Subsystem Design

PWM Control Using Push-Buttons

1 Description

This subsystem example demonstrates how to vary the period and duty cycle of a Pulse Width Modulation (PWM) output based on inputs from a set of push buttons.

This subsystem example demonstrates using the two switches on the MSPM0 LaunchPad™ Development Kit to change the period and duty cycle of two PWM output channels. TIMA controls these channels with CC0 and CC1. Pressing switch S1 changes the PWM period of the timer. Pressing S2 changes the duty cycle on only the CC0 output. This example also demonstrates the use of a one-shot timer with an interrupt to handle input switch debouncing.

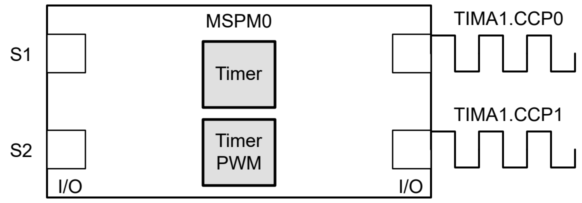


Figure 1-1. Subsystem Function Block Diagram

2 Required Peripherals

Table 2-1 describes the *required* integrated peripherals.

Table 2-1. Required Peripherals

Sub-block Functionality	Peripheral Use	Notes
Switches	2 × GPIO	Shown as *GPIO_SWITCHES* in code
PWM Output	1 × TIMx	Shown as *PWM* in code
Timer	1 × TIMx	Shown as *TIMER* in code for debouncing logic

3 Design Steps

1. Determine multiple configurations for the output PWM signal including frequency and duty cycle based on design requirements.
2. Generate timer configuration code utilizing SysConfig.
3. Configure the required switches (GPIOs) in SysConfig.
4. Create cases for each timer configuration from steps 1–2.
5. Write application code that loops through the timer configurations based on interrupts triggered by pressing the switches. See Figure 5-1 and Figure 5-2 for an overview of the software.

4 Design Considerations

1. **Race conditions**: To avoid the possibility of missing a button press, the application code quickly sets flags within the respective interrupt handler. This allows the processor to get-in-get-out and minimize the time frame that another triggered interrupt needs to be pending.
2. **Debouncing**: Debouncing of the buttons can cause unintended interrupts to affect the way the application code runs. To prevent this from happening, the interrupt is disabled inside of the interrupt handler that is triggered. A timer is utilized to re-enable any disabled interrupts about 10ms after entering the interrupt handler.

5 Software Flow Chart

Figure 5-1 and Figure 5-2 show the code flow diagrams for Main() and the various ISRs for Figure 1-1, respectively.

