AN13358

FlexIO Emulation on SPI Slave Device which Works in Continuous Mode with Dynamic Frame Size

Rev. 0 — 23 August 2021 Application Note

Contents

1 Introduction

This application note illustrates the FlexIO emulation on Serial Peripheral Interface (SPI) slave device which works in continuous mode with dynamic frame size on "i.MX RT1010".

For this application note, the hardware board is "RT1010 EVK RevC". The software is *SDK 2.9.1* and the demo code is developed based on `edma_lpspi_transfer` slave project and the path is

`boards\evkmimxrt1010\driver_examples\flexio\spi\edma_lpspi_transfer\slave`.

FlexIO is an on-chip peripheral available on NXP i.MX RT series. It is a highly configurable module which is capable of emulating a wide range of communication protocols, such as UART, I^2C, SPI, and I^2S.

2 FlexIO overview

The FlexIO module of the "i.MX RT1010" provide the following key features:

- Array of 32-bit shift registers with transmit, receive, and data match modes.
- Double buffered shifter operation for continuous data transfer.
- Automatic start/stop bit generation.
- Interrupt, Direct Memory Access (DMA), or polled transmit/receive operation.
- Programmable baud rates independent of bus clock frequency, with support for asynchronous operation during stop modes.
- Highly flexible 16-bit timers with support for a variety of internal or external trigger, reset, with enable and disable conditions.
- Programmable logic mode for integrating external digital logic functions on-chip or combining pin/shifter/timer functions to generate complex outputs.
- Programmable state machine for offloading basic system control functions from CPU with support for up to eight states, eight outputs, and three selectable inputs per state.

Figure 1 gives a high-level overview of FlexIO timers and shifters configuration.

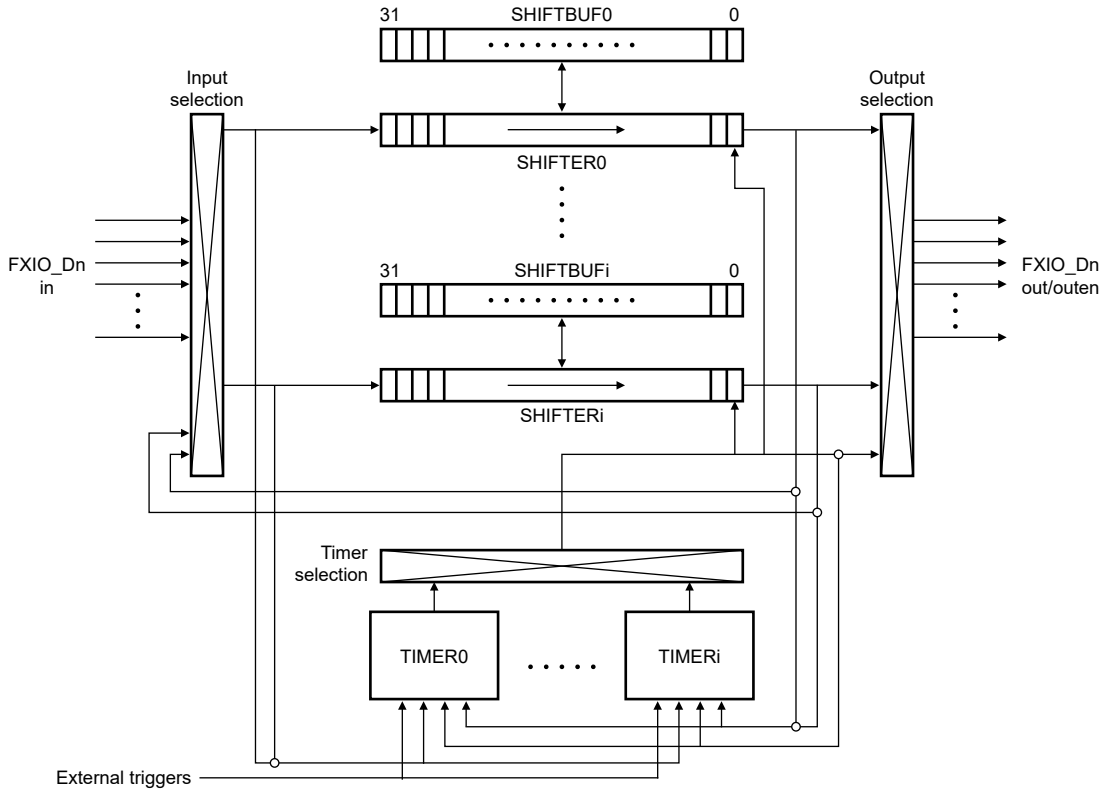

Figure 1. FlexIO block diagram

From the FLEXIO_PARAM register, users can read the amount of these resources, for example, shifter, timer, pin, and trigger. For instance, there are eight shifters, eight timers, 32-pins, and two external triggers in "i.MX RT1010" (In this device, FlexIO only has 27-pins).

