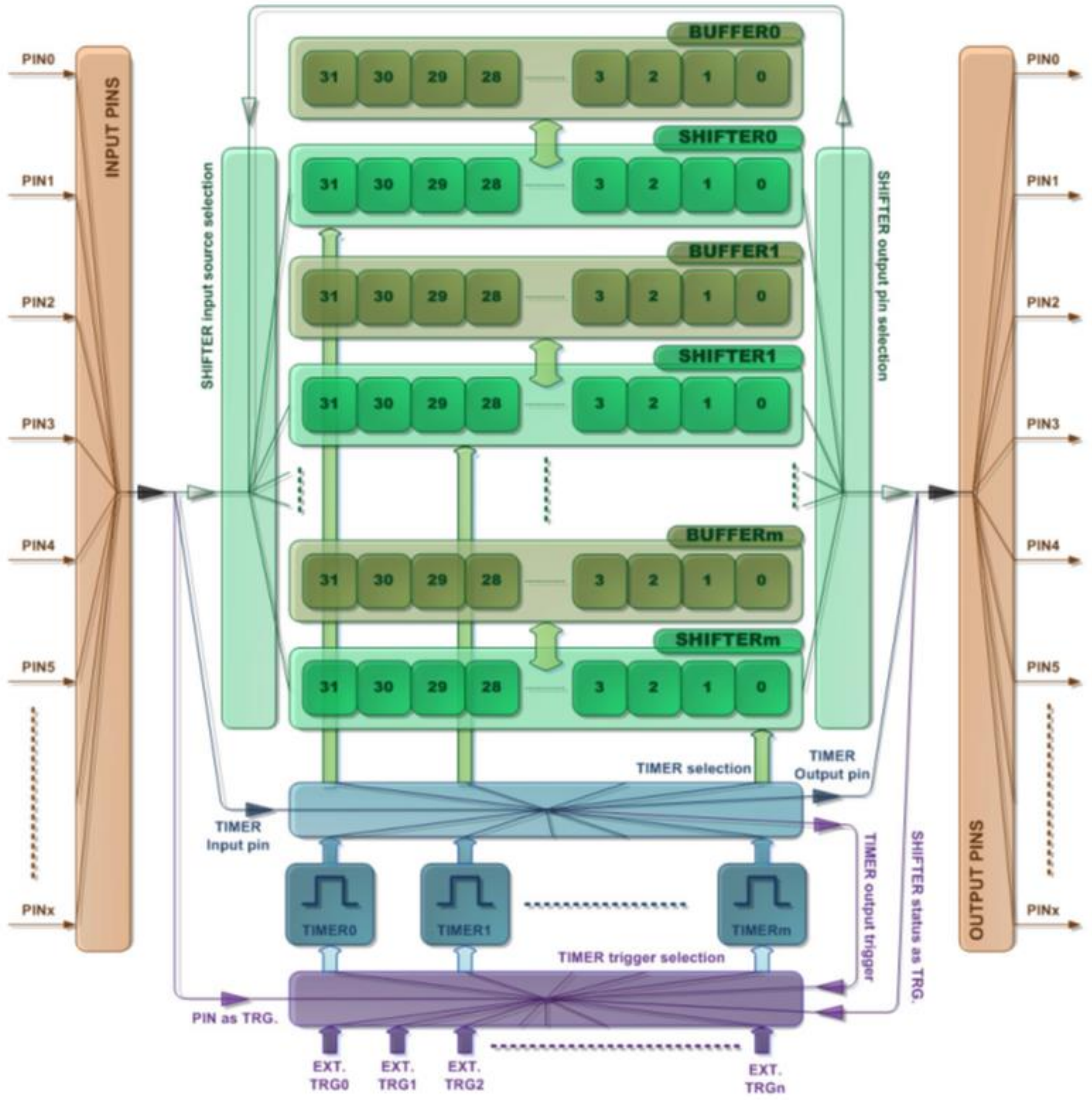


**Figure 1. FlexIO block diagram**

The following key features are provided:

- 32-bit shifters with transmit, receive, and data match modes
- Double buffered shifter operation
- 16-bit timers with high flexibility support for a variety of internal or external triggers, and Reset/ Enable/Disable/ Decrement conditions

- Automatic start/stop bit generation/check
- Interrupt, DMA, or polling mode operation
- Shifters, timers, pins, and triggers can be flexibly combined to operate

Transmit and receive are two basic modes of the shifters. If one shifter is configured to transmit mode, it loads data from its buffer register and shifts data out to its assigned pin bit by bit. If one shifter is configured to receive mode, it shifts data in from its assigned pin and stores data in its buffer register. The load, store, and shift operations are all controlled by the shifter’s assigned timer.

The timers can also be configured as different operation modes according to your requirement, including dual 8-bit counters baud/bit mode, dual 8-bit counters PWM mode, and single 16-bit counter mode.

## 3. Emulating Dual SPI by using FlexIO

This section describes how to emulate dual SPI by using FlexIO. For this application, the Freescale MAPS platform MAPS-KS22F256, which is shown in the below figure, has been used.

