Freescale Semiconductor, Inc.
Application Note

Document Number: AN5116
Rev. 0, 04/2015

Emulating IRDA by Using FlexIO

How to use FlexIO SDK UART driver to emulate IRDA

1. Introduction

The Kinetis Software Development Kit (SDK) provides robust peripheral drivers, stacks, middleware, and example applications designed to simplify and accelerate application development on any Kinetis MCU. The FlexIO peripheral was initially introduced in the Kinetis KL43 family. SDK 1.2 GA has fully supported this peripheral with many protocol drivers such as UART, I2C, I2S, and SPI.

This use case uses the FlexIO UART driver to create an IRDA protocol, like Kinetis, with UART supporting IRDA. It uses two additional timers besides the FlexIO UART driver to encode and decode the UART signal into IRDA waveform. The Freescale FRDM platform is used for this demonstration.

Contents

2. FlexIO SDK driver overview

SDK 1.2 GA/SA fully supports FlexIO drivers to emulate the following protocols:

- 12C
- SPI
- UART
- I2S

To make it easier for the user to use these drivers, the SDK 1.2 GA/SA package provides some examples on how to these drivers. For example, the FlexIO UART driver uses two timers and two shifters to transmit and receive data asynchronously.

3. Required hardware and software

This document describes the example application based on the Freescale FRDM system. The basic concept can be easily implemented on the customized hardware as well.

The application can be easily set up using the following FRDM boards:

- FRDM-KL43Z Freescale Freedom development board

This use case is delivered with the SDK 1.2 KL33Z64 SA package. The user can download the full package from freescale.com/ksdk.

4. IRDA timer encoding and decoding configuration

UART data is in NRZ format. To encode this data into the IRDA protocol, one FlexIO timer in dual 8-bit counters PWM mode is needed to modulate the NRZ data. To receive the IRDA signal, one FlexIO timer in dual 8-bit counters baud/bit mode is needed to decode IRDA signal into NRZ format. This use case diagram, similar to Figure 1, is provided in SDK 1.2 SA for KL33Z64 standalone package: (*examples/frdmkl43zkl33z4/driver_examples/flexio/irda*).

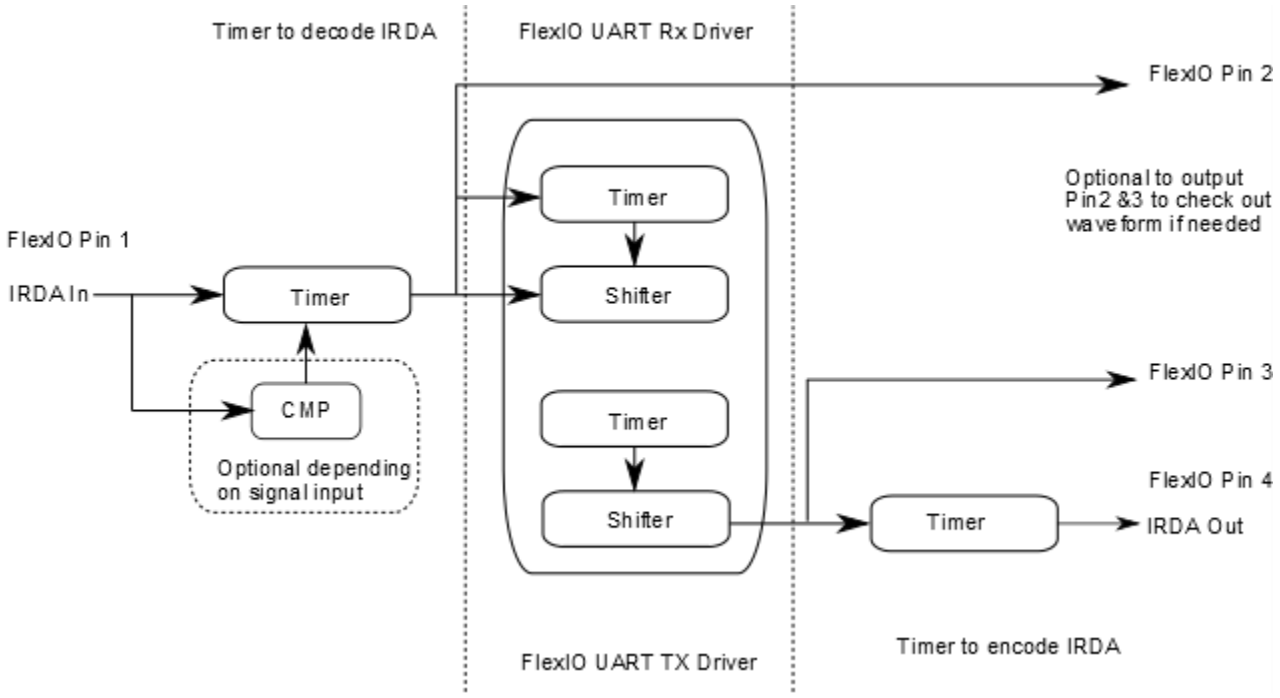

Figure 1. FlexIO UART driver and IRDA encode/decode

The waveform is similar to Figure 2:

