AN14110

Emulating I2C with the FlexIO of i.MX 93 based on Different Operating Systems

Rev. 1 — 15 November 2023 Application note

Document information

Information	Content
Keywords	AN14110, i.MX 93, FlexIO, iMX93EVK
Abstract	This document describes the emulation of I2C with the FlexIO module of i.MX 93 device based on different operating systems, such as Linux, BareMetal, and Zephyr.

1 Introduction

Flexible input/output (FlexIO) is a highly configurable module, which is capable of emulating a wide range of communication protocols, such as UART, I2C, SPI, and I2S. FlexIO was originally exclusive to NXP MCU products. However, due to its ability to emulate various interfaces easily, it has now been ported to various MPU platforms, such as i.MX 93.

In this document, the FlexIO emulation of the I2C bus master is taken as an example to introduce a new module in MPU. It covers the basic principles, hardware implementation, and the software implementation in different operating systems, such as Linux, BareMetal, and Zephyr.

2 FlexIO module overview

The i.MX 93 SoC features two FlexIO instances. Figure 1 shows a high-level overview of the FlexIO module. The rich resources of shifters, counters, and pins make the module work with different functionalities.

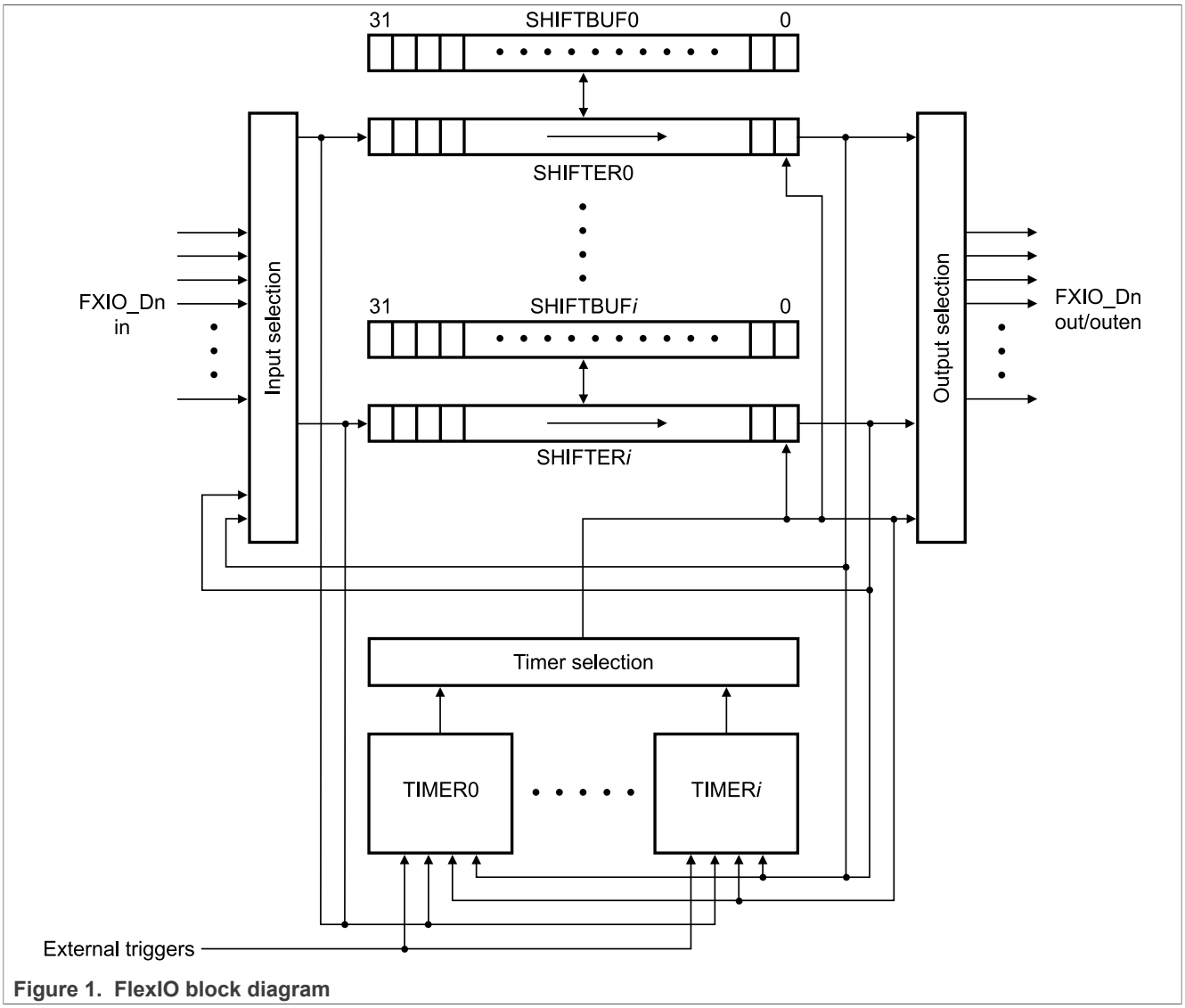

Figure 1. FlexIO block diagram

The hardware resources in the FlexIO module are:

