AN14196

Flex Pulse Width Modulator (FlexPWM) Usage on MCX N Series

Rev. 1.0 — 2 May 2024 Application note

Document information

Information	Content
Keywords	AN14196, FlexPWM, MCX Nx4x, MCX N23x, motor control, PWM
Abstract	This document describes how to use the FlexPWM module on MCX N series. This document introduces several operation modes, including the corresponding implementation, to provide reference for different applications.

1 Introduction

The Flex Pulse Width Modulator (FlexPWM) module contains PWM submodules, each of which can be used to control a single half-bridge power stage. This module generates various switching patterns, including highly sophisticated waveforms, which is ideal for controlling the motor.

The FlexPWM module has the following main features:

- 16-bit resolution for center, edge-aligned, and asymmetrical PWMs
- Dithering to simulate enhanced resolution when fine edge placement is not available
- PWM outputs operating as complementary pairs or independent channels
- Ability to accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Support for synchronization to external hardware or other PWM
- Double buffered PWM registers:
  - Integral reload rates from 1 to 16
  - Half-cycle reload capability
- Multiple output trigger events can be generated per PWM cycle via hardware
- Support for double-switching PWM outputs
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom dead time insertion
- Each complementary pair can operate with its own PWM frequency and dead time values
- Individual software control for each PWM output
- All outputs can be programmed to change simultaneously via a FORCE_OUT event
- PWM_X pin can optionally output a third PWM signal from each submodule
- Channels that are not used for PWM generation can be used for buffered output compare functions and for input capture functions
- Enhanced dual-edge capture functionality

This document describes how to use the FlexPWM module on the MCX N series. This document introduces several operation modes, including the corresponding implementation, to provide reference for different applications. In the meantime, PWM operation logic, such as register reload logic and fractional delay logic helps the user further understand the FlexPWM module.

2 Block diagram

Figure 1 shows the FlexPWM block diagram:

- This device has two instances of the PWM module: PWM0 and PWM1.
- Each module contains four submodules (each submodule has its own timebase) and has four fault channels. Each channel accommodates four distinct fault inputs.
- Each FAULTx pin can be mapped arbitrarily to control any combination of the PWM outputs. By default, submodule 0 is regarded as the master for internal synchronization control.
- Only submodule 0 outputs control signals master reload, master force, master synchronize (sync), and aux clock and other submodules (1/2/3) or external components receive them.
- Alternatively, the external signals PWM_EXT_SYNC, EXT_FORCE, and EXT_CLK can control and synchronize together these four submodules.
- In addition, each submodule can generate independent output trigger signals to trigger events in other components, and an interrupt signal for a CPU interrupt response.

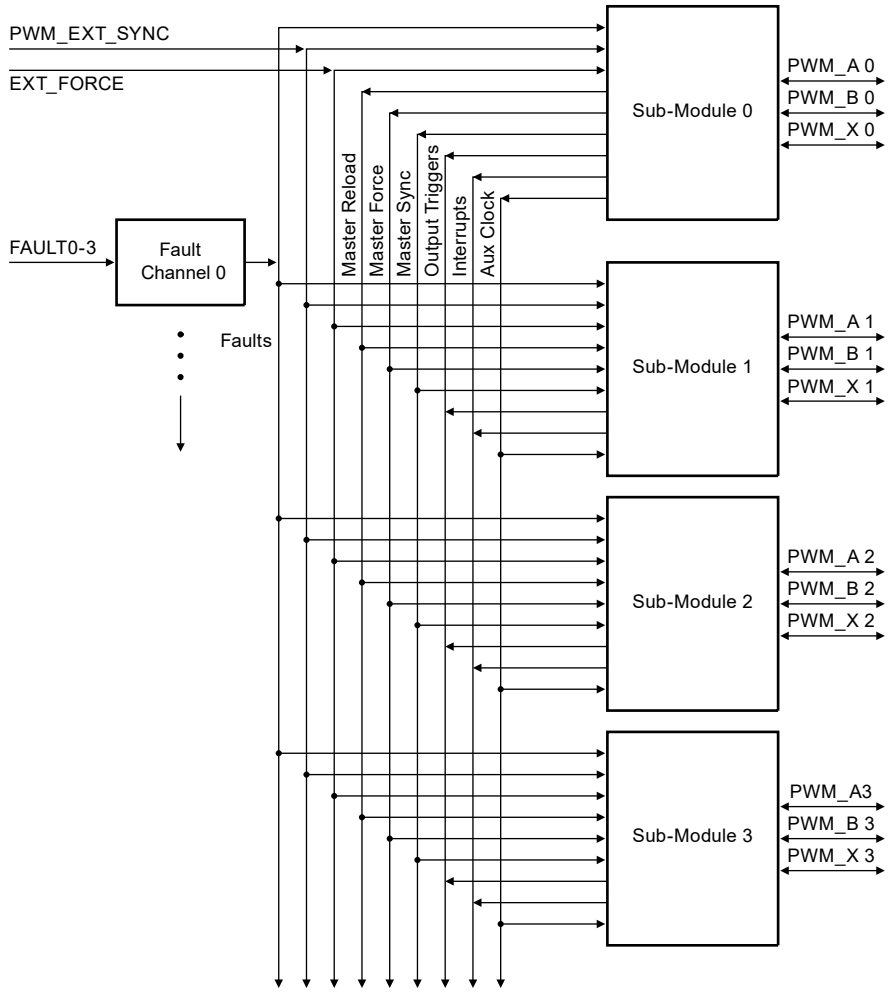

Figure 1. FlexPWM block diagram

Figure 2 shows the details of the PWM submodule:

- In each case, two comparators and associated VALx registers are used for each PWM output signal.
- One comparator and VALx register control the turn-on edge, while a second comparator and VALx register control the turn-off edge.
- The generation of the local sync signal is performed the same way as the other PWM signals in the submodule.
- While comparator 0 causes a rising edge of the local sync signal, comparator 1 generates a falling edge.
- Comparator 1 is also hardwired to the reload logic to generate the full-cycle reload indicator.

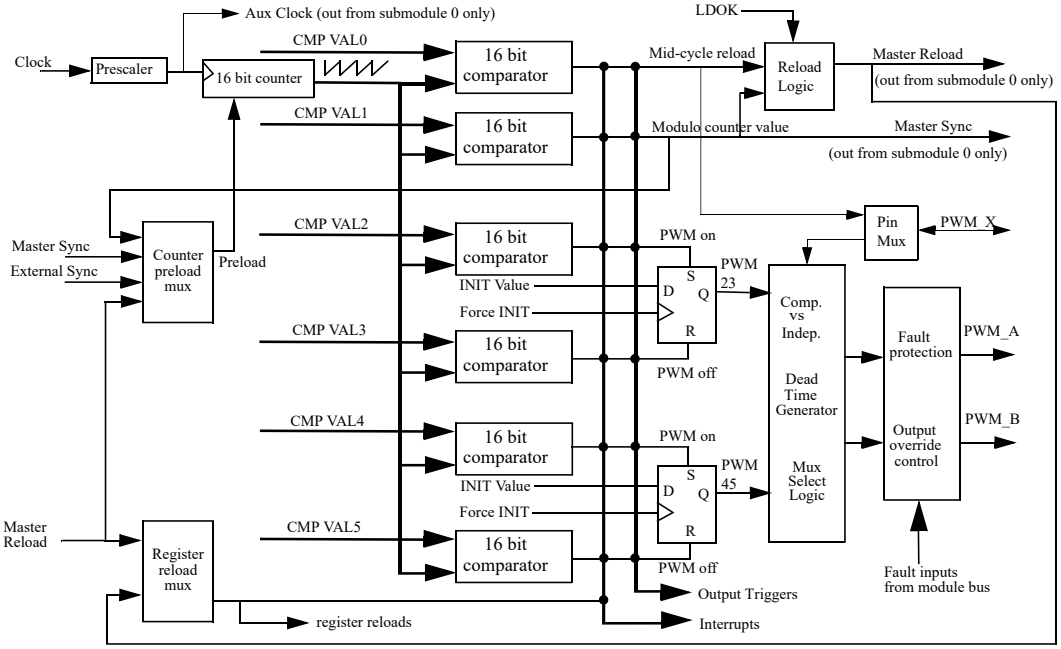

Figure 2. PWM submodule block diagram

- If VAL1 is controlling the modulus of the counter and VAL0 is half of the VAL1 register minus the INIT value, then the half-cycle reload pulse occurs halfway through the timer count period. Also, the local sync has a 50 % duty cycle.
- On the other hand, if the VAL1 and VAL0 registers are not required for register reloading or counter initialization, they can be used to modulate the duty cycle of the local sync signal. Therefore, effectively turning it into an auxiliary PWM signal (PWM_X). It is assumed that the PWM_X pin is not used for another function such as input capture or dead time distortion correction.
- Including the local sync signal, each submodule generates three PWM signals where the software has complete control over each edge of each of the signals.
- If the comparators and edge value registers are not required for PWM generation, use them for other functions. The other functions can be output compares, generating output triggers, or generating interrupts at time intervals.
- The 16-bit comparators shown in Figure 2 are "equal to or greater than" not just "equal to" the comparators.
- In addition, if both the set and reset of the flip-flop are asserted, then the flop output goes to 0.

3 Functional description

FlexPWM is a powerful module mainly for generating PWM and implementing many different PWM functions, as described in this section.

3.1 PWM capability

The MCX N series FlexPWM module contains four submodules, each of which has its own timebase and PWM capability. They can work independently of each other or in synchronization. In this section, only one submodule is used as an example.

3.1.1 Center-aligned PWM

Center-aligned PWM provides a single PWM signal. Here, half of the PWM period appears before a center point and the other half after the center point, as shown in Figure 3. It reduces 1-bit duty cycle resolution, because the PWM duty cycle can only be changed twice per PWM clock.

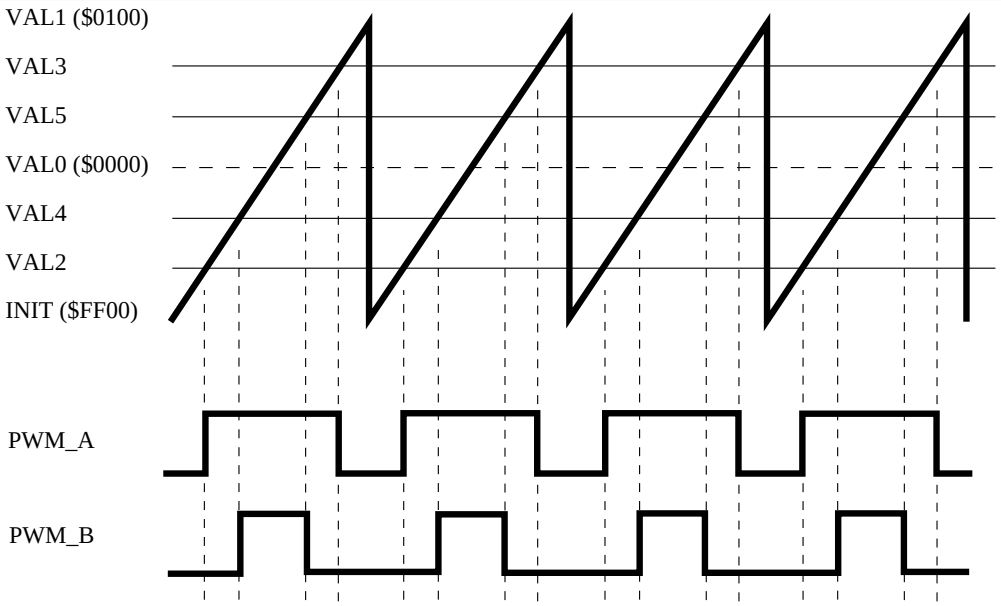

Figure 3. Center-aligned PWM example

For a center-aligned PWM on the MCX N series, the center point is specified to be half the value of the sum of the INIT value and VAL1. Therefore, the turn-on edge value (VAL2, VAL4) for each pulse must be updated so that the difference between center point and turn-on edge is half the duty cycle. Also, update the turn-off edge value (VAL3, VAL5) so that the difference between turn-off edge and center point is half the duty cycle.

If all PWM signal edge calculations follow this same convention, then the signals are center-aligned relative to each other. The center alignment between the signals is not restricted to symmetry around the zero count value, because any other number can also work. However, centering on zero provides the greatest range in signed mode and also simplifies the calculations.

The center-aligned PWM is used for multiple power switches or multi-phase converters, such as, half-bridge and full-bridge inverter. Using this PWM greatly improves system EMI and THD.

3.1.2 Edge-aligned PWM

An edge-aligned PWM provides a single PWM signal. Here, one out of two edges of the PWM are aligned to the period boundary and the duty cycle determines the other edge, as shown in Figure 4. It allows maximum duty cycle resolution using an edge-aligned PWM, because the PWM duty cycle can be changed per PWM clock.

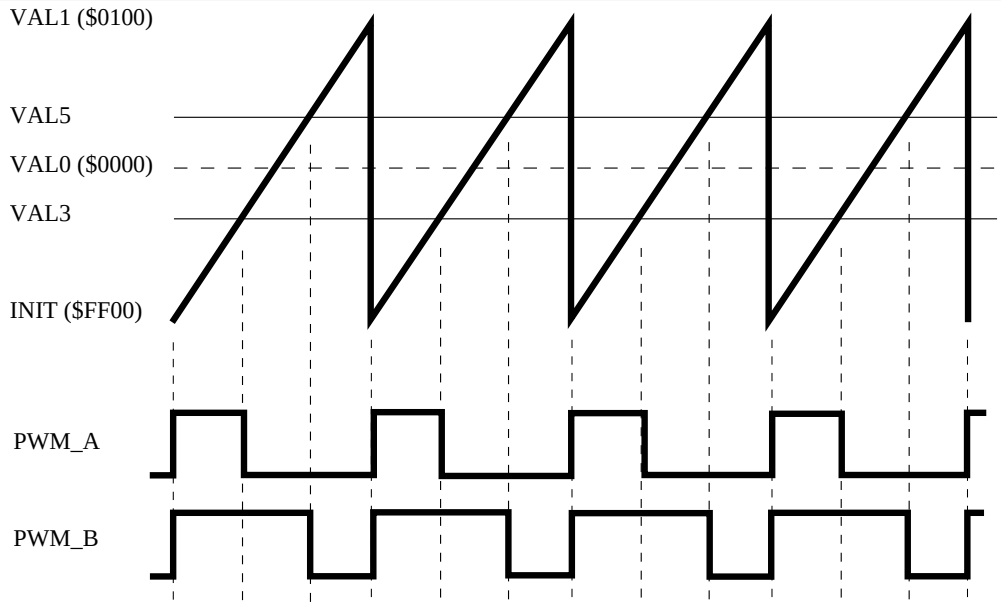

Figure 4. Edge-aligned PWM example

For an edge-aligned PWM on the MCX N series, the turn-on edge value (VAL2, VAL4) for each pulse is specified to be the INIT value. Therefore, only the turn-off edge value (VAL3, VAL5) must be updated to change the duty cycle.

3.1.3 Phase-shifted PWM

Phase-shifted PWM provides PWM signals that are synchronized but phase-shifted relative to each other, as shown in Figure 5. Phase shifting does not affect the duty cycle. Therefore, the average load voltage is not affected.