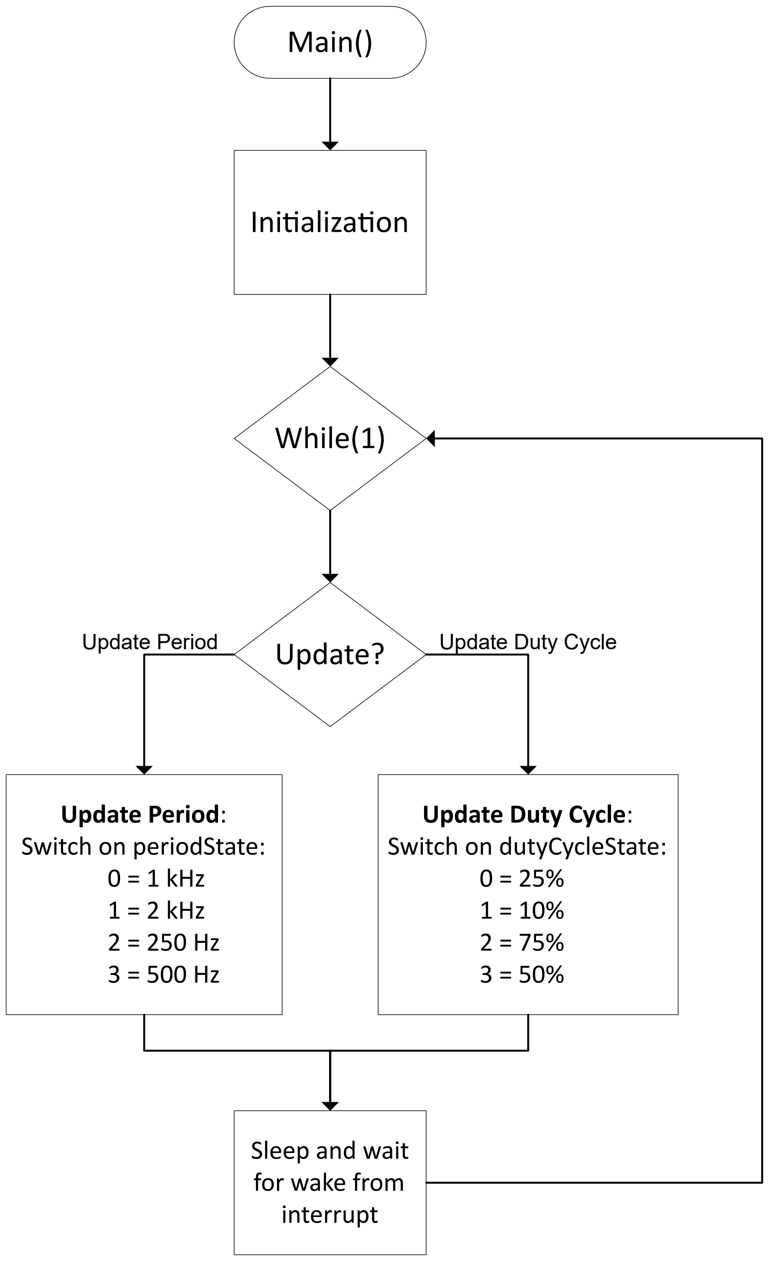


Figure 5-1. Application Main() Software Flow Chart

