SMART ARM-based Microcontrollers

AT10842: Using the Timer Counter for Control Applications in SAM L22

APPLICATION NOTE

Introduction

This application note describes the following features of the Timer/Counter for Control Applications available on the Atmel® | SMART SAM L22.

1. Circular Buffer
2. One-shot Operation
3. Output Matrix with DTI
4. Swap
5. Pattern Generation
6. Ramp2
7. Ramp2A
8. Dual Slope PWM
9. Counter Operation
10. Fault Operation
11. Dithering
12. Capture Operation

It provides details for configuring the above features of the Timer/Counter for Control Applications. It also contains code examples to simplify the use of TCC in typical applications.

All the software examples specified in this document are provided in ASF (Atmel® Software Framework).

For more information about the features of TCC module, refer SAM L22 device datasheet.

Features

- Up to four compare/capture channels (CC) with
  - Double buffered period setting
  - Double buffered compare or capture channel
  - Circular buffer on period and compare channel registers
- Waveform generation
  - Frequency generation

  - Single-slope pulse-width modulation (PWM)
  - Dual-slope pulse-width modulation with half-cycle reload capability
- Input capture
  - Event capture
  - Frequency capture
  - Pulse-width capture
- Waveform extensions
  - Configurable distribution of compare channels outputs across port pins
  - Low- and high-side output with programmable dead-time insertion
  - Waveform swap option with double buffer support
  - Pattern generation with double buffer support
  - Dithering support
- Fault protection for safe drivers disabling
  - Two recoverable fault sources
  - Two non-recoverable fault sources
  - Debugger can be source of non-recoverable fault
- Input event
  - Two input events for counter
  - One input event for each compare channel
- Output event
  - Three output events (Count, Retrigger, and Overflow) available for counter
  - One Compare Match/Input Capture output event for each channel
- Interrupts
  - Overflow and Retrigger interrupt
  - Compare Match/Input Capture interrupt
  - Interrupt on fault detection

Table of Contents

1. Abbreviations

ASF	Atmel Software Framework
CC	Compare/capture
DTI	Dead-time Insertion
EDGB	Embedded Debugger
EVSYS	Event System
GCLK	Generic clock
IDE	Integrated Development Environment
OTMX	Output Matrix
PER	Period
SMPS	Switching Mode Power Supply
TCC	Timer/Counter for Control Applications

2. Pre-requisites

The solutions discussed in this document require basic familiarity with the following tools.

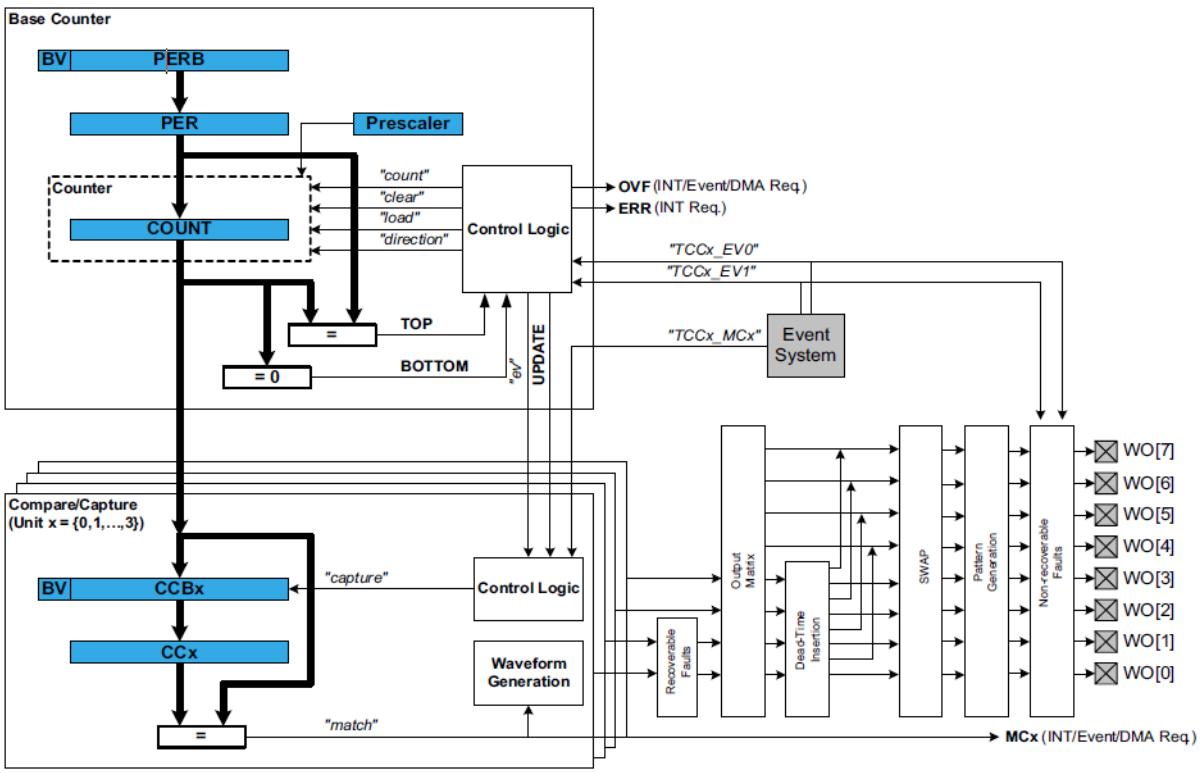
- Atmel Studio 7 or above
- SAM L22 Xplained Pro

3. TCC

3.1. TCC Overview

The Timer/Counter for Control Applications (TCC) module provides a set of timing and counting related functionalities, such as the generation of periodic waveforms, the capture of a periodic waveform's frequency/duty cycle, software timekeeping for periodic operations, waveform extension control, fault detection etc. It enables low- and high-side output with optional dead-time insertion. It can also generate a synchronized bit pattern across the waveform output pins. The fault options enable fault protection for safe and deterministic handling, disabling, and/or shut down of external drivers. Waveform extensions are intended for use in different types of motor control, ballast, LED, H-bridge, power converter, and other types of power control applications. The counter size of the TCC module is 24-bit.

Figure 3-1. Timer/Counter Block Diagram



3.2. Functional Description

The TCC module consists of following sections:

- Base Counter
- Compare/Capture channels, with waveform generation
- Waveform extension control and fault detection
- Interface to the event system, DMAC, and the interrupt system

The base counter can be configured to either count a pre-scaled generic clock or events from the event system (TCCx, with event action configured to counting). The counter value can be used by compare/capture channels which can be set up either in compare mode or capture mode. In capture mode, the

counter value is stored when a configurable event occurs. This mode can be used to generate timestamps used in event capture, or it can be used for the measurement of a periodic input signal's frequency/duty cycle.

In compare mode, the counter value is compared against one or more of the configured channels' compare values. When the counter value coincides with a compare value, an action can be taken automatically by the module, such as generating an output event or toggling a pin when used for frequency or PWM signal generation.

The connection of events between modules requires the use of the SAM Event System Driver (EVSYS) to route output event of one module to the input event of another. For more information on event routing, refer to the event driver (EVSYS) documentation.

In compare mode, when output signal is generated, extended waveform controls are available, to arrange the compare outputs into specific formats. The output matrix can change the channel output routing; Pattern generation unit can overwrite the output signal lines to specific states. The fault protection feature of the TCC supports recoverable and non-recoverable faults.

3.3. Special Considerations

As the TCC module have more waveform output pins than the number of compare/capture channels, the free pins (with number higher than number of channels) will reuse the waveform generated by channels subsequently. E.g., if the number of channels is four and number of wave output pins is eight, channel 0 outputs will be available on out pin 0 and 4, channel 1 output on wave out pin 1 and 5, and so on.

4. Setup

The example code provided in this application note uses the SAM L22 Xplained Pro kit and as the hardware and Atmel Studio 7 as IDE for application development.

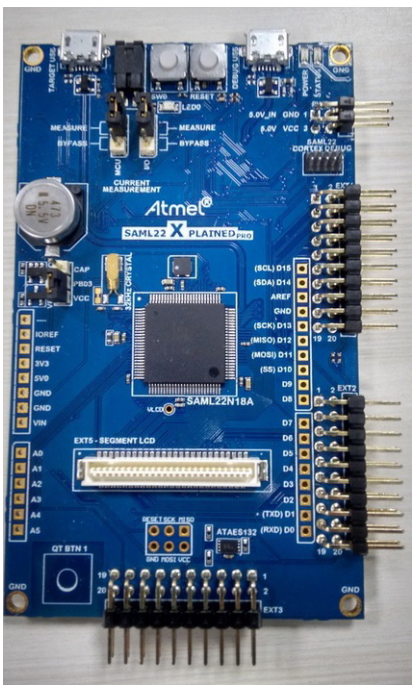
The overview of this section contains information about hardware setup and software setup.

4.1. Hardware Setup

The SAM L22 Xplained Pro kit will be used to run the example application. This is an evaluation kit that allows connecting multiple external components via a wing connector. A wing board is a self-contained board that can be connected to the Xplained Pro using a wing connector. The SAM L22 Xplained Pro kit has three such wing connector marked as EXT1, EXT2, and EXT3.

There are two USB ports on the SAM L22 Xplained Pro board - **DEBUG USB** and **TARGET USB**. For debugging using the Embedded debugger EDBG, **DEBUG USB** port has to be connected.

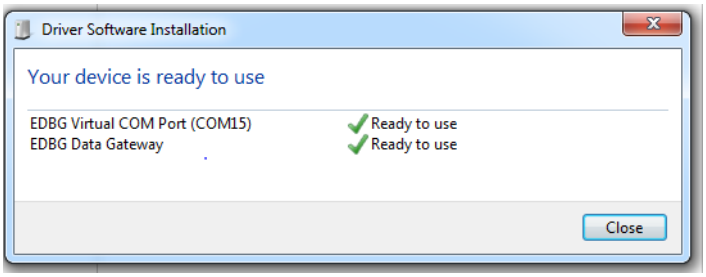
Figure 4-1. SAM L22 Xplained Pro Board



4.2. Software Setup

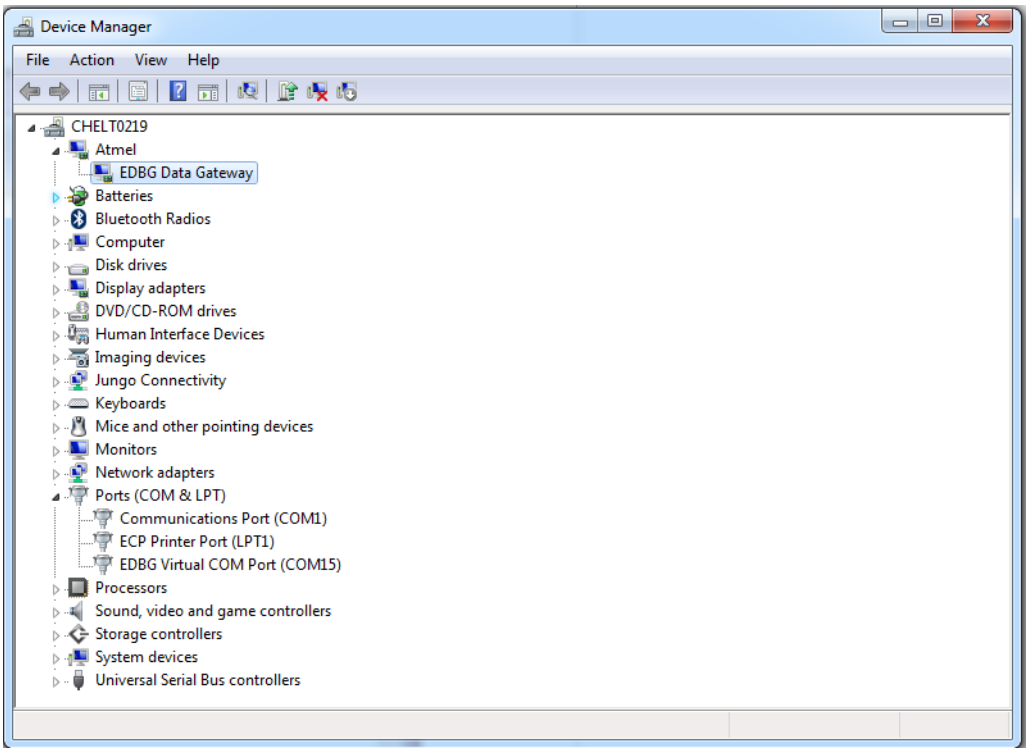
When SAM L22 Xplained Pro kit is connected to the PC, the Windows® Task bar will pop-up a message as shown in the following figure:

Figure 4-2. SAM L22 Xplained Pro Driver Installation



If the driver installation is successful, EDBG will be listed in the Device Manager as shown in the following figure:

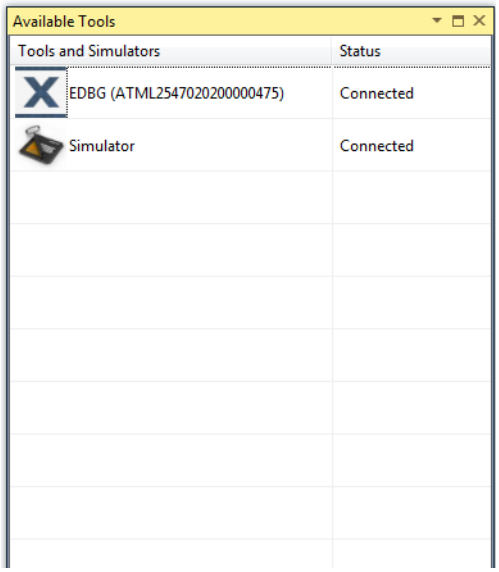
Figure 4-3. Successful EDBG Driver Installation



To ensure that the EDBG tool is getting detected in Atmel Studio,

1. Open Atmel Studio7, Go to **View** → **Available Atmel Tools**. The EDBG should get listed in the tools as "EDBG" and the tool status should display as **Connected**. This indicates that the tool is communicating with Atmel Studio.

Figure 4-4. EDBG under Available Atmel Tools

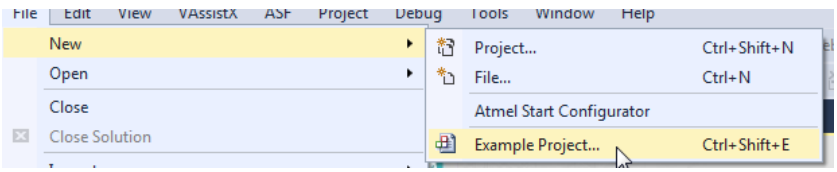


2. If the tool does not get displayed in **Available Atmel Tools**, disconnect the tool and reconnect again.
3. Right click on the tool in the **Available Tools** list, click on **Upgrade**. This will check whether the firmware in the tool is up to date. Click on **upgrade** to upgrade the firmware of the tool to latest version. In case you get "upgrade failed" error, cycle power the tool and then try upgrading again.

The SAM L22 TCC Features example code is available in the latest ASF with Atmel Studio. Following steps should be followed to load the SAM L22 TCC features example code in the Atmel Studio:

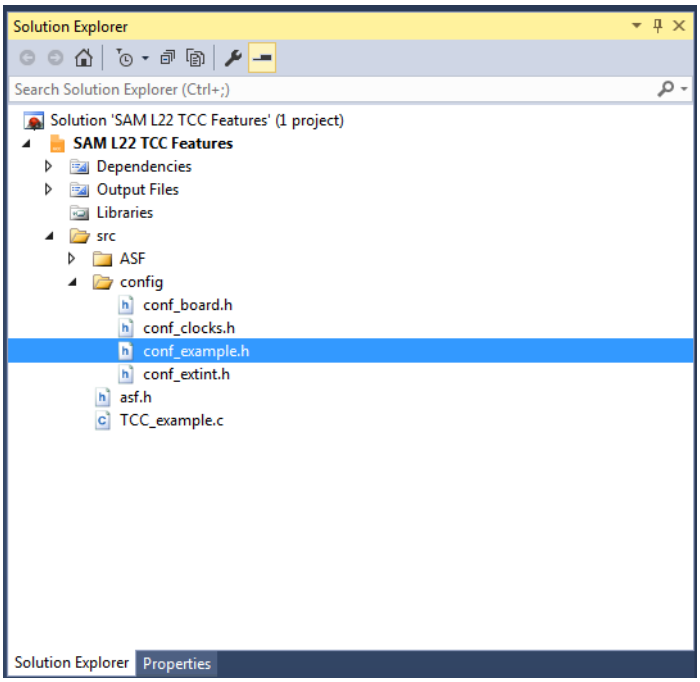
- To load the example project in Atmel Studio, go to **File** → **New** and click on **Example Project**. The shortcut key for to do this is (CTRL +Shift + E).