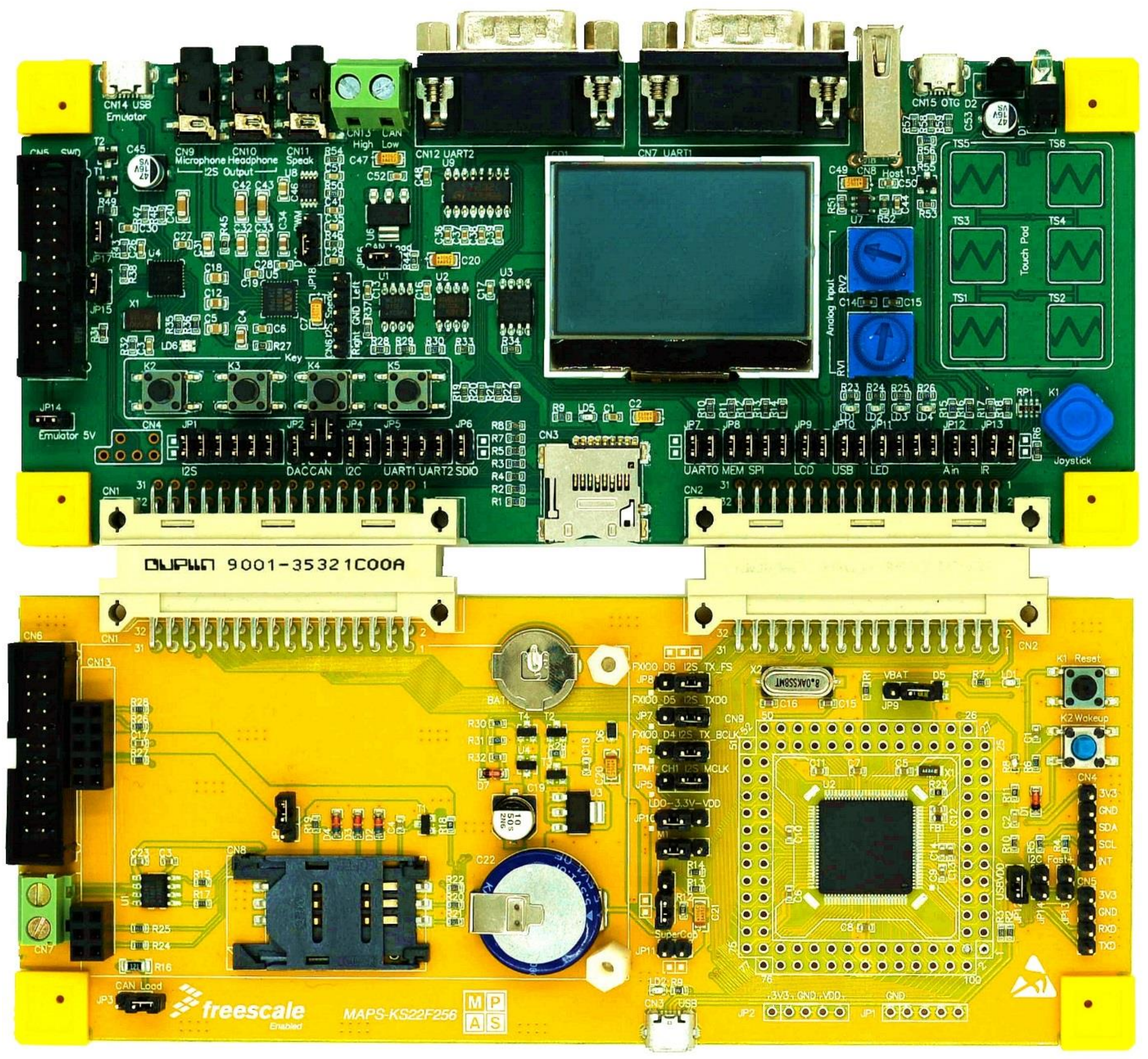


**Figure 2. MAPS platform MAPS-KS22F256**

In this application, FlexIO D0~D3 pins are configured as SPI master, FlexIO D4~D7 pins are configured as SPI slave. Make the connections between the master and slave using 4 external wires. You can use OpenSDA or general UART debug console to check the result of data loopback transfer.

The following diagram shows the hardware platform and data flows:

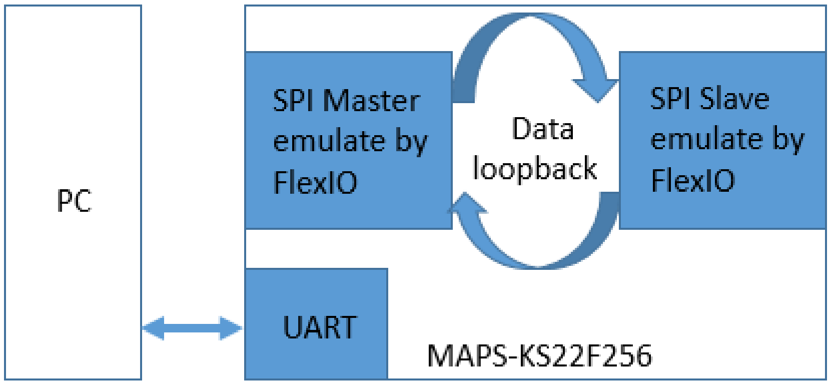


**Figure 3. Hardware platform and data flows**

When you initially open a document from the template, you are prompted to enter the values for the document's metadata.

SPI bus master can be emulated using:

- 2 Shifters: one shifter is used as the data transmitter and the other shifter is the receiver.
- 2 Timers: one timer is used for the SPI_CS output generation, and the other timer is used for the load/store/shift control of the two shifters and SPI_SCK generation.
- 4 Pins: these are used as SPI_CS, SPI_SCK, SPI_SOUT, and SPI_SIN.

The following diagram shows the master resource assignment.