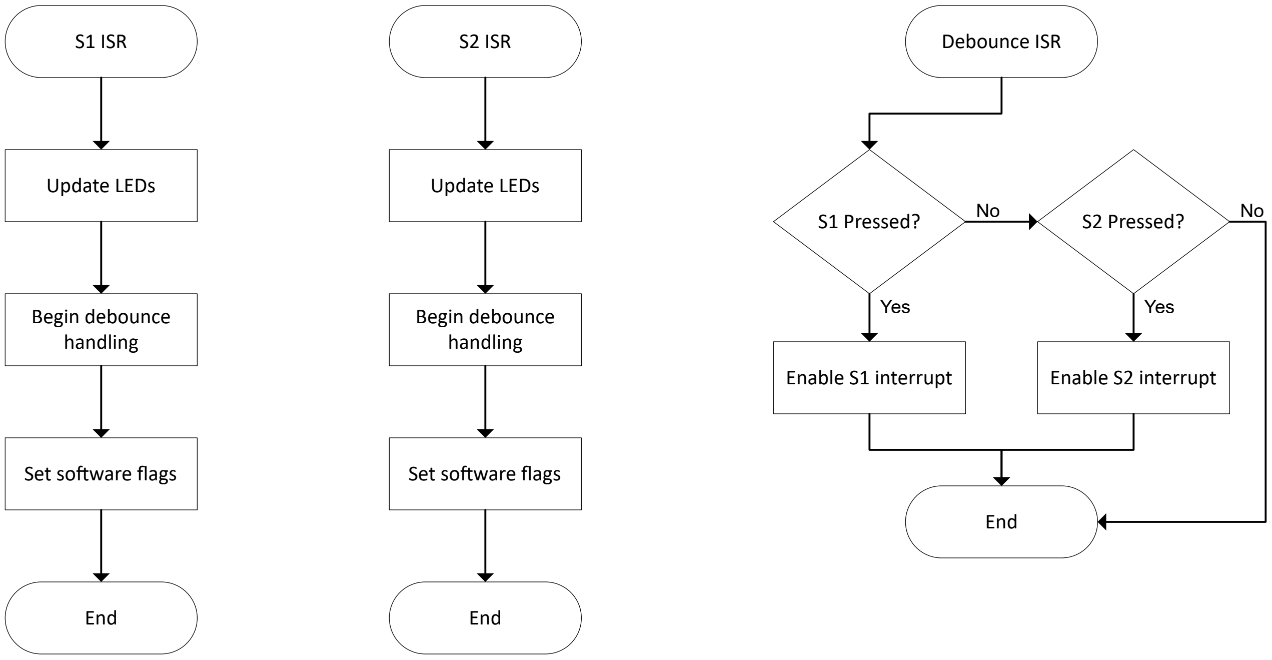
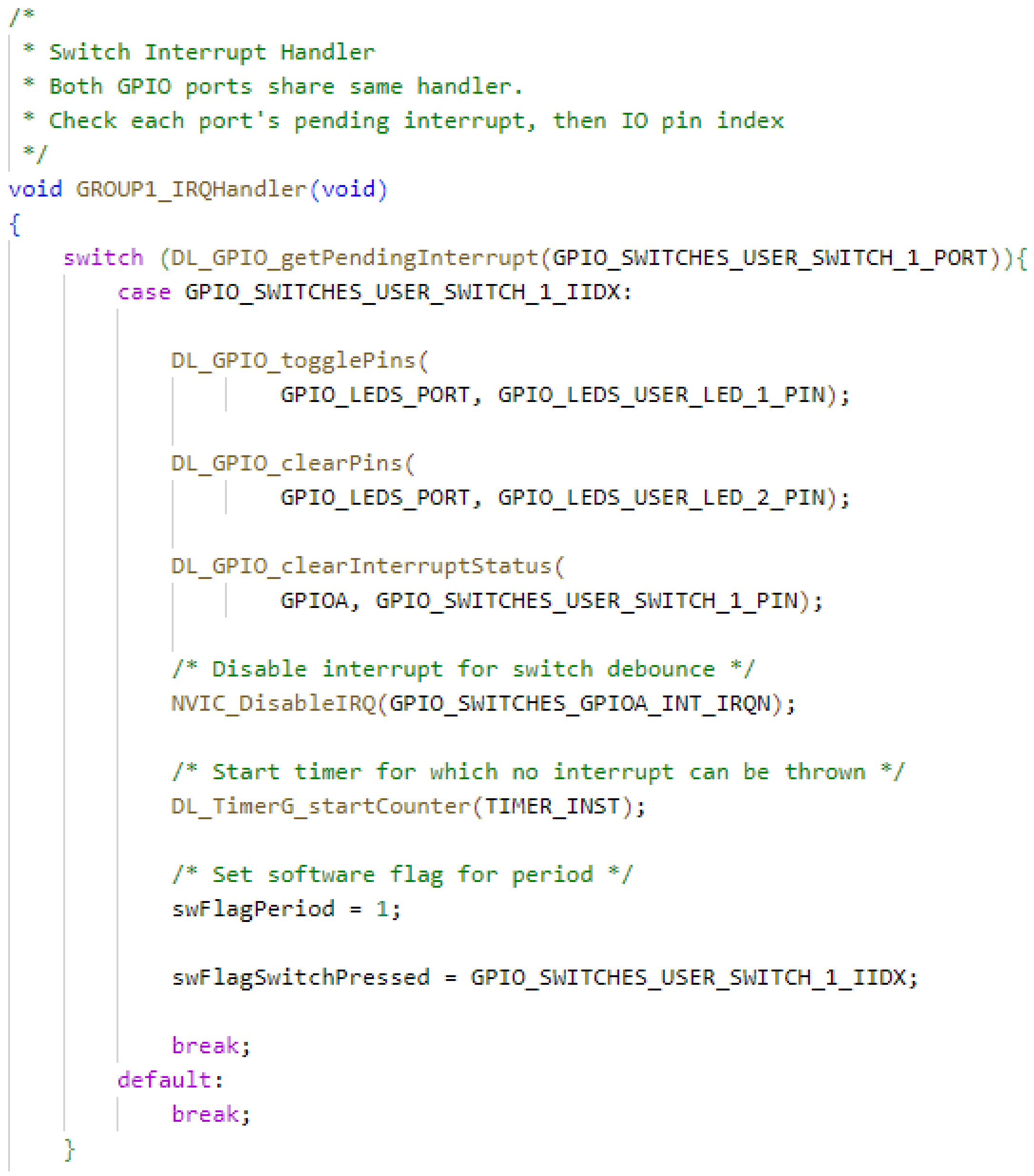


