AN14776

MCU8080 LCD Interface Implemented by FlexIO on MCXE31B

Rev. 1.0 — 22 September 2025 Application note

Document information

Information	Content
Keywords	AN14776, 8080 bus, FlexIO, LCD module
Abstract	This application implements a demo that emulates an 8080 bus by using FlexIO. Several driver functions are implemented for writing and reading to drive a TFT LCD module.

1 Introduction

The FlexIO module is a highly configurable peripheral, which allows users to implement a variety of functions. Depending on the module version, it can do the following:

- Emulate serial communication protocols, such as UART, SPI, I2C, I2S, and so on.
- Emulate parallel communication protocols, such as camera interface, Motorola 68 K bus, Intel 8080 bus, and so on.
- Generate the PWM waveform.
- Implement logic functions.
- Implement state machine functions.

Graphic TFT LCD modules are widely used in embedded applications that require GUI functions. The 8080 parallel bus is a common interface of a TFT LCD module.

This document describes how to use the FlexIO module to emulate the 8080 parallel bus and to drive a graphical TFT LCD with the 8080 bus interface.

The application is based on the recently launched MCXE31B MCU. The MCXE 31 product series further extends the highly-scalable portfolio of the MCX E family in the commercial and industrial industries with the Arm Cortex-M7 core at a higher frequency (up to 160 MHz), more memory, SIL-2 rating, and advanced security module. Scalable memory footprints offer up to 4 MB of flash memory and up to 512 kB of SRAM.

2 FlexIO overview

FlexIO (Flexible Input/Output) is a highly flexible peripheral module in the MCXE31B MCU that can emulate various communication protocols and interfaces.

3 Features and module block diagram

The FlexIO module version of the MCXE31B provides the following key features:

- An array of 32-bit shift registers (also known as shifters) with transmit, receive, and data match modes, a double buffered structure for continuous transfer, and a concatenation mechanism to support large transfer sizes.
- Highly flexible 16-bit timers with support for a variety of internal or external trigger, reset, enable, and disable conditions.
- An automatic start/stop bit generation and check.
- 1, 2, 4, 8, 16, or 32 multibit shift widths for parallel interface support.
- An interrupt, DMA, or polled transmit/receive operation.
- Programmable baud rates supporting asynchronous operations during stop modes.
- A programmable logic mode for integrating external digital logic functions on-chip or combining pin/shifter/timer functions to generate complex outputs.
- A programmable state machine for offloading basic system control functions from the CPU.

Figure 1 shows a high-level overview of the module.

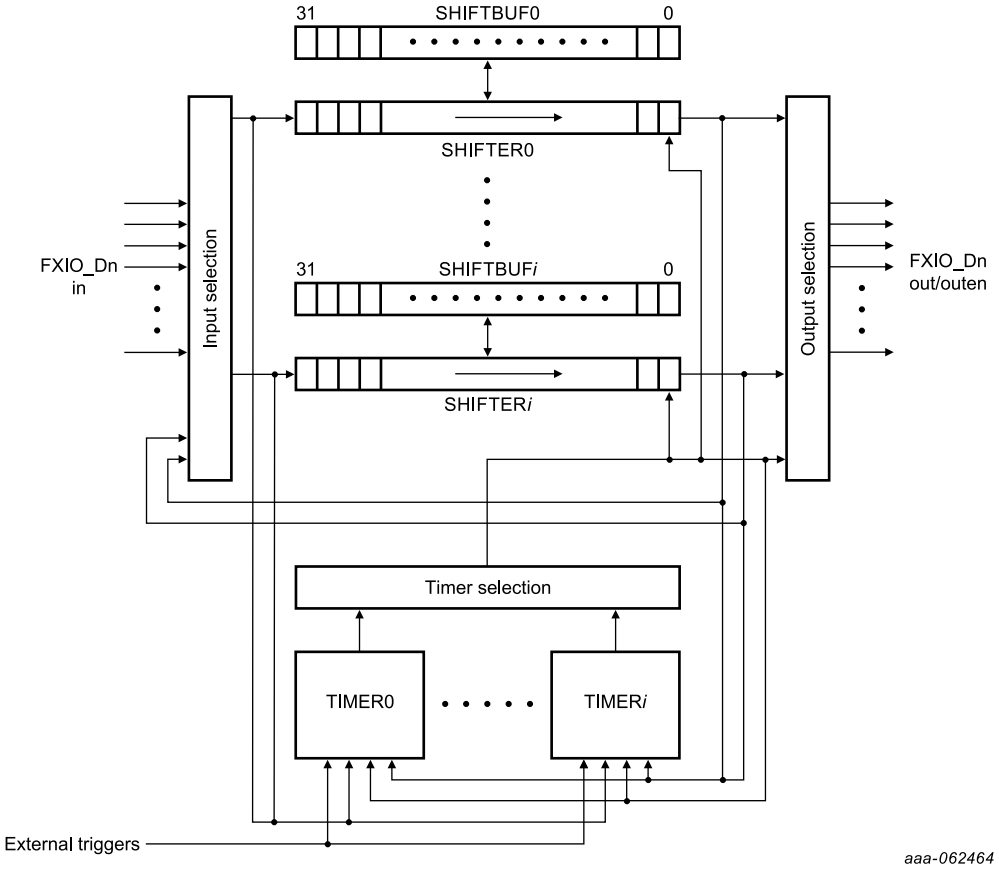


Figure 1. FlexIO block diagram

3.1 Internal logic connection

To satisfy various requirements, the internal logic connections are very flexible and intricate. Some of the capabilities are as follows:

- Any pin can be assigned to a shifter for input or output.
- Any timer can be assigned to a shifter for shift control.
- Any pin can be assigned to a timer for timer output or input.
- Timer triggers can originate from shifter flags, pins, or outside the FlexIO module.
- A shift can be on a rising or a falling edge of the shift clock.
- A pin direction and polarity is configurable.
- The trigger polarity is configurable.
- A timer's enable, disable, decrement, and reset conditions may originate from the trigger, pin, adjacent timer, and so on.

For detailed information, see the appropriate MCU reference manual.

4 Shifters and timers

The above description shows that FlexIO hardware resources consist of shifters, timers, and pins. The amount of these resources for a given MCU can be read from the PARAM register. For example, the FlexIO module of MCXE31 has 8 shifters, 8 timers, and 32 pins.

Transmit and receive are 2 basic modes of the shifters. When a shifter is in a transmit mode, it loads data from its buffer register and then shifts the data out to its assigned pin/pins. When a shifter is in a receive mode, it shifts data in from its assigned pin/pins and then stores the data into its buffer register. Loading, storing, and shifting operations are controlled by the shifter's assigned timer.