- 32-bit shifter x 8
- 16-bit timer x 8
- Pin x 32

Shifter operations:

- Transmit mode
- Receive mode
- Match Store mode
- Match Continuous mode
- State mode
- Logic mode

Timer operations:

- Timer 8-bit Baud Counter mode
- Timer 8-bit High PWM mode
- Timer 16-bit Counter mode
- Timer 16-bit Counter Disable mode
- Timer 8-bit Word Counter mode
- Timer 8-bit Low PWM mode
- Timer 16-bit Input Capture mode

Pin operations:

- Parallel interface
- Pin synchronization
- Pin override
- Pin interrupt

3 Emulating I2C bus master

This section describes how to emulate the I2C bus master with FlexIO. For this application, the NXP i.MX 93 11x11 EVK FlexIO emulates an I2C interface to communicate with the NXP USB PD PHY IC PTN5110NHQZ.

3.1 General description

I2C controller mode can be supported using two timers, two shifters, and two pins. One timer is used to generate the SCL output and another timer is used to control the shifters. The two shifters are used to transmit and receive each word. When receiving, the transmitter must transmit FFh to the 3-state output. FlexIO inserts a stop bit after every word to generate and verify the ACK or NACK.

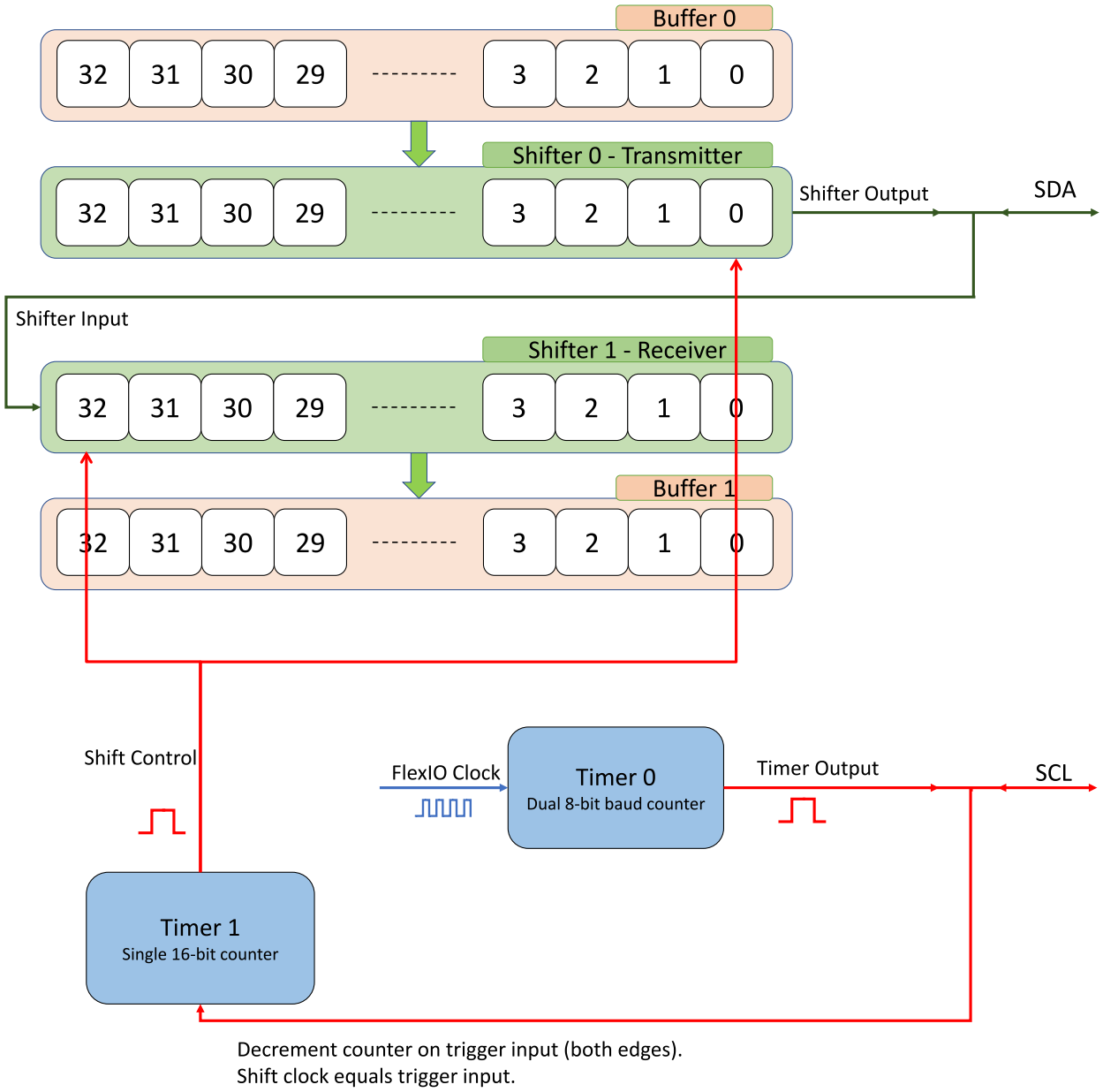

Figure 2. Resource assignment of FlexIO to emulate I2C master

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

3.2 Configuration

This section provides detailed configurations of the shifters and timers.

NOTE

The items listed in this section are the initial settings of the shifters and timers. The software updates some of these settings according to the transmissions.

- Shifter 0 is used as the transmitter. Shifter 1 is used as the receiver. They have the following initial configurations.

Table 1. Configurations for Shifter 0 and Shifter 1

Items	Shifter 0 configurations	Shifter 1 configurations
Shifter mode	Transmit	Receive
Timer selection	Timer 1	Timer 1
Timer polarity	On posedge of shift clock	On negedge of shift clock