3 Emulating a SPI slave device in continuous mode

NOTE

Read the *Emulating SPI with the FlexIO on i.MX RT Series MCU* (document AN12780) before proceeding.

3.1 SPI slave configuration in discontinuous mode

To emulate an SPI slave device, the following resources are needed:

- One Timer - for the load/store/shift control of the two shifters.
- Two Shifters - one for data transmitter and the other for receiver.
- Four Pins – used as SPI_CS, SPI_SCK, SPI_MOSI, and SPI_MISO.

Figure 2 shows the FlexIO SPI slave configuration diagram.

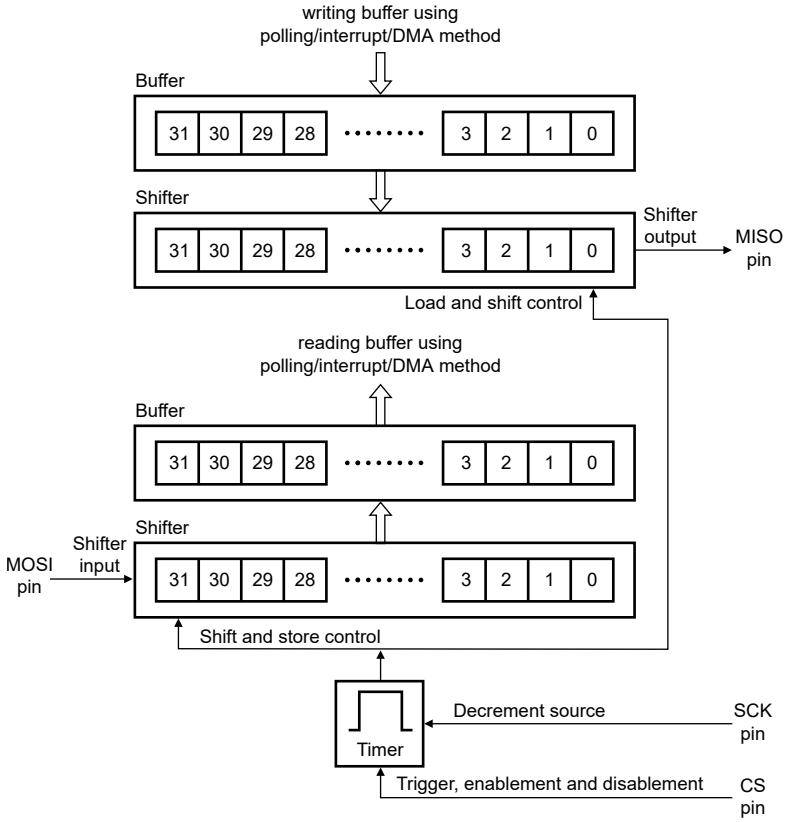

Figure 2. FlexIO SPI slave block diagram

In slave mode, timer 0 is used by the SPI slave to acquire `SPI_SCK` signal on `FlexIO_D26` pin from master to load/store/shift control of the two shifters. The `SPI_SCK` and `SPI_CS` signal are configured as inputs and driven by the SPI bus master. Select pin `FlexIO_D0` of `SPI_CS` as the trigger input to timer 0. Shifter 0 is used as SPI slave transmitter on pin `FlexIO_D21`, shifter 1 is used as SPI slave receiver on pin `FlexIO_D22`.

3.2 SPI slave configuration in continuous mode

To support continuous mode, the timer must configure as disabled on trigger raising edge of the Chip Select (CS) signal. Besides this, the timer must be enabled all the time during a frame transfer.

NOTE

The "frame" may be one byte or several bytes.

However, the timer cannot be disabled at the end of a frame, that means an additional load event occurs after the last bit of the last word in a frame. See Figure 3, the bit marked with green color. The reason is that, the shifter which in transmit mode, it loads the data from the `SHIFBUF` on the expiration of the timer. That means an invalid data goes on the next frame.

