Freescale Semiconductor, Inc.
Application Note

Document Number: AN5239
Rev. 0, 01/2016

Emulating Hardware State Machine Using FlexIO Module

By: Rastislav Pavlanin

1. Introduction

The FlexIO module available on Kinetis devices is a highly configurable peripheral module which provides a lot of functions. Depending on the module version, it can:

- Emulate serial communication interfaces (UART, I^2C, SPI, I^2S, one wire, etc.).
- Emulate parallel interfaces (CMOS camera, Motorola 68k, Intel 8080, etc.).
- Generate user-defined complex timing waveforms and triggers.
- Create logic functions on inputs using the logic look-up table.
- Create hardware state machines.

The FlexIO module is available only on specific Kinetis devices, including Kinetis L and Kinetis K series. It is required to have a basic understanding of the FlexIO module to fully understand this document.

Contents

This application note describes the state mode of the FlexIO module, which enables you to create a hardware state machine to offload the CPU. This document demonstrates the state machine using an example of intersection traffic lights emulation, based on sensors providing information about the dedicated lane traffic jam or night mode. This document includes a brief description of the FlexIO state mode, traffic lights example, the FlexIO module example configuration, and the software used. This example completely offloads the CPU. The CPU is in an infinite loop without any interrupt handling, while the FlexIO module handles the traffic lights. The MCU can also enter the very-low-power mode to reduce current consumption as much as possible. In the case of the K80 device used in traffic light emulation, you can reduce the current consumption down to ~500 μA. You can further reduce the current consumption using the 4 MHz IRC and one FlexIO timer to generate 1 kHz clock for the other FlexIO resources. Use the lower external clock to rapidly reduce current consumption.

2. FlexIO Module State Mode

The FlexIO module state mode enables you to create a hardware state machine with up to eight states. Each state uses three selectable inputs and eight dedicated outputs. Such state machine offloads the CPU rapidly when compared to the software approach.

NOTE

The state mode is not available on FlexIO module version 1.0 (e.g., KL43Z256 MCU). It is only available on FlexIO modules version 1.1 or higher (e.g., K80FN256 MCU). The FlexIO module version is defined in the FLEXIOx_VERID register using the MAJOR and MINOR fields.

Figure 1 shows the I/O assignments for a given state in general. The three inputs to the state can be represented by a set of three FlexIO pins (starting from FLEXIOx_SHIFTCTLn[PINSEL] value) or by three LSB bits of the next (n+1) shifter. Select this using the input source selection field INSRC in the shifter configuration register FLEXIOx_SHIFTCFGn[INSRC]. Configure the inputs' PINPOL polarity in the shifter control register FLEXIOx_SHIFTCTLn[PINPOL] when the FlexIO pins are selected as inputs. The eight outputs from the state are fixed and assigned to FlexIO pins FXIO_D0-FXIO_D7 (all states share the same outputs). Disable the outputs FXIO_D0-FXIO_D7 partially by setting the adequate bits in the PWIDTH[3:0], SSTOP[1:0], and SSTART[1:0] fields in the shifter configuration register FLEXIOx_SHIFTCFGn (see Figure 1). Configure the behavior of the outputs (push-pull, open drain, and so on) using the PINCFG shifter pin configuration field in the shifter control register FLEXIOx_SHIFTCTLn.

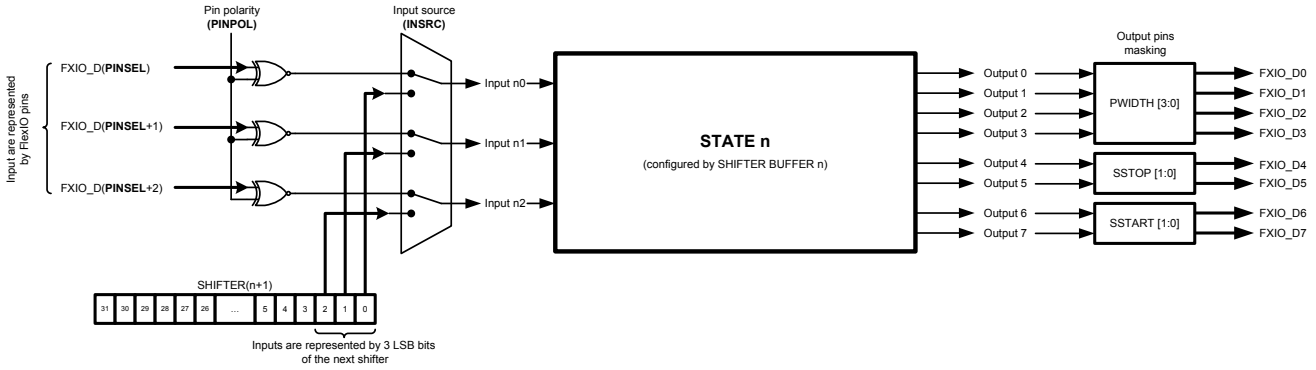

Figure 1. FlexIO state I/O assignments

The state mode is selected when the SMOD field in the shifter control register FLEXIOx_SHIFTCTLn[SMOD] is set to 0x6. The shifter state register FLEXIOx_SHIFTSTATE defines the current state when you select the state mode and enable the FlexIO module. It is set to 0x0 by default (after reset). Initialize it if shifter 0 is not configured to the state mode to avoid incorrect state machine startup. When started correctly, FLEXIOx_SHIFTSTATE points to the current state, which is defined by the shifter buffer n register FLEXIO_SHIFTERBUFFn value. Its 32-bit value contains the current-state outputs' configuration (FLEXIOx_SHIFTBUFF[31:24]) and the next state selection (FLEXIOx_SHIFTBUFF[23:0]); see Figure 2. The 24 LSBs of the shifter buffer register value represent eight groups of three bits. The three bits in each group define the value of the next state. The group (next state) is selected according to the combination of inputs. If the input combination is 000, then the value defined by the FLEXIOx_SHIFTBUFF[2:0] bits represents the next state. If the input combination is 011, then the value defined by the FLEXIOx_SHIFTBUFF[11:9] bits represents the next state.