Figure 5-2. Application ISR Software Flow Charts

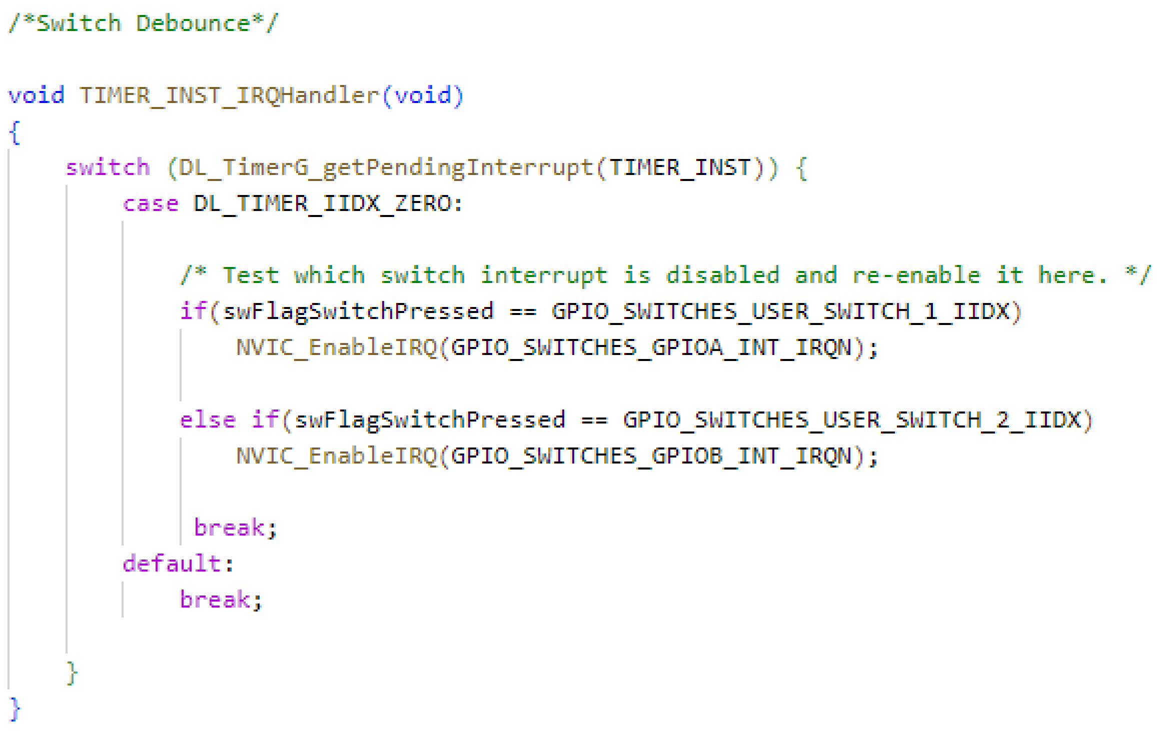
6 Application Code

The application code cycles through the different configurations for frequency and duty cycle based on interrupts triggered by button presses from S1 and S2. To handle button debouncing, the corresponding interrupt is disabled in the interrupt handler, and re-enabled about 10ms after exiting.



```
/*
 * Switch Interrupt Handler
 * Both GPIO ports share same handler.
 * Check each port's pending interrupt, then IO pin index
 */
void GROUP1_IRQHandler(void)
{
    switch (DL_GPIO_getPendingInterrupt(GPIO_SWITCHES_USER_SWITCH_1_PORT)){
        case GPIO_SWITCHES_USER_SWITCH_1_IIDX:

            DL_GPIO_togglePins(
                    GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_1_PIN);

            DL_GPIO_clearPins(
                    GPIO_LEDS_PORT, GPIO_LEDS_USER_LED_2_PIN);

            DL_GPIO_clearInterruptStatus(
                    GPIOA, GPIO_SWITCHES_USER_SWITCH_1_PIN);

            /* Disable interrupt for switch debounce */
            NVIC_DisableIRQ(GPIO_SWITCHES_GPIOA_INT_IRQN);

            /* Start timer for which no interrupt can be thrown */
            DL_TimerG_startCounter(TIMER_INST);

            /* Set software flag for period */
            swFlagPeriod = 1;

            swFlagSwitchPressed = GPIO_SWITCHES_USER_SWITCH_1_IIDX;

            break;
        default:
            break;
    }
```

Figure 6-1. Switch 1 Interrupt Handler



```
/*Switch Debounce*/

void TIMER_INST_IRQHandler(void)
{
    switch (DL_TimerG_getPendingInterrupt(TIMER_INST)) {
        case DL_TIMER_IIDX_ZERO:

            /* Test which switch interrupt is disabled and re-enable it here. */
            if(swFlagSwitchPressed == GPIO_SWITCHES_USER_SWITCH_1_IIDX)
                NVIC_EnableIRQ(GPIO_SWITCHES_GPIOA_INT_IRQN);

            else if(swFlagSwitchPressed == GPIO_SWITCHES_USER_SWITCH_2_IIDX)
                NVIC_EnableIRQ(GPIO_SWITCHES_GPIOB_INT_IRQN);

             break;
        default:
            break;

    }
}
```

Figure 6-2. Switch Debounce Interrupt Handler

7 Results

Figure 7-1 and Figure 7-2 show the results of the push-button controlled PWM subsystem example.

