AN14980

Emulating WS2812 Bus with FlexIO on MCX A366

Rev. 1.0 — 23 March 2026 Application note

Document information

Information	Content
Keywords	AN14980, MCX, MCX A, MCX A366, WS2812, FlexIO, DMA
Abstract	This application note describes how to use the FlexIO module to emulate the WS2812 LED strip bus interface on MCX A366.

1 Introduction

This application note describes how to use the Flexible Input/Output (FlexIO) module to emulate the WS2812 LED strip bus on MCX A366. Using the FlexIO module can generate all the necessary WS2812 signals with Direct Memory Access (DMA).

1.1 MCX A366 introduction

The MCX A366 processor is based on the Arm Cortex-M33 platform with a CPU clock of up to 240 MHz. This application note presents a method to control WS2812 addressable RGB LEDs on the MCX A366 of NXP by using FlexIO to emulate an SPI-like Master Out Slave In (MOSI) data stream. A single FlexIO timer generates a fixed-rate **slot clock**, while a single FlexIO shifter serializes a pre-encoded bitstream whose duty pattern within each bit cell represents logic **0** or **1** per the WS2812 protocol. Unlike true Serial Peripheral Interface (SPI), the external clock is not used and the high/low timing of the data line carries the symbol information. The solution is lightweight, deterministic, and scales to long LED chains by adding eDMA to feed the shifter without CPU intervention.

1.2 FlexIO introduction

FlexIO is a highly configurable, pin-flexible peripheral block found on many NXP MCUs (for example, i.MX RT, S32K, Kinetis, and MCX). Instead of being a fixed UART/SPI/I²C/I²S/etc, FlexIO gives you building blocks, Shifters, Timers, and Pins, that you can wire up (in software) to emulate a wide range of serial/parallel interfaces or generate complex waveforms. It shines when you need custom protocols, parallelism, or higher I/O throughput than standard GPIO bit-banging can deliver.

Typical devices expose several Shifters (serial in/out), several Timers (clocking/word timing), and a bank of FlexIO pins (mapped to SoC pads via IOMUX). The exact counts and max clocks vary by MCU.

Why use FlexIO?

Protocol Emulation: Build UART, SPI, I²C, I²S/PCM, camera interface, 8080/6800-style MCU LCD, NeoPixel (WS2812/WS2811), one-wire, custom busses, and so on.

Parallelism: Drive multiple lanes in parallel (for example, multiple WS2812 strips in parallel, parallel RGB, or parallel camera).

Offload the CPU: With DMA feeding/reading FlexIO shifters and hardware timers generating bit/word clocks, the CPU overhead is minimal.

Pin Flexibility: Map signals to convenient pins (subject to IOMUX options) and reconfigure in software as your design evolves.

Bridging Gaps: When a dedicated peripheral is missing or already used, FlexIO can fill in.

Figure 1 shows the FlexIO block diagram.

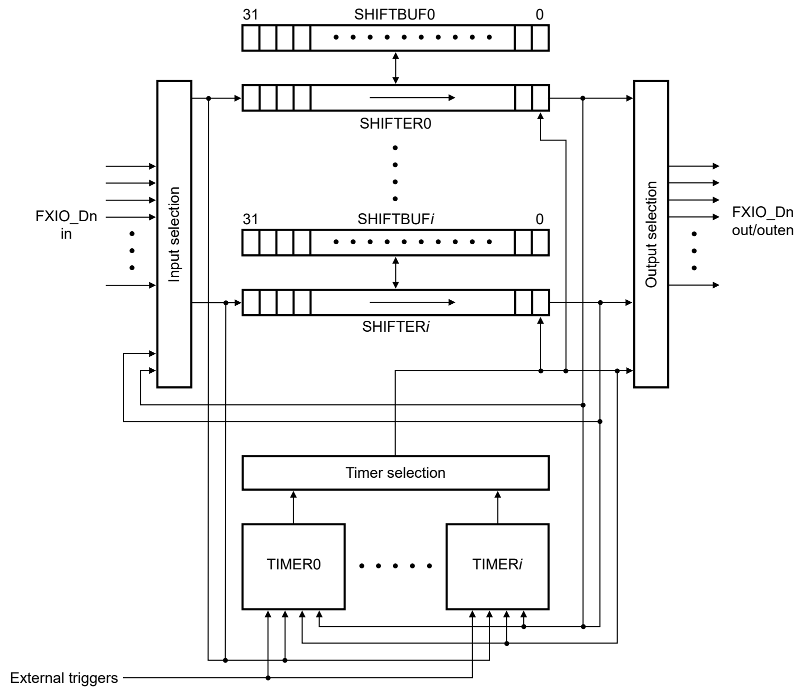


Figure 1. FlexIO block diagram

The number of timers, shift registers, and pins depend on the implementation on the microcontroller. For example, the FlexIO on the MCX A366 has:

- 4 x 32-bit shifters
- 4 x 16-bit timers
- 32 x input/output pins

1.3 WS2812 introduction

The WS2812 (often seen as WS2812B) is an RGB LED with a built-in LED driver and controller. Each “pixel” contains a red, green, and blue LED plus a tiny IC that handles color and brightness. Multiple pixels can be chained together and controlled with just one data wire.

1.3.1 WS2812 Key features

WS2812 Key features include:

- 24-bit color: 8-bit per channel (R, G, B) for 16.7M colors (data order is usually GRB).
- Single-wire protocol: 800 kHz timing-based serial signal (no clock line).
- Daisy-chain: Each LED takes its own 24 bits, then forwards the rest to the next LED.
- Built-in PWM: Smooth dimming handled inside the LED; your MCU only sends color data.
- Compact: The cCommon package is 5050 (5 mm × 5 mm), ideal for strips and matrices.

1.3.2 How WS2812 works

The principles that the WS2812 works include:

- The microcontroller sends a stream of bits at ~800 kbit/s.
- Each LED latches the first 24 bits (usually GRB), displays that color, and passes the remaining bits down the chain.

- Holding the data line LOW for > 50 µs acts as a reset/latch, updating all LEDs at once.