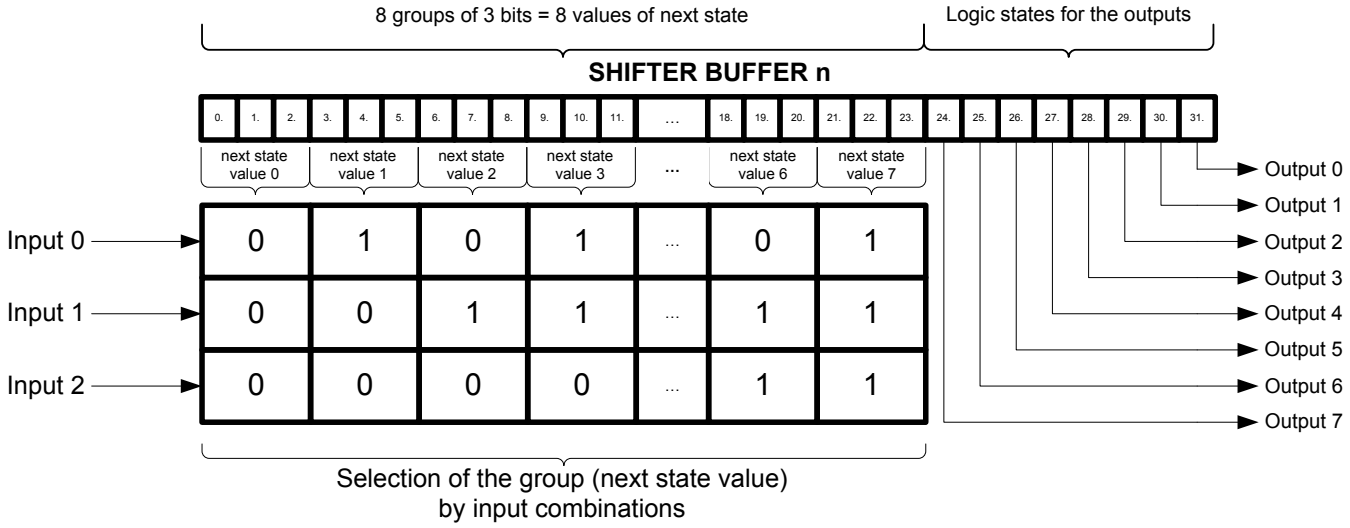

Figure 2. FlexIO STATE configuration

The next state transition is generated when the timer selected by the current state expires (see Figure 3). It depends on the TIMSEL timer select value in the shifter control register FLEXIOx_SHIFTCTLn[TIMSEL]. An appropriate shifter status flag is set when the current state pointer STATE in the shifter state register FLEXIOx_SHIFTSTATE points to the adequate shifter. Use the status flag to trigger the interrupt/DMA. Switch to a particular state to trigger the CPU/DMA, or even force a timer to assert the FlexIO output trigger (trigger the start of the ADC conversion). The flag clears when you select a different state.

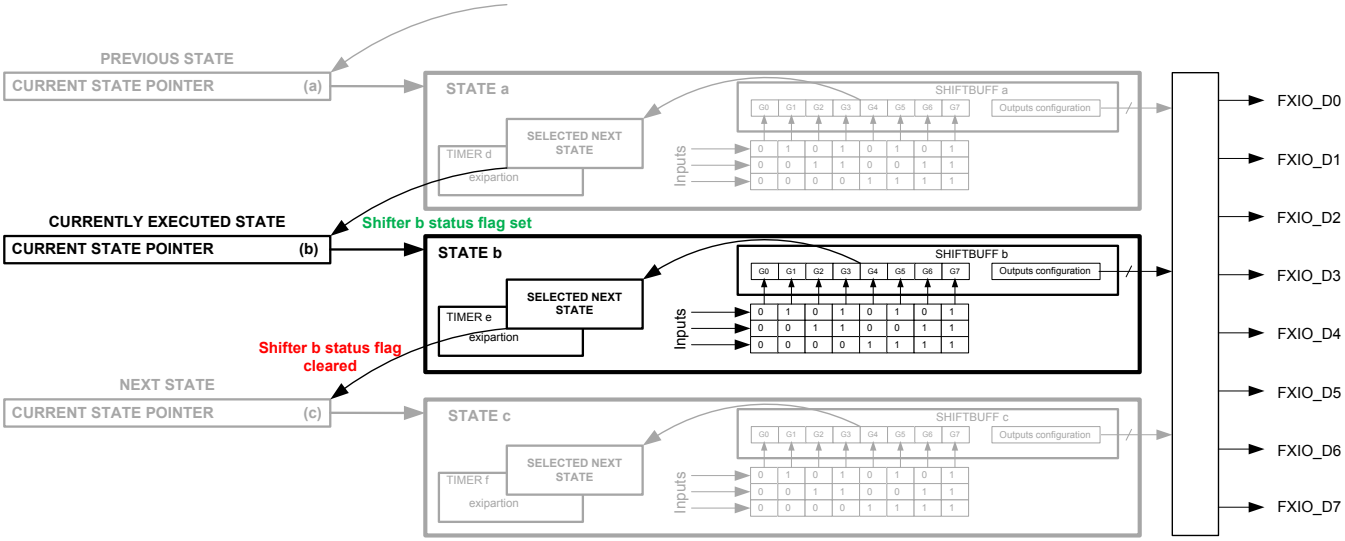

Figure 3. FlexIO state transitions

3. Traffic Lights Emulation Using FlexIO State Mode

The state machine created by the FlexIO state mode is hardware-based and its number of states, available inputs/outputs, and other resources is limited. Use this state machine in applications that are changing their states based on sensor inputs, such as conveyor lines with optical sensors/limit switches as inputs and motor contactors/relays as outputs. The example of traffic lights emulation described in this document demonstrates the basic functionality of the FlexIO state mode.

3.1. Intersection configuration

The traffic lights state machine utilizes seven states, eight (+1) outputs, and three inputs. Each state defines a different combination of traffic lights at the intersection. The eight (+1) outputs represent the equivalent lights on four traffic lights. Some of the lights are shared among the particular traffic semaphores. The three inputs represent two busy lane sensors at the intersection and one night mode sensor. Each state in this example shares the same inputs. Figure 4 shows the emulated intersection. L1-L4 represent the particular traffic lights (including the L1_1 green light for the L1 semaphore). i0-i2 represent particular input sensors. i0 senses the busyness of road 1, i1 senses the busyness of road 2, and i2 senses the day light for switching to the night mode.

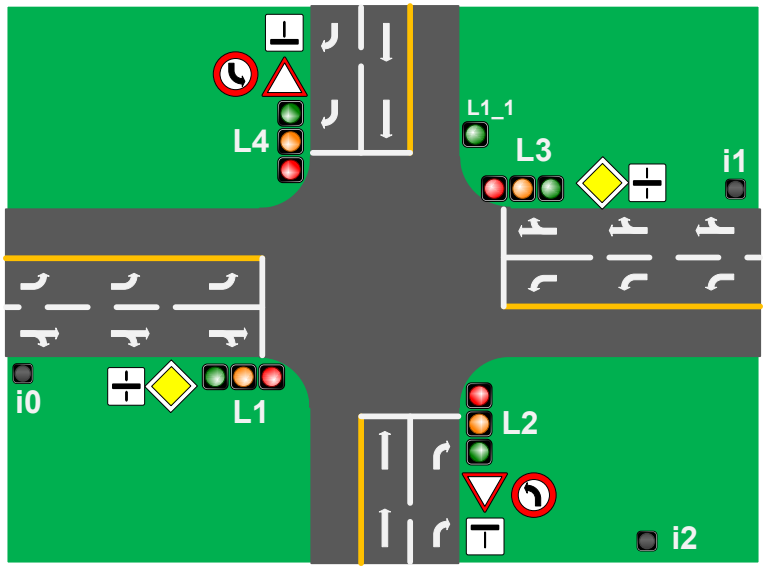

Figure 4. Emulated intersection

NOTE

The traffic lights example used in this document does not follow the strict commercial rules for such applications. The example serves to demonstrate a state machine created using the FlexIO module.

3.2. State machine diagram

When creating a state machine with finite number of states (finite state machine), create a state diagram first. The state diagram is used to demonstrate the state machine functionality. It includes state-to-state transitions based on the input logic value selection. Figure 5 shows the setting of outputs for each state. Because the number of outputs in the FlexIO module state mode is limited to eight (D0-D7), some of the semaphore lights share the same outputs, e.g., L2 and L4 semaphores share the same outputs for each light (D3 green, D1 yellow, D4 red). The seventh state S6 represents the state when all lights are switched off and the lights are in the night mode (in cooperation with the fourth state S3). In the FlexIO module, there is also output L1_1 (Dx additional green light of L1), which does not directly use the state machine outputs, but utilizes the FlexIO logic mode. This means that the additional green light output's L1_1 (Dx) value is dependent on the logical function applied to output D0 (green light on L1) and output D7 (red light on L3).