The timers can also be configured in different operating modes as needed, including a dual 8-bit counters baud/bit mode, a dual 8-bit counters PWM mode, and a single 16-bit counter mode. The dual 8-bit counters baud/bit mode usually implements a data transmitter. In this mode, the lower eight bits of the 16-bit timer divide the module clock source to generate the desired baud rate, and the higher 8 bits count the shift bits of a frame. After it is enabled, the timer loads the initial value from its compare register and starts to decrement the count. When the lower eight bits decrement to 0, the timer's shift clock and its output signal are toggled to generate a rising or a falling edge. The higher 8 bits' decrement counts by 1. The shift clock drives the shifter. The timer output signal usually drives a pin for the clock output, such as the SCK of an SPI master and the WR of the 8080 bus. After that, the lower 8 bits reload the initial value to start another decrement cycle. 2 decrement cycles make up a shift cycle, which drives the shifter to shift 1 beat. When the whole 16 bits' decrement counts to 0, all data bits in the shifts are shifted out. Then, the timer is disabled before another transfer frame.

5 General configurations and operations

The FlexIO can emulate various communication protocols. However, to emulate a dedicated peripheral and to handle the transmit and receive process, the FlexIO must be configured using software. To implement a master transmitter, a shifter is configured in a transmit mode and the assigned timer is configured in a dual 8-bit counters baud/bit mode. The timer decrement clock originates from the module clock. The timer trigger originates from the shifter's flag with reversed polarity. Filling the shifter's buffer via the polling/interrupt/DMA clears the shifter flag, which enables the timer to start the decrement count. The decrement of the timer drives the shifter to shift data out and generates the clock output signal.

To implement a receiver, a shifter is configured in a receive mode. The timer is configured in a dual 8-bit counters baud/bit mode for a synchronous master receiver, such as the 8080 bus reading implementation. This timer mode is also used for asynchronous receiver, such as the UART receiver. The receive process is similar to that of a master transmitter but the data is shifted in to the shifter rather than shifted out. For the synchronous receiver, the assigned timer is configured in a single 16-bit counter mode, such as the SPI slave receiver. The decrement clock originates from the pin input, such as the SPI SCK signal. The timer trigger originates from another pin, such as the SPI CS signal. The master device enables the timer and controls the decrement via pins. Similarly, the decrement of the timer drives the shifter to shift in the data.

6 FlexIO parallel transfer

MCXE31B supports both serial and parallel transfers. Data is always shifted from the MSB to the LSB in a shifter. In the serial transfer mode, the data is shifted out bit by bit from the LSB (Bit 0) and shifted in bit by bit from the MSB (Bit 31). This process is shown in Figure 2-a. In a parallel transfer mode, the data is shifted out from the *n* LSBs of a shifter and shifted in from the *n* MSBs, where *n* is the parallel bus width. Figure 2-b shows the use case of n = 8.

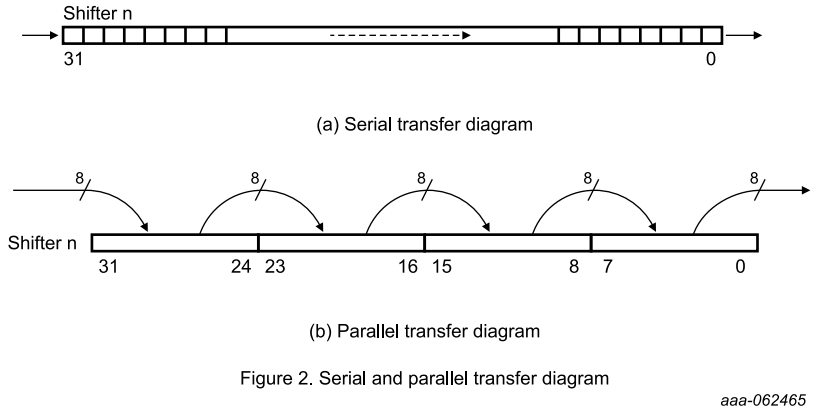


Figure 2. Serial and parallel transfer diagrams

The parallel transfer mode is as follows:

- Data is shifted *n* bits on each shift clock, where *n* is the configured bus width.
- 4, 8, 16, or 32 bus width is supported.
- Combine multiple shifters together for concatenation to support large transfer sizes and use the DMA method to access the shifter buffer registers for high-speed transfers and low-power operations. Figure 2 shows the shifter concatenation diagram where additional shifters work as FIFOs.

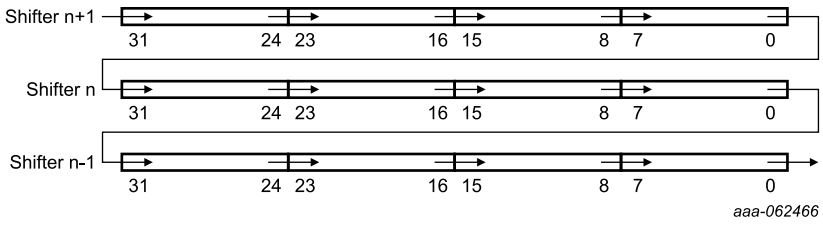


Figure 3. Shifter concatenation diagram

- Only specific shifters (SHIFTER0 and SHIFTER4) support parallel outputting to FlexIO pins. However, all shifters, except SHIFTER0, support outputting to the adjacent low-order shifters.
- Similarly, only specific shifters (SHIFTER3 and SHIFTER7) support parallel inputting from FlexIO pins. However, all shifters, except SHIFTER7, support inputting from the adjacent high-order shifters.
- Any FlexIO pin can be a parallel output/input pin. However, the pin indexes must be successive for a specific usage, such as pin0 to pin7, or pin1 to pin8, and so on for a 8-bit bus width.

7 Parallel bus sequence for LCD modules

The 8080 parallel interface used for LCD modules consists of 8 or 16 bidirectional data lines (data bus), 1 chip-select line (CS), 1 writing-latch line (WR), 1 reading-latch line (RD), and 1 data/command-select line (RS). The CS, WR, RD, and RS are all low-level active. A low level of the CS selects the slave device. The rising edge of the WR line is a data write latch signal (clock). The rising edge of the RD line is a data read latch signal (clock). The RD must be high-level when a writing sequence is in progress. Similarly, the WR must be high-level when a reading sequence is in progress. The RS is a data/command select signal. A low level of the RS indicates the command (or address) transfers. A high level of the RS indicates data transfers. The RS is also known as DC or ALE (Address Latch Enable). At the beginning of a writing/reading transfer, a command/address-writing sequence specifies the target address. The data transfers can be 1 or more beats. Figure 4 shows the writing sequence. A data transfer occurs during the writing sequence under a 0-beat command type.