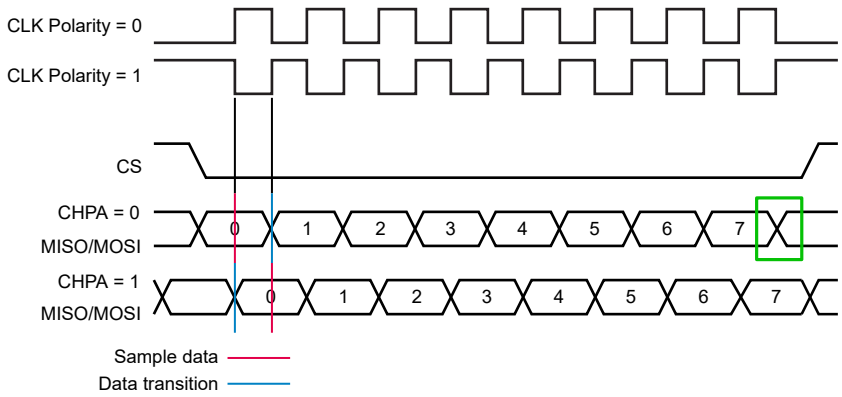

Figure 3. FlexIO SPI slave block diagram

For RX shifter, it triggers an additional store event that rises the CS pin. The CS pin rise up disables the timer. Therefore, the RX shifter stores the content to `SHIFBUF` and triggers a new DMA loop. Invalid data is stored to the buffer and this invalid data should be 0.

To avoid these problems, reset or flush the shifter can be a choice. Because FlexIO does not have a register bit to reset/flush a shifter, the following configuration can be used.

- TX shifter: Disable the shifter and enable it with TX mode, the TX shifter is flushed.
- RX shifter: Read it as buffer register then the shifter is flushed.

In demo code, shifter0 is used as the TX shifter and shifter1 is used as the RX shifter, so that the following API can be used after a frame has been transmitted.

```
void FLEXIO_SPI_FlushShifters(FLEXIO_SPI_Type *base)
{
 volatile uint32_t tmp;
 base->flexioBase->SHIFTCTL[base->shifterIndex[0]] &= ~FLEXIO_SHIFTCTL_SMOD_MASK;
 base->flexioBase->SHIFTCTL[base->shifterIndex[0]] |=
FLEXIO_SHIFTCTL_SMOD(kFLEXIO_ShifterModeTransmit);
 tmp = base->flexioBase->SHIFTBUF[base->shifterIndex[1]];
 __DSB();
}
```

In fact, this additional load occurs during the transmission of every byte in a frame, but because it is a continuous working mode, no exception is generated.

To achieve the continuous mode, Table 1 lists the configuration for timer 0, configuration marked as **bold text** is the key point which is different with the default SDK. After this modification, the slave device can work in continuous mode.

Table 1. Configurations for timer 0

Items	Configurations
Trigger select	Trigger from FlexIO_D0 input
Trigger polarity	Active low
Trigger source	Internal trigger
Pin config	Output disable
Pin select	FlexIO_D26

Table continues on the next page...

Table 1. Configurations for timer 0 (continued)

Items	Configurations
Pin polarity	Active high
Timer mode	Single 16-bit counter mode
Timer output	Timer output is logic zero when enabled and is not affected by timer reset
Timer decrement	Decrement counter on pin input, shift clock equals pin input
Timer reset	Timer never resets
Timer disable	**Timer disabled on trigger falling**[1]
Timer enable	Timer enabled on trigger rising edge[1]
Timer stop bit	Disable
Timer start bit	Disable
Timer compare	15 ((bitCountPerChar * 2 - 1)

1. Because the trigger polarity setting is active low, so that the enable signal is the rising edge (The real input signal is a falling edge).

Table 2 lists the configuration for shifter 0.

Table 2. Configurations for shifter 0 TX mode

Items	Configurations
Timer select	Timer 0
Timer polarity	Shift on falling edge of shift clock
Pin config	Shifter pin output
Pin select	FlexIO_D21
Pin polarity	Active high
Shifter mode	Transmit mode
Input source	Input from pin
Shifter stop bit	Disable
Shifter start bit	Disable, transmitter loads data on enable

Table 3 lists the configuration for shifter 1.

Table 3. Configurations for shifter 1 RX mode

Items	Configurations
Timer select	Timer 0
Timer polarity	Shift on rising of shift clock
Pin config	Output disable
Pin select	FlexIO_D22

Table continues on the next page...