1.3.3 WS2812 basic specs

WS2812 basic specs include:

- Supply voltage: 5 V (nominal).
- Logic level: Data **HIGH** is ideally near 0.7×VDD; with 5 V strips. If your MCU is 3.3 V, a level shifter is often recommended.
- Timing:
  - “0” bit ≈ T0H ~0.35 µs, T0L ~0.8 µs
  - “1” bit ≈ T1H ~0.7 µs, T1L ~0.6 µs
  - Reset: LOW > 50 µs

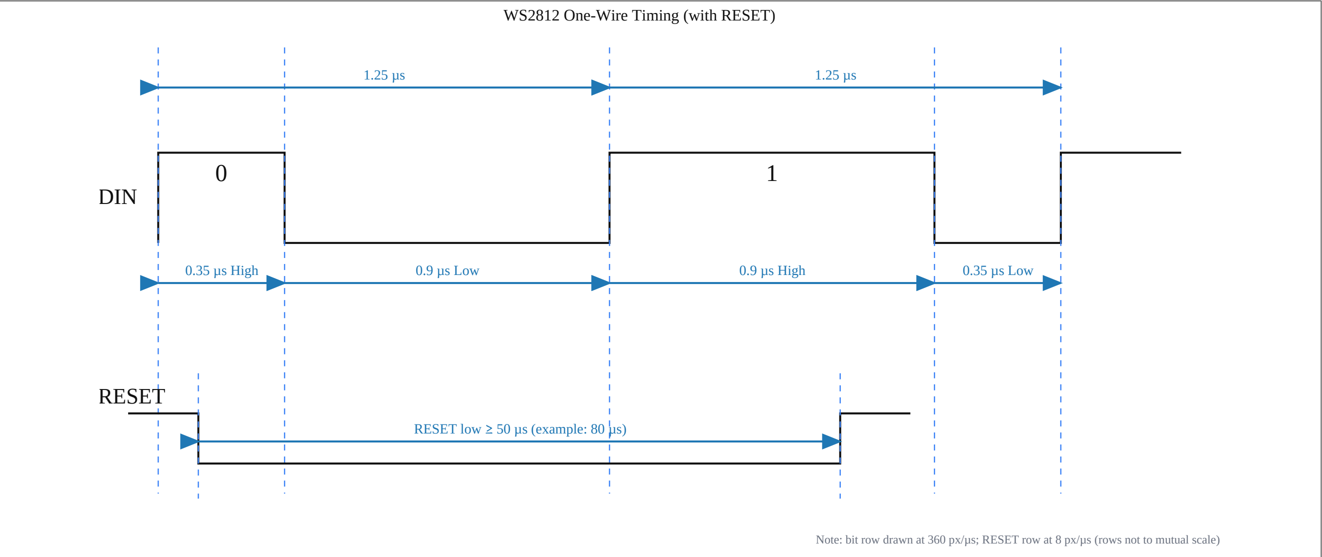

WS2812 One-Wire Timing (with RESET)
1.25 µs
1.25 µs
DIN
0
1
0.35 µs High
0.9 µs Low
0.9 µs High
0.35 µs Low
RESET
RESET low ≥ 50 µs (example: 80 µs)
Note: bit row drawn at 360 px/µs; RESET row at 8 px/µs (rows not to mutual scale)

Figure 2. WS2812 bus timing

- Current: Up to ~60 mA per LED at full white (≈20 mA per color). Plan your power accordingly.

1.3.4 WS2812 pinout

A standard WS2812/WS2812B addressable RGB LED package contains four pins. Each LED integrates an RGB chip and a digital control IC within a 5050 (5×5 mm) package. The pins are:

- VDD – Power supply pin
  This pin provides the main power for both the LED and the internal control circuitry.
  Typical operating voltage: +3.5 V to +5.3 V
- DIN – Data input pin
  This pin receives the serialized control signal from a microcontroller or the previous LED in the chain.
  The first LED takes the first 24-bit GRB color data, latches it, and passes the rest forward.
- VSS – Ground pin
  Electrical ground reference is shared by the LED and its internal IC.
  To ensure a proper signal interpretation, the pin must be connected to the ground of the controller.
- DOUT – Data output pin
  Output reshaped, regenerated digital data to the next LED in the chain.
  This pin allows easy cascading of many WS2812 LEDs in a single wire topology, as shown in Figure 3.

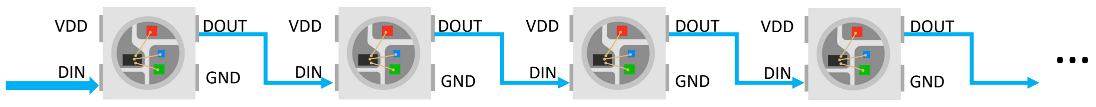


Figure 3. WS2812 data transfer

2 FlexIO emulating WS2812 bus

This section introduces the FlexIO emulating WS2812 bus.

2.1 Principle of operation

WS2812 encodes data using the pulse width of a single data line within a fixed bit period. We treat the transmit shifter of the FlexIO as an SPI-MOSI-style serializer and clock it with a FlexIO timer at a constant “slot” rate. Each WS2812 data bit is expanded into a few slots (for example, an 8-slot scheme), where the pattern of ‘1’ and ‘0’ slots yields the required high-time vs low-time. In effect, FlexIO produces a clockless SPI-MOSI waveform whose duty timing matches the WS2812 ‘0’/‘1’ definitions, while the external pin output is only the data line (no SCK output is needed).

2.2 Encoding scheme

Before transmission, the application code converts GRB color bytes into a pre-encoded slot stream. A common choice is an 8-slot mapping per WS2812 bit:

- Logic 0 → 1 1 0 0 0 0 0 0 (short high, long low)
- Logic 1 → 1 1 1 1 1 0 0 0 (long high, short low)

RESET above 50µS ‘0’ → 50x 0 0 0 0 0 0 0 0 (50x 8bits low ~ 62.5µS @6.4MHz SPI rate)

WS2812B and WS2812C require a RESET/LATCH window to keep the line LOW for at least 280 µS. With a slot clock of 6.4 MHz, this equals to or bigger than **240** bytes of 0x00 (300 µs target).