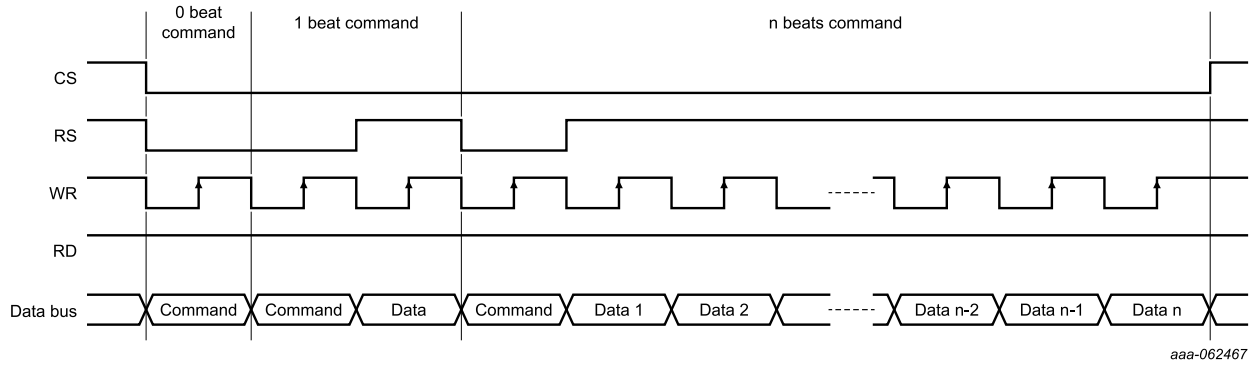


Figure 4. 8080 bus writing sequence diagram

Figure 5 shows the reading sequence. A dummy reading beat can occur between the command-writing beat and the first data-reading beat, depending on the bus slave.

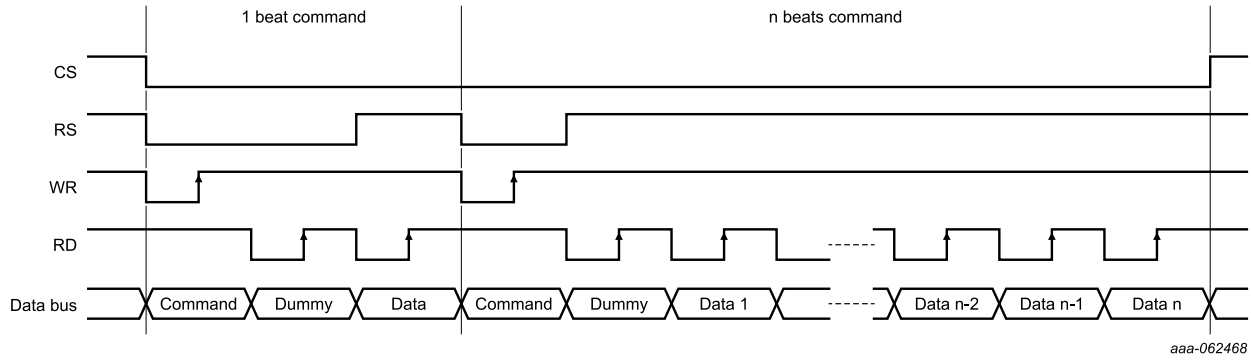


Figure 5. 8080 bus reading sequence diagram

8 Emulating 8080 bus and driving LCD module

This section describes how to use the FlexIO and emulate the 8080 bus interface to drive a TFT LCD module.

9 Configuring FlexIO to emulate 8080 bus

You can configure the FlexIO to emulate the 8080 bus in different ways, such as different bus width, baud rate, the number of the concatenated shifters, specific shifters, pins, and timer to use, and so on. Set up the proper configurations based on the application requirements.

In this application, 8-bit and 16-bit bus widths are implemented. Because module configurations and software drivers are very similar between the 8-bit and 16-bit bus implementations, the following sections only describe the 8-bit implementation. Both writing and reading functions have been implemented. The writing function is implemented by configuring the shifters in a transmit mode. The reading function is implemented by configuring the shifters in a receive mode. Both 1-beat transfer and multibeat transfer drivers have been implemented. The 1-beat transfers transfer small-size data, such as configuring an LCD driver IC's registers. The shifter concatenation is not used for 1-beat transfers because only 1 shifter is used. 1 transfer sequence requires the timer to generate only 1 shift clock. 8 bits are transferred in/out simultaneously. In this application, a polling method accesses the shifter buffers for the 1-beat transfers.

The multibeat transfers support large transfer sizes, such as transferring frame data to an LCD module. 1 transfer sequence requires the timer to generate multiple shift clocks. The number of beats per 1 transfer sequence is related to the number of the concatenated shifters and the bus width. All shifters are 32-bit. 1

shifter supports (at most) a 4-beat transfer for the 8-bit bus width. 2 shifters support (at most) 8 beats and so on. In this application, all 8 shifters are used and 32 beats are supported for the 8-bit bus width. The DMA method accesses the shifter buffers for the multibeat transfers in this application.

In a real application, writing and reading do not occur at the same time. 1-beat and multibeat transfers never operate simultaneously. The shifters and pins are shared between different functions. Reconfigure the FlexIO module at each switching of functions.

There are 4 implementations for writing and reading (1-beat and multibeat transfers):

1. 1-beat writing
2. 1-beat reading
3. Multibeat writing
4. Multibeat reading

In this application, the 1-beat writing is widely used, such as configuring the LCD driver IC's registers, transferring smaller frame data, and transferring the command beat in all of the 4 implementations, and so on. The multibeat writing mainly transfers larger frame data to an LCD. The 1-beat reading and multibeat reading are not used in this application.

Because 1-beat transfers are a simpler version of the multibeat transfers, but multibeat writing and multibeat reading are briefly described in the following sections.

9.1 Multibeat writing

Figure 6 shows the FlexIO module configuration for multibeat writing transfers.