NOTE

The green light L1_1 (Dx) must only light when both the L1 green (D0) and the L3 red (D7) lights are lit, which leads to a simple logic conjunction $Dx = D0 \,\&\, D7$. The state machine described below uses open-drain output configuration which negates this meaning. This represents an implementation of a negated logic disjunction $Dx = \overline{D0 \mid D7}$.

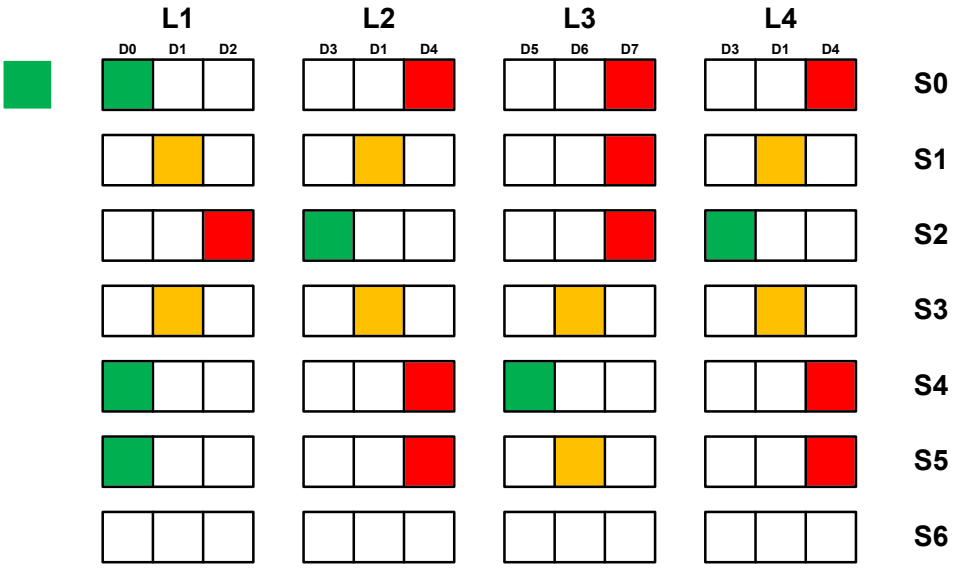

Figure 5. Setting of outputs for each state

The state diagram of the traffic lights is shown in Figure 6. It demonstrates the transitions between the particular states and their dependencies on the input configuration. The states' configuration and transitions are based on the type of intersection (see Figure 4). The input combination has a binary meaning and the bit order is i2i1i0. For example, 001 means i2 = 0, i1 = 0, and i0 = 1. Each transition is initiated by the expiration of the timer selected by the currently active state.

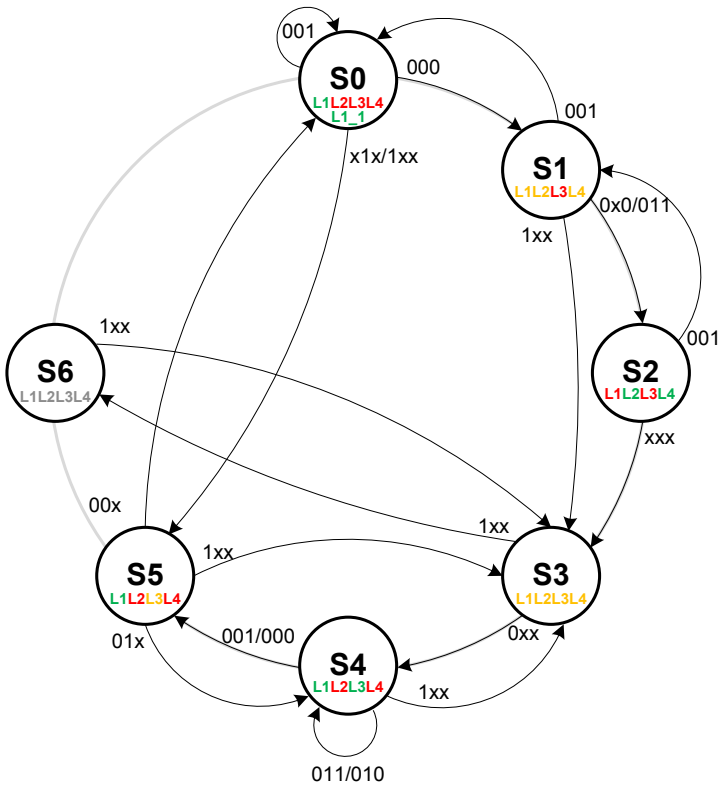

Figure 6. State diagram

Table 1 shows the particular state input/output/timing configurations, which are based on the state diagram. The content of this table is helpful during software development, because it shows how to configure a particular state defined by the shifter buffer register (see Figure 2). State S0 represents the busiest line, because its interval is the highest (7 s). State S4 represents the highest-priority line, because even if state S0 and its busyness sensor i0 are both active, the state machine forces to switch to the S4 state when its busyness sensor i1 (S4 state lane busyness sensor) becomes active. Sensor i2 has the highest priority (night mode), which switches directly to yellow lights blinking (night mode switches between states S3 and S6) when active.

Table 1. Configuration of states based on FlexIO shifter buffer registers configuration

Inputs configurations																					
	i2	D17	0	0	0	0	1	1	1	1											
	i1	D18	0	0	1	1	0	0	1	1											
	i0	D19	0	1	0	1	0	1	0	1										TIM configuration	
			Next state transition								Output configuration									State duration	Timer reset pin assignment
											D7	D6	D5	D4	D3	D2	D1	D0	HEX		
State	S0		1	0	5	5	5	5	5	5	1	0	0	1	0	0	0	1	0x91	7s	D15 (L1_1)
	S1		2	0	2	2	3	3	3	3	1	0	0	0	0	0	1	0	0x82	1s	D1
	S2		3	1	3	3	3	3	3	3	1	0	0	0	1	1	0	0	0x8C	3s	D3
	S3		4	4	4	4	6	6	6	6	0	1	0	0	0	0	1	0	0x42	1s	D1
	S4		5	5	4	4	3	3	3	3	0	0	1	1	0	0	0	1	0x31	5s	D5
	S5		0	0	4	4	3	3	3	3	0	1	0	1	0	0	0	1	0x51	1s	D6
	S6		3	3	3	3	3	3	3	3	0	0	0	0	0	0	0	0	0x00	2s	D1
			SHIFTBUFF configuration																		

4. Traffic Lights Demonstration Hardware Configuration

The Tower System modular development platform is used to build the traffic lights demonstration application. The demonstration uses these Tower System modules (see Figure 7):

- TWR-K80F150M controller module (Kinetis K80 includes FlexIO module version 1, which enables state mode configuration).
- TWR-ELEV PRIMARY and TWR-ELEV SECONDARY (no dummy version).
- TWR-PROTO.