Table 3. Configurations for shifter 1 RX mode (continued)

Items	Configurations
Pin polarity	Active high
Shifter mode	Receive mode
Input source	Input from pin
Shifter stop bit	Disable
Shifter start bit	Disable, transmitter loads data on enable

3.3 SPI slave configuration in continuous mode with dynamic frame size

As the default SDK settings, the DMA is used to move data to improve efficiency. But DMA must receive a fixed size of data before it can generate an interrupt. In practical applications, the size of the data transmitted in each frame is not fixed, and the slave device usually does not know how much data there is in a frame. In this case, the method of generating an interrupt after receiving the specified size of data using DMA cannot meet the actual requirements. During the transmission of LPSPI, the falling edge of CS pin means the beginning of the transfer, and the rising edge of CS means the end of the transfer. Therefore, the slave device can know that a frame of data has completed the transmission by detecting the rising and falling edges of the CS pin.

In order to achieve this function, a timer can be used to detect the status of the CS pin. The basic method it that using a timer works under 16-bit counter mode and the count value is 0. That means once the timer timeout, it can generate a compare event and there is a FLEXIO interrupt generated. The slave device handles this interrupt and know that a frame data has been received. Now, there are two questions. The first is when the timer enables and what is the timer decrement source, and the second is how the slave knows how much data it has received using DMA.

For the first question, the timer can be enabled by the falling edge of pin or trigger signal. About the timer decrement source, both decrement on pin input or trigger input can be used. By these settings, the timer can be enabled by CS pin falling edge, then when CS pin rises up, the compare event occur and at the same time a FlexIO interrupt generated. Besides this, the timer also must be disabled on timer compare event.

Table 4 and Table 5 lists the detailed settings by using falling edge of pin and trigger signal.

Table 4. Configurations for timer 1 using edge of pin

Items	Configurations
Trigger select	NA
Trigger polarity	NA
Trigger source	NA
Pin config	output disable
Pin select	FlexIO_D00
Pin polarity	Active low
Timer mode	Single 16-bit counter mode
Timer output	Timer output is logic one when enabled and is not affected by timer reset
Timer decrement	Decrement counter on pin input (both edges) shift clock equals pin input
Timer reset	Timer never resets
Timer disable	Timer disabled on Timer compares

Table continues on the next page...

Table 4. Configurations for timer 1 using edge of pin (continued)

Items	Configurations
Timer enable	Timer enabled on pin rising edge[1]
Timer stop bit	Disable
Timer start bit	Disable
Timer compare	0

1. Because the pin polarity setting is active low, so that the enable signal is the rising edge (The real input signal is a falling edge).

Table 5. Configurations for timer 1 using edge of trigger

Items	Configurations
Trigger select	Trigger from FlexIO_D0 input
Trigger polarity	Active low
Trigger source	Internal trigger
Pin config	Output disable
Pin select	NA
Pin polarity	NA
Timer mode	Single 16-bit counter mode
Timer output	Timer output is logic one when enabled and is not affected by timer reset
Timer decrement	Decrement counter on trigger input (both edges) shift clock equals timer output
Timer reset	Timer never resets
Timer disable	Timer disabled on timer compares
Timer enable	Timer enabled on trigger rising edge[1]
Timer stop bit	Disable
Timer start bit	Disable
Timer compare	0

1. Because the trigger polarity setting is active low, so that the enable signal is the rising edge (The real input signal is a falling edge).

The second question is how the slave knows how much data it has received using DMA. There is an API can be used:

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCountEDMA(FLEXIO_SPI_Type *base,
 flexio_spi_slave_edma_handle_t *handle,
 size_t *count)
```

But for this API, note the following points:

- The comment of this API is not very clear (has updated on *SDK 2.11.0 version*), the function for this API is to get the number of bytes transferred so far by FlexIO SPI DMA.

- The result of this API (count, the third parameter on API) cannot be used directly, because when enable the continuous mode, there is an overrun, it triggers a new DMA loop. So that, the received data size should be:

```
size = count - 1; //The count is the third parameter on API
```

After getting the size, the following API must be used to abort a FlexIO SPI using DMA. For now, the real function of this API is `STOP` transfer, not `Abort`, that means some modification work must be done. This issue is fixed on *SDK 2.11.0 version*.

```
void FLEXIO_SPI_SlaveTransferAbortEDMA(FLEXIO_SPI_Type *base, flexio_spi_slave_edma_handle_t *handle)
```

Update this API on `fsl_flexio_spi_edma.c` , line "409" and "410", use the following code instead of the old one.