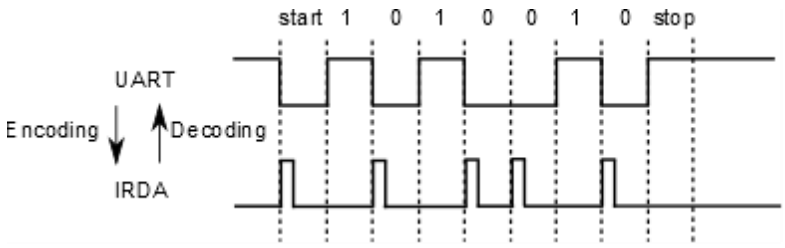

Figure 2. FlexIO UART and IRDA waveform

4.1. IRDA encoding timer configuration

The encoding timer is configured to be triggered by the UART NRZ data falling edge, which is the first edge for the start signal. The following lists the details of the timer configuration.

Timer Control Register (FLEXIO_TIMCTLn):

- TRGSEL: Select FlexIO UART driver TX output pin.
- TRGPOL: Select trigger polarity active low – 1.
- TRGSRC: Select trigger source internal – 1.
- PINCFG: Select timer pin output – 11.
- PINSEL: Select timer pin to use. Select a pin which is not used by FlexIO UART TX.
- PINPOL: Select timer output polarity. This can be high true or low true depending on the external IRDA device used.

- TIMOD: Select timer running mode, dual 8-bit counters PWM mode.

Timer Configuration Register (FLEXIO_TIMCFGn):

- TIMOUT: Select timer output one, not affected by timer reset – 00.
- TIMDEC: Select timer decreased on FlexIO clock – 00.
- TIMRST: Select timer never reset – 000.
- TIMDIS: Select timer disabled on timer compared – 010.
- TIMENA: Select timer enabled on trigger event high, which means input low for this case – 010.
- TSTOP: Select timer stop bit enabled on timer disabled – 010.
- TSTART: Select timer start bit disabled – 0.

Timer Compare Register (FLEXIO_TIMCMPn):

- CMP: Timer value to set. The time is running in dual 8-bit counter mode, so the lower 8-bits configure the high period of the output to (CMP[7:0] + 1) * 2. The upper 8-bits configure the low period of the output to (CMP[15:8] + 1) * 2.

4.2. IRDA decoding time configuration

The decoding timer is configured to be triggered by the IRDA data rising edge, which is the first edge for the start signal. The following lists the details of the timer configuration.

Timer Control Register (FLEXIO_TIMCTLn):

- TRGSEL: Select FlexIO triggered by CMP0 output or FlexIO pin input, depending on user application.
- TRGPOL: Select trigger polarity active low or high depending on user application, the IRDA receiver output signal polarity.
- TRGSRC: Select trigger source external or internal depending on TRGSEL configuration.
- PINCFG: Select timer pin output if user wants to check out the decoding signal to meet NRZ format or not. Otherwise, disable it.
- PINSEL: Select timer pin to use.
- PINPOL: Select timer output polarity, active low – 1.
- TIMOD: Select timer running mode, dual 8-bit counters baud/bit mode.

Timer Configuration Register (FLEXIO_TIMCFGn):

- TIMOUT: Select timer output one. Not affected by timer reset – 00.
- TIMDEC: Select timer decreased on FlexIO clock – 00.
- TIMRST: Select timer reset on timer trigger rising edge –110.

- TIMDIS: Select timer disabled on timer compared – 010.
- TIMENA: Select timer enabled on trigger high – 110.
- TSTOP: Select timer stop disabled – 000.
- TSTART: Select timer start bit disabled – 0.

Timer Compare Register (FLEXIO_TIMCMPn):

- CMP: Timer value to set. The time is running in dual 8-bit counters baud/bit mode. The lower 8-bits configures the baud rate divider equal to (CMP[7:0] + 1) * 2. The upper 8-bits configure the number of bits in each word equal to (CMP[15:8] + 1) * 2, which is not used for this case.

5. SDK driver example for FlexIO IRDA

This use case is implemented in the KL33Z64 SDK 1.2 SA package. The example code is located in *{installation path}/examples/src/flexio/irda*, and the supported IDE workspace files are located in *{installation path}/examples/frdmkl43zkl33z/driver_examples/flexio/irda/{IDE}*. This example can be easily ported to any hardware platform. For SDK driver examples, main.c provides user details on how to run the demo.