Figure 4-5. Creating Example Project in Atmel Studio



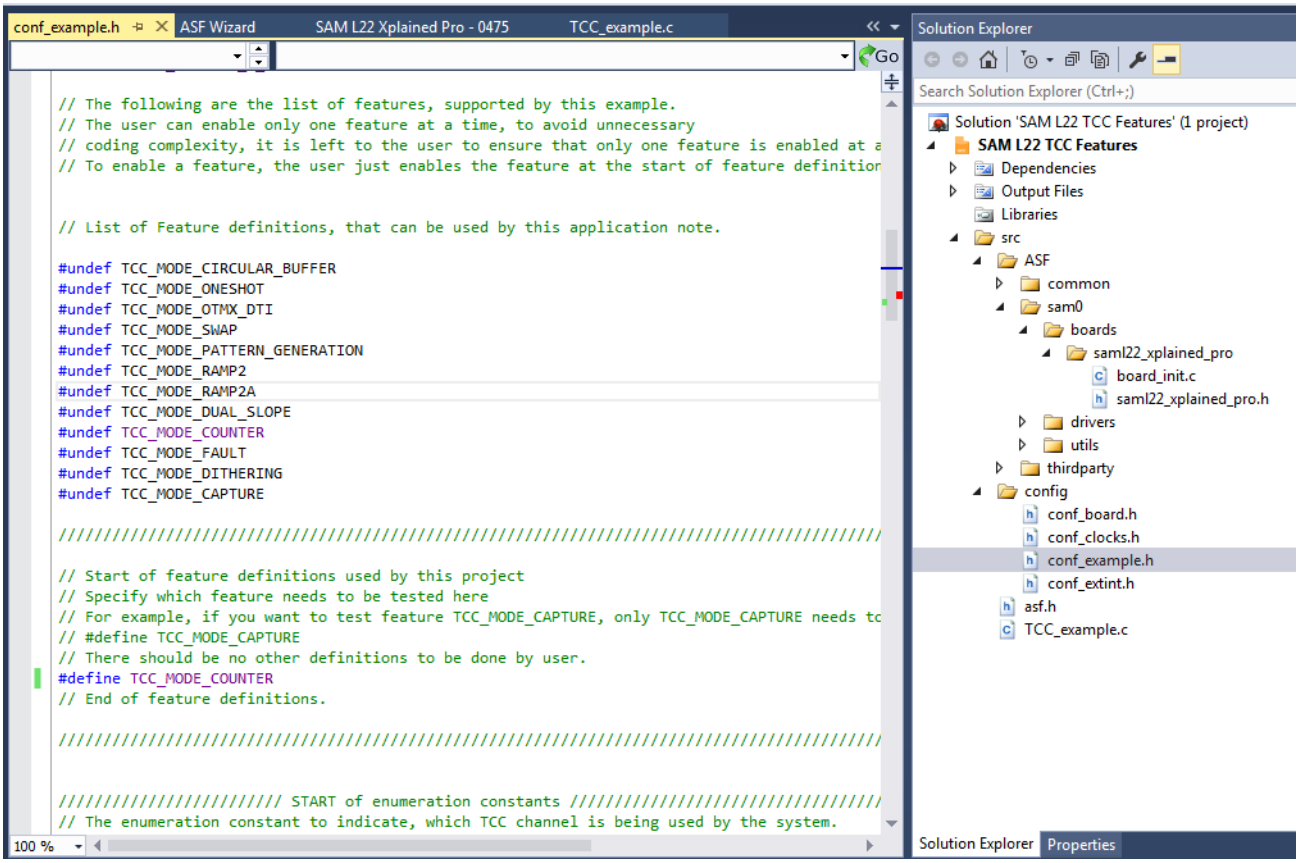
- Enter **SAM L22 TCC Features Example** in the search box from **New Example Project from ASF** so that it will show the **SAM L22 TCC Features Example** project solution available in the ASF
- Provide a name for the project and select the destination path and click **OK**. The location of the project is selected by choosing a specific Folder in **Location** Tab.
- After clicking **OK**, the **SAM L22 TCC Features Example** project has been loaded in the Atmel Studio as shown in the following figure

Figure 4-6. Solution Explorer View of SAM L22 TCC Features Example Project



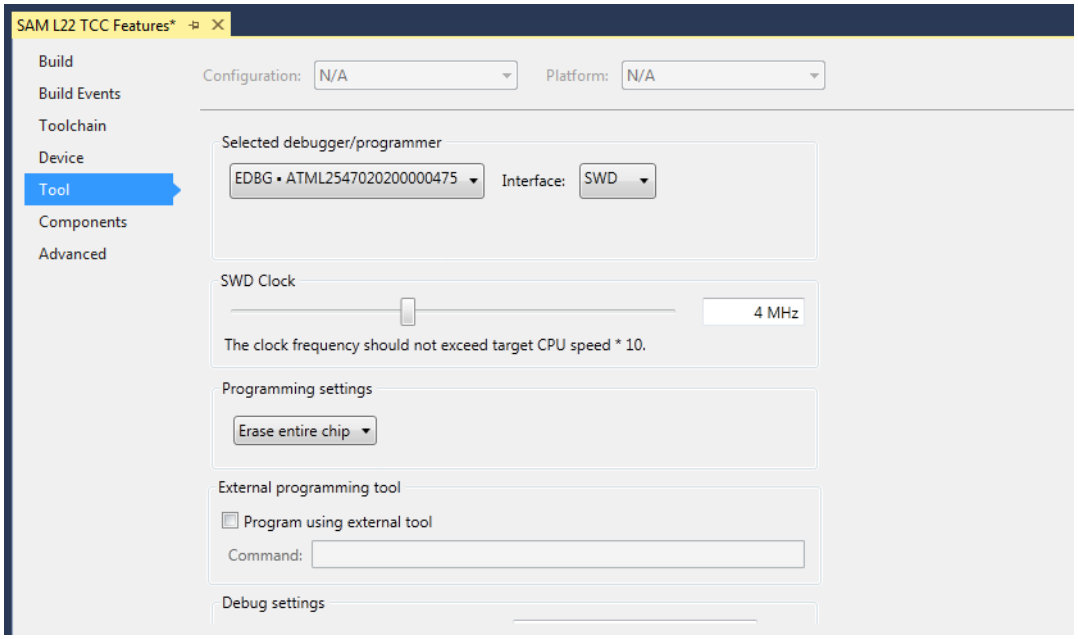
- **SAM L22 TCC Features Example** project contains `conf_example.h` file, which has the macro definitions for each feature. Only one feature should be enabled at a time for the proper operation of this application.

Figure 4-7. TCC Features Definition in conf_example.h



- After enabling the desired feature in `conf_example.h` file, compile the project by selecting **Build → Build solution**.
- To debug this example project code in Atmel Studio, Configure the Tool and Interface in the Project properties. To open the project properties, go to **Project menu → Properties**. In the project properties, go to **Tool tab → Under the Selected Debugger/Programmer**, select the tool as **EDBG** and interface as **SWD** as shown in the following figure.

Figure 4-8. Tool and Interface Settings



To program and execute the application, there are two options in Atmel Studio:

- Start a debug session on the board, where the user will be able to program and debug.
- Program the generated hex file into the controller and execute the application.

Both these options can be done on SAM L22 Xplained Pro as shown in the following figures.

Figure 4-9. Start without Debugging

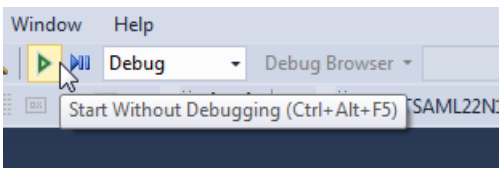
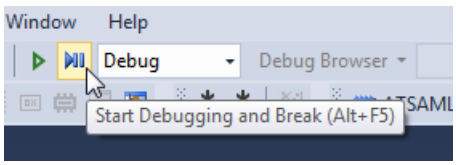


Figure 4-10. Start Debugging and Break



5. TCC Features Demonstration

5.1. Timer Mode Configuration

The example application code (SAM L22 TCC Features Example Project) contains the source code for each configuration for the following features . The `conf_example.h` contains the macro definitions related to each configuration and it helps to configure each mode by enabling each feature in `conf_example.h`.

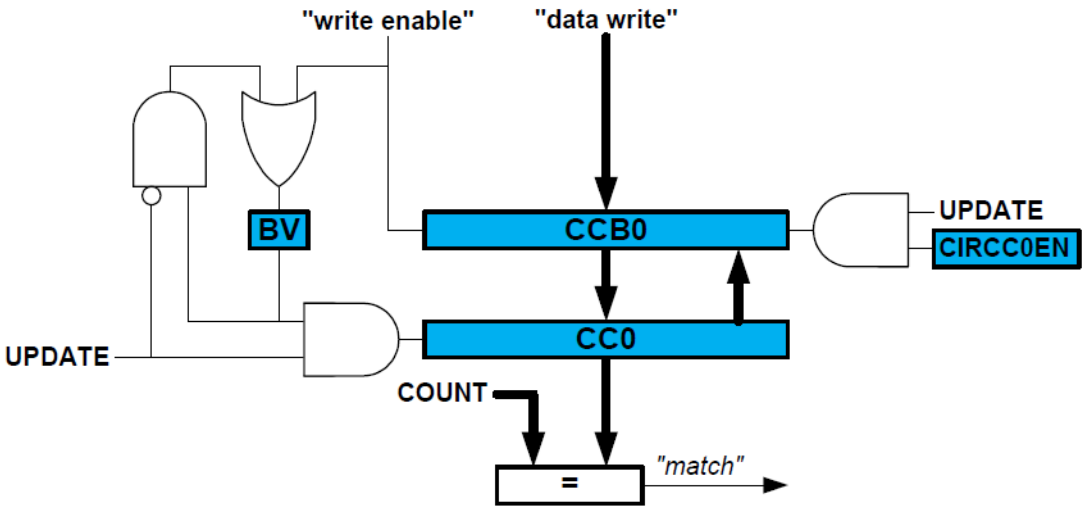
TCC clock frequency, TCC clock divider, and PERIOD values are defined based on the `GLCK_SOURCE, TCC_CLOCK_DIVIDER, and TCC_PERIOD_VALUE` values in the `conf_example.h` for each TCC features.

Also configurations specific to the feature can also be modified by the user for example RAMP2 configuration will have `TCC_RAMP2_MATCH_CHANNEL_0` and `TCC_RAMP2_MATCH_CHANNEL_1` as well.

5.2. Circular Buffer

The Period register (PER) and the compare channels register (CC0 to CC3) support circular buffer operation. When circular buffer operation is enabled, at each update condition, the PER or CCx values are copied into the corresponding buffer registers and the values in the buffer registers are copied into the PER or CCx registers. This mode uses compare channels of TCC to generate output signals with different pulse width in alternate cycles. It is mainly used in RAMP operations.

Figure 5-1. Circular Buffer on Channel 0



5.2.1. Circular Buffer Mode Configuration

The Circular Buffer feature has been enabled through `#define TCC_MODE_CIRCULAR_BUFFER` and `#undef` the rest of the TCC features in the `conf_example.h`.

For example, here two different compare values are loaded in the CC0 and CC0B register respectively to view the circular buffer effect on the channel 0. Hence the WO [0] and circular buffer for the channel 0 are enabled; the output signals are obtained with different pulse width on alternate cycles.

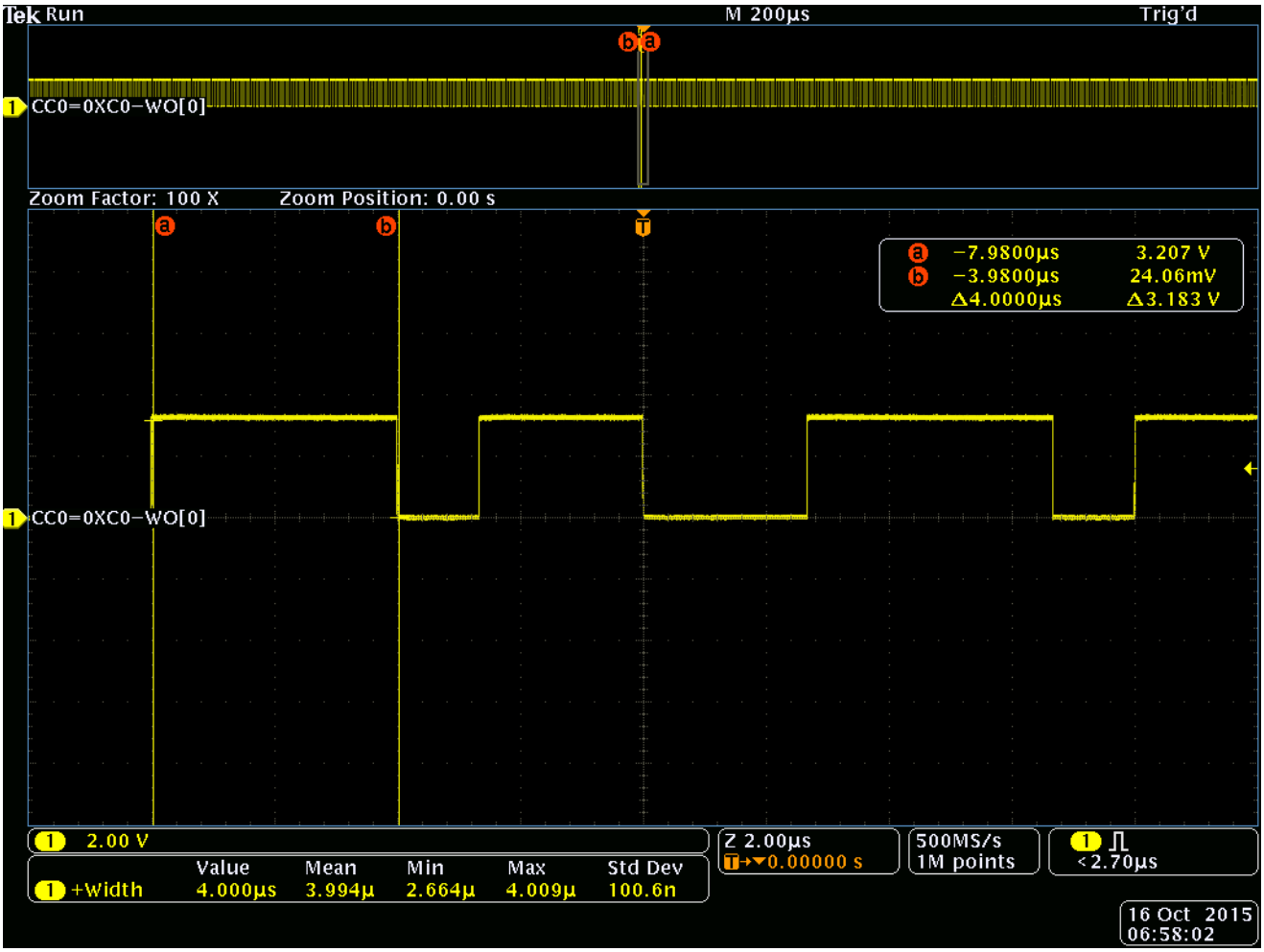
5.2.2. Code Snippet

```
/* Configure the TCC Waveform Output pins for waveform generation output
        */
        config_tcc.pins.enable_wave_out_pin[TCC_MATCH_CAPTURE_CHANNEL_0] =
true;
        config_tcc.pins.wave_out_pin[TCC_MATCH_CAPTURE_CHANNEL_0] =
PIN_PB18F_TCC0_WO0;/* Configure the Alternate function of GPIO pins for TCC
        functionality */

        config_tcc.pins.wave_out_pin_mux[TCC_MATCH_CAPTURE_CHANNEL_0] =
MUX_PB18F_TCC0_WO0;/* Load the CC0 and CCB0 values respectively for the circular
buffer
        operation */
        stat = tcc_set_double_buffer_compare_values(&tcc_instance,
TCC_MATCH_CAPTURE_CHANNEL_0, CC0_Value, CCB0_Value);/* Enable the Circular Buffer
feature for the Compare Channel 0
        */

        stat = tcc_enable_circular_buffer_compare(&tcc_instance,
TCC_MATCH_CAPTURE_CHANNEL_0);
```

5.2.3. Waveform Output

The output scope snapshots Figure 5-2 Circular Buffer Enable CC0 = 0xC0 on page 15 and Figure 5-3 Circular Buffer Enable CC0 = 0x80 on page 16 are captured from the SAM L22 Xplained Pro PB18 available in the EXT3 connector Pin No 9.