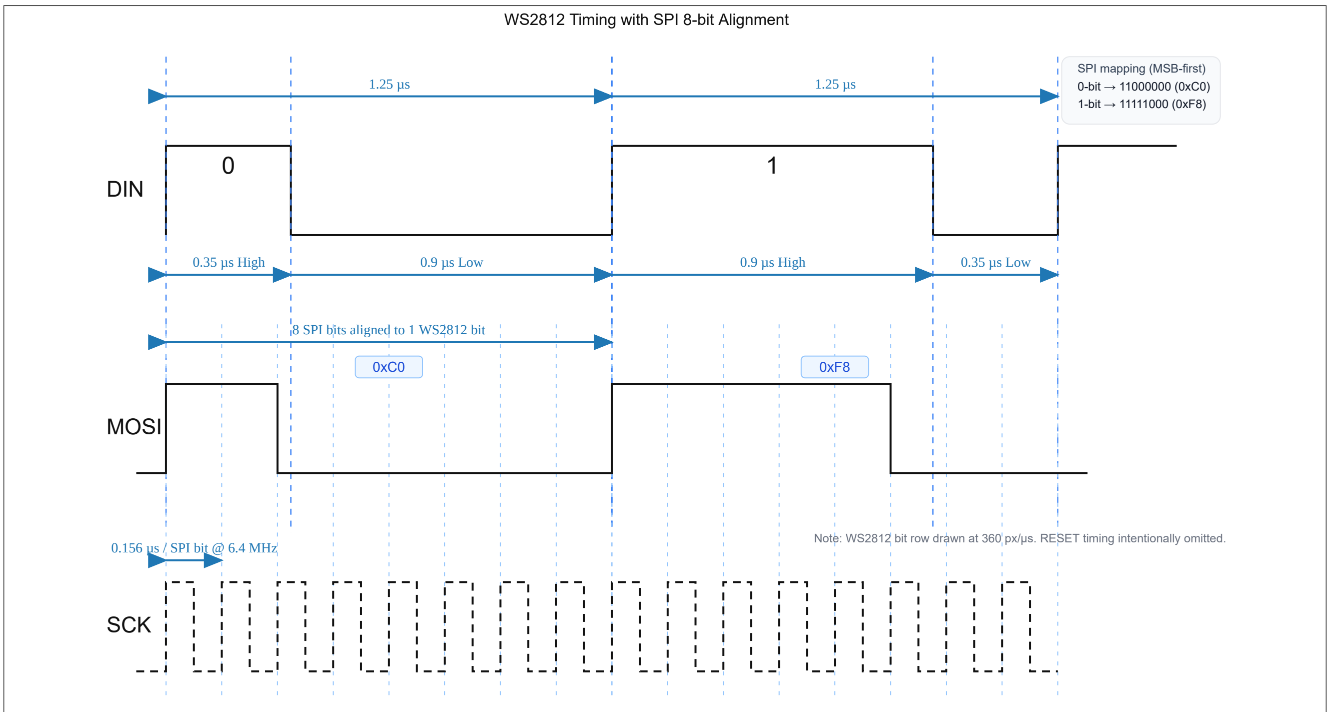

WS2812 Timing with SPI 8-bit Alignment
1.25 µs
1.25 µs
SPI mapping (MSB-first)
0-bit → 11000000 (0xC0)
1-bit → 11111000 (0xF8)
DIN
0
1
0.35 µs High
0.9 µs Low
0.9 µs High
0.35 µs Low
8 SPI bits aligned to 1 WS2812 bit
0xC0
0xF8
MOSI
0.156 µs / SPI bit @ 6.4 MHz
Note: WS2812 bit row drawn at 360 px/µs. RESET timing intentionally omitted.
SCK

Figure 4. WS2812 bus timing with SPI alignment

This keeps the total bit period constant and implements the correct duty ratio for each symbol. The result is packed into 32-bit words and queued for the shifter.

2.3 FlexIO configuration

Timer: Configure one FlexIO timer as a periodic slot clock. Its period defines the slot width.

Shifter: Configure one FlexIO shifter in Transmit mode, clocked by that timer, with the MOSI pin (`FXIO_Dx`) set as an active-high output. No external SCK is routed. The timer only advances the shifter internally.

Run sequence:

1. Start the timer.
2. Stream the encoded words into SHIFTBUF.
3. Stop the timer.
4. Hold the line low for the WS2812 reset/latch interval at the end of each frame.

2.4 Scaling to long chains

Enable the DMA request of the shifter and use eDMA to move 32-bit words from RAM to SHIFTBUF on TX-empty events. This sustains long slot streams with zero CPU jitter, leaving the M33 core free for application work.

3 Software

The source code project of this application note is public on NXP Application Code Hub which users can download from Github directly.

The key source code files are *drv_flexio_ws2812.c* and *app_dma_init.c*. The implementation drives WS2812 addressable LEDs by emulating an SPI-like MOSI waveform with FlexIO and streaming a pre-encoded bitstream via eDMA to guarantee precise timing at scale. Each LED chain ("strip") is associated with one FlexIO timer + one FlexIO shifter + one eDMA channel, enabling parallel updates of up to four strips with near-zero CPU load.

- app_dma_init.c – Global EDMA initialization, per-strip handle creation, and optional channel MUX routing to connect FlexIO shifter TX-empty to the correct DMA channel.
- drv_flexio_ws2812_dma.c – WS2812 encoder, FlexIO (timer+shifter) configuration, DMA enable/disable, EDMA TCD setup and callbacks, and per-strip non-blocking send routines.

3.1 Software architecture

The software architecture is described as below:

- Bitstream buffers per strip – **ws2812_strip1_bitstream[] / ws2812_strip2_bitstream[] / ws2812_strip3_bitstream[] / ws2812_strip4_bitstream[]** hold the duty-encoded bytes that represent WS2812 '0'/'1' symbols. A 240-byte zero prologue is reserved at the buffer head for the reset/latch interval; encoding begins at bit_index = 240.
- Encoder – **Convert_RGB_To_WS2812_Bitstream()** converts an input RGB[3] array to GRB order and maps each bit to one output byte (`0xF8` for '1' and `0xC0` for '0'), producing a MOSI-style waveform with the correct high/low ratio inside each 1.25 µs WS2812 bit cell.
- FlexIO front-end – For each enabled strip, a FlexIO timer (Dual-8-bit baud/bit mode) generates the internal bit clock and a FlexIO shifter (Transmit mode) drives the MOSI-equivalent data pin. The timer is configured to start/stop on trigger/compare events and is bound to the shifter that sources the data.
- DMA back-end – eDMA moves the encoded bytes from RAM to the shifter buffer (SHIFTBUF) one byte per minor loop until the entire frame is sent (major loop), then disables DMA and signals completion via a callback. Separate eDMA handles/channels per strip allow concurrent transfers.

3.2 WS2812 data encoding

The WS2812 data encoding is described as below:

- Color ordering: Input **rgb_data[i][3]** is reordered to GRB before bit expansion.
- Bit-to-byte mapping:
  - Logic **1** → 0xF8 (binary 11111000, that is, longer high time, shorter low time)
  - Logic **0** → 0xC0 (binary 11000000, that is, shorter high time, longer low time)
  - These MOSI patterns, clocked by the FlexIO timer, synthesize the WS2812 pulse-width encoding on the single data line.
- Reset/Latch handling: The function zero-fills the first 240 bytes and starts encoding at index 240. It provides an initial low window on the line for WS2812 reset/latch timing before the first pixel data. (Exact timing is governed by your bit clock; the prologue ensures a sufficiently long idle-low.)