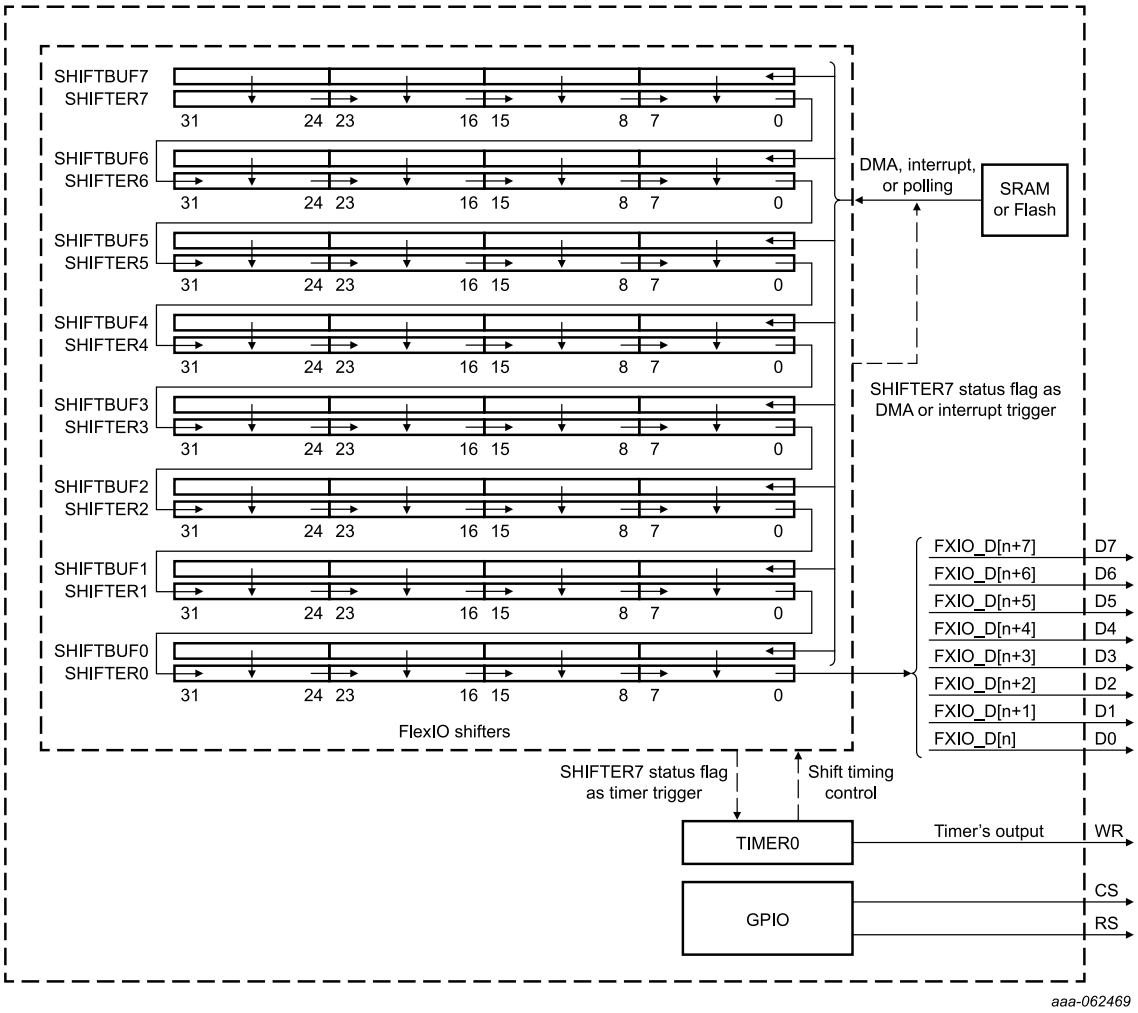


Figure 6. Multibeat writing configuration

In the configuration, all 8 shifters are concatenated together. TIMER0 generates a shift clock and a WR signal. D0~D7 and WR originate from FlexIO pins. Additional GPIO pins drive the CS and RS signals. The SHIFTER7 status flag triggers TIMER0 and generates the DMA request.

The following steps describe multibeat writing transfers using the DMA method:

1. Configure DMA, FlexIO module, and GPIO. Enable the DMA request of the SHIFTER7 status flag.
2. The DMA request responds immediately after the enablement of the request. The DMA copies data from SRAM or flash to the SHIFTBUF0~SHIFTBUF7 shifter buffers. In total, 32 bytes are copied per 1 DMA request.
   Note: *The addresses of SHIFTBUF0~SHIFTBUF7 are successive.*
3. Filling the shifter buffers clears the shifter status flags, which enables the selected TIMER0.
4. TIMER0 signals a loading event. Data are loaded from the SHIFTBUF0~SHIFTBUF7 shifter buffers to the SHIFTER0~SHIFTER7 shifters.
5. The loading event empties the shifter buffers, which sets the shifter status flags and triggers another DMA request. The DMA fills SHIFTBUF0~SHIFTBUF7 with new data and the shifter status flags are cleared again.
6. TIMER0 starts to decrement count after the loading event. It generates the timer shift clock along with the decrement to control the shifters shifting data out and generates the timer output to drive the WR signal.

7. After configuring the 32 shift clocks, TIMER0 decrement counts to 0 and a compare event occurs. Then, TIMER0 is disabled for a short time.
8. Because the shifter buffers contain valid data and the SHIFTER7 status flag is 0 at this point, TIMER0 is enabled again. Then, steps 3-8 repeat.
9. After all data has been copied to the shifter buffers, the DMA completes the major loop. No more data is copied and TIMER0 is not triggered after the last compare event. A DMA interrupt is generated to indicate the completion of the major loop.
10. If the total transfer size is not divisible by 32, the additional bytes are transferred in a 1-beat mode using a polling method.

The CS is pulled low before the transfer sequence by software and pulled up after the transfer sequence. The RS is pulled up before the command transmission and pulled up again after that.

9.2 Multibeat reading

Figure 7 shows the FlexIO module configuration for multibeat reading transfers.

