**NXP Semiconductors**
Application Note

Document Number: AN5280
Rev. 1, 06/2016

# Using Kinetis FlexIO to Drive a Graphical LCD

## 1. Introduction

The FlexIO peripheral is a flexible hardware peripheral provided by NXP that allows the emulation of many different serial and parallel data interfaces, state machines, and logic tables. FlexIO can manage pins, clocks, PWMs, data interfaces and more in hardware, without needing assistance from the CPU core. This flexibility can unload the CPU while it executes the application code, or enable the CPU core to be disabled in a low-power mode, and reduce current consumption.

This example introduces the flexio_lcd driver and an application which uses the FlexIO to drive a graphical LCD. Specifically, it is configured to drive the TWR-LCD-RGB development module, which includes a TFT graphical LCD. The LCD resolution is 480 x 272, and this example uses a parallel 16-bit interface for the pixel data. This LCD also requires 4 timing control signals. The FlexIO generates all the timing, and drives all of these signals. The enhanced DMA (eDMA) in the Kinetis MCU is also used to move the pixel data from the frame buffer to the FlexIO. The example does this all in hardware, and continuously drives the LCD without needing the CPU core.

**Contents**

1. Introduction ... 1
2. FlexIO Overview ... 3
3. LCD Signals and Timing Overview ... 6
4. FlexIO LCD Signal Generation ... 8
4.1. Horizontal Timing ... 9
4.2. Vertical Timing ... 11
4.3. Output Enable and Pixel Data ... 12
5. FlexIO_LCD Bus Bandwidth Performance ... 15
6. FlexIO_LCD Driver ... 16
7. Using flexio_TWR-LCD-RGB Example Application ... 18
7.1. Configure the Hardware ... 18
7.2. Build the Example Project ... 23
7.3. Program the Flash ... 24
7.4. Run the Example ... 25
8. Conclusion ... 26
9. Revision History ... 26
Appendix A. flash_QuadSPI.bat Results ... 27
Appendix B. Output Enable Pin Shifter Logic Table ... 30
Appendix C. IAR Debug Log when programming flash ... 31
Appendix D. Troubleshooting Tips ... 33
D.1. No image is displayed, only the driver patterns ... 33
D.2. Display is all white ... 33
D.3. Display shows colors, but patterns and images are not correct ... 34

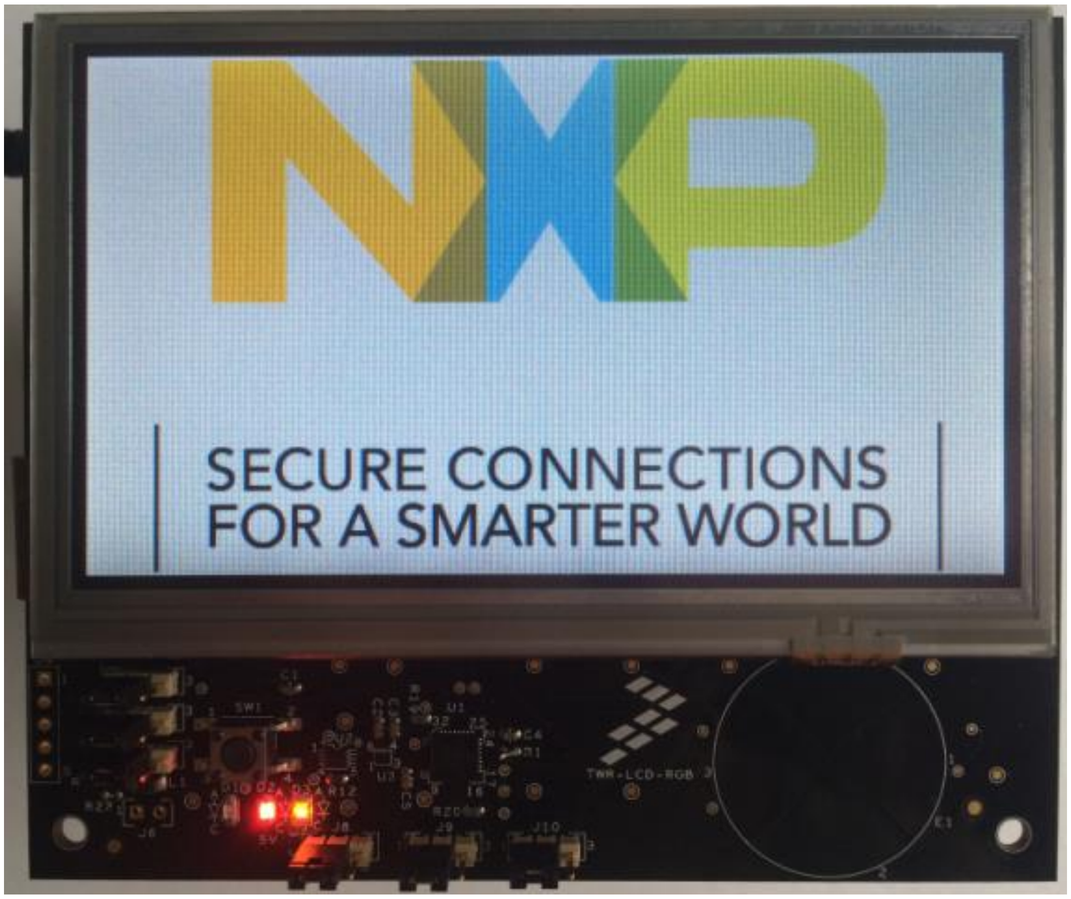


**Figure 1. TWR-LCD-RGB driven by FlexIO**

This example was developed for the TWR-K80F150M development board which features a K80 MCU. The K8x family is a great option for this use-case because it includes the FlexIO, with the latest FlexIO features that enable the implementation of an LCD interface. The K8x family also includes an external SDRAM interface, which is a good memory option to store the frame buffer for the LCD, and a high-speed QuadSPI flash interface, which is well-suited for storing the images and other resources for the GUI. *Figure 2* shows a block diagram of the system with the K80 and interfaces to the LCD, SDRAM, and QuadSPI.

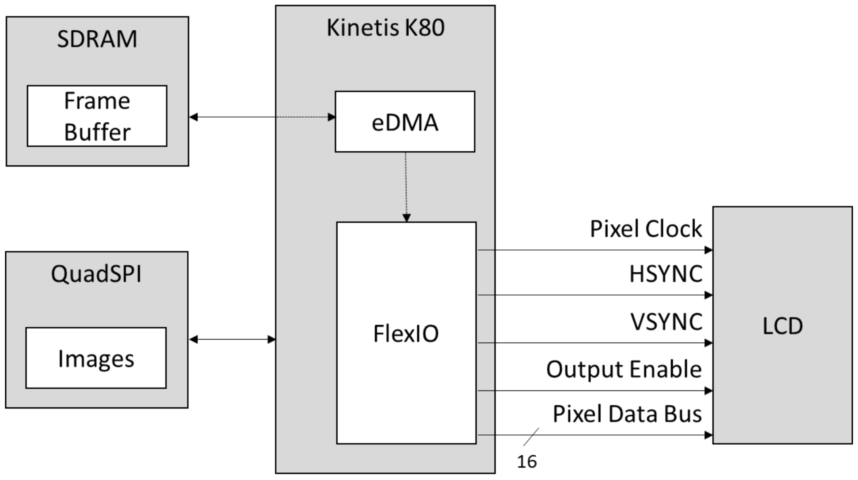


**Figure 2. Example system block diagram**

This example uses the flexio_lcd driver, which simplifies the hardware configuration for the application. The application only needs to setup the driver configuration with some parameters about the LCD requirements and FlexIO pins. Then the driver will configure the FlexIO and eDMA hardware to drive the LCD. This example uses the following MCU peripheral resources:

- 8 FlexIO Timers
- 7 FlexIO Shifters
  - 1 shifter used in Logic Mode, others use Transmit Mode
  - 1 shifter uses Parallel mode, for 16-bit interface
- 20 FlexIO pins
- 1 eDMA channel
- SDRAM controller
- QuadSPI controller (optional)

## 2. FlexIO Overview

The FlexIO peripheral is a group of timers and shift registers. It is highly configurable allowing for many different options for these timers and shifters to be interconnected, triggered, enabled, disabled, reset, and clocked. These options and features provide great flexibility in generating signals and transferring data.

*Figure 3* shows a diagram of a FlexIO timer. When a FlexIO shifter is used, it is controlled by one of these timers. The timer controls when the shifting occurs, and when the shifters are loaded with new data. The timer modes provide options in controlling these shifters, including a baud rate control. The baud rate control gives the timer more flexibility in the timing of the shifts relative to the clock source. The flexio_lcd driver uses these timer features and shifter control to generate the LCD timing signals.

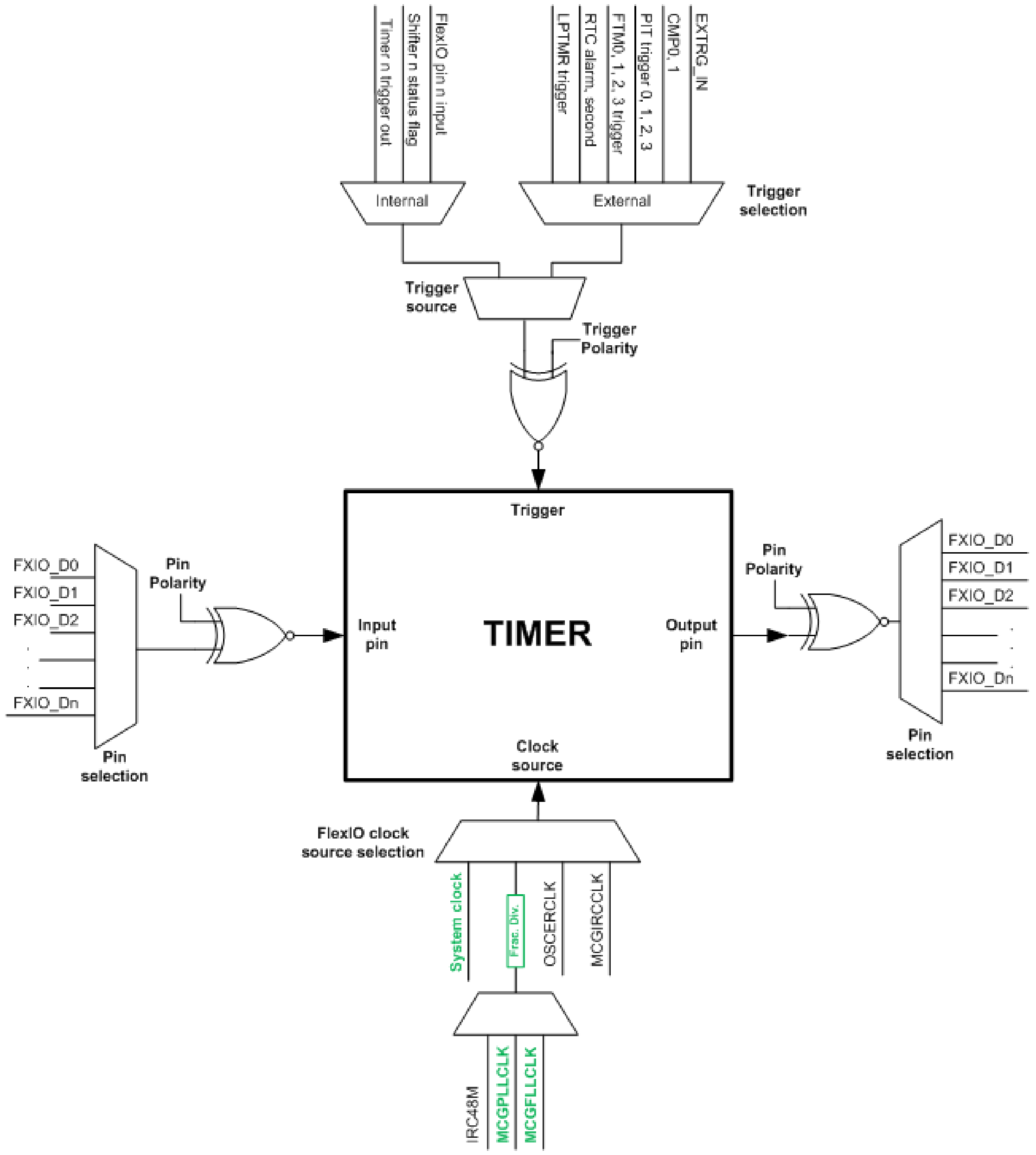


**Figure 3. FlexIO Timer Diagram**

*Figure 4* is the diagram of a FlexIO shifter. Shifters also support multiple modes. The flexio_lcd mainly uses Transmit mode, which latches data in from the shift buffer, and shifts it out a pin. Input to a shifter can also be chained to another shifter. For the pixel data clocked out to the LCD, the driver chains multiple shifters together to maximize the data throughput from the frame buffer to the shift buffers. The driver also uses shifters to drive the HSYNC and VSYNC timing signals. The shifters allow the precise timing of these signals.