3.3 FlexIO configuration (per strip)

The FlexIO configuration (per strip) is described as below:

- Timer (SCK surrogate): Configured in Dual 8-bit Baud/Bit mode with internal trigger and compare-based disable, output pin enabled (named "CLKPIN" per strip). The compare value is derived from **CLOCK_GetFlexioClkFreq()** to achieve the target sub-bit rate for the encoded bytes.
- Shifter (MOSI): Configured in Transmit mode, active-high, clocked by the above timer, and routed to the strip's data pin ("DATPIN"). Start/stop bits are disabled and the input source is set from Pin (not used for TX).
- Module bring-up: **FLEXIO_WS2812_DMA_Init()** gates and attaches the FlexIO clock, releases reset, initializes the module, then programs (timer, shifter) pairs for each enabled strip and populates a small FLEXIO_WS2812TX_Type device context (flexioBase, shifterIndex[0], timerIndex[0], and so on).

3.4 DMA data path and control

The DMA data path and control is described as below:

- DMA init: **app_dma_init()** initializes DMA0, creates one eDMA handle per enabled strip, and (when present) assigns the correct channel mux request source for each FlexIO shifter TX-empty request.
- Enabling requests: The driver calls **FLEXIO_WS2812TX_EnableDMA()** to enable the status-to-DMA request of the shifter, so each TX-empty event pulls the next byte from memory.
- Transfer configuration:
  - Destination = **WS2812 TX data register** (wrapper returns FLEXIO_GetShifterBufferAddress(..., kFLEXIO_ShifterBufferBitSwapped, shifter) + 3), keeping the on-wire bit order correct.
  - Source = xfer.txData pointer (your encoded buffer).
  - Minor loop = 1 byte; Major loop = total bytes in the encoded frame.
  - The driver sets txInProgress = true, starts DMA, and the FlexIO timer/shift engine clocks bytes onto the pin autonomously.
  - Completion: The eDMA callback (**FLEXIO_WS2812TX_EDMACallback()**) disables DMA, marks txInProgress = false, and invokes the user-registered completion callback (per strip).

3.5 Public send APIs (per strip)

Each strip exposes a non-blocking **FLEXIO_WS2812_StripN_Send()**:

- Builds a flexio_ws2812_transfer_t with txData = ws2812_stripN_bitstream and dataSize = WS2812_STRIPN_BITSTREAM_SIZE.
- Creates/initializes an eDMA transfer handle (via **FLEXIO_WS2812TX_CreateHandleEMDA()**) and registers a strip-specific completion callback that sets completeFlagN = true.
- Starts the eDMA-backed transfer with **FLEXIO_WS2812_TransferEDMA()**.

This pattern exists for Strip1/Strip2/Strip3/Strip4, enabling independent and concurrent updates.

3.6 Multi-strip and scalability

The multi-strip and scalability are described as below:

- Up to four lanes are supported by conditional compilation macros (FLEXIO_WS2812_ENABLESTRIP{1..4}), each lane bound to its own FlexIO timer/shifter and its own DMA channel for deterministic parallel streaming.
- Because each minor loop is 1 byte, eDMA serves the shifter at a high rate without CPU intervention. It sustains long chains reliably — your CPU only prepares the next encoded buffer while the current frame streams out.

4 Hardware

To connect FRDM-MCXA366 with each WS2812 LED strips, perform the steps as shown in Table 1, Figure 5, and Figure 6.

1. Connect the J15 of the FRDM-MCXA366 with PC. This port provides the firmware download, debug, and provide 5V0 power supply.
2. Connect VDD and GND of each WS2812 LED strip with FRDM-MCXA366 J5' pin7 (5 V) and pin8 (GND).
3. Connect the pin25/pin26/pin27/pin28 of FRDM-MCXA366 J8 with DIN of each WS2812 LED strip.

Table 1. MCX A344 memory map

FlexIO signals	FRDM-MCXA366	WS2812 LED strip	DuPont wire color in hardware connection
FlexIO D28	P3_20/J8's pin25	LED strip 1's DIN	Green
FlexIO D29	P3_29/J8's pin26	LED strip 2's DIN	Orange
FlexIO D30	P2_22/J8's pin27	LED strip 3's DIN	Blue
FlexIO D31	P3_23/J8's pin28	LED strip 4's DIN	Purple
5V0 VDD	J5's pin7	LED strip 1/2/3/4's VDD	Red
GND	J5's pin8	LED strip 1/2/3/4's GND	Black