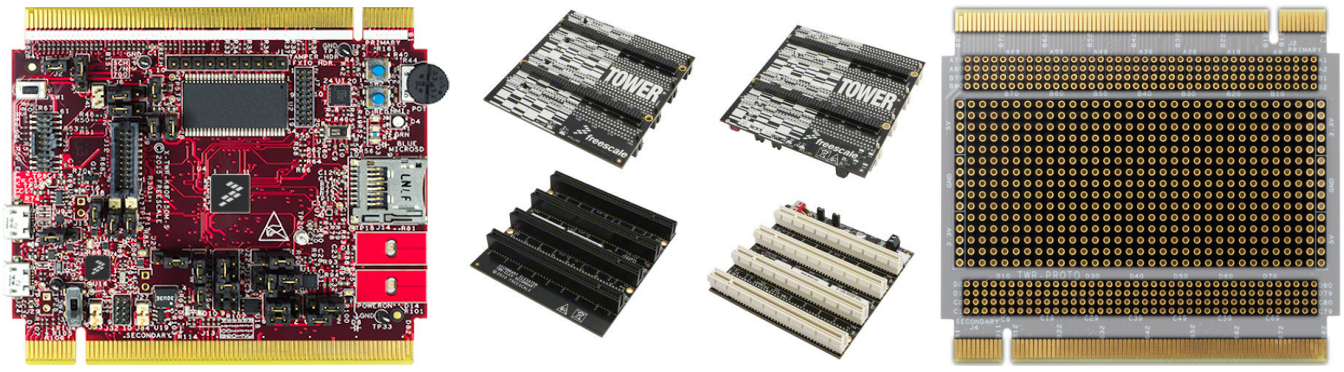

Figure 7. Tower System boards used in the application

Because not all of the required FlexIO pins are available on the controller board PCI express connectors, it is necessary to make some changes (described later on). The TWR-ELEV modules interconnect the FlexIO pins with the TWR-PROTO module.

The TWR-PROTO module includes assembled LEDs (green, yellow, and red) to demonstrate the state machine outputs (traffic lights) and photo sensors/jumper header to demonstrate state machine inputs (road lane busyness or night mode), as shown in this figure:

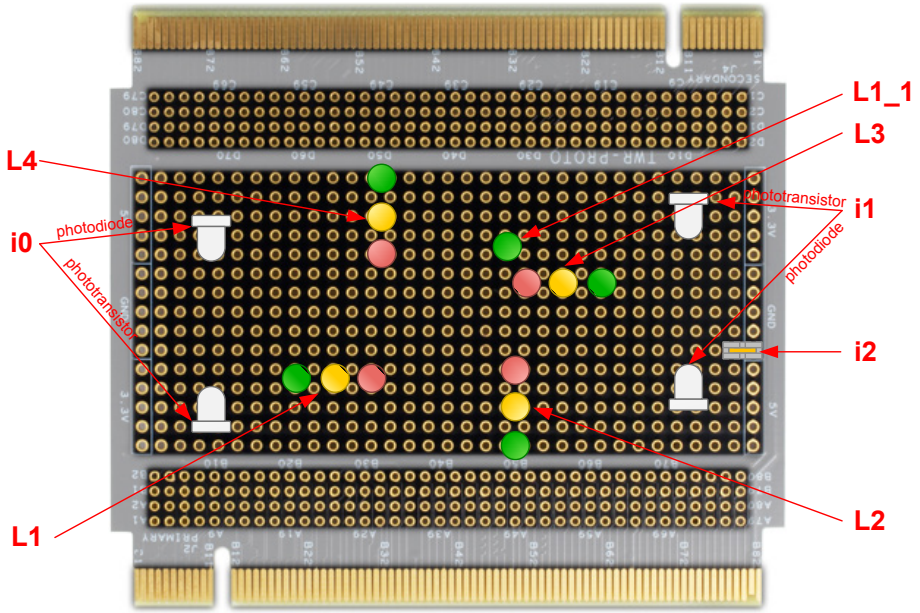

Figure 8. Placement of components on TWR-PROTO module

Table 2 shows the hardware configuration required to be made on the prototype board. The table describes how to connect state machine inputs (sensors) and outputs (LEDs) to create the demonstration application.

Figure 9 shows the schematic, including pull-up resistors used in the demonstration application. All FlexIO outputs that are connected to the LEDs are configured as open-drain.

NOTE

The FlexIO pins FXIO_D2 and FXIO_D3 are not available on the TWR-K80F150M PCI express connector. Make the interconnection directly on the TWR-K80F150M to make these outputs available on the TWR-PROTO. The simplest way is to make changes to the FLEXIO HEADER available on the TWR-K80F150M module. Connect PTB2 (FXIO_D2) to PTA12 and PTB3 (FXIO_D3) to PTA13. This connection is shown in Figure 10. Both the PTA12 and PTA13 are available on the TWR-K80F150M PCI express connector. Disable these pins (configure them into high impedance) by software. The TWR-PROTO then accepts PTA12 and PTA13 as PTB2 and PTB3.

Table 2. Hardware configuration

State machine outputs						
FLEXIO pin		MCU pin		ELEVATOR pin	Traffic light LEDs (cathode pin)	
assignment	configuration	assignment	functionality		trafic light	LED color
FXIO_D0	output (open drain)	PTB0	ALT7 (Flexio)	A38	L1	green
FXIO_D1	output (open drain)	PTB1	ALT7 (Flexio)	A37	L1 , L2, L4	yellow
FXIO_D2	output (open drain)	PTB2/PTA12	ALT7 (Flexio)/ALT0 (disabled)	A25	L1	red
FXIO_D3	output (open drain)	PTB3/PTA13	ALT7 (Flexio)/ALT0 (disabled)	A23	L2, L4	green
FXIO_D4	output (open drain)	PTB10	ALT7 (Flexio)	B70	L2, L4	red
FXIO_D5	output (open drain)	PTB11	ALT7 (Flexio)	B69	L3	green
FXIO_D6	output (open drain)	PTB18	ALT7 (Flexio)	B66	L3	yellow
FXIO_D7	output (open drain)	PTB19	ALT7 (Flexio)	B72	L3	red

State machine inputs (all state share the same inputs)						
FLEXIO pin		MCU pin		ELEVATOR pin	Traffic lights sensors	
assignment	configuration	assignment	functionality			
FXIO_D17	input	PTC9	ALT7 (Flexio)	A74	i0	phototransistor collector pin
FXIO_D18	input	PTC10	ALT7 (Flexio)	A75	i1	phototransistor collector pin
FXIO_D19	input	PTC11	ALT7 (Flexio)	B71	i2	jumper header pin

Logic mode pins						
FLEXIO pin		MCU pin		ELEVATOR pin	Traffic light LEDs (cathode pin)	
assignment	configuration	assignment	functionality		trafic light	LED color
FXIO_D13	input	PTC1	ALT7 (Flexio)	A67	L1	green
FXIO_D14	input	PTC6	ALT7 (Flexio)	A71	L3	red
FXIO_D15	output (open drain)	PTC7	ALT7 (Flexio)	A72	L1_1	green