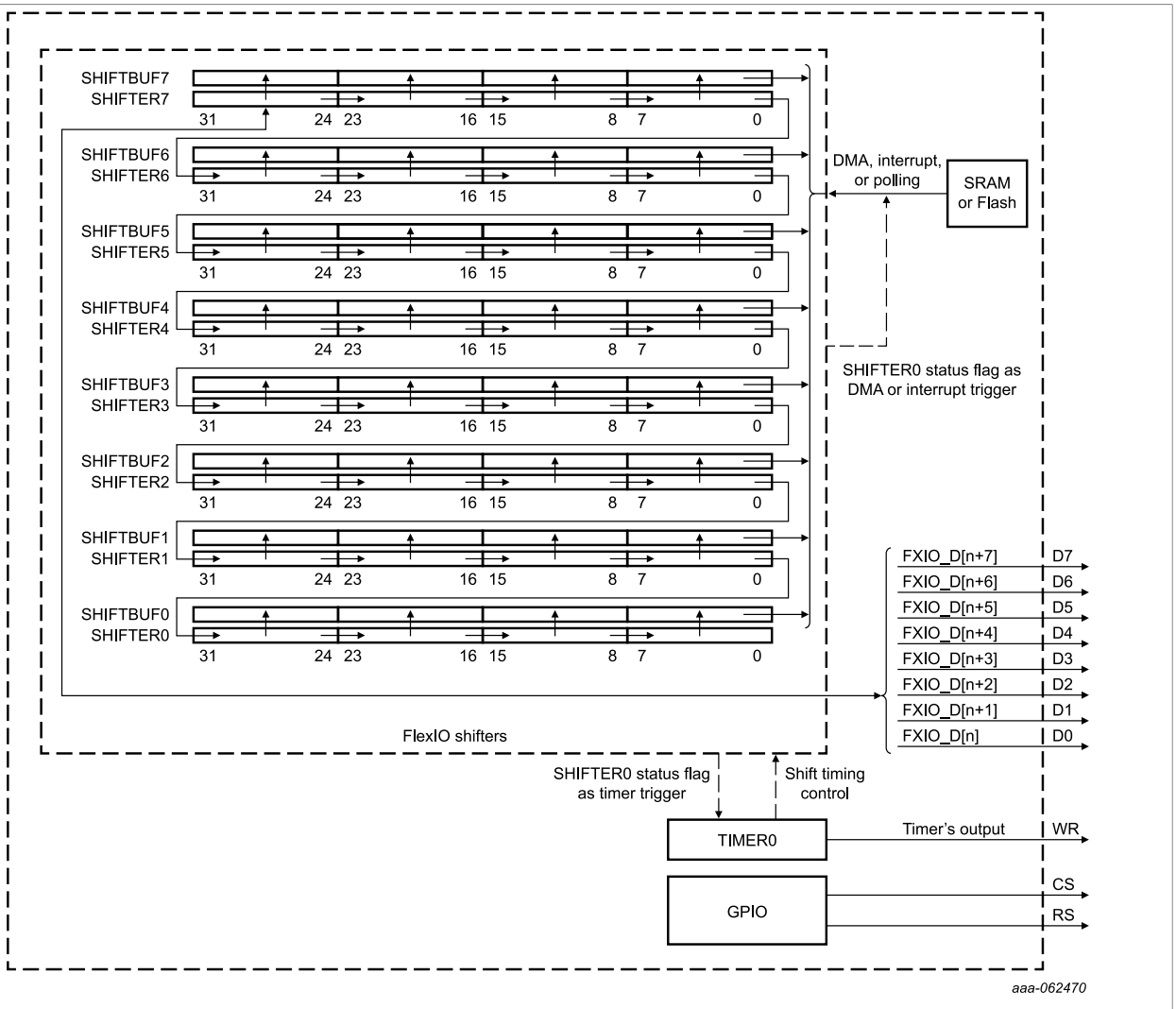


Figure 7. Multibeat reading configuration

In the configuration, all 8 shifters are concatenated together. TIMER0 generates the shift clock and the RD signal. The D0-D7 and the RD are based on FlexIO pins. Additional GPIO pins are used to drive the CS and the RS signal. SHIFTER0 status flag triggers TIMER0 and generates the DMA request.

The following items describe the process of multibeat reading transfers that use the DMA method:

1. Configure DMA, FlexIO module, and GPIO. Enable the DMA request of the SHIFTER0 status flag.
2. TIMER0 is enabled immediately after the configuration and starts to decrement the count. It generates the timer shift clock along with the decrement to control the shifters shifting data in and generates the timer output to drive the RD signal.
3. TIMER0 decrement counts to 0 and a compare event occurs after 32 shift clocks. Then, TIMER0 is disabled.
4. A storing event is signaled by TIMER0 after the compare event. Data is stored from the SHIFTER0-SHIFTER7 shifters to the SHIFTBUF0-SHIFTBUF7 shifter buffers.
5. The storing event fills up the shifter buffers, which sets the shifter status flags and triggers a DMA request.
6. The DMA request is responded. The DMA copies data from the SHIFTBUF0-SHIFTBUF7 shifter buffers to the SRAM. 32 bytes are copied per 1 DMA request.
7. The DMA reading operation clears the shifter flags, which enables TIMER0 again. Then, steps 2-7 repeat.
8. After the major loop completes, TIMER0 is triggered once more to shift in the last 32 bytes. A DMA interrupt is generated to indicate the completion of the DMA major loop.
   Note: *The DMA major loop size does not include the last minor loop to avoid enabling TIMER0 after completing all transfers.*
9. If the total transfer size is not divisible by 32, the additional bytes are transferred in a 1-beat mode using a polling method.

The CS is pulled low before the transfer sequence by software and pulled up after the transfer sequence. The RS is pulled up before the command transmission and pulled up again after that.

10 Driver functions

The SDK package of FRDM-MCXE31B provides drivers and examples for driving the 8080 LCD.

The example file is located in boards\frdmmcxe31b\driver_examples\flexio\mculcd. There are 3 examples: DMA mode, interrupt mode, and polling mode (respectively). This application note introduces the functionality of the FlexIO driving the 8080 LCD interface through a polling mode.

10.1 FlexIO API driver functions

1. status_t FLEXIO_MCULCD_Init(FLEXIO_MCULCD_Type *base, flexio_mculcd_config_t *config, uint32_t srcClock_Hz)
   - Function: Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO MCULCD hardware, and configures the FlexIO MCULCD with the FlexIO MCULCD configuration
   - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
   - Parameter: config – Pointer to the flexio_mculcd_config_t structure
   - Parameter: srcClock_Hz – FlexIO source clock in Hz
   - Return: kStatus_Success for initialization success, kStatus_InvalidArgument for initialization failed due to an invalid argument
     Note: *The configuration structure can be filled by the user, or set with default values by FLEXIO_MCULCD_GetDefaultConfig.*
2. void FLEXIO_MCULCD_GetDefaultConfig(flexio_mculcd_config_t *config)
   - Function: Gets the default configuration to configure the FlexIO MCULCD

- Parameter: config – Pointer to the flexio_mculcd_config_t structure
- Default configuration values:
  - config->enable = true
  - config->enableInDoze = false
  - config->enableInDebug = true
  - config->enableFastAccess = true
  - config->baudRate_Bps = 96000000U (96 MHz)

3. status_t FLEXIO_MCULCD_SetBaudRate(FLEXIO_MCULCD_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
   - Function: Sets the desired baud rate
   - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
   - Parameter: baudRate_Bps – Desired baud rate in bit-per-second for all data lines combined
   - Parameter: srcClock_Hz – FLEXIO clock frequency in Hz
   - Return: kStatus_Success for set successfully, kStatus_InvalidArgument for could not set the baud rate
4. void FLEXIO_MCULCD_SetSingleBeatWriteConfig(FLEXIO_MCULCD_Type *base)
   - Function: Configures the FlexIO for a single-beat write mode, setting up the corresponding timer and shifter configurations
   - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
   - Usage: Used to configure hardware settings for the 8080 bus single data write operations
5. void FLEXIO_MCULCD_SetSingleBeatReadConfig(FLEXIO_MCULCD_Type *base)
   - Function: Configures FlexIO for a single-beat read mode, setting up the corresponding timer and shifter configurations
   - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
   - Usage: Used to configure hardware settings for the 8080 bus single data read operations
6. void FLEXIO_MCULCD_SetMultiBeatsWriteConfig(FLEXIO_MCULCD_Type *base)
   - Function: Configures the FlexIO for a multibeat write mode, suitable for the DMA transfer of large amounts of data
   - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
   - Usage: Used to configure continuous multiple data write operations, typically used with the DMA to improve transfer efficiency
7. void FLEXIO_MCULCD_SetMultiBeatsReadConfig(FLEXIO_MCULCD_Type *base)
   - Function: Configures the FlexIO for a multibeat read mode, suitable for the DMA transfer of large amounts of data
   - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
   - Usage: Used to configure continuous multiple data read operations, typically used with the DMA to improve transfer efficiency
8. void FLEXIO_MCULCD_WriteCommandBlocking(FLEXIO_MCULCD_Type *base, uint32_t command)
   - Function: Writes a command to the LCD in a blocking manner, using the polling method to wait for transfer completion
   - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
   - Parameter: command – Command value to be written
   - Feature: This function blocks until the command transfer is complete
9. void FLEXIO_MCULCD_WriteDataArrayBlocking(FLEXIO_MCULCD_Type *base, const void *data, size_t size)
   - Function: Writes a data array to the LCD in a blocking manner, using the polling method
   - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
   - Parameter: data – Pointer to the data array to be written (const void*)
   - Parameter: size – Size of the data array (bytes)