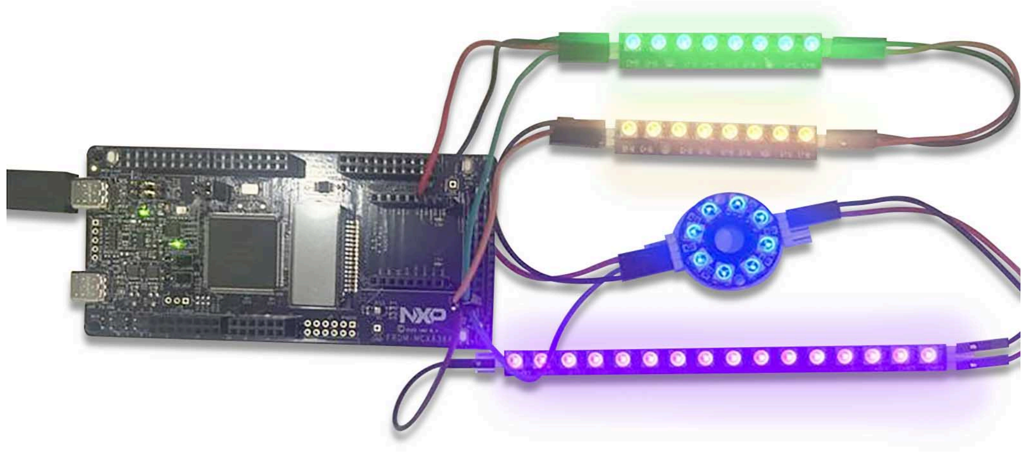


Figure 5. FRDM-MCXA366 connected to WS2812 strips

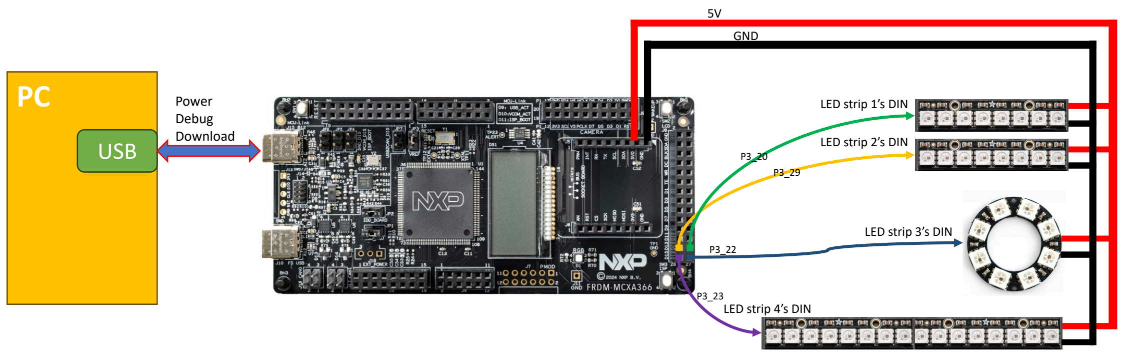
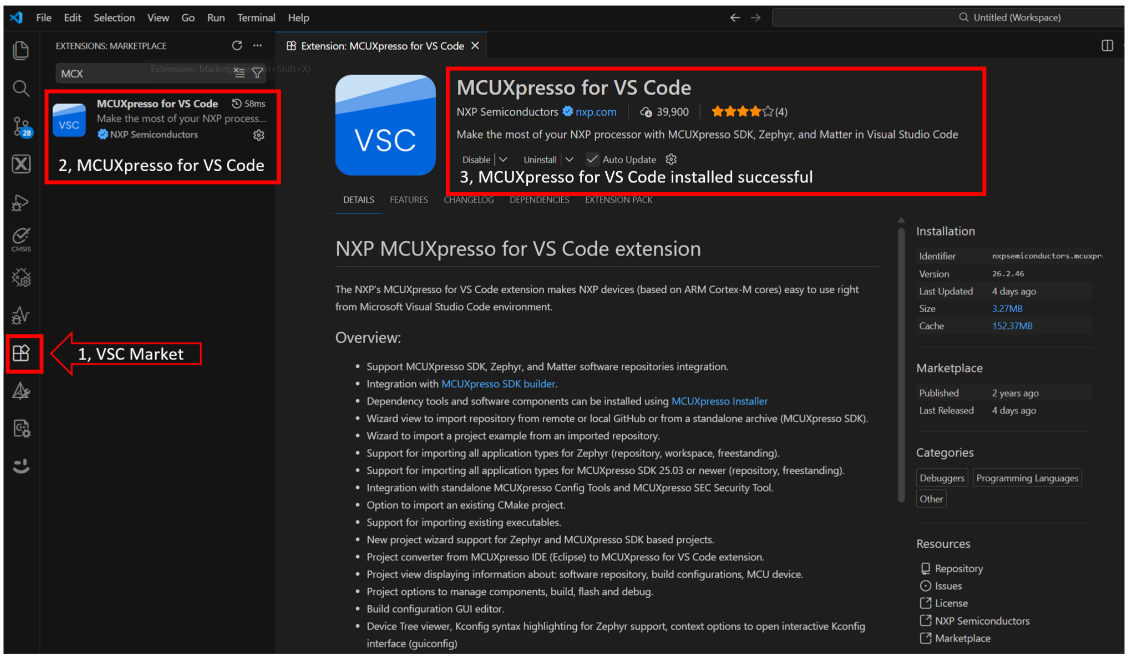


Figure 6. Hardware connection block diagram

5 Example project

This example code is based on MCUXpresso for Visual Studio Code (VSC). Install Visual Studio Code and the MCUXpresso for VS Code extension from the VS Code Marketplace before using this example.



MCUXpresso for VS Code 58ms
Make the most of your NXP process...
NXP Semiconductors

MCUXpresso for VS Code
NXP Semiconductors nxp.com | 39,900 | ★★★★☆ (4)
Make the most of your NXP processor with MCUXpresso SDK, Zephyr, and Matter in Visual Studio Code

NXP MCUXpresso for VS Code extension

The NXP's MCUXpresso for VS Code extension makes NXP devices (based on ARM Cortex-M cores) easy to use right from Microsoft Visual Studio Code environment.

Overview:

- Support MCUXpresso SDK, Zephyr, and Matter software repositories integration.
- Integration with MCUXpresso SDK builder.
- Dependency tools and software components can be installed using MCUXpresso Installer
- Wizard view to import repository from remote or local GitHub or from a standalone archive (MCUXpresso SDK).
- Wizard to import a project example from an imported repository.
- Support for importing all application types for Zephyr (repository, workspace, freestanding).
- Support for importing all application types for MCUXpresso SDK 25.03 or newer (repository, freestanding).
- Integration with standalone MCUXpresso Config Tools and MCUXpresso SEC Security Tool.
- Option to import an existing CMake project.
- Support for importing existing executables.
- New project wizard support for Zephyr and MCUXpresso SDK based projects.
- Project converter from MCUXpresso IDE (Eclipse) to MCUXpresso for VS Code extension.
- Project view displaying information about: software repository, build configurations, MCU device.
- Project options to manage components, build, flash and debug.
- Build configuration GUI editor.
- Device Tree viewer, Kconfig syntax highlighting for Zephyr support, context options to open interactive Kconfig interface (guiconfig)

Installation

Identifier	nxpsemiconductors.mcuxpr
Version	26.2.46
Last Updated	4 days ago
Size	3.27MB
Cache	152.37MB