Figure 5-2. Circular Buffer Enable CC0 = 0xC0



The TCC clock frequency = 48MHz

The TCC Clock divider = 1

Time Period for 1 Count = 1/48000000 = 20.83333ns

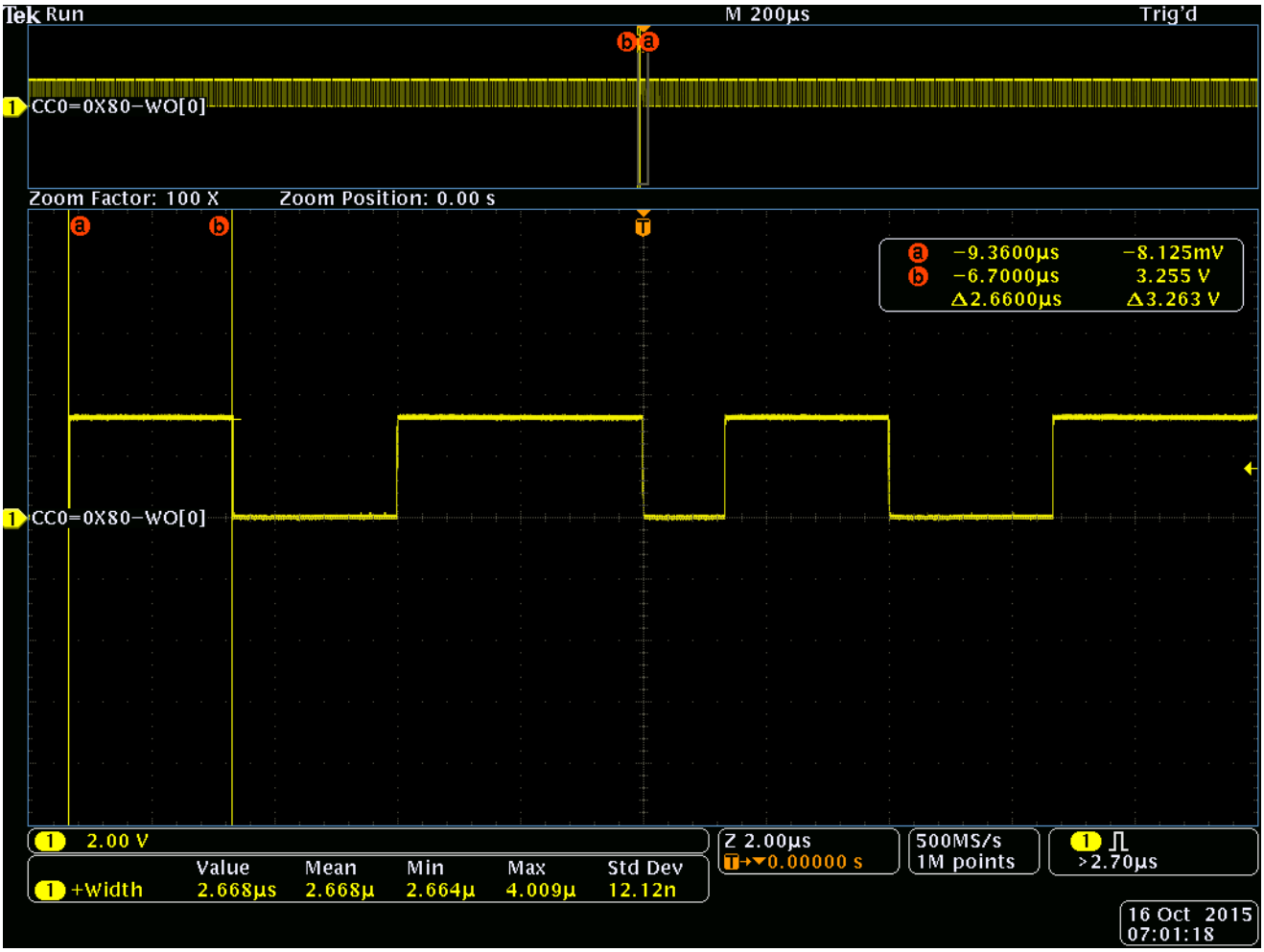
For the CC0 value 0xC0 = 192 * 20.8333ns

= 4.0000µs

For the CC0 value 0x80 = 128* 20.8333ns

= 2.666µs

Figure 5-3. Circular Buffer Enable CC0 = 0x80



5.3. One-shot Operation

When one-shot feature is enabled, the counter automatically stops on the next counter overflow or underflow condition. When the counter is stopped, STOP bit in the STATUS register will be set.

This one-shot operation can be enabled by writing a one to the One-Shot bit in the Control B Set register (CTRLBSET.ONESHOT) and disabled by writing a one to the One-Shot bit in the Control B Clear register (CTRLBCLR.ONESHOT). The one-shot operation can be restarted by using retrigger software command, a retrigger event or a start event. When the counter restarts its operation, Stop bit in the Status register (STATUS.STOP) is get cleared.

5.3.1. One-shot Operation Configuration

The one-shot operation feature has been enabled through `#define TCC_MODE_ONESHOT` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this mode, configure the compare match value in CC2 channel WO [6] for the waveform output. Since the Port pin PC27 is connected with LED0 of the SAM L22 Xplained Pro, it will control the ON time of LED0. Pressing BUTTON_0 of the SAM L22 Xplained Pro restarts the timer as the pulse will be obtained on the PC27 pin it will drive the LED0. It is important to enable the inversion of waveform input WO [3], since the PC27 pin is connected to the cathode pin of LED0.

5.3.2. Code Snippet

```
/* Configure the TCC Waveform Output pins for waveform generation output
        */
        config_tcc.pins.enable_wave_out_pin[TCC_CHANNEL_NUM_3] = true;
        config_tcc.pins.wave_out_pin[TCC_CHANNEL_NUM_3] = PIN_PC27F_TCC0_WO3;/*
Configure the alternate function of GPIO pins for TCC
        functionality */

        config_tcc.pins.wave_out_pin_mux[TCC_CHANNEL_NUM_3] = MUX_PC27F_TCC0_WO3;/*
Configure the Match value for the compare channel 2 for LED0 ON
        time*/

        config_tcc.compare.match[2] = 31250;
      /* Invert the Waveform output[6] channel to drive LED0
        */

        config_tcc.wave_ext.invert[TCC_CHANNEL_NUM_3] = true;
      /* Enable the One shot Feature */

        config_tcc.counter.oneshot = true;void oneshot_operation(void)
{
        while(port_pin_get_input_level(BUTTON_0_PIN));
        while(!port_pin_get_input_level(BUTTON_0_PIN));
        tcc_set_count_value(&tcc_instance, 0);
        tcc_restart_counter(&tcc_instance);
}
```

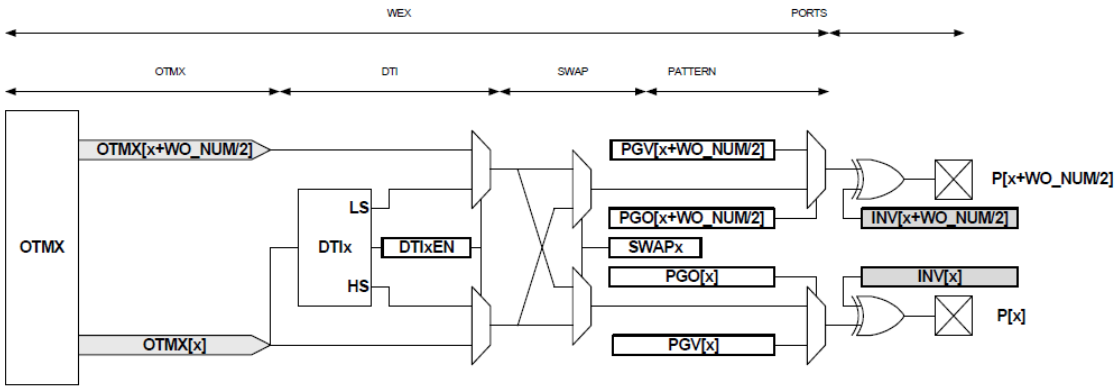
5.3.3. LED0 Output

After enabling this mode, LED0 will blink once for the time period loaded in CC2 channel. As the one-shot feature is enabled the counter will be stopped . If the Button [Button_0] is pressed, then it will restart the counter operation by retrigger command and blinks LED0 once again.

5.4. Output Matrix with DTI for PWM

The output matrix (OTMX) can distribute and route the TCC waveform outputs across the port pins in different configurations, each optimized for different application types. The OTMX [1:0] bits in the WEXCTRL register define the output matrix configuration. The block diagram of waveform extension detail is shown in the following figure.

Figure 5-4. Waveform Extension Stage Details



The output matrix (OTMX) unit distributes compare channels, according to the selectable configurations, as shown in the following table.

Table 5-1. Output Matrix Channel Pin Routing Configuration

Value	OTMX[x]							
0x0	CC3	CC2	CC1	CC0	CC3	CC2	CC1	CC0
0x1	CC1	CC0	CC1	CC0	CC1	CC0	CC1	CC0
0x2	CC0	CC0	CC0	CC0	CC0	CC0	CC0	CC0
0x3	CC1	CC1	CC1	CC1	CC1	CC1	CC1	CC0

- Configuration 0x0 is default configuration. The channel location is the default one and channels are distributed on outputs modulo the number of channels. Channel 0 is routed to the Output matrix output OTMX [0], Channel 1 to OTMX [1]. If there are more outputs than channels, then channel 0 is duplicated to the Output matrix output OTMX[CC_NUM], channel 1 to OTMX[CC_NUM+1] and so on.
- Configuration 0x1 distributes the channels on output modulo half the number of channels; this gives the lower channels twice the number of output locations than the default configuration. This provides for example, control of the four transistors of a full bridge using only two compare channels. Using pattern generation, some of these four outputs can be overwritten by a constant level, enabling flexible drive of a full bridge in all quadrant configurations.
- Configuration 0x2 distributes the compare channel 0 (CC0) to all port pins. With pattern generation, this configuration can control a stepper motor.
- Configuration 0x3 distributes the compare channel CC0 to first output and the channel CC1 to all other outputs. Together with pattern generation and the fault extension this configuration can control up to seven LED strings, with a boost stage.

5.4.1. Dead Time Insertion (DTI)

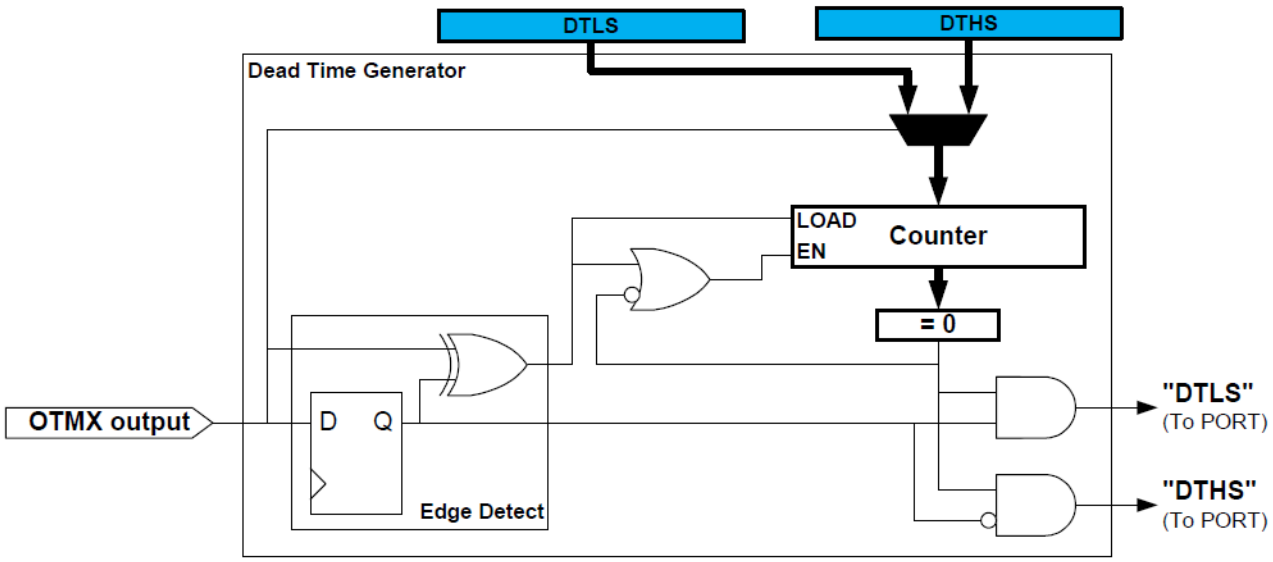
In a system driven by a pair of transistors operating in the Complementary Output mode it is completely forbidden to enable simultaneously the two FETs on the same side. This would lead to Shoot Through (a short circuit from power supply to ground).

Because the power output devices cannot switch instantaneously, some amount of time must be provided between the turn-off event of one PWM output in a complementary pair and the turn-on event of the other transistor.

The dead time insertion (DTI) unit splits the four lower OTMX outputs into two non-overlapping signals, the non-inverted low side (LS) and inverted high side (HS) of the waveform output with optional dead-time insertion between LS and HS switching.

The dead-time insertion (DTI) unit generates OFF time with the non-inverted low side (LS) and inverted high side (HS) of the WG output forced at low level. This OFF time is called dead time, and dead-time insertion ensures that the LS and HS will never switch simultaneously. The DTI stage consists of four equal dead-time insertion generators; one for each of the first four channels. The four channels have a common register which controls the dead time and is independent of high side and low side setting.

Figure 5-5. Block Diagram of Dead Time Generator



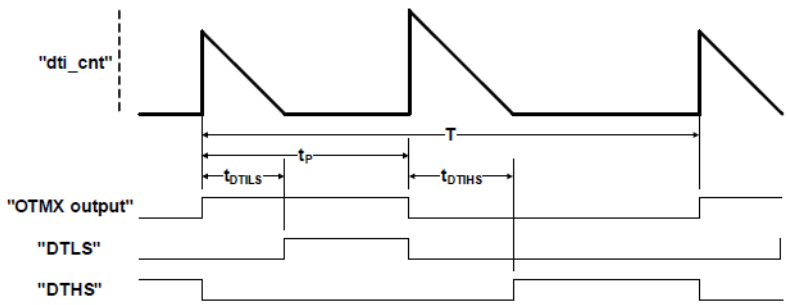
DTIENx [x=3-0] Dead-time Insertion Generator x Enable (8 -11) bits in the WEXCTRL register enable the Dead Time Insertion function for each channel.

The dead time function in the PWM control avoids the drivers of the same set of PWMs (PWMxH and PWMxL) from being ON simultaneously due to the operating speed of the driver during output generation.