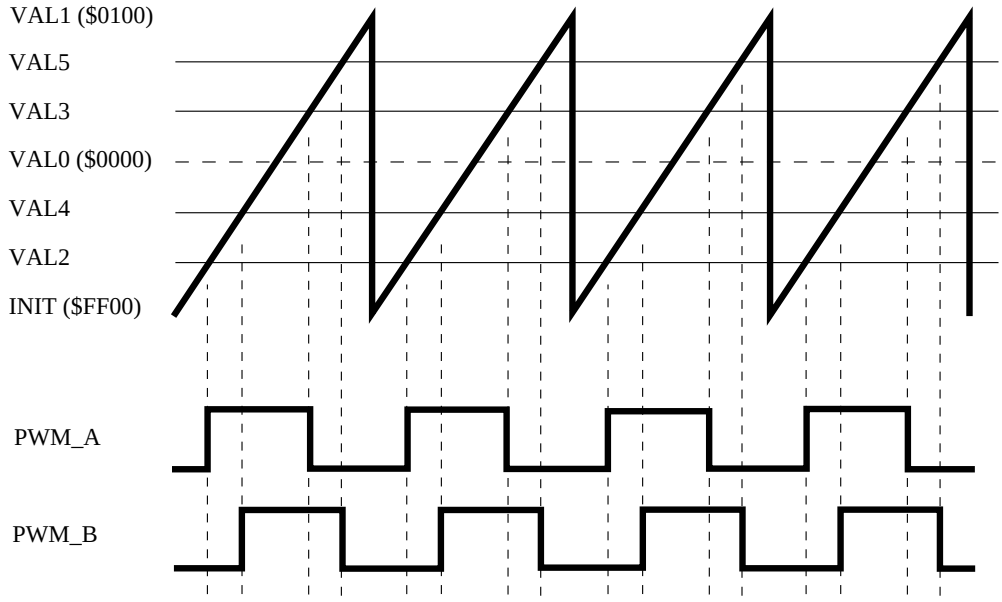

Figure 5. Phase-shifted PWM example

For phase-shifted PWM on the MCX N series, the duty cycles of PWM_A and PWM_B are the same. However, PWM_B has a phase delay relative to PWM_A.

VLA2 and VAL3 define the output of PWM_A. The phase-shift value must be added to VAL2 and VAL3 respectively to get the values of VAL4 and VAL5.

Note: *When a phase shift is required between submodule 0 and other submodules, use the PHASEDLY register instead of adding an offset to the turn-on and turn-off edges.*

3.1.4 Double-switching PWM

Double-switching PWM (DBLPWM) output is supported to aid in single shunt current measurement and three-phase reconstruction. This method supports two independent rising edges and two independent falling edges per PWM cycle. The VAL2 and VAL3 registers are used to generate the PWM_A. The VAL4 and VAL5 are used to generate the PWM_B. The two channels, PWM_A and PWM_B from force out logic, are combined using XOR logic (force out logic), as shown in Figure 6. The DBLPWM signal can be run-through the dead time insertion logic.

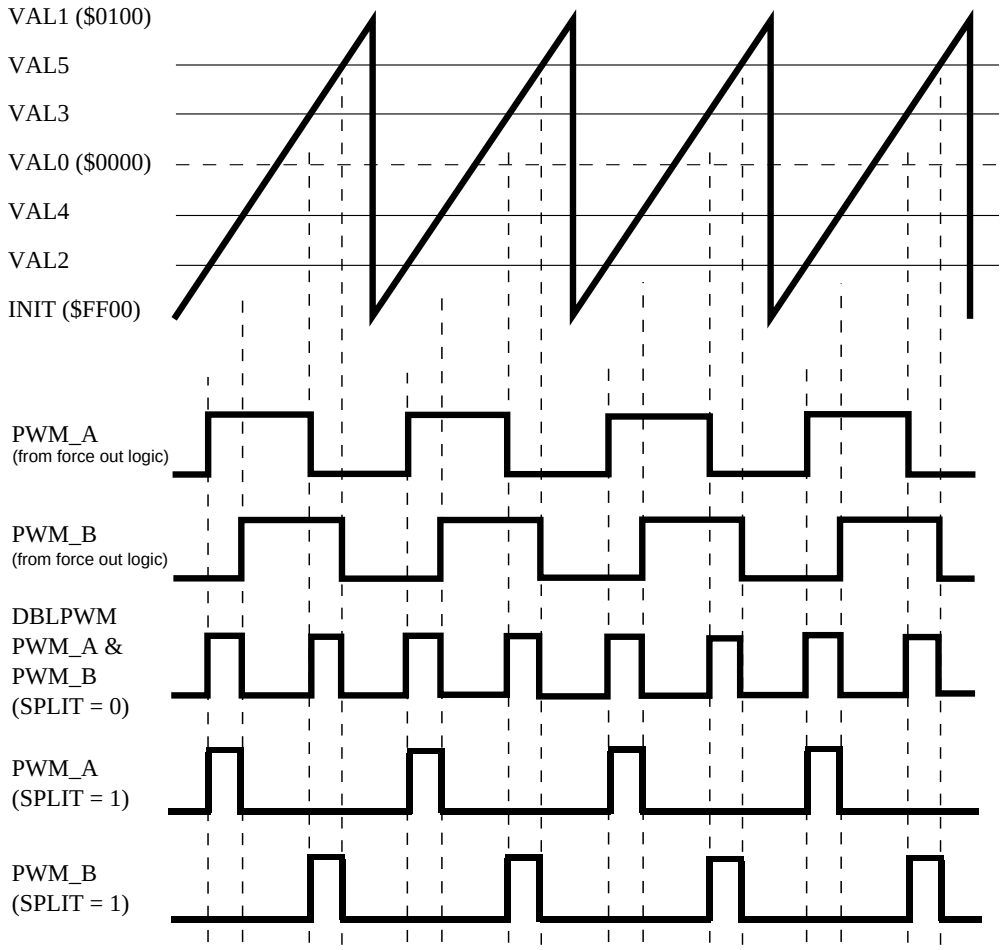

Figure 6. Double-switching PWM example

3.1.5 ADC triggering

In cases where the timing of the ADC triggering is critical, it must be scheduled as a hardware event instead of software activated. With this PWM module, multiple ADC triggers can be generated in hardware per PWM cycle without the requirement of another timer module, as shown in Figure 7. When specifying a complementary mode of operation, only two edge comparators are required to generate the output PWM signals for a given submodule. It means that the other comparators are free to perform other functions. In this example, the

software does not need to respond quickly after the first conversion to set up other conversions that must occur in the same PWM cycle.

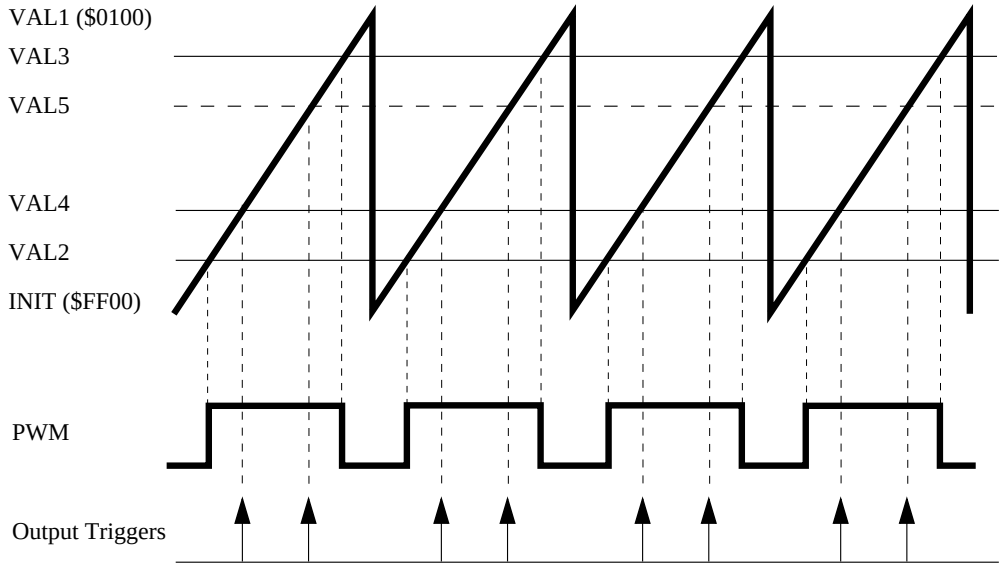

Figure 7. Multiple output trigger generations in hardware

Each submodule has its own timer, therefore, it is possible for each submodule to run at a different frequency. One of the options possible is to have one or more submodules running at a lower frequency, but still synchronized to the timer in submodule 0.

Figure 8 shows how this feature can be used to schedule ADC triggers over multiple PWM cycles. Use the lower-frequency submodule to control the sampling frequency of the software control algorithm where multiple ADC triggers are now scheduled over the entire sampling period. Figure 8 shows all submodule comparators being used for ADC trigger generation.

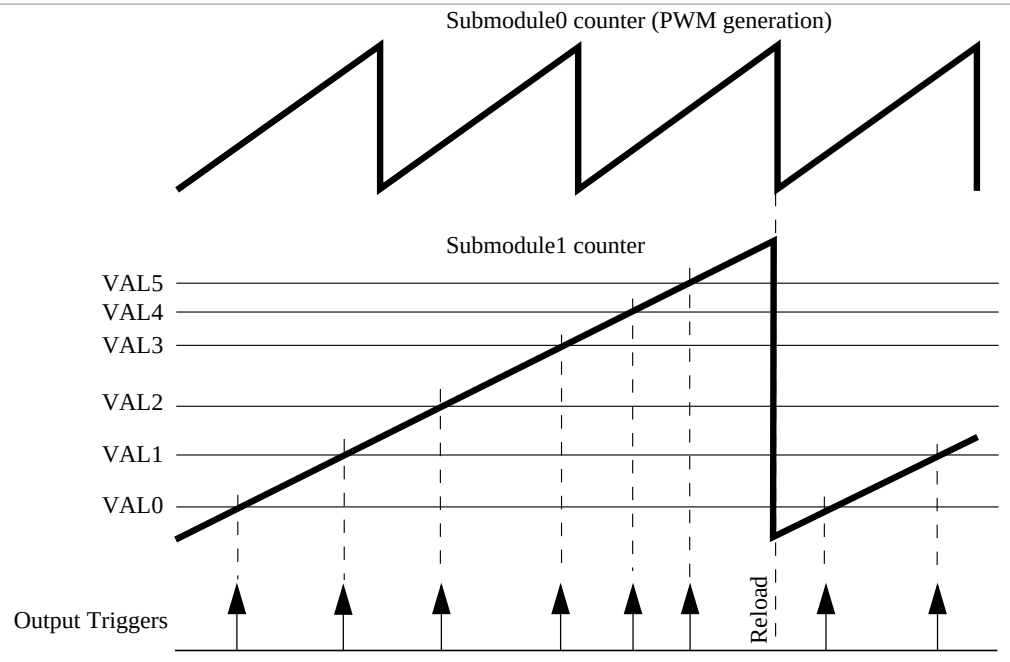

Figure 8. Multiple output triggers over several PWM cycles

3.1.6 Enhanced capture capabilities (E-capture)

When a PWM pin is not used for PWM generation, it can be used to perform input captures. For PWM generation, compare register values specify both edges of the PWM signals. When the PWM pin is programmed for input capture, both registers work on the same pin to capture multiple edges. The capture toggles from one (CVAL0) to the other (CVAL1), in either a free-running or one-shot fashion. Programming the desired edge of each capture circuit, period, and pulse width of an input signal can be measured easily without rearming the circuit. In addition, each edge of the input signal can clock an 8-bit counter where the counter output is compared to a user-specified value (EDGCMP). When the counter output equals EDGCMP, the value of the submodule timer is captured and the counter is automatically reset. This feature counts a specified number of edge events and then performs a capture and interrupt. Figure 9 illustrates some of the functionality of the E-capture circuit.

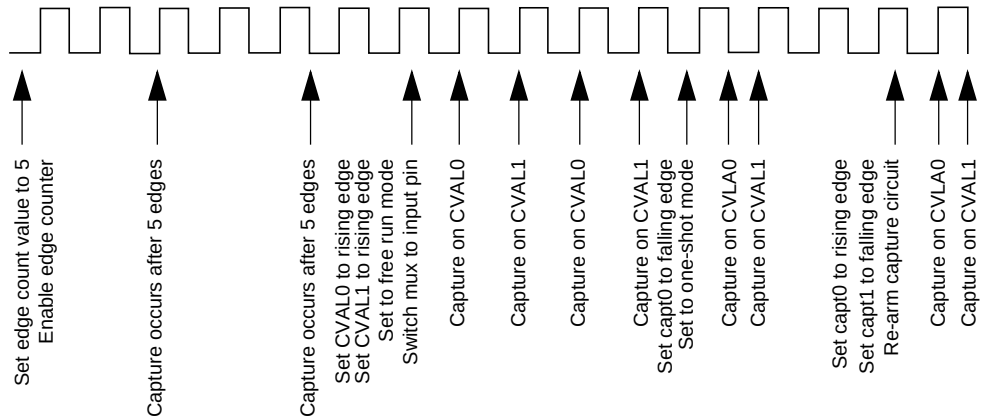

Figure 9. Capture capabilities of the E-capture circuit

When a submodule is used for PWM generation, its timer counts to the modulus value used to specify the PWM frequency and is reinitialized. This timer does not count through all the numbers and the timer reset represents a discontinuity in the 16-bit number range. Therefore, using this timer for input captures on one of the other pins (for example, PWM_X) has limited utility. However, when measuring a signal that is synchronous to the PWM frequency, the timer modulus range is suited for the application.

As shown in Figure 10, the output of a PWM power stage is connected to the PWM_X pin that is configured for free running input captures. The CVAL0 capture circuitry is programmed for rising edges and the CVAL1 capture circuitry is set for falling edges. Therefore, a new load pulse width data is acquired every PWM cycle. To calculate the pulse width, subtract the CVAL0 register value from the CVAL1 register value. This measurement is beneficial when performing dead time distortion correction on a half-bridge circuit driving an inductive load. Also, these values can be compared directly to the VALx registers responsible for generating the PWM outputs to obtain a measurement of system propagation delays.

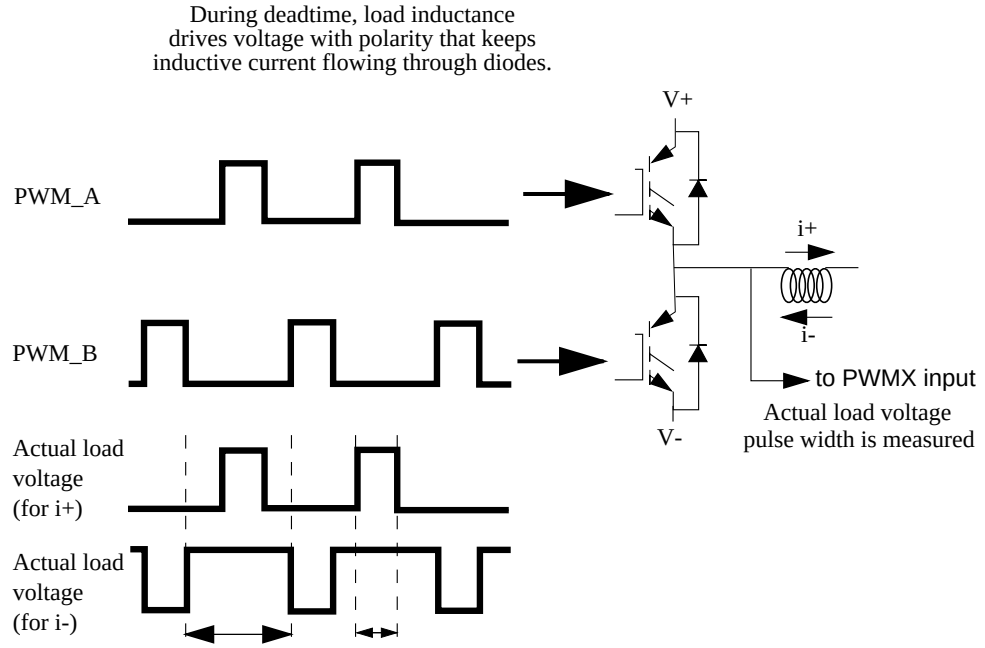

Figure 10. Output pulse width measurement is possible with the E-Capture circuit

3.2 Operation

This section describes the operation logic of the PWM in detail.