Marketplace

Published	2 years ago
Last Released	4 days ago

Categories

Debuggers
Programming Languages
Other

Resources

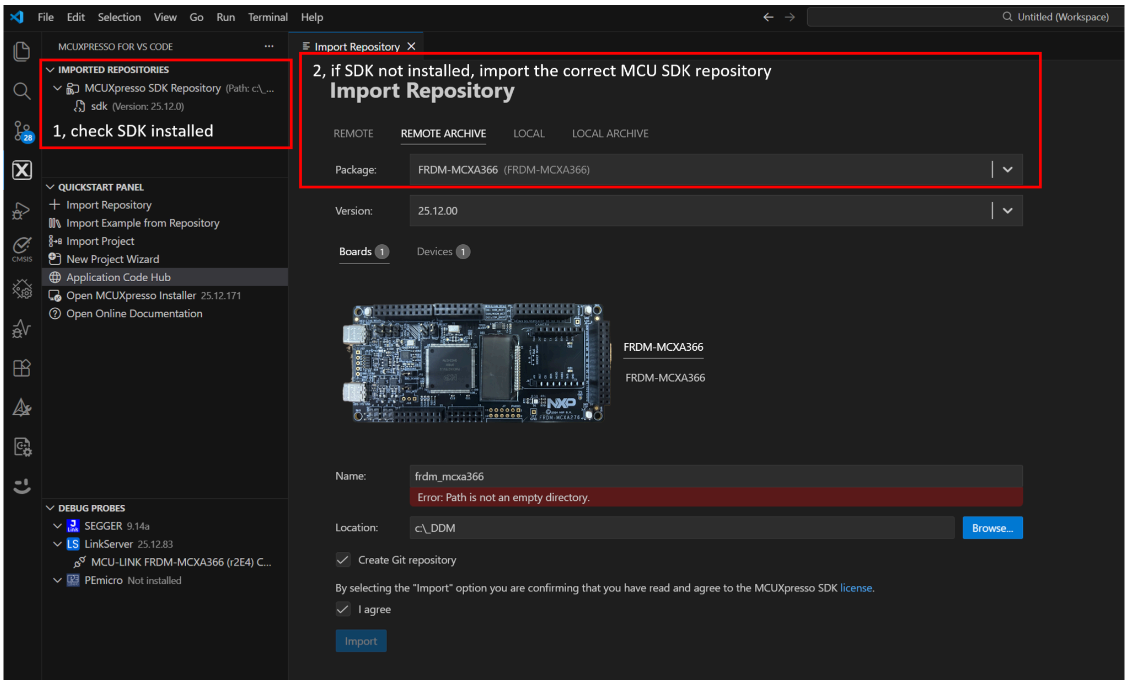
Repository
Issues
License
NXP Semiconductors
Marketplace

Figure 7. MCUXpresso for Visual Studio Code

5.1 MCUXpresso for Visual Studio Code

The example of MCUXpresso for Visual Studio Code is described as below:

1. Import the FRDM-MCXA366 SDK repository, as shown in Figure 8.


1, check SDK installed

2, if SDK not installed, import the correct MCU SDK repository

Import Repository

Package: FRDM-MCXA366 (FRDM-MCXA366)

Version: 25.12.00

Name: frdm_mcxa366

Error: Path is not an empty directory.

Location: c:_DDM

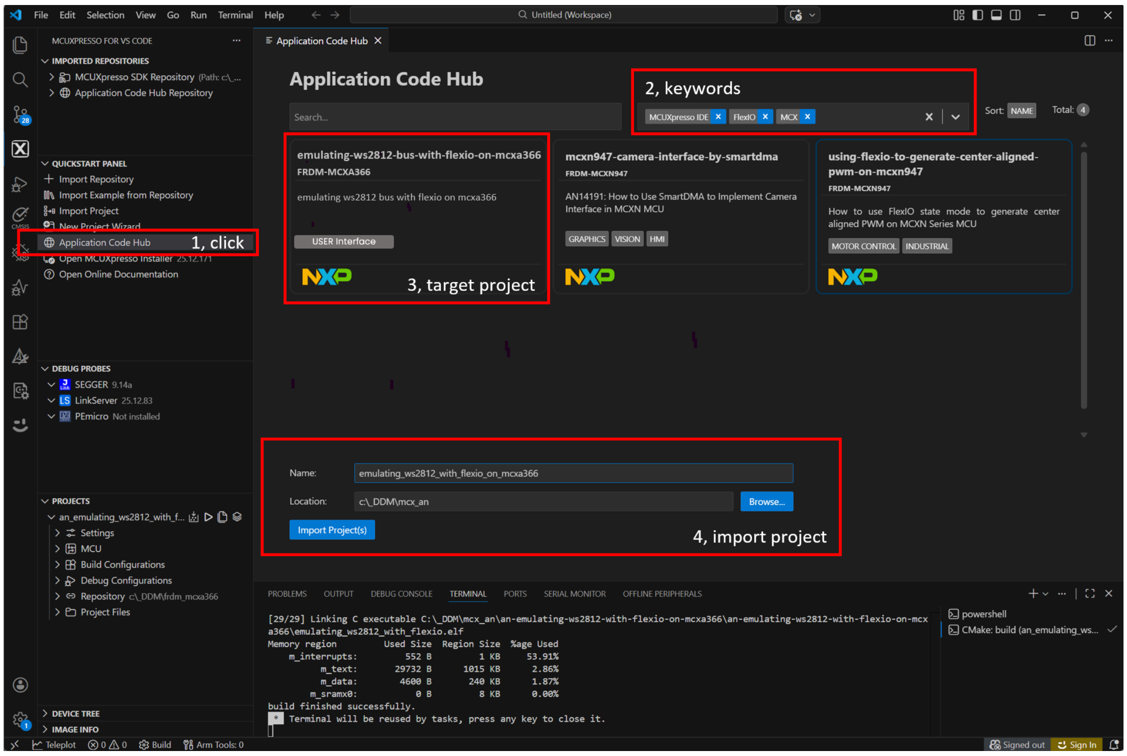
Create Git repository

By selecting the "Import" option you are confirming that you have read and agree to the MCUXpresso SDK license.

I agree

Figure 8. Import FRDM-MCXA366 SDK repository

2. Use the Application Code Hub to import this example from the GitHub repository, as shown in Figure 9.


1, click

2, keywords

3, target project

4, import project

Application Code Hub

Keywords: MCUXpresso IDE, FlexIO, MCX

emulating-ws2812-bus-with-flexio-on-mcxa366
FRDM-MCXA366
emulating ws2812 bus with flexio on mcxa366
USER Interface

mcxn947-camera-interface-by-smartdma
FRDM-MCXN947
AN14191: How to Use SmartDMA to Implement Camera Interface in MCXN MCU
GRAPHICS VISION HMI

using-flexio-to-generate-center-aligned-pwm-on-mcxn947
FRDM-MCXN947
How to use FlexIO state mode to generate center aligned PWM on MCXN Series MCU
MOTOR CONTROL INDUSTRIAL

