NXP Semiconductors
Application Note

Document Number: AN12174
Rev. 0, 06/2018

Using FlexIO to emulate communications and timing peripherals

1. Introduction

The FlexIO is a new on-chip peripheral available on Kinetis and S32K microcontroller families. It is highly configurable and capable of emulating a wide range of communication protocols, such as UART, I2C, SPI, I2S and LIN that are showed in this document and others more like J1850, I3C, Manchester.

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

These compilations of examples create as simple software demo based on S32K SDK (Software Development Kit version that is included into S32DS_v2018) and Bare Metal quick examples to give a better approach about what is FlexIO, with these examples the user can emulate different communication modules and pwm signals.

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 S32K1xx family.

Table 1. **Resources used for communication protocols**

Use Case	Supported using	FlexIO use	Comments
UART Application	Two independent parts **Transmit & Receive:** **Transmit**: one Timer, one Shifter and one Pin **Receive:** one Timer, one Shifter and one Pin	50%	Allows polling and interrupt mode Configurable bit order (bit swapped buff MSB first) Multiple transfers can be supported using DMA controller Does not support automatic insertion of parity bits
SPI Master	Two timers, Two shifters, Four pins	50%	CPHA=0 and CPHA=1 supported
SPI Slave	One timer, Two shifters, Four pins	33%	CPHA=0 and CPHA=1 supported
I2C Master	Two timers, Two shifters, Two pins	50%	FlexIO inserts a stop bit after every word to generate/verify the ACK/NACK
I2S Master	Two timers, Two shifters, Four pins	50%	Data transfers can be supported using the DMA

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

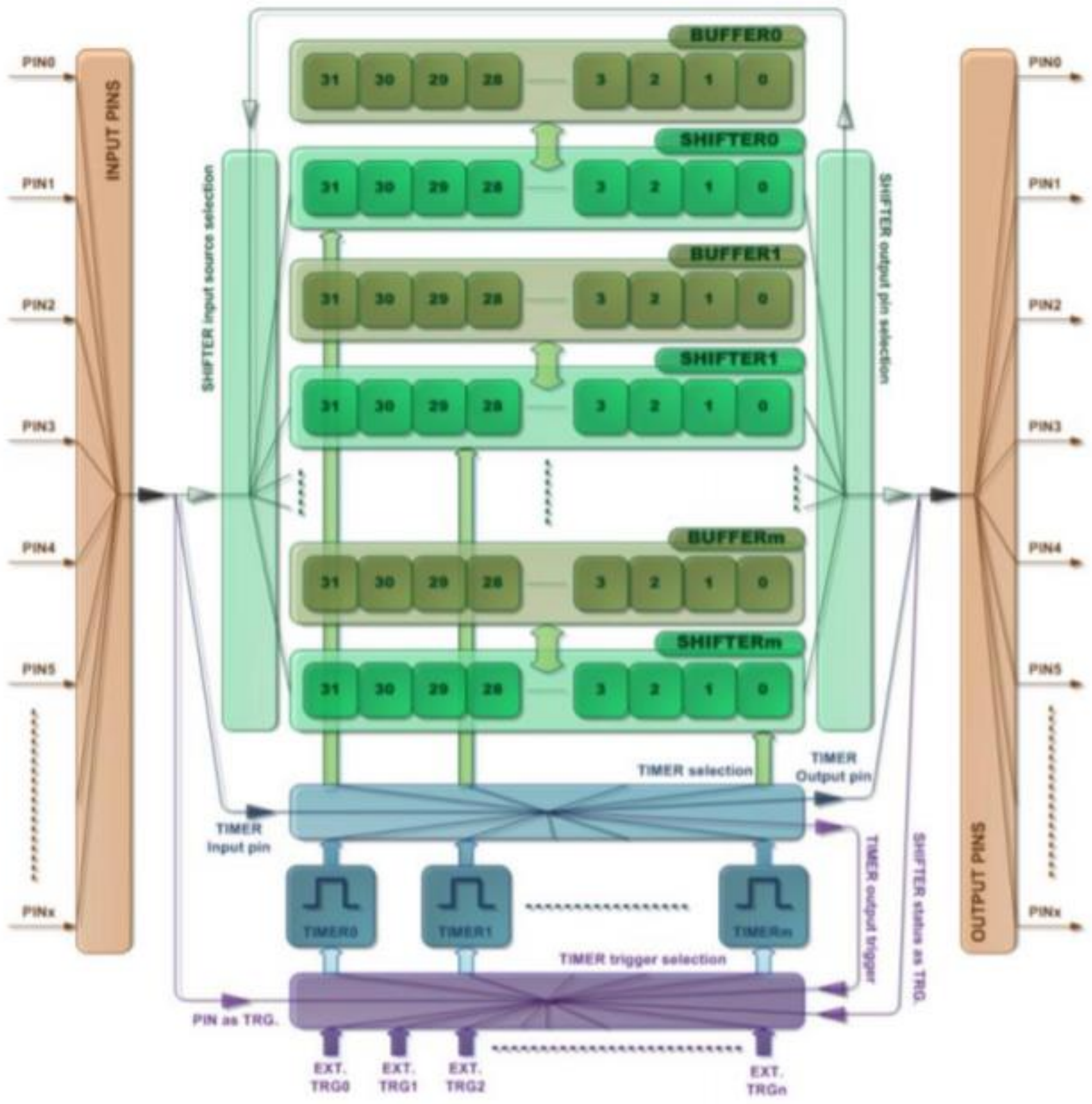


Figure 1. **FlexIO block diagram**

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 per your requirement, including dual 8-bit counters baud/bit mode, dual 8-bit counters PWM mode, and single 16-bit counter mode.

3. Emulating UART Using FlexIO

3.1. Instructions

For detailed explanation of the FlexIO refer to microcontroller reference manual. This example creates a simple software demo based on S32K SDK and including bare metal configuration example drivers for user to use FlexIO to emulate the UART.

This section describes how to emulate UART by using FlexIO. For this application, the evaluation development board S32K144EVB-Q100, in Figure 2 has been used.

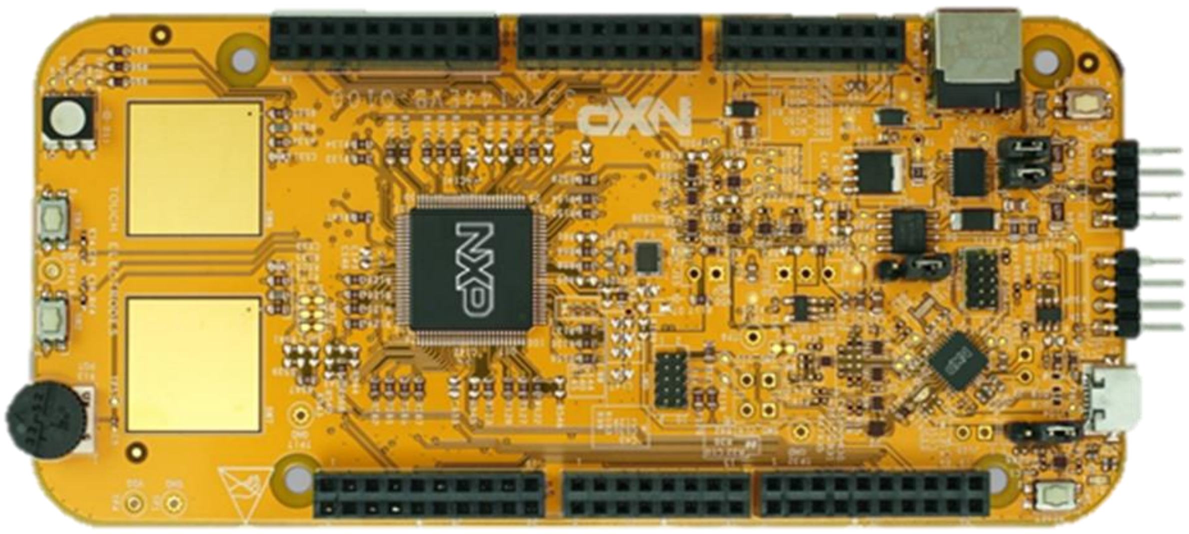


Figure 2. **Development board S32K144EVB-Q100**

In this application, FlexIO D0 pin is configured as UART TXD, FlexIO D1 pin is configured as UART RXD. Make the connections between the TXD and RXD using 1 external wire. You can use a general serial terminal/console to verify the result of data transfer.

The following diagram shows the hardware platform and data flows:

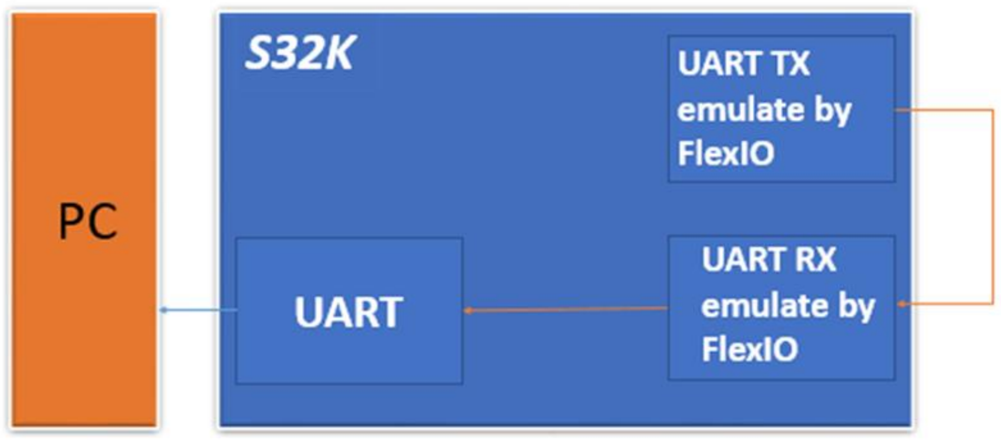


Figure 3. **Hardware Platform and Data Flows of this Application**

UART transmit uses the following resources:

- 1 timer — configured as an 8-bit counter mode to control data shift.
- 1 pin — controlled by timer to output data from SHIFTBUF.
- 1 shift — controlled by timer to shift data from SHIFTBUF and configured start bit and stop bit.

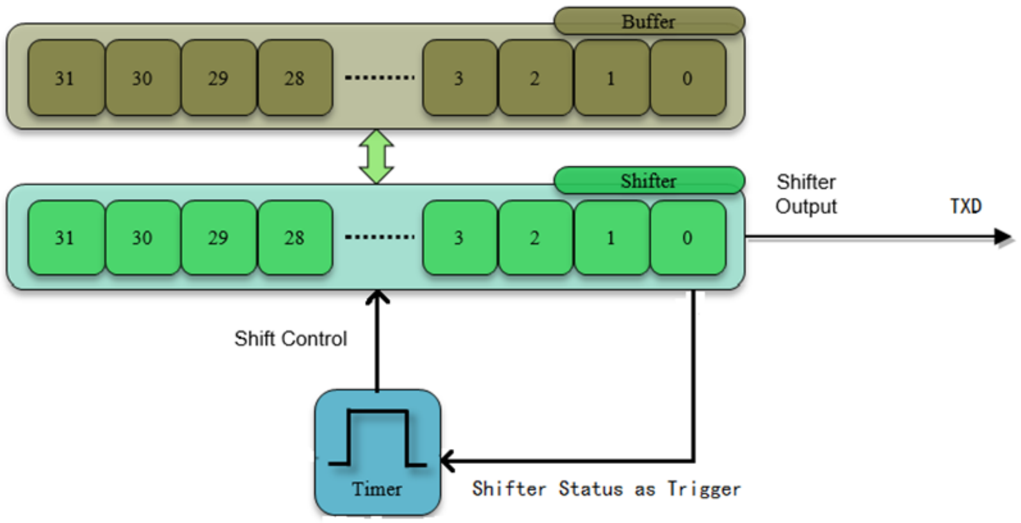


Figure 4. **Resource Assignment of FlexIO to Emulate UART Transmit**

UART receive uses the following resources:

- 1 timer — configured as an 8-bit counter mode to control data shift.
- 1 pin — controlled by timer to input data into SHIFTBUF.