Pin selection	Pin 28	Pin 28
Pin configuration	Open-drain or bidirectional output enable	Output disabled
Pin polarity	Active low	Active high
Input source	From pin	From pin
Start bit	Value 0	Disabled
Stop bit	Value 1	Value 0
Buffer used	Bit Byte Swapped register	Bit Byte Swapped register

- Timer 0 is used to generate SCL output and to trigger timer 1. Timer 1 is used to control the Shifter 0 and Shifter 1.

Table 2. Configurations for Timer 0 and Timer 1

Items	Timer 0 configurations	Timer 1 configurations
Timer mode	Dual 8-bit baud/bit mode	Single 16-bit counter mode
Trigger selection	Shifter 0 status flag	Shifter 0 status flag
Trigger polarity	Active low	Active low
Trigger source	Internal	Internal
Pin selection	Pin 29	Pin 29
Pin configuration	Open-drain or bidirectional output enable	Output disabled
Pin polarity	Active high	Active low
Timer initial output	Output logic 0 when enabled and not affected by timer reset	Output logic 1 when enabled and not affected by timer reset
Timer decrement source	Decrement counter on FlexIO clock. Shift clock equals timer output	Decrement counter on pin input (both edges). Shift clock equals pin input.
Timer enable condition	On Trigger high	On timer 0 enable
Timer disable condition	On timer compare (upper 8 bits match and decrement)	On timer 0 disable
Timer reset condition	On timer pin equal to timer output	Never reset
Start bit	Enabled	Enabled
Stop bit	Enabled on timer disable	Enabled on timer compare
Timer compare value	`(((N*9+1)*2-1)<<8) | baudrate_divider` "N*9+1" = Nbytes * [8(bits per byte) + ACK/NACK] + Stop Condition	`8*2-1` "8" means the number bits in a single frame

3.3 Hardware design

By default, the i.MX 93 Evaluation Kit (EVK) configures PTN5110NHQZ using LPI2C3. However, GPIO_IO28 and GPIO_IO29 can be muxed as two FlexIO pins. You can test the FlexIO emulation of I2C without any hardware rework. Figure 3 shows the related parts of the schematic.