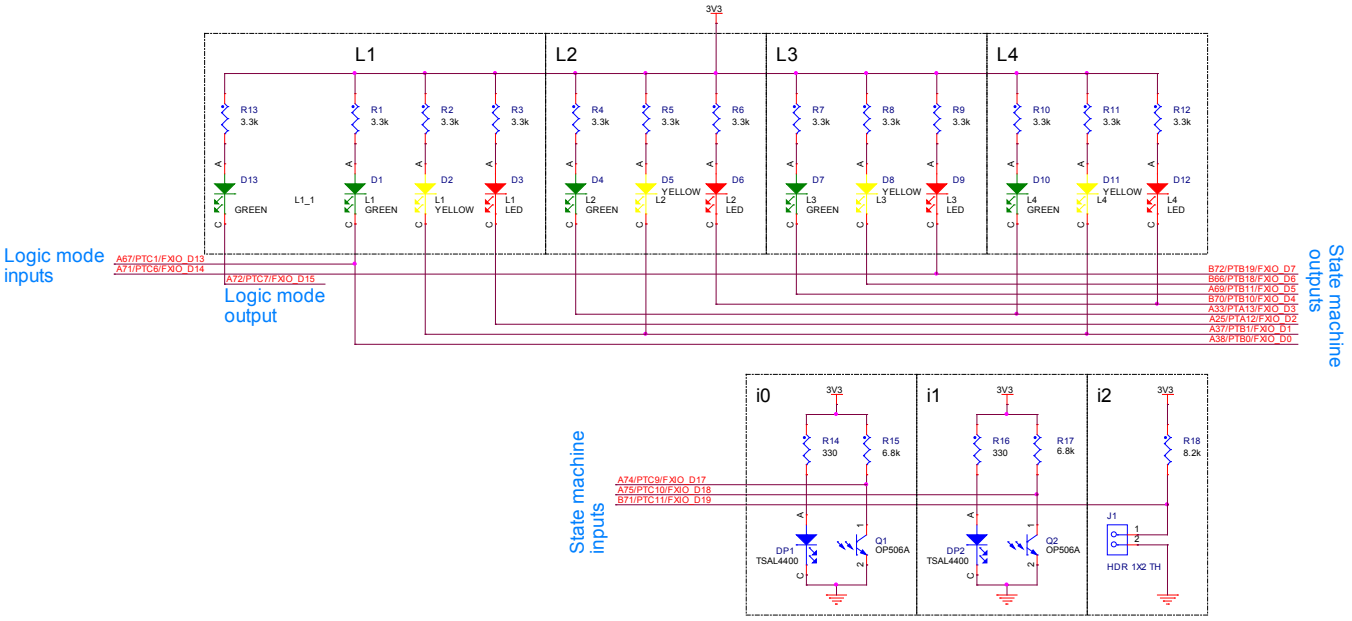

Figure 9. Schematic of TWR-PROTO assembly

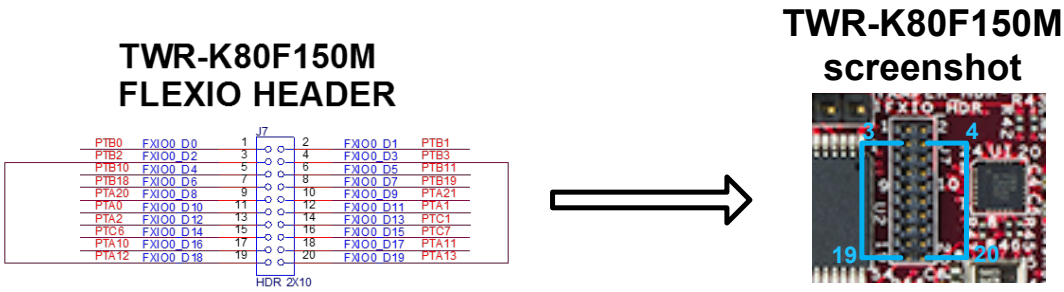

Figure 10. TWR-K80F150M FlexIO header interconnection

5. Traffic Lights Demonstration Software Description

The traffic lights demonstration software utilizes KSDK 1.3 support for TWR-K80F150M.

5.1. Hardware initialization module

The KSDK 1.3 support uses board-specific definition to initialize clocks, ports, and so on. It is a simple configuration, because the demonstration application mostly uses only one peripheral module—FlexIO. Enable and configure the required ports' pins, MCU clock, and FlexIO module. The *hardware_init* function includes:

- Enabling the required ports' clock gates (at least PORTA, PORTB, and PORTC).
- Selecting the FlexIO source clock (the 4 MHz IRC is selected) and enabling the FlexIO clock gate.
- Selecting PLL as the main clock source for the MCU (normal RUN mode at 120 MHz).

```
void hardware_init(void)
{
  mcg_modes_t mcg_mode_test;

  /* Enable clock for PORTs */
  CLOCK_SYS_EnablePortClock(PORTA_IDX);
  CLOCK_SYS_EnablePortClock(PORTB_IDX);
  CLOCK_SYS_EnablePortClock(PORTC_IDX);
  CLOCK_SYS_EnablePortClock(PORTD_IDX);
  CLOCK_SYS_EnablePortClock(PORTE_IDX);

  /* Enable clock gate to FlexIO module*/
  CLOCK_HAL_SetFlexioSrc(SIM, 0, kClockFlexioSrcMcgIrClk);
  SIM_HAL_EnableClock(SIM, kSimClockGateFlexio0);

  /* Init board clock */
```

```
  App_ClockInit();
  mcg_mode_test = (CLOCK_HAL_GetMcgMode(MCG));
  while(mcg_mode_test != kMcgModePEE);
}
```

The PLL working in the PEE mode is selected as the main clock source for the MCU (MCGCLKOUT), and the external oscillator pins and the oscillator module must be configured properly. The application clock-initialization function includes configuring pins PTA18 and PTA19 to external clock functionality, and initiating the oscillator module before entering the *CLOCK_SYS_SetConfiguration()* function. This configures the appropriate Multi-purpose Clock Generator module (MCG) to PEE normal RUN mode. The application clock initialization function also includes a function to enter the very-low-power stop (VLPS) mode.

```
static void App_ClockInit(void)
{
  SMC_HAL_SetProtection(SMC, kAllowPowerModeVlp);
  PORT_HAL_SetMuxMode(EXTAL0_PORT, EXTAL0_PIN, EXTAL0_PINMUX);
  PORT_HAL_SetMuxMode(XTAL0_PORT, XTAL0_PIN, XTAL0_PINMUX);
  BOARD_InitOsc0();
  CLOCK_SYS_SetConfiguration(&g_RunClockConfig);
}
```

The clock configuration structure is defined as the macro below. The most important features used in the configuration macro are the selection of the PEE mode with an appropriate selection of PLL dividers, to enable the 4 MHz IRC to enter the stop mode, enable OSC module, and select the appropriate CORE/SYSTEM:BUS:FLEXBUS:FLASH clock dividers.

```
const clock_manager_user_config_t g_RunClockConfig = NORMAL_RUN_CLOCK_CONFIGURATION;


#define NORMAL_RUN_CLOCK_CONFIGURATION
{
    .mcgConfig =
    {
        .mcg_mode               = kMcgModePEE,
        .irclkEnable            = true,
        .irclkEnableInStop      = true,
        .ircs                   = kMcgIrcFast,
        .fcrdiv                 = 0U,
        .frdiv                  = 3U,
        .drs     = kMcgDcoRangeSelLow,
        .dmx32   = kMcgDmx32Default,
        .oscsel  = kMcgOscselOsc,
        .pll0EnableInFllMode = false,
```

```
        .pll0EnableInStop  = false,
        .prdiv0            = 0x00U,
        .vdiv0             = 0x04U,
    },
    .simConfig =
    {
        .pllFllSel = kClockPllFllSelPll,
        .pllflldiv = 1,
        .pllfllfrac = 0,
        .er32kSrc  = kClockEr32kSrcRtc,
        .outdiv1   = 0U,
        .outdiv2   = 1U,
        .outdiv3   = 1U,
        .outdiv4   = 4U,
    },
    .oscerConfig =
    {
        .enable       = true,
        .enableInStop = false,
    }
}
```