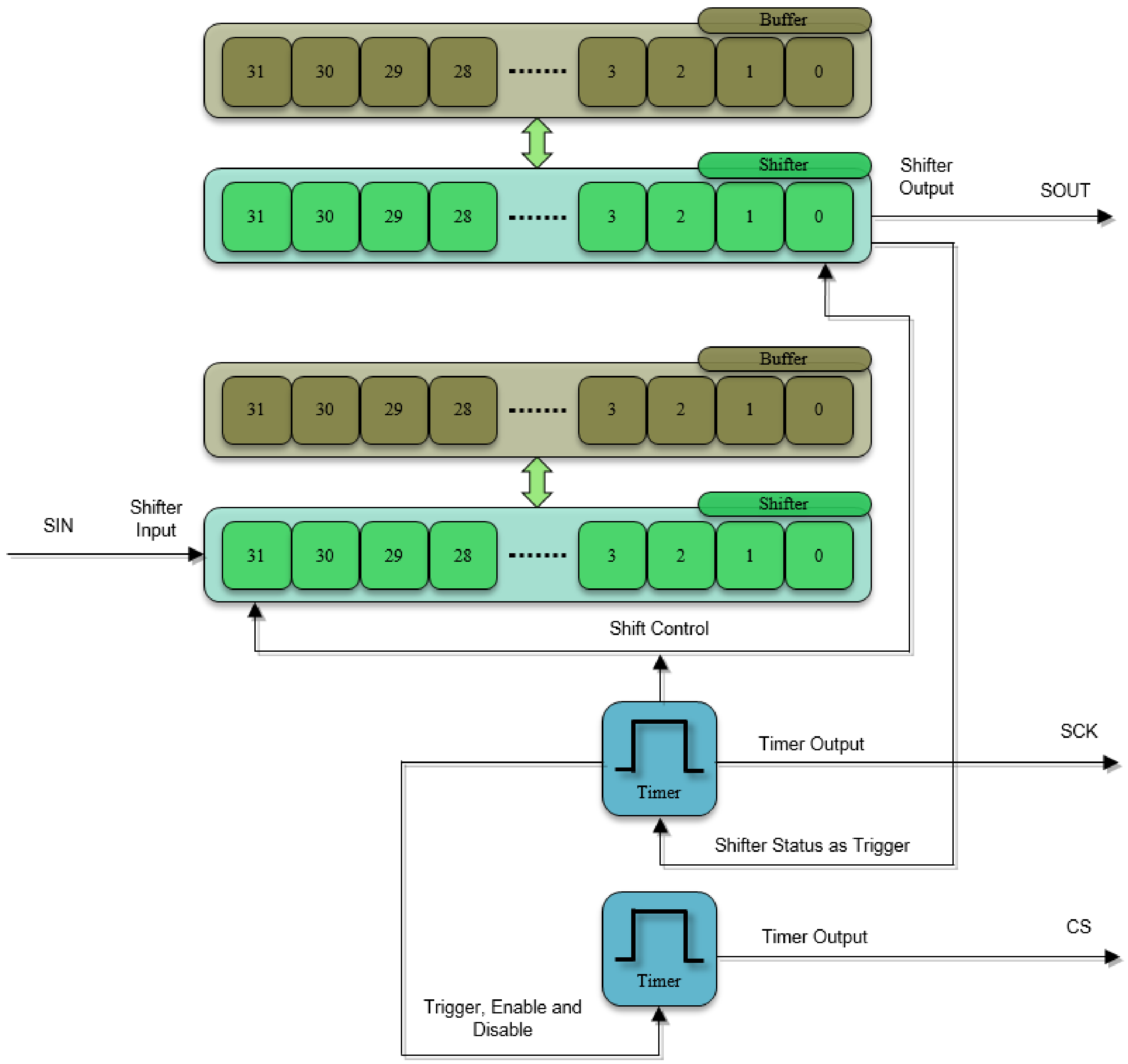


**Figure 4. Resource Assignment of FlexIO to Emulate SPI Master**

The SPI bus slave can be emulated using:

- 2 Shifters: one shifter is used as the data transmitter and the other shifter as the receiver.
- 1 Timer: used for the load/store/shift control of the two shifters.
- 4 Pins: the pins are used as SPI_CS, SPI_SCK, SPI_SOUT, and SPI_SIN.

The following diagram shows the slave resource assignment.