- 1 shift — controlled by timer to shift data into SHIFTBUF and configured if the input data has start bit and stop bit.

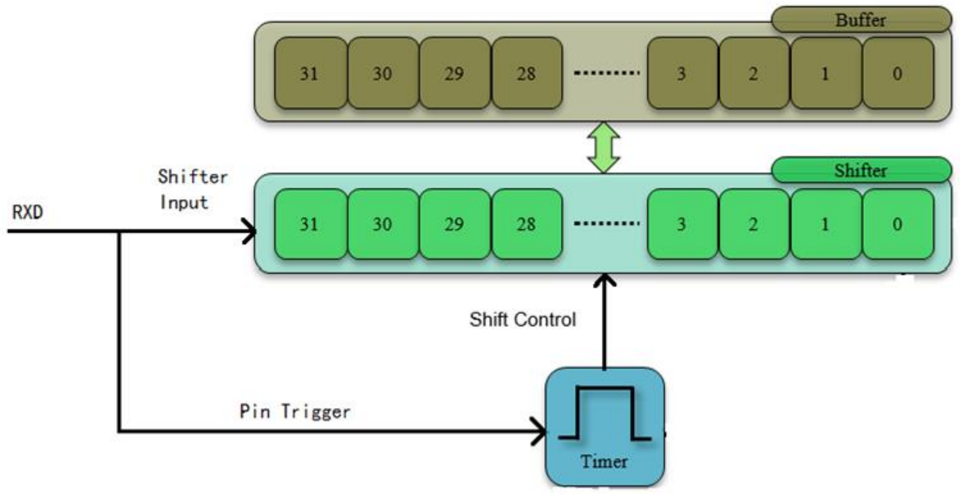


Figure 5. **Resource assignment of FlexIO to emulate UART receive**

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

3.2. Configuring 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 UART baud rate= 115200, and UART bit count= 8-bit, one start bit, one stop bit, no parity bit. Some of these settings must be changed by software to support the different UART features. To understand these configurations, refer to the following sections and the S32K reference manual.

3.2.1. UART transmit configurations

Configurations for shifter 0

Shifter 0 is used as the UART on pin FlexIO_D0 as TXD. It has the following initial configurations.

Table 2. **Configurations for shifters 0/1**

Items	**Configurations**	
Shifter mode	transmit	
Timer selection	timer 0	
Timer polarity	on posedge of shift clock	
Pin selection	pin 0	
Pin configuration	pin output	
Pin polarity	active high	

Input source	from pin	
Start bit	start bit value 0	
Stop bit	Stop bit value 1	
Buffer used	SHIFTBUF[7:0] to initiate an 8-bit transfer	

Configurations for timer 0

Timer 0 is used by the UART to shift control of the shifter0. The shifter status flag is set SHIFTBUF has been loaded with data from Shifter and is cleared each time the SHIFTBUF register is 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 configuration	output disabled
Timer initial output	output logic 1 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.

3.2.2. UART receive configurations

Configurations for shifter 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.

Shifter 1 is used as the UART on pin FlexIO_D1 as RXD. It has the following initial configurations.

Table 4. **Configurations for shifter 1**

Items	**configurations**
Shifter mode	receive
Timer selection	timer 1
Timer polarity	on negedge of shift clock
Pin selection	pin 1
Pin configuration	output disabled
Pin polarity	active high
Input source	from pin
Start bit	start bit value 0
Stop bit	start bit value 1
Buffer used	SHIFTBUF[31:24] to receive data

Configurations for timer 1

Timer 1 is used by the UART to shift control of the shifter1. The pin1 rising edge is configured to be enable the timer 1. The shifter begins to shift in 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 1 has the following initial configurations.

Table 5. **Configurations for timer 1**

Items	**Configurations**
Timer mode	dual 8-bit counters baud/bit mode.
Trigger selection	trigger from pin1
Trigger polarity	active high
Trigger source	external trigger
Pin configuration	output disabled
Timer initial output	output logic 1 when enabled, not affect by reset
Timer decrement source	decrement on FlexIO clock, shift on timer output
Timer enable condition	on pin rising edge
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)) [2]

2) 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.

Software implementation overview

This section describes the software implementation focused on SDK (Software Development Kit version that is included into S32DS_v2018), but please be aware that baremetal examples codes (following the same configuration showed on tables on this docs) are delivered within the same SW package of the Application note, all of functions can be directly used by user in their own codes with minor changes.

Features

- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transmit and receive functions
- Configurable baud rate and number of bits
- Single stop bit only
- Parity bit not supported

Initialization

Before using any FlexIO driver the device must first be initialized using function **FLEXIO_DRV_InitDevice**. Then the FLEXIO_UART Driver must be initialized, using function **FLEXIO_UART_DRV_Init().** It is possible to use more driver instances on the same FlexIO device, if sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using **FLEXIO_UART_DRV_Deinit().** This will release the hardware resources, allowing other driver instances to be initialized.

3.3. Functions description

FLEXIO_DRV_InitDevice function enables the clock gating of the FlexIO IP module and selects the proper peripheral clock source for FlexIO. The FLEXIO_CLK defined in the source file is exactly the frequency of the peripheral clock source. It resets the FlexIO IP module by SW and re-enables it. This is a general FlexIO IP module initialization function, called before using its shifters and timers.

FLEXIO_UART_DRV_Init function configures the timer0 as a dual 8-bit counters baud/bit mode to shift data out with pin0 output to emulate UART TXD and configures the timer1 as a dual 8-bit counters baud/bit mode to shift data in with pin1 input to emulate UART RXD. The ‘baud’ parameter means the baud rate of **UART. The ‘bits’ parameter means the bit number of one UART frame.**

3.3.1. FLEXIO_UART_DRV_SendData

This function is used to send data via FlexIO UART.

3.3.2. FLEXIO_UART_DRV_ReceiveDataBlocking

This is used to receive data via FlexIO UART using block mode.

To use the FlexIO to emulate UART to send and receive data, you can call these functions in sequence like in the `main()` function. It configures the baud rate to 115200 bps, and PTD0 as TXD, PTD1 as RXD.

3.4. Running the demos

This demo runs on S32K144EVB-Q100. The FlexIO pins assignment for TXD and RXD are shown in the following table:

FlexIO UART TXD	
FlexIO UART TXD Pin:FlexIO_D0	PTD0
FlexIO UART RXD	
FlexIO UART RXD Pin:FlexIO_D1	PTD1

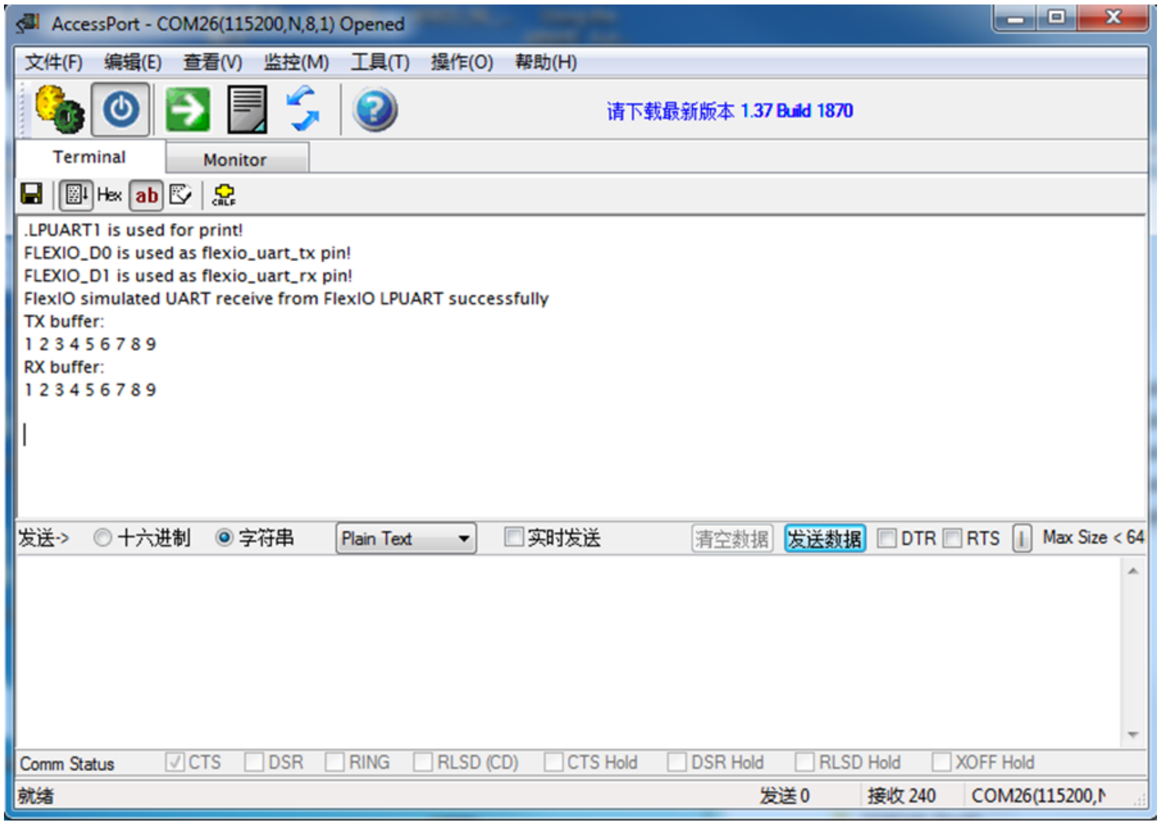
Table 6. **FlexIO pins assignment table for UART TXD and RXD**

You must make the connections between the TXD and RXD by using 1 external wire before downloading the program image to the MCU via J-link or OpenSDA:

- UART TXD <----> UART RXD.

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 S32K144EVB-Q100 board
- Open the UART debug terminal on your PC with 8in1 and 115200bps settings
- Open the project by S32DS workspace on your PC
- Rebuild all files and download the image into target
- After the data finishes transferring the results are printed on the master terminal



```
.LPUART1 is used for print!
FLEXIO_D0 is used as flexio_uart_tx pin!
FLEXIO_D1 is used as flexio_uart_rx pin!
FlexIO simulated UART receive from FlexIO LPUART successfully
TX buffer:
1 2 3 4 5 6 7 8 9
RX buffer:
1 2 3 4 5 6 7 8 9
```

AccessPort - COM26(115200,N,8,1) Opened
文件(F) 编辑(E) 查看(V) 监控(M) 工具(T) 操作(O) 帮助(H)
请下载最新版本 1.37 Build 1870
Terminal
Monitor
Hex
发送-> 十六进制 字符串
Plain Text
实时发送
清空数据 发送数据
DTR RTS Max Size < 64
Comm Status CTS DSR RING RLSD (CD) CTS Hold DSR Hold RLSD Hold XOFF Hold
就绪 发送 0 接收 240 COM26(115200,N

4. Emulating Dual SPI by using FlexIO

This section describes how to emulate dual SPI by using FlexIO. For this application, the S32K144EVB-Q100, which is shown in the figure 2, has been used.