Figure 11 is a high-level block diagram of output PWM generation.

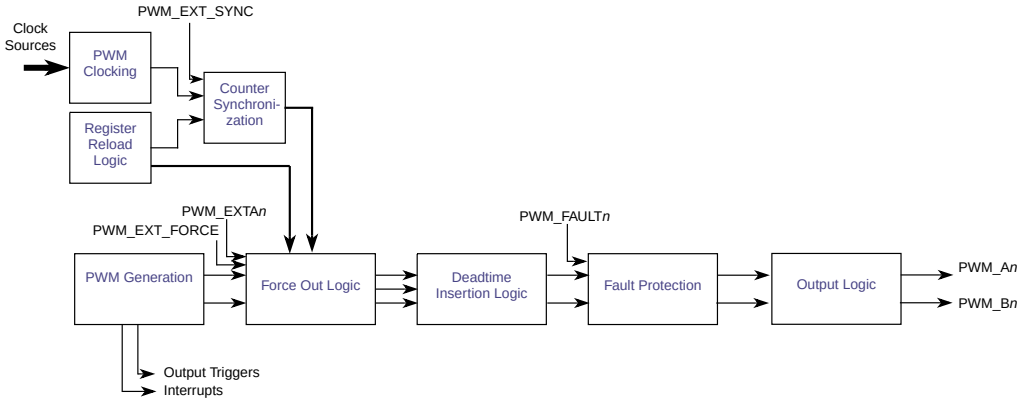

Figure 11. High-level output PWM generation block diagram

3.2.1 Register reload logic

The register reload logic determines when the outer set of registers for all double-buffered register pairs must be transferred to the inner set of registers. The register reload event can be scheduled to occur every "n" PWM cycle using CTRL[LDFQ] and CTRL[FULL], which is defined by the VAL1 register. A half-cycle reload option is also supported (CTRL[HALF]) where the reload can take place in the middle of a PWM cycle. The half-cycle point is defined by the VAL0 register and need not be exactly in the middle of the PWM cycle.

As shown in Figure 12, the reload signal from submodule 0 can be broadcast as the master reload signal. As a result, allowing the reload logic from submodule 0 to control the reload of registers in other submodules.

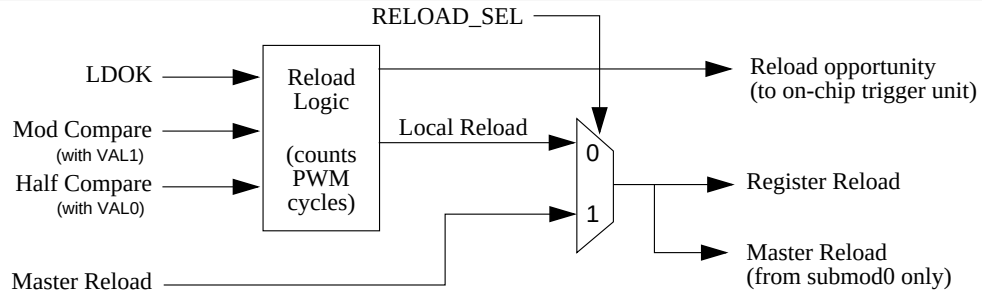

Figure 12. Register reload logic

3.2.2 Counter synchronization

The 16-bit counter shown in Figure 13 can be initialized with an INIT value by the following four possible sources:

- Local sync
- Master reload
- Master sync
- PWM_EXT_SYNC

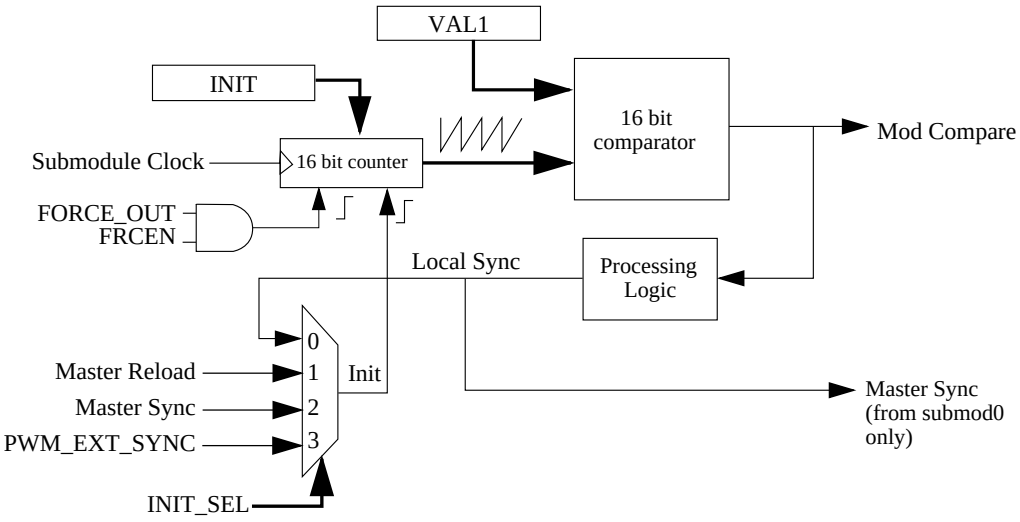

Figure 13. Submodule counter synchronization

If local sync is selected as the counter initialization signal, the counter counts until its output equals VAL1. VAL1 is used to specify the counter modulus value. Then, VAL1 within the submodule effectively controls the timer period (and therefore, the PWM frequency generated by that submodule) and everything works on a local level. To initialize the counter, configure the master sync signal (which originates as the local sync from submodule 0). As a result, the counter period of any submodule can be locked to the period of the counter in submodule 0. The VAL1 register and associated comparator of the other submodules are free for other functions such as PWM generation, input captures, output compares, or output triggers.

If the master reload signal is selected as the source for counter initialization, then the period of the counter of any submodule is locked to the reload frequency of submodule 0.

Note: *This master reload signal only originates from submodule 0.*

Since the reload frequency is optional and can vary from one to sixteen, it supports generating multi-frequency PWM signals in synchronization.

If the PWM_EXT_SYNC signal is selected as the source for counter initialization, an external source can control the counter period in all submodules.

Note: *This PWM_EXT_SYNC signal originates on-chip or off-chip depending on the system architecture.*

As a result, it gets easier to implement synchronization between the FlexPWM module and the external signal.

In addition, the counter can be initialized optionally on the assertion of the FORCE_OUT signal, if CTRL2[FRCEN] is set. FORCE_OUT signal is provided mainly for synchronous switching of multiple PWM outputs. As shown in Figure 13, it constitutes a second initialization input into the counter.

As a result, the counter initializes regardless which signal is selected as the counter initialization signal.

In summary, if multiple PWMs with the same frequency are required to synchronize, the master sync signal (submodule 0 selects local sync for counter initialization) is recommended to initialize the counters in slave submodules (submodule 1/2/3) if all PWM signals are from internal submodules. Otherwise, if some PWM signals are from another PWM module or are off-chip, the PWM_EXT_SYNC signal is recommended to initialize the counters in all submodules.

If it is necessary to synchronize multiple PWMs with different frequencies, the following two methods are recommended:

- Method 1:
  1. Configure the highest frequency PWM signal from submodule 0. This submodule uses local sync to initialize the counter for internal PWM synchronization, or PWM_EXT_SYNC for external PWM synchronization.
  2. Select master reload (reload rate depends on the frequency ratio of different PWMs) signal for counter initialization in slave submodules (submodule 1/2/3).
- Method 2:
  1. To initialize counters for the submodules, which generate the higher frequency PWM signals, select local sync.
  2. Configure their FORCE source as master sync signal or EXT_FORCE signal.
     Note: *For the master sync signal, submodule 0 is configured to generate lowest frequency PWMs. For the EXT_FORCE signal, another slave submodule is configured to generate lowest frequency PWMs. This submodule is required to output a local sync trigger as an EXT_FORCE signal input for other submodules.*
  3. Configure the submodule (which generates lowest frequency PWMs) to select local sync to initialize the counter for internal PWM synchronization, or EXT_SYNC for external PWM synchronization.

3.2.3 Force out logic

For each submodule, the software can select between the following eight signal sources, for the FORCE_OUT signal depending on the chip architecture:

1. Local CTRL2[FORCE]
2. Master force signal from submodule 0
3. Local reload signal
4. Master reload signal from submodule 0
5. Local sync signal
6. Master sync signal from submodule 0
7. EXT_SYNC signal from on or off-chip
8. EXT_FORCE signal from on or off-chip

The local signals change the signals on the output pins of the submodule without regard for synchronization with other submodules. However, if all the signals on all the submodule outputs change at the same time, the master, EXT_SYNC, or EXT_FORCE signals must be selected.

Figure 14 illustrates the force logic. The SEL23 and SEL45 fields each choose from one of the following four signals that can be supplied to the submodule outputs:

- PWM signal
- Inverted PWM signal
- A binary level specified by software via the OUT23 and OUT45 bits
- The PWM_EXTA or PWM_EXTB alternate external control signals

The selection can be determined ahead of time. When a FORCE_OUT event occurs, these values are presented to the signal selection mux. This mux immediately switches the requested signal to the output of the mux for further processing downstream.

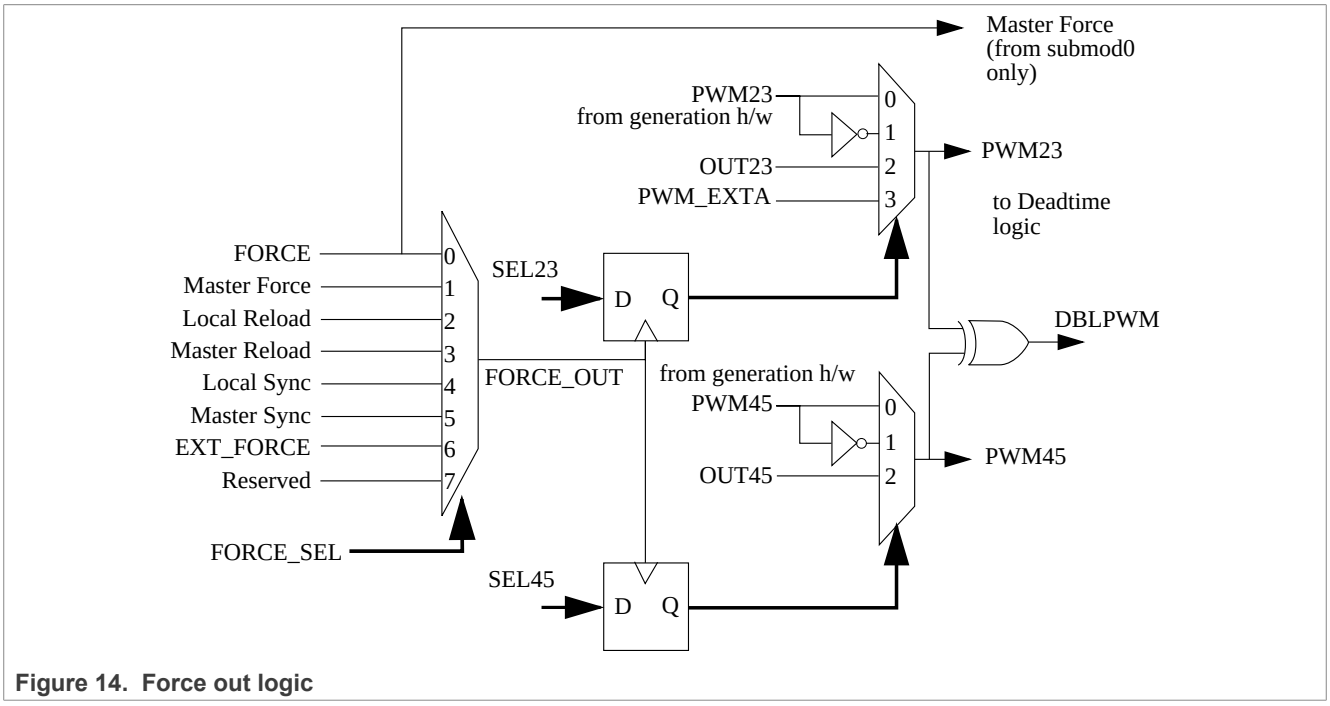

Figure 14. Force out logic

3.2.4 Independent or complementary channel operation

Writing a logic 1 to CTRL2[INDEP] configures the pair of PWM outputs as two independent PWM channels. The VALx pair of each PWM operating independently of the other output controls each PWM output.

Writing a logic 0 to CTRL2[INDEP] configures the PWM output as a pair of complementary channels. In complementary channel operation, the PWM pins are paired as shown in Figure 15. The MCTRL[IPOL] determines the signal that is connected to the output pin (PWM23 or PWM45).

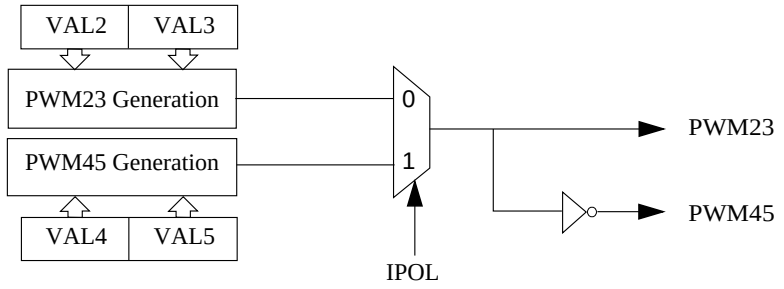

Figure 15. Complementary channel pair

The complementary channel operation is for driving top and bottom transistors in a motor drive circuit, as shown in Figure 16.

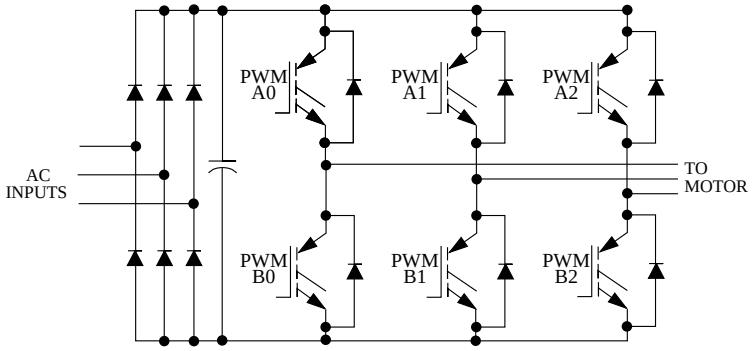

Figure 16. Typical 3-phase AC motor drive

The complementary operation allows the use of the dead time insertion feature.

3.2.5 Dead time insertion logic

Figure 17 shows the dead time insertion logic of each submodule, which is used to create non-overlapping complementary signals when not in independent mode.