The required port pins' configuration is presented in the hardware_init module. The configuration includes all FlexIO pins dedicated to port configuration (see Table 2). All FlexIO pins used are configured for the FlexIO functionality, which represents pin multiplexer configuration alternative 7. However, PTA12 and PTA13 are disabled to act as high impedance. The application pins' configuration function is:

```
void configure_app_flexio_pins(void)
{
  // Configure FlexIO state mode output pins
  PORT_HAL_SetMuxMode(PORTB,0u,kPortMuxAlt7); ///< FXIO_D0
  PORT_HAL_SetMuxMode(PORTB,1u,kPortMuxAlt7); ///< FXIO_D1
  PORT_HAL_SetMuxMode(PORTB,2u,kPortMuxAlt7); ///< FXIO_D2
  PORT_HAL_SetMuxMode(PORTB,3u,kPortMuxAlt7); ///< FXIO_D3
  PORT_HAL_SetMuxMode(PORTB,10u,kPortMuxAlt7); ///< FXIO_D4
  PORT_HAL_SetMuxMode(PORTB,11u,kPortMuxAlt7); ///< FXIO_D5
  PORT_HAL_SetMuxMode(PORTB,18u,kPortMuxAlt7); ///< FXIO_D6
  PORT_HAL_SetMuxMode(PORTB,19u,kPortMuxAlt7); ///< FXIO_D7
  // Configure FlexIO state mode input pins
```

```
  PORT_HAL_SetMuxMode(PORTC,9u,kPortMuxAlt7); ///< FXIO_D17
  PORT_HAL_SetMuxMode(PORTC,10u,kPortMuxAlt7); ///< FXIO_D18
  PORT_HAL_SetMuxMode(PORTC,11u,kPortMuxAlt7); ///< FXIO_D19
  // Configure FlexIO Logic mode input pins
  PORT_HAL_SetMuxMode(PORTC,1u,kPortMuxAlt7); ///< FXIO_D13
  PORT_HAL_SetMuxMode(PORTC,6u,kPortMuxAlt7); ///< FXIO_D14
  // Configure FlexIO Logic mode output pin
  PORT_HAL_SetMuxMode(PORTC,7u,kPortMuxAlt7); ///< FXIO_D15
  // Configure FlexIO timers 0-6 input
  PORT_HAL_SetMuxMode(PORTC,8u,kPortMuxAlt7); ///< FXIO_D16

  // to support PTB2, PTB3 - PTA12, PTA13 should be in high impedance
  PORT_HAL_SetMuxMode(PORTA,12u,kPortPinDisabled); ///< bridge for PTB2
  PORT_HAL_SetMuxMode(PORTA,13u,kPortPinDisabled); ///< bridge for PTB3
}
```

5.2. Configuring FlexIO timers

As shown in Table 1, the state transitions in the state machine are based on time interval (in seconds). It is important to find a source of clock capable of such timing. Considering the 16-bit mode of the FlexIO timer is used for state duration, then the maximum source clock interval for the FlexIO module is:

$$\textit{Eq. 1} \qquad \frac{65535}{max.time\ interval\ of\ state} = 9362Hz$$

The TWR-K80F150M module enables using the slow IRC (32 kHz) as the lowest source of clock for the FlexIO, but it is still too high. Use an additional timer to decrease the clock and use it as a trigger for the specified state timers. Each timer's state then decrements according to the trigger input. In such case, do not use the slow IRC strictly as a source clock. Select the 4 MHz fast IRC as the FlexIO source clock.

5.2.1. Configuring timer 7

Use this timer to generate the base clock for state duration timing. Configure the timer to generate 1 kHz (1 ms period) clock from the FlexIO clock source (which is configured to MCGIRCLK) and select the fast IRC 4 MHz clock. To generate a 1 kHz clock from the 4 MHz FlexIO clock, divide it by 4000. Configure the timer to decrement on the FlexIO clock in a 16-bit mode with the compare value set to 3999. No additional changes in the configuration are needed.

```
const flexio_timer_config_t flexio_timer7_config = FLEXIO_TIM_CLOCK_CONFIG;


#define FLEXIO_TIM_CLOCK_CONFIG
{
  .trgsel = 0x0,
```

```
  .trgpol = kFlexioTimerTriggerPolarityActiveHigh,
  .trgsrc = kFlexioTimerTriggerSourceInternal,
  .pincfg = kFlexioPinConfigOutputDisabled,
  .pinsel = 0,
  .pinpol = kFlexioPinActiveHigh,
  .timod  = kFlexioTimerModeSingle16Bit,
  .timout = kFlexioTimerOutputZeroNotAffectedByReset,
  .timdec = kFlexioTimerDecSrcOnFlexIOClockShiftTimerOutput,
  .timrst = kFlexioTimerResetNever,
  .timdis = kFlexioTimerDisableNever,
  .timena = kFlexioTimerEnabledAlways,
  .tstop  = kFlexioTimerStopBitDisabled,
  .tstart = kFlexioTimerStartBitDisabled,
  .timcmp = 3999,
}
```

5.2.2. Configuring timers 0-5

Use these timers to define the time interval for the dedicated state duration. Each state uses a dedicated timer (e.g., S0 uses TIM0) to configure its time interval according to Table 1. Each of these timers uses timer 7 as the trigger and decrements according to the trigger input. The decrement clock is 1 ms.
The timers are configured to a 16-bit mode and they reset on the rising edge of the timer input pin. Reset the dedicated timer before entering the dedicated state due to synchronization. The best way to reset a timer in the state mode is to use a dedicated shifter status flag as a trigger for the dedicated timer, and to configure the timer reset to reset on the rising edge of the trigger. In this case, timer 7 is used as a trigger for the timers due to a missing low-frequency clock source. It is then required to find a different technique to reset the appropriate timer. Because each state changes at least one of the state outputs (FXIO_D0 – FXIO_D7), apply the feedback from this output to the timer input pin and reset the timer based on the rising edge of the pin (see the last column of Table 1).

```
const flexio_timer_config_t flexio_timer0_config = FLEXIO_TIM_STATE_CONFIG(7000,15);
const flexio_timer_config_t flexio_timer1_config = FLEXIO_TIM_STATE_CONFIG(1000,1);
const flexio_timer_config_t flexio_timer2_config = FLEXIO_TIM_STATE_CONFIG(3000,3);
const flexio_timer_config_t flexio_timer3_config = FLEXIO_TIM_STATE_CONFIG(1000,1);
const flexio_timer_config_t flexio_timer4_config = FLEXIO_TIM_STATE_CONFIG(5000,5);
const flexio_timer_config_t flexio_timer5_config = FLEXIO_TIM_STATE_CONFIG(1000,6);


#define FLEXIO_TIM_STATE_CONFIG(time,pin)
{
  .trgsel = FLEXIO_HAL_TIMER_TRIGGER_SEL_TIMn(7),
  .trgpol = kFlexioTimerTriggerPolarityActiveHigh,
```

```
  .trgsrc = kFlexioTimerTriggerSourceInternal,
  .pincfg = kFlexioPinConfigOutputDisabled,
  .pinsel = pin,
  .pinpol = kFlexioPinActiveLow,
  .timod  = kFlexioTimerModeSingle16Bit,
  .timout = kFlexioTimerOutputZeroAffectedByReset,
  .timdec = kFlexioTimerDecSrcOnTriggerInputShiftTimerOutput,
  .timrst = kFlexioTimerResetOnTimerPinRisingEdge,
  .timdis = kFlexioTimerDisableNever,
  .timena = kFlexioTimerEnabledAlways,
  .tstop  = kFlexioTimerStopBitDisabled,
  .tstart = kFlexioTimerStartBitDisabled,
  .timcmp = time,
}
```

5.2.3. Configuring timer 6

Timer 6 (assigned to state 6) is configured in the same way as timers 0-5. The only difference is that it is reset on the falling edge of the input pin (defined by pin's polarity).

```
const flexio_timer_config_t flexio_timer6_config = FLEXIO_TIM_STATE6_CONFIG(2000,1);
#define FLEXIO_TIM_STATE6_CONFIG(time,pin)
{
  .trgsel = FLEXIO_HAL_TIMER_TRIGGER_SEL_TIMn(7),
  .trgpol = kFlexioTimerTriggerPolarityActiveHigh,
  .trgsrc = kFlexioTimerTriggerSourceInternal,
  .pincfg = kFlexioPinConfigOutputDisabled,
  .pinsel = pin,
  .pinpol = kFlexioPinActiveHigh,
  .timod  = kFlexioTimerModeSingle16Bit,
  .timout = kFlexioTimerOutputZeroAffectedByReset,
  .timdec = kFlexioTimerDecSrcOnTriggerInputShiftTimerOutput,
  .timrst = kFlexioTimerResetOnTimerPinRisingEdge,
  .timdis = kFlexioTimerDisableNever,
  .timena = kFlexioTimerEnabledAlways,
  .tstop  = kFlexioTimerStopBitDisabled,
  .tstart = kFlexioTimerStartBitDisabled,
  .timcmp = time,
}
```