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 general UART debug console to check the result of data loopback transfer. The following diagram shows the hardware platform and data flows:

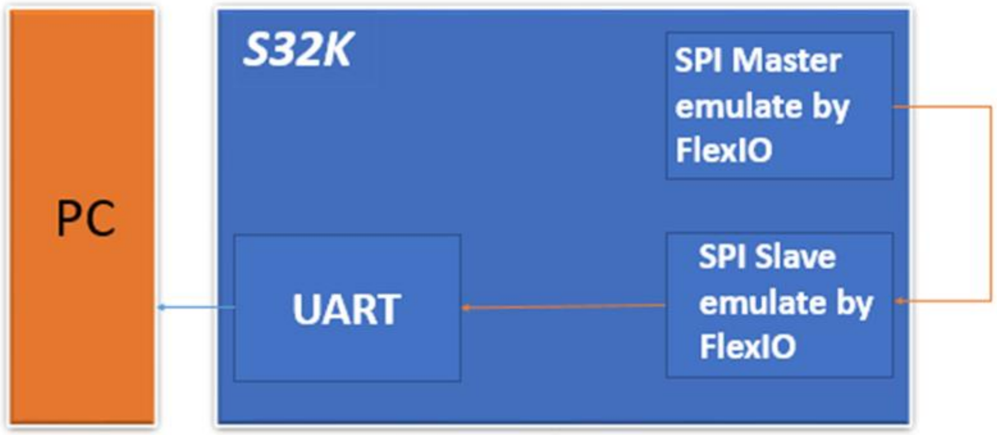


Figure 6. **Hardware platform and data flows**

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.

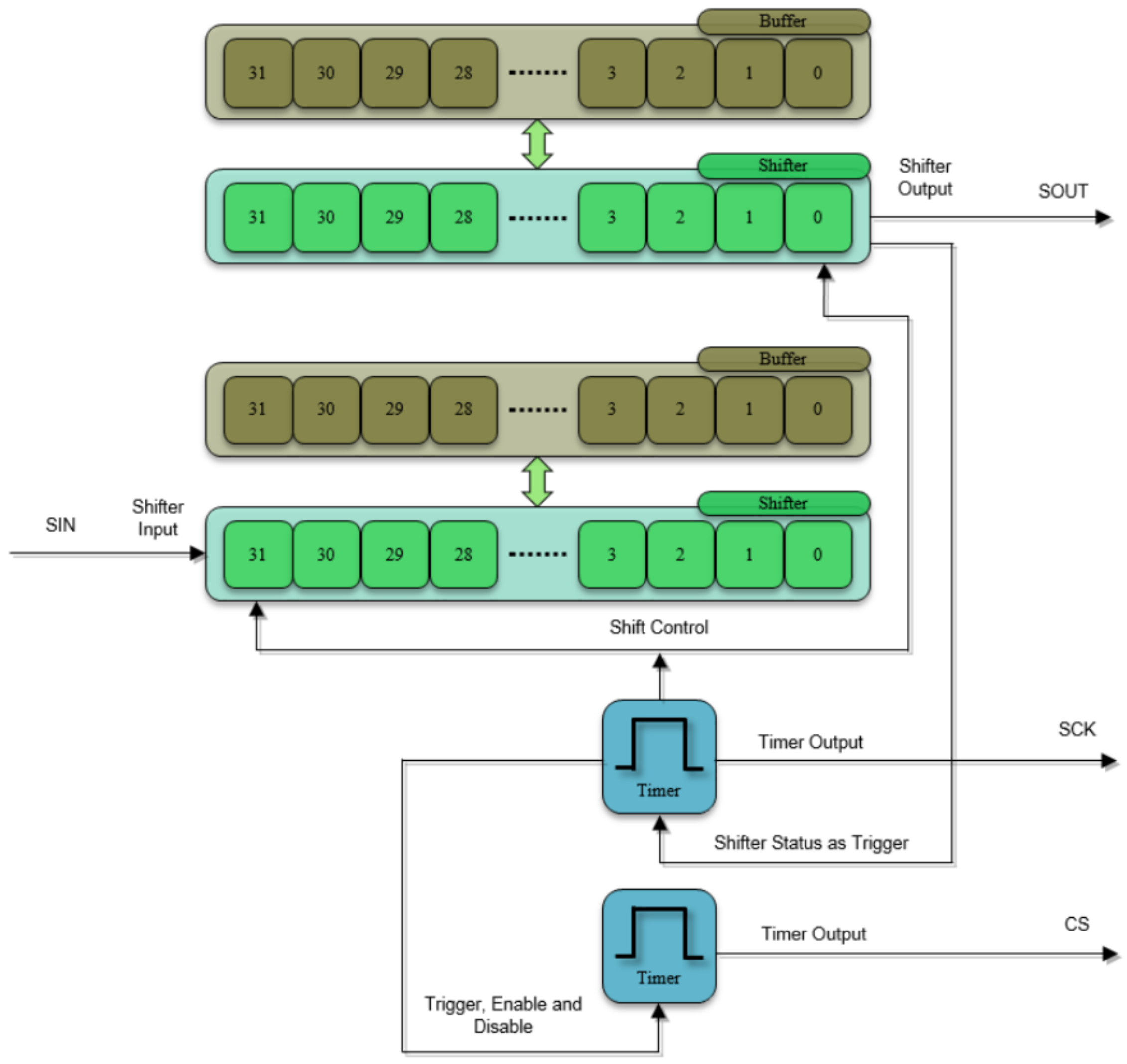


Figure 7. **Resource Assignment of FlexIO to Emulate SPI Master**

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.

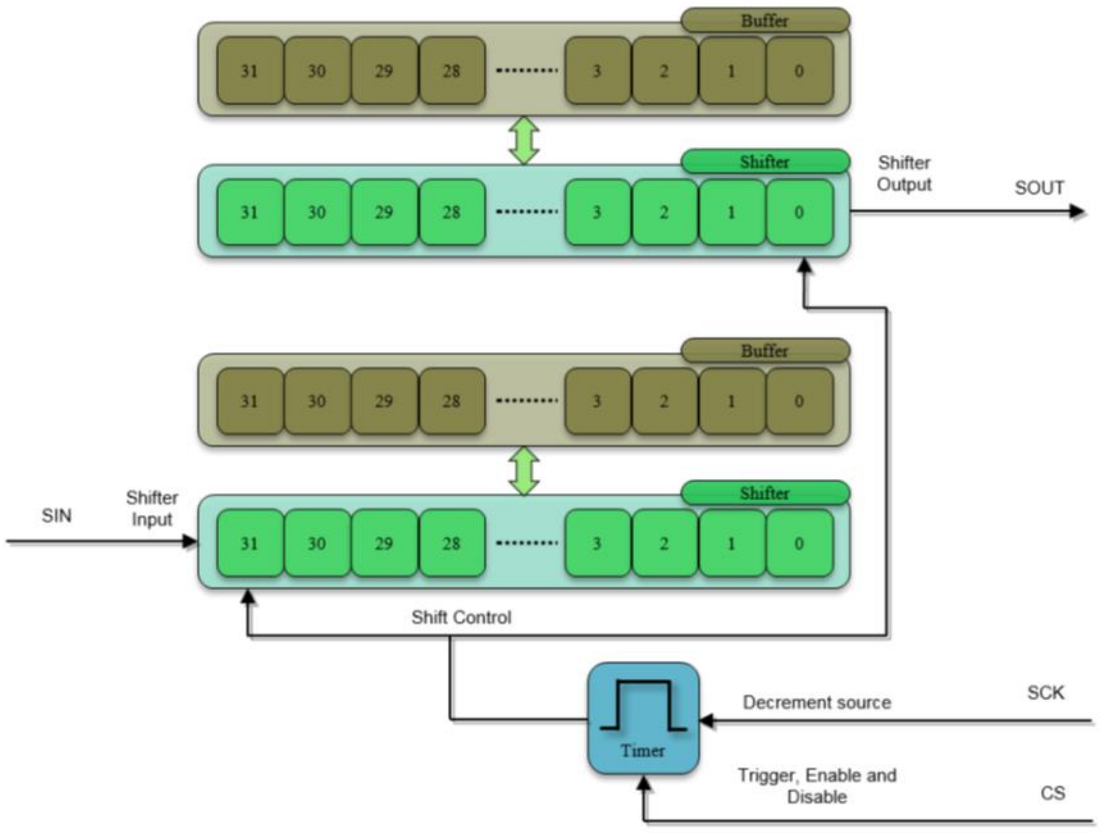


Figure 8. **Resource Assignment of FlexIO to Emulate SPI slave**

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

4.1. 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 S32K reference manual.

4.1.1. 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 7. **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 8. **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 9. **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

Items	Configurations
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))[3]

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 10. **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	Timer never reset
Start bit	disabled
Stop bit	disabled
Timer compare value	0xFFFF

4.1.2. SPI slave Configurations

Configurations for Shifter 2

[3] 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.

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

Table 11. **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 12. **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 13. **Configurations for Timer 2**

Items	**Configurations**
Timer mode	single 16-bit counter mode
Trigger selection	trigger from FlexIO pin 7 of SPI_CS

Items	Configurations
Trigger polarity	active low
Trigger source	internal trigger
Pin selection	pin 6
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 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 never reset
Start bit	disabled
Stop bit	disabled
Timer compare value	(n*2-1)[4]

4.2. Software overview

The FLEXIO_SPI Driver allows communication on an SPI bus using the FlexIO module in the S32144K processor.

Features:

- Master or slave operation
- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transfer functions
- Configurable baud rate
- Configurable clock polarity and phase
- Configurable bit order and data size
- Functionality

Initialization

Before using any FlexIO driver the device must first be initialized using function FLEXIO_DRV_InitDevice. Then the FLEXIO_SPI Driver must be initialized, using functions FLEXIO_SPI_DRV_MasterInit() or FLEXIO_SPI_DRV_SlaveInit(). It is possible to use more driver instances on the same FlexIO device, if sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using FLEXIO_SPI_DRV_MasterDeinit() or

[4] n is the number of bytes in the transmission.

FLEXIO_SPI_DRV_SlaveDeinit(). This will release the hardware resources, allowing other driver instances to be initialized other.

Master Mode

Master Mode provides functions for transmitting or receiving data to/from an SPI slave. Baud rate is provided at initialization time through the master configuration structure, but can be changed at runtime by using FLEXIO_SPI_DRV_MasterSetBaudRate() function. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_SPI_DRV_MasterGetBaudRate() after FLEXIO_SPI_DRV_MasterSetBaudRate() to check what baud rate was set.

To send or receive data, use function FLEXIO_SPI_DRV_MasterTransfer(). The transmit and receive buffers, together with parameters for the transfer are provided through the flexio_spi_transfer_t structure. If only transmit or receive is desired, any one of the Rx/Tx buffers can be set to NULL. This driver does not support continuous send/receive using a user callback function. The callback function is only used to signal the end of a transfer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return STATUS_SUCCESS, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling FLEXIO_SPI_DRV_MasterGetStatus(). If the transfer is still ongoing this function will return STATUS_BUSY. If the transfer is completed, the function will return either STATUS_SUCCESS or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode, the function FLEXIO_SPI_DRV_MasterGetStatus() ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode, the DMA channels that will be used by the driver are received through the configuration structure. The channels must be initialized by the application before the flexio_spi driver is initialized. The flexio_spi driver will only set the DMA request source.

Slave Mode

Slave Mode is similar to master mode, the main difference being that the FLEXIO_SPI_DRV_SlaveInit() function initializes the FlexIO module to use the clock signal received from the master instead of generating it. Consequently, there is no SetBaudRate function in slave mode. Other than that, the slave mode offers a similar interface to the master mode.