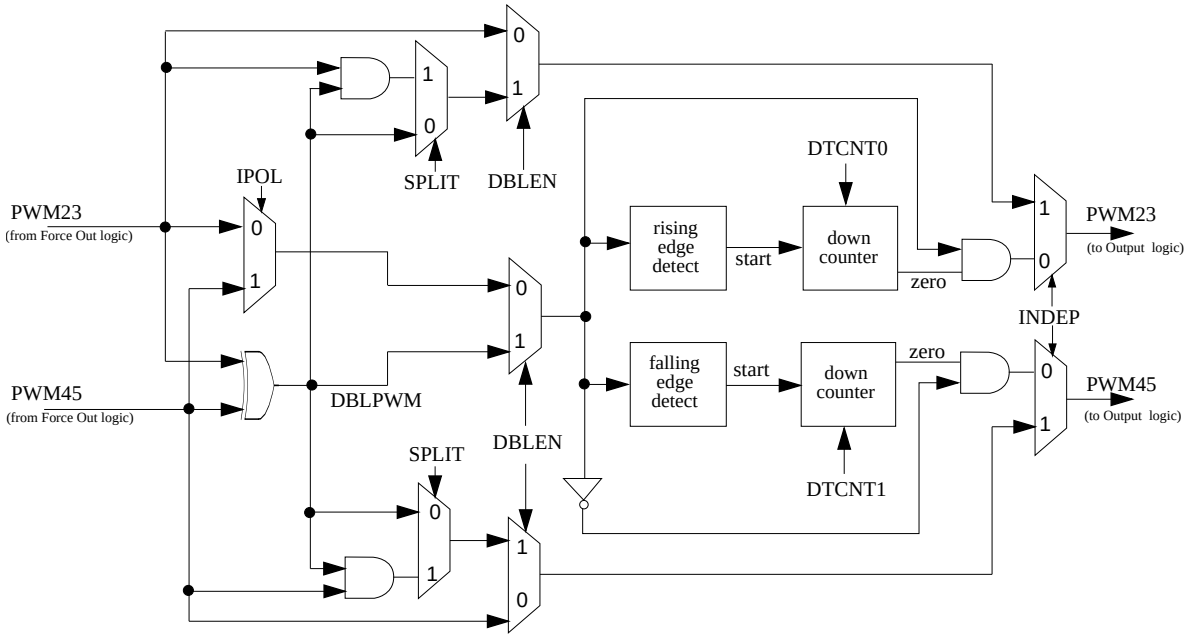

Figure 17. Dead time insertion logic

While in the complementary mode, a PWM pair can be used to drive top/bottom transistors, as shown in Figure 17. When the top PWM channel is active, the bottom PWM channel is inactive, and vice versa.

Note: *To avoid short-circuiting the DC bus and endangering the transistor, ensure that there is no overlap of conducting intervals between the top and bottom transistors. However, the characteristics of the transistor result in longer switching-off time than switching-on time. To avoid the conducting overlap of top and bottom transistors, dead time must be inserted in the switching period, as shown in Figure 18.*

The dead time generators insert software-selectable activation delays into the pair of PWM outputs automatically. The dead time registers (DTCNT0 and DTCNT1) specify the number of IPBus clock cycles to use

for dead time delay. Every time the rising edge and falling edge are detected, the down counter starts, and a dead time is inserted. Dead time forces PWM outputs in the pair to the inactive state.

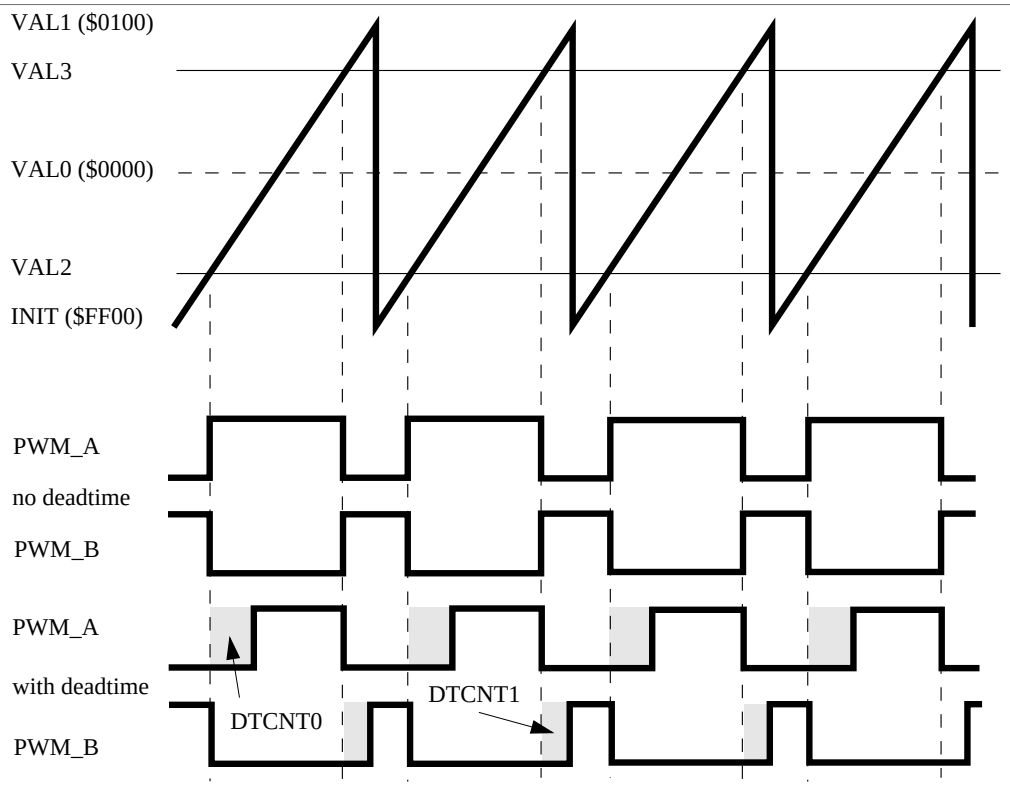

Figure 18. Dead time insertion

3.2.6 Fractional delay logic

Applications, which require more resolution than a single IPBus clock period, use the fractional delay logic. The fractional delay logic helps to achieve fine resolution on the rising and falling edges of the PWM_A and PWM_B outputs. Enable the use of the fractional delay logic by setting FRCTRL[FRACx_EN]. The FRACVALx registers act as a fractional clock cycle in addition to the turn-on and turn-off count specified by the VAL2, VAL3, VAL4, or VAL5 registers. The FRACVAL1 register acts as a fractional increase in the PWM period as defined by VAL1. If FRACVAL1 is programmed to a non-zero value, then the largest value for the VAL1 register is 0xFFFE for unsigned usage or 0x7FFE for signed usage. This limit is required to avoid counter rollovers when accumulating the more fractional period.

Both the fractional enables (1, 23, 45) and the fractional values (1 to 5) are double-buffered and reloaded at the same time as the value registers.

For each PWM cycle, increase the value compare point by 1. The value compare is increased when there is an overflow on the double-buffered value of the fractional register plus the 5-bit accumulated fractional value. The accumulated fractional value starts at zero, therefore, it is impossible to overflow the first PWM cycle.

At the end of each PWM cycle, if the corresponding fractional enable is set then the accumulated fractional value increments by the fractional value (double-buffered value). The accumulated fractional value is 5 bits, therefore, for the overflow case only the remainder is kept. If the corresponding fractional enable is clear, then the accumulated fractional value is reset. It is the only way to reset the accumulated fractional value.

The accumulated fractional values are not accessible to the software.

As an example, assume the following conditions:

- INIT = 0x0000
- VAL1 = 0x000F
- VAL2 = 0x0000
- VAL3 = 0x0007
- FRACVAL3 = 0x00

It causes the PWM output to have a 50 % duty cycle. It is high for a count value of 0x0 to 0x7; at which point it goes low until the counter reaches the maximum value of 0xF.

If FRACVAL3 changes to a value of 0x17, then the active time for the PWM output = 8 cycles + (23 / 32) cycle = 8.719 cycles. Therefore, a high duty cycle = (8.719 cycles / 16 cycles) x 100 % = 54.49 %.

Another case involves fine-tuning the PWM period using the FRACVAL1 register. If the user wants a period of 100.25 clock cycles, program VAL1 with 0x0064 and FRACVAL1 with 0x4000. The fractional value accumulates so that every 4 PWM cycles are one clock cycle longer (101 instead of 100). Rising and falling edges of the PWM outputs also uses the accumulated fraction to delay their edges and maintain a consistent 100.25 cycles spacing between corresponding edges from one cycle to the next.

3.2.6.1 Fractional delay logic without NanoEdge placement block

The PWM can use dithering to simulate fine edge control for the submodules not supported by the NanoEdge placer. Enable this feature by setting the FRCTRL[FRAC1_EN], FRCTRL[FRAC23_EN], and FRCTRL[FRAC45_EN] bits. The PWM period or the PWM edges dither from the nearest whole number values to achieve an average value equivalent to the programmed fractional value. The added cycles are based on the accumulation of the fractional component. For example, if the user wants the PWM period to be 50.25 clock cycles, program VAL1 with 0x0032 and FRACVAL1 with 0x4000. Mostly, the PWM period is going to be 50 cycles long but occasionally it is 51 cycles long to achieve a long-term average of 50.25 cycles.

For the submodules not supported by the NanoEdge placer, the clock frequency does not require any specific value for correct operation.

3.2.7 Output logic

Figure 19 shows the output logic of each submodule including how each PWM output has an individual fault disabling, polarity control, and output enable. It allows for maximum flexibility when interfacing with the external circuitry.

The PWM23 and PWM45 signals, which are output from the dead time logic (as shown in Figure 19) are positive true signals. In other words, the high-level signals must result in the corresponding transistor in the PWM inverter being turned ON. The voltage level required at the PWM output pin to turn the transistor ON or OFF is a logic function between the pin and the transistor. Therefore, it is imperative that the user program OCTRL[POLA] and OCTRL[POLB] before enabling the output pins. A fault condition can result in the PWM output being put in a high-impedance state, forced to a logic 1 state or logic 0 state. It depends on the values programmed into the OCTRL[PWMxFS] fields.

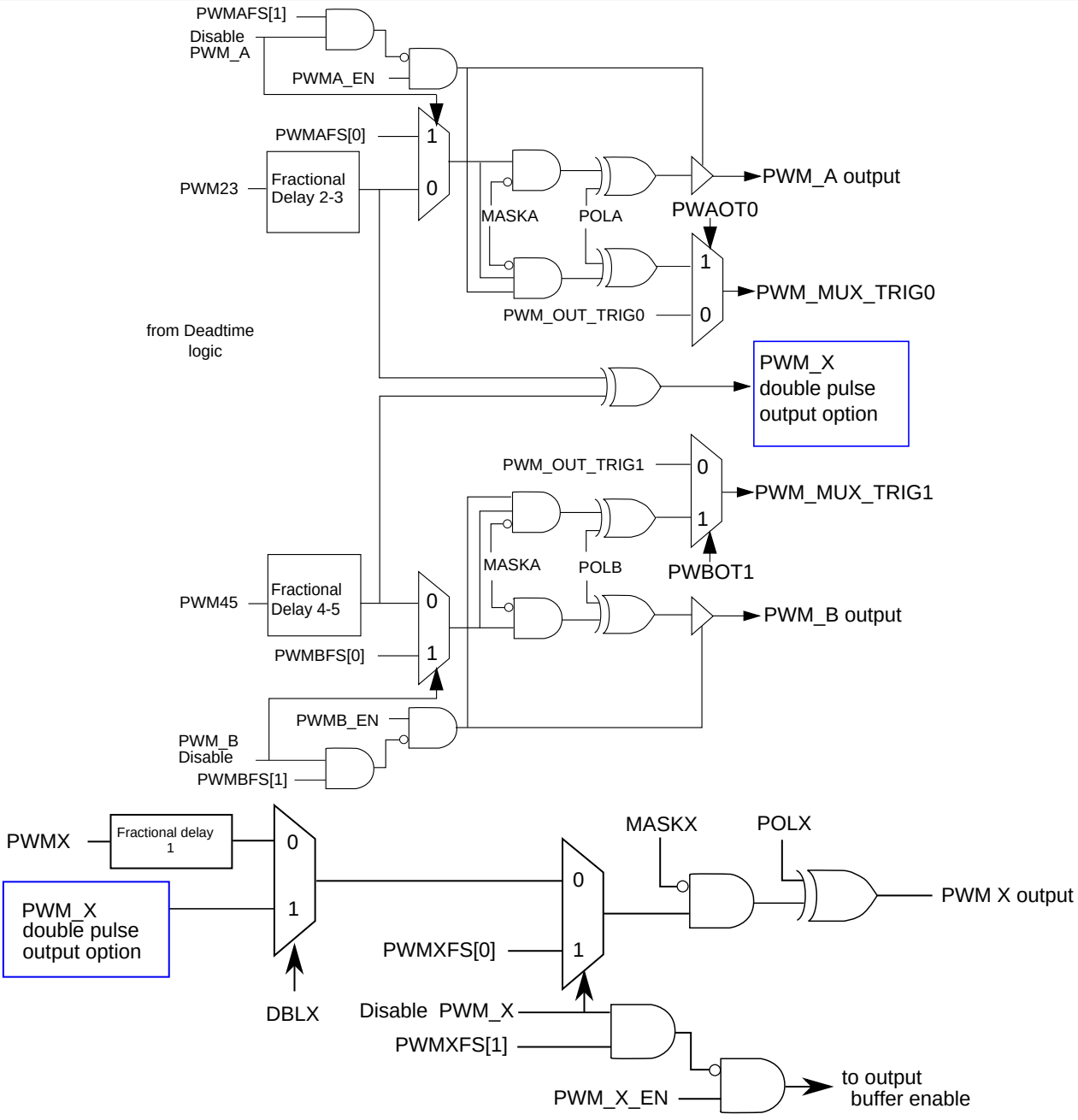

Figure 19. Output logic

3.2.8 E-capture logic

Figure 20 is a block diagram of the E-capture circuit. On entering the pin input, the signal is split into two paths as follows:

- One path goes straight to a mux input where software can pass the signal directly to the capture logic for processing.
- The other path connects the signal to an 8-bit counter, which counts both the rising and falling edges of the input signal.

The output of this counter is compared to an 8-bit value specified by the user (EDGCMPx). If the two values are equal, the comparator generates a pulse that resets the counter. To process this pulse by the capture logic, it is also supplied to the mux input where the software can select it. This feature allows the module to count a specified number of edge events and then perform a capture and interrupt.

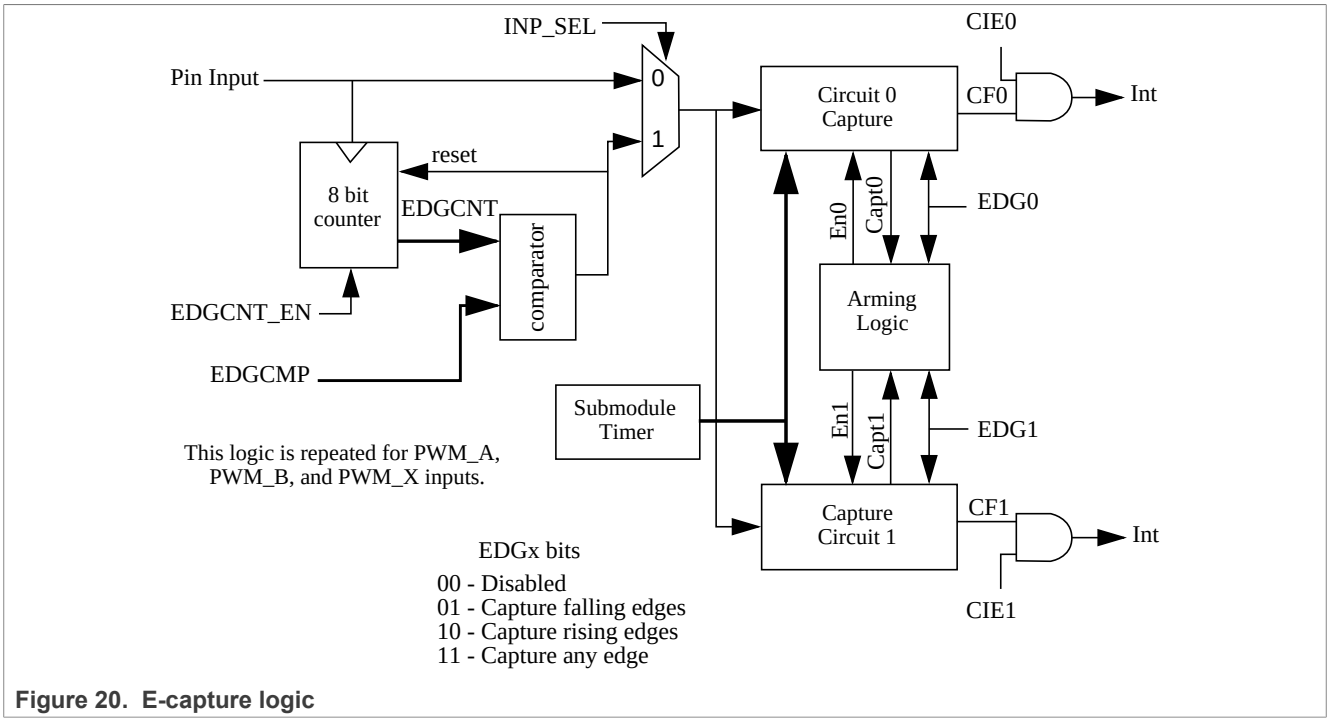

Figure 20. E-capture logic

Based on the mode selection, the mux selects either the pin input or the comparator output from the counter/comparator circuit processed by the capture logic. The selected signal is routed to two separate capture circuits, which work in tandem to capture the sequential edges of the signal. The CAPTCTRLx[EDGx1] and CAPTCTRLx[EDGx0], determine the type of edge captured by each circuit, whose functionality is listed in Figure 20. Also the arming logic controls the operation of the capture circuits, which allows captures to be performed in a free-running (continuous) or one-shot mode. In free-running mode, the capture sequences are performed indefinitely. If both capture circuits are enabled, they work together in a ping-pong style where a capture event from one circuit leads to the arming of the other and vice versa. In one-shot mode, only one capture sequence is performed. If both capture circuits are enabled, capture circuit 0 is first armed and when a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are also capable of generating an interrupt to the CPU.