5.3. Configuring FlexIO shifters

Configuring the shifters in cooperation with the shifter buffers' configuration is important in the traffic lights (state machine) configuration. The shifter buffer register value (32-bits) includes the definition of the next eight states (24-bits) and also the output configuration of a dedicated state (8-bits). The next eight states are represented by eight groups of three bits. The three bits in each group represent the number of the next state (from 0 to 7). Select the dedicated group (next state) by combining the input pins. Define these features in the software using a specific structure definition.

```
typedef struct {
  const uint32_t nextStateGroup0 : 3;
  const uint32_t nextStateGroup1 : 3;
  const uint32_t nextStateGroup2 : 3;
  const uint32_t nextStateGroup3 : 3;
  const uint32_t nextStateGroup4 : 3;
  const uint32_t nextStateGroup5 : 3;
  const uint32_t nextStateGroup6 : 3;
  const uint32_t nextStateGroup7 : 3;
  const uint32_t outputCfg : 8;
} flexio_shifter_state_cfg_t;
```

Initialize the features according to data from Table 1.

```
const flexio_shifter_state_cfg_t flexio_shifter_states_cfg[7] =
{
  {1,0,5,5,5,5,5,5,0x91},
  {2,0,2,2,3,3,3,3,0x82},
  {3,1,3,3,3,3,3,3,0x8C},
  {4,4,4,4,6,6,6,6,0x42},
  {5,5,4,4,3,3,3,3,0x31},
  {0,0,4,4,3,3,3,3,0x51},
  {3,3,3,3,3,3,3,3,0x00},
};
```

These data define the output configuration and the next state transitions for each state. This case utilizes seven states (seven shifters) and all eight outputs. The state mode uses seven shifters with eight dedicated output pins. One shifter is used in a logic mode to create the ninth output, which represents the additional green light (L1_1) of light L1.

5.3.1. Configuring shifters 0-6 (state-mode shifters)

Configure the first seven shifters (SHIFTER0-6) to a state mode. Use the appropriate timers (TIM0-6) to define the state-time interval. Configure all of their outputs to open-drain. All shifters (all states) share the same three inputs, starting with FlexIO pin 17 (FXIO_D17, FXIO_D18, and FXIO_D19), as shown in Table 1 and Table 2. The input pin polarity is inverted, because the sensors connected to the input pins are grounded (see the schematic in Figure 9) when activated.

```
const flexio_shifter_config_t flexio_shifter0_config = FLEXIO_SHIFTER_STATE_CONFIG(0,17);
const flexio_shifter_config_t flexio_shifter1_config = FLEXIO_SHIFTER_STATE_CONFIG(1,17);
const flexio_shifter_config_t flexio_shifter2_config = FLEXIO_SHIFTER_STATE_CONFIG(2,17);
const flexio_shifter_config_t flexio_shifter3_config = FLEXIO_SHIFTER_STATE_CONFIG(3,17);
const flexio_shifter_config_t flexio_shifter4_config = FLEXIO_SHIFTER_STATE_CONFIG(4,17);
const flexio_shifter_config_t flexio_shifter5_config = FLEXIO_SHIFTER_STATE_CONFIG(5,17);
const flexio_shifter_config_t flexio_shifter6_config = FLEXIO_SHIFTER_STATE_CONFIG(6,17);


#define FLEXIO_SHIFTER_STATE_CONFIG(timer,input)
{
  .timsel = timer,
  .timpol = kFlexioShifterTimerPolarityOnPositive,
  .pincfg = kFlexioPinConfigOpenDrainOrBidirection,
  .pinsel = input,
  .pinpol = kFlexioPinActiveHigh,
  .smode  = kFlexioShifterModeState,
  .pwidth = 0,
  .insrc  = kFlexioShifterInputFromPin,
  .sstop  = kFlexioShifterStopBitDisable,
  .sstart = kFlexioShifterStartBitDisabledLoadDataOnEnable,
}
```

5.3.2. Configuring shifter 7 and shifter buffer 7 (logic mode)

Use shifter 7 in the logic mode to extend the number of outputs by 1. This shifter ensures a proper functionality of the additional green light L1_1 on traffic light L1. This green light lights only when L1 green (output D0) and L3 red (output D7) lights are lit. In all other combinations of these two lights (outputs), the L1_1 switches off. A direct feedback of outputs D0 and D7 is not possible due to the shifter logic mode pin selection incompatibility. Connect the D0 and D7 signals to the additional FlexIO pins that are used as the inputs to the logic mode look-up table (see the schematic in Figure 9). From the perspective of hardware used, it is possible to use only pins D13 and D14 as the inputs and D15 as the output of the look-up table. Because shifter 7 is selected to be used in the logic mode, use only the pins starting from shifter number + 4 in the logic mode (7+4=11). Pins D11, D12, D13, D14, and shifter 7 output can be used as inputs and pin D15 as the output in the logic mode.

On TWR-K80F150M, it is only possible to use pins D13 and D14 as the inputs, and pin D15 as the output. Therefore, the state machine output D0 is connected to the logic mode input D13, and output D7 is connected to input D14. The logic mode output D15 is connected to the additional green light L1_1 (green LED). The unused inputs D11 and D12 are masked by the SSTART[1:0] bits in the shifter configuration register (*.sstart = kFlexioShifterStartBitHigh*). They are not listed in the look-up table (including the fifth input—shifter 7 output). The unused inputs do not have any effect on the logic mode output. Because the shifter 7 output is not considered as a look-up table input, the configuration of PINSEL and PINPOL in the shifter control register FLEXIO0_SHIFTCTL7 (and PWIDTH in the shifter configuration register FLEXIO0_SHIFTCFG7) does not matter.

```
const flexio_shifter_config_t flexio_shifter7_config = FLEXIO_SHIFTER_LOGIC_CONFIG(7);


#define FLEXIO_SHIFTER_LOGIC_CONFIG()
{
  .timsel = 0x0,
  .timpol = kFlexioShifterTimerPolarityOnPositive,
  .pincfg = kFlexioPinConfigOpenDrainOrBidirection,
  .pinsel = 0x0,
  .pinpol = kFlexioPinActiveHigh,
  .smode  = kFlexioShifterModeLogic,
  .pwidth = 0x0,
  .insrc  = kFlexioShifterInputFromPin,
  .sstop  = kFlexioShifterStopBitDisable,
  .sstart = kFlexioShifterStartBitHigh,
}
```

The logic function applied to the look-up table is defined by this logic conjunction:

$$\textit{Eq. 2} \qquad D15 = D13 \& D14$$

The look-up table is as follows:

D13 (input)	0	0	1	1
D14 (input)	0	1	0	1
D15 (output)	0	0	0	1

Because the state machine outputs D0 and D7 (connected to the logic mode inputs D13 and D14) use open-drain configuration and the inputs D13 and D14 cannot be inverted, it is required to reflect this in the look-up table. The look-up table changes to:

D13 (input)	1	1	0	0
D14 (input)	1	0	1	0
D15 (output)	0	0	0	1

This means that the logic function changes to the negated logic disjunction:

$$Eq.\ 3 \qquad D15 = \overline{D13|D14}$$

When you apply this to the final look-up table (considering FlexIO logic mode shifter buffer configuration), it looks like this:

Table 3. Look-up table applied to logic mode

Inputs			Combinations																																					
FXIO_D11		A	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
FXIO_D12		B	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
FXIO_D13		C	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
FXIO_D14		D	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Shifter 7 output		E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output			Shifter Buffer 7 value (Look-up table values)																																							
FXIO_D15	BIN	Y	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1						
	HEX		0				0				0				f				0				0				0				f									

When you apply the negated logic conjunction to the logic mode look-up table inputs D13 and D14, then the shifter buffer 7 has the value of 0x000F000F. The unused inputs of the look-up table are not taken into consideration.

```
const uint32_t flexio_shifter_logic_cfg = 0x000F000F;
```

5.4. Main function

The main function calls the *hardware_init()* and *configure_app_flexio_pins()* functions, and configures the FlexIO module according to the shifter (the shifter buffer and timer configuration are explained earlier in this document). It then enters the very-low-power stop mode to reduce the current consumption as much as possible (with the demonstration application consuming ~500 μA).

```
int main(void)
{
  hardware_init();
  configure_app_flexio_pins();

  FlexIO_StateMachineInit();

  while(1)
  {
    SMC_HAL_SetMode(SMC, &smc_vlps_mode_config);
  }
}
```

The FlexIO state machine initialization function configures all shifters/timers/shifter buffers and enables the FlexIO module.

```
static void FlexIO_StateMachineInit (void)
{
  uint32_t *p_u32Temp = (uint32_t*)&flexio_shifter_states_cfg;

  FLEXIO_HAL_SetShifterBuffer(FLEXIO0, 0, *(uint32_t*)p_u32Temp);
  FLEXIO_HAL_SetShifterBuffer(FLEXIO0, 1, *(uint32_t*)++p_u32Temp);
  FLEXIO_HAL_SetShifterBuffer(FLEXIO0, 2, *(uint32_t*)++p_u32Temp);
  FLEXIO_HAL_SetShifterBuffer(FLEXIO0, 3, *(uint32_t*)++p_u32Temp);
  FLEXIO_HAL_SetShifterBuffer(FLEXIO0, 4, *(uint32_t*)++p_u32Temp);
  FLEXIO_HAL_SetShifterBuffer(FLEXIO0, 5, *(uint32_t*)++p_u32Temp);
  FLEXIO_HAL_SetShifterBuffer(FLEXIO0, 6, *(uint32_t*)++p_u32Temp);
  FLEXIO_HAL_SetShifterBuffer(FLEXIO0, 7, flexio_shifter_logic_cfg); // logic table

  FLEXIO_HAL_ConfigureShifter(FLEXIO0, 0, &flexio_shifter0_config);
  FLEXIO_HAL_ConfigureShifter(FLEXIO0, 1, &flexio_shifter1_config);
  FLEXIO_HAL_ConfigureShifter(FLEXIO0, 2, &flexio_shifter2_config);
  FLEXIO_HAL_ConfigureShifter(FLEXIO0, 3, &flexio_shifter3_config);
  FLEXIO_HAL_ConfigureShifter(FLEXIO0, 4, &flexio_shifter4_config);
  FLEXIO_HAL_ConfigureShifter(FLEXIO0, 5, &flexio_shifter5_config);
  FLEXIO_HAL_ConfigureShifter(FLEXIO0, 6, &flexio_shifter6_config);
  FLEXIO_HAL_ConfigureShifter(FLEXIO0, 7, &flexio_shifter7_config);

  FLEXIO_HAL_ConfigureTimer(FLEXIO0, 0, &flexio_timer0_config);
  FLEXIO_HAL_ConfigureTimer(FLEXIO0, 1, &flexio_timer1_config);
  FLEXIO_HAL_ConfigureTimer(FLEXIO0, 2, &flexio_timer2_config);
  FLEXIO_HAL_ConfigureTimer(FLEXIO0, 3, &flexio_timer3_config);
  FLEXIO_HAL_ConfigureTimer(FLEXIO0, 4, &flexio_timer4_config);
  FLEXIO_HAL_ConfigureTimer(FLEXIO0, 5, &flexio_timer5_config);
  FLEXIO_HAL_ConfigureTimer(FLEXIO0, 6, &flexio_timer6_config);
  FLEXIO_HAL_ConfigureTimer(FLEXIO0, 7, &flexio_timer7_config);

  FLEXIO_HAL_SetFlexioEnableCmd(FLEXIO0, true);
}
```

The power stop mode configuration structure initialization sets the normal very-low-power stop mode:

```
static smc_power_mode_config_t smc_vlps_mode_config = SMC_VLPS_MODE_CONFIG;


#define SMC_VLPS_MODE_CONFIG
{
    .powerModeName = kPowerModeVlps,
    .stopSubMode = kSmcStopSub0,
    .porOptionValue = kSmcPorDisabled,
    .ram2OptionValue = kSmcRam2DisPowered,
    .pstopOptionValue = kSmcPstopStop,
    .lpoOptionValue = kSmcLpoDisabled,
}
```

6. Conclusion

This application note describes the state mode of the FlexIO module available on some of the Kinetis devices (FlexIO version 1.1). It focuses on:

- The basic features of the FlexIO state mode.
- Configuring the FlexIO module to create a simple hardware state machine.
- Implementing the state machine into the KSDK 1.3. software platform.

The state mode available on the FlexIO module version 1 enables creating a hardware state machine with up to eight states, eight outputs shared for each state, and three inputs selectable for each state. The maximal configuration of the state machine (eight states, eight outputs, three different inputs for each state) fully exploits the resources within the FlexIO module (all shifters used, timer usage is dependent on state transition timing, and all 32 pins are used). You can also create a smaller state machine (lower number of states and outputs, shared inputs). Then you can use the other FlexIO resources for serial communication or for any other purpose. There are these limits in the FlexIO state mode for you to consider:

- State outputs are fixed to the specified FlexIO pins (FXIO_D0-FXIO_D7).
- States' inputs (configured as FlexIO pins) are not individually selectable (they are selectable as a group of three pins, starting from PINSEL number selection).

With such hardware state machine, you can fully offload the MCU. The MCU can enter a low-power mode, which reduces the power consumption rapidly (FlexIO is fully functional in the wait modes, and it operates asynchronously in the stop modes, down to the VLPS mode). The FlexIO module version 1.0 does not include the state mode.

Creating a hardware state machine using the FlexIO module is shown using a simple example (traffic light emulation). The basic criteria to create such hardware state machine easily is to create a proper state diagram which explains the individual state-to-state transitions based on inputs' configuration, state configuration, and so on. Configuring the FlexIO module is then very simple and quick.

The hardware configuration can be more complex when you use general-purpose development hardware. Some special connections are required due to inaccessibility of the dedicated FlexIO pins on the TWR-K80F150M module. Avoid this complexity by developing application-dedicated hardware.

The KSDK 1.3 software platform is used to create code for the dedicated hardware-based state machine. The simplest part of such application is the software, due to easy-to-use KSDK 1.3 software package. The demonstration application presented in this application note describes proper FlexIO resource configuration and FlexIO module initialization. You can include software intervention using interrupts or simple data transition using DMA transfers, based on timer/shifters event requests.

7. Revision history

This table summarizes the changes done to this document since the initial release:

Table 4. Revision history

Revision number	Date	Substantive changes
0	01/2016	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5239
Rev. 0
01/2016