Name: emulating_ws2812_with_flexio_on_mcxa366

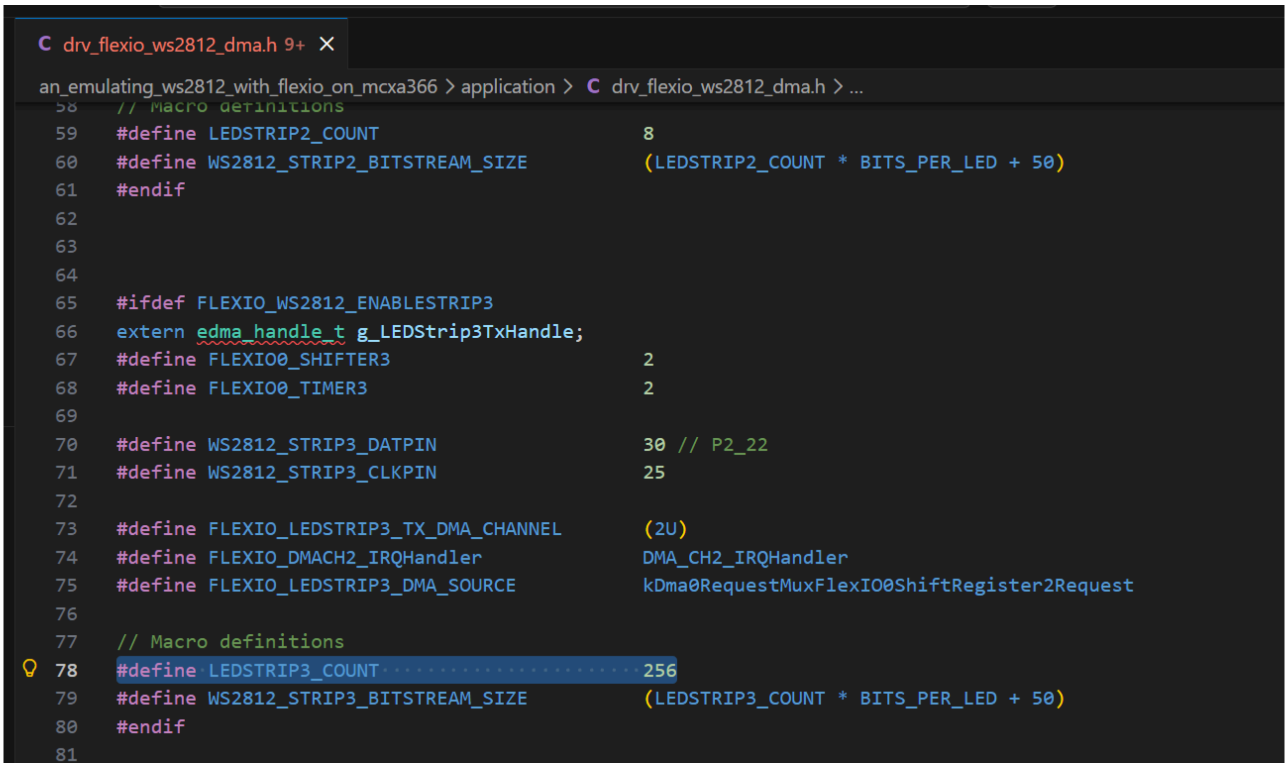
Location: c:_DDM\mcx_an

```
[29/29] Linking C executable C:\_DDM\mcx_an\an-emulating-ws2812-with-flexio-on-mcxa366\an-emulating-ws2812-with-flexio-on-mcx
a366\emulating_ws2812_with_flexio.elf
Memory region         Used Size  Region Size  %age Used
    m_interrupts:         552 B         1 KB     53.91%
          m_text:       29732 B      1015 KB      2.86%
          m_data:        4600 B       240 KB      1.87%
        m_sramx0:           0 B         8 KB      0.00%
build finished successfully.
 *  Terminal will be reused by tasks, press any key to close it.
```

Figure 9. Import ACH project into MCUXpresso for VSC

5.2 Software configuration

Update `LEDSTRIP1_COUNT`/`LEDSTRIP2_COUNT`/ `LEDSTRIP3_COUNT`/`LEDSTRIP4_COUNT` in *drv_flexio_ws2812_dma.h* to match the number of pixels on each WS2812 strip, as shown in Figure 10.


drv_flexio_ws2812_dma.h 9+
an_emulating_ws2812_with_flexio_on_mcxa366 > application > drv_flexio_ws2812_dma.h > ...

```
// Macro definitions
#define LEDSTRIP2_COUNT                         8
#define WS2812_STRIP2_BITSTREAM_SIZE            (LEDSTRIP2_COUNT * BITS_PER_LED + 50)
#endif



#ifdef FLEXIO_WS2812_ENABLESTRIP3
extern edma_handle_t g_LEDStrip3TxHandle;
#define FLEXIO0_SHIFTER3                        2
#define FLEXIO0_TIMER3                          2

#define WS2812_STRIP3_DATPIN                    30 // P2_22
#define WS2812_STRIP3_CLKPIN                    25

#define FLEXIO_LEDSTRIP3_TX_DMA_CHANNEL         (2U)
#define FLEXIO_DMACH2_IRQHandler                DMA_CH2_IRQHandler
#define FLEXIO_LEDSTRIP3_DMA_SOURCE             kDma0RequestMuxFlexIO0ShiftRegister2Request

// Macro definitions
#define LEDSTRIP3_COUNT                         256
#define WS2812_STRIP3_BITSTREAM_SIZE            (LEDSTRIP3_COUNT * BITS_PER_LED + 50)
#endif

```

Figure 10. Update WS2812 strip LED numbers

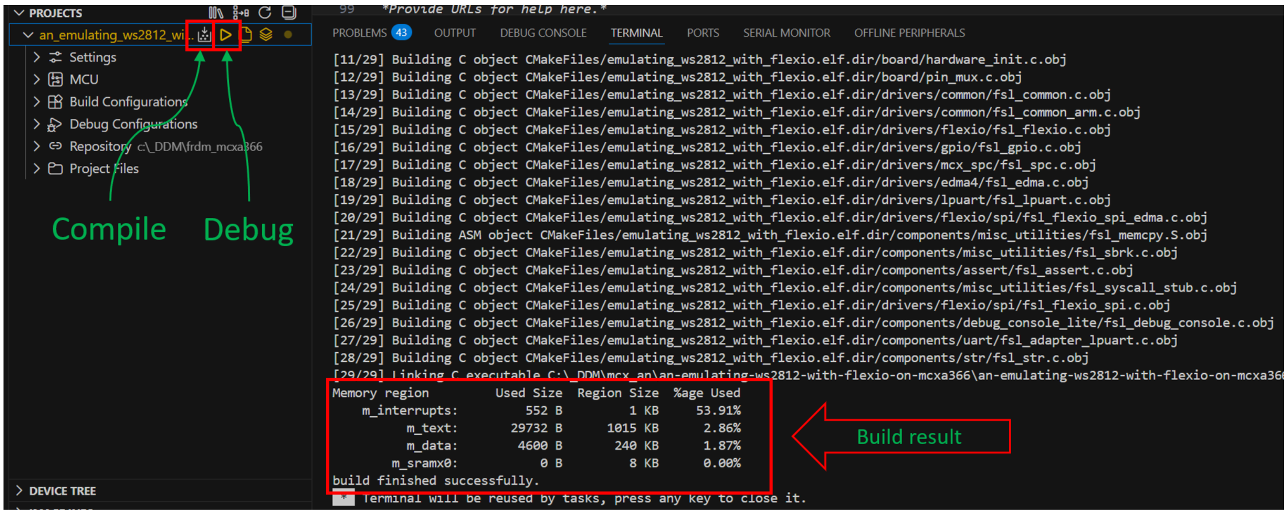
5.3 Run demo

After the demo project has been imported successfully, perform the steps as shown in Figure 11:

1. Select the project and click **Build Project** (located next to the project name) to compile it.
2. Click **Debug** to flash the firmware to the FRDM-MCXA366.
3. Click **Run**, or press the **RESET** button of the board, to launch the demo.