FLEXIO_SPI_DRV_MasterTransfer() can be used to initiate transfers, and FLEXIO_SPI_DRV_SlaveGetStatus() is used to check the status of the transfer and advance the transfer in polling mode. All other specifications from the Master Mode description apply for Slave Mode too

4.3. Implementation

application have been implemented in this application, these are based on S32K SDK (Software Development Kit version that is included into S32DS_v2018),

The demo software includes two source files: main.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

Before using any FlexIO driver the device must first be initialized using function FLEXIO_DRV_InitDevice.

Then the FLEXIO_SPI Driver must be initialized, using functions FLEXIO_SPI_DRV_MasterInit() or FLEXIO_SPI_DRV_SlaveInit(). It is possible to use more driver instances on the same FlexIO device, if sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using FLEXIO_SPI_DRV_MasterDeinit() or FLEXIO_SPI_DRV_SlaveDeinit(). This will release the hardware resources, allowing other driver instances to be initialized other. 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

- **FLEXIO_SPI_DRV_MasterTransfer**

4.4. Running the demos

This demo runs on S32K144EVB-Q100. The FlexIO pins assignment for SPI master and slave are shown in the following table:

Table 14. **FlexIO pins assignment table for SPI master and slave**

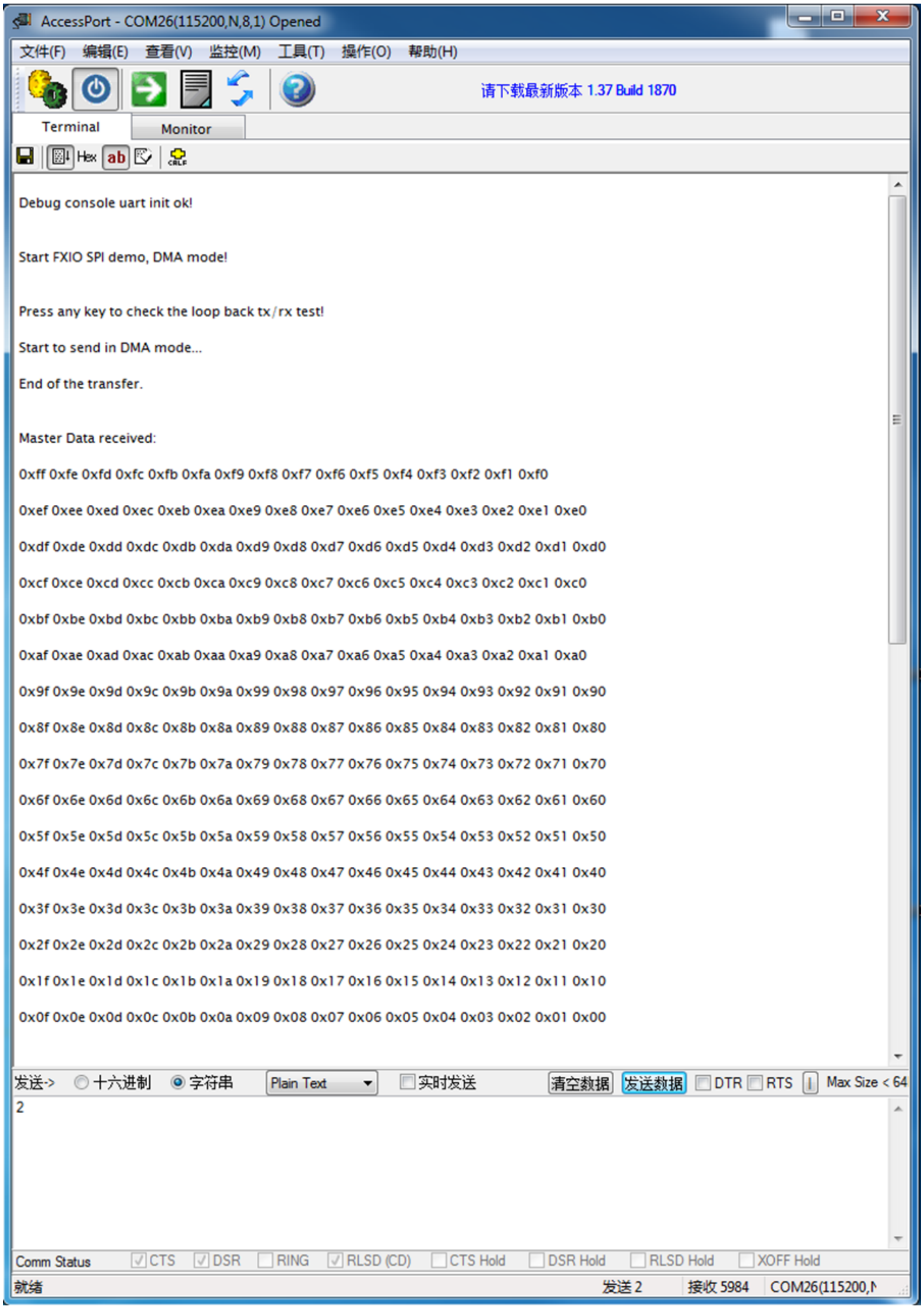
FlexIO SPI Master	
FlexIO SPI master TX Pin:FlexIO_D0	PTD0
FlexIO SPI master RX Pin:FlexIO_D1	PTD1
FlexIO SPI master SCK Pin:FlexIO_D2	PTE15
FlexIO SPI master CS Pin:FlexIO_D3	PTE16
FlexIO SPI Slave	
FlexIO SPI slave TX Pin:FlexIO_D4	PTE10
FlexIO SPI slave RX Pin:FlexIO_D5	PTE11
FlexIO SPI slave SCK Pin:FlexIO_D6	PTA8
FlexIO SPI slave CS Pin:FlexIO_D7	PTA9

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 S32K144EVB-Q100 board
- Open the UART debug terminal on your PC with 8in1 and 115200bps settings
- Open the project by S32DS workbench on your PC
- Rebuild all files and download the image into target
- Press any key to run the demo
- After the data finishes transferring the results are printed on the master terminal.


AccessPort - COM26(115200,N,8,1) Opened
文件(F) 编辑(E) 查看(V) 监控(M) 工具(T) 操作(O) 帮助(H)
请下载最新版本 1.37 Build 1870
Terminal
Monitor
Hex
ab

```
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
```

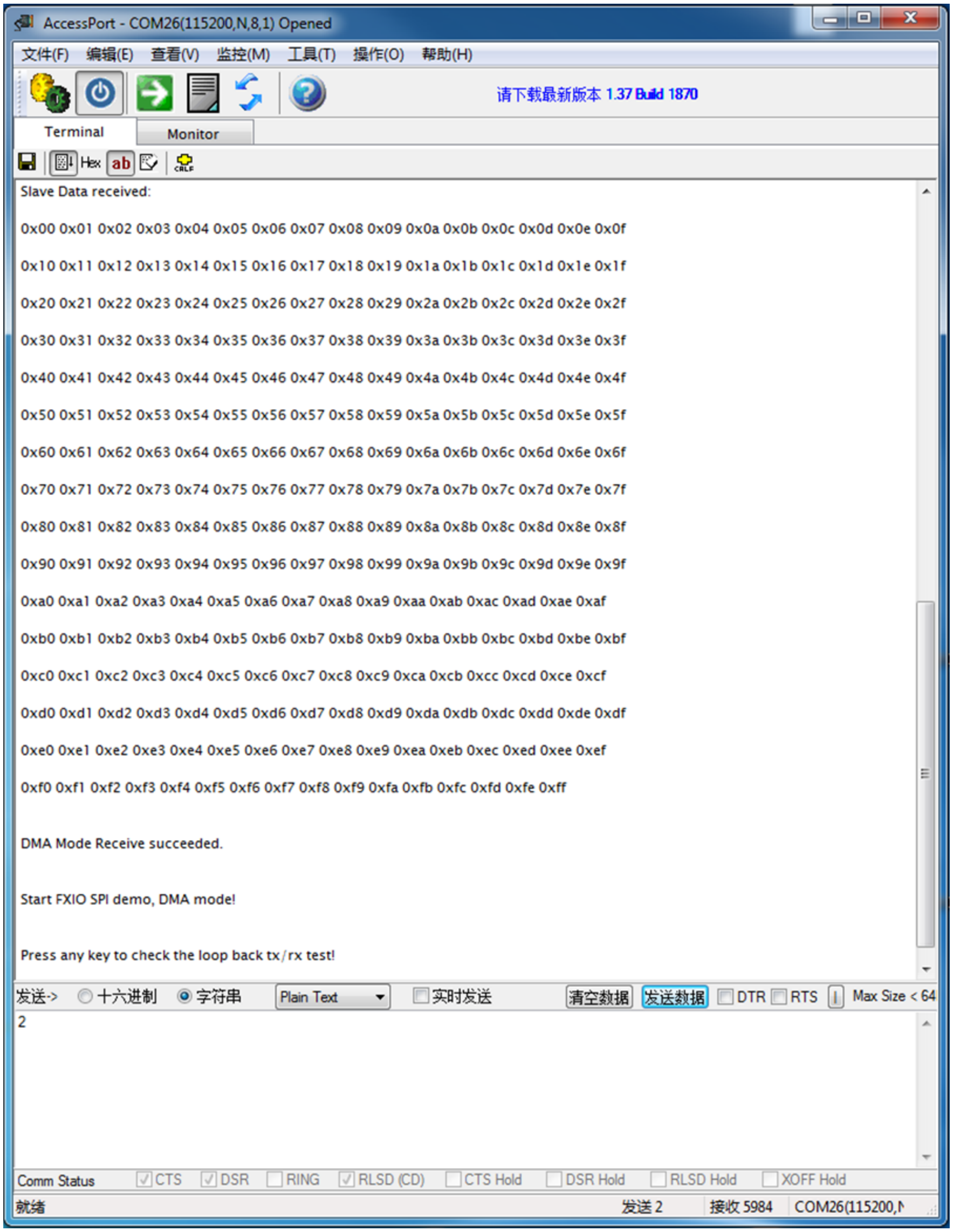
发送-> ○ 十六进制 [✓ 字符串] Plain Text ☐ 实时发送 清空数据 发送数据 ☐ DTR ☐ RTS Max Size < 64

```
2
```

Comm Status ☑ CTS ☑ DSR ☐ RING ☑ RLSD (CD) ☐ CTS Hold ☐ DSR Hold ☐ RLSD Hold ☐ XOFF Hold

就绪 发送 2 接收 5984 COM26(115200,N

Figure 9. **Terminal utility output**


AccessPort - COM26(115200,N,8,1) Opened
文件(F) 编辑(E) 查看(V) 监控(M) 工具(T) 操作(O) 帮助(H)
请下载最新版本 1.37 Build 1870
Terminal
Monitor
发送-> 十六进制 [✓ 字符串] Plain Text 实时发送 清空数据 发送数据 DTR RTS Max Size < 64
Comm Status CTS DSR RING RLSD (CD) CTS Hold DSR Hold RLSD Hold XOFF Hold
就绪 发送 2 接收 5984 COM26(115200,N

```
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 test!
```

```
2
```

Figure 10. **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.

5. Emulating I2C Bus Master by using FlexIO

5.1. Introduction

This section describes how to emulate I2C bus Master by using FlexIO. For this application, the development board S32K144EVB-Q100 and communicate with six-axis sensor FXOS8700CQ, in which a general I2C interface is integrated. The data read from FXOS8700CQ is sent to PC terminal through a UART port.

The following diagram shows the hardware platform and data flows. For more information on Open-SDA in the diagram, see the relevant materials of the development board.)