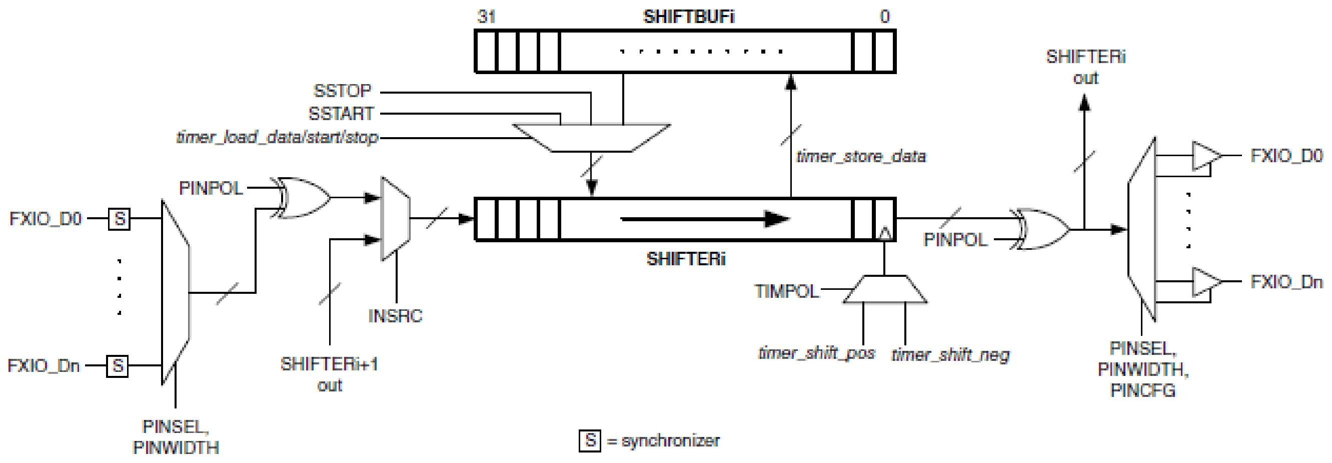


**Figure 4. FlexIO Shifter Diagram**

These timers and shifters are ideal for serial communication, which is one of the main use cases for the FlexIO. But the FlexIO peripheral was also improved with more features, giving greater flexibility and enabling more use cases. FlexIO version 1.1 includes new features like parallel mode, logic and state modes for the shifters. The Kinetis K8x family was one of the first MCUs with these features. For other MCUs, the reference manual documents if these features are included. Also, the register FLEXIOx_VERID can be read to check the version of the FlexIO peripheral in that MCU.

The parallel mode enables the ability to drive a graphical LCD. With this mode, the shifter shifts out multiple bits at a time. FlexIO supports parallel output for 4, 8, 16, and 32 bits at a time. This feature is ideal for clocking out pixel data on a parallel bus to an LCD. This example uses 16-bit parallel mode for the LCD. When designing the system with parallel mode, it is important to understand that the parallel pins must be sequential FlexIO pins. The shifter configuration determines which pins are used in parallel mode by two register bit fields. FLEXIOx_SHIFTCTLn[PINSEL] specifies the lowest pin number used with the sequential parallel pins, and FLEXIOx_SHIFTCFGn[PWIDTH] sets the number of parallel pins.

The shifter Logic Mode is another new feature used in the flexio_lcd driver. This mode turns a shifter into a logic look-up table. It has up to 5 inputs to the logic function: 4 FlexIO pins and 1 shifter. The output drives a pin, or can be routed back to other shifters and timers. The shift buffer holds the logic look-up table. This Logic Mode can help reduce external components, and provides more flexibility with the FlexIO options. This example uses Logic Mode to drive the LCD Output Enable pin, logically ANDing together other signals generated by FlexIO timers.

Another advantage FlexIO provides is operation in low-power modes. For applications that are battery powered, and current consumption is critical, FlexIO allows all this functionality while maximizing battery life. On the Kinetis K8x MCUs, the FlexIO can continue operating down to Very Low-Power Stop (VLPS) mode. This is a low-power mode where the CPU core and the peripheral clocks are stopped, and the on-chip voltage regulator is in a low-power mode. The eDMA can also operate in VLPS mode when asynchronous mode is enabled. For smaller LCDs where the frame buffer is small enough to fit in the internal SRAM of the MCU, the LCD can still be driven in VLPS mode. The FlexIO will continue to run and trigger the eDMA, which can then fetch the pixel data from the internal SRAM. For larger LCDs where the frame buffer is in external SDRAM, like this example, the WAIT power mode can be used. WAIT mode stops the CPU core clock, but continues to clock the peripherals and SDRAM.

With these features and the tremendous flexibility, the FlexIO peripheral is a great option for controlling graphical LCDs.

## 3. LCD Signals and Timing Overview

This example uses the NXP development board TWR-LCD-RGB, which features the LCD module NL4827HC19-05B from NEC. This LCD will be used as a reference when reviewing LCD signals and timing. Graphical LCDs typically have a parallel data bus which latches in the color value for a pixel. A pixel clock signal is an input to the LCD, and latches in the pixel data with each clock period. The LCD will scan through all the pixels in the display, starting with the pixel in the first column and first row of the display. With each pixel clock period, the LCD will cycle to the next pixel in the horizontal row (see *Figure 5*). After a row is complete, it will cycle to the first pixel in the next row. When all rows in the frame are completed, it will cycle back to the first row. This pixel data is constantly refreshed with each frame, as these displays typically are not able to store the pixel data.

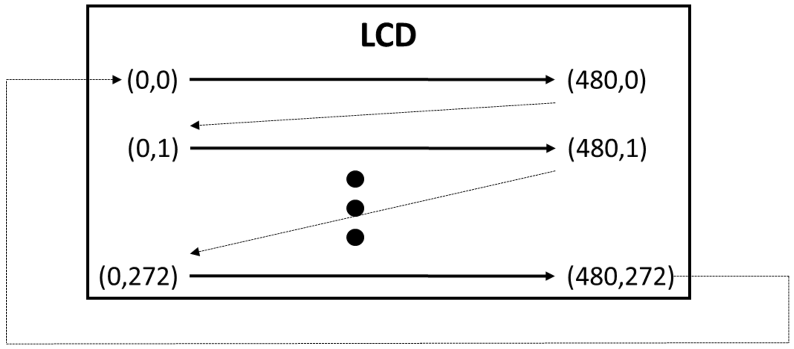


**Figure 5. LCD pixel scanning direction**

LCDs also use other timing control signals, such as Horizontal Sync (HSYNC), Vertical Sync (VSYNC), and Output Enable (OE). The HSYNC is a pulse that signals the start of a new horizontal row of pixels. This HSYNC pulse will repeat for each row through the display frame. The VSYNC pulse signals the start of a new frame, when the display completes the last row and cycles back to the first row. The Output Enable signal controls when the LCD latches in the pixel data, and this signal is typically disabled during the HSYNC and VSYNC pulses.

LCDs typically have strict timing requirements for the HSYNC and VSYNC pulses, and the related Output Enable. The HSYNC timing is based on a number of pixel clock periods, as this signal controls the timing for the pixels within each row. The VSYNC timing is based on a number of HSYNC periods, as this signal controls the timing for the number of rows in the frame. With the NL4827HC19-05B, the HSYNC and VSYNC pulses are active low. Output Enable is active high, and is disabled before these sync pulses. Also, Output Enable is held in the low disabled state for some time after the sync pulses.

The timing of Output Enable related to the HSYNC and VSYNC pulses is important when interfacing to an LCD, and how the FlexIO is used to generate these timing signals will be covered in detail. Some common terms used to describe this timing are front and back porch. With this LCD, the front porch is the duration from when OE is disabled to when the sync pulse starts (see *Figure 6*). The back porch is the duration from when the sync pulse ends to when OE is enabled. OE is enabled while pixel data is transferred, and is disabled around each HSYNC and VSYNC pulse. With this LCD, the Output Enable signal has both a horizontal and vertical component. *Figure 6* shows a generic sync pulse with the front and back porch. The timing shown here is just for illustration, and not specific to the LCD in this example.

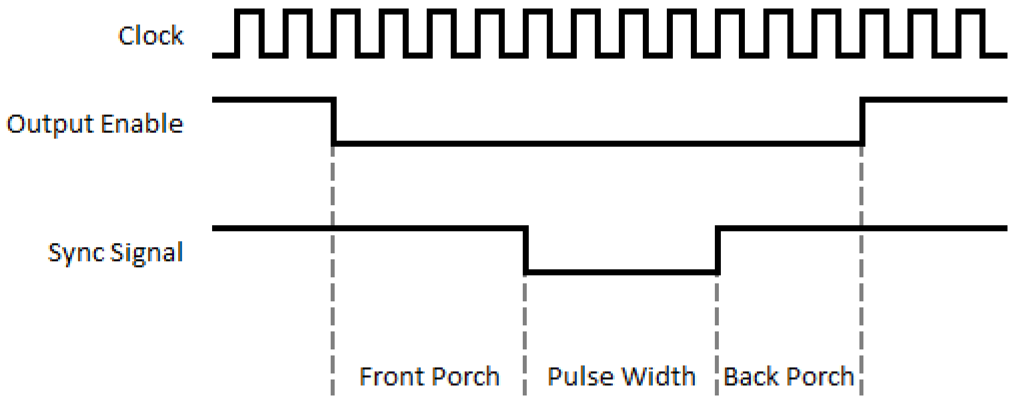


**Figure 6. Sync pulse timing with Front and Back Porch**

The HSYNC pulse width, front and back porches are specified in pixel clock periods. To give some specifics used in this example, the HSYNC pulse width is 2 clock periods. The front porch is 2 periods and the back porch is 28. That means for each horizontal row, OE is disabled for 32 pixel clock periods, and during each row, OE is enabled for 480 periods to latch all the pixels in that row. Therefore, the horizontal period between HSYNC pulses is 512 pixel clock periods.