Table 3. IOMUX of I2C Pins

IOPAD	Alt1	Alt7
GPIO_IO28	i2c3.SDA	flexio1.FLEXIO[28]
GPIO_IO29	i2c3.SCL	flexio1.FLEXIO[29]

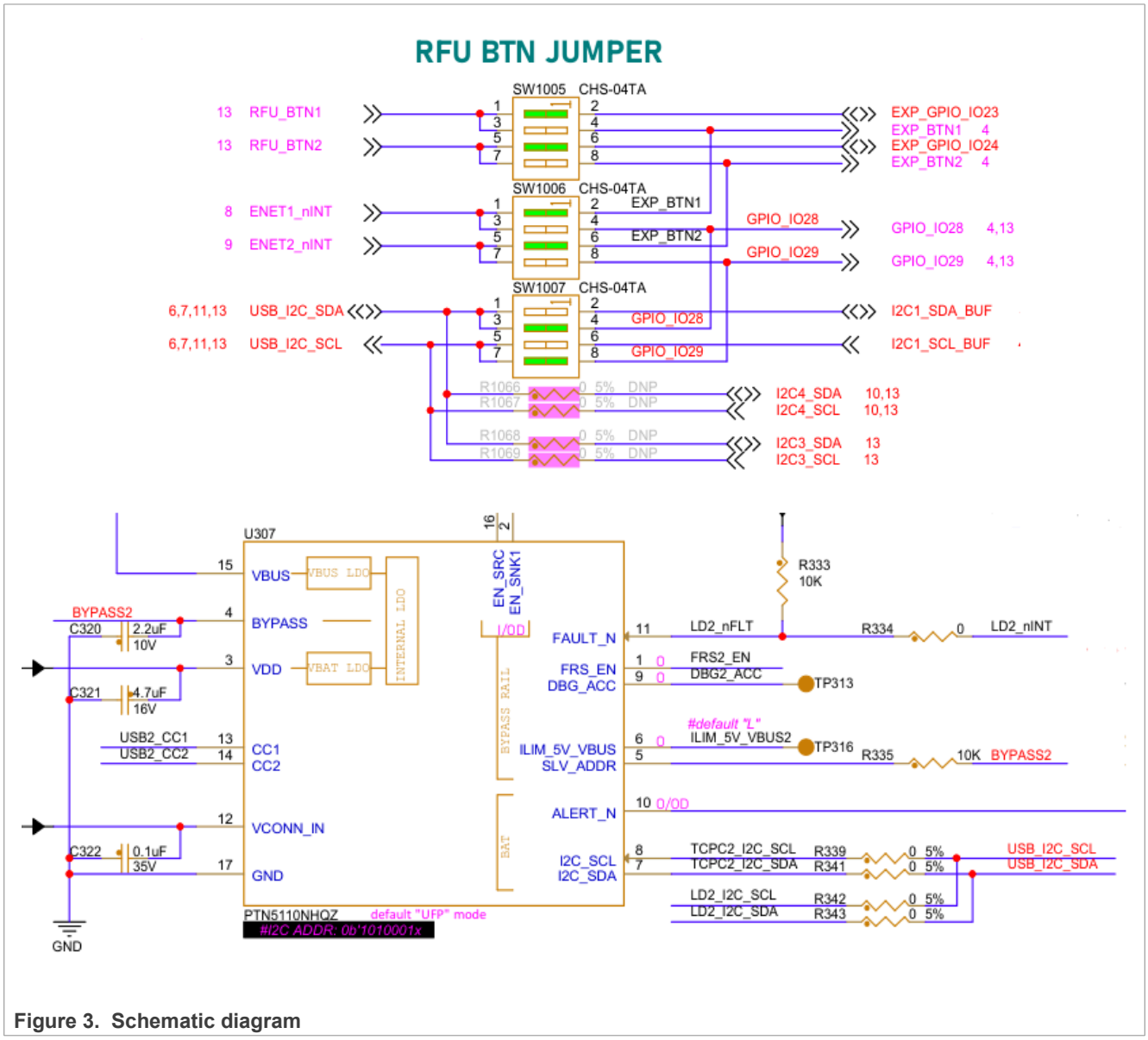

Figure 3. Schematic diagram

3.4 Software implement

This section describes the software implementation as per the following:

- Implementation based on Linux with A55
- Implementation in SDK with M33

3.4.1 Linux with A55

This section describes the software implementation based on Linux L6.1.36_2.1.0. The kernel driver can be divided into two parts. The multifunction driver (MFD) is applied as the FlexIO core driver. FlexIO I2C master driver node is the child node of the core driver node.

```
## kernel driver files
include/linux/mfd/imx-flexio.h
drivers/mfd/imx-flexio.c    //compatible = "nxp,imx-flexio"
drivers/i2c/busses/i2c-flexio.c  //compatible = "nxp,imx-flexio-i2c-master"
```