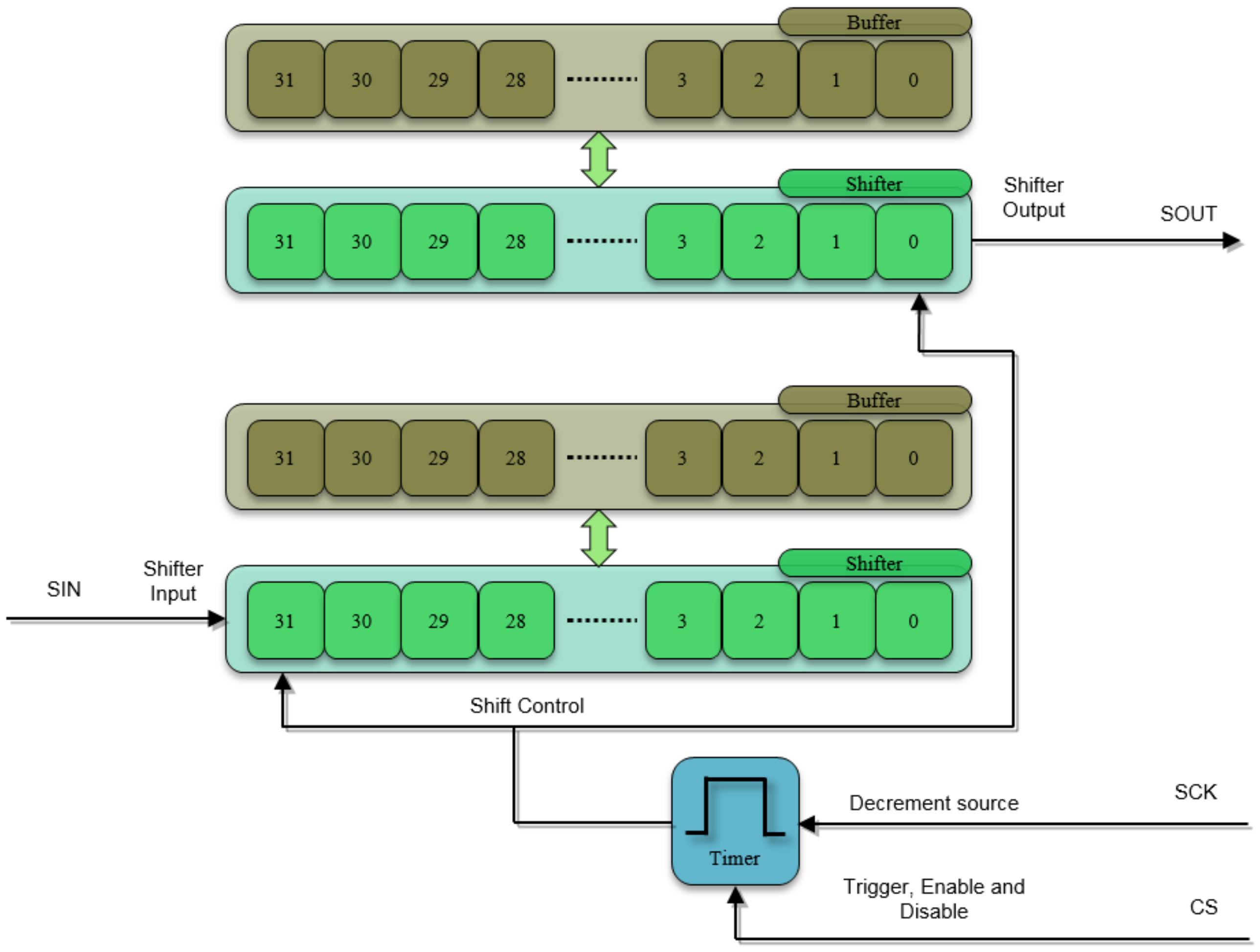


**Figure 5. Resource Assignment of FlexIO to Emulate SPI Slave**

Detailed configurations and usage information are provided in the following sections.

## 3.1 Configurations of the Shifters and Timers

This section provides detailed configurations of the shifters and timers. Note that the items listed in this section are the initial setting with CPHA= 0, SPI baud rate= 2 MHz, and SPI bit count= 8-bit, by default. Some of these settings must be changed by software to support the different SPI features. To understand these configurations, refer to the following sections and the KS22 reference manual.

### 3.1.1 SPI master configurations

Configurations for shifter 0

Shifter 0 is used as the SPI master transmitter on pin FlexIO_D0 as SPI_SOUT. It has the following initial configurations.

**Table 1. Configurations for shifter 0**

| Items | Configurations |
|---|---|
| Shifter mode | transmit |
| Timer selection | timer 0 |
| Timer polarity | on negative of shift clock |
| Pin selection | pin 0 |
| Pin configuration | pin output |
| Pin polarity | active high |
| Input source | from pin |
| Start bit | disabled, transmitter loads data on enable |
| Stop bit | disabled |
| Buffer used | bit byte swapped register |

Configurations for shifter 1

Shifter 1 is used as the SPI master receiver on pin FlexIO_D1 as SPI_SIN. It has the following initial configurations.

**Table 2. Configurations for shifter 1**

| Items | Configurations |
|---|---|
| Shifter mode | receive |
| Timer selection | timer 0 |
| Timer polarity | on positive of shift clock |
| Pin selection | Pin 1 |
| Pin configuration | output disabled |
| Pin polarity | active high |
| Input source | from pin |
| Start bit | disabled, transmitter loads data on enable |
| Stop bit | disabled |
| Buffer used | bit byte swapped register |

Configurations for timer 0

Timer 0 is used by the SPI master to generate SPI_SCK output on pin FlexIO_D2 and load/store/shift control of the two shifters. The shifter status flag is set and cleared each time the SHIFTBUF register is written and read, which means the data in the SHIFTBUF has been transferred to the Shifter (SHIFTBUF is empty). The shifter status flag 0 is configured to be the trigger of the timer 0, so as soon as the SHIFTBUT is written, the status flag is cleared and timer 0 is enabled. The shifter begins to shift out the data on the negative edge of the clock until the timer is disabled. The timer is disabled when the timer counter counts down to 0. Timer 0 has the following initial configurations.

**Table 3. Configurations for timer 0**

| Items | Configurations |
|---|---|
| Timer mode | dual 8-bit counters baud/bit mode. |
| Trigger selection | shifter 0 status flag |
| Trigger polarity | active low |
| Trigger source | internal trigger |
| Pin selection | pin 2 |
| Pin configuration | output enable |
| Pin polarity | active high |
| Timer initial output | output logic 0 when enabled, not affect by reset |
| Timer decrement source | decrement on FlexIO clock, shift on timer output |
| Timer enable condition | on trigger high |
| Timer disable condition | on timer compare |
| Timer reset condition | Timer never reset |
| Start bit | enabled |
| Stop bit | enabled on timer disable |
| Timer compare value | ((n*2-1)<<8) \| (baudrate_divider/2-1)) [1] |

[1] - n is the number of bytes in the transmission. Baudrate_divider is a value used for dividing the baud rate from the FlexIO clock source.