The VSYNC pulse width, front, and back porches are specified in HSYNC pulses. For this example, the VSYNC pulse width is 2 HSYNC periods. The front porch is 2 periods, and the back porch is 10. That requires OE to be disabled during each start of frame for 14 periods. During each frame, OE is enabled for 272 rows. Therefore, the vertical period between VSYNC pulses is 286 HSYNC periods.

*Figure 7* shows a generic timing diagram of these 4 control signals. This diagram is just to illustrate the relationship of these signals, and the timing shown is not specific to the LCD in this example. The Output Enable is low around each HSYNC pulse, with a front and back porches related to the pixel clock period. But Output Enable is also low around the VSYNC pulse, with those front and back porches related to the HSYNC period. This illustrates why the Output Enable has both a horizontal and vertical component, and the FlexIO is configured to generate all these timing signals.

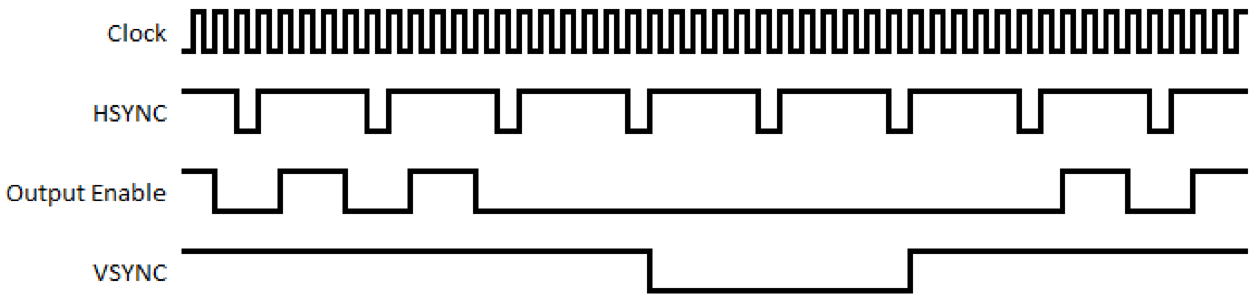


**Figure 7. Simplified timing diagram of LCD control signals**

# 4. FlexIO LCD Signal Generation

FlexIO drives all the LCD signals continuously in hardware, without any code execution required once the peripheral is started. The FlexIO generates these signals by configuring the timers and shifters to meet the LCD timing requirements. This section will give an overview of how the flexio_lcd driver configures the FlexIO for this example. Further details on the FlexIO timer and shifter configuration options and features are documented in the Kinetis MCU Reference Manual.

This example uses 8 FlexIO timers for the following purposes:

0. Pixel Clock Pin Timer
1. Horizontal Period Timer
2. Horizontal Output Enable Timer
3. Horizontal Output Enable Pin Timer
4. Vertical Period Timer
5. Vertical Output Enable Timer
6. Vertical Output Enable Pin Timer
7. Data Timer

Also, this example uses 7 FlexIO shifters, 4 of which are chained together for shifting the pixel data:

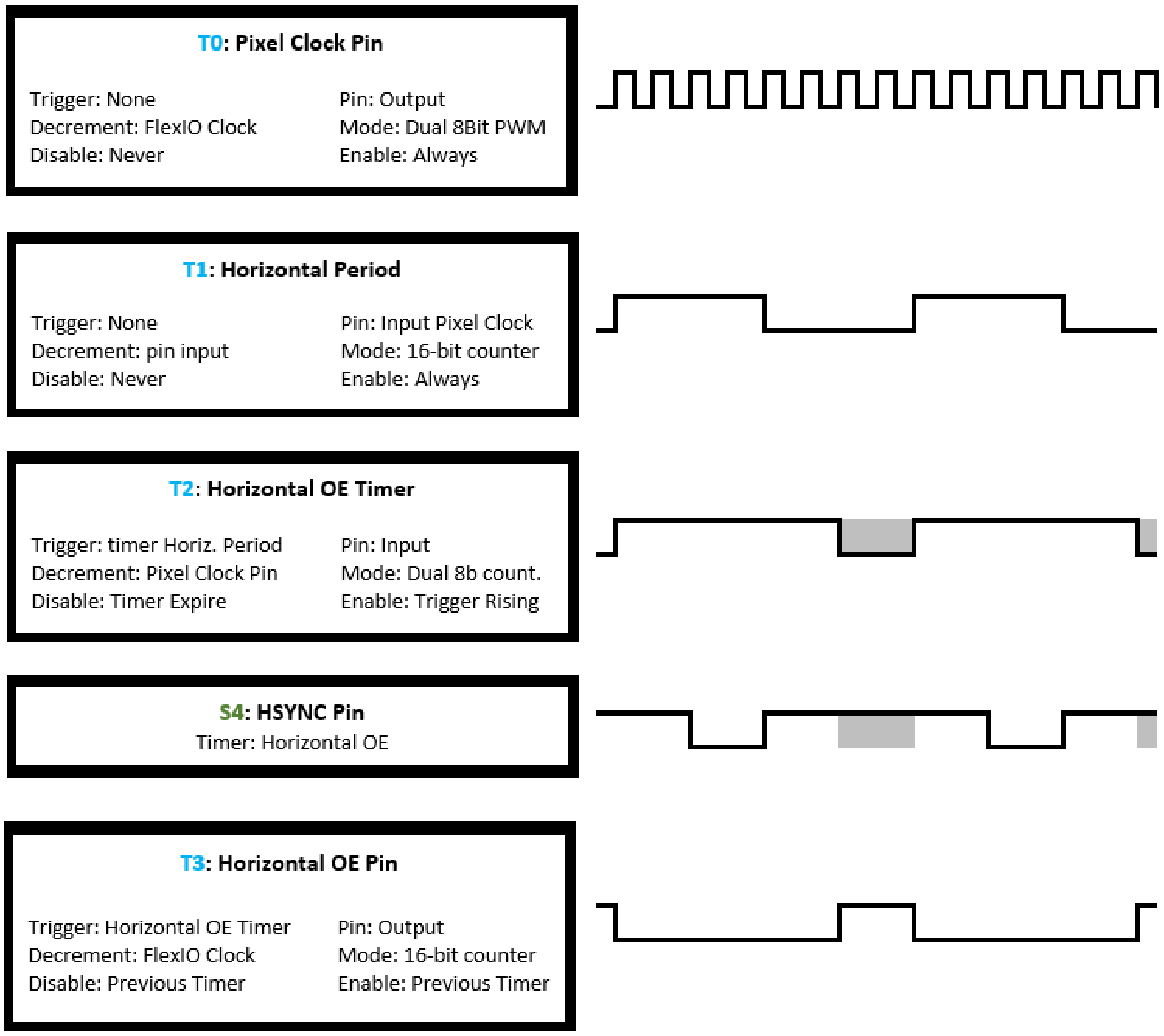
8. Data Shifter 0
9. Data Shifter 1
10. Data Shifter 2
11. Data Shifter 3
12. HSYNC Pin Shifter
13. VSYNC Pin Shifter
14. unused
15. Output Enable Pin Shifter (used in logic mode).

The first signal generated is the pixel clock. A FlexIO timer is used here to generate a simple PWM waveform, and will drive that output to the LCD. The same signal is also used as an input back into the FlexIO for the timing of the other timers and shifters. The clock source for this timer is the FlexIO clock source, and the timer counter will decrement with each rising edge of that clock. This timer uses the 8bit PWM mode to generate a PWM with a 50 % duty cycle. This PWM mode uses the lower 8bits of the timer compare register (FLEXIOx_TIMCMPn) for the high period of the clock signal, and the upper 8bits define the low period. The flexio_lcd driver calculates this compare value based on the FlexIO clock source frequency and the pixel clock frequency parameter passed to the driver. The result is a clock output pin which drives the LCD and controls all the remaining FlexIO timing.

## 4.1. Horizontal Timing

The horizontal timing controls the HSYNC pulse and period, and the horizontal component of the Output Enable signal. This timing is based on the pixel clock period, and uses the clock output pin from the Pixel Clock timer. The FlexIO generates this timing using a combination of timers and shifters.

*Figure 8* shows these horizontal timing signals. Each horizontal period starts on the rising edge of the Horizontal Period signal. The Horizontal OE Timer controls the timing for the HSYNC pulse width, front and back porch. The figure shows when this timer is disabled in the gray shaded areas, which is also when the Horizontal OE Pin is enabled/high. While this timer is enabled, the HSYNC Pin Shifter drives the HSYNC pin, creating the HSYNC pulse width, and front and back porch.



**T0: Pixel Clock Pin**

Trigger: None
Decrement: FlexIO Clock
Disable: Never
Pin: Output
Mode: Dual 8Bit PWM
Enable: Always

**T1: Horizontal Period**

Trigger: None
Decrement: pin input
Disable: Never
Pin: Input Pixel Clock
Mode: 16-bit counter
Enable: Always

**T2: Horizontal OE Timer**

Trigger: timer Horiz. Period
Decrement: Pixel Clock Pin
Disable: Timer Expire
Pin: Input
Mode: Dual 8b count.
Enable: Trigger Rising

**S4: HSYNC Pin**

Timer: Horizontal OE

**T3: Horizontal OE Pin**

Trigger: Horizontal OE Timer
Decrement: FlexIO Clock
Disable: Previous Timer
Pin: Output
Mode: 16-bit counter
Enable: Previous Timer

**Figure 8. Horizontal Timing Signals**

The base for the horizontal timing is controlled by the Horizontal Period timer. This timer generates a horizontal period event based on the HSYNC period. This period event is used by other horizontal timers and shifters. The period event is generated using the 16bit counter mode of the timer, where the rising edge of the timer output triggers a new horizontal period. Horizontal timing is based on a number of pixel clock periods, so this timer uses the pixel clock pin as an input and is decremented by edges on that pin. The value for the timer compare register is calculated by the flexio_lcd driver, adding the following parameters from the driver configuration:

- Number of pixels per row
- HSYNC Front Porch duration
- HSYNC Back Porch duration
- HSYNC pulse width

The Horizontal Period timer is always enabled, and the timer output rising edge will be used as a trigger for the remaining horizontal timing components.

The next timer is the Horizontal Output Enable Timer. This timer generates the horizontal component of the Output Enable signal. It is enabled when the OE signal is disabled around the HSYNC pulse. This timer is used to control the shifting of the HSYNC pin shifter discussed below.  It is enabled every horizontal period, triggered by the Horizontal Period Timer, and disables when OE is enabled for a new horizontal row. The output of this timer is inverted from the Output Enable pin polarity because its output enables the HSYNC Pin Shifter.

The Pixel clock pin is an input to this timer, and the timer counter decrements on the pixel clock edges. The timer mode is Dual 8bit counter. This mode is used for the shifter, because the timer decrements on both edges of the pixel clock. In this mode, the lower 8bits of the timer compare register control the number of clock edges per shift. These bits are setup to shift every 2$^{nd}$ clock edge, so the shifting occurs on the rising edge of the pixel clock. The upper 8bits of the compare register control how many shifts occur while the timer is enabled. The flexio_lcd driver calculates this value from the driver configuration by adding the HSYNC pulse width, and the front and back porch widths. The result is a timer that controls the duration of the OE disabled pulse for the horizontal component and HSYNC shifting.