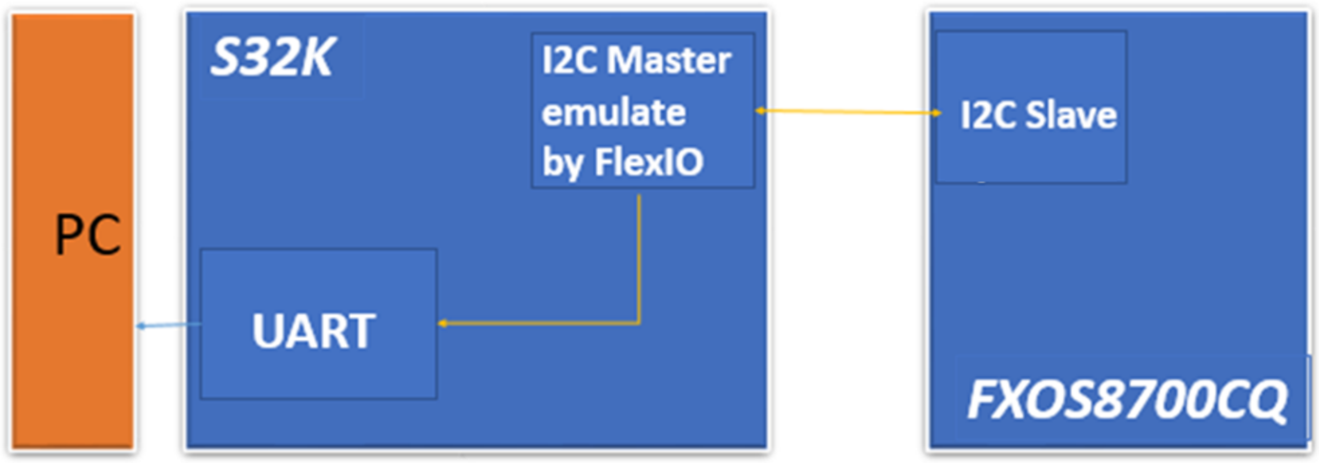


Figure 1. **S32K144EVB-Q100 connected with FXOS8700CQ**

5.2. General description

I2C bus master is emulated using:

- Two shifters — Respectively used as a transmitter and a receiver.
- Two timers —One is used for the SCL output generation, and the other is used for the load/store/shift control of the two shifters.
- Two pins — Respectively used as SDA and SCL.

Following figure shows the resource assignment:

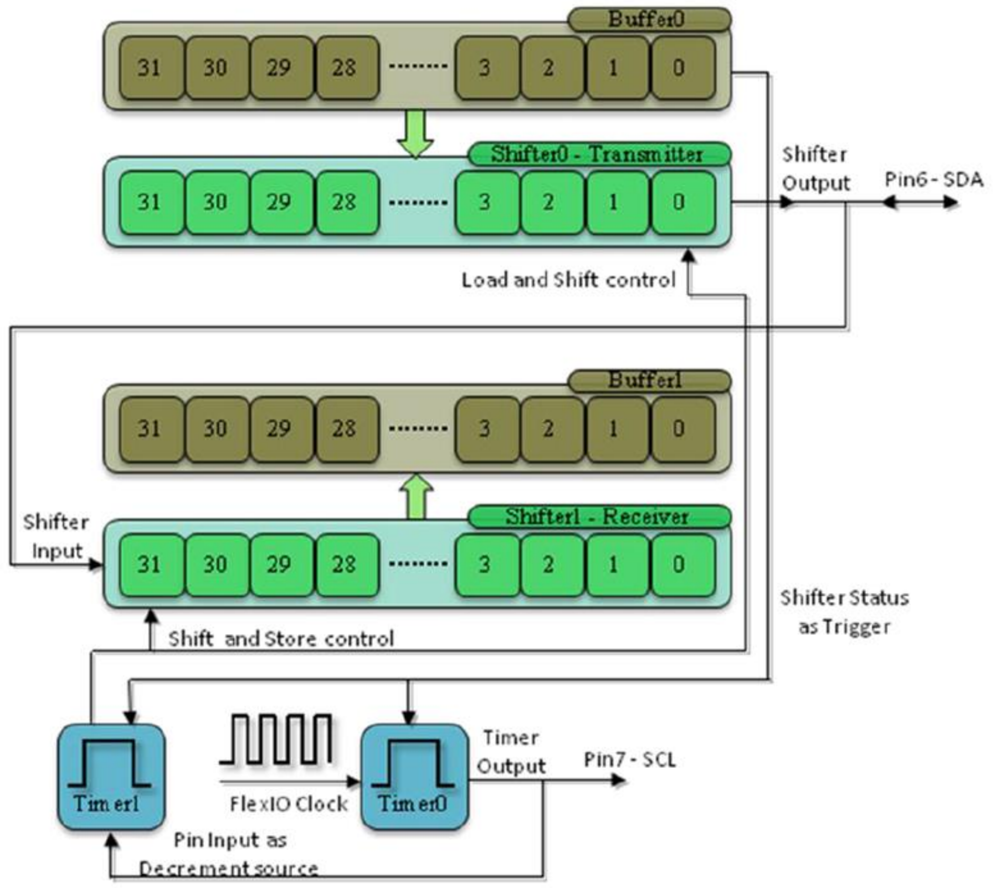


Figure 2. **Resource Assignment of FlexIO to Emulate I2C Master**

5.3. Configurations of the shifters and timers

This section provides detailed configurations of the shifters and timers.

To understand these configurations, see the following descriptions and the reference manual.

5.3.1. Configurations for Shifter 0

Shifter 0 is used as the transmitter. It has the following initial configurations.

Table 15. **Initial configuration of Shifter 0**

Items	**Configurations**
Shifter mode	transmit
Timer selection	timer 1
Timer polarity	on posedge of shift clock
Pin selection	pin 0
Pin configuration	open drain or bidirectional output enable
Pin polarity	active low

Items	Configurations
Input source	from pin
Start bit	Value 0
Stop bit	Value 1
Buffer used	bit byte swapped register

5.3.2. Configurations for Shifter 1

Shifter 1 is used as the receiver. It has the following initial configurations.

Table 16. **Initial configuration of Shifter 1**

Items	Configurations
Shifter mode	Receive
Timer selection	timer 1
Timer polarity	on negedge of shift clock
Pin selection	pin 0
Pin configuration	Output disabled
Pin polarity	active high
Input source	from pin
Start bit	Disable
Stop bit	Value 0
Buffer used	bit byte swapped register

5.3.3. Configurations for Timer 0

Timer 0 is used to generate SCL output and to trigger timer 1. It has the following initial configurations.

Table 17. **Initial configuration of 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 1
Pin configuration	Open drain or bidirectional 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	On timer pin equals to timer output

Items	**Configurations**
Start bit	enabled
Stop bit	enabled on timer disable
Timer compare value	((n*9+1)*2-1<<8) | (baudrate_divider))[5]

5.3.4. Configurations for Timer 1

Timer 1 is used to control the Shifter 0 and Shifter 1. It has the following initial configurations.

Table 18. **Initial configuration of Timer 1**

Items	**Configurations**
Timer mode	Single 16-bit counter mode
Trigger selection	shifter 0 status flag
Trigger polarity	active low
Trigger source	internal trigger
Pin selection	pin 1
Pin configuration	Output disabled
Pin polarity	active low
Timer initial output	output logic 1 when enabled, not affect by reset
Timer decrement source	Decrement on pin input, shift clock equal to pin input
Timer enable condition	On timer 0 enable
Timer disable condition	On timer 0 disable
Timer reset condition	Never reset
Start bit	enabled
Stop bit	enabled on timer compare
Timer compare value	0x0F

5.4. Software overview

The FLEXIO_I2C Driver allows communication on an I2C bus using the FlexIO module in the S32144K processor.

Features

- Master operation only
- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transmit and receive functions
- 7-bit addressing

[5] 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.

- Clock stretching
- Configurable baud rate
- Functionality

Initialization

Before using any FlexIO driver the device must first be initialized using function FLEXIO_DRV_InitDevice. Then the FLEXIO_I2C Driver must be initialized using functions FLEXIO_I2C_DRV_MasterInit(). It is possible to use more driver instances on the same FlexIO device, if sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using FLEXIO_I2C_DRV_MasterDeinit(). This will release the hardware resources, allowing other driver instances to be initialized.

Master Mode

Master Mode provides functions for transmitting or receiving data to/from any I2C slave. Slave address and baud rate are provided at initialization time through the master configuration structure, but they can be changed at runtime by using FLEXIO_I2C_DRV_MasterSetBaudRate() or FLEXIO_I2C_DRV_MasterSetSlaveAddr(). Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_I2C_DRV_MasterGetBaudRate() after FLEXIO_I2C_DRV_MasterSetBaudRate() to check what baud rate was set.

To send or receive data to/from the currently configured slave address, use functions FLEXIO_I2C_DRV_MasterSendData() or FLEXIO_I2C_DRV_MasterReceiveData() (or their blocking counterparts). Parameter sendStop can be used to chain multiple transfers with repeated START condition between them, for example when sending a command and then immediately receiving a response. The application should ensure that any send or receive transfer with sendStop set to false is followed by another transfer. The last transfer from a chain should always have sendStop set to true. This driver does not support continuous send/receive using a user callback function. The callback function is only used to signal the end of a transfer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return STATUS_SUCCESS, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling FLEXIO_I2C_DRV_MasterGetStatus(). If the transfer is still ongoing this function will return STATUS_BUSY. If the transfer is completed, the function will return either STATUS_SUCCESS or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode, the function FLEXIO_I2C_DRV_MasterGetStatus() ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode, the DMA channels that will be used by the driver are received through the configuration structure. The channels must be initialized by the application before the flexio_i2c driver is initialized. The flexio_i2c driver will only set the DMA request source.

Before using the FLEXIO_I2C Driver the FlexIO clock must be configured. Refer to SCG HAL and PCC HAL for clock configuration.

The following tips are the key points of using FlexIO to emulate I2C bus master.

Dealing with the transmitter and receiver: From the above configurations, you can see that there are two shifters, sharing one timer (Timer 1) and SDA pin (Pin 0), respectively used as the transmitter and receiver. So, both shifters are serviced for each byte in the transmissions. The receiver is read after each byte transmission to clear the receive buffer and status flag. The transmitter must transmit 0xFF to tristate the output when receiving.

Timer trigger settings: The triggers of the timers are set as the transmitter’s status flag. Filling a byte into the transmitter’s buffer negates the status flag enables the timer 0 to start the decrement count. The decrement source of timer 1 is set as the SCL pin input. Each SCL edge makes timer 1 to decrease by 1. Two SCL edges make timer 1to go through one period, which results in the data in the shifters to shift by one bit.

TIMCMP (Timer Compare Register) settings: TIMCMP of a timer stores the timer’s module value. The high 8 bits of TIMCMP 0 need to be set to a value that equals to the amount of SCL edges (both edges) in a transmission. The number of the data bytes, ACK/NACK bit, and stop-condition bit should be taken into consideration when calculating the value. The low 8 bits of TIMCMP 0 are used to configure the baud rate. The value set to TIMCMP 1 is calculated based on the number bits in a single frame.

Operations for double-buffered shifters: The shifters are designed as double-buffered structure. To avoid underflow when transmitting or receiving multiple bytes in high baud rate, the transmit shifter and its buffer should be filled data by software in advance. The process can be divided into the following steps: (1) Filling the first byte into the transmit buffer; (2) Waiting for the first byte to be loaded into the transmit shifter by polling the status flag. Filling the second byte into the transmit buffer; (3) Waiting for the first byte is shifted out and the second byte is loaded into the transmit shifter. The first received byte is shifted into the receive shifter during this period. Waiting for the first received byte to be stored into the receive buffer. Reading the receive buffer. Filling the third byte into the transmit buffer. From the third step to the last, polling /interrupt/DMA mode can be used. In respect of the time delay, DMA is less than interrupt, and interrupt is less than polling.