Configurations for timer 1

Timer 1 is used by the SPI master to generate the SPI_CS output on pin FlexIO_D3. Timer 1 is configured to be enabled when the timer 0 is enabled. The compare register is configured to the 16-bit counter and set to 0xFFFF.With this value the timer never compares and is always active when the timer is enabled. Timer 1 has the following initial configurations.

**Table 4. Configurations for timer 1**

| Items | Configurations |
|---|---|
| Timer mode | single 16-bit counter mode |
| Trigger selection | trigger from timer0 |
| Trigger polarity | active high |
| Trigger source | internal trigger |
| Pin selection | pin 3 |
| Pin configuration | output enable |
| Pin polarity | active low |
| Timer initial output | output logic 1 when enabled, not affect by reset |
| Timer decrement source | decrement counter on FlexIO clock, Shift clock equals timer output. |
| Timer enable condition | on timer0 enable |
| Timer disable condition | on timer0 disable |
| Timer reset condition | never reset |
| Start bit | disabled |

Table 4. Configurations for timer 1

| Items | Configurations |
|---|---|
| Stop bit | disabled |
| Timer compare value | 0xFFFF |

### 3.1.2 SPI slave Configurations

Configurations for Shifter 2

Shifter 2 is used by the SPI slave transmitter on pin FlexIO_D4 as SPI_SOUT. It has the following initial configurations.

Table 5. Configurations for Shifter 2

| Items | Configurations |
|---|---|
| Shifter mode | transmit |
| Timer selection | timer 2 |
| Timer polarity | on negative of shift clock |
| Pin selection | pin 4 |
| Pin configuration | output enable |
| Pin polarity | active high |
| Input source | from pin |
| Start bit | disabled, transmitter loads data on enable |
| Stop bit | disabled |
| Buffer used | bit byte swapped register |

Configurations for Shifter 3

Shifter 3 is used by the SPI slave receiver on pin FlexIO_D5 as SPI_SIN. It has the following initial configurations.

Table 6. Configurations for Shifter 3

| Items | Configurations |
|---|---|
| Shifter mode | receive |
| Timer selection | timer 2 |
| Timer polarity | on positive of shift clock |
| Pin selection | Pin 5 |
| Pin configuration | output disabled |
| Pin polarity | active high |
| Input source | from pin |
| Start bit | disabled, transmitter loads data on enable |
| Stop bit | disabled |
| Buffer used | bit byte swapped register |

Configurations for Timer 2

Timer 2 is used by the SPI slave to acquire SPI_SCK on pin FlexIO_D6 from master to load/store/shift control of the two shifters. In slave mode, the SPI_SCK and SPI_CS signal are configured as inputs and driven by the SPI bus master. The transmit data is transferred at every SPI_SCK clock edge of each frame to the shift register when the SPI_CS signal is asserted. As a result, select pin FlexIO_D7 of SPI_CS as the trigger input to Timer 2. It has the following initial configurations.

**Table 7. Configurations for Timer 2**

| Items | Configurations |
|---|---|
| Timer mode | single 16-bit counter mode |
| Trigger selection | trigger from FlexIO pin 7 of SPI_CS |
| Trigger polarity | active low |
| Trigger source | internal trigger |
| Pin selection | pin 6 |
| Pin configuration | output disable |
| Pin polarity | active high |
| Timer initial output | output logic 0 when enabled, not affected by reset |
| Timer decrement source | decrement on pin input, Shift clock equals pin input |
| Timer enable condition | on trigger rising edge |
| Timer disable condition | timer is never disabled |
| Timer reset condition | timer is never reset |
| Start bit | disabled |
| Stop bit | disabled |
| Timer compare value | (n*2-1) [2] |

[2] - n is the number of bytes in the transmission.

## 3.2 Software implementation overview

Several driver functions have been implemented in this application, these are based on the HAL (Hardware Abstraction Layer) of the Freescale KSDK (Kinetis Software Development Kit).

A software package is provided along with this application note. The package contains a pruned KSDK based on V1.3, which this application requires.

The demo software includes two source files: main_loopback.c and retarget.c. The source files provide the following functions: configure the FlexIO to emulate dual SPI, DMA configurations and data verification after transfer can be directly used by user in their own codes with minor changes. This demo is realized in DMA mode as it has better performance compared with polling mode and interrupt mode.

FlexIO SPI Initialize Function

The Initialize Function is used to configure the shifters and timers in the application's initialization phase. The prototypes are:

*FlexIO_SPI_Master_Init();*

*FlexIO_SPI_Slave_Init();*

These two functions are used to configure both the SPI master and the slave. They feature FlexIO clock source setting, external pin mux setting, and the shifter/timer configuration for the SPI master and the slave. In the two prototypes, KSDK FlexIO HAL drivers are called and there are some parameters defined as macros which can be modified during initialization for different usage.

DMA Configuration Function

This demo is to realize FlexIO emulate dual SPI loopback transfer by DMA mode. The following prototypes are used to configure the DMA and DMAMUX.

The prototypes are:

*DMA_Init();*

*ConfigDMAfor_SPI_MASTER_TX();*

*ConfigDMAfor_SPI_MASTER_RX();*

*ConfigDMAfor_SPI_SLAVE_TX();*

*ConfigDMAfor_SPI_SLAVE_RX();*

Transmit/Receive Function

This function is used to start transmit/receive one byte data in SPI master or slave mode. As the subsequent transfer is handled by DMA automatically, no CPU intervention is needed.