The dts file below is a merge of the following two files: arch/arm64/boot/dts/freescale/imx93.dtsi and arch/arm64/boot/dts/freescale/imx93-11x11-evk-flexio-i2c.dts

```
## dts
/{
aliases {
i2c8 = &flexio_i2c;
 };
};

&lpi2c3 {
        status = "disabled";
        /delete-node/ tcpc@51;
};

flexio1: flexio@425c0000 {
 #address-cells = <1>;
 #size-cells = <1>;
 compatible = "nxp,imx-flexio";
 reg = <0x425c0000 0x10000>;
 interrupts = <GIC_SPI 53 IRQ_TYPE_LEVEL_HIGH>;
 clocks = <&clk IMX93_CLK_FLEXIO1_GATE>,
   <&clk IMX93_CLK_FLEXIO1_GATE>;
 clock-names = "per", "ipg";
 assigned-clocks = <&clk IMX93_CLK_FLEXIO1_GATE>;
 assigned-clock-parents = <&clk IMX93_CLK_FLEXIO1>;
 assigned-clock-rates = <24000000>;
 status = "okay";

 flexio_i2c: i2c-master {
  #address-cells = <1>;
  #size-cells = <0>;
  compatible = "nxp,imx-flexio-i2c-master";
  clock-frequency = <100000>;
  pinctrl-names = "default", "sleep";
  pinctrl-0 = <&pinctrl_flexio_i2c_master>;
  pinctrl-1 = <&pinctrl_flexio_i2c_master>;
  sda = /bits/ 8 <28>;
  scl = /bits/ 8 <29>;
  status = "okay";
 };
};
```

```
&iomuxc {
 pinctrl_flexio_i2c_master: flexiogrp {
  fsl,pins = <
   MX93_PAD_GPIO_IO29__FLEXIO1_FLEXIO029    0xb9e
   MX93_PAD_GPIO_IO28__FLEXIO1_FLEXIO028    0xb9e
  >;
 };
};
```

In the FlexIO core driver imx-flexio.c, some basic setup functions are implemented. When more interfaces are implemented with FlexIO, more functions can be added to work as common codes.

Table 4. ICore functions of FlexIO

Register	Functions
FlexIO Control Register (CTRL)	`void flexio_sw_reset(void *base, unsigned int reg);void flexio_get_default_ctrl(struct flexio_control *ctrl);void flexio_setup_ctrl(void *base, struct flexio_control *ctrl, unsigned int reg);`
Shifter Control N register (SHIFTCTL0 - SHIFTCTL7)	`void flexio_setup_shiftctl(void *base, struct flexio_shifter_control *ctl, unsigned int reg);`
Shifter Configuration N register (SHIFTCFG0 - SHIFTCFG7)	`void flexio_setup_shiftcfg(void *base, struct flexio_shifter_config *cfg, unsigned int reg);`
Timer Control N register (TIMCTL0 - TIMCTL7)	`void flexio_setup_timerctl(void *base, struct flexio_timer_control *ctl, unsigned int reg);`
Timer Configuration N register (TIMCFG0 - TIMCFG7)	`void flexio_setup_timercfg(void *base, struct flexio_timer_config *cfg, unsigned int reg);`

In the FlexIO I2C master driver, the most configurations in Section 3.2 are done in `imx_flexio_init_hardware()`. The driver behavior is described with i2c_adapter and i2c_algorithm since the I2C subsystem framework is applied.

```
struct i2c_algorithm {
        int (*master_xfer)(struct i2c_adapter *adap, struct i2c_msg *msgs, int
 num);
  ……
};
```

```
static int imx_flexio_i2c_master_xfer(struct i2c_adapter *adap, struct i2c_msg
 msgs[], int num);
--> //one more byte is slave addr, only 1 bit for STOP/Repeated START
 xfer_len = msg->len + 1;
--> //Max bytes is 14 because high 8bit of TIMCMP can only reach 255. 18*N + 1 <
 255, so N <=14
setup_xfer_count(i2c_dev, xfer_len);
 --> //Write or read according to the xfer_msg flag
i2c_master_write or i2c_master_read
```

Then another important function is the interrupt processing function of FlexIO, which is the key part of the driver.

```
static irqreturn_t imx_flexio_i2c_isr(int irq, void *dev_id);
 --> //Responsible for writing date and sending ack signal
  if (shiftstat & TRANSMIT_STAT) {}
 --> //Responsible for reading date and checking ack signal
  if (shiftstat & RECEIVE_STAT) {}
```

The specific process of ISR is complicated. For details, refer to FlexIO I2C driver in BSP release.