Dead time must be inserted when any of the PWM I/O pin pairs are operating in the Complementary Output mode. Four DTI insertion functions (DTIEN0 to DTIEN3) control the four lowest OTMX outputs.

The 8-bit dead-time counter is decremented by one for each peripheral clock cycle, until it reaches zero. A nonzero counter value will force both the low side and high side outputs into their OFF state. When the output matrix (OTMX) output changes, the dead-time counter is reloaded according to the edge of the input. When the output changes from low to high (positive edge) it initiates counter reload of the DTLS register, and when the output changes from high to low (negative edge) reload the DTHS register.

Figure 5-6. Dead Time Generator Timing Diagram



5.4.2. Output Matrix with DTI Configuration

The Output Matrix with DTI for PWM mode feature has been enabled through `#define TCC_MODE_OTMX_DTI` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this mode, the waveform outputs 0, 1, 2, and 6 for the PWM output signals are enabled. The waveform output 0 and 1 are configured to view the Dead Time Insertion effect on the waveform output pins 2 and 6. Also the waveform output 1 is inverted, since the Dead time for the channel CC0 is not enabled . By

enabling the DTIEN bit for the channel, the complementary output for the particular channel can be obtained.

Load the Compare match values on the CC0, CC1, and CC2 channels appropriately to generate the waveform outputs. Enable the DTI on the channel and define the DTHS (dead time high side) and DTLS (dead time low side) using WEXCTRL register as per application need.

5.4.3. Code Snippet

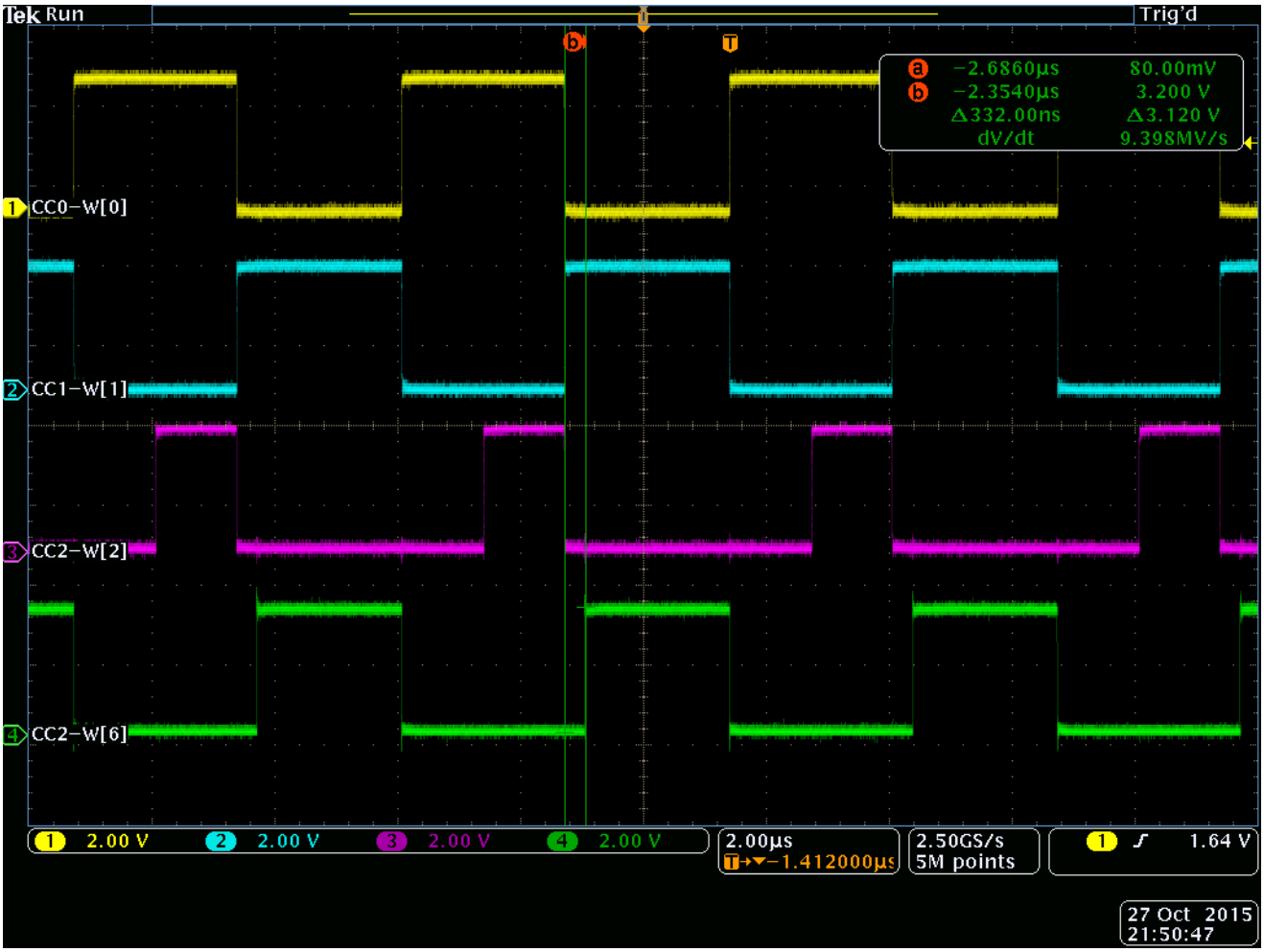
```
/* Configure the TCC Waveform Output pins for waveform generation output
        */
        config_tcc.pins.enable_wave_out_pin[0] = true;
        config_tcc.pins.enable_wave_out_pin[1] = true;
        config_tcc.pins.enable_wave_out_pin[2] = true;
        config_tcc.pins.enable_wave_out_pin[6] = true;
                    config_tcc.pins.wave_out_pin[0] =
PIN_PB184F_TCC0_WO0;
        config_tcc.pins.wave_out_pin[1] = PIN_PA09F_TCC0_WO1;
        config_tcc.pins.wave_out_pin[2] = PIN_PA18F_TCC0_WO2;
        config_tcc.pins.wave_out_pin[6] = PIN_PA12F_TCC0_WO6;/* Configure the
Alternate function of GPIO pins for TCC
        functionality */

        config_tcc.pins.wave_out_pin_mux[0] = MUX_PB184F_TCC0_WO0;
        config_tcc.pins.wave_out_pin_mux[1] = MUX_PA09F_TCC0_WO1;
        config_tcc.pins.wave_out_pin_mux[2] = MUX_PA18F_TCC0_WO2;
        config_tcc.pins.wave_out_pin_mux[6] = MUX_PA12F_TCC0_WO6;/* Configure the
compare channel values for the duty cycle control
        and load the 0x80 value for 50% duty cycle */
    config_tcc.counter.period = TCC_PERIOD_VALUE;
                config_tcc.compare.match[0] = TCC_PERIOD_VALUE/2;
                config_tcc.compare.match[1] = TCC_PERIOD_VALUE/2;
                config_tcc.compare.match[2] = TCC_PERIOD_VALUE/2;
    /* Invert the Waveform output[1] channel to view the DTI effect */
                config_tcc.wave_ext.invert[1] = true;
```

5.4.4. OTMX with DTI Enable for Channel 2 Waveform Output

The output scope snapshots Figure 5-7 OTMX with DTI for DTHS Measurement on page 21 and Figure 5-8 OTMX with DTI for DTLS Measurement on page 22 are captured from the SAM L22 Xplained Pro of EXT2 and EXT3 connector.

- PB18 – Waveform Output 0
- PA09 – Waveform Output 1
- PA18 – Waveform Output 2
- PA12 – Waveform Output 6

Figure 5-7. OTMX with DTI for DTHS Measurement



In the diagram above, the DTHS is measured for the PWM complementary output.

The TCC clock frequency = 48MHz

The TCC Clock divider = 1

Time Period for 1 Count = 1/48000000

= 20.833ns

DTHS time = 16 * 20.83333

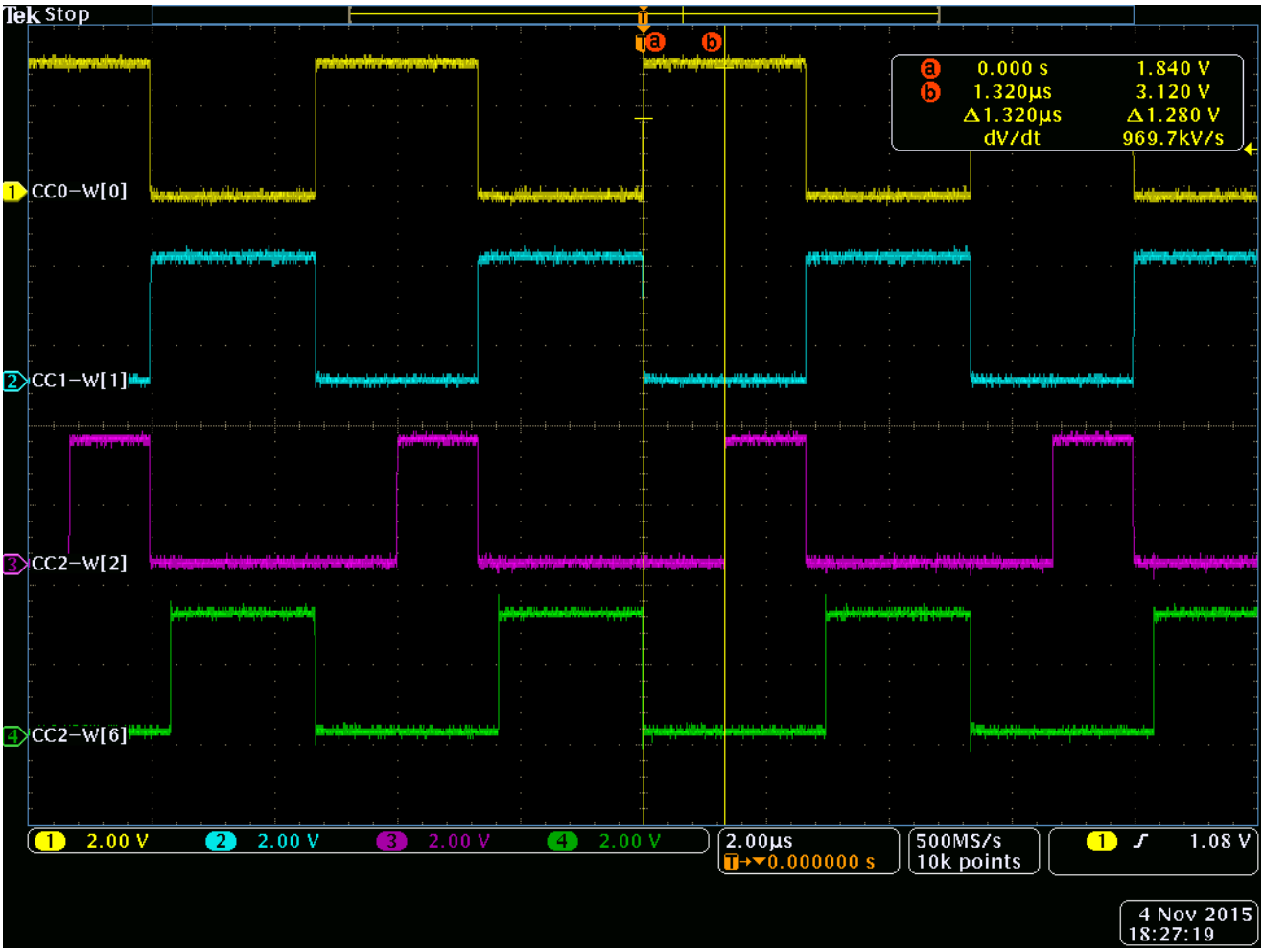
= 333.333ns

DTLS time = 64 * 20.8333

= 1.3333µs

Since this application uses the internal OSC 8MHz as a source for the CPU clock, the DTLS, and DTHS have some tolerance.

Figure 5-8. OTMX with DTI for DTLS Measurement



5.5. SWAP Operation

The SWAP feature is useful to switch simultaneously two output signals. The swap (SWAP) unit can be used to swap waveform pin outputs. The SWAP units in the TCC module can be seen as a four port pair of slices.

- SWAP0 acting on port pins (WO[0], WO[WO_NUM/2 +0])
- SWAP1 acting on port pins (WO[1], WO[WO_NUM/2 +1])

And more generally:

- SWAPx acting on port pins (WO[x], WO[WO_NUM/2 +x])

The Bits 27:24 – SWAPx [x=3-0] of WAVE register: Setting these bits enables output swap of DTI outputs [x] and [x+WO_NUM/2].

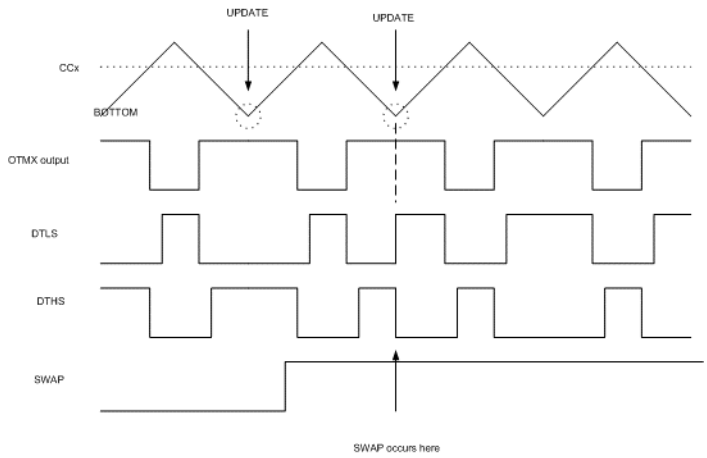
The swap function is very useful in BLDC motor control and can be used for fast decay motor control. It allows the immediate change of top and bottom transistors in the phase. Using this function the rotor commutation and speed control can be divided into two independent program parts. The state of the control signals can be changed immediately when required by the motor position (phase commutation) without changing the content of the PWM value registers. These changes can be accomplished asynchronously to the PWM duty cycle update.

When chopping current threshold is reached, the H-bridge can operate in two different current recirculation modes:

- An asynchronous mode if current re-circulates through the diodes (in FETs or external). The user cannot control the occurrence of the alternate path creation.
- A synchronous mode if enabling and disabling FETs in order to promote an alternate path

Two synchronous modes can be used: fast decay or slow decay. Fast and slow refer to the current decay mode and not the motor speed. It is the opposite for speed. In fast decay mode, the motor will slow down in speed while in slow decay mode, the motor stops very quickly.

Figure 5-9. SWAP Operation Illustration



5.5.1. SWAP Mode Configuration

The SWAP feature has been enabled through `#define TCC_MODE_SWAP` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this mode configure and enable the Waveform outputs 0 and 4 for single slope PWM waveform generation, by enabling the DTIEN0 bit for the channel 0, the complementary output for the channel is obtained on the WO [4] pin. Using the WEXCTRL register, the DTLS and DTHS for the complementary output is defined. After the configuration part is done, it continuously waits for the Button press [Button_0] available in the SAM L22 Xplained Pro. If the button is pressed, then it will toggle the SAWP0 bit in the WAVE register for the SWAP operation. So that WO [0] pin waveform can be output on the WO [4] and vice versa.