- Application: Suitable for transferring image data or large amounts of parameter data

10. void FLEXIO_MCULCD_ReadDataArrayBlocking(FLEXIO_MCULCD_Type *base, void *data, size_t size)
    - Function: Reads a data array from the LCD in a blocking manner, using the polling method
    - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
    - Parameter: data – Pointer to the buffer to store the read data
    - Parameter: size – Size of the data to be read (bytes)
    - Application: Suitable for reading the LCD status or image data
11. void FLEXIO_MCULCD_WriteSameValueBlocking(FLEXIO_MCULCD_Type *base, uint32_t value, size_t size)
    - Function: Repeatedly writes the same value to the LCD in a blocking manner, typically used for solid color filling
    - Parameter: base – Pointer to the FLEXIO_MCULCD_Type structure
    - Parameter: value – Value to be repeatedly written
    - Parameter: size – Number of times to repeat the writing
    - Application: Particularly suitable for LCD screen clearing or filling it with a single color

11 Hardware platforms

The FRDM-MCXE31B board is a design and evaluation platform based on the NXP MCXE31B MCU. The NXP MCXE31B MCU is based on an Arm Cortex-M7 core, running at speeds of up to 160 MHz with a 2.97 V - 5.5 V supply.

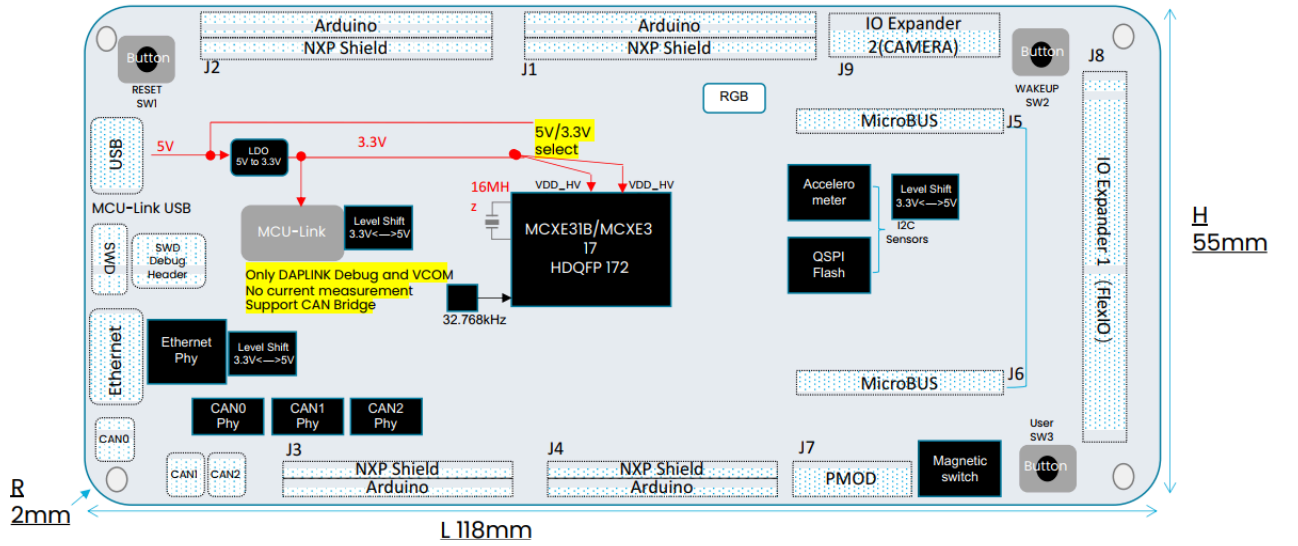


Figure 8. FRDM-MCXE31B board block diagram

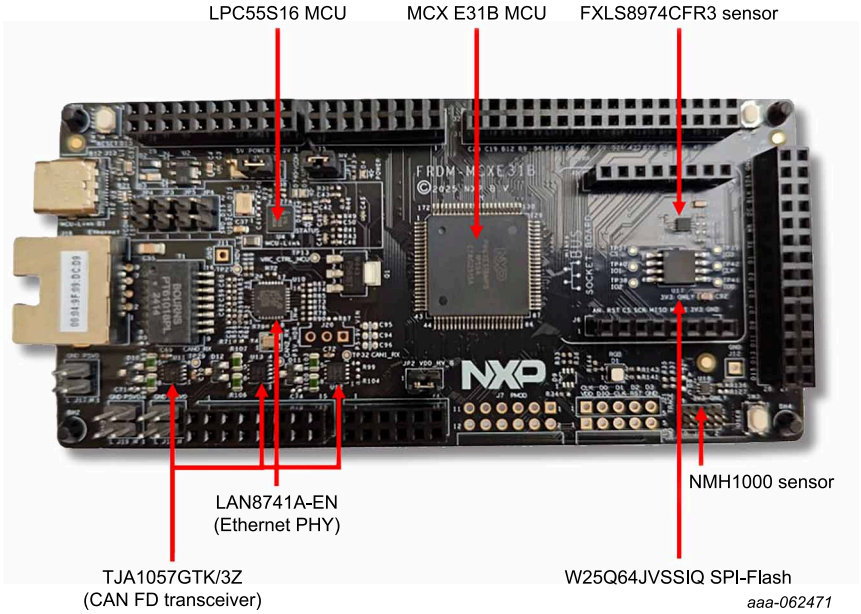


Figure 9. Top view of the FRDM-MCXE31B board

The LCD-PAR-S035 is a 3.5” 480x320 IPS TFT LCD module with a wide viewing angle and 5-point capacitive touch functionality. The LCD module can be controlled either through an SPI or a parallel (8/16bit) 8080/6800 and it is compatible with the existing evaluation kits with a PMOD connector as well as selected FRDM-X evaluation kits with a parallel LCD connector.

Figure 10. LCD-PAR-S035

The system block diagram of this application demo is shown in Figure 11.

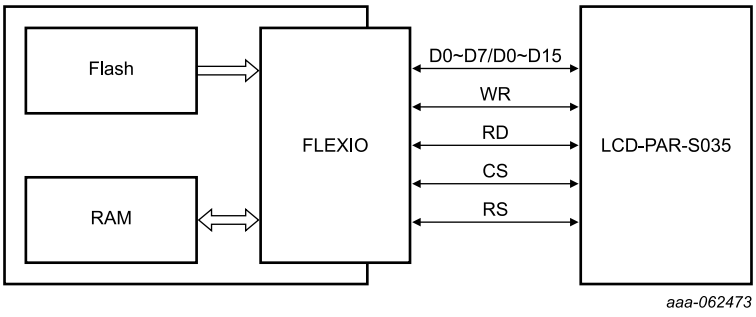


Figure 11. System block diagram

12 FlexIO configurations and hardware connections

This section provides detailed configurations of the FlexIO registers and hardware connections between the FRDM-MCXE31B and the LCD module.