ACK/ NACK bit generation and check: An ACK or NACK bit is needed after each transmission in I2C bus protocol. The SSTOP bit of the transmitter can be used to generate ACK/NACK. Set SSTOP bit to 0 to generate ACK, and set it to 1 to generate NACK. The SSTOP bit of the receiver can be used to check ACK/NACK. To shift the ACK/NACK bit out/in, timer 0 needs to output two additional edges. When timer 1 has decreased to 0, the following SCL configured edge will shift the ACK/NACK bit out/in.

Repeated START and STOP signal generation: As a (repeated) START signal is a falling edge and STOP signal is a rising edge of SDA during SCL in high level. Therefore, SDA needs to be set to high in advance (during SCL in low level) to generate repeated START signal, and set low to generate STOP signal. These can be implemented by loading one additional byte into transmitter on the last falling edge of SCL in each byte transmission. However, only the highest bit will be shifted out. This is also controlled by the timers. Therefore, 0xFF is loaded if generating a repeated START signal, and 0x00 is loaded if generating a STOP signal. The (repeated) START and STOP signal's level are respectively determined by the SSTART and SSTOP in the transmitter Shifter Configuration Register. Before the first data bit shifts out, the configured start bit is loaded into the transmit shifter and is then shifted out. When timer 0 decrement counts to 0, the configured stop bit will be load into transmit shifter and then shifted out. Another point, the shifter will immediately load the stop bit when the shifter is initially configured for transmit mode if a stop bit is enabled.

5.5. implementation

This section describes the software implementation. Several driver functions have been implemented in this application, which are based on the S32K SDK(Software Development Kit version that is included into S32DS_v2018).

The demo runs in polling mode. This section mainly describes several major driver functions.

5.5.1. Initialize Function

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

FlexIO_I2C_Init()

5.5.2. Transmit Function

This function is used to transmit one or more bytes to a given address of the slave device. The prototype is:

*status_t FLEXIO_I2C_DRV_MasterSendData (flexio_i2c_master_state_t * master, const uint8_t * txBuff, uint32_t txSize, bool sendStop)*

where:

master - Pointer to the FLEXIO_I2C master driver context structure.

txBuff - pointer to the data to be transferred

txSize - length in bytes of the data to be transferred

sendStop - specifies whether or not to generate stop condition after the transmission returns

Error or success status returned by API

5.5.3. Receive Function

This function is used to receive one or more bytes from a given address of the slave device. The prototype is:

*status_t FLEXIO_I2C_DRV_MasterReceiveData (flexio_i2c_master_state_t * master,*
*uint8_t * rxBuff,*
uint32_t rxSize,
bool sendStop
)

where:

master	Pointer to the FLEXIO_I2C master driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the reception
Returns	*Error or success status returned by API.*

5.6. Running the demos

This demo runs on S32K144EVB-Q100. The FlexIO pins assignment for I2C master are shown in the following table:

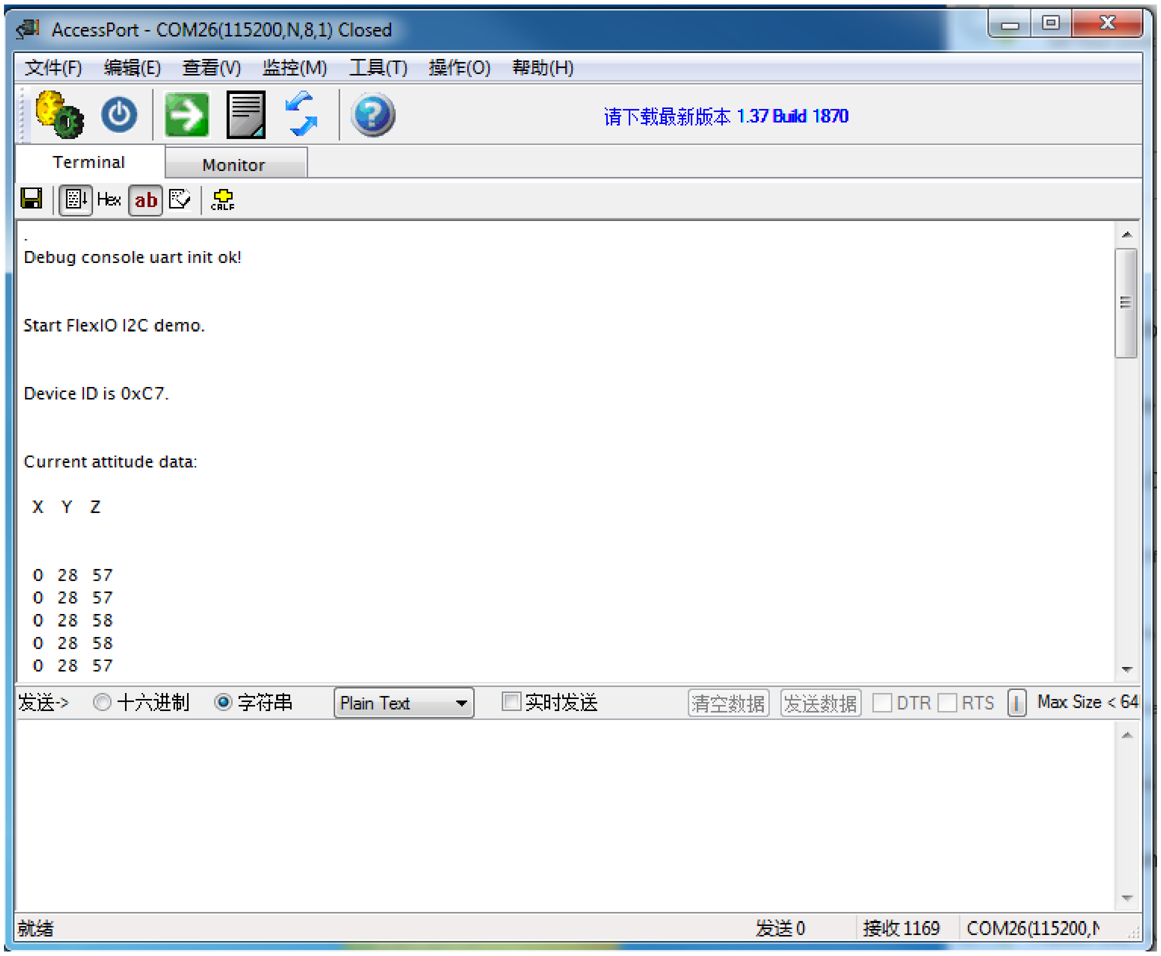
Table 19. **FlexIO pins assignment table for I2C master**

FlexIO I2C Master	
FlexIO I2C master SDA Pin:FlexIO_D0	PTD0
FlexIO I2C master SCL Pin:FlexIO_D1	PTD1

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 S32K144EVB-Q100 board
- Open the UART debug terminal on your PC with 8in1 and 115200bps settings
- Open the project by S32DS workbench on your PC
- Rebuild all files and download the image into target

- The results are printed on the master terminal.



AccessPort - COM26(115200,N,8,1) Closed
文件(F) 编辑(E) 查看(V) 监控(M) 工具(T) 操作(O) 帮助(H)
请下载最新版本 1.37 Build 1870
Terminal
Monitor

```
.
Debug console uart init ok!


Start FlexIO I2C demo.


Device ID is 0xC7.


Current attitude data:

 X  Y  Z


 0  28  57
 0  28  57
 0  28  58
 0  28  58
 0  28  57
```

发送-> 十六进制 字符串 Plain Text 实时发送 清空数据 发送数据 DTR RTS Max Size < 64
就绪 发送 0 接收 1169 COM26(115200,N

Figure 3. **Terminal Utility Output**

6. Generating PWM by Using FlexIO

6.1. Introduction

This use case creates a simple software demo based on the SDK (Software Development Kit version that is included into S32DS_v2018) and basic baremetal driver for a easily FlexIO implementation as PWM with frequency and duty configurations.

This section describes how to generate the PWM by FlexIO. For this application, the development board S32K144EVB-Q100, in Figure 2 has been used.

In the application, FlexIO generates a PWM waveform on the FXIO0_D0 (PTD0) pin. PTD0 is also connected to BLUE LED. Use PWM to lighten the BLUE LED with different duty to show the brightness change under different duty configurations.

6.2. General description

Generating PWM uses the following resources:

One timer — configured as an 8bit PWM mode to control the related pin output.

One pin — controlled by timer to toggle pin out and generate PWM.

Following figure shows the resource assignment:

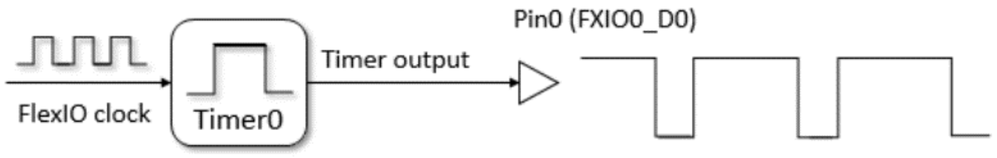


Figure 4. **Resource Assignment of FlexIO to generate PWM**

The detailed configurations and usage information is provided in the following sections.

6.3. Configurations of the timer

This section provides detailed configurations of the timer.

To understand these configurations, see the following descriptions and the reference manual.

Configurations for Timer 0

Timer 0 generates the PWM output to pin 0. It has the following initial configurations.

Table 1. **Initial configuration of Timer 0**

Items	**Configurations**
Timer mode	Dual 8-bit PWM mode
Trigger selection	N/A
Trigger polarity	N/A
Trigger source	Internal
Pin selection	Pin 0
Pin configuration	Output enabled
Pin polarity	Active high
Timer initial output	Output logic 1 when enabled, not affect by reset

Timer decrement source	Decrement on FlexIO clock, Shift on timer output
Timer enable condition	Always enabled
Timer disable condition	Never disabled
Timer reset condition	Never reset
Start bit	Disabled
Stop bit	Disabled
Timer compare value	(((FLEXIO_CLK / freq) * (100 - duty) / 100 - 1) << 8) | ((FLEXIO_CLK / freq) * duty / 100 - 1) (((FLEXIO_CLK*Low_Period)/2-1)<<8)|((FLEXIO_CLK*High_Period)/2-1) or ((((FLEXIO_CLK/freq)*(100-duty)/100)/2-1)<<8)|(((FLEXIO_CLK/freq)*duty/100)/2-1)

The following tips are the key points of using FlexIO to generate PWM.

- Timer mode configuration (TIMCTL[TIMOD] and TIMCTL[TIMDEC])
  - Select the dual 8-bit PWM mode. In this mode, the lower 8-bits of the counter only decrease when the timer output pin is high, and upper 8-bits only decrease when the timer output pin is low. When the lower 8-bits of the counter decreases to 0, the timer output would toggle and lead the upper 8-bits decrease. The lower 8-bits control the PWM high pulse width and the upper 8-bits control the low pulse width.
  - Configure the FlexIO clock (FLEXIO_CLK[1]) as the timer counter decrease source. Therefore, the counter decreases on every FlexIO clock if decrease condition met.
- Timer compare value (TIMCMP[CMP]):
  - The timer compare value is loaded into the timer counter when the timer is first enabled, when the timer is reset or decreased to 0. Enter the dual 8-bits value into this compare register to load the value to ensure every cycle of the PWM has the timer counter reloaded from CMP. This compare value controls the PWM frequency and duty.
  - Timer compare value = (((FLEXIO_CLK / freq) * (100 - duty) / 100 – 1) << 8) | ((FLEXIO_CLK / freq[2]) * duty[3] / 100 - 1)
- Timer output configuration (TIMCFG[TIMOUT])
  - The timer output is set to logic one (high on pin) when enabled the timer is not affected by reset. In 8-bits PWM mode, you must set the timer output to 1 when the timer enabled, otherwise the lower 8-bits value of the counter would not decrease as mentioned above.