The HSYNC Pin Shifter is the next piece for this horizontal timing. This is a FlexIO shift register that shifts out the state of the HSYNC pin. It is active when the Horizontal Output Enable Timer is enabled, and the shifting is controlled by that timer. The timing of the HSYNC pulse width, front and back porch are controlled by loading a value in the shift register and shifting it out the pin. Each bit in the shifter value represents the state of the HSYNC pin for each pixel clock period. For this example, the first two shifter bits are 1s for the front porch. The next two bits are 0s for the HYSNC pulse width.  Then the remaining 28 bits are 1s for the back porch. The flexio_lcd driver calculates this shift register value based on the driver configuration.

The Horizontal OE Pin Timer is the final piece for the horizontal timing. This timer is used to drive an output pin for the horizontal component of the OE signal. The OE signal to the LCD is a logical AND of the horizontal and vertical OE components. The FlexIO uses a shifter in Logic mode to AND these signals together, and therefore requires each component signal to be output on a FlexIO pin. One thing to note here is that these component outputs are not used by the LCD; the LCD only needs the ANDed OE signal. The example uses these FlexIO outputs only to route to another shifter internally. The signals do not need to come out of the MCU, and the MCU pins tied to these FlexIO signals can be used for other peripheral signals or GPIO.

The output of this Horizontal OE Pin Timer is simply the inverted value of the trigger timer: the Horizontal Output Enable Timer. That trigger timer is unable to output to a pin directly because it uses the pixel clock as an input pin to decrement the timer, and it is triggered by the Horizontal Period timer.

So this Horizontal OE Pin Timer is used to drive the Horizontal OE pin output based on the trigger timer. It is enabled and disabled by the trigger timer. The pin is set as an output, and the polarity is active low to invert the output from the trigger timer. The result is an output pin for the horizontal component of the OE signal.

## 4.2. Vertical Timing

The vertical timing signals are similar to the horizontal timing. The key difference is that the vertical timing is all clocked by the HSYNC pulses, rather than the pixel clock. The vertical timing controls the VSYNC pulse and period, and the vertical component of the Output Enable signal.

*Figure 9* shows these vertical timing signals. Each vertical period starts on the rising edge of the Vertical Period signal. The Vertical OE Timer controls the timing for the VSYNC pulse width, and front and back porch. The figure shows when this timer is disabled in the gray shaded areas, which is also when the Vertical OE Pin is enabled/high. While this timer is enabled, the VSYNC Pin Shifter drives the VSYNC pin, creating the VSYNC pulse, and front and back porch.

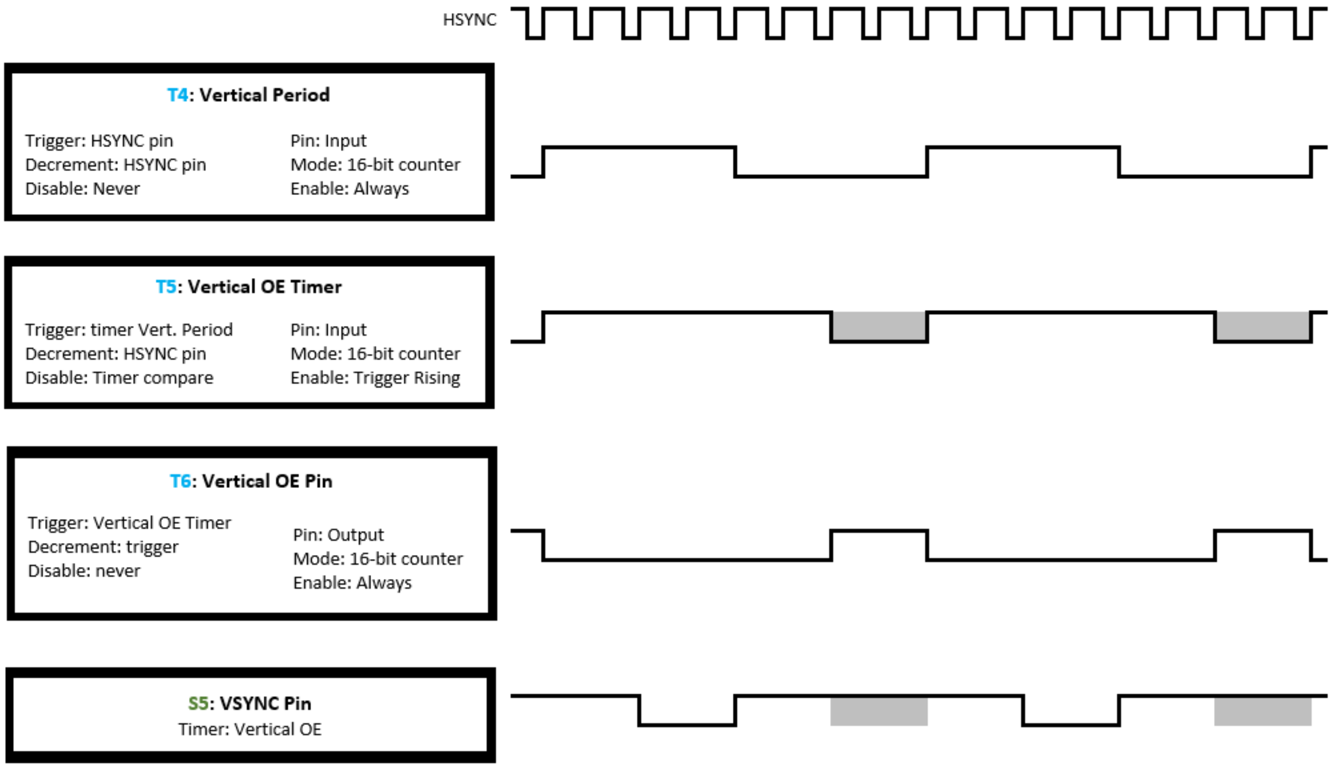


**Figure 9. Vertical Timing Signals**

The Vertical Period timer generates a vertical period event based on the VSYNC period. This period event is used by other vertical timers and shifters. The period event is generated using the 16bit counter mode of the timer, where the rising edge of the timer output triggers a new vertical period. Vertical timing is based on a number of HSYNC pulses, so this timer uses the HSYNC pin as an input and is decremented by edges on that pin. This timer is always enabled, and the timer output rising edge will be used as a trigger for the remaining vertical timing components. The value for the timer compare register is calculated by the flexio_lcd driver, adding the following parameters from the driver configuration:

- Number of rows per frame

- VSYNC Front Porch duration
- VSYNC Back Porch duration
- VSYNC pulse width

The next timer is the Vertical Output Enable Timer. This timer generates the vertical component of the Output Enable signal. It is enabled when the OE signal is disabled around the VSYNC pulse. This timer is used to control the shifting of the VSYNC pin shifter discussed below. It is enabled every vertical period, triggered by the Vertical Period Timer, and disables when OE is enabled for a new frame. The output of this timer is inverted from the Output Enable polarity pin because its output enables the VSYNC Pin Shifter.

The Vertical OE Pin Timer drives an output pin for the vertical component of the OE signal. The output of this timer is the inverted value of the trigger timer: the Vertical Output Enable Timer. This timer is enabled and disabled by that same trigger timer. The pin is set as an output, and the polarity is active low to invert the output from the trigger timer. The result is an output pin for the vertical component of the OE signal.

The VSYNC Pin Shifter shifts out the state of the VSYNC pin. It is active when the Vertical Output Enable Timer is enabled, and the shifting is controlled by that timer. The timing of the VSYNC pulse width, and front and back porch are controlled by loading a value in the shift register and shifting it out the pin. Each bit in the shifter value represents the state of the VSYNC pin for each HSYNC period. The flexio_lcd driver calculates this shift register value based on the driver configuration.

## 4.3. Output Enable and Pixel Data

With the horizontal and vertical timing signals generated, the Output Enable signal can be generated and control the parallel pixel data outputs. *Figure 10* shows the timing for these signals. The Output Enable signal is the logical AND of the horizontal and vertical OE pins. The timing diagram shows the different low pulses of the OE signal, which are around the HSYNC or VSYNC pulses. The Data Timer controls the Data Shifters, and is enabled/disabled by the OE Pin. The diagram shows when this timer is disabled with the gray shading. The Data Shifters shift out the parallel pixel color data to the LCD.  These shifters are active when the Data Timer is enabled, shaded in black. When active, the shifters shift out pixel data on ever pixel clock rising edge. The gray shading shows when the shifters are inactive because the OE Pin is disabled/low. There is no activity on the pixel data pins during that time.

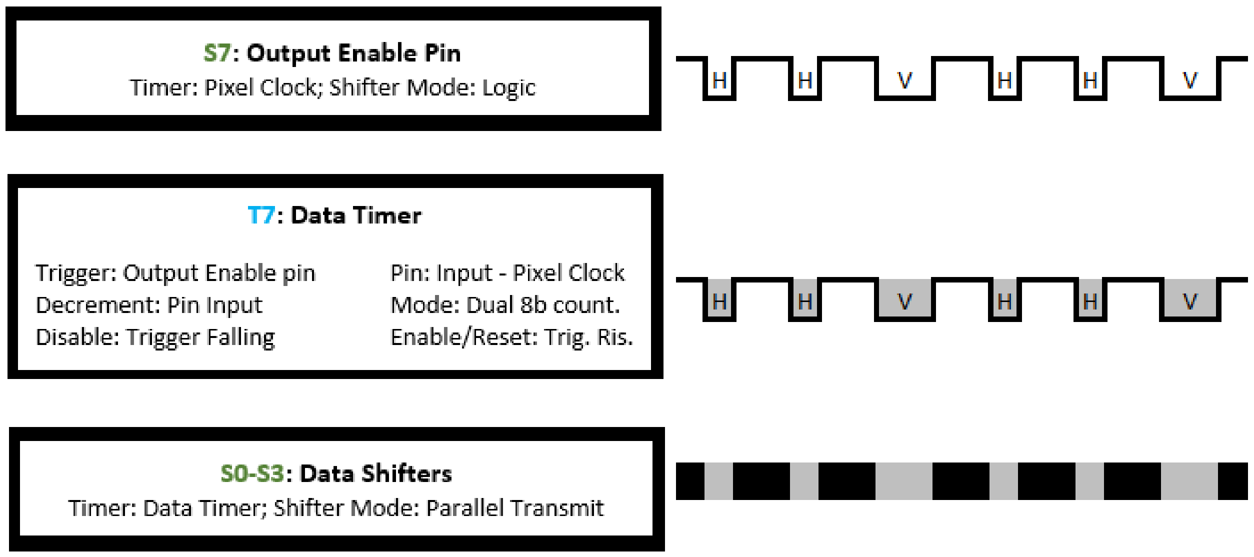


**Figure 10. Output Enable and Pixel Data timing**

This example uses multiple Data Shifters chained together. Shifter0 drives the 16 parallel output pins with the pixel data. Four shifters are used for the pixel data, with Shifter1 feeding Shifter0, and incrementing up to Shifter 3. This means the shifters together provide 128bits of pixel data. The more shifters chained for the data, the lower the load frequency required to write to the shift buffers. In this case, 128bits of data can be written to the shift buffers with each DMA trigger event. Shifter0 shifts out 16bits with each shift. Therefore, the Data shifters will load a pixel group of 8 pixels with each shifter load event.

The Output Enable Pin Shifter is used to combine the horizontal and vertical components of the OE signal into the final OE signal for the LCD. FlexIO shifters support a Logic Mode, which is a logic table with 5 inputs and 32 possible outputs. This example uses two of those inputs for the horizontal and vertical OE pins. In this Logic Mode, the FlexIO shift buffer stores the logic look-up table. The value used for the shift buffer logically ANDs the two input signals. Also in this logic mode, the shifter SHIFTCFG SSTART and SSTOP bit fields are used to mask unused inputs to the logic table. Since this example only uses 2 of the 5 potential inputs, these masks are used to mask the 3 unused inputs.