3.2.9 Fault protection

Fault protection can control any combination of PWM output pins. A logic 1 generates faults on any of the FAULTx pins. This polarity can be changed via FCTRL[FLVL]. Each FAULTx pin can be mapped arbitrarily to any of the PWM outputs. When the fault protection hardware disables PWM outputs, the PWM generator continues to run. Only the output pins are forced to logic 0, logic 1, or high-impedance depending on the values of OCTRL[PWMxFS].

The fault decoder disables PWM pins selected by the fault logic and the disable mapping (DISMAPn) registers. Figure 21 shows an example of the fault disable logic. Each bank of bits in DISMAPn controls the mapping for a single PWM pin, see Table 1.

The fault protection is enabled even when the PWM module is not enabled. Therefore, a fault is latched in and must be cleared to prevent an interrupt when the PWM is enabled.

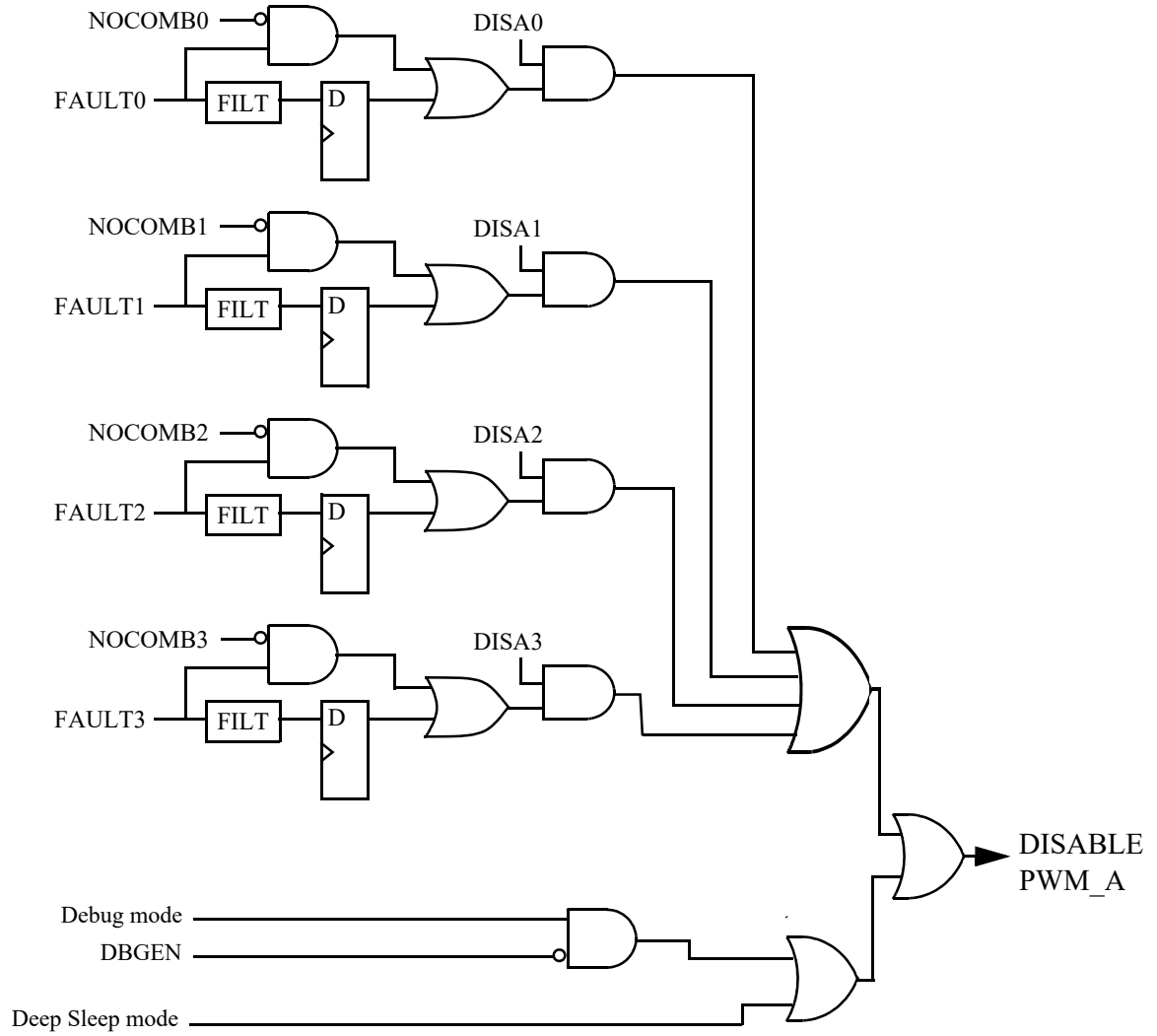

Figure 21. Fault decoder for PWM_A

Table 1. Fault mapping

PWM pin for a submodule	Controlling register bits
PWM_A	DISMAP0[DIS0A]
PWM_B	DISMAP0[DIS0B]
PWM_C	DISMAP0[DIS0X]

3.2.9.1 Fault pin filter

Each fault pin has a programmable filter that can be bypassed. The sampling period of the filter can be adjusted with FFILT[FILT_PER]. The number of consecutive samples that must agree before an input transition is recognized can be adjusted using FFILT[FILT_CNT]. Setting FFILT[FILT_PER] to 0 disables the input filter for a given FAULTx pin.

On detecting a logic 0 on the filtered FAULTx pin (or a logic 1 if FCTRL[FLVLx] is set), the corresponding FSTS[FFPINx], and FSTS[FFLAGx] bits are set. FSTS[FFPINx] remains set if the filtered FAULTx pin is 0. Clear FSTS[FFLAGx] by writing a logic 1 to FSTS[FFLAGx].

If the FIEx bit and FAULTx pin interrupt enable bit are set, then FSTS[FFLAGx] generates a CPU interrupt request. The interrupt request latch remains set until:

- Software clears FSTS[FFLAGx] by writing a logic 1 to the bit
- Software clears the FIEx bit by writing a logic 0 to it
- A reset occurs

Even with the filter enabled, a combinational path exists from the FAULTx inputs to the PWM pins. This path bypasses the filter when FCTRL20[NOCOMBx] = 0. This logic is also capable of holding a fault condition in the event of loss of clock to the PWM module.

3.2.9.2 Automatic fault clearing

Setting an automatic clearing mode bit, FCTRL[FAUTOx] configures faults from the FAULTx pin for automatic clearing.

Figure 22 shows the following:

- When FCTRL[FAUTOx] is set, disabled PWM pins are enabled when the FAULTx pin returns to logic 1 and a new PWM full or half cycle begins.
- If FSTS[FFULLx] is set and the fault condition on FAULTx disappears, then disabled PWM pins are enabled at the start of the next full cycle.
- If FSTS[FHALFx] is set, disabled PWM pins are enabled at the start of a half cycle. Clearing FSTS[FFLAGx] does not affect disabled PWM pins when FCTRL[FAUTOx] is set.

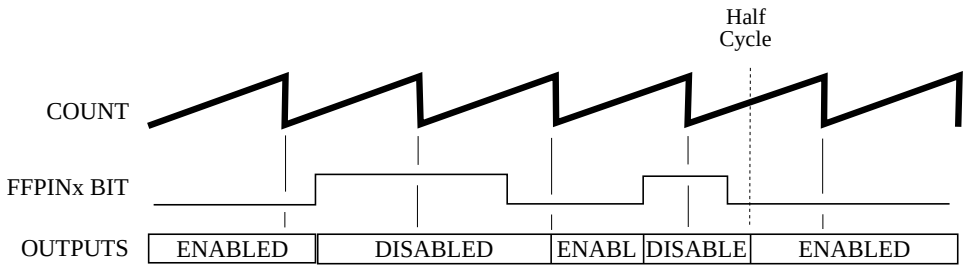

Figure 22. Automatic fault clearing

3.2.9.3 Manual fault clearing

Setting an automatic clearing mode bit, FCTRL[FAUTOx] configures faults from the FAULTx pin for automatic clearing.

Clearing the automatic clearing mode bit, FCTRL[FAUTOx] configures faults from the FAULTx pin for manual clearing:

- If the fault safety mode bits FCTRL[FSAFEx] are clear, then PWM pins disabled by the FAULTx pins are enabled when:
  - The software clears the corresponding FSTS[FFLAGx] flag.
  - The pins are enabled when the next PWM full or half cycle begins regardless of the logic level detected by the filter at the FAULTx pin. See Figure 23:
    - If FSTS[FFULLx] is set, disabled PWM pins are enabled at the start of a full cycle.
    - If FSTS[FHALFx] is set, disabled PWM pins are enabled at the start of a half cycle.

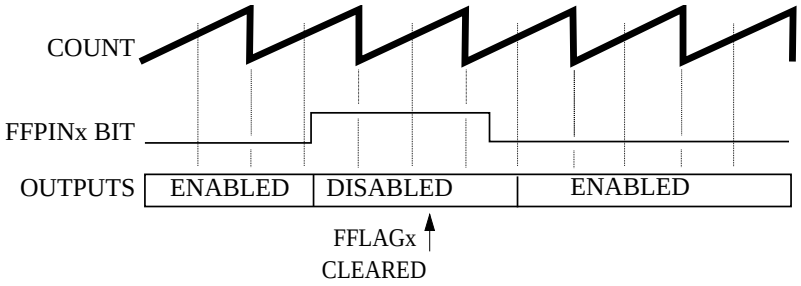

Figure 23. Manual fault clearing (FCTRL[FSAFEx] = 0, FSTS[FFULLx] = 1)

- If the fault safety mode bits FCTRL[FSAFEx] are set, PWM pins disabled by the FAULTx pins are enabled when:
  - The software clears the corresponding FSTS[FFLAGx] flag.
  - The filter detects a logic zero on the FAULTx pin at the start of the next PWM full or half cycle boundary. See Figure 24:
    - If FSTS[FFULLx] is set, disabled PWM pins are enabled at the start of a full cycle.
    - If FSTS[FHALFx] is set, disabled PWM pins are enabled at the start of a half cycle.

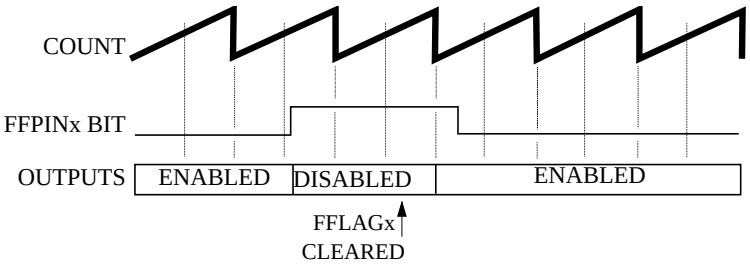

Figure 24. Manual fault clearing (FCTRL[FSAFEx] = 1, FSTS[FHALFx] = 1)

3.2.9.4 Fault testing

FTST[FTEST] is used to simulate a fault condition on each of the fault inputs within that fault channel.

4 Experiment on board

The example code for the MCX N series helps understand the implementation of the different PWM functions.

Note: *This application note describes only the experiment with MCX Nx4x. For board connection information on performing an experiment with MCX N23x, refer to the readme file in the software package.*

To test the PWM functions, the FRDM-MCXN947 board is used, as shown in Figure 25. The example code is based on the SDK example: `\SDK_2_14_0_FRDM-MCXN947\boards\frdmmcxn947\driver_examples\pwm`

The user must be familiar with the above example and the related hardware.

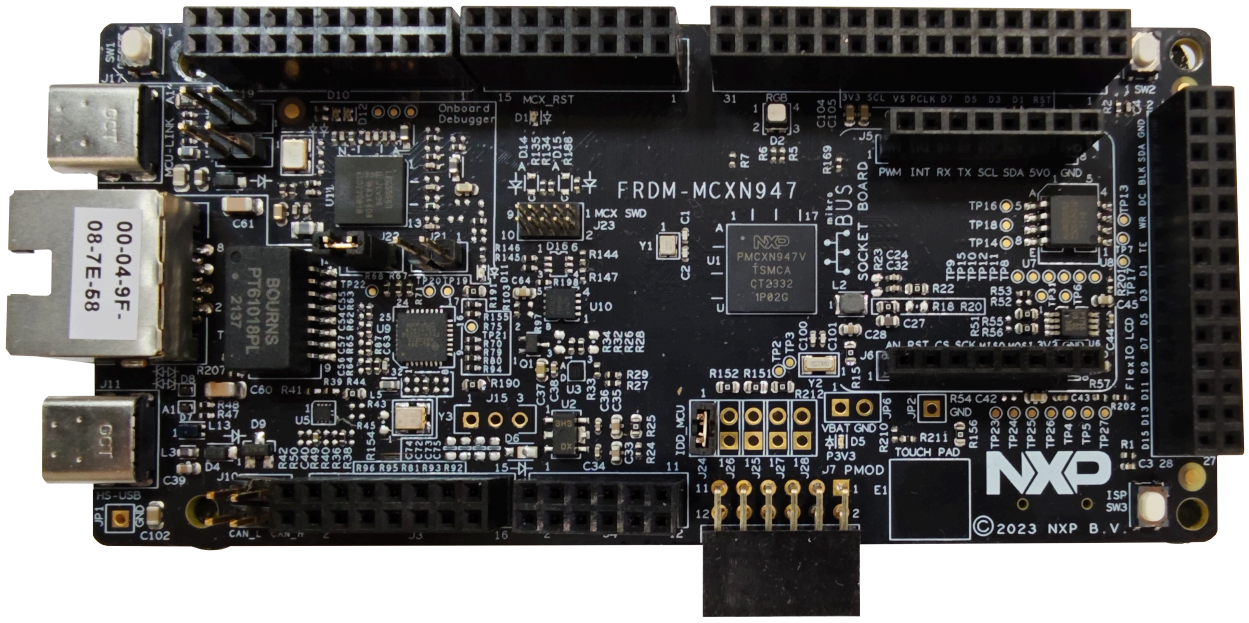

Figure 25. FRDM-MCXN947 board

4.1 Board setup

The FDRM-MCXN947 board ships with an onboard debugger. Use a USB-C cable to connect to the board via J17 for downloading and debugging.