```
EDMA_AbortTransfer(handle->txHandle);
EDMA_AbortTransfer(handle->rxHandle);
```

After solving these two problems, the slave can receive data frames with changing data size. But there is still a limitation in this case, that is the size of a frame should be less than or equal to the max size of DMA receive size.

4 Example code

Before running the code, the board must be set up.

- Remove the resistor R90 and weld 0 Ω resistor to R800.
- Connect the master and slave on the EVK board by the following settings.

For details, see Table 6.

Table 6. Example code

Pin name	Master (LPSPI1)		Pin name	Slave (FlexIO SPI)
SOUT	J57-8	← →	SIN	J26-6
SIN	J57-10	← →	SOUT	J26-4
SCK	J57-12	← →	SCK	J26-8
CS	J57-6	← →	CS	J56-10

The specific implementation code is provided in the attachment of this application note, open the `edma_lpspi_transfer` slave project and replace the files with the files in attachment.

Path: `boards\evkmimxrt1010\driver_examples\flexio\spi\edma_lpspi_transfer\slave`

On the default settings, master sends 16 bytes to the slave and the slave generates a FlexIO interrupt. The slave can receive a maximum of 64 bytes. User can change the following parameters to configure the size of a frame.

```
masterXfer.dataSize = TRANSFER_SIZE;
```

Figure 4 shows the waveform of the SPI signal in continuous mode. After a frame has been transmit/received, a simple check will check whether any problems occurred during the transmission. If no problem occurs, inputting any character in the serial terminal starts a new frame of data transmission.

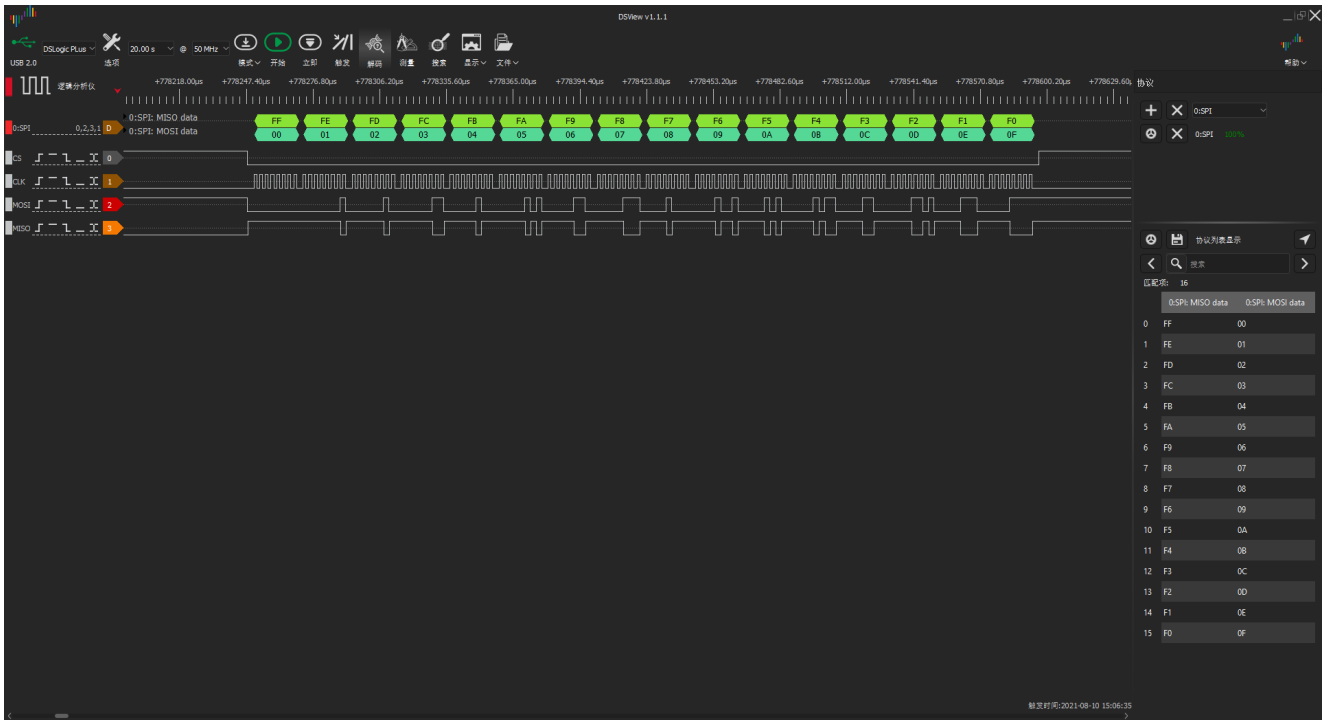

Figure 4. Waveform of the SPI signal in continuous mode

5 Revision history

Table 7 summarizes the changes done to this document since the initial release.

Table 7. Revision history

Revision number	Date	Substantive changes
0	23 August 2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 23 August 2021
Document identifier: AN13358