One note when using the flexio_lcd driver is that with Logic Mode, the input and output pins for the logic table are fixed based on the shifter number used. In this example, the output OE pin is on FXIO_D15. That output pin requires using Shifter 7 for this logic table, and then the input pins to the logic table are limited to FXIO_D[11-14]. This example has the Horizontal OE Pin on FXIO_D12, and the Vertical OE Pin is on FXIO_D14. These two pins are only used internally, and not routed outside the MCU. The logic table for this configuration is included in *Appendix B, Output Enable Pin Shifter Logic Table*. For more details on using Logic Mode, refer to the MCU Reference Manual.

The Data Timer controls the Data Shifters, and is enabled by the Output Enable pin. It will only shift when Output Enable is high. Also, the timer counter is reset by the OE rising edge to start a new horizontal row. It is clocked from the Pixel Clock pin, and uses Dual 8bit Counter mode to control the shift timing. Because the timer decrements on both edges of the pixel clock, the lower 8bits of the compare register are set to 1 to shift on every rising edge of the pixel clock. The upper 8bits control how many pixels are shifted per loading of the shift buffers, which is 8 in this example. This value for the

Timer compare register is calculated by the flexio_lcd driver using the number of data shifters and the parallel data bus width provided in the driver configuration.

The Data Shifters drive the parallel data bus to the LCD, and are chained together. They are controlled by the Data Timer. The shift buffers are loaded with the pixel color data values from the frame buffer by the DMA. The Data Shifters all have identical configurations, except that Shifter0 is the only shifter that drives the output pins. The other shifters are chained together and feed Shifter0.

The FlexIO start and stop bit features are also used with the Data Timer and Shifters to generate the correct pixel data timing for the LCD. These start/stop bit controls are typically used when the FlexIO emulates serial communications. But the added flexibility in the timing from these bits helps in this example. If the start/stop bits are disabled in this example, there is an issue with the pixel data when OE is disabled between horizontal rows. At the end of the row, when the Data Shifters output the last pixel, they will also load a new group of pixels from the shift buffers. But the Data Timer is also disabled at the end of the row with OE going low. When OE goes back high at the start of the next row, the Data Timer is re-enabled, and forces another load of pixel data to the Data Shifters. This double-load means the pixel group loaded at the end of the previous row was lost and never shifted out to the LCD. The result is a distorted image on the LCD, because a group of pixels is skipped with each row.

To avoid this issue, the start/stop bits are used. With this change in timing, the shifters are not loaded at the end of the row. Without the double-loading, the pixel group is no longer skipped. The flexibility with the FlexIO start/stop bits is another useful feature of this peripheral. When using the FlexIO to shift data with specific timing requirements, consider the different options for the start/stop bits to meet the required timing. Here is a summary of start/stop bit settings used for the Data Timer and Data Shifters:

- Data Timer Start Bit - enabled
- Data Timer Stop Bit – enable on timer disable
- Data Shifter Start Bit – disabled, load data on first shift
- Data Shifter Stop Bit – enabled, value 0 (the stop bit value is not important in this example)

With these settings for the FlexIO timers and shifters, the FlexIO is able to generate all the signal timing and pixel data to send the full frame to the LCD.

## 5. FlexIO_LCD Bus Bandwidth Performance

The FlexIO and DMA can be configured to drive LCDs of different resolutions, color depth, and pixel clock frequencies. The performance required to drive larger LCDs is limited by the bus bandwidth moving the pixel data from the frame buffer to the FlexIO peripheral. *Understanding LCD Memory and Bus Bandwidth Requirements* (document AN3606) discusses in more detail how bus bandwidth of the system limits the type of LCD that can be driven.

The FlexIO_LCD maximum bus bandwidth from SDRAM in the Kinetis K8x family is 31 MBytes/s. This performance is achieved using SDRAM with a 16-bit data port, clocked at the maximum 60 MHz, and a CAS Latency of 2, like the TWR-K80F150M development board. This performance assumes using the eDMA to transfer bursts of 16Bytes from SDRAM to four FlexIO shift buffers. Based on this maximum bandwidth of 31 MB/s, *Table 1* shows the maximum FlexIO_LCD pixel clock frequencies for different LCD color depths in bits per pixel (bpp).

**Table 1. FlexIO_LCD Maximum Pixel Clock Frequency**

| LCD Color Depth (bpp) | FlexIO_LCD Maximum Pixel Clock Frequency (MHz) |
|---|---|
| 17 - 32 | 7.7 |
| 9 - 16 | 15.5 |
| 8 | 31.0 |
| 4 | 61.9 |

This maximum bus bandwidth can be used with LCD parameters to calculate the percentage of bus bandwidth required for that LCD, see *Equation 1*. In this example using the TWR-LCD-RGB display, the pixel clock is 9.23 MHz (120 MHz system clock / 13). The color depth is 16 bpp. Therefore, this LCD setup requires 18.5 MB/s bus bandwidth, which is 59.6 % of the maximum bus bandwidth.

**Equation 1: LCD Bus Bandwidth Percentage**

$$\text{LCD Bus Bandwidth Percentage (\%)} = \frac{\text{(bpp in memory/8) x pixel clock frequency (MHz)}}{\text{Maximum Bus Bandwidth (MB/s)}}$$

# 6. FlexIO_LCD Driver

This example includes a driver to help abstract the application from configuring the FlexIO and eDMA registers directly. The driver is based on the Kinetis Software Development Kit (KSDK) v1.3.0, and is designed to drop into the KSDK installation. There is a broad selection of LCDs available, with different control signals and timing requirements. This driver will likely not support all LCDs as is, but it is provided here as a reference, and can be modified as needed for other LCDs.

The driver uses a configuration structure to configure the driver. This structure is initialized by the application and passed as an argument to the driver APIs, similar to other KSDK drivers. With this configuration, the driver has everything needed to configure the FlexIO and eDMA peripherals. The table below shows the parameters included in the driver configuration.

**Table 2. FlexIO_LCD driver configuration parameters**

| Parameter | Description |
|---|---|
| flexioInstance | Instance of FlexIO peripheral, in case the MCU has multiple FlexIO peripherals |
| flexioBase | FlexIO module base address for the registers, used by the FlexIO HAL |
| datPinStartIdx | Lowest number of FlexIO pins used for pixel data. This configures the starting pin for the parallel data bus. |
| datPinWidth | Data bus width for flexio_lcd. Data bus will use FlexIO pins from datPinStartIdx to (datPinStartIdx + datPinWidth -1) |
| pclkPinIdx | FlexIO pin number for Pixel clock pin (PCLK) |
| hsyncPinIdx | FlexIO pin number for Horizontal sync pin (HSYNC) |
| vsyncPinIdx | FlexIO pin number for Vertical sync pin (VSYNC) |
| oePinIdx | FlexIO pin number for Output Enable pin (OE) |
| horizOEPinIdx | FlexIO pin number for internal pin used for horizontal Output Enable |
| vertOEPinIdx | FlexIO pin number for internal pin used for vertical Output Enable |
| oeLogicTable | Shift buffer value for OE logic table |
| pclkFreq | Pixel Clock Frequency, in Hz |
| pixelsX | Pixels in X horizontal dimension |
| pixelsY | Pixels in Y vertical dimension |
| bytesPerPixel | Bytes per pixel |
| shifterStartIdx | First shifter index used for pixel data |
| numDataShifters | Number of Shifters for pixel data chained together. Starts with shifterStartIdx |
| timerStartIdx | Lowest FlexIO Timer index used for flexio_lcd |
| hsyncPulseWidth | in PCLK periods |
| hsyncFrontPorch | in clock periods before HSYNC pulse |
| hsyncBackPorch | in clock periods after HYSNC pulse |
| vsyncPulseWidth | in HSYNC pulses |
| vsyncFrontPorch | in HSYNC pulses before VSYNC pulse |

Table 2. FlexIO_LCD driver configuration parameters

| Parameter | Description |
|---|---|
| | |
| vsyncBackPorch | in HSYNC pulses after VSYNC pulse |
| dmaChannel | DMA channel used with FlexIO |
| dmaIRQ | DMA IRQ number for selected channel |

Once the driver configuration is initialized, the application uses two driver functions. FLEXIO_LCD_Init() initializes the FlexIO and eDMA peripherals, and calls the pin mux function for the FlexIO pins. FLEXIO_LCD_Start() starts the eDMA and FlexIO peripherals. After calling this function, the frame buffer will be displayed on the LCD. The arguments for both of these functions are a pointer to the driver configuration structure, and a pointer to the frame buffer. These functions are included in flexio_lcd.c.

The function FLEXIO_LCD_Init() calls the Hardware Abstract Layer (HAL) function FLEXIO_LCD_HAL_Configure(), which configures the FlexIO peripheral, including all the timers and shifters. This function is in flexio_lcd_hal.c. Also, the eDMA is configured in FLEXIO_LCD_DRV_eDma_Init(), located in flexio_lcd_edma_driver.c.

In addition to configuring the hardware, the flexio_lcd driver includes some test patterns to help test the system and help troubleshoot. The example application is written to display these patterns, and cycle through them using a push button. The pattern functions are located in flexio_lcd.c. These functions draw the patterns in the frame buffer using software, and are much smaller than storing images for the display. In this example, they are stored in internal flash, and do not require the QuadSPI flash.

The pattern function FLEXIO_LCD_FrameColorBars() draws a color bar test pattern, using 8 colors. This pattern helps to confirm the LCD colors look correct. The function FLEXIO_LCD_FrameBlackWhiteColumns() alternates pixel columns with black and white pixels. This pattern stresses the memory bus from the SDRAM, as the data bits toggle from all 1s to all 0s at a high frequency. This will help identify board issues, like a noisy bus or ground bounce. The last pattern function is FLEXIO_LCD_FrameVerticalStripes(), which alternates vertical blue and yellow stripes. The stripes are 8 pixels wide. This pattern is used because of the how data is written from the frame buffer to the FlexIO shift buffers. In this example, four 32-bit shifters are used for the pixel data, which is data for 8 pixels. Since these vertical stripes change color on FlexIO write boundaries, the pattern helps show if there are issues related to the FlexIO timing, like underruns when data writes from the frame buffer take too long.

# 7. Using flexio_TWR-LCD-RGB Example Application

This example application uses NXP development hardware and enablement software driving a graphical LCD using the flexio_lcd driver. The application stores the frame buffer in external SDRAM. A QuadSPI flash is also used to store the images displayed on the LCD. The QuadSPI provides plenty of storage space for all the images and graphics needed for a GUI.

To replicate the demo, the hardware and software listed below are utilized. The hardware requires some modifications to route the FlexIO signals to the LCD. The rest of this section will provide details for configuring the hardware, building the example project, programming the flash, and running the example.

Hardware for example:

- TWR-LCD-RGB – LCD Module for Tower System
- TWR-K80F150M – K80 Tower Board, Schematic RevB1 or later
- TWR-PROTO – prototyping board used to route signals between K80 and LCD
- TWR-ELEV – Tower elevator boards

Software for example:

- Kinetis Software Development Kit (KSDK) v1.3.0 mainline release
- IAR Embedded Workbench for ARM®– tested with v7.50.2
- Segger OpenSDA V2.1 app – on-board debugger app for the TWR-K80F150M board
- Included example project

## 7.1. Configure the Hardware

The TWR-K80F150M and TWR-PROTO boards require some modifications to route the K80 FlexIO signals to the LCD on the TWR-LCD-RGB board. Then the Tower can be assembled for the example. The TWR-LCD-RGB and TWR-ELEV boards require no changes for this example.