5.5.2. Code Snippet

```
/* Configure the TCC Waveform Output pins for waveform generation output
        */
        config_tcc.pins.enable_wave_out_pin[0] = true;
        config_tcc.pins.enable_wave_out_pin[4] = true;
        config_tcc.pins.wave_out_pin[0] = PIN_PA04F_TCC0_WO0;
        config_tcc.pins.wave_out_pin[4] = PIN_PA22F_TCC0_WO4;/* Configure the
alternate function of GPIO pins for TCC
        functionality */

        config_tcc.pins.wave_out_pin_mux[0] = MUX_PA04F_TCC0_WO0;
        config_tcc.pins.wave_out_pin_mux[4] = MUX_PA22F_TCC0_WO4;/* Configure the
compare channel values for the duty cycle control
        and load the 0x80 value for 50% duty cycle */

        config_tcc.compare.match[0] = 0x80;/* Enable the Dead Time Insertion
Generator for the channel 0 (CC0)
        */
```

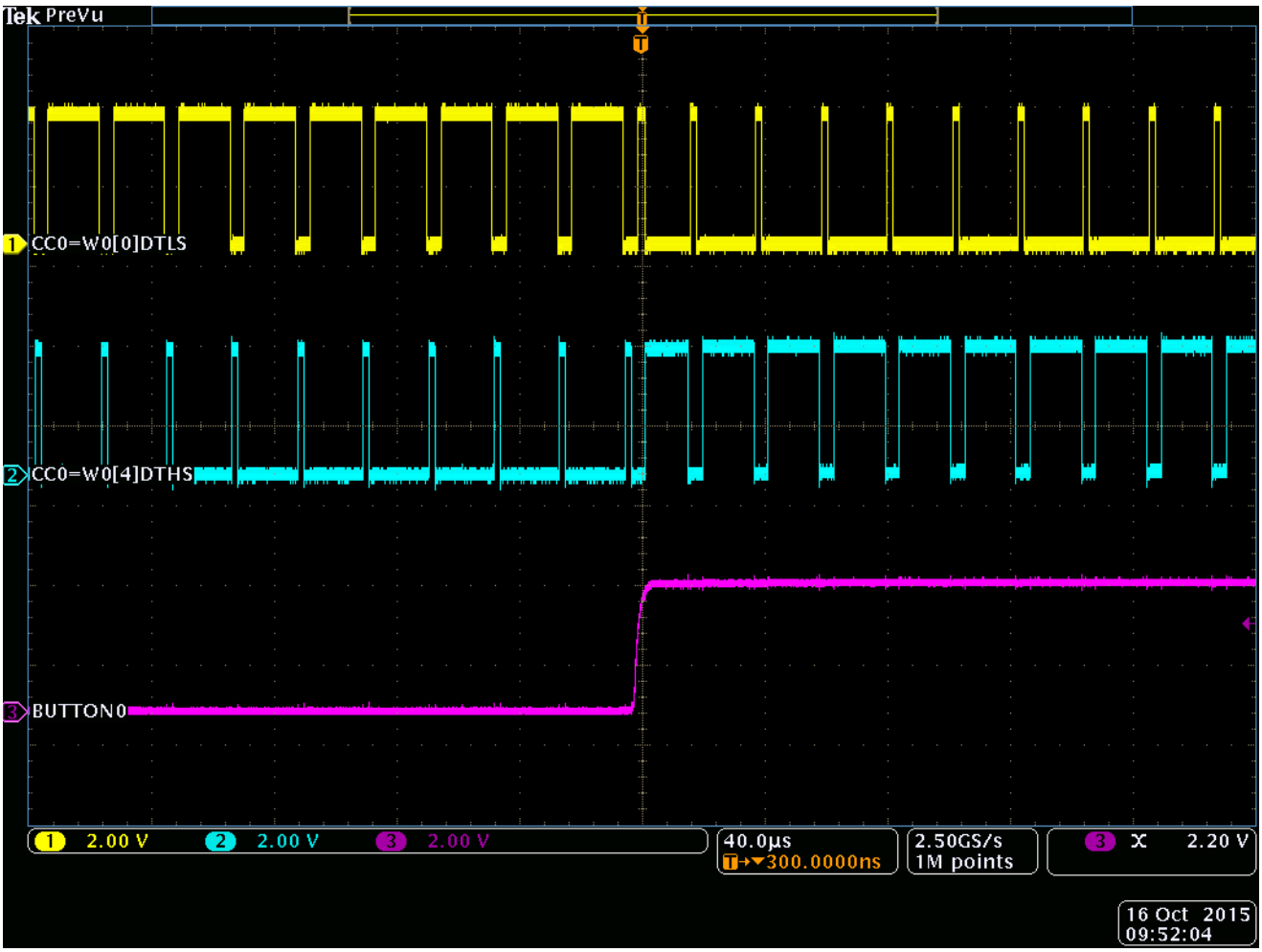
```
        TCC0->WEXCTRL.reg |= TCC_WEXCTRL_DTIEN0; /* Define the High side time and
Low side time for Dead Time
        generation */

        TCC0->WEXCTRL.reg |= TCC_WEXCTRL_DTLS(0x20) | TCC_WEXCTRL_DTHS(0x60);void
swap_operation(void) {
        while(port_pin_get_input_level(BUTTON_0_PIN));
        while(!port_pin_get_input_level(BUTTON_0_PIN));
        TCC0->WAVE.reg ^= TCC_WAVE_SWAP0;
      }
```

5.5.3. SWAP Waveform Output

The output scope snapshot in Figure 5-10 SWAP Operation on page 24 is captured from the SAM L22 Xplained Pro of EXT1, EXT2, and EXT3 .

- PA04 – Waveform Output 0
- PA22 – Waveform Output 4
- PA14 – BUTTON_0_PIN

Figure 5-10. SWAP Operation



The TCC clock frequency = 48MHz

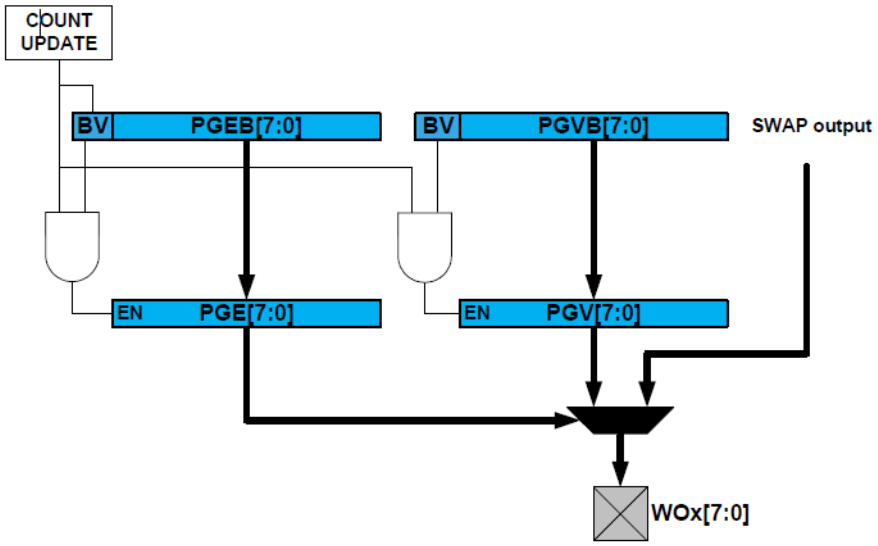
The TCC Clock divider = 1

Time Period for 1 Count = 1/48000000 = 20.833ns

5.6. Pattern Generation

The pattern generation unit is used to generate synchronized bit pattern across the waveform output pins. As with other double buffered timer/counter registers, the register update is synchronized to the UPDATE condition set by the timer/counter waveform generation mode. If the application does not need synchronization, the application code can simply access the PGEx and PGVx registers directly. A value 1 in the PGEx bit group of PATT register overrides the corresponding SWAP output with the corresponding PGVx value. The PATTBV bit of the STATUS register used to set when a new value is written to the PATTB register. The PATTBV bit is automatically cleared by hardware on UPDATE condition or by writing a one to this bit. When double buffering is enabled, PGVB and PGEB bits value of PATTB register is copied into the corresponding PGV and PGE bits value of PATT register on an update condition. Pattern Generator can be used with PWM signals which have built-in DTI. A block diagram of the pattern generator is shown in Figure 5-11 Block Diagram of Pattern Generator on page 25.

Figure 5-11. Block Diagram of Pattern Generator



5.6.1. Pattern Generation Configuration

The Pattern Generation feature has been enabled through `#define TCC_MODE_PATTERN_GENERATION` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this mode, the waveform outputs 0, 1, 2, and 3 for the pattern generation are enabled. Here four patterns are defined for bipolar stepper motor with the waveform output . Configure the OTMX [1:0] bits into 0x2 in the WEXCTRL register in such way that to get the CC0 waveform output on all the four waveform output pins.

The application waits for the compare match flag to set and then clears the Compare Match Interrupt flag of the same. Then it will load the next pattern on the PGVB (Pattern Generation Value Buffer) appropriately for the next pattern.

- In this application note the delay required between the patterns is not implemented. Add the appropriate delay between loading the pattern as mentioned in the stepper motor datasheet.

5.6.2. Code Snippet

```
/* Configure the TCC
       Waveform Output pins for waveform generation output */
```

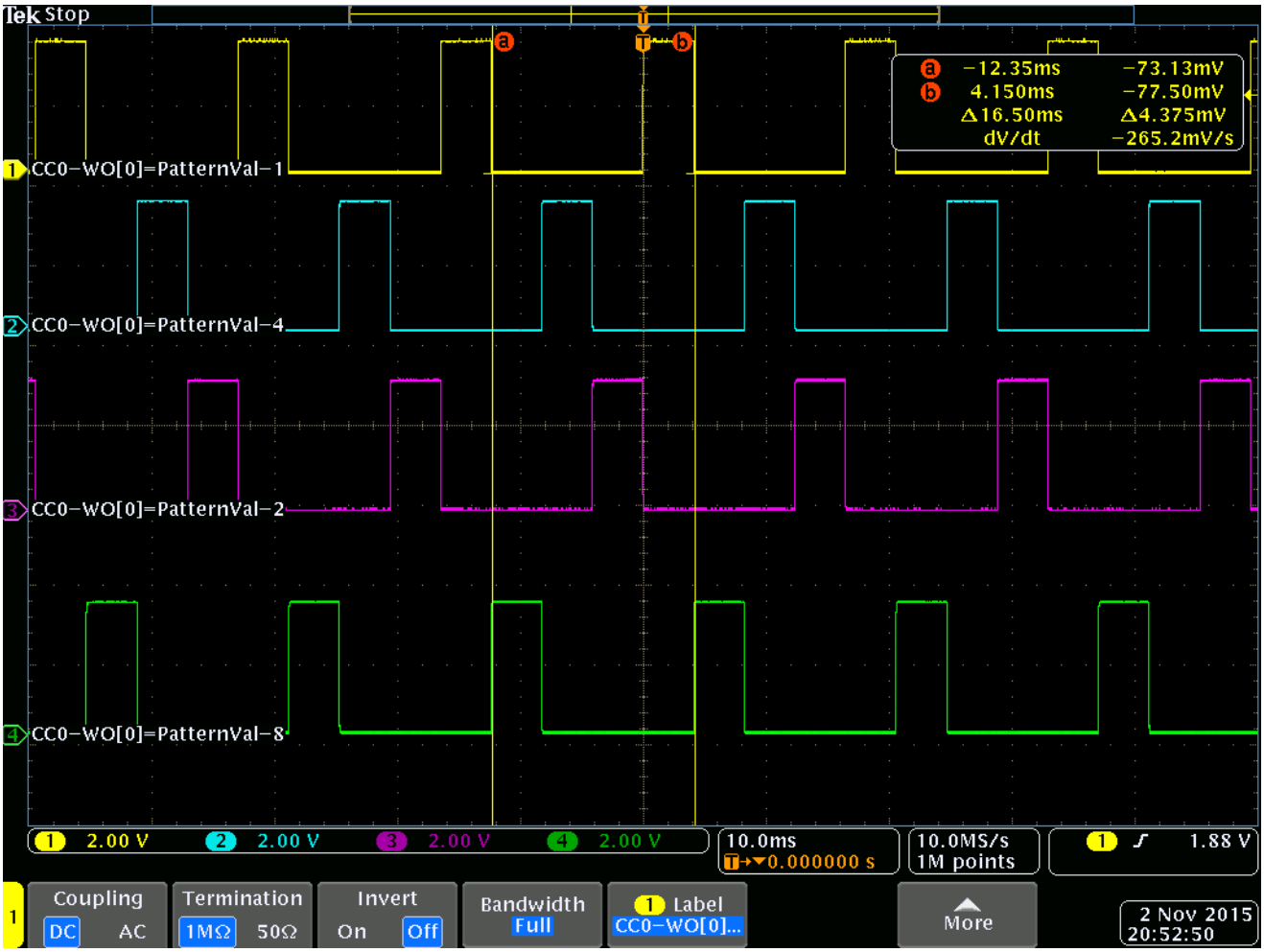
```
        config_tcc.pins.enable_wave_out_pin[0] = true;
        config_tcc.pins.enable_wave_out_pin[1] = true;
        config_tcc.pins.enable_wave_out_pin[2] = true;
        config_tcc.pins.enable_wave_out_pin[3] = true;
        config_tcc.pins.wave_out_pin[0] = PIN_PB18F_TCC0_WO0;
        config_tcc.pins.wave_out_pin[1] = PIN_PA09F_TCC0_WO1;
        config_tcc.pins.wave_out_pin[2] = PIN_PA18F_TCC0_WO2;
        config_tcc.pins.wave_out_pin[3] = PIN_PB21F_TCC0_WO3;/* Configure the
Alternate function of GPIO pins for TCC
        functionality */

        config_tcc.pins.wave_out_pin_mux[0] = MUX_PB18F _TCC0_WO0;
        config_tcc.pins.wave_out_pin_mux[1] = MUX_PA09F _TCC0_WO1;
        config_tcc.pins.wave_out_pin_mux[2] = MUX_PA18F _TCC0_WO2;
        config_tcc.pins.wave_out_pin_mux[3] = MUX_PB21F _TCC0_WO3;
        config_tcc.double_buffering_enabled = true;/* Configure the compare channel
values for the duty cycle control
        and Load the 0x7FFF value for 50% duty cycle */

        config_tcc.compare.match[0] = 0x7FFF;/* Configure the Output Matrix Channel
for Pattern Generation of
        Stepper Motor */

        TCC0->WEXCTRL.reg |= TCC_WEXCTRL_OTMX(2); /* Enable the Pattern Generator
Output for 4 Waveform Outputs
        and Load the PATT and PATTB register values respectively for Stepper Motor
Pattern
        Generation */

        TCC0->PATT.reg = TCC_PATT_PGE(0x0F) | TCC_PATT_PGV(SM_Pattern[i+
+]);
        TCC0->PATTB.reg = TCC_PATTB_PGEB(0x0F) | TCC_PATTB_PGVB(SM_Pattern[i++]);
        void pattern_generation(void) {
        if(i == 4)
        i = 0;
        while(!TCC0->INTFLAG.bit.MC0);
        TCC0->INTFLAG.bit.MC0 = 1;
        TCC0->PATTBUF.reg = TCC_PATTBUF_PGEB(0x0F) | TCC_PATTBUF_PGVB(SM_Pattern[i+
+]);
        while (CONF_PWM_MODULE->SYNCBUSY.reg & (1u << 16));
        }
```

5.6.3. Pattern Generation Waveform Output

The following scope snapshot is captured from the SAM L22 Xplained Pro of EXT1, EXT2, and EXT3.

- PB18F – Waveform Output 0
- PA09F – Waveform Output 1
- PA18F – Waveform Output 2
- PB21F – Waveform Output 3

Figure 5-12. Pattern Generation



The TCC clock frequency = 16MHz

The TCC Clock divider = 1

Time Period for 1 Count = 1/16000000 = 62.5ns

For the CC0 value 0x7FFF = 0x7FFF * 62.5ns

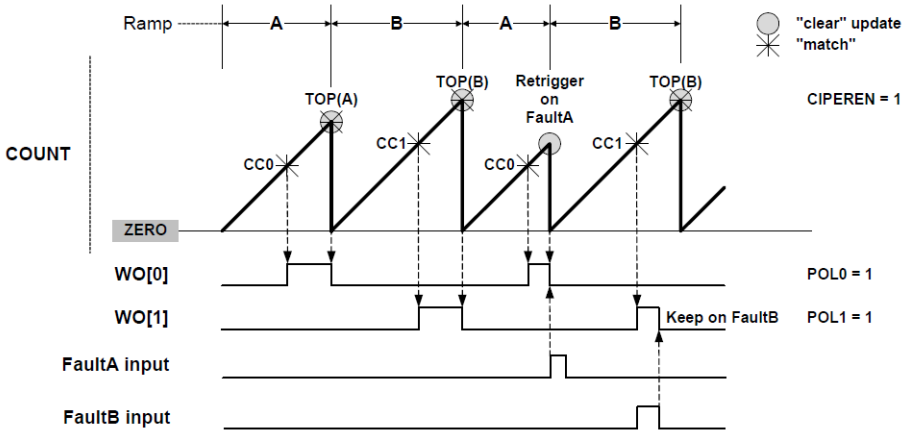
= 2.047ms

5.7. Ramp2 Operation

These operations are dedicated for Half-Bridge and Push-Pull SMPS topologies, where two consecutive Timer/counter cycles are interleaved, as shown in Figure 5-13 RAMP2 Standard Operation on page 28. In cycle A, odd channels output is disabled, and in cycle B, even channels output are disabled.