The varying duty cycles are changed by pressing S2 on the LaunchPad. The duty cycle only changes for CC0.

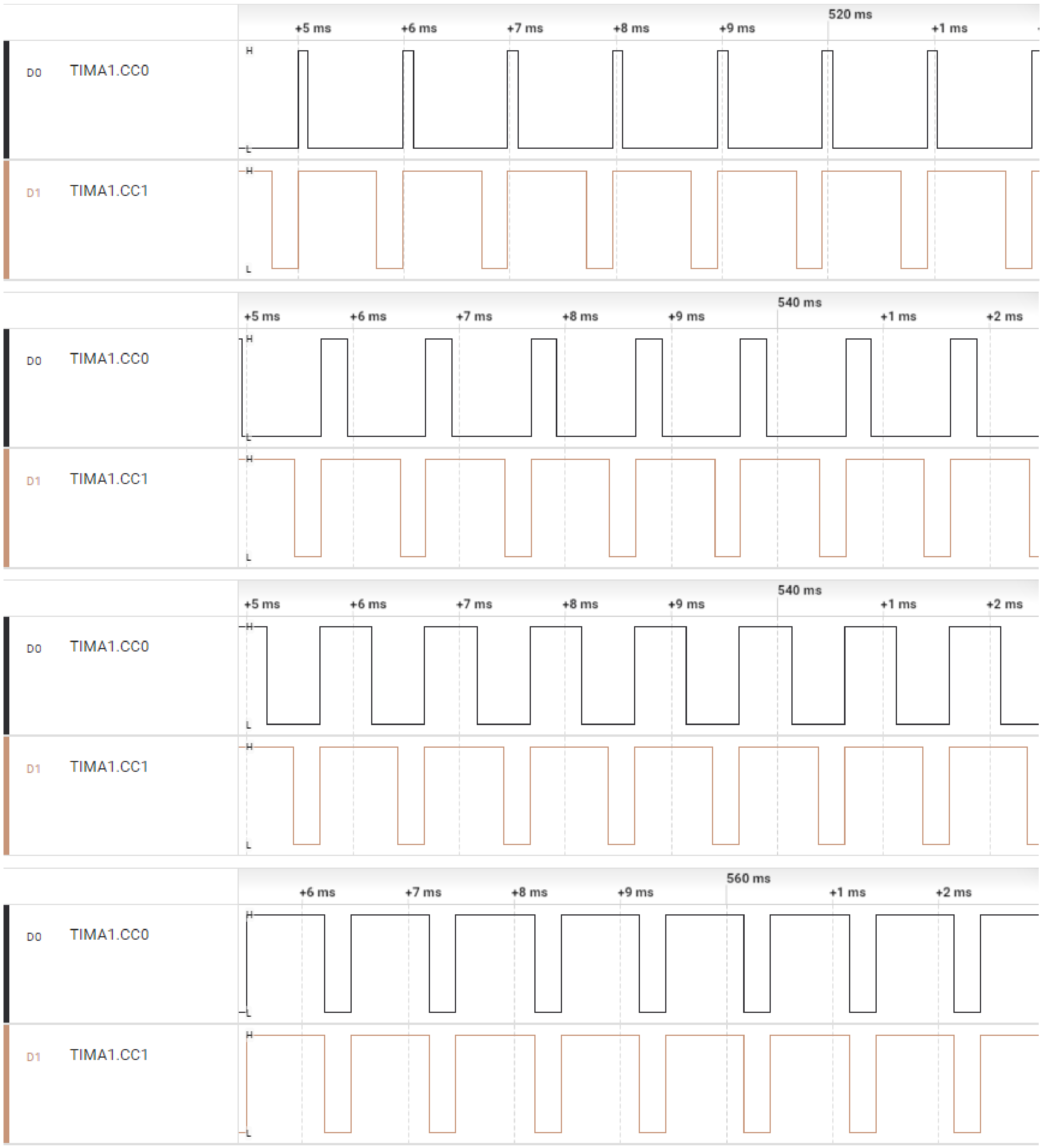


Figure 7-1. 1kHz Frequency With Duty Cycle Varying Between 10%, 25%, 50%, and 75%

The varying periods are changed by pressing S1 on the LaunchPad. The period changes for both CC0 and CC1.