### 7.1.1. TWR-K80F150M Board Setup

The example project is setup to use the Segger JLink OpenSDA V2.1 debugger application for the on-board debugger. This is not the default OpenSDA debugger programmed on the board. To update the OpenSDA app, hold down the Reset button on the TWR-K80F150M board when connecting the USB cable to the OpenSDA USB connector J24. This forces the OpenSDA into bootloader mode, and enumerates in the PC as a new BOOTLOADER drive. Then, in the PC, drag-and-drop the Segger OpenSDA application file to the BOOTLOADER drive. After copying the file, unplug the board and plug it back in. Now the OpenSDA will enumerate as the Segger JLink debugger.

To connect the FlexIO signals from the K80 to the LCD, short the following pads for these non-populated resistors:

Table 3. TWR-K80F150M Resistor Pads to short

| Resistor | Elevator Signal | Demo Signal |
|---|---|---|
| R106 | D8 | FXIO0_D31 |
| R111 | D38 | FXIO0_D30 |
| R112 | D39 | FXIO0_D29 |
| R113 | D40 | FXIO0_D28 |
| R295 | C38 | FXIO0_D26 |
| R299 | C37 | FXIO0_D27 |
| R31 | B40 | FXIO0_D16 |
| R32 | B39 | FXIO0_D17 |
| R34 | B37 | FXIO0_D11 |

Some components need to be removed from the TWR-K80F150M board to prevent contention with other signals.

Table 4. TWR-K80F150M Remove Components

| Component | Elevator Signal | Demo Signal | Reason |
|---|---|---|---|
| R298 | C7 | FXIO0_D17 | Conflicts with TWR-LCD-RGB I2C |
| R288 | C8 | FXIO0_D16 | Conflicts with TWR-LCD-RGB I2C |

**NOTE**

If the board schematic revision is RevB or older, there is an additional component to remove. The capacitor C11 needs to be removed, as it conflicts with the SDRAM. Production versions of this board RevB1 and later already have this capacitor removed.

The following jumper settings need to be changed from their default configuration.

Table 5. TWR-K80F150M Jumper Changes

| Jumper | Setting | Reason |
|---|---|---|
| J6 | Remove | UART signal conflicts with SDRAM clock signal |
| J8 | Remove | UART signal conflicts with SDRAM address signal |

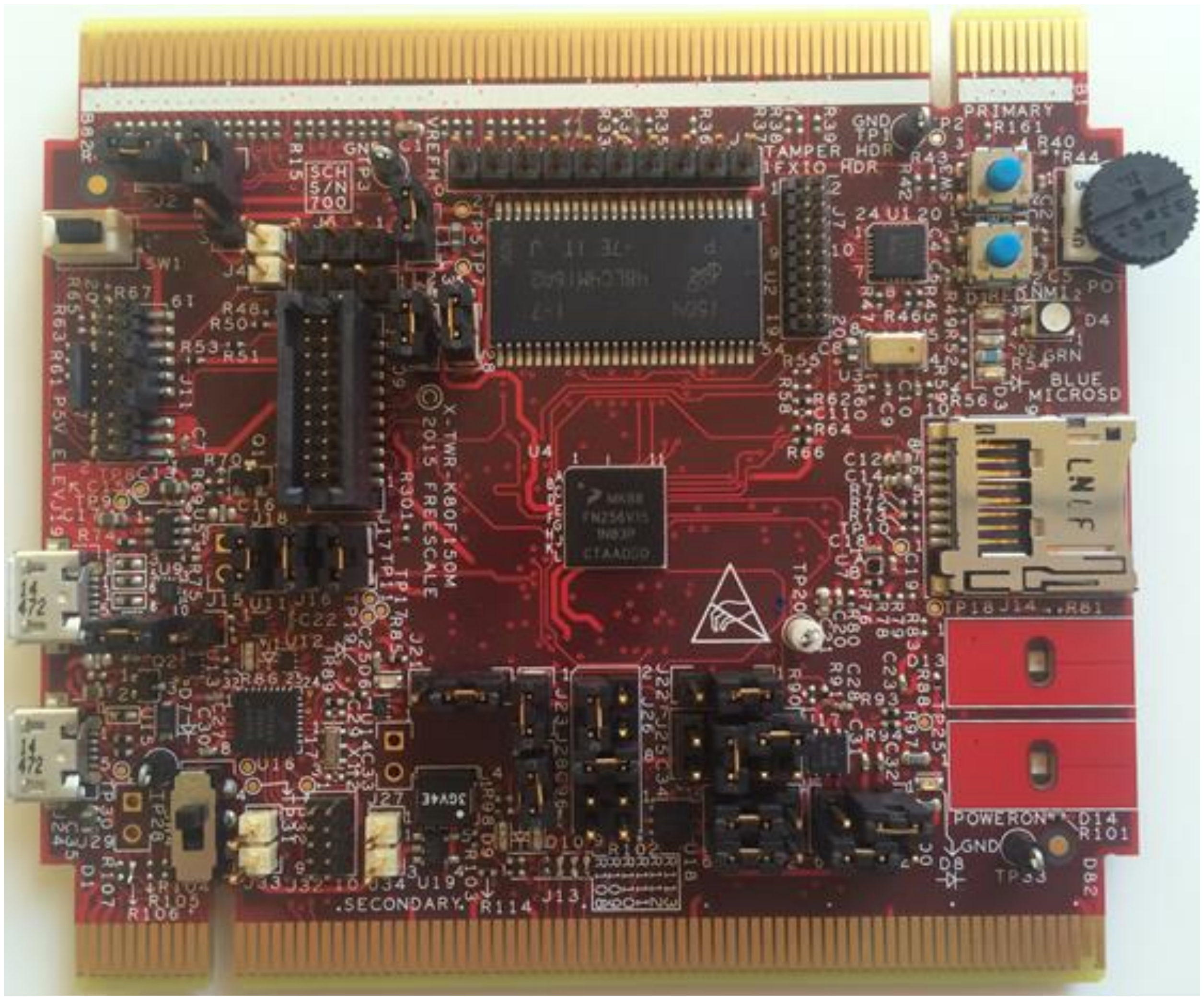


**Figure 11. TWR-K80F150M prepared for example**

### 7.1.2. TWR-PROTO Board Setup

In the Tower system, signals are routed between boards through the TWR-ELEV boards. These Tower pins provide access to connect or re-route these signals as needed. One method to connect signals is using the TWR-PROTO board, and shorting different Tower pins together. For this example, the TWR-PROTO board is used to make the connections between the K80 FlexIO signals and the TWR-LCD-RGB signals. *Table 6* shows all of the connections required on the TWR-PROTO board in the two right columns. For example, on the TWR-PROTO board, add a wire to short pin B37 to D32. The example uses a 16-bit interface to the LCD, using the RGB565 format. The LCD supports a 24-bit interface, so the LSB pins of each color of the LCD are tied to ground since they are unused.

**Table 6. Required TWR-PROTO Connections**

| | | | Connect these two signals together on TWR-PROTO board | |
|---|---|---|---|---|
| **TWR-LCD-RGB Signal** | **FlexIO Signal** | **K80 Port Pin** | **K80 Tower Pin** | **TWR-LCD-RGB Tower Pin** |
| RGB_CLK | FXIO0_D11 | PTA1 | B37 | D32 |
| RGB_HSYNC | FXIO0_D7 | PTB19 | B72 | D27 |
| RGB_VSYNC | FXIO0_D6 | PTB18 | B66 | D28 |
| RGB_OE | FXIO0_D15 | PTA5 | A22 | D44 |
| RGB_R7 | FXIO0_D31 | PTD15 | D8 | C80 |
| RGB_R6 | FXIO0_D30 | PTD14 | D38 | D80 |
| RGB_R5 | FXIO0_D29 | PTD13 | D39 | D79 |
| RGB_R4 | FXIO0_D28 | PTD12 | D40 | D78 |
| RGB_R3 | FXIO0_D27 | PTD11 | C37 | D64 |
| RGB_R2 | VSS | VSS | GND | D63 |
| RGB_R1 | VSS | VSS | GND | C64 |
| RGB_R0 | VSS | VSS | GND | C63 |
| RGB_G7 | FXIO0_D26 | PTD10 | C38 | C62 |
| RGB_G6 | FXIO0_D25 | PTD9 | A8 | D54 |
| RGB_G5 | FXIO0_D24 | PTD8 | A7 | D53 |
| RGB_G4 | FXIO0_D23 | PTA17 | A21 | D52 |
| RGB_G3 | FXIO0_D22 | PTA16 | A59 | C57 |
| RGB_G2 | FXIO0_D21 | PTA15 | A24 | C56 |
| RGB_G1 | VSS | VSS | GND | C55 |
| RGB_G0 | VSS | VSS | GND | C54 |
| RGB_B7 | FXIO0_D20 | PTA14 | A42 | C53 |
| RGB_B6 | FXIO0_D19 | PTA13 | B61 | C52 |
| RGB_B5 | FXIO0_D18 | PTA12 | B62 | C51 |
| RGB_B4 | FXIO0_D17 | PTA11 | B39 | C50 |
| RGB_B3 | FXIO0_D16 | PTA10 | B40 | D48 |
| RGB_B2 | VSS | VSS | GND | D47 |
| RGB_B1 | VSS | VSS | GND | D46 |
| RGB_B0 | VSS | VSS | GND | D45 |

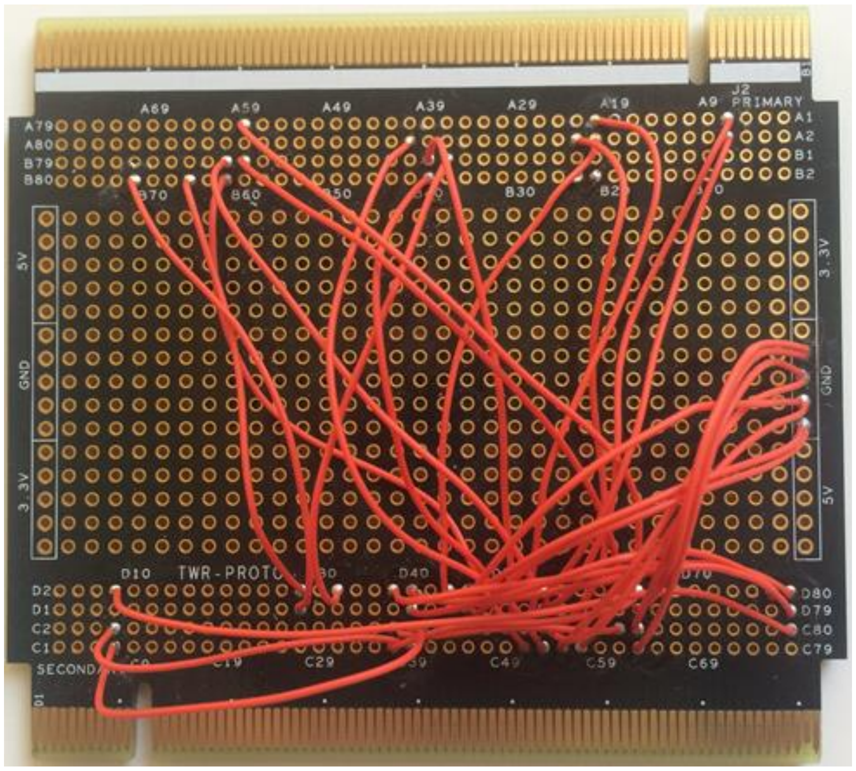


Figure 12. TWR-PROTO board with Tower connections

### 7.1.3. Assemble the Tower System

After the hardware modifications are complete, the Tower system can be assembled. Ensure the TWR-LCD-RGB module is plugged into the secondary elevator board, which has the black edge connectors. The TWR-LCD-RGB is already connected to this elevator board when shipped, so there is likely no change required for this step. Next, plug the TWR-K80F150M and TWR-PROTO boards into the primary elevator card. Make sure the edge of these boards with the white stripe on the silk screen is plugged into the white edge connectors of the primary elevator board. Then plug the secondary elevator board on the other side of the Tower. *Figure 13* shows an image of the assembled Tower system.