Ramp A and B periods are controlled through PER register value. Period register value can have different values on each ramp by enabling the circular buffer option CIPEREN bit in the WAVE register. The 4th and 5th bits RAMP [1:0] in the WAVE register configure the RAMP mode. The RAMP2 mode uses two compare channels TCC to generate two output signals, or one output signal with another CC channel enabled in capture mode.

Figure 5-13. RAMP2 Standard Operation



5.7.1. RAMP2 Configuration

The RAMP2 feature has been enabled through `#define TCC_MODE_RAMP2` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this mode, the Waveform outputs 0 and 1 for the single slope PWM output signals are enabled. Configure the Compare match channel values for the Channel 0 and 1. RAMP2 mode can be configured through WAVE register.

5.7.2. Code Snippet

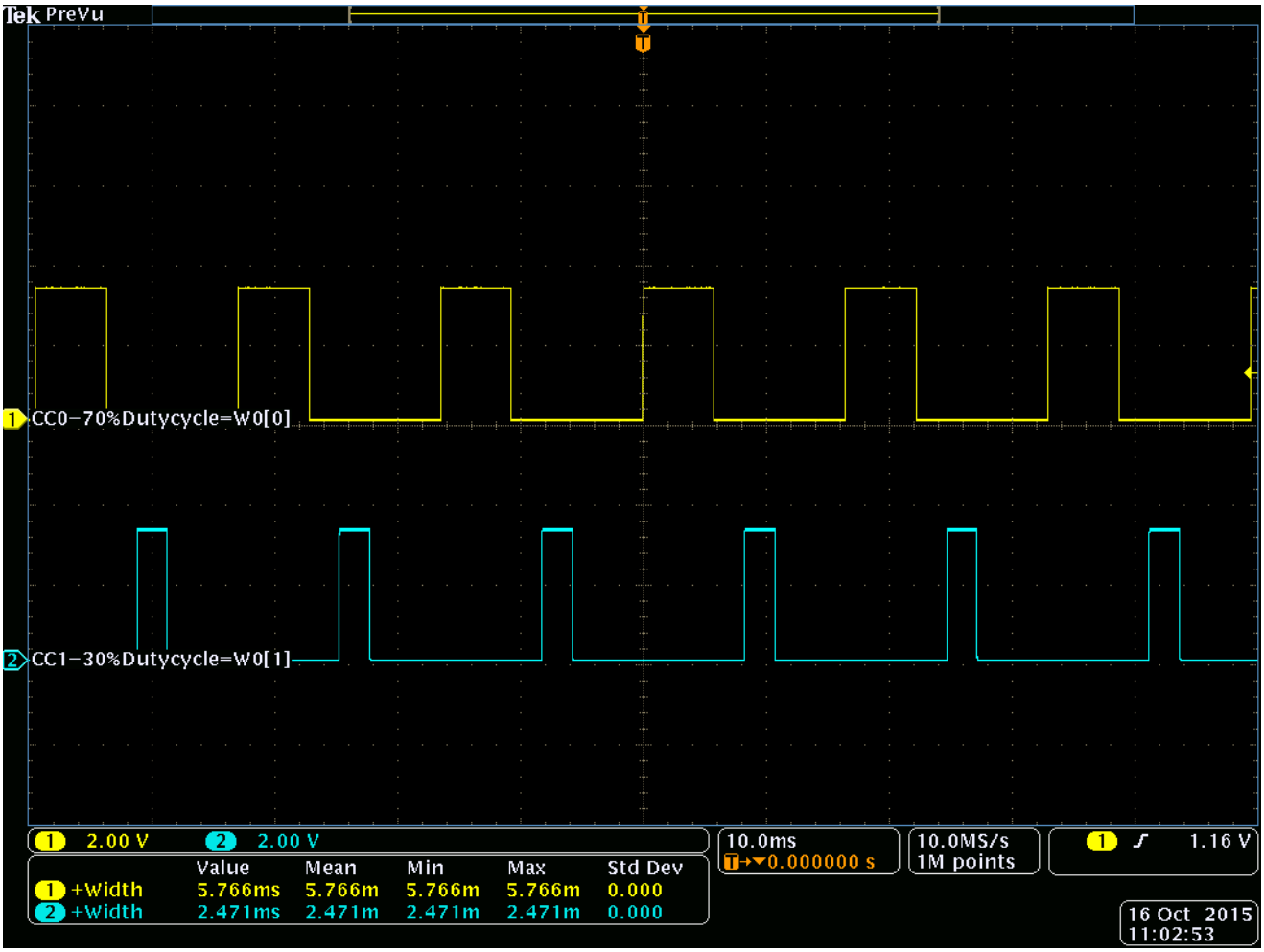
```
/* Configure the TCC Waveform Output pins for waveform generation output
        */
        config_tcc.pins.enable_wave_out_pin[0] = true;
        config_tcc.pins.enable_wave_out_pin[1] = true;
        config_tcc.pins.wave_out_pin[0] = PIN_PB18F_TCC0_WO0;
        config_tcc.pins.wave_out_pin[1] = PIN_PA09E_TCC0_WO1;/* Configure the
Alternate function of GPIO pins for TCC
        functionality */

        config_tcc.pins.wave_out_pin_mux[0] = MUX_PB18F _TCC0_WO0;
        config_tcc.pins.wave_out_pin_mux[1] = MUX_PA09E _TCC0_WO1;
      /* Configure the RAMP mode operation as RAMP2 mode */

        config_tcc.compare.wave_ramp = TCC_RAMP_RAMP2;
      /* Configure the compare channel values for the duty cycle
        control and load the 0xB333 value for 70% duty cycle */

        config_tcc.compare.match[0] = 0xB333;/* Load the 0x4CCC value for 30% duty
cycle */

        config_tcc.compare.match[1] = 0x4CCC;
```

5.7.3. RAMP2 Waveform Output

The following output scope snapshot is captured from the SAM L22 Xplained Pro of EXT3 connector.

- PB18 – Waveform Output 0
- PA09 – Waveform Output 1

Figure 5-14. RAMP2 Operation



The TCC clock frequency = 16MHz

The TCC Clock divider = 1

Time Period for 1 Count = 1/16000000 = 125ns

For the CC0 value 0xB333 = 0xFFFF-0xB333 * 125ns

= 5.7343ms

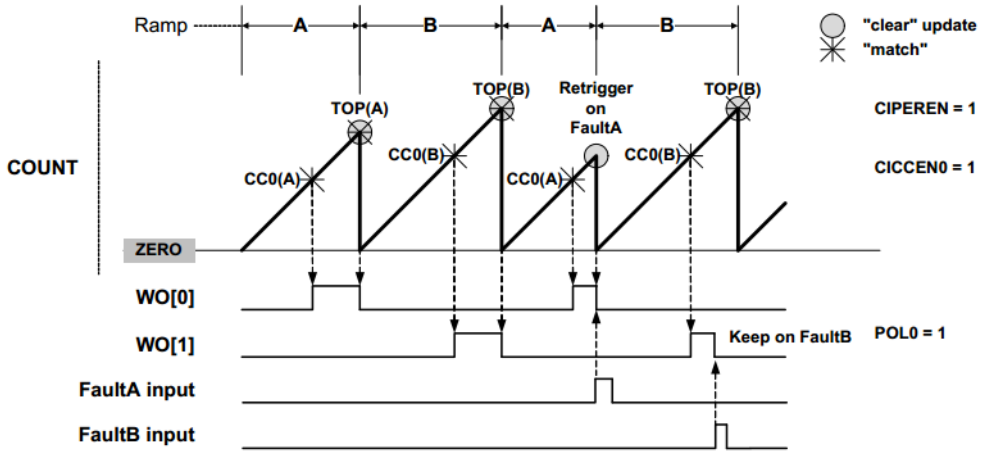
For the CC1 value 0x4CCC = 0X4CCC * 125ns

= 2.4575ms

5.8. Ramp2A Operation

RAMP2 Alternate operation is similar to RAMP2 except that the CC0 register controls both WO [0] and WO [1] compare outputs. For RAMP2A operation mode, the circular buffer mode allows having two dedicated period and compare values for each of the cycle A/B. This is similar to RAMP2 mode, with the difference that only one channel is used for waveform generation and the second channel can be used for capture operation.

Figure 5-15. RAMP2A Operation



5.8.1. RAMP2A Configuration

The RAMP2A feature has been enabled through `#define TCC_MODE_RAMP2A` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this mode, the Waveform outputs WO[0] and WO[1] for the single slope PWM output signals are enabled. Configure the Compare match channel value for the Channel 0 only. RAMP2A mode can be configured in the WAVE register.

5.8.2. Code Snippet

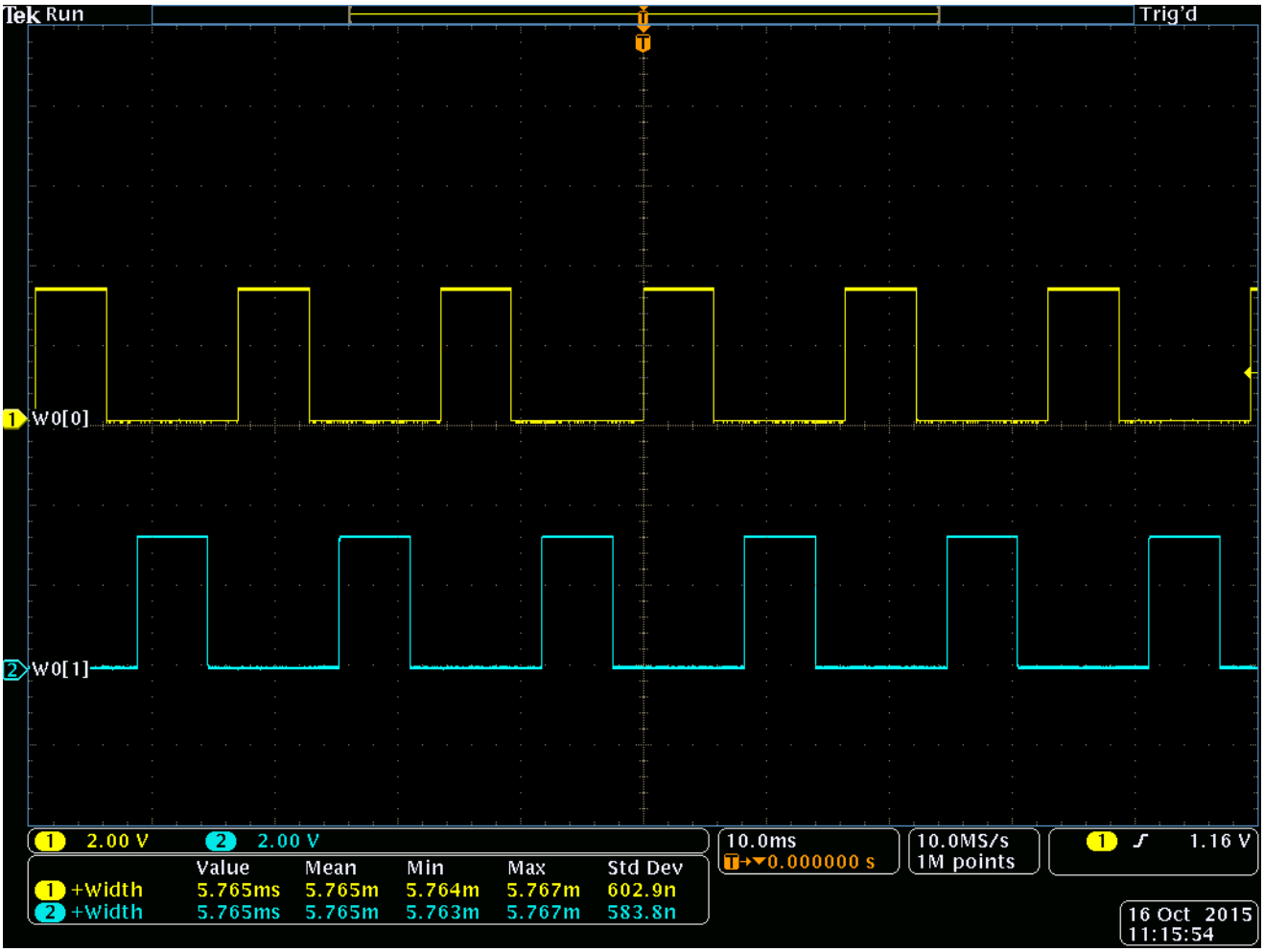
```
/* Configure the TCC Waveform Output pins for waveform generation output
*/            config_tcc.pins.enable_wave_out_pin[0] = true;
                config_tcc.pins.enable_wave_out_pin[1] = true;
                config_tcc.pins.wave_out_pin[0] = PIN_PB18F _TCC0_WO0;
                config_tcc.pins.wave_out_pin[1] = PIN_PA09E _TCC0_WO1;
        /* Configure the Alternate function of GPIO pins for TCC functionality */
                config_tcc.pins.wave_out_pin_mux[0] = MUX_PB18F _TCC0_WO0;
                config_tcc.pins.wave_out_pin_mux[1] = MUX_PA09E _TCC0_WO1;
        /* Configure the RAMP mode operation as RAMP2 mode */
                config_tcc.compare.wave_ramp = TCC_RAMP_RAMP2A;
        /* Configure the compare channel values for the duty cycle control and load
the 0xB333 value
      for 70% duty cycle */
                config_tcc.compare.match[0] = 0xB333;
```

5.8.3. RAMP2A Waveform Output

The following output scope snapshot is captured from the SAM L22 Xplained Pro of EXT3 connector.

- PB18 – Waveform Output 0
- PA09 – Waveform Output 1

Figure 5-16. RAMP2A Operation Output



Output showcasing both waveform outputs have alternate ON/OFF cycles between them with pulse width always corresponds to CC0 value in both cycles.

The TCC clock frequency = 8MHz

The TCC Clock divider = 1

Time Period for 1 Count = 1/8000000 = 125ns

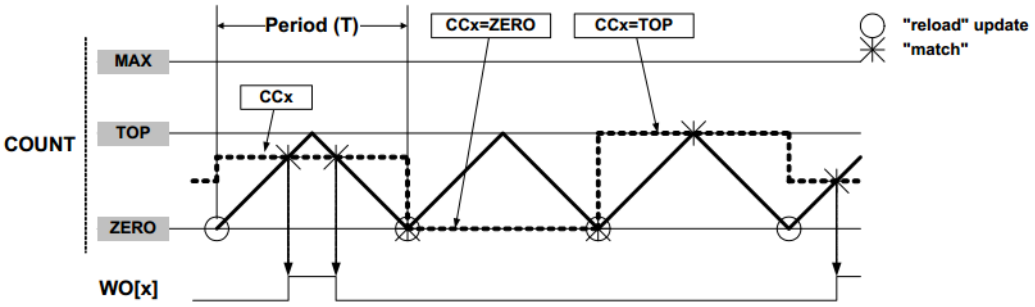
For the CC0 value 0xB333 = 0xB333 * 125ns

= 5.7343ms

5.9. Dual Slope PWM Operations

For dual-slope PWM generation, the period (TOP) is controlled by PER, while CCx control the duty cycle of the generated waveform output. Following figure shows how the counter repeatedly counts from ZERO (BOTTOM) to PER and then from PER to ZERO. The waveform generator output is set on compare match when up-counting, and cleared on compare match when down-counting.