5.1. FlexIO UART driver initialization

The user needs to configure the FlexIO UART driver work mode like the following:

```
flexio_user_config_t userConfig =
{
    .useInt = true,
    .onDozeEnable = false,
    .onDebugEnable = true,
    .fastAccessEnable = false
};
CLOCK_SYS_EnableFlexioClock(FLEXIO_INSTANCE);
freq = CLOCK_SYS_GetFlexioFreq(FLEXIO_INSTANCE);

FLEXIO_DRV_Init(instance,&userConfig);
 /* Fill in uart config data */
uartConfig.bitCounter = kFlexIOUart8BitsPerChar;
uartConfig.baudRate = FLEXIO_UART_BAUDRATE;
uartConfig.uartMode = flexioUART_TxRx;
uartConfig.txConfig.pinIdx = FLEXIO_UART_TX_PIN;
uartConfig.txConfig.shifterIdx = FLEXIO_UART_TX_SHIFTER;
uartConfig.txConfig.timerIdx = FLEXIO_UART_TX_TIMER;
uartConfig.rxConfig.pinIdx = FLEXIO_UART_RX_PIN;
uartConfig.rxConfig.shifterIdx = FLEXIO_UART_RX_SHIFTER;
uartConfig.rxConfig.timerIdx = FLEXIO_UART_RX_TIMER;
 uartState.rxBuff= rxBuff;
 /* init the uart module with base address and config structure*/
FLEXIO_UART_DRV_Init(instance, &uartState, &uartConfig);
```

5.2. FlexIO IRDA driver initialization

The user needs to configure the FlexIO timer to encode and decode UART signals:

```
    txConfig.baudrate = uartConfig.baudRate;
    txConfig.flexioFrequency = freq;
    rxConfig = txConfig;
    rxConfig.timerIdx = FLEXIO_UART_IRDA_RX_TIMER;
    rxConfig.timerPinIdx = FLEXIO_UART_RX_PIN;
#if LOOPBACK_TEST
    rxConfig.trigPinIdx = FLEXIO_UART_IRDA_TX_PIN;
#else
    rxConfig.trigPinIdx = FLEXIO_UART_IRDA_RX_PIN;
#endif
    txConfig.timerIdx = FLEXIO_UART_IRDA_TX_TIMER;
    txConfig.timerPinIdx = FLEXIO_UART_IRDA_TX_PIN;
    txConfig.trigPinIdx = FLEXIO_UART_TX_PIN;
#if IRDA_RX_CMP0_TRIG
    // Configuration for cmp
    cmp_state_t cmpState;
    cmp_comparator_config_t cmpUserConfig;
    cmp_sample_filter_config_t cmpSampleFilterConfig;
    cmp_dac_config_t cmpDacConfig;

    // Disable rising interrupt
    // Disable falling interrupt
    // Init the CMP comparator.
    CMP_DRV_StructInitUserConfigDefault(&cmpUserConfig,
(cmp_chn_mux_mode_t)BOARD_CMP_CHANNEL, kCmpInputChnDac);
    cmpUserConfig.risingIntEnable = false;
    cmpUserConfig.fallingIntEnable = false;
    CMP_DRV_Init(CMP_INSTANCE, &cmpState, &cmpUserConfig);

    // Configure the internal DAC when in used.
    cmpDacConfig.dacEnable = true;
    cmpDacConfig.dacValue = IRDA_RX_CMP0_DAC_VALUE; // 0U - 63U
    cmpDacConfig.refVoltSrcMode = kCmpDacRefVoltSrcOf2;
    CMP_DRV_ConfigDacChn(CMP_INSTANCE, &cmpDacConfig);

    // Configure the Sample/Filter Mode.
    cmpSampleFilterConfig.workMode = kCmpContinuousMode;
    CMP_DRV_ConfigSampleFilter(CMP_INSTANCE, &cmpSampleFilterConfig);

    // Start the CMP function.
    CMP_DRV_Start(CMP_INSTANCE);

#endif

    //configure FlexIO timers to decode IRDA signals
    FLEXIO_IRDA_Init(FLEXIO, &rxConfig, &txConfig);

    FLEXIO_DRV_Start(instance);
```

5.3. FlexIO IRDA driver test

For the encoding and decoding timer to always work, use the FlexIO UART driver to trigger the working timer. The following example provides the user three tests for using the FlexIO UART driver.

- Non-blocking for both TX/RX

- Blocking for TX
- Blocking for RX

6. References

The references listed below have additional information regarding FlexIO for the Kinetis L family. Find a particular reference manual, data sheet, or errata report by choosing a device on the Kinetis (freescale.com/Kinetis) pages and select the family you are interested in to find more information. To find latest SDK installer, visit www.freescale.com/ksdk.

- **MCU Reference Manuals:** The reference manuals contain MCU-specific implementation details in the Chip Configuration chapters and include a detailed description of the Resets and Power Management Features of each MCU.
- **MCU Data Sheet Specifications:** The data sheet includes all MCU specifications, including clock rates, low power module power consumption expectations, and so forth.
- **Errata for MCUs:** Device errata identify what functionality and/or specification is not being met due to a problem with the MCU. Most issues have workarounds.

7. Revision History

Table 1. **Revision history**

Revision Number	Date	Substantive changes
0	4/2015	Initial Release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5116
Rev. 0
04/2015