Upon reset, the WS2812 LED strips display: strip 1 Green, strip 2 Orange, strip 3 Blue, and strip 4 Purple. It indicates that the example is running correctly.

If the colors appear as listed in Figure 11, the example has started successfully.



```
99    *Provide URLs for help here.*
```

```
[11/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/board/hardware_init.c.obj
[12/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/board/pin_mux.c.obj
[13/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/common/fsl_common.c.obj
[14/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/common/fsl_common_arm.c.obj
[15/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/flexio/fsl_flexio.c.obj
[16/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/gpio/fsl_gpio.c.obj
[17/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/mcx_spc/fsl_spc.c.obj
[18/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/edma4/fsl_edma.c.obj
[19/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/lpuart/fsl_lpuart.c.obj
[20/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/flexio/spi/fsl_flexio_spi_edma.c.obj
[21/29] Building ASM object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/components/misc_utilities/fsl_memcpy.S.obj
[22/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/components/misc_utilities/fsl_sbrk.c.obj
[23/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/components/assert/fsl_assert.c.obj
[24/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/components/misc_utilities/fsl_syscall_stub.c.obj
[25/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/drivers/flexio/spi/fsl_flexio_spi.c.obj
[26/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/components/debug_console_lite/fsl_debug_console.c.obj
[27/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/components/uart/fsl_adapter_lpuart.c.obj
[28/29] Building C object CMakeFiles/emulating_ws2812_with_flexio.elf.dir/components/str/fsl_str.c.obj
[29/29] Linking C executable C:\_DDM\mcx_an\an-emulating-ws2812-with-flexio-on-mcxa366\an-emulating-ws2812-with-flexio-on-mcxa36
Memory region         Used Size  Region Size  %age Used
   m_interrupts:         552 B         1 KB     53.91%
         m_text:       29732 B      1015 KB      2.86%
         m_data:        4600 B       240 KB      1.87%
       m_sramx0:           0 B         8 KB      0.00%
build finished successfully.
 *  Terminal will be reused by tasks, press any key to close it.
```

Figure 11. Compile, debug, and Run project

6 Summary

This application note demonstrates that the FRDM-MCXA366 platform can reliably drive up to four independent WS2812 LED strips in parallel by pairing one FlexIO timer and one FlexIO shifter per strip and streaming a pre-encoded MOSI-style bitstream via eDMA. Each strip has its own DMA channel and a non-blocking send routine, so long chains can be updated concurrently with deterministic timing and near-zero CPU overhead.

7 References

The following are some additional documents that you can refer to for more information on the MCX A devices:

- MCX A366 Reference Manual (document MCXAP144M240F60RM)
- FRDM-MCXA366 Board User Manual (document UM12438)
- Emulating Dual SPI Using FlexIO (document AN5242)
- WS2812 Data Sheet
- NXP FlexIO Generator for the WS2812B LED Stripe Protocol | MCU on Eclipse
- WS2812B RGB LED Pinout, Working, Interfacing Arduino and Applications

8 Acronyms

Table 2 lists the acronyms used in this document.

Table 2. Acronyms

Acronym	Description
ADC	Analog-to-Digital Converter
AOI	AND/OR/INVERT
BGA	Ball Grid Array
CAN	Controller Area Network
CMC	Core Mode Controller
DAC	Digital-to-Analog Converter

Table 2. Acronyms*...continued*

Acronym	Description
DMA	Direct Memory Access
DVFS	Dynamic Voltage and Frequency Scaling
DWT	Data Watchpoint and Trace
ECC	Error Correction Code
EIM	Error Injection Module
EMC	ElectroMagnetic Compatibility
ERM	Error Recording Module
ESD	ElectroStatic Discharge
FD	Flexible Data rate
FlexCAN	Flexible Data Rate Controller Area Network
FlexIO	Flexible Input/Output
FlexPWM	Flexible Pulse Width Modulator
FMC	Flash Memory Controller
FMU	Flash Memory Module
FS	Full Speed
GPIO	General-Purpose Input/Output
HVD	High-Voltage Detect
HVQFN	Heat sink, Very thin, Quad Flat package, Non-leaded
I/O	Input/Output
I^2C	Inter-Integrated Circuit
I3C	Improved Inter-Integrated Circuit
ISP	In-system programming
ITM	Instruction Trace Macro cell
JTAG	Joint Test Action Group
LCD	Liquid-Crystal Display
LFBGA	Low-profile, Fine pitch, Ball Grid Array
LPCAC	Low-Power Cache Controller
LPI2C	Low-Power Inter-Integrated Circuit
LPSPI	Low-Power Serial Peripheral Interface
LPUART	Low-Power Universal Asynchronous Receiver/Transmitter
LQFP	Low-profile, Quad Flat Package
LVD	Low-Voltage Detect
MBC	Memory Block Checker
MCU	MicroController Unit
OpAmp	Operational amplifier
OS	Operating System

Table 2. Acronyms*...continued*

Acronym	Description
QDC	Quadrature Decoder
QFN	Quad flat package, non-leaded
QFP	Quad Flat Package
RAM	Random-Access Memory
SOSC	System oscillator
SPC	System Power Control
SRAM	Static random-access memory
SWD	Serial Wire Debug
SWO	Serial wire debug trace data output
TCM	Tightly Coupled Memory
TDI	Test Data Input
TDO	Test Data Output
TMS	Test Mode Select
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VFBGA	Very thin, fine pitch, ball grid array
WUU	Wake-Up Unit

9 Note about the source code in the document

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

10 Revision history

Table 3 summarizes the revisions to this document.

Table 3. Revision history

Document ID	Release date	Description
AN14980 v.1.0	23 March 2026	Initial public release

Legal information

Definitions

Disclaimers

Trademarks

Tables

Figures

Contents

Date of release: 23 March 2026
Document identifier: AN14980