To capture the output waveform, the user must probe the PWM signal using an oscilloscope or logic analyzer. As shown in Figure 26:

- PWM1_A0 (P2_6) is connected at J3-15.
- PWM1_B0 (P2_7) is connected at J3-13.
- TEST_PIN (P2_4) is connected at J3-11.
- ADC0_A2 (P4_23) is connected at J8-28.
  Note: *The P2_4 is used as a TEST_PIN to indicate the actions of ADC triggering and E-Capture.*

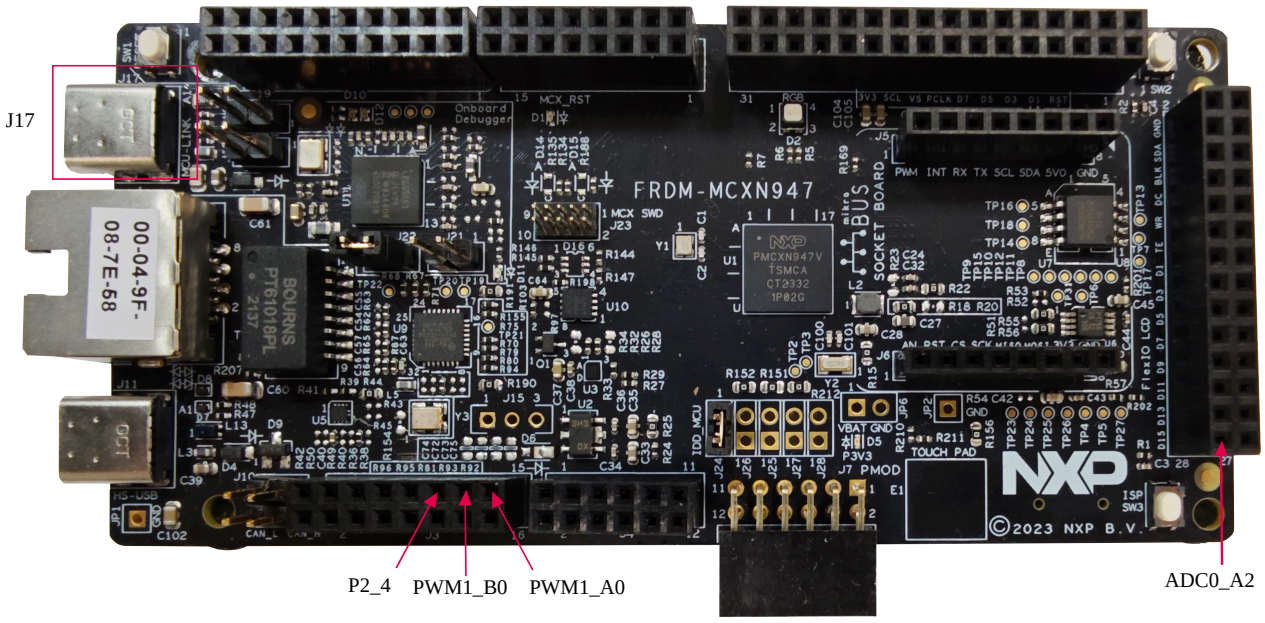

Figure 26. FRDM-MCXN947 board connection

4.2 FlexPWM function test

To implement the different functions of FlexPWM, several configurations are required in the code.

Note: *In the test, submodule 0 is used as an example, if other submodule is used, find the corresponding registers.*

Some of the main configurations are as follows:

- Initialize the clock, attach FRO 12M to FLEXCOMM4 for debug console.
- Initialize the pins that must be used, including P1_8 (FC4_P0), P1_9 (FC4_P1), P2_6 (PWM1_A0), P2_7 (PWM1_B0), P4_23 (ADC0_A2), and P2_4 (TEST_PIN).
- Initialize and enable the FlexPWM module:
  - Use the IP bus clock as the source clock for the PWM submodule.
  - Use the local reload signal to reload registers.
  - Use full cycle reload as the reload mode.
  - Set operation modes for PWM_A and PWM_B, including independent mode and complementary mode, which depend on the PWM operating mode.
  - Set PWM fault disable mapping for submodule 0 PWM_A and PWM_B.
  - Parameter settings are given as follows:
    - SM0INIT: Indicate the start of the PWM period
    - SM0VAL0: Indicate the center value
    - SM0VAL1: Indicate the end of the PWM period
    - SM0DTCNT0 and SM0DTCNT1: Set up the dead time value
  - Configure other related registers according to the PWM mode and enable PWM output.
- Initialize and enable the LPADC module:
  - Use the FRO HF as the source clock for the ADC module.
  - Use VREF_OUT driven from the VREF block as the reference voltage.
  - Do calibration before enable.
  - Set conversion CMD configuration.

– Set the trigger configuration; configure the trigger0 and trigger1 for ADC triggering.
– Enable the LPADC interrupt.

In the example code, the user can set different PWM modes using keyboard input.

1. Open a UART debug terminal and configure the settings as shown in Figure 27:
   - Baud rate: 115200
   - Data: 8 bits
   - Parity: None
   - Stop bits: 1

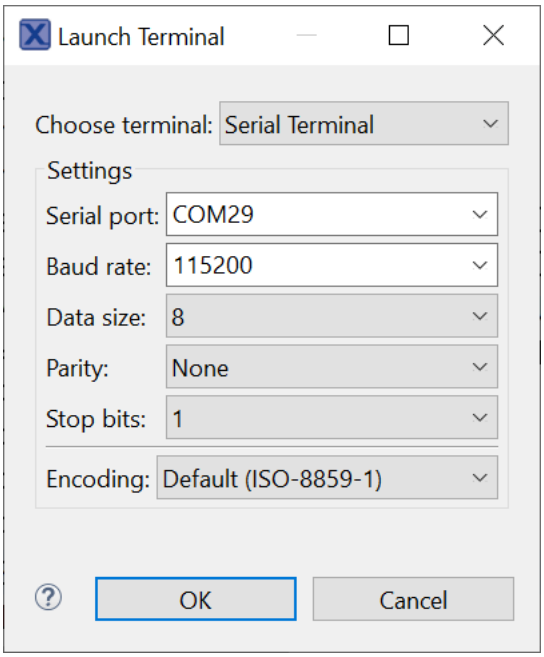

Figure 27. UART debug terminal configuration

2. Click the reset button. The terminal displays the prompt information as shown in Figure 28. To select the PWM mode, the user can input '0', '1' or '2', and so on, from the PC keyboard.

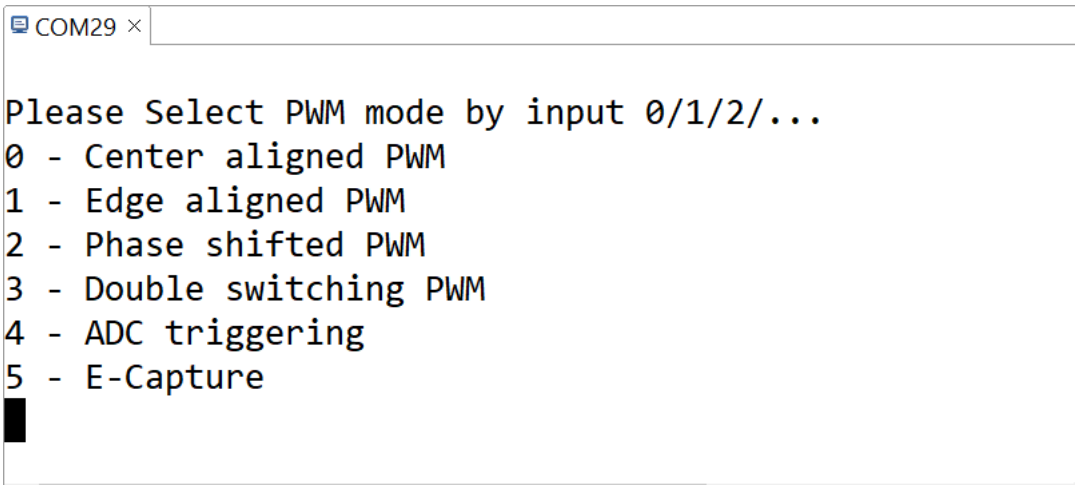

Figure 28. PWM mode selection menu

4.2.1 Center-aligned PWM

In the center-aligned PWM mode:

- The PWM_A and PWM_B run in independent mode.
- SM0VAL2/SM0VAL3 (SM0VAL4/SM0VAL5) respectively define the rising edge and falling edge of PWM_A (PWM_B).
- PWM_MODULO represents the total number of counts in one cycle of PWM. It is equal to the PWM source clock frequency (150 MHz in this example) divided by the PWM frequency (10 kHz in this example).
- Set 0 as the center of symmetry. The values of SM0VAL2/SM0VAL3 (SM0VAL4/SM0VAL5) are symmetric about 0.
  The core code for center aligned PWM is as follows:

```
pwmHighPulse[0] = PWM_MODULO * dutyCyclePercent[0] / 100UL;    // PWM counter value for PWM_A
pwmHighPulse[1] = PWM_MODULO * dutyCyclePercent[1] / 100UL;    // PWM counter value for PWM_B

/* Center aligned PWM */
if(g_pwmMode == 0)
{
    BOARD_PWM_BASEADDR -> SM[0].VAL2 = PWM_VAL2_VAL2((uint16_t)(-(pwmHighPulse[0] / 2)));
    BOARD_PWM_BASEADDR -> SM[0].VAL3 = PWM_VAL3_VAL3((uint16_t)( (pwmHighPulse[0] / 2)));
    BOARD_PWM_BASEADDR -> SM[0].VAL4 = PWM_VAL4_VAL4((uint16_t)(-(pwmHighPulse[1] / 2)));
    BOARD_PWM_BASEADDR -> SM[0].VAL5 = PWM_VAL5_VAL5((uint16_t)( (pwmHighPulse[1] / 2)));
}
```

- PWM_A and PWM_B work in the center-aligned PWM mode with 60 % and 40 % duty cycles, respectively, as shown in Figure 29.

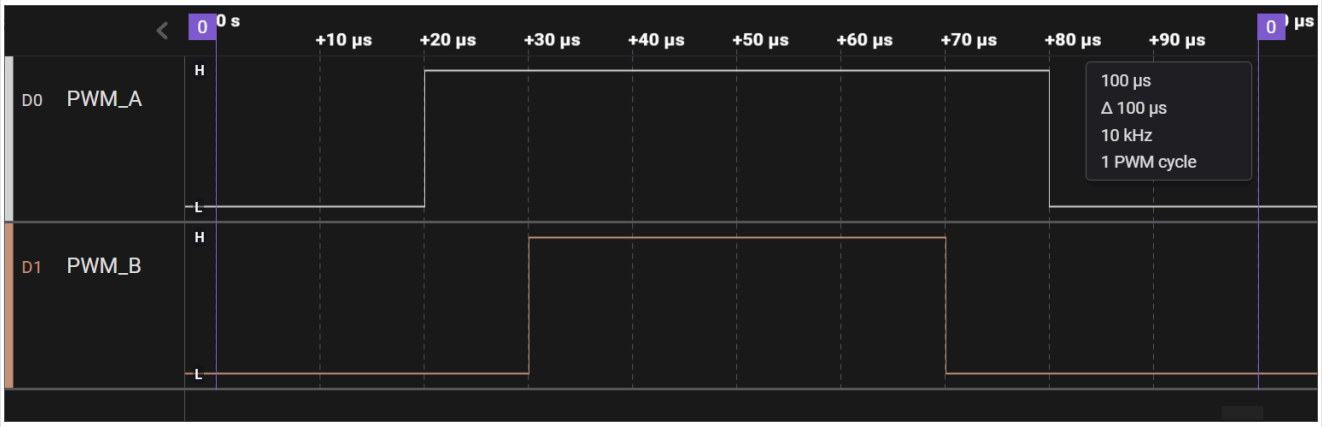

Figure 29. Center-aligned PWM waveform

4.2.2 Edge-aligned PWM

In the edge-aligned PWM mode:

- The PWM_A and PWM_B run in independent mode.
- SM0VAL2/SM0VAL3 (SM0VAL4/SM0VAL5) respectively define the rising edge and falling edge of PWM_A (PWM_B).
- Set both SM0VAL2 and SM0VAL4 as the initial counting value.
- PWM_A and PWM_B generate rising edges at the same time, and achieve output with different duty cycles by setting SM0VAL3 and SM0VAL5.
  The core code for edge-aligned PWM is as follows:

```
pwmHighPulse[0] = PWM_MODULO * dutyCyclePercent[0] / 100UL;    // PWM counter value for PWM_A
pwmHighPulse[1] = PWM_MODULO * dutyCyclePercent[1] / 100UL;    // PWM counter value for PWM_B

/* Edge aligned PWM */
else if(g_pwmMode == 1)
{
    BOARD_PWM_BASEADDR -> SM[0].VAL2 = PWM_VAL2_VAL2((uint16_t)(-(PWM_MODULO / 2)));
```

```
    BOARD_PWM_BASEADDR -> SM[0].VAL3 = PWM_VAL3_VAL3((uint16_t)(-(PWM_MODULO / 2) + pwmHighPulse[0]));
    BOARD_PWM_BASEADDR -> SM[0].VAL4 = PWM_VAL4_VAL4((uint16_t)(-(PWM_MODULO / 2)));
    BOARD_PWM_BASEADDR -> SM[0].VAL5 = PWM_VAL5_VAL5((uint16_t)(-(PWM_MODULO / 2) + pwmHighPulse[1]));
}
```

- PWM_A and PWM_B work in the edge-aligned PWM mode with 60 % and 40 % duty cycles, respectively, as shown in Figure 30.

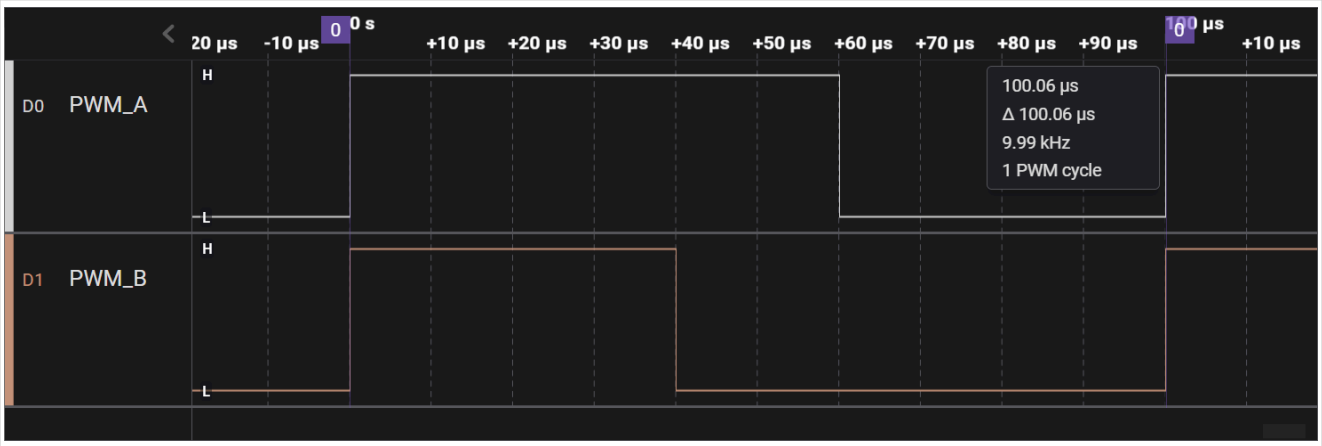

Figure 30. Edge-aligned PWM waveform

4.2.3 Phase-shifted PWM

In the phase-shifted PWM mode:

- The PWM_A and PWM_B run in independent mode.
- SM0VAL2/SM0VAL3 (SM0VAL4/SM0VAL5) respectively define the rising edge and falling edge of PWM_A (PWM_B).
- Set the phase and duty cycle information of PWM_A through SM0VAL2 and SM0VAL3.
- Then, use SM0VAL4 and SM0VAL5 to set the phase shift of PWM_B relative to PWM_A.
  The core code for phase-shifted PWM is as follows:

```
pwmHighPulse[0] = PWM_MODULO * dutyCyclePercent[0] / 100UL;    // PWM counter value for PWM_A
phaseA = PWMA_PHASE * PWM_MODULO / 360UL;                      // Phase for PWM_A
phaseShift = PWM_PHASESHIFT * PWM_MODULO / 360UL;              // Phase shift for PWM_B
/* Phase shifted PWM */
else if(g_pwmMode == 2)
{
    BOARD_PWM_BASEADDR -> SM[0].VAL2 = PWM_VAL2_VAL2((uint16_t)(-(PWM_MODULO / 2) + phaseA));
    BOARD_PWM_BASEADDR -> SM[0].VAL3 = PWM_VAL3_VAL3((uint16_t)(-(PWM_MODULO / 2) + phaseA + pwmHighPulse[0]));
    BOARD_PWM_BASEADDR -> SM[0].VAL4 = PWM_VAL4_VAL4((uint16_t)(-(PWM_MODULO / 2) + phaseA + phaseShift));
    BOARD_PWM_BASEADDR -> SM[0].VAL5 = PWM_VAL5_VAL5((uint16_t)(-(PWM_MODULO / 2) + phaseA + phaseShift +
 pwmHighPulse[0]));
}
```

- PWM_A and PWM_B work in the phase-shifted PWM mode:
  - The phase of PWM_A is 36 degrees (1/10 PWM cycle).
  - The phase shift of PWM_B relative to PWM_A is 36 degrees (1/10 PWM cycle), as shown in Figure 31.