Once the FlexIO I2C driver is ready, it can be used as a common I2C driver. The proper I2C device nodes can be added into the device tree as follows.

```
&flexio_i2c {
        ptn5110_2: tcpc@51 {
                compatible = "nxp,ptn5110";
                reg = <0x51>;
                interrupt-parent = <&gpio3>;
                interrupts = <27 IRQ_TYPE_LEVEL_LOW>;
                status = "okay";
    ……
    ……
        };
};
```

3.4.2 BareMetal with M33

This section describes the software implementation in SDK with M33 running. The driver can be divided into two parts, one is the FlexIO core driver and the other is the interface driver for specific cases.

```
## driver files
devices/MIMX9352/drivers/fsl_flexio.h
devices/MIMX9352/drivers/fsl_flexio.c
devices/MIMX9352/drivers/fsl_flexio_i2c_master.h
devices/MIMX9352/drivers/fsl_flexio_i2c_master.c

boards/mcimx93evk/driver_examples/flexio/i2c/read_accel_value_transfer
```

The demo `read_accel_value_transfer` is trying to read the registers of IMU. To be aligned with the case in Linux, the pinmux configuration is changed and the PTN5110 read function is added as follows.

```
IOMUXC_SetPinMux(IOMUXC_PAD_GPIO_IO29__FLEXIO1_FLEXIO29, 1U);
IOMUXC_SetPinConfig(IOMUXC_PAD_GPIO_IO29__FLEXIO1_FLEXIO29, IOMUXC_PAD_DSE(15U)
 |
                         IOMUXC_PAD_FSEL1(2U) | IOMUXC_PAD_OD_MASK);
IOMUXC_SetPinMux(IOMUXC_PAD_GPIO_IO28__FLEXIO1_FLEXIO28, 1U);
IOMUXC_SetPinConfig(IOMUXC_PAD_GPIO_IO28__FLEXIO1_FLEXIO28, IOMUXC_PAD_DSE(15U)
 |
                         IOMUXC_PAD_FSEL1(2U) | IOMUXC_PAD_OD_MASK);
```

```
static bool I2C_example_read_PTN5110_VENDOR_ID(void)
{
    uint16_t vendor_id  = 0x00;
    uint8_t i           = 0;
    uint32_t j          = 0;

    flexio_i2c_master_config_t masterConfig;

    FLEXIO_I2C_MasterGetDefaultConfig(&masterConfig);
    masterConfig.baudRate_Bps = I2C_BAUDRATE;

    if (FLEXIO_I2C_MasterInit(&i2cDev, &masterConfig, FLEXIO_CLOCK_FREQUENCY) !=
 kStatus_Success)
    {
        PRINTF("FlexIO clock frequency exceeded upper range. \r\n");
```

```
        return false;
    }

    FLEXIO_I2C_MasterTransferCreateHandle(&i2cDev, &g_m_handle,
 flexio_i2c_master_callback, NULL);

    flexio_i2c_master_transfer_t masterXfer;
    memset(&masterXfer, 0, sizeof(masterXfer));

    masterXfer.slaveAddress   = 0x51;
    masterXfer.direction      = kFLEXIO_I2C_Read;
    masterXfer.subaddress     = 0x00;
    masterXfer.subaddressSize = 1;
    masterXfer.data           = (uint8_t *)&vendor_id;
    masterXfer.dataSize       = 2;


    for (i = 0; i < 127; i++)
    {
        completionFlag          = false;
        FLEXIO_I2C_MasterTransferNonBlocking(&i2cDev, &g_m_handle, &masterXfer);

        /*  wait for transfer completed. */
        while ((nakFlag == false) && (completionFlag == false))
        {
        }

        if (nakFlag == true)
        {
            nakFlag = false;
            for (j = 0; j < 0x1FFF; j++)
            {
                __NOP();
            }
        }

        if (completionFlag == true)
        {
            PRINTF("The Vendor ID is 0x%x\r\n", vendor_id);
        }
        else
        {
            PRINTF("Failed to Get the Vendor ID\r\n");
        }

        for (j = 0; j < 0xFFF; j++)
        {
            __NOP();
        }
    }

    return 0;
}
```