The prototypes are:

*FLEXIO_SPI_HAL_PutDataMSB(flexio_spi_dev_t *devPtr, uint32_t dat);*

*FLEXIO_SPI_HAL_GetDataMSB(flexio_spi_dev_t *devPtr);*

- *devPtr* - *structure of configuring the flexio spi device;*
- *dat – data to be transfter;*

Data Verification Function

This function is to use debug the console and to print and verify the data after loopback transfer between the SPI master and the slave. The prototypes are:

*Debug_LPUART_Init();*

*PrintArray(uint8_t *Array, uint32_t Length);*

- *Array – transmit/receive data array to be print on debug console;*
- *Length – the length of data array to be printed;*

*CompareArray(uint8_t *Array1, uint8_t *Array2, uint32_t Length)*

- *Array1*- *expected correct data array to be compared;*
- *Array2*- *actual transmit/receive data array;*
- *Length* - *the length of data array to be compared;*

## 3.3 Running the demos

This demo runs on MAPS-KS22F256 along with the MAPS-Dock platform. The FlexIO pins assignment for SPI master and slave are shown in the following table:

**Table 8. FlexIO pins assignment table for SPI master and slave**

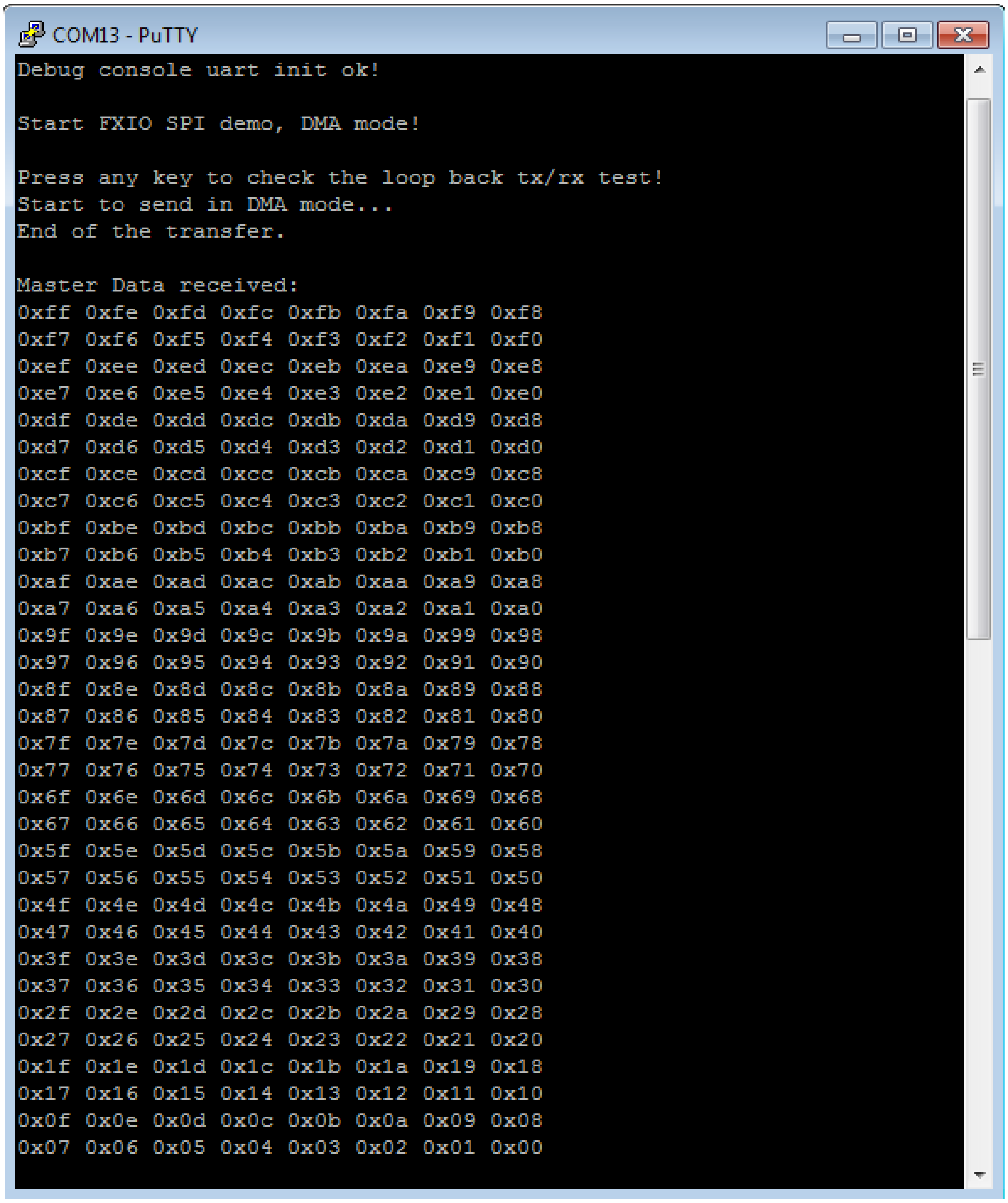
| **FlexIO SPI Master** | |
|---|---|
| FlexIO SPI master TX Pin:FlexIO_D0 | PTC12:CN9D pin84 |
| FlexIO SPI master RX Pin:FlexIO_D1 | PTC13:CN9D pin85 |
| FlexIO SPI master SCK Pin:FlexIO_D2 | PTC14:CN9D pin86 |
| FlexIO SPI master CS Pin:FlexIO_D3 | PTC15:CN9D pin87 |
| **FlexIO SPI Slave** | |
| FlexIO SPI slave TX Pin:FlexIO_D4 | PTC16:CN9D pin90 |
| FlexIO SPI slave RX Pin:FlexIO_D5 | PTC17:CN9D pin91 |
| FlexIO SPI slave SCK Pin:FlexIO_D6 | PTD0:CN9D pin93 |
| FlexIO SPI slave CS Pin:FlexIO_D7 | PTD1:CN9D pin94 |