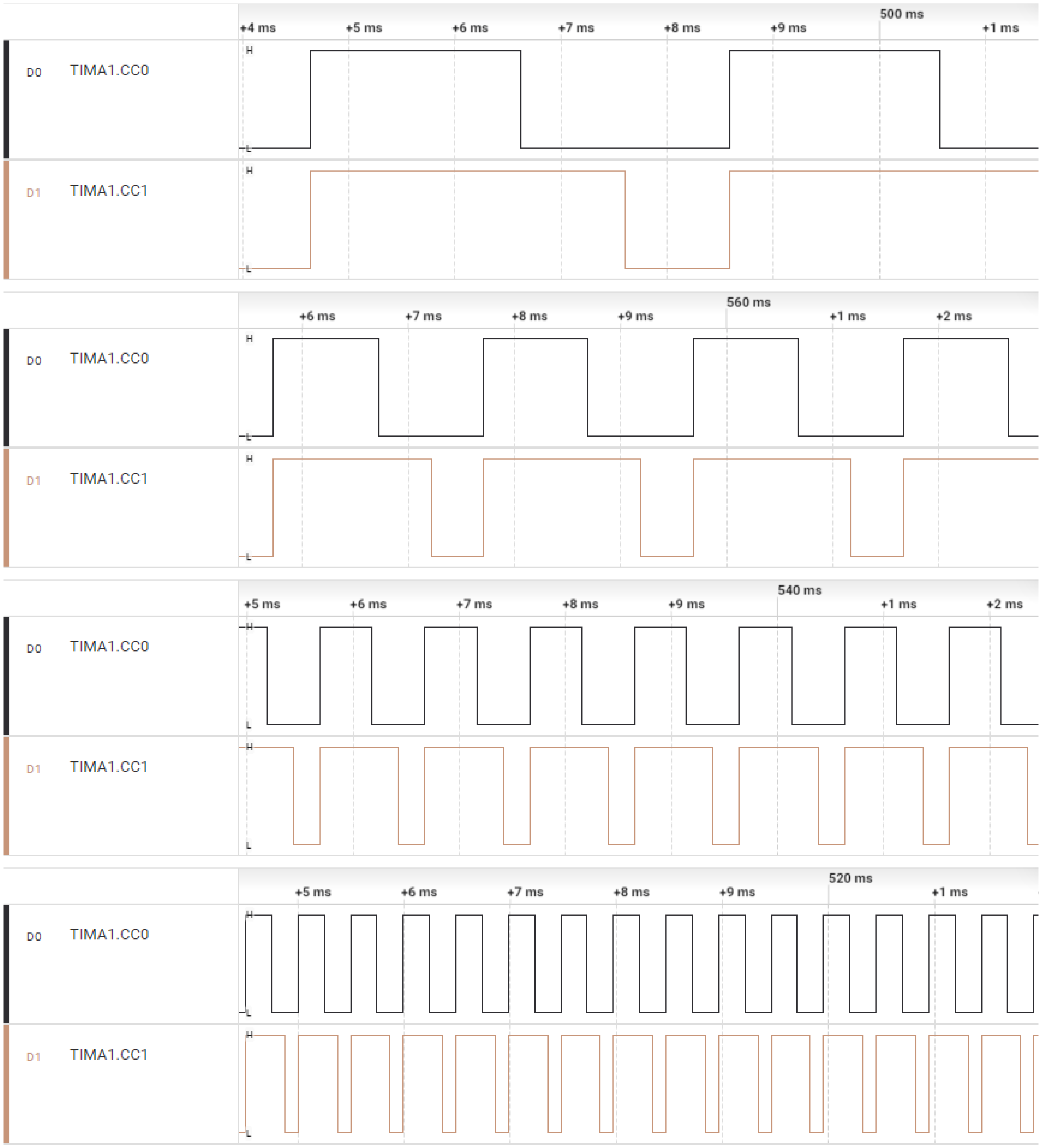


Figure 7-2. 50% Duty Cycle With Frequency Varying Between 250Hz, 500Hz, 1kHz, and 2kHz

8 Additional Resources

- Texas Instruments, Download the MSPM0 SDK
- Texas Instruments, Learn more about SysConfig
- Texas Instruments, MSPM0C LaunchPad™ Development Kit
- Texas Instruments, MSPM0L LaunchPad™ Development Kit
- Texas Instruments, MSPM0G LaunchPad™ Development Kit
- Texas Instruments, MSPM0 Academy

9 E2E

See the TI E2E™ support forums to view discussions and post new threads to get technical support for utilizing MSPM0 devices in designs.