12.1 FlexIO configurations

Table 1 to Table 4 provide primary register configurations of the FlexIO for the 4 operating modes.

Table 1. 1-beat writing

Register	Value	Comment
SHIFTCFG0	0x0007_0100	Configure shifter stop bit disabled, start bit disabled and loading data on enabled, 8-bit parallel width.
SHIFTCTL0	0x0003_0002	Configure shift clock from TIMER0, shift on posedge of shift clock. Configure shifter's pin as output, pin index starting from 0, and pin polarity active high. Configure shifter mode as transmit.
TIMCMP0	0x0000_0101	Set TIMCMP [15:8] =number of beats x 2 - 1 = 1 x 2 - 1 = 1 for 1 beat. Set TIMCMP [7:0] =baud rate divider / 2 - 1 = 4 / 2 - 1 = 1.
TIMCFG0	0x0000_2200	Configure timer output logic 1 when enabled and not affected by reset, decrement on FlexIO clock, shift clock equals timer output, never reset, disabled on timer compare, enabled on trigger high, stop bit disabled, and start bit disabled.
TIMCTL0	0x01C3_1081	Configure SHIFT0 status flag as timer trigger, and trigger polarity active low. Configure timer's pin as output, pin index as 16 (WR), pin polarity active low. Configure timer mode as 8-bit counters baud/bit.

Table 2. 1-beat reading

Register	Value	Comment
SHIFTCFG7	0x0007_0000	Configure shifter input from pin, stop bit disabled, start bit disabled and loading data on enabled, 8-bit parallel width.
SHIFTCTL7	0x0080_0001	Configure shift clock from TIMER0, shift on negedge of shift clock. Configure shifter's pin as input, pin index starting from 0, and pin polarity active high. Configure shifter mode as receive.
TIMCMP0	0x0000_0101	Set TIMCMP [15:8] =number of beats x 2 - 1 = 1 x 2 - 1 = 1 for 1 beat. Set TIMCMP [7:0] =baud rate divider / 2 - 1 = 4 / 2 - 1 = 1.
TIMCFG0	0x0000_2220	Configure timer output logic 1 when enabled and not affected by reset, decrement on FlexIO clock, shift clock equals timer output, never reset, disabled on timer compare, enabled on trigger high, stop bit enabled on timer disabled (for internal signal synchronization to disable timer in time), and start bit disabled.
TIMCTL0	0x1DC3_1181	Configure SHIFT7 status flag as timer trigger, and trigger polarity active low. Configure timer's pin as output, pin index as 17 (RD), pin polarity active low. Configure timer mode as 8-bit counters baud/bit.

Table 3. Multibeat writing

Register	Value	Comment
SHIFTCFG0...7	0x0007_0100	Configure shifter input from next shifter's output, stop bit disabled, start bit disabled and loading data on enabled, 8-bit parallel width.
SHIFTCTL0	0x0003_0002	Configure shift clock from TIMER0, shift on posedge of shift clock. Configure shifter's pin as output, pin index starting from 0, and pin polarity active high. Configure shifter mode as transmit.
SHIFTCTL1...7	0x0000_0002	Configure transmit using TIMER0, shift on posedge of shift clock. Configure shifter's pin as output disabled, pin index starting from 0, and pin polarity active high. Configure shifter mode as transmit.
TIMCMP0	0x0000_3F01	Set TIMCMP [15:8] =number of beats x 2 - 1 = 32 x 2 - 1 = 63 for 32 beats. Set TIMCMP [7:0] =baud rate divider / 2 - 1 = 4 / 2 - 1 = 1.
TIMCFG0	0x0000_2200	Configure timer output logic 1 when enabled and not affected by reset, decrement on FlexIO clock, shift clock equals timer output, never reset, disabled on timer compare, enabled on trigger high, stop bit disabled, and start bit disabled.
TIMCTL0	0x1DC3_1081	Configure SHIFT7 status flag as timer trigger, and trigger polarity active low. Configure timer's pin as output, pin index as 16 (WR), pin polarity active low. Configure timer mode as 8-bit counters baud/bit.

Table 4. Multibeat reading

Register	Value	Comment
SHIFTCFG0...6	0x0007_0100	Configure shifter input from next shifter's output, stop bit disabled, start bit disabled and loading data on enabled, 8-bit parallel width.
SHIFTCFG7	0x0007_0000	Configure shifter input from pin, stop bit disabled, start bit disabled and loading data on enabled, 8-bit parallel width.
SHIFTCTL0...7	0x0080_0001	Configure shift clock from TIMER0, shift on negedge of shift clock. Configure shifter's pin as input, pin index starting from 0, and pin polarity active high. Configure shifter mode as receive.
TIMCMP0	0x0000_3F01	Set TIMCMP [15:8] =number of beats x 2 - 1 = 32 x 2 - 1 = 63 for 32 bits. Set TIMCMP [7:0] =baud rate divider / 2 - 1 = 4 / 2 - 1 = 1.
TIMCFG0	0x0000_2220	Configure timer output logic 1 when enabled and not affected by reset, decrement on FlexIO clock, shift clock equals timer output, never reset, disabled on timer compare, enabled on trigger high, stop bit enabled on timer disabled (for internal signal synchronization to disable timer in time), and start bit disabled.
TIMCTL0	0x01C3_1181	Configure SHIFT0 status flag as timer trigger, and trigger polarity active low. Configure timer's pin as output, pin index as 17 (RD), pin polarity active low. Configure timer mode as 8-bit counters baud/bit.

12.2 Hardware connections

Table 5 and Table 6 provide the hardware connections between the FRDM-MCXE31B module and the LCD module. Use external wires to set up these connections. Table 5 lists the data connections of the FlexIO pins to the LCD module. Note that FXIO_D8 to FXIO_D15 pins are dedicated for a 16-bit bus interface.