Figure 5-17. Dual-Slope Pulse Width Modulation



Using dual-slope PWM results in a lower maximum operation frequency compared to single-slope PWM operation. The Period register (PER) defines the PWM resolution. The PWM can be configured for one of the following Dual Slope configurations.

TCC_WAVE_GENERATION_DOUBLE_SLOPE_CRITICAL:

Double-slope (count up and down), non-center-aligned: Top is the PER register, CC[x] controls duty cycle while counting up and CC[x+N/2] controls it while counting down.

TCC_WAVE_GENERATION_DOUBLE_SLOPE_BOTTOM:

Double-slope (count up and down), interrupt/event at Bottom .Top is the PER register, output becomes active when count is greater than CCx.

TCC_WAVE_GENERATION_DOUBLE_SLOPE_BOTH:

Double-slope (count up and down), interrupt/event at Bottom and Top. Top is the PER register, output becomes active when count is lower than CCx.

TCC_WAVE_GENERATION_DOUBLE_SLOPE_TOP:

Double-slope (count up and down), interrupt/event at Top . Top is the PER register, output becomes active when count is greater than CCx.

5.9.1. Dual Slope Configuration

The Dual Slope feature has been enabled through `#define TCC_MODE_DUAL_SLOPE` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this mode, the Waveform outputs for the dual slope PWM output signals are enabled. Configure the Compare match channel values for the Compare Channel 0. In this mode, it is observed that the waveform output shows twice the pulse width compared to single slope PWM. This uses `TCC_WAVE_GENERATION_DOUBLE_SLOPE_BOTH` configuration explained above.

5.9.2. Code Snippet

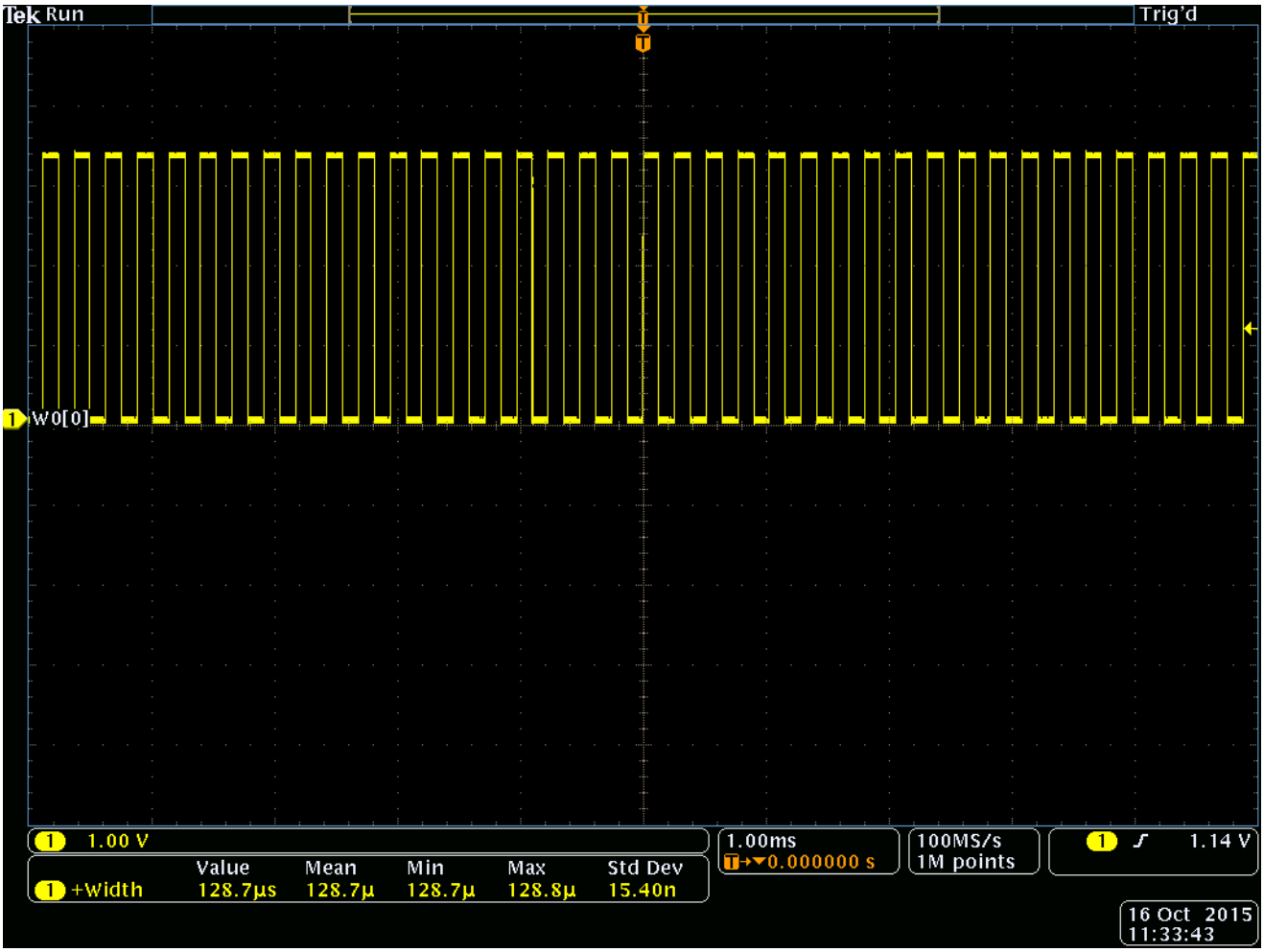
```
config_tcc.compare.wave_generation = TCC_WAVE_GENERATION_DOUBLE_SLOPE_TOP;
        /* Configure the TCC clock source and its divider value */
        config_tcc.counter.clock_source = GLCK_SOURCE;
        config_tcc.counter.clock_prescaler = TCC_CLOCK_DIVIDER;

        /* Configure the value for TOP value */
        config_tcc.counter.period = TCC_PERIOD_VALUE;
```

5.9.3. Dual Slope Waveform Output

The following output scope snapshot is captured from the SAM L22 Xplained Pro of EXT3 connector.

PB18 – Waveform Output

Figure 5-18. Dual-Slope Pulse Width Modulation Output



Output showcasing waveform outputs have twice the pulse width size because of Dual slope advantage.

The TCC clock frequency = 8MHz

The TCC Clock divider = 1

Time Period for 1 Count = 1/8000000 = 125ns

ON TIME (0x3FF-0x1FF)*2 = 2*512 * 125ns

= 0.128ms

5.10. Counter Operations

The counter can be set to count up or down. When the counter is counting up and the top value is reached, the counter will wrap around to zero on the next clock cycle. When counting down, the counter will wrap around to the top value when zero is reached.

5.10.1. Counter Configuration

The Counter feature has been enabled through `#define TCC_MODE_COUNTER` and `#undef` the rest of the TCC features in the `conf_example.h`. In this mode, the led is toggled at four different intervals, based on different initial startup value of the compare channel.

5.10.2. Code Snippet

```
/* Configure different channels with different compare match values */
             config_tcc.compare.match[0] =  900;
             config_tcc.compare.match[1] =  930;
             config_tcc.compare.match[2] = 1100;
             config_tcc.compare.match[3] = 1250;    //!
        [setup_register_callback]
        tcc_register_callback(&tcc_instance,
tcc_callback_to_toggle_led,TCC_CALLBACK_OVERFLOW);
        tcc_register_callback(&tcc_instance,
tcc_callback_to_toggle_led,TCC_CALLBACK_CHANNEL_0);
        tcc_register_callback(&tcc_instance,
tcc_callback_to_toggle_led,TCC_CALLBACK_CHANNEL_1);
        tcc_register_callback(&tcc_instance,
tcc_callback_to_toggle_led,TCC_CALLBACK_CHANNEL_2);
        tcc_register_callback(&tcc_instance,
tcc_callback_to_toggle_led,TCC_CALLBACK_CHANNEL_3);
       //! [setup_register_callback]
       //! [setup_enable_callback]
        tcc_enable_callback(&tcc_instance, TCC_CALLBACK_OVERFLOW);
        tcc_enable_callback(&tcc_instance, TCC_CALLBACK_CHANNEL_0);
        tcc_enable_callback(&tcc_instance, TCC_CALLBACK_CHANNEL_1);
        tcc_enable_callback(&tcc_instance, TCC_CALLBACK_CHANNEL_2);
        tcc_enable_callback(&tcc_instance, TCC_CALLBACK_CHANNEL_3);
        //! [setup_change_events_faults]
```

5.10.3. Output

The LED toggles at various speeds based on four different counter channels.

5.11. Fault Operations

Recoverable faults can restart or halt the timer/counter. Two faults, called Fault A and Fault B, can trigger recoverable fault actions on compare channels CC0 and CC1 from the timer/counter. The compare channels outputs can be clamped to inactive state as long as the fault condition is present, or from the first valid fault condition detection and until the end of the timer/counter cycle. In case of Non Recoverable fault, the fault can drive the output to a pre-defined output level.

5.11.1. Fault Configuration

The Fault configuration feature has been enabled through `#define TCC_MODE_FAULT` and `#undef` the rest of the TCC features in the `conf_example.h`. In this mode, a fault trigger (recoverable fault) by external condition (button press) is identified and the LED is turn ON (trigger fault) and turn OFF the fault /LED in the next button press event. The PA07 pin is connected to the fault line pin PA18, a button press can toggle the output port of PA07 to toggle fault lines, based on which LED is on if fault is ON, otherwise LED is OFF.

5.11.2. Code Snippet I – To Configure TCC module for Fault Configuration

```
void configure_tcc(void)
        {
        /* Structure used to store the TCC configuration parameters */
        struct tcc_config config_tcc;

        /* Fill the Structure with the default values */
        tcc_get_config_defaults(&config_tcc, CONF_PWM_MODULE);
        config_tcc.compare.match[TCC_MATCH_CAPTURE_CHANNEL_0] =
DEFAULT_MATCH_COMPARE;
        config_tcc.counter.period = DEFAULT_PERIOD;

        /* Configure the single slope PWM waveform generation for waveform output
*/
```

```
        config_tcc.compare.wave_generation =
TCC_WAVE_GENERATION_SINGLE_SLOPE_PWM;

        /* Configure the TCC clock source and its divider value */
         config_tcc.counter.clock_source = GLCK_SOURCE;
        config_tcc.counter.clock_prescaler = TCC_CLOCK_DIVIDER;

        /* Configure the TCC Waveform Output pins for waveform generation output
*/
        config_tcc.pins.enable_wave_out_pin[0] = true;
        config_tcc.pins.wave_out_pin[0] = PIN_PB18F_TCC0_WO0;

        /* Configure the Alternate function of GPIO pins for TCC functionality
*/
        config_tcc.pins.wave_out_pin_mux[0] = MUX_PB18F_TCC0_WO0;
        config_tcc.wave_ext.recoverable_fault[TCC_MATCH_CAPTURE_CHANNEL_0].source =
TCC_FAULT_SOURCE_ENABLE;

config_tcc.wave_ext.recoverable_fault[TCC_MATCH_CAPTURE_CHANNEL_0].halt_action =
TCC_FAULT_HALT_ACTION_SW_HALT;

        /* Initialize the TCC0 channel and define the its registers with
configuration defined in the config_tcc */
        stat = tcc_init(&tcc_instance, TCC0, &config_tcc);
        //! [setup_events]
         struct tcc_events events;
         memset(&events, 0, sizeof(struct tcc_events));
          //! [setup_change_events_faults]
        events.on_event_perform_channel_action[0] = true;
        //! [setup_events_enable]
        tcc_enable_events(&tcc_instance, &events);
        /* Enable the TCC module */
       tcc_enable(&tcc_instance);
        }
```

5.11.3. Code Snippet II - To Configure EIC module and Event System for Fault Configuration

```
//! [callback_eic]
        static void eic_callback_to_clear_halt_fault(void)
        {        port_pin_set_output_level(CONF_TEST_PIN_OUT, true);
        }
        //! [callback_eic]

        //! [config_eic]
        static void configure_eic(void)
{
       //! [eic_chan_setup]
        //! [eic_setup_1]
        struct extint_chan_conf config;
        extint_chan_get_config_defaults(&config);
        config.filter_input_signal = true;
        config.detection_criteria  = EXTINT_DETECT_BOTH;
        config.gpio_pin = CONF_FAULT_EIC_PIN;
        config.gpio_pin_mux = CONF_FAULT_EIC_PIN_MUX;
       extint_chan_set_config(CONF_FAULT_EIC_LINE, &config);
       //! [eic_setup_4]
       struct extint_events events;
       memset(&events, 0, sizeof(struct extint_events));
       events.generate_event_on_detect[CONF_FAULT_EIC_LINE] = true;
       //! [eic_event_setup_2]
        extint_enable_events(&events);

extint_register_callback(eic_callback_to_clear_halt_fault,CONF_FAULT_EIC_LINE,
EXTINT_CALLBACK_TYPE_DETECT);

extint_chan_enable_callback(CONF_FAULT_EIC_LINE,EXTINT_CALLBACK_TYPE_DETECT);}
        //! [config_eic]
        //! [config_event]
```

```
        static void configure_event(void)
{
        //! [event_setup_1]
        struct events_config config;
        events_get_config_defaults(&config);
        config.generator = CONF_EVENT_GENERATOR_ID;
        config.path      = EVENTS_PATH_ASYNCHRONOUS;
        events_allocate(&event_resource, &config);
        events_attach_user(&event_resource, CONF_FAULT_EVENT_USER);
}
```