You must make the connections between the master and slave by using 4 external wires before downloading the program image to the MCU via J-link or OpenSDA:

- SPI master TX <----> SPI slave RX
- SPI master RX <----> SPI Slave TX
- SPI master SCK <----> SPI slave SCK
- SPI master CS <----> SPI slave CS

After that is complete, follow the next steps to run the demo and check the result:

- Plug in the Micro USB to connect the PC and target the MAPS-KS22F256 board and the MAPS-Dock board
- Open the UART debug terminal on your PC with 8in1 and 115200bps settings
- Connect CN7/UART1 on MAPS-Dock board to the PC host
- Open the project by IAR workbench on your PC
- Rebuild all files and download the image into target on-chip flash
- Press any key to run the demo
- After the data finishes transferring the results are printed on the master terminal.



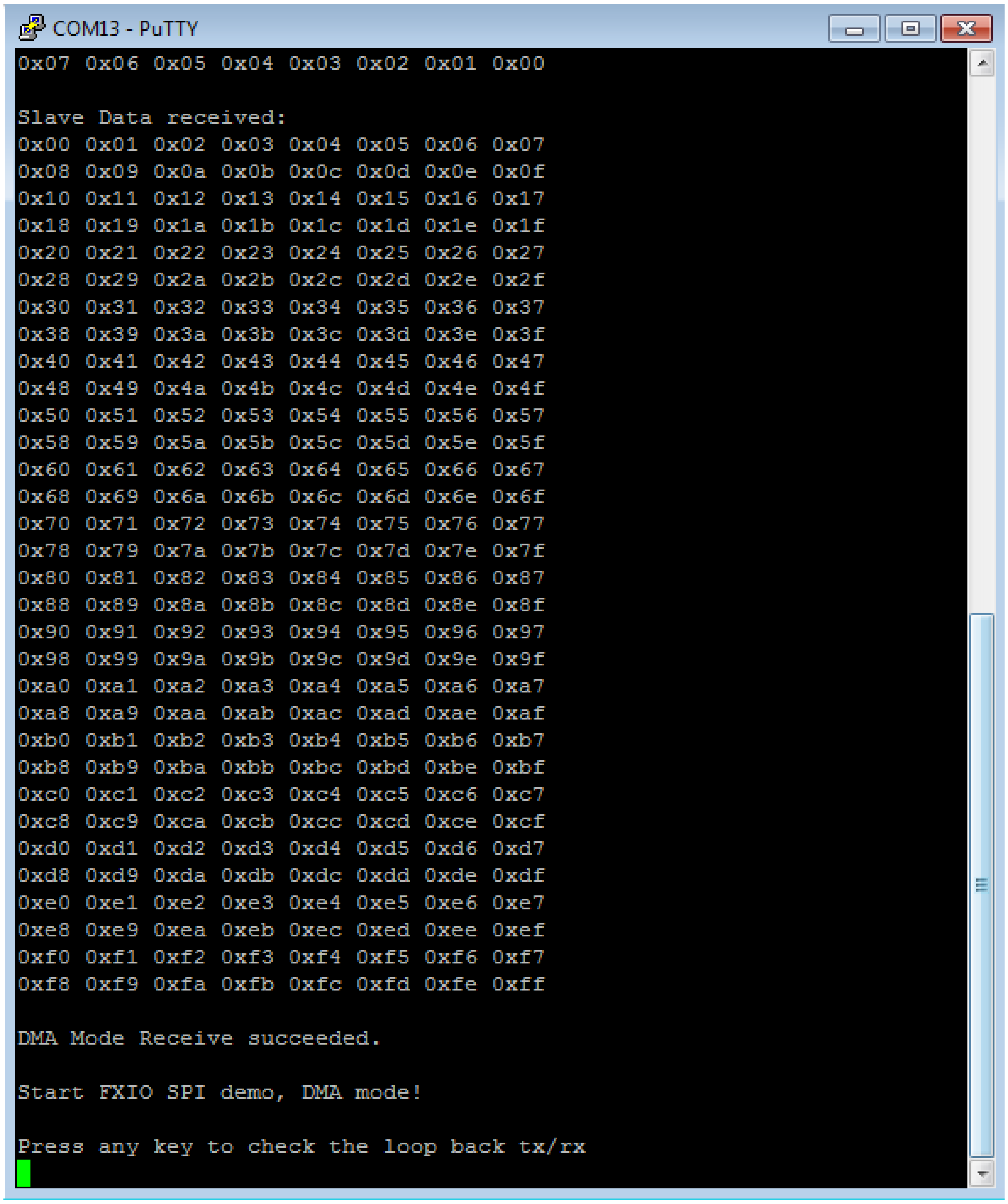
```
Debug console uart init ok!

Start FXIO SPI demo, DMA mode!

Press any key to check the loop back tx/rx test!
Start to send in DMA mode...
End of the transfer.

Master Data received:
0xff 0xfe 0xfd 0xfc 0xfb 0xfa 0xf9 0xf8
0xf7 0xf6 0xf5 0xf4 0xf3 0xf2 0xf1 0xf0
0xef 0xee 0xed 0xec 0xeb 0xea 0xe9 0xe8
0xe7 0xe6 0xe5 0xe4 0xe3 0xe2 0xe1 0xe0
0xdf 0xde 0xdd 0xdc 0xdb 0xda 0xd9 0xd8
0xd7 0xd6 0xd5 0xd4 0xd3 0xd2 0xd1 0xd0
0xcf 0xce 0xcd 0xcc 0xcb 0xca 0xc9 0xc8
0xc7 0xc6 0xc5 0xc4 0xc3 0xc2 0xc1 0xc0
0xbf 0xbe 0xbd 0xbc 0xbb 0xba 0xb9 0xb8
0xb7 0xb6 0xb5 0xb4 0xb3 0xb2 0xb1 0xb0
0xaf 0xae 0xad 0xac 0xab 0xaa 0xa9 0xa8
0xa7 0xa6 0xa5 0xa4 0xa3 0xa2 0xa1 0xa0
0x9f 0x9e 0x9d 0x9c 0x9b 0x9a 0x99 0x98
0x97 0x96 0x95 0x94 0x93 0x92 0x91 0x90
0x8f 0x8e 0x8d 0x8c 0x8b 0x8a 0x89 0x88
0x87 0x86 0x85 0x84 0x83 0x82 0x81 0x80
0x7f 0x7e 0x7d 0x7c 0x7b 0x7a 0x79 0x78
0x77 0x76 0x75 0x74 0x73 0x72 0x71 0x70
0x6f 0x6e 0x6d 0x6c 0x6b 0x6a 0x69 0x68
0x67 0x66 0x65 0x64 0x63 0x62 0x61 0x60
0x5f 0x5e 0x5d 0x5c 0x5b 0x5a 0x59 0x58
0x57 0x56 0x55 0x54 0x53 0x52 0x51 0x50
0x4f 0x4e 0x4d 0x4c 0x4b 0x4a 0x49 0x48
0x47 0x46 0x45 0x44 0x43 0x42 0x41 0x40
0x3f 0x3e 0x3d 0x3c 0x3b 0x3a 0x39 0x38
0x37 0x36 0x35 0x34 0x33 0x32 0x31 0x30
0x2f 0x2e 0x2d 0x2c 0x2b 0x2a 0x29 0x28
0x27 0x26 0x25 0x24 0x23 0x22 0x21 0x20
0x1f 0x1e 0x1d 0x1c 0x1b 0x1a 0x19 0x18
0x17 0x16 0x15 0x14 0x13 0x12 0x11 0x10
0x0f 0x0e 0x0d 0x0c 0x0b 0x0a 0x09 0x08
0x07 0x06 0x05 0x04 0x03 0x02 0x01 0x00
```