Table 5. FlexIO data line pins

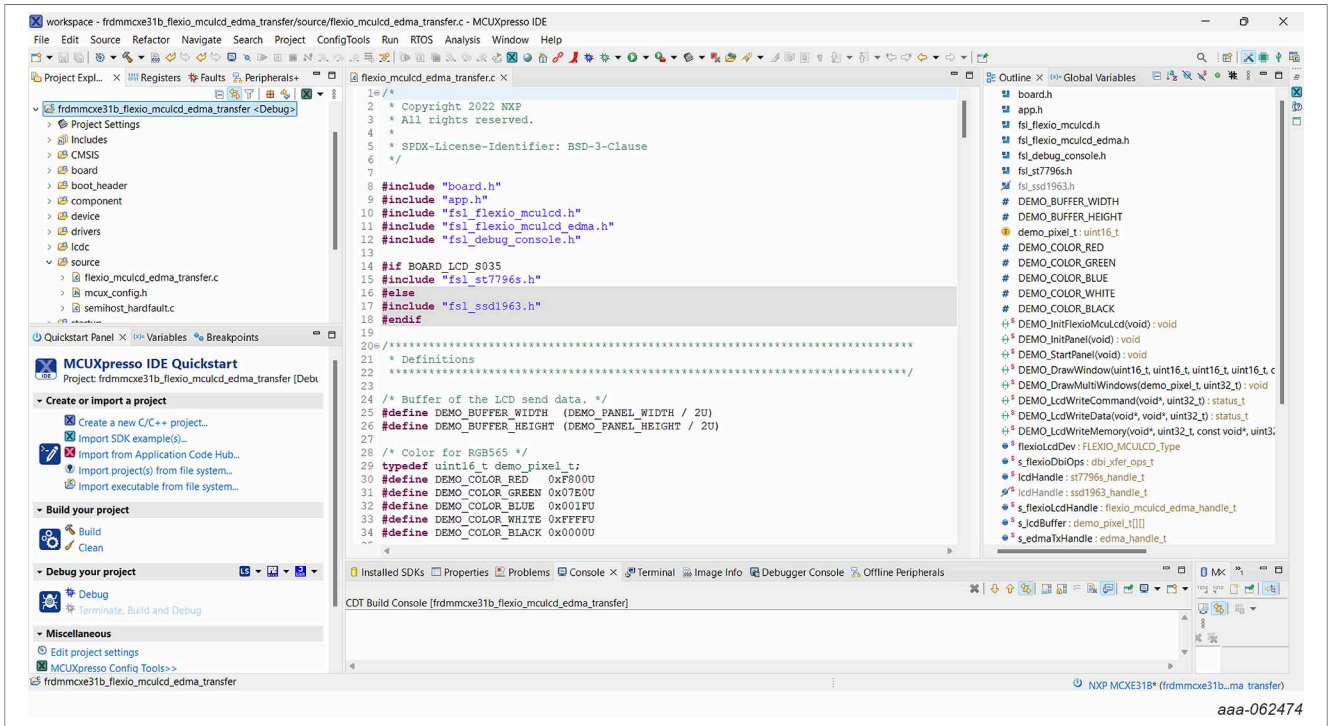
MCU pin	LCD signal	FlexIO pin	Board connector	Data bit
PTC23	D0	FLEXIO0_D16	J8-13	Data bit 0
PTC24	D1	FLEXIO0_D17	J8-14	Data bit 1
PTC25	D2	FLEXIO0_D18	J8-15	Data bit 2
PTC26	D3	FLEXIO0_D19	J8-16	Data bit 3
PTC27	D4	FLEXIO0_D20	J8-17	Data bit 4
PTC28	D5	FLEXIO0_D21	J8-18	Data bit 5
PTC29	D6	FLEXIO0_D22	J8-19	Data bit 6
PTC30	D7	FLEXIO0_D23	J8-20	Data bit 7
PTC31	D8	FLEXIO0_D24	J8-21	Data bit 8
PTD20	D9	FLEXIO0_D25	J8-22	Data bit 9
PTD21	D10	FLEXIO0_D26	J8-23	Data bit 10
PTD22	D11	FLEXIO0_D27	J8-24	Data bit 11
PTD23	D12	FLEXIO0_D28	J8-25	Data bit 12
PTD24	D13	FLEXIO0_D29	J8-26	Data bit 13
PTD26	D14	FLEXIO0_D30	J8-27	Data bit 14
PTD27	D15	FLEXIO0_D31	J8-28	Data bit 15

Table 6. Control signal pins

MCU pin	LCD signal	FlexIO pin	Board connector	Description
PTC6	I2C_SDA	-	J8-4	I2C data line
PTC7	I2C_SCL	-	J8-3	I2C clock line
PTC9	BLK	-	J8-6	Backlight control
PTC8	D/C	-	J8-8	Data/command select
PTC11	/WR	FLEXIO0_D15	J8-10	Write enable (active low)
PTC10	TE	-	J8-12	Tearing effect signal
PTB0	/RD	FLEXIO0_D14	J8-11	Read enable (active low)
PTB1	/CS	-	J8-9	Chip select (active low)
PTB14	/INT	-	J8-5	Interrupt signal
PTB15	/RST	-	J8-7	Reset signal (active low)

13 Running the demo

Download and install SDK_25_09_00_FRDM-MCXE31B to the MCUXpresso IDE. Import the frdmmcxe31b_flexio_mculcd_edma_transfer SDK example.

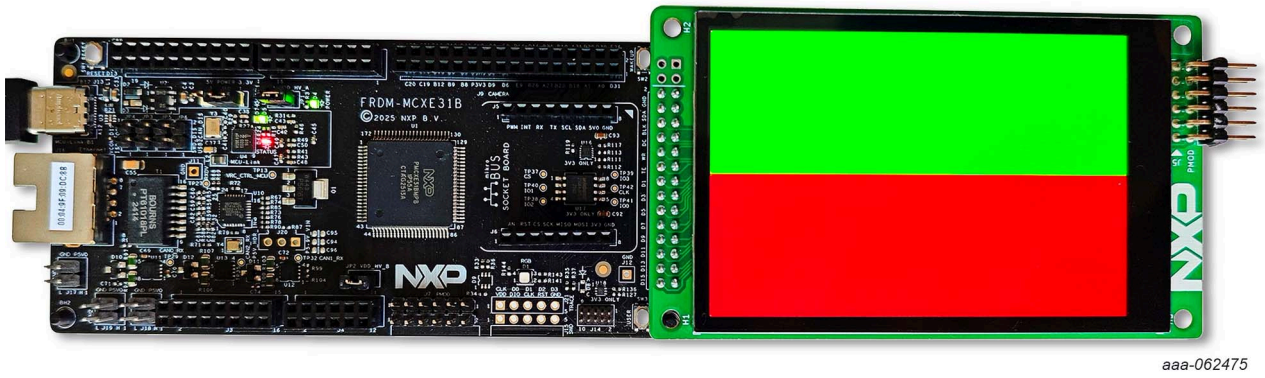


aaa-062474

Figure 12. MCUXpresso IDE

Use USB-Type C cable to connect the computer and J13 of FRDM-MCXE31B. Compile and download the code to the board.

Push the reset button. The LCD display is shown in Figure 13.



aaa-062475

Figure 13. LCD display

14 Conclusion

This application implements a demo that emulates an 8080 bus by using FlexIO. Several driver functions are implemented for writing and reading to drive a TFT LCD module. The measurements indicate that the performance is satisfactory with 60 fps needed to refresh a 320 x 480 x 2 sized LCD module.

15 Revision history

Table 7. Revision history

Document ID	Release date	Description
AN14776 v.1.0	22 September 2025	• Initial version

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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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