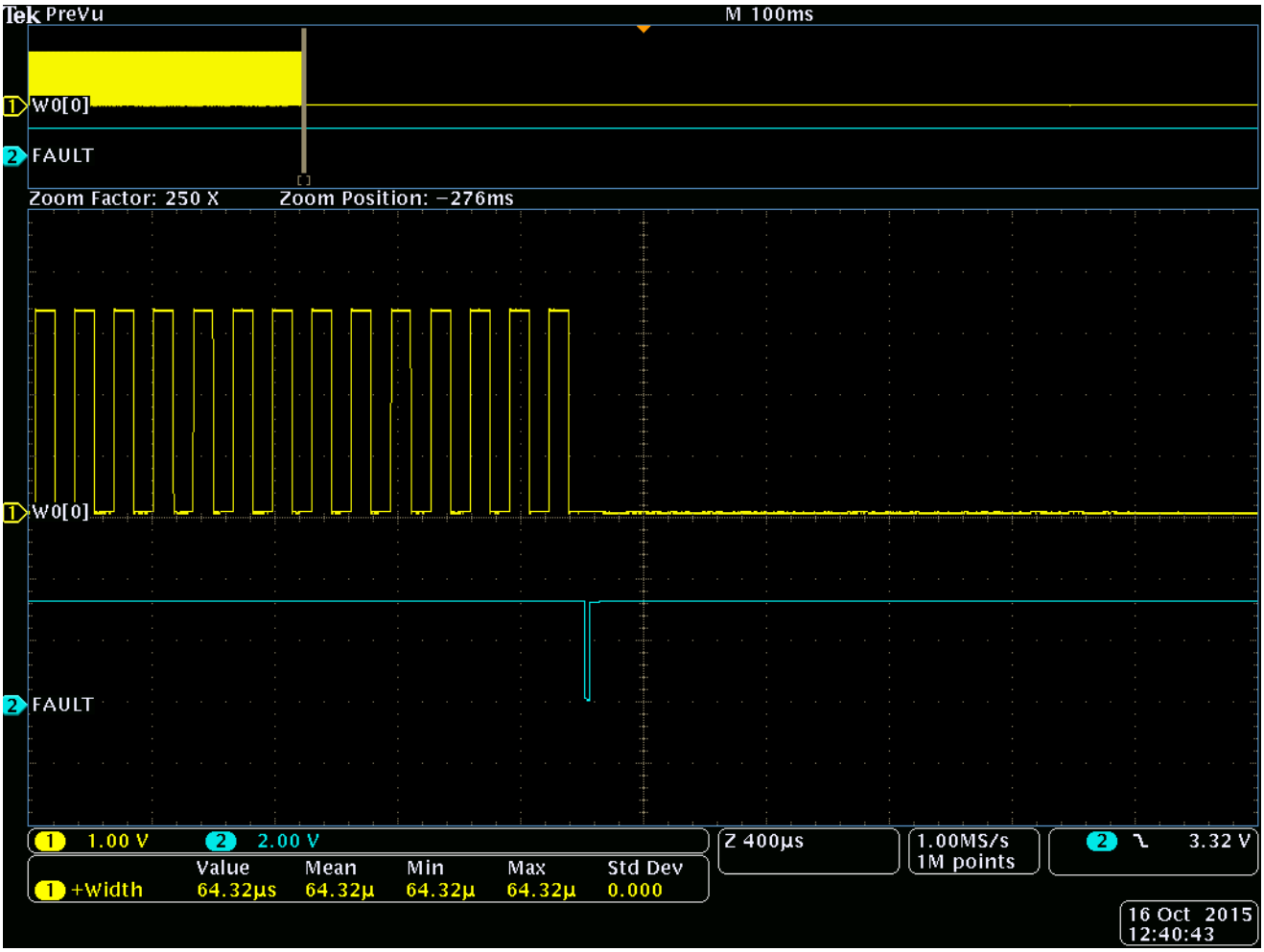
5.11.4. Code Snippet III – Main Application Code to Detect Fault and Turn ON/OFF Fault.

```
        //Inside Main
        uint32_t tcStatus=0;
        unsigned long temp = TCC_STATUS_RECOVERABLE_FAULT_OCCUR(0);
        port_get_config_defaults(&config_pin);

        config_pin.direction = PORT_PIN_DIR_OUTPUT;
        port_pin_set_config(CONF_TEST_PIN_OUT, &config_pin);
        port_pin_set_output_level(CONF_TEST_PIN_OUT, true);
        configure_eic();
        configure_event();
        tcc_clear_status(&tcc_instance,TCC_STATUS_RECOVERABLE_FAULT_OCCUR(0));

        if (!port_pin_get_input_level(SW0_PIN))
               {
               /* Set fault */
               while(!port_pin_get_input_level(SW0_PIN));
               port_pin_set_output_level(CONF_TEST_PIN_OUT, false);
               tcStatus = tcc_get_status(&tcc_instance);
               if(!port_pin_get_output_level(LED_0_PIN))
               {
              // Turn off LED and clearm alarm status..

tcc_clear_status(&tcc_instance,TCC_STATUS_RECOVERABLE_FAULT_OCCUR(0));
               LED_Off(LED_0_PIN);
               }
               else if( (tcStatus & temp) == temp)
               {
               // If alarm set, drive LED.
               LED_On(LED_0_PIN);
               port_pin_set_output_level(CONF_TEST_PIN_OUT, true);
               }
              }
        // end of common fault and capture condition
```

5.11.5. Output

Output showcasing normal PWM. PA18 is the Fault input pin and should be connect to PA07. PA07 is the pin which is set to 0 or 1 to simulate the fault input. When SW0 button is for first time, fault input is generated and output waveform is stopped.When switch is pressed for second time, the fault input clears and the output waveform resumes again on PB18. This behavior continues every time the button is pressed.

Figure 5-19. Fault Output



5.12. DITHERING

The TCC supports dithering on Pulse-width or Period on a 16, 32, or 64 PWM cycles frame.

Dithering consists in adding some extra clocks cycles in a frame of several PWM cycles (16, 32, or 64 depending the configuration). The extra clock cycles are added on some of the compare match signal, one at a time, through a "blue noise" process that minimizes the flickering on the resulting dither patterns.

Dithering makes possible to improve the accuracy of the average output pulses width or period.

Dithering is enabled by writing the corresponding configuration in the CTRLA.RESOLUTION field:

- DITH4 enable dithering every 16 PWM frames
- DITH5 enable dithering every 32 PWM frames
- DITH6 enable dithering every 64 PWM frames

The least significant bits of COUNT, PER, CCx registers define the number of extra cycles to add into the frame (DITHERCY). The remaining bits of COUNT, PER, CCx registers define the compare value itself.

Display hardware, including early computer video adapters and many modern LCDs used in mobile phones and inexpensive digital cameras, show a much smaller color range than more advanced displays. One common application of dithering is to more accurately display graphics containing a greater range of colors than the hardware is capable of showing. For example, dithering might be used in order to display

a photographic image containing millions of colors on video hardware that is only capable of showing 256 colors at a time. The 256 available colors would be used to generate a dithered approximation of the original image. Without dithering, the colors in the original image might simply be "rounded off" to the closest available color, resulting in a new image that is a poor representation of the original. Dithering takes advantage of the human eye's tendency to "mix" two colors in close proximity to one another. It is also used in Lighting control system.

5.12.1. DITHERING Configuration

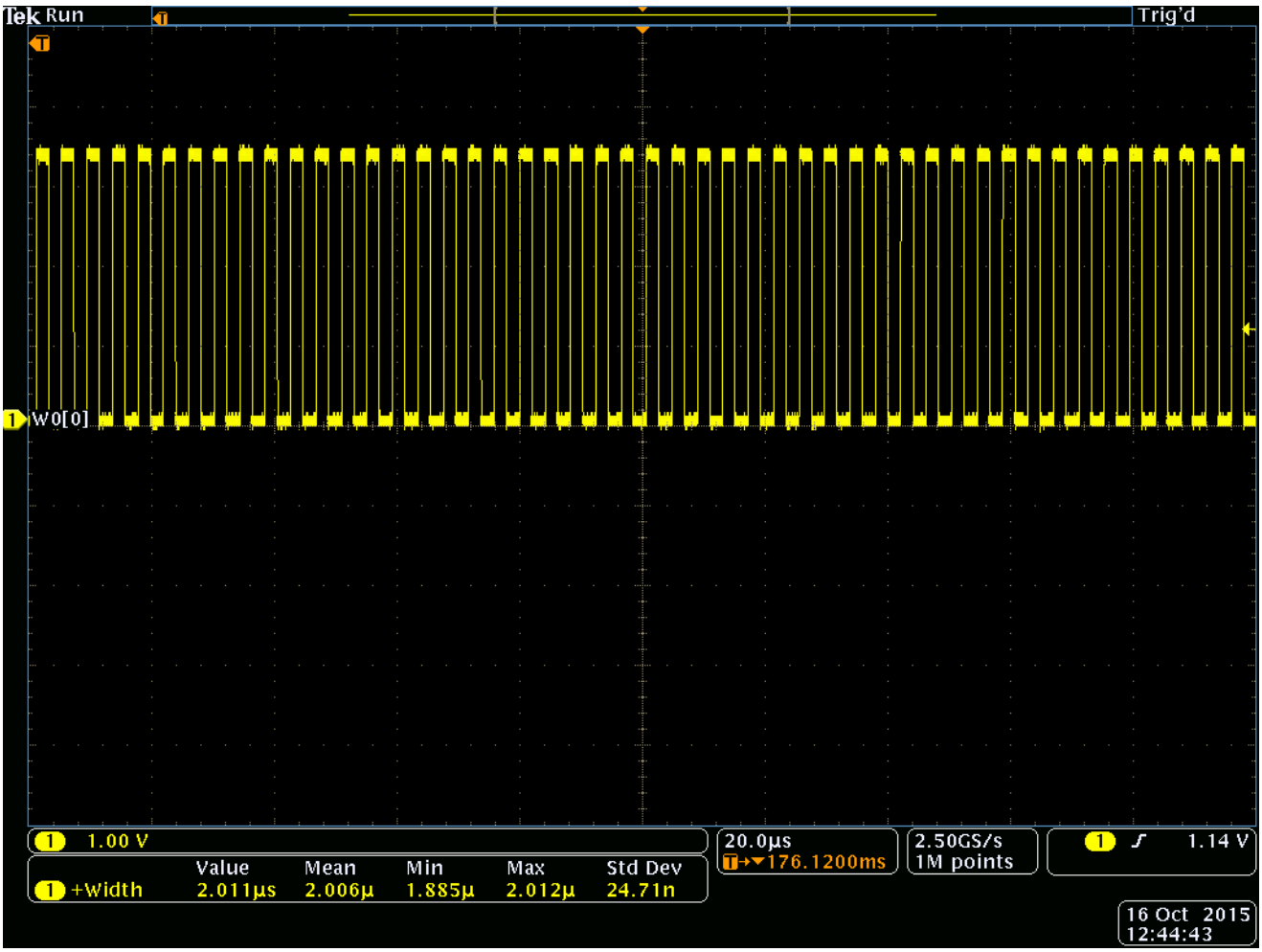
The Dithering feature has been enabled through `#define TCC_MODE_DITHER` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this mode, the Waveform outputs 0 for the single slope PWM output signals (the default configuration) is enabled. The Dithering option with Dithering on every 32 PWM frames is also enabled, since adding only the following configuration is sufficient to enable dithering feature, irrespective of the TCC feature used.

5.12.2. Code Snippet

```
    /* Configure the TCC Waveform Output pins for waveform generation output */
      config_tcc.pins.enable_wave_out_pin[TCC_MATCH_CAPTURE_CHANNEL_0] = true;
      config_tcc.pins.wave_out_pin[TCC_MATCH_CAPTURE_CHANNEL_0] =
PIN_PB18F_TCC0_WO0;

    /* Configure the Alternate function of GPIO pins for TCC functionality
*/
    config_tcc.pins.wave_out_pin_mux[TCC_MATCH_CAPTURE_CHANNEL_0] =
MUX_PB18F_TCC0_WO0;
    TCC0->CTRLA.bit.RESOLUTION = 2;

    while (TCC0->SYNCBUSY.reg & TCC_SYNCBUSY_CTRLB)
    {              /* Wait for sync */
    }
```

5.12.3. Dithering Waveform Output



Information can be verified by capturing the waveform output using laptop/computer. Verify that the number of pulses available for 32 PWM cycles is more, because of DITHERING feature.

5.13. Capture Operations

To enable and use capture operations, the Match or Capture Channel x Event Input Enable (MCEIx) bit must be enabled in the Event Control register (EVCTRL.MCEIx). The capture channels to be used must also be enabled in the Capture Channel x Enable bit in the Control A register (CTRLA.CPTENx) before capture can be performed.

TCC supports Event Capture, Period and Pulse-Width Capture Action (PPW), Capture Operation is triggered based on when event is triggered, in Event capture, the COUNT value is identified on event trigger, whereas in PPW mode, the Period and Pulse width are captured in CC0 and CC1 respectively.

5.13.1. Capture Configuration

The capture feature has been enabled through `#define TCC_MODE_CAPTURE` and `#undef` the rest of the TCC features in the `conf_example.h`.

In this example, the pulse width and period ofthe input signalis captured. TCC0 is configured with event action 1 enabled and the event action is set to PWP (Pulse Width Period)which captures the pulse width of the input signal in CC0 register and period in CC1 register.

The square wave input signal is applied to an EIC channel which is configured to detect and generate event on high level. This event is routed to TCC0 through an event system channel which is configured in asynchronous mode.

5.13.2. Code Snippet I – Configure USART

```
    struct usart_config config_usart;
    usart_get_config_defaults(&config_usart);
    config_usart.baudrate    = 115200;
    config_usart.mux_setting = EDBG_CDC_SERCOM_MUX_SETTING;
    config_usart.pinmux_pad0 = EDBG_CDC_SERCOM_PINMUX_PAD0;
    config_usart.pinmux_pad1 = EDBG_CDC_SERCOM_PINMUX_PAD1;
    config_usart.pinmux_pad2 = EDBG_CDC_SERCOM_PINMUX_PAD2;
    config_usart.pinmux_pad3 = EDBG_CDC_SERCOM_PINMUX_PAD3;
    stdio_serial_init(&usart_instances,EDBG_CDC_MODULE, &config_usart);
    usart_enable(&usart_instances);
```

5.13.3. Code Snippet II – Configures Event and the Main Application for Capture Operation

```
void configu_eic(void)
        {
        struct extint_chan_conf config_extint_chan;
        extint_chan_get_config_defaults(&config_extint_chan);
        config_extint_chan.gpio_pin = PIN_PA07A_EIC_EXTINT7;
        config_extint_chan.gpio_pin_mux = MUX_PA07A_EIC_EXTINT7;
        config_extint_chan.gpio_pin_pull = EXTINT_PULL_NONE;
        config_extint_chan.detection_criteria = EXTINT_DETECT_HIGH;
        extint_chan_set_config(7, &config_extint_chan);
        struct extint_events config_events =
        {
        .generate_event_on_detect[7] = true
        };
        extint_enable_events(&config_events);
        }
void configure_evsys(void)
        {
        struct events_config config;
        events_get_config_defaults(&config);
        config.clock_source = GCLK_GENERATOR_3;
        config.generator = EVSYS_ID_GEN_EIC_EXTINT_7;
        config.path      = EVENTS_PATH_ASYNCHRONOUS;
        config.edge_detect = EVENTS_EDGE_DETECT_BOTH;
        events_allocate(&event_resources, &config);
        events_attach_user(&event_resources, EVSYS_ID_USER_TCC0_EV_1);
        }
```

5.13.4. Code Snippet III – Configure TCC for Capture Operation

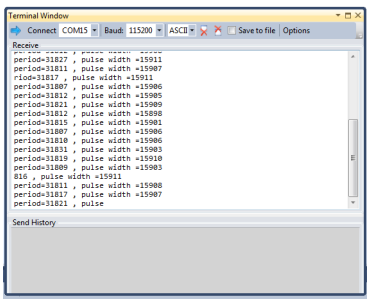
```
void configu_tcc(void)
        {
        struct tcc_config config_tcc;
        tcc_get_config_defaults(&config_tcc, TCC0);
        config_tcc.counter.clock_source = GCLK_GENERATOR_3;
        config_tcc.counter.clock_prescaler = TCC_CLOCK_PRESCALER_DIV1;
        config_tcc.compare.channel_function[0] =
TCC_CHANNEL_FUNCTION_CAPTURE;
        config_tcc.compare.channel_function[1] =
TCC_CHANNEL_FUNCTION_CAPTURE;
        config_tcc.compare.channel_function[2] =
TCC_CHANNEL_FUNCTION_CAPTURE;
        config_tcc.compare.channel_function[3] =
TCC_CHANNEL_FUNCTION_CAPTURE;
```

```
        config_tcc.double_buffering_enabled = false;
        tcc_init(&tcc_instances, TCC0, &config_tcc);
        struct tcc_events events_tcc =
        {
        .input_config[0].modify_action = false,
        .input_config[1].modify_action = true,
        .on_input_event_perform_action[1] = true,
        .input_config[1].action =
TCC_EVENT1_ACTION_PULSE_WIDTH_PERIOD_CAPTURE,
        };
        tcc_enable_events(&tcc_instances, &events_tcc);
        tcc_enable(&tcc_instances);
        }

#ifdef TCC_MODE_CAPTURE
configure_usart();
configu_eic();
configure_evsys();
while (1)
{
while(!(TCC0->INTFLAG.bit.MC1));
TCC0->INTFLAG.bit.MC1 = 1;
period = tcc_get_capture_value(&tcc_instances, 1);
pulse_width = tcc_get_capture_value(&tcc_instances, 0);
printf("period=%ld , pulse width =%ld \r\n", period , pulse_width);
}
#endif
```

5.13.5. Output

The output is displayed using EDBG console. Open the EDBG serial console (**Terminal window** of Atmel Studio) with the following setting 115200 Baud Rate, 8-bits, No Parity, 1 Stop Bit and Flow Control is set as None.

Figure 5-20. Input Capture Output



Output showing the Period (CC1) and Pulse width (CC0) of the input signal in UART Terminal.

6. Revision History

Doc. Rev.	Date	Comments
42625A	02/2016	Initial document release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**