3.4.3 Zephyr with A55

This section describes the software implementation in Zephyr OS running on A55. The driver can be divided into two parts—one is the FlexIO core driver and the other is the interface driver for specific cases. The FlexIO core driver and I2C drivers of Zephyr are not available in the current release. To add the support of FlexIO I2C

in Zephyr, see the patch file in Attachments. Zephyr has a Linux code style but sometimes works together with the SDK.

```
## driver files
drivers/i2c/i2c_mcux_flexio.h
drivers/i2c/i2c_mcux_flexio.c
drivers/misc/mcux_flexio/mcux_flexio.h
drivers/misc/mcux_flexio/mcux_flexio.c
```

```
## dts:
flexio1: flexio@425c0000 {
compatible = "nxp,imx-flexio";
 #address-cells = <1>;
 #size-cells = <0>;
 reg = <0x425c0000 0x10000>;
 interrupts = <GIC_SPI 53 IRQ_TYPE_LEVEL IRQ_DEFAULT_PRIORITY>;
 interrupt-parent = <&gic>;
 clocks = <&ccm IMX_CCM_FLEXIO1_CLK 0x70 12>;
 status = "disabled";
};

&pinctrl {
 pinmux_flexio1_i2c: pinmux_flexio1_i2c {
  group0 {
   pinmux =
    <&iomuxc1_gpio_io28_flexio_flexio_flexio1_flexio28>, /* SDA */
    <&iomuxc1_gpio_io29_flexio_flexio_flexio1_flexio29>; /* SCL */
   drive-open-drain;
   bias-pull-up;
   slew-rate = "fast";
   drive-strength = "x4";
  };
 };
};

&flexio1 {
 status = "okay";

 flexio1_i2c: flexio1_i2c {
  compatible = "nxp,imx-flexio-i2c-master";
  status = "okay";
  clock-frequency = <I2C_BITRATE_STANDARD>;
  #address-cells = <1>;
  #size-cells = <0>;
  pinctrl-0 = <&pinmux_flexio1_i2c>;
  pinctrl-names = "default";
  sda-pin = <28>;
  scl-pin = <29>;
  shifters = <0 1>;
  timers = <0 1 2>;
 };
};
```

The function `i2c_mcux_flexio_transfer` deals with the process of I2C scan, read, and write. Even if it runs in A55 cores, the drivers in the SDK that are named as hal are performing as the key parts.

```
static int i2c_mcux_flexio_transfer(const struct device *dev, struct i2c_msg
 *msgs,
```

```
          uint8_t num_msgs, uint16_t addr)
{
 ……

 /* Iterate over all the messages */
 for (int i = 0; i < num_msgs; i++) {

  ……

  transfer.slaveAddress = addr;
  transfer.direction = (msgs->flags & I2C_MSG_READ)
   ? kFLEXIO_I2C_Read : kFLEXIO_I2C_Write;
  transfer.subaddress = 0;
  transfer.subaddressSize = 0;
  transfer.data = msgs->buf;
  transfer.dataSize = msgs->len;

  /* Start the transfer */
  status = FLEXIO_I2C_MasterTransferNonBlocking(config->flexio_i2c,
    &data->handle, &transfer);

  ……

  if (msgs->len == 0) {
   if (kFLEXIO_I2C_ReceiveNakFlag & FLEXIO_I2C_MasterGetStatusFlags(config-
>flexio_i2c)) {
    FLEXIO_I2C_MasterTransferAbort(config->flexio_i2c, &data->handle);
    ret = -EIO;
    break;
   }
  }
  /* Move to the next message */
  msgs++;
 }

 ……

}
```

3.5 Running the test

Consider PTN5110 as an example, which is mounted on LPI2C3 by default. The same pins are configured to be FlexIO pins with no hardware rework. Table 5 describes the test that keeps reading the VENDOR_ID and should get the default value as “0x1FC9”.

Table 5. Test detail

Group	Offset	Name	Type	Default value	Bit field	Description
Identification registers	00h	VENDOR_ID	read-only word	0x1FC9	15:0	Vendor ID A unique 16-bit unsigned integer. Assigned by the USB-IF to the vendor.

3.5.1 Linux with A55

Since the FlexIO I2C driver conforms to the I2C subsystem software framework, it can be tested as a common I2C master or slave device. To get the VENDOR_ID of PTN5110, use the `i2cdetect/i2ctransfer` command in Linux as shown in Figure 4.

```
root@imx93evk:~# i2cdetect -l
i2c-0   i2c             44340000.i2c                            I2C adapter
i2c-1   i2c             44350000.i2c                            I2C adapter
i2c-8   i2c             425c0000.flexio:i2c-master              I2C adapter
root@imx93evk:~#
root@imx93evk:~# i2cdetect -y 8
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 UU 52 53 -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- 71 -- 73 -- -- -- --
root@imx93evk:~#
root@imx93evk:~# i2ctransfer -f -y 8 w1@0x51 0x00 r2
Error: Sending messages failed: Input/output error
root@imx93evk:~# i2ctransfer -f -y 8 w1@0x51 0x00 r2
0xff 0xff
root@imx93evk:~# i2ctransfer -f -y 8 w1@0x51 0x00 r2
0xc9 0x1f
root@imx93evk:~# i2ctransfer -f -y 8 w1@0x51 0x00 r2
Error: Sending messages failed: Input/output error
root@imx93evk:~# i2ctransfer -f -y 8 w1@0x51 0x00 r2
0xff 0xff
root@imx93evk:~# i2ctransfer -f -y 8 w1@0x51 0x00 r2
0xc9 0x1f
```