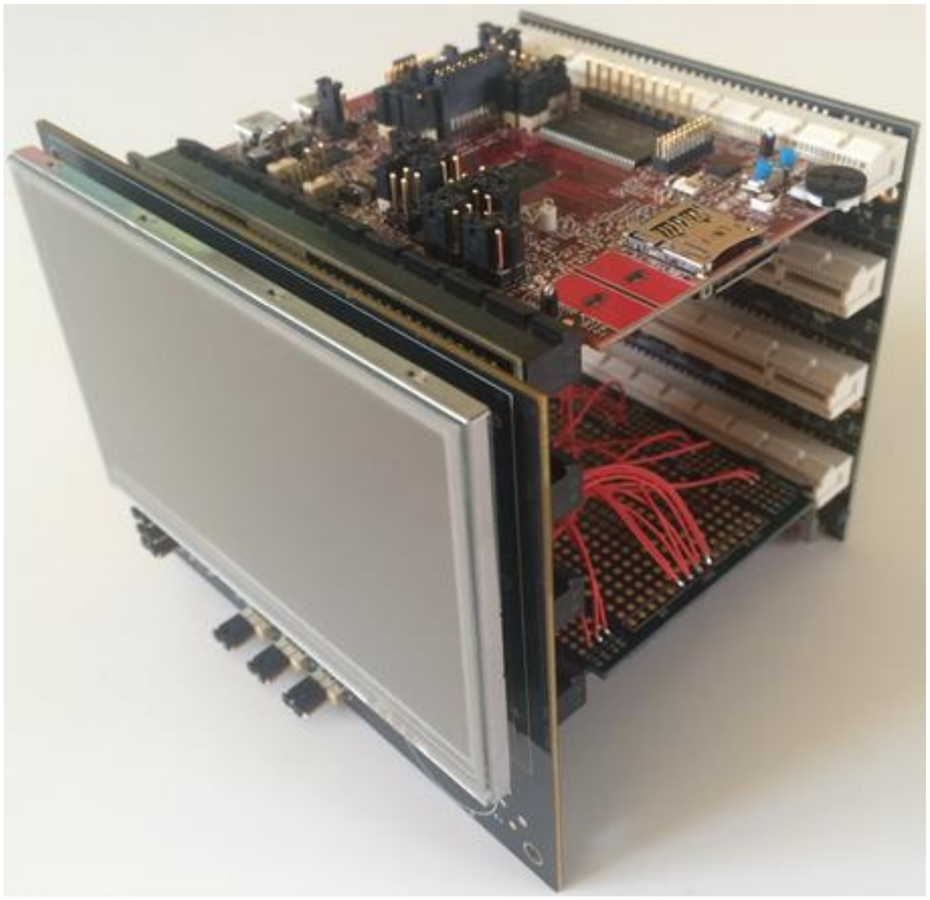


Figure 13. Assembled Tower system

## 7.2. Build the Example Project

The example is built on KSDK v1.3.0 mainline release. Download and install the KSDK first. Then the compressed example file should be extracted to the root KSDK directory. By default, this would be the path C:\Freescale\KSDK_1.3.0. This will add the flexio_TWR-LCD-RGB example directory to the demo_apps directory.

Then, open the IAR workspace file. If using the default install location, the workspace file is located at C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-RGB\iar\flexio_TWR-LCD-RGB.eww

In IAR, do a batch build to build both the KSDK platform library and the example project. Use the menu Project->Batch Build. Select the Debug batch, and click the Make button.

When done, the application will be ready to load in the flash.

## 7.3. Program the Flash

The example uses the QuadSPI flash to store the images displayed on the LCD. The ROM bootloader in the K80 is used to program the QuadSPI. This example includes a batch file that will connect with the ROM bootloader through USB, and program both the QuadSPI and internal flash.

Programming the QuadSPI can take some time depending on the amount of memory to program. But if the images are the only memory stored in QuadSPI, then the QuadSPI only needs to be updated when adding new images or modifying existing images. If the images are already in the QuadSPI, and the application code is changed, then the IAR debugger can be used to program the internal flash and debug the application. This will save time compared to running the batch file again. And this example will also work without programming the QuadSPI flash. Instead of displaying images from QuadSPI, the example can also draw patterns in the frame buffer to display.

The following sections describe how to program the flash using both methods. To see the images included in the example, be sure to run the batch file at least once.

### 7.3.1. Batch file for ROM Bootloader to program QuadSPI and Internal Flash

First, the ROM bootloader on the K81 needs to be started. The USB interface to the bootloader is used with the batch file. The best option is to use two micro-USB cables, both connected to the TWR-K80F150M board. To force bootloader mode, hold down the blue button SW2. While holding SW2, press and release the black reset button SW1. Then release SW2. Now the ROM bootloader should be started.

Next, run the batch file to program both the QuadSPI and the internal flash with the example application image. The batch file is located at C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps \flexio_TWR-LCD-RGB\iar\flash_QuadSPI.bat. Programming the QuadSPI with these example images will take a few minutes.

When the batch file is done, press the reset button on the TWR-K80F150M. The ROM Bootloader will run for 5 seconds, and then it will start the application. At this point, an image should appear on the LCD. If that is not the case, see *Appendix D Troubleshooting Tips.*

### 7.3.2. Programming Internal Flash with IAR Debugger

If the QuadSPI does not need to be programmed then the IAR debugger can be used to program the internal flash and debug the application. Ensure that a USB cable is connected to the OpenSDA USB connector J24. Then in IAR, click the Download and Debug button on the toolbar. With this project, the IAR flash loader will give a warning with a pop-up, and will also show an error in the Debug Log. These warnings are caused because the IAR flash loader sees the memory intended for the QuadSPI, but does not know what to do with that data. The flash loader will still be able to download successfully to the internal flash. When the warning pops-up, click the OK button, and the debugger will continue and flash the K80. *Appendix C, IAR Debug Log when programming flash* shows the log after successfully programming the internal flash and connecting with the debug, including these warnings and errors.

## 7.4. Run the Example

After the flash is programmed, the example application can run, and will show an image on the LCD. If the QuadSPI is programmed with the images, one of these images will be the default image on the display. Otherwise, the application will draw one of the driver patterns on the display. With multiple images in the QuadSPI, the example will do a slide show, cycling through each image periodically. The blue button SW2 on the TWR-K80F150M board manually cycles through the images in the QuadSPI. The blue SW3 button manually cycles through the driver patterns. If these buttons are pressed, the slide show will stop changing images automatically. The example uses a 16-bit interface to the LCD, using the RGB565 format.

One note about this application and the debug console. KSDK applications typically include the debug console utility for printf() and other stdio functions. With the TWR-K80F150M, the UART signals connected to the OpenSDA circuit conflict with the SDRAM signals. Since this application uses the SDRAM for the frame buffer, the UART signals are disconnected by removing jumpers J6 and J8. Therefore, a terminal cannot be connected to the debug console. Different UART pins with some additional hardware could be connected for that feature if needed.

## 8. Conclusion

FlexIO is a powerful and flexible peripheral that can be useful in many applications, including driving a graphical LCD. The flexibility with the timers, shifters, and pins allows the FlexIO to interface with many different types of displays, and other peripherals. There is a broad offering of Kinetis MCUs with FlexIO, and the list continues to grow. The flexio_lcd driver simplifies configuring the hardware for the application. These Kinetis features and software resources enable a wide range of applications, with quick time-to-market. Visit www.nxp.com to learn more.

## 9. Revision History

**Table 7. Revision history**

| Revision number | Date | Substantive changes |
|---|---|---|
| 1 | 06/2016 | • In *Section 7.4:*<br>– Added text to first paragraph “With multiple images in the QuadSPI, the example will do a slide show, cycling through each image periodically. The blue button SW2 on the TWR-K80F150M board manually cycles through the images in the QuadSPI. The blue SW3 button manually cycles through the driver patterns. If these buttons are pressed, the slide show will stop changing images automatically”.<br>• In *Appendix A:*<br>– Results updated for latest batch file. |
| 0 | 04/2016 | Initial release |

# Appendix A. flash_QuadSPI.bat Results

This appendix is a listing of the command line output from running the flash_QuadSPI.bat batch file. It shows the results of a successful programming of the flash using the ROM bootloader, and may be helpful troubleshooting any issues. To see these results, run the batch file from the command line in Windows. That way, the results can be viewed after the batch file completes.

```
C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\elftosb.exe -V -c flexio_TWR-LCD-RGB.bd -o flexio_TWR-LCD-
RGB.sb
Boot Section 0x00000000:
  LOAD | adr=0x68000000 | len=0x00000200 | crc=0xf7185382 | flg=0x0000
  LOAD | adr=0x00000000 | len=0x000003c0 | crc=0x0c4c6b8f | flg=0x0000
  LOAD | adr=0x00000400 | len=0x00006190 | crc=0xdb549d5f | flg=0x0000
  LOAD | adr=0x68001000 | len=0x0007f808 | crc=0xdca5212b | flg=0x0000

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost -u -- flash-erase-all-unsecure
Inject command 'flash-erase-all-unsecure'
Successful generic response to command 'flash-erase-all-unsecure'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost -u -- write-memory 0x20000000
.\..\qspi_config_block.bin
Inject command 'write-memory'
Preparing to send 512 (0x200) bytes to the target.
Successful generic response to command 'write-memory'

(1/1)100% Completed!
Successful generic response to command 'write-memory'
Response status = 0 (0x0) Success.
Wrote 512 of 512 bytes.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost -u -- configure-quadspi 1 0x20000000
Inject command 'configure-quadspi'
Successful generic response to command 'configure-quadspi'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- flash-erase-region 0x68000000 0x80000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- flash-erase-region 0x68080000 0x80000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
```

```
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- flash-erase-region 0x68100000 0x80000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- flash-erase-region 0x68180000 0x80000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- flash-erase-region 0x68200000 0x80000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- flash-erase-region 0x68280000 0x80000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- flash-erase-region 0x68300000 0x80000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- flash-erase-region 0x68380000 0x80000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>echo 10:49:56.88
10:49:56.88

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>.\..\bin\blhost.exe -u -- receive-sb-file flexio_TWR-LCD-RGB.sb
Inject command 'receive-sb-file'
Preparing to send 548928 (0x86040) bytes to the target.
Successful generic response to command 'receive-sb-file'

(1/1)100% Completed!
Successful generic response to command 'receive-sb-file'
Response status = 0 (0x0) Success.
```

```
Wrote 548928 of 548928 bytes.

C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar>echo 10:54:31.44
```

# Appendix B. Output Enable Pin Shifter Logic Table

Below is the logic table used for the Output Enable Pin shifter. Based on this table, the shift buffer SHIFTBUF7 is set to 0xCC00CC00.