[1] FLEXIO_CLK is the clock from clock modules like SCG or MCG, used to control the FlexIO timing.

[2] The frequency of the PWM generated.

[3] The duty of the PWM waveform.

Following figure shows the timing and signal examples. For example, TIMCP=0x58:

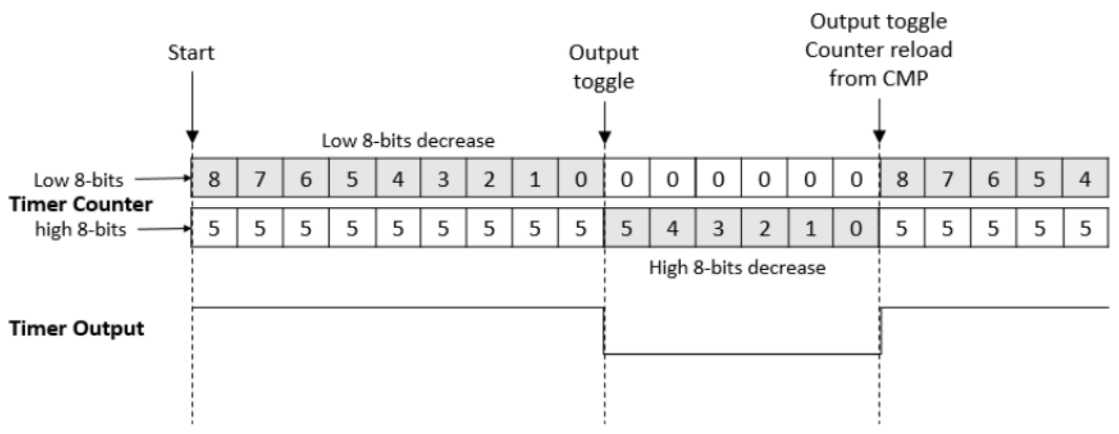


Figure 5. **Timing of the configurations**

6.4. Software implementation overview

This section describes the software implementation. Several driver functions have been implemented in this application are based on the S32K SDK (Software Development Kit version that is included into S32DS_v2018).

A software package is provided along with this application note. The demo software includes only one source file, main.c. This source file provides the following functions to configure the FlexIO as a PWM generator, all of them can be directly used by user in their own codes with minor changes:

- FLEXIO_DRV_InitDevice(uint32_t instance, flexio_device_state_t *deviceState)
- flexio_pwm_init(uint32_t freq, uint8_t duty)
- flexio_pwm_start(void)
- flexio_pwm_stop(void)

6.4.1. flexio_init()

flexio_init() function enables the clock gating of the FlexIO IP module and selects the proper peripheral clock source for FlexIO. The FLEXIO_CLK defined in the source file is exactly the frequency of the peripheral clock source. It resets the FlexIO IP module by SW and re-enables it. This is a general FlexIO IP module initialization function, called before using its shifters and timers.

6.4.2. flexio_pwm_init()

flexio_pwm_init() function configures the timer0 as a 8-bits PWM mode with pin0 output to generate the PWM waveform. The timer detail configurations can be found in section 6.3. The ‘freq’ parameter of this function is the specified PWM frequency you want to generate. The parameter must be within the range of [MIN_FREQ, MAX_FREQ] macros defined in the source file. The ‘duty’ parameter is the specified duty in unit of %, with a range of [1, 99].

6.4.3. flexio_pwm_start()

flexio_pwm_start() function enables the timer0 by setting TIMOD to 8-bits PWM and start generating the PWM.

6.4.4. flexio_pwm_stop()

flexio_pwm_stop() function disables the timer0 by setting TIMOD and disable generating the PWM.

To use the FlexIO as PWM generator, you can call these three functions in sequence like in the main() function. It configures the PWM frequency to 8 KHz, and duty from 99 to 1, to change the brightness of the BLUE LED by PTD0.

6.5. Running the demos

You can download a program image to the microcontroller with Open-SDA. The PC host obtains a serial port after a USB cable is connected between the PC host and the Open-SDA USB on S32K144EVB-Q100.

Download and run the demo. The user can look at the RED LED with brightness changing from min to max. The PWM signal can also be captured with an oscilloscope on the PTD0 pin, this shows the exact frequency and duty.

7. Emulating I2S bus Master using FlexIO

This section describes how to emulate I2S bus Master by using FlexIO. For this application, the evaluation board S32K144EVB-Q100, in following figure has been used.

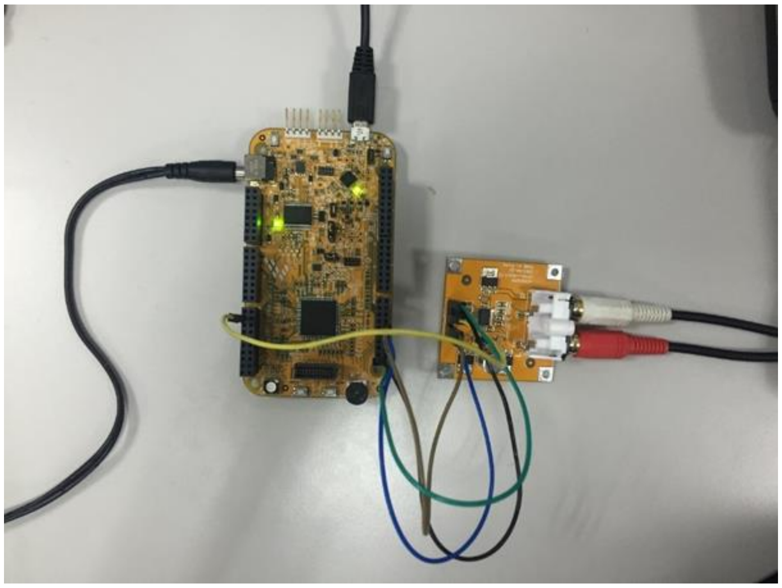


Figure 6. **S32K144EVB-Q100 connected with PCM5102A**

In the application, FlexIO emulates an I2S interface to communicate with the PCM5102A (digital-to-analog converter), in which a general I2S interface is integrated. The audio data is sent from S32K144 to PCM5101A to display.

The following diagram shows the hardware platform and data flows. For more information on Open-SDA in the diagram, see the relevant materials of the NXP development board.)

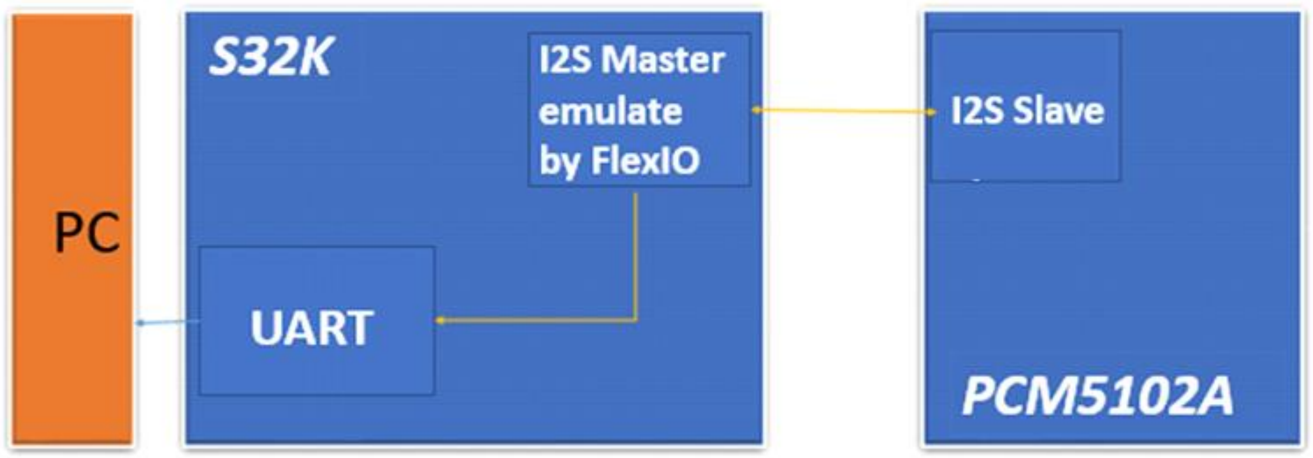


Figure 7. **Hardware Platform and Data Flows of this Application**

7.1. General description

I2S bus master is emulated using:

One shifters —used as a data transmitter.

Two timers —one is used for the load control of the shifter and BCLK output generation, and the other is used for the LRCLK output generation.

Three pins — respectively used as DATA, LRCLK and BCLK.

Following figure shows the resource assignment:

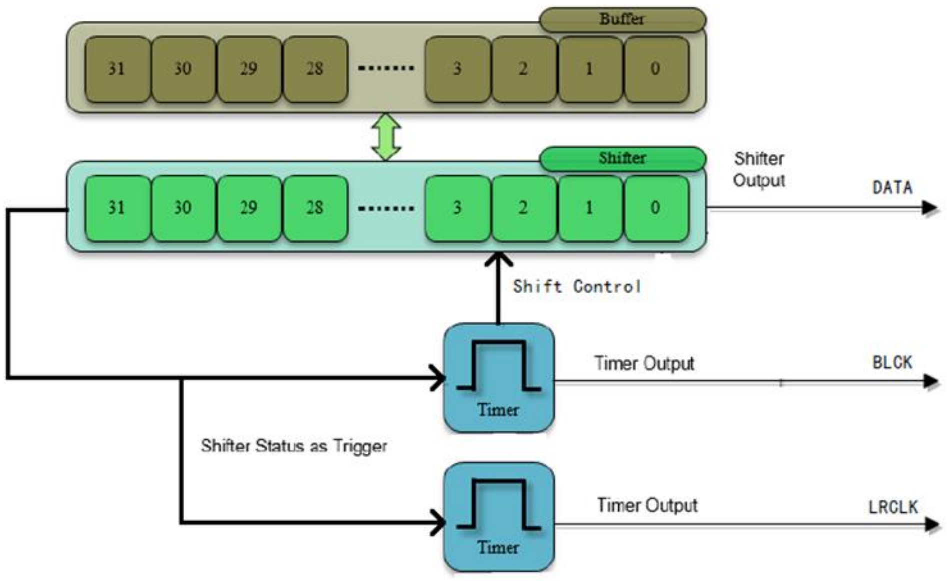


Figure 8. **Resource Assignment of FlexIO to Emulate I2S**

The detailed configurations and usage information is provided in the following sections.

7.2. Configurations of the shifters and timers

This section provides detailed configurations of the shifters and timers.

To understand these configurations, see the following descriptions and the reference manual.

Configurations for Shifter 0

Shifter 0 is used as the data transmitter. It has the following initial configurations.

Table 2. **Initial configuration of Shifter 0**

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

Configurations for Timer 0

Timer 0 is used for the load control of the shifter and BCLK output generation. It has the following initial configurations.

Table 3. **Initial configuration of 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 1
Pin configuration	output enable
Pin polarity	active low
Timer initial output	output logic 0 when enabled, affect by reset
Timer decrement source	decrement on FlexIO clock, shift on timer output
Timer enable condition	on trigger high
Timer disable condition	never disable
Timer reset condition	On timer pin equals to timer output
Start bit	enabled
Stop bit	Disabled
Timer compare value	((n*2-1)<<8) | (baudrate_divider/2-1))[6]

Configurations for Timer 1

Timer 1 is used to the LRCLK output generation. It has the following initial configurations.