Figure 4. VENDOR_ID of PTN5110 in Linux with A55

It can be found that the register VENDOR_ID is already received. However, if the command is called frequently, communication failures may occasionally occur. This issue is caused by untimely interrupt response and handling. The same case is also tested in a Linux-RT environment, but gets a little better result. The error rate increases significantly with the increase of CPU loading.

3.5.2 BareMetal on M33

FlexIO has been widely used in MCU before, and its real time and emulation stability are reliable. The test in this section runs on the M33 core of i.MX 93.

```
FlexIO I2C example read PTN5110 VENDOR_ID
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
The Vendor ID is 0x1fc9
```

Figure 5. VENDOR_ID of PTN5110 in BareMetal with M33

3.5.3 Zephyr on A55

Similar to Linux, Zephyr can use commands such as `i2c scan/read/write` to operate I2C devices conveniently. No errors are found during the test process.

```
zephyr:~$ i2c scan flexio1_i2c
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 51 52 53 -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- 71 -- 73 -- -- -- --
```

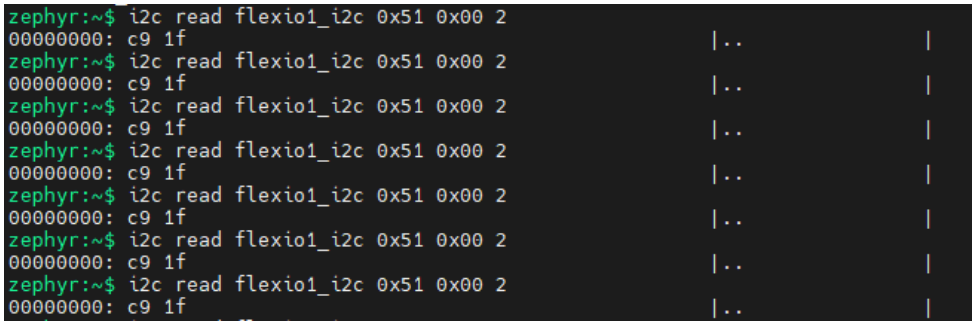

Figure 6. VENDOR_ID of PTN5110 in Zephyr with A55

4 Conclusion

FlexIO can be emulated as the I2C master in different operating systems. However, it is not recommended to use FlexIO I2C in a Linux environment directly. Some check mechanisms should be applied to make sure the read or write operation is executed correctly in the Linux environment. Linux is not an RTOS, which means it cannot guarantee the interrupt latency or interrupt responding time. Linux may generate big interrupt latency while FlexIO IP can only tolerate small interrupt latency.

In a BareMetal or RTOS environment, such as Zephyr, FlexIO can work normally as expected.

Another optional plan is planned in the future. A core could request M core to perform FlexIO I2C communication through RPMsg, making full use of the real-time capabilities of M core.

5 References

Table 6 lists the resources that can be referred for more information.

Table 6. References

Resource	Link/how to access
i.MX 93 Applications Processor Reference Manual	IMX93RM
Emulating I2C Bus Master by using FlexIO Application Note	AN5133

6 Acronym

Table 7 lists and defines the acronyms used in this document.

Table 7. Acronyms

Term	Definition
FlexIO	Flexible input/output
I2C	Inter-integrated circuit
I2S	Inter-IC sound
OSI	Open systems interconnection
PHY	Physical interface of the OSI model
PWM	Pulse width modulation
RTOS	Real-time operating system

Table 7. Acronyms...*continued*

Term	Definition
SDK	Software development kit
SPI	Serial peripheral interface
UART	Universal asynchronous receiver transmitter

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 8 summarizes revisions to this document.

Table 8. Revision history

Revision number	Release date	Description
1	15 November 2023	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

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.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. 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.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

For more information, please visit: https://www.nxp.com

Date of release: 15 November 2023
Document identifier: AN14110