**Table 8. Output Enable Pin Shifter Logic Look-up Table**

| **SHIFTERi[0]**<br>unused | **FXIO_D[x+3]**<br>vertOEPinIdx<br>FXIO_D[14] | **FXIO_D[x+2]**<br>unused | **FXIO_D[x+1]**<br>horizOEPinIdx<br>FXIO_D[12] | **FXIO_D[x]**<br>unused | **Logic Output to FXIO_D[x+4]**<br>oePinIdx<br>FXIO_D[15] | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | SHIFTBUFi[ 0 ] | 0 |
| 0 | 0 | 0 | 0 | 1 | SHIFTBUFi[ 1 ] | 0 |
| 0 | 0 | 0 | 1 | 0 | SHIFTBUFi[ 2 ] | 0 |
| 0 | 0 | 0 | 1 | 1 | SHIFTBUFi[ 3 ] | 0 |
| 0 | 0 | 1 | 0 | 0 | SHIFTBUFi[ 4 ] | 0 |
| 0 | 0 | 1 | 0 | 1 | SHIFTBUFi[ 5 ] | 0 |
| 0 | 0 | 1 | 1 | 0 | SHIFTBUFi[ 6 ] | 0 |
| 0 | 0 | 1 | 1 | 1 | SHIFTBUFi[ 7 ] | 0 |
| 0 | 1 | 0 | 0 | 0 | SHIFTBUFi[ 8 ] | 0 |
| 0 | 1 | 0 | 0 | 1 | SHIFTBUFi[ 9 ] | 0 |
| 0 | 1 | 0 | 1 | 0 | SHIFTBUFi[ 10 ] | 1 |
| 0 | 1 | 0 | 1 | 1 | SHIFTBUFi[ 11 ] | 1 |
| 0 | 1 | 1 | 0 | 0 | SHIFTBUFi[ 12 ] | 0 |
| 0 | 1 | 1 | 0 | 1 | SHIFTBUFi[ 13 ] | 0 |
| 0 | 1 | 1 | 1 | 0 | SHIFTBUFi[ 14 ] | 1 |
| 0 | 1 | 1 | 1 | 1 | SHIFTBUFi[ 15 ] | 1 |
| 1 | 0 | 0 | 0 | 0 | SHIFTBUFi[ 16 ] | 0 |
| 1 | 0 | 0 | 0 | 1 | SHIFTBUFi[ 17 ] | 0 |
| 1 | 0 | 0 | 1 | 0 | SHIFTBUFi[ 18 ] | 0 |
| 1 | 0 | 0 | 1 | 1 | SHIFTBUFi[ 19 ] | 0 |
| 1 | 0 | 1 | 0 | 0 | SHIFTBUFi[ 20 ] | 0 |
| 1 | 0 | 1 | 0 | 1 | SHIFTBUFi[ 21 ] | 0 |
| 1 | 0 | 1 | 1 | 0 | SHIFTBUFi[ 22 ] | 0 |
| 1 | 0 | 1 | 1 | 1 | SHIFTBUFi[ 23 ] | 0 |
| 1 | 1 | 0 | 0 | 0 | SHIFTBUFi[ 24 ] | 0 |
| 1 | 1 | 0 | 0 | 1 | SHIFTBUFi[ 25 ] | 0 |
| 1 | 1 | 0 | 1 | 0 | SHIFTBUFi[ 26 ] | 1 |
| 1 | 1 | 0 | 1 | 1 | SHIFTBUFi[ 27 ] | 1 |
| 1 | 1 | 1 | 0 | 0 | SHIFTBUFi[ 28 ] | 0 |
| 1 | 1 | 1 | 0 | 1 | SHIFTBUFi[ 29 ] | 0 |
| 1 | 1 | 1 | 1 | 0 | SHIFTBUFi[ 30 ] | 1 |
| 1 | 1 | 1 | 1 | 1 | SHIFTBUFi[ 31 ] | 1 |

# Appendix C. IAR Debug Log when programming flash

```
Thu Jan 28, 2016 14:15:38: Loaded macro file: C:\Program Files (x86)\IAR
Systems\Embedded Workbench 7.50.2\arm\config\debugger\Freescale\Kxx.dmac
Thu Jan 28, 2016 14:15:38: Flash download warning: 522248 out of 522248
bytes from data record CODE:[0x68001000,0x68080807] will not be flashed
Thu Jan 28, 2016 14:15:38: There were warnings while generating flash loader
input.
See the Debug Log window for details.
Thu Jan 28, 2016 14:17:13: Loaded macro file: C:\Program Files (x86)\IAR
Systems\Embedded Workbench
7.50.2\arm\config\flashloader\Freescale\FlashK80Xxxx256K.mac
Thu Jan 28, 2016 14:17:15: JLINK command: ProjectFile =
C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar\settings\flexio_TWR-LCD-RGB_Debug.jlink, return = 0
Thu Jan 28, 2016 14:17:15: Device "MK80FN256XXX15" selected.
Thu Jan 28, 2016 14:17:15: DLL version: V5.10d, compiled Dec  9 2015
12:49:14
Thu Jan 28, 2016 14:17:15: Firmware: J-Link OpenSDA 2 compiled Oct 13 2015
12:10:56
Thu Jan 28, 2016 14:17:15: Selecting SWD as current target interface.
Thu Jan 28, 2016 14:17:15: JTAG speed is initially set to: 32 kHz
Thu Jan 28, 2016 14:17:15: Found SWD-DP with ID 0x2BA01477
Thu Jan 28, 2016 14:17:15: Found SWD-DP with ID 0x2BA01477
Thu Jan 28, 2016 14:17:16: Found Cortex-M4 r0p1, Little endian.
Thu Jan 28, 2016 14:17:16: FPUnit: 6 code (BP) slots and 2 literal slots
Thu Jan 28, 2016 14:17:16: CoreSight components:
Thu Jan 28, 2016 14:17:16: ROMTbl 0 @ E00FF000
Thu Jan 28, 2016 14:17:16: ROMTbl 0 [0]: FFF0F000, CID: B105E00D, PID:
000BB00C SCS
Thu Jan 28, 2016 14:17:16: ROMTbl 0 [1]: FFF02000, CID: B105E00D, PID:
003BB002 DWT
Thu Jan 28, 2016 14:17:16: ROMTbl 0 [2]: FFF03000, CID: B105E00D, PID:
002BB003 FPB
Thu Jan 28, 2016 14:17:16: ROMTbl 0 [3]: FFF01000, CID: B105E00D, PID:
003BB001 ITM
Thu Jan 28, 2016 14:17:16: ROMTbl 0 [4]: FFF41000, CID: B105900D, PID:
000BB9A1 TPIU
Thu Jan 28, 2016 14:17:16: ROMTbl 0 [5]: FFF42000, CID: B105900D, PID:
000BB925 ETM
Thu Jan 28, 2016 14:17:16: Hardware reset with strategy 0 was performed
Thu Jan 28, 2016 14:17:16: Initial reset was performed
Thu Jan 28, 2016 14:17:17: ----- Prepare hardware for Flashloader -----
Thu Jan 28, 2016 14:17:17: 2304 bytes downloaded (18.00 Kbytes/sec)
Thu Jan 28, 2016 14:17:17: Loaded debugee: C:\Program Files (x86)\IAR
Systems\Embedded Workbench
7.50.2\arm\config\flashloader\Freescale\FlashK80Fxxx256K.out
Thu Jan 28, 2016 14:17:17: Target reset
Thu Jan 28, 2016 14:17:18: Unloaded macro file: C:\Program Files (x86)\IAR
Systems\Embedded Workbench
7.50.2\arm\config\flashloader\Freescale\FlashK80Xxxx256K.mac
```

```
Thu Jan 28, 2016 14:17:18: Downloaded
C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar\debug\flexio_TWR-LCD-RGB.out to flash memory.
Thu Jan 28, 2016 14:17:18: Hardware reset with strategy 0 was performed
Thu Jan 28, 2016 14:17:18: 546674 bytes downloaded into FLASH (308.23
Kbytes/sec)
Thu Jan 28, 2016 14:17:18: Loaded debugee:
C:\Freescale\KSDK_1.3.0\examples\twrk80f150m\demo_apps\flexio_TWR-LCD-
RGB\iar\debug\flexio_TWR-LCD-RGB.out
Thu Jan 28, 2016 14:17:18: Hardware reset with strategy 0 was performed
Thu Jan 28, 2016 14:17:18: Target reset
Thu Jan 28, 2016 14:17:19: There were 1 error and 1 warning during the
initialization of the debugging session.
```

# Appendix D. Troubleshooting Tips

## D.1. No image is displayed, only the driver patterns

This likely means the QuadSPI has not been programmed successfully. The firmware reads the image table stored in QuadSPI. If the first entry in that table is not programmed, the application will not attempt to show images, and will only draw the driver patterns. To resolve this, run the batch file to program the QuadSPI, and ensure the operations in the batch file complete successfully without any errors. See *Section 7.3.1, Batch file for ROM Bootloader to program QuadSPI and Internal Flash*.

## D.2. Display is all white

When the LCD pixels are all white, this means the LCD is not receiving some of the control signals: pixel clock, HSYNC, VSYNC or Output Enable. There are many potential reasons for this, and here is a list of things to check:

- Verify the Tower cards are connected to the correct elevator board. Each edge of the Tower cards can plug into either elevator board, and it is easy to get them reversed. See *Section 7.1.3, Assemble the Tower System.*

- Verify the TWR-K80F150M jumpers J6 and J8 are removed. If these jumpers are installed, they interfere with the SDRAM.

- On the TWR-K80F150M board, ensure capacitor C11 is removed. This capacitor will interfere with the SDRAM. On schematic RevB1 and later, this capacitor should not be populated, however on older boards it is.

- Try pushing the SW3 button to change the pattern. If the driver patterns display, but no images, then the QuadSPI flash is likely not programmed correctly. See *Appendix D.1, No image is displayed, only the driver patterns*.

- Use an oscilloscope to check if the four signals pixel clock, HSYNC, VSYNC, and OE are getting toggled as described in *Section 3, LCD Signals and Timing Overview*. Probe the Tower signals connected to the TWR-LCD-RGB board on the secondary elevator board. This may show which signal(s) has the issue. If some signals look good but not all, then it is likely a connection issue. Review the hardware setup steps and verify those connections. See *Section 7.1.1, TWR-K80F150M Board Setup* and *Section 7.1.2, TWR-PROTO Board Setup*.

- If none of the four control signals look correct, then it is likely an issue with the application. Connect the debugger and start the application to see if the application appears to run correctly. See *Section 7.3.2, Programming Internal Flash with IAR Debugger*.

- With the debugger, press button SW3 to draw a pattern in the frame buffer. Then halt the debugger and look at the memory in the frame buffer. If the memory doesn't appear to match the pattern, then it is likely a hardware issue with the SDRAM. Review these steps for the *TWR-K80F150M board setup*.

## D.3. Display shows colors, but patterns and images are not correct

This likely means there is an issue with the timing of the LCD control signals, and it is likely a hardware setup issue. See *Section D.2, Display is all white* for troubleshooting tips.

Document Number: AN5280
Rev. 1
06/2016