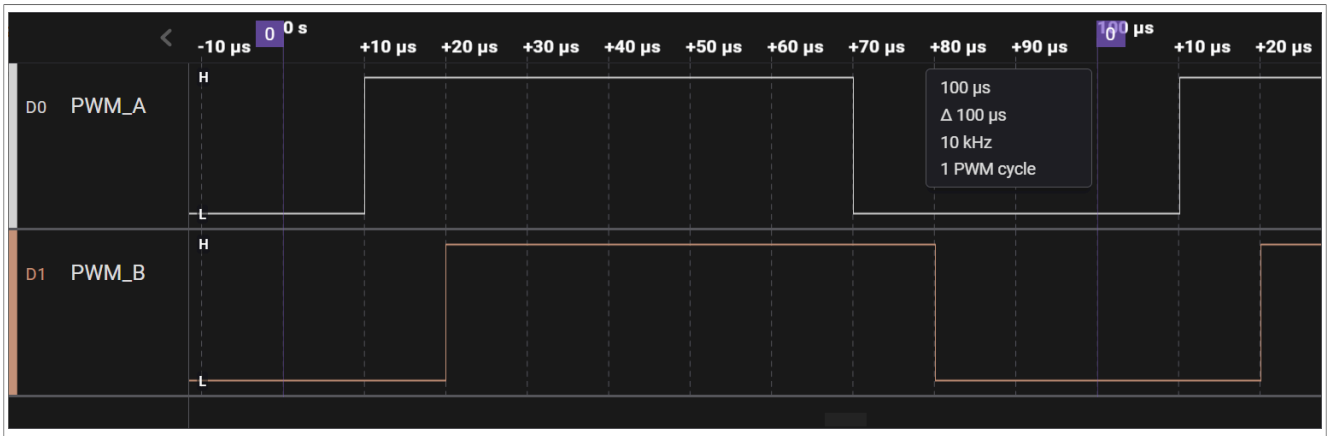

Figure 31. Phase-shifted PWM waveform

4.2.4 Double-switching PWM

In the double-switching PWM mode:

- The PWM_A and PWM_B run in independent mode.
- SM0VAL2/SM0VAL3 (SM0VAL4/SM0VAL5) respectively define the rising edge and falling edge of PWM_A (PWM_B).
- The user can flexibly set the rising and falling edges of PWM_A and PWM_B.
- Use SM0VAL2 and SM0VAL4 to set the rising edges of PWM_A and PWM_B.
- Use SM0VAL3 and SM0VAL5 to set the falling edges of PWM_A and PWM_B respectively. Figure 32 shows the PWM_A source and PWM_B source.
- When SM0CTRL[DBLEN] = 1 and SM0CTRL[SPLIT] = 0, the PWM_A and PWM_B output is PWM_A source XOR PWM_B source.
- When SM0CTRL[DBLEN] = 1 and SM0CTRL[SPLIT] = 1, the PWM_A output is the pulse that occurs when the PWM_A source is 1 and the PWM_B source is 0. The PWM_B output is the pulse that occurs when the PWM_B source is 1 and the PWM_A source is 0.
  The core code for double-switching PWM is as follows:

```
risingA  = PWMA_RISING  * PWM_MODULO / 100UL;        // Rising  value for PWM_A
fallingA = PWMA_FALLING * PWM_MODULO / 100UL;       // Falling value for PWM_A
risingB  = PWMB_RISING  * PWM_MODULO / 100UL;      // Rising  value for PWM_B
fallingB = PWMB_FALLING * PWM_MODULO / 100UL;      // Falling value for PWM_B

/* Double switching PWM */
else if(g_pwmMode == 3)
{
    BOARD_PWM_BASEADDR -> SM[0].VAL2 = PWM_VAL2_VAL2((uint16_t)(-(PWM_MODULO / 2) + risingA));
    BOARD_PWM_BASEADDR -> SM[0].VAL3 = PWM_VAL3_VAL3((uint16_t)(-(PWM_MODULO / 2) + fallingA));
    BOARD_PWM_BASEADDR -> SM[0].VAL4 = PWM_VAL4_VAL4((uint16_t)(-(PWM_MODULO / 2) + risingB));
    BOARD_PWM_BASEADDR -> SM[0].VAL5 = PWM_VAL5_VAL5((uint16_t)(-(PWM_MODULO / 2) + fallingB));
    if(g_dblen_bit == 1)
    {
        BOARD_PWM_BASEADDR -> SM[0].CTRL |= PWM_CTRL_DBLEN_MASK;
    }
    if(g_split_bit == 1)
    {
        BOARD_PWM_BASEADDR -> SM[0].CTRL |= PWM_CTRL_SPLIT_MASK;
    }
}
```

- The rising and falling edges of PWM_A are set to 20 and 60 respectively (range from 0 to 100).
- The rising and falling edges of PWM_B are set to 40 and 80 respectively (range from 0 to 100), as shown in Figure 32.

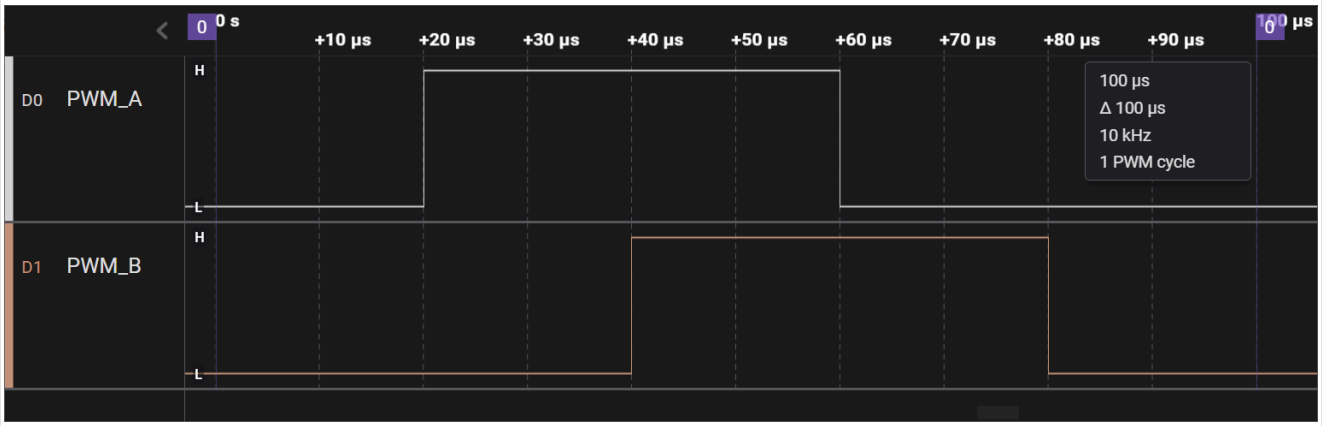

Figure 32. Double-switching PWM waveform (DBLEN = 0, SPLIT = 0)

- For SM0CTRL[DBLEN] = 1, the double switching PWM behavior is enabled, as shown in Figure 33.

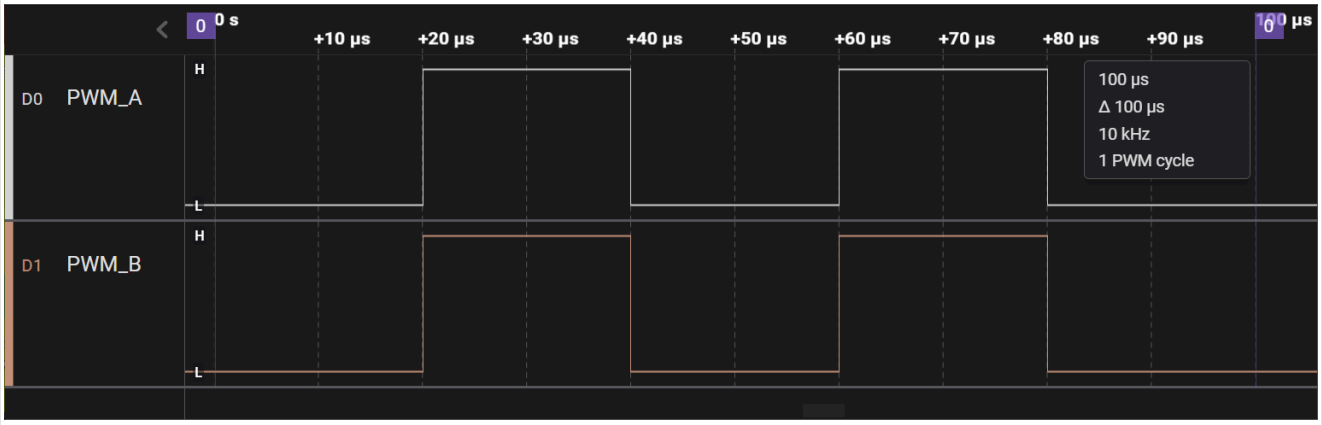

Figure 33. Double-switching PWM waveform (DBLEN = 1, SPLIT = 0)

- Figure 34 represents the outputs of PWM_A and PWM_B under the conditions of SM0CTRL[SPLIT] = 0 and SM0CTRL[SPLIT] = 1 respectively.

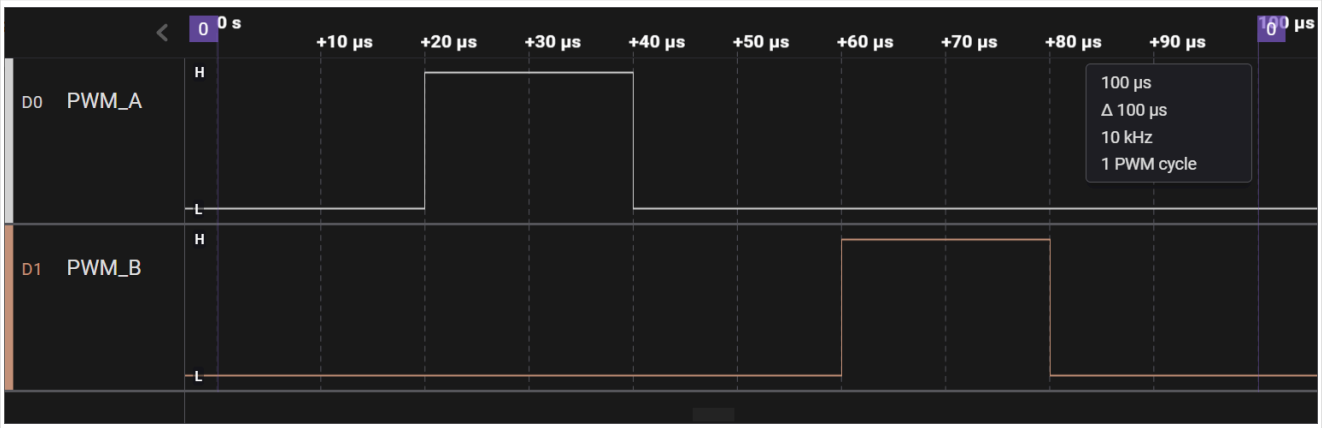

Figure 34. Double-switching PWM waveform (DBLEN = 1, SPLIT = 1)

4.2.5 ADC triggering

In the ADC triggering mode:

- The PWM_A and PWM_B run in complementary mode.

- SM0VAL2 and SM0VAL3 respectively define the rising edge and falling edge of PWM_A.
- PWM_B is complementary to PWM_A, therefore, SM0VAL4 and SM0VAL5 are free to be configured as ADC triggers.
- To enable the corresponding triggers (PWM_OUT_TRIG0 and PWM_OUT_TRIG1) on SM0VAL4 and SM0VAL5, set bit 4 and bit 5 of SM0TCTRL[OUT_TRIG_EN].
- Configure the ADC0 trigger0 and trigger1.
- Link PWM1_SM0_MUX_TRIG0 to ADC0_TRIG0 and PWM1_SM0_MUX_TRIG1 to ADC0_TRIG1.
- After trigger0 and trigger1 interrupt are enabled, the ADC triggering function is ready. When the counter value matches the SM0VAL4 and SM0VAL5 values, ADC0 trigger0 and ADC0 trigger1 interrupts are generated.
- In complementary mode, dead time operation is supported. SM0DTCNT0[DTCNT0] and SM0DTCNT1[DTCNT1] are used to insert dead time at the rising edge of PWM_A and the falling edge of PWM_B, respectively.
- P2_4 is used as TEST_PIN to show LPADC interrupt.
  The core code for ADC triggering is as follows:

```
pwmHighPulse[0] = PWM_MODULO * dutyCyclePercent[0] / 100UL;   // PWM counter value for PWM_A
pwmTrigger[0] = PWM_TRIG0 * PWM_MODULO / 100UL;                  // Trigger0 value set
pwmTrigger[1] = PWM_TRIG1 * PWM_MODULO / 100UL;                  // Trigger1 value set
deadTimeVal = PWM_DEADTIME * (PWM_SRC_CLK_FREQ / 1000000UL) / 1000UL; // Deadtime value set

/* ADC triggering */
else if(g_pwmMode == 4)
{
    BOARD_PWM_BASEADDR -> SM[0].VAL2 = PWM_VAL2_VAL2((uint16_t)(-(pwmHighPulse[0] / 2)));
    BOARD_PWM_BASEADDR -> SM[0].VAL3 = PWM_VAL3_VAL3((uint16_t)( (pwmHighPulse[0] / 2)));
    BOARD_PWM_BASEADDR -> SM[0].VAL4 = PWM_VAL4_VAL4((uint16_t)(-(PWM_MODULO / 2) + pwmTrigger[0]));;
    BOARD_PWM_BASEADDR -> SM[0].VAL5 = PWM_VAL5_VAL5((uint16_t)(-(PWM_MODULO / 2) + pwmTrigger[1]));
    BOARD_PWM_BASEADDR -> SM[0].DTCNT0 = deadTimeVal;
    BOARD_PWM_BASEADDR -> SM[0].DTCNT1 = deadTimeVal;
    /* PWM_OUT_TRIG0 will set when the counter value matches the VAL4 value */
    BOARD_PWM_BASEADDR -> SM[0].TCTRL |= PWM_TCTRL_OUT_TRIG_EN(1 << 4);
    /* PWM_OUT_TRIG1 will set when the counter value matches the VAL5 value */
    BOARD_PWM_BASEADDR -> SM[0].TCTRL |= PWM_TCTRL_OUT_TRIG_EN(1 << 5);
}

void DEMO_LPADC_IRQ_HANDLER_FUNC(void)
{
   GPIO_PortToggle(GPIO2, 1U << 4U);              // Indicate entry interrupt
   g_LpadcInterruptCounter++;
#if (defined(FSL_FEATURE_LPADC_FIFO_COUNT) && (FSL_FEATURE_LPADC_FIFO_COUNT == 2U))
   if (LPADC_GetConvResult(DEMO_LPADC_BASE, &g_LpadcResultConfigStruct, 0U))
#else
   if (LPADC_GetConvResult(DEMO_LPADC_BASE, &g_LpadcResultConfigStruct))
#endif   /* FSL_FEATURE_LPADC_FIFO_COUNT */
   {
         g_LpadcConversionCompletedFlag = true;
   }

   DEMO_LPADC_BASE -> STAT |= ADC_STAT_TCOMP_INT_MASK;  // Clear trigger interrupt flag

   SDK_ISR_EXIT_BARRIER;
}
```

- PWM_A and PWM_B work in the complementary mode with the 60 % duty cycle. PWM_A and PWM_B implement ADC triggering at 40 and 60 (range from 0 to 100), as shown in Figure 35.