**Figure 6. Terminal Utility Output**



```
0x07 0x06 0x05 0x04 0x03 0x02 0x01 0x00

Slave Data received:
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17
0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f
0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27
0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f
0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37
0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47
0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f
0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67
0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f
0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77
0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f
0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87
0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f
0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97
0x98 0x99 0x9a 0x9b 0x9c 0x9d 0x9e 0x9f
0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7
0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf
0xb0 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7
0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf
0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7
0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf
0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7
0xd8 0xd9 0xda 0xdb 0xdc 0xdd 0xde 0xdf
0xe0 0xe1 0xe2 0xe3 0xe4 0xe5 0xe6 0xe7
0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef
0xf0 0xf1 0xf2 0xf3 0xf4 0xf5 0xf6 0xf7
0xf8 0xf9 0xfa 0xfb 0xfc 0xfd 0xfe 0xff

DMA Mode Receive succeeded.

Start FXIO SPI demo, DMA mode!

Press any key to check the loop back tx/rx
```

**Figure 7. Terminal Utility Output (continued)**

The software aims to ensure that SPI master transmits 256 bytes of 0x0~0xFF to slave. Simultaneously the slave transmits inverted 256 bytes of 0xFF~0x0 to the master. You can check that the data transfer result is correct using the debug terminal.

## 4 Conclusion

FlexIO is a new peripheral on some of the Kinetis microcontrollers. Due to the high flexibility of the shifters and timers, FlexIO has the capability to emulate a wide range of protocols.

This application note describes how to emulate dual SPI using FlexIO. The application is based on KSDK HAL. An SPI application can be easily implemented based on the driver functions in this application. Although this demo runs on MAPS-KS22F256, the user can port them to other parts of Kinetis chips with FlexIO.

## 5 Additional information

The FlexIO module can currently be found on the following Kinetis series MCUs: KL17, K27, KL33, and KL43. The FlexIO modules on these platforms have the same hardware resources and capabilities.

The next version of FlexIO is in development and will be launched with new Kinetis parts. The version has more abundant hardware resources and capabilities that are more powerful. Some of the key features are listed below.

- Up to 8 Shifters
- Up to 8 Timers
- Up to 32 Pins
- Parallel data transmission is supported, which enables the emulations of camera interface, Motorola 68K and Intel 8080 bus
- Programmable logic blocks allowing external digital logic to be integrated on-chip
- Programmable state machine for offloading basic system control functions from CPU

## 6 References

1. MAPS-KS22F256 and MAPS-Dock: Freescale MAPS Platform for Kinetis MCUs

2. KINETIS_SDK: Software Development Kit for Kinetis MCUs

3. Kinetis KS22 introduction

4. KS22 Reference Manual (doc. KS22P100M120SF0RM)

## 7 Revision history

Table 9. Revision history

| Revision number | Date | Substantive changes |
|---|---|---|
| 0 | 01/2016 | Initial release |

Document Number: AN5242
Rev. 0
01/2016