Table 4. **Initial configuration of Timer 1**

Items	**Configurations**
Timer mode	Single 16-bit counter mode
Trigger selection	shifter 0 status flag
Trigger polarity	active low
Trigger source	internal trigger
Pin selection	pin 2
Pin configuration	Output enabled
Pin polarity	active high
Timer initial output	output logic 0 when enabled, affect by reset
Timer decrement source	decrement on FlexIO clock, shift on timer output
Timer enable condition	On timer 0 enable
Timer disable condition	never disable
Timer reset condition	Never reset
Start bit	disable
Stop bit	disable
Timer compare value	(baudrate_divider/2-1)[7]

[6] 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

7.3. Software implementation overview

This section describes the software implementation. Several driver functions have been implemented in this application, which are based on the S32K SDK (Software Development Kit version that is included into S32DS_v2018).

The demo runs in polling mode. This section mainly describes several major driver functions.

Initialize Function

The Initialize Function is used to configure the shifters and timers in the application’s initialization phase. The prototype is:

void FlexIO_I2S_Init(void)

Transmit Function

This function is used to transmit one or more bytes to a given address of the slave device. The prototype is:

*status_t FLEXIO_I2S_DRV_MasterSendData (flexio_i2s_master_state_t * master,*
*const uint8_t * txBuff,*
uint32_t txSize
)
where,

master	-Pointer to the FLEXIO_I2S master driver context structure.
txBuff	-pointer to the data to be transferred
txSize	-length in bytes of the data to be transferred
Returns	**-** Error or success status returned by API

7.4. Running the demos

This demo runs on S32K144EVB-Q100. The FlexIO pins assignment for I2S master and slave are shown in the following table:

Table 5. **FlexIO pins assignment table for I2S master**

FlexIO I2S Master	
FlexIO I2S master DATA Pin:FlexIO_D0	PTD0
FlexIO I2S master BCLK Pin:FlexIO_D1	PTD1
FlexIO I2S master LRCLK Pin:FlexIO_D2	PTE15

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

[7] Baudrate_divider is a value used for dividing the baud rate from the FlexIO clock source.

Plug in the Micro USB to connect the PC and target the S32K144EVB-Q100 board

Open the UART debug terminal on your PC with 8in1 and 115200bps settings

Open the project by S32DS workbench on your PC

Rebuild all files and download the image into target

Press any key to run the demo

After the data begin to transfer you can hear the music.

8. Emulating LIN Master/Slave by using FlexIO

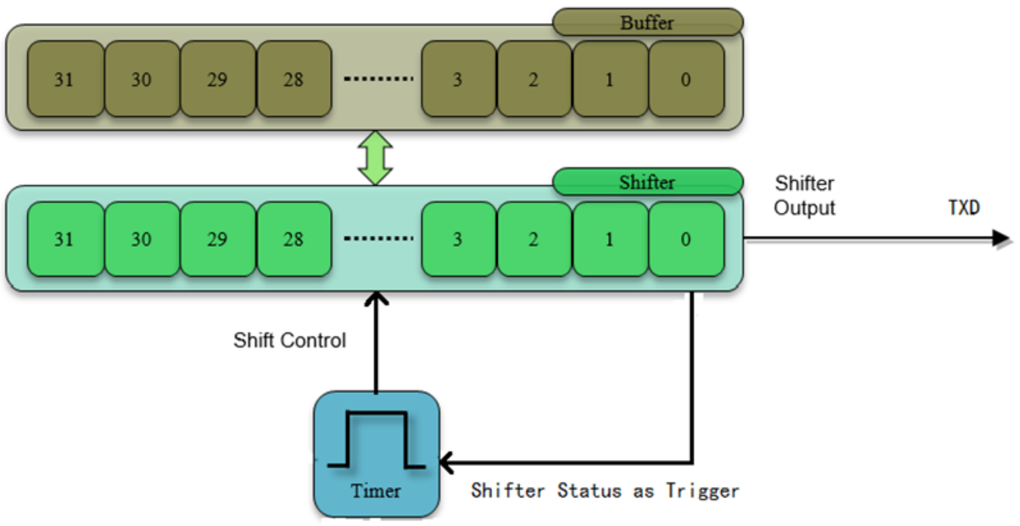
8.1. Introduction

The standalone peripheral module FlexIO is used as an additional peripheral module of the microcontroller and is not a replacement of the LIN/UART peripheral. This example creates a simple software demo based on S32K SDK and including bare metal configuration example drivers for you to use FlexIO to emulate the LIN.

8.2. Emulating LIN using FlexIO

LIN implementation uses the following FlexIO module resources,

- Transmitter configuration
  - One 16-bits timer - Dual 8-bit baud mode.
  - One 32-bits shifter - Transmit mode.
  - Two Output pins
    - controlled by timer to generate the baud rate.
    - to output shifter buffer data.
- Receiver configuration:
  - One 16-bits timer - Dual 8-bit baud mode.
  - One 32-bits shifter - Receive mode.
  - Two I/O pins
    - generating baud rate.
    - shifter input data.



Note: Timers and Shifters configuration is based in UART emulation chapter 3#.

Available FlexIO resources after this implementation are 2 timers, 2 shifters and 4 pins, that can be used for generate a PWM signal, emulate a UART, I2C, i2S or even another Master LIN instance.

8.3. Configurations the Shifters and Timers

Transmitter:

In timer reset Modify CMP value to:

- 32 bits Transfer for Master sending ID or

- 8 bits Transfer for Master sending a byte.

• Load shifter buffer with

- PID << 24u | 0x555000

- Data to send.

• Shift the data to the pin output.

• Start and stop bits are automatically loaded before or after data.

Receiver

The data is shifted in when the store event is signaled.

• The status flag indicates when data can be read (generate interrupt).

• Store into the shifter buffer.

• Reading shifter buffer byte swapped register.

LIN, Master and Slave use the same FlexIO configuration with minimum changes,

When Master mode is used: Star and Stop bit are disable.

When Slave mode is used: first time the timer compare is configured to wait for 13-bits of break field.

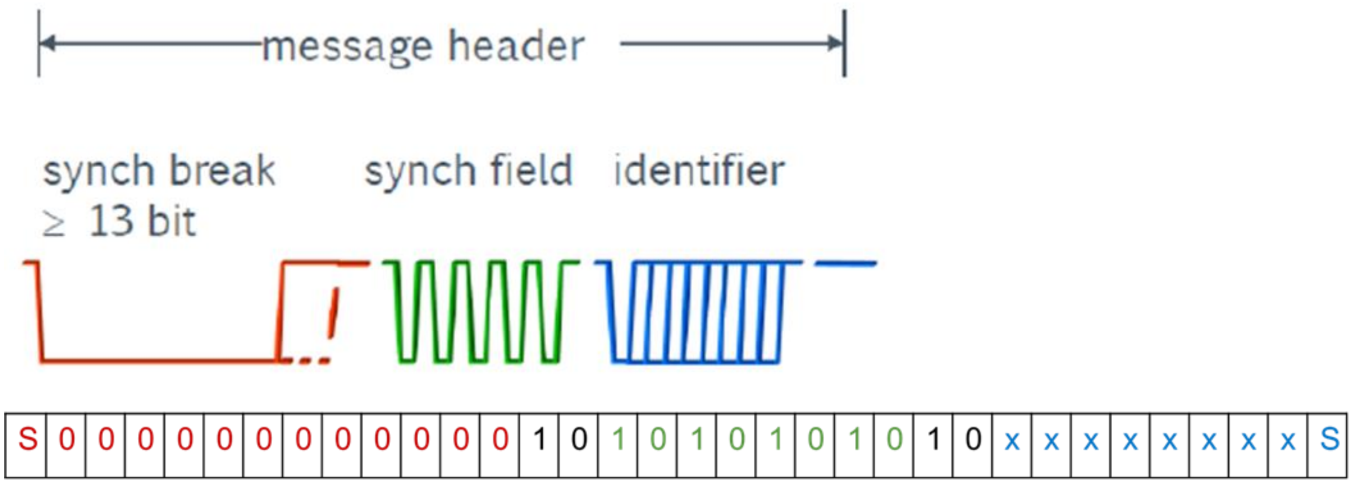

message header
synch break
≥ 13 bit
synch field
identifier
S 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 x x x x x x x x S

Table 6. **Initial configuration of Shifter 0**

Items	**Configurations**
Shifter mode	transmit
Timer selection	timer 0
Timer polarity	on posedge of shift clock
Pin selection	pin 0
Pin configuration	output enable
Pin polarity	active high
Input source	from pin
Start bit	transmitter outputs start bit value ‘0’ before loading data on first shift
Stop bit	transmitter outputs stop bit value ‘1’ on store
Buffer used	shifter buffer

Table 7. **Initial configuration of Timer 0**

Items	**Configurations**
Timer mode	dual 8-bit counters baud mode
Trigger selection	shifter 0 status flag
Trigger polarity	active low
Trigger source	internal trigger
Pin selection	pin 1
Pin configuration	output disable
Pin polarity	active high
Timer initial output	output logic 0 when enabled, 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	on trigger rising edge
Start bit	enabled
Stop bit	enabled on timer compare and timer disable
Timer compare value	$((transfer_{size} \ll 1) - 1) \ll 8 \mid \left(\frac{FlexIO_{freq}}{BaudRate \ll 1}\right) - 1$

Table 8. **Initial configuration of Shifter 1**

Items	**Configurations**
Shifter mode	receive
Timer selection	timer 1
Timer polarity	on negedge of shift clock
Pin selection	pin 2
Pin configuration	output disable
Pin polarity	active high
Input source	from pin
Start bit	transmitter outputs start bit value ‘0’ before loading data on first shift
Stop bit	transmitter outputs stop bit value ‘1’ on store
Buffer used	shifter buffer byte swapped

Table 9. **Initial configuration of Timer 1**

Items	**Configurations**
Timer mode	dual 8-bit counters baud mode
Trigger selection	pin 2 input
Trigger polarity	active low
Trigger source	internal trigger
Pin selection	pin 3
Pin configuration	output disabled
Pin polarity	active high
Timer initial output	output logic 0 when enabled, affect by reset
Timer decrement source	decrement on FlexIO clock, shift on timer output
Timer enable condition	on trigger rising edge
Timer disable condition	on timer compare
Timer reset condition	Never reset
Start bit	enable
Stop bit	enable on timer compare and timer disable
Timer compare value	$((transfer_{size} \ll 1) - 1) \ll 8$

	$\| \left(\frac{FlexIO_{freq}}{BaudRate \ll 1}\right) - 1$

FlexIO input frequency is the SYSTEM OSC divide by 8 = 1Mhz

Table 10. **Configuration of Timer 0 to send LIN header**

Items	**Configurations**
Start bit	disable
Stop bit	Disable

Available FlexIO resources after this implementation are 2 timers, 2 shifters and 4 pins, that can be used for generating a PWM signal, emulate a UART, I2C, i2S or even another Master LIN instance.

9. Conclusion

FlexIO is a new peripheral on S32K family. Thanks to the high flexibility of the shifters and timers, FlexIO has the capability to emulate a wide range of protocols such as SPI, I2C, UART, I2S, LIN and generate PWM, but is not limited only to emulate these peripherals the high flexibility helps us to emulate many other communications interfaces.

10. Revision history

Table 11. **Revision history**

Revision number	Date	Substantive changes
0	06/2018	Initial release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12174
Rev. 0
06/2018