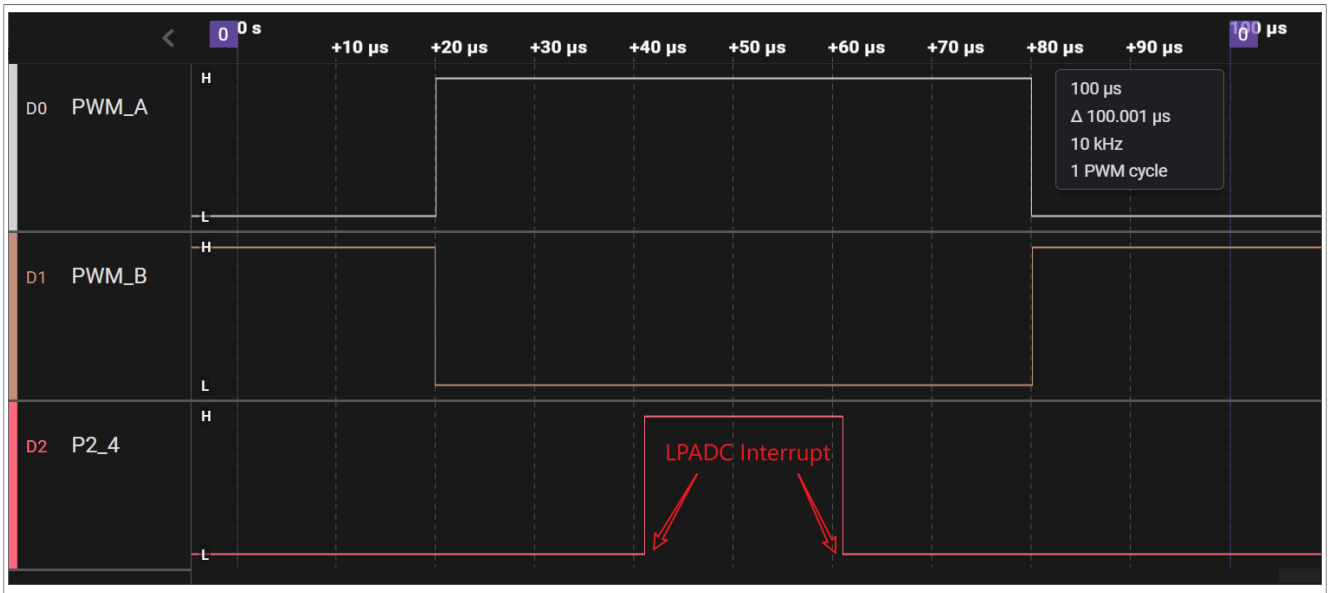

Figure 35. ADC triggering waveform

- Figure 36 shows the result of the ADC sampling, which approaches 4095 when the input is 3.3 V.

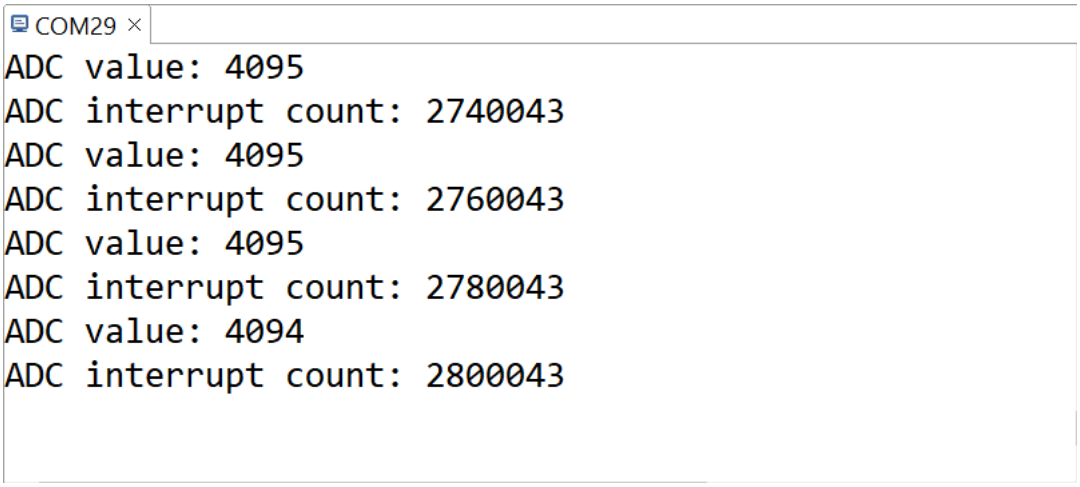

Figure 36. ADC sampling value

4.2.6 E-capture

In the E-capture mode:

- PWM_A normally acts as the PWM output, while PWM_B acts as an input for E-capture.
- SM0VAL2 and SM0VAL3 respectively define the rising edge and falling edge of PWM_A.
- For different applications, the user can select different sources as input for E-Capture.
- When SM0CAPTCTRLB[INP_SELB] = 0, the raw PWM_B input signal is selected as the source.
- When SM0CAPTCTRLB[INP_SELB] = 1, the output of the edge counter/compare is selected as the source.
- In addition, the user can set the type of edge to capture by setting the SM0CAPTCTRLB[EDGB0]/[EDGB1] when SM0CAPTCTRLB[INP_SELB] = 0.
- The user can set the edge counter value by setting the SM0CAPTCOMPB[EDGCMPB] when SM0CAPTCTRLB[INP_SELB] = 1.

- If the user want to use one-shot mode, enable it through SM0CAPTCTRLB[ONESHOTB]. P2_4 is used as TEST_PIN to show the capture action.
  The core code for E-capture is as follows:

```
pwmHighPulse[0] = PWM_MODULO * dutyCyclePercent[0] / 100U;    // PWM counter value for PWM_A

/* E-Capture */
else if(g_pwmMode == 5)
{
    BOARD_PWM_BASEADDR -> SM[0].VAL2 = PWM_VAL2_VAL2((uint16_t)(-(PWM_MODULO / 2)));
    BOARD_PWM_BASEADDR -> SM[0].VAL3 = PWM_VAL3_VAL3((uint16_t)(-(PWM_MODULO / 2) + pwmHighPulse[0]));

    /* 0: Raw PWM_B input signal selected as source, 1: Set the output of the edge counter/compare circuitry as
 the source for the input capture */
    BOARD_PWM_BASEADDR -> SM[0].CAPTCTRLB |= PWM_CAPTCTRLB_INP_SELB(0U);
    /* Capture 0 capture rising edges, only works when INP_SELB = 0 */
    BOARD_PWM_BASEADDR -> SM[0].CAPTCTRLB |= PWM_CAPTCTRLB_EDGB0(2U);
    /* Capture 1 capture falling edges, only works when INP_SELB = 0 */
    BOARD_PWM_BASEADDR -> SM[0].CAPTCTRLB |= PWM_CAPTCTRLB_EDGB1(1U);
    /* Edge Counter B Enabled */
    BOARD_PWM_BASEADDR -> SM[0].CAPTCTRLB |= PWM_CAPTCTRLB_EDGCNTB_EN(1U);
    /* Enabled one shot mode */
    //  BOARD_PWM_BASEADDR -> SM[0].CAPTCTRLB |= PWM_CAPTCTRLB_ONESHOTB(1U);
    /* Input capture operation as specified by CAPTCTRLB[EDGBx] is enabled */
    BOARD_PWM_BASEADDR -> SM[0].CAPTCTRLB |= PWM_CAPTCTRLB_ARMB(1U);
    /* Set the compare value associated with the edge counter for the PWM_B input capture, works when INP_SELB =
 1 */
    BOARD_PWM_BASEADDR -> SM[0].CAPTCOMPB  = PWM_CAPTCOMPB_EDGCMPB(5U);

    BOARD_PWM_BASEADDR -> SM[0].INTEN     |= PWM_INTEN_CB0IE(1U);     // Counter B0 interrupt Enabled
    BOARD_PWM_BASEADDR -> SM[0].INTEN     |= PWM_INTEN_CB1IE(1U);     // Counter B1 interrupt Enabled

    EnableIRQ(FLEXPWM1_SUBMODULE0_IRQn);
}

void FLEXPWM1_SUBMODULE0_IRQHandler(void)
{
    GPIO_PortToggle(GPIO2, 1U << 4U);                                  // Indicate entry interrupt
    BOARD_PWM_BASEADDR -> SM[0].STS |= PWM_STS_CFB0(1U);               // Clear Capture B0 interrupt flag
    BOARD_PWM_BASEADDR -> SM[0].STS |= PWM_STS_CFB1(1U);               // Clear Capture B1 interrupt flag
    //  BOARD_PWM_BASEADDR -> SM[0].CAPTCTRLB |= PWM_CAPTCTRLB_ARMB(1U);  // Re-arm Capture circuit

    SDK_ISR_EXIT_BARRIER;
}
```

- In this mode, PWM_B is set as input and samples the signal of PWM_A at 60 % duty cycle.
- For SM0CAPTCTRLB[INP_SELB] = 0, capture 0 is set to capture the rising edge and capture 1 is set to capture the falling edge, as shown in Figure 37.

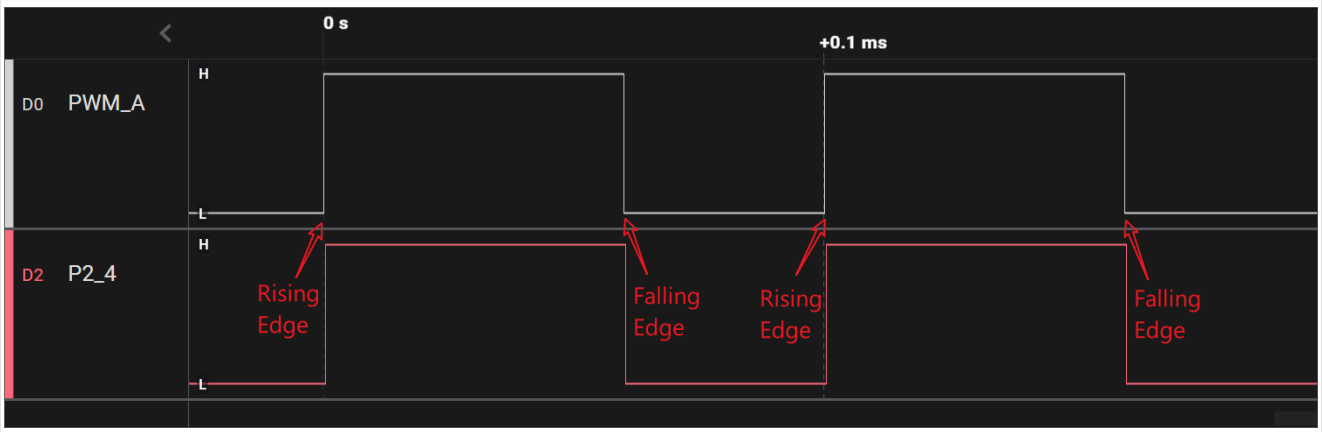

Figure 37. E-Capture waveform (INP_SELB = 0)

- For SM0CAPTCTRLB[INP_SELB] = 1, the compare value of the edge counter is set to 5, and the interrupt is triggered every 5 edges, as shown in Figure 38.

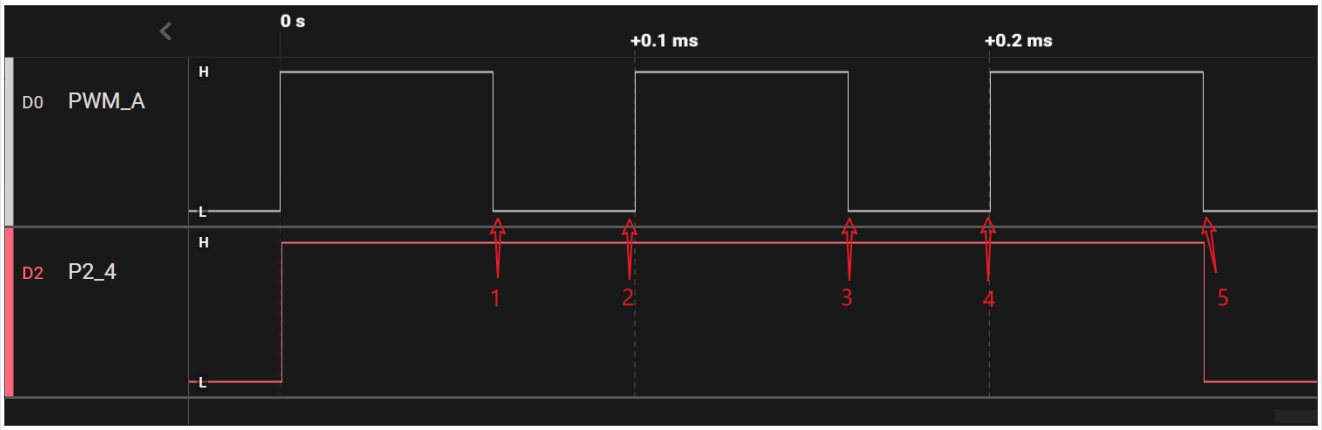

Figure 38. E-Capture waveform (INP_SELB = 1)

5 Design considerations

To ensure correct operation of the FlexPWM module on the MCX N series, use the following list of considerations:

- The PWM output pins are required to connect a strong external pulldown or pullup resistor (1 kΩ to 10 kΩ) close to the pin. It is done to ensure safety status under uncertain conditions.
- To disable PWM outputs manually regardless of duty cycle and clock settings, the following are the two methods:
  - Disable PWM output by clearing the corresponding bit of PWM_OUTEN, which results in a tri-state output on the PWM pin.
  - Disable PWM output by setting the corresponding bit of PWM_MASK. Follow it by a FORCE_OUT event, which results in logic zero output prior to output polarity on the PWM pin.
- Ensure to meet these boundary conditions for correct PWM generation:
  INIT ≤ VAL2 (VAL4), VAL3 (VAL5) ≤ VAL1
- When variable frequency PWMs are required, keep INIT constant and change frequency only through changing VAL1.

6 Conclusion

This application note summarizes the structure and functions of the FlexPWM module on the MCX N series. It provides example code to understand the implementation of the functions better, making it easy for users to use this module properly in their projects. Finally, some design considerations are shared with the users to help them use the module effectively.

7 Reference

The references used to supplement this document are as follows:

- *Using eFlexPWM with MC56F82xx DSC* (document AN4485)
- *MCX Nx4x Reference Manual* (document MCXNX4XRM)

8 Note about the source code in the document

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

9 Revision history

Table 2 summarizes the revisions to this document.

Table 2. Revision history

Document ID	Release date	Description
AN14196 v.1.0	02 May 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

For more information, please visit: https://www.nxp.com

Date of release: 2 May 2024
Document identifier: AN14196