GTM-IP Application note

AN023 – TIM Timeout detection functionality

Table of contents

1 Introduction

This application note describes the TIM timeout detection functionality as implemented in GTM-IP generation 3 and 4 devices. It utilizes the TIM Timeout Detection Unit (TDU). The TDU can be used to detect timeout in different use case scenarios. These use cases are described in the following subsections.

1.1 Use cases for timeout detection

1.1.1 Timeout detection for individual edge types

Figure 1.1 shows the timeout detection for an input signal when measured from rising to rising edge. The counter TO_CNT is started with a rising edge and is reset when the next rising edge is detected. The TO_CNT stops, when it reaches the timeout value TOV. This application scenario is covered in subsection 3.2.1 in more detail.
The GTM-IP timeout detection functionality offers the possibility to configure, if the timeout detection unit remains stopped, when the timeout condition is detected or if the timeout detection is restarted on the next valid edge. In Figure 1.1 and Figure 1.2 the timeout detection unit remains stopped.

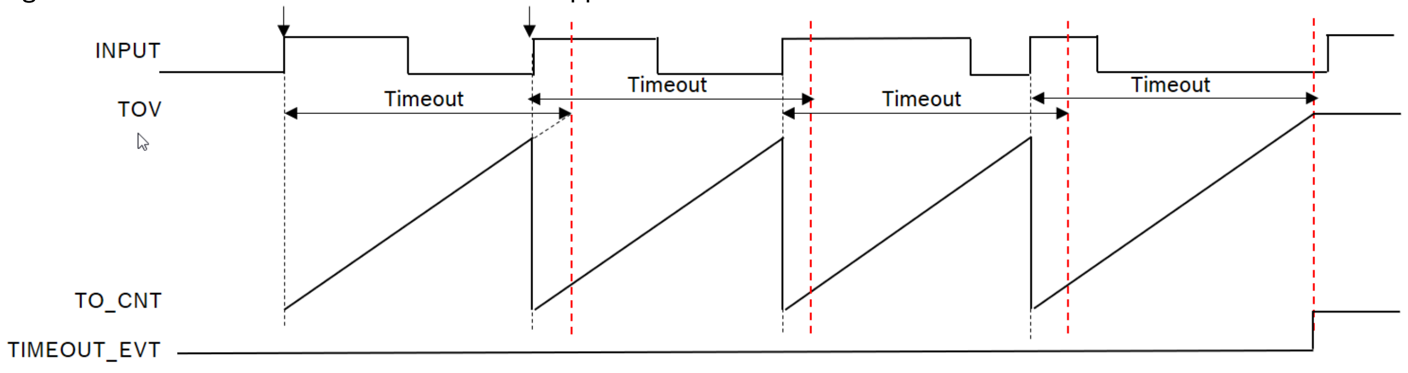

Figure 1.1: Timeout detection for individual edges (rising to rising edge)

Figure 1.2 shows the timeout detection for an input signal when measured from falling to falling edge.

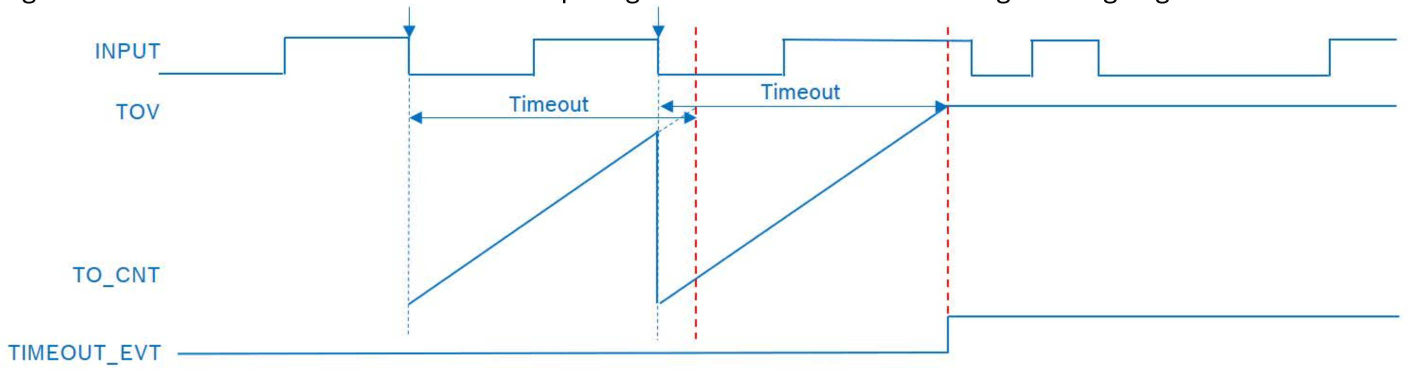

Figure 1.2: Timeout detection for individual edges (falling to falling edge)

Subsection 3.2.2 describes this use case in combination with the usage of a prescaler, to be able to measure very long timeout durations.

the setup of the TIM Timeout Detection Unit for the aforementioned use cases in more detail.

1.1.2 Timeout detection for both edge types

The TIM TDU can also detect timeouts using both edges of an input signal. In principle, when both edges are used, the signal high and low level duration is measured and should not exceed a certain threshold value. Figure 1.3 shows the use case, where the timeout is measured using the same timeout threshold value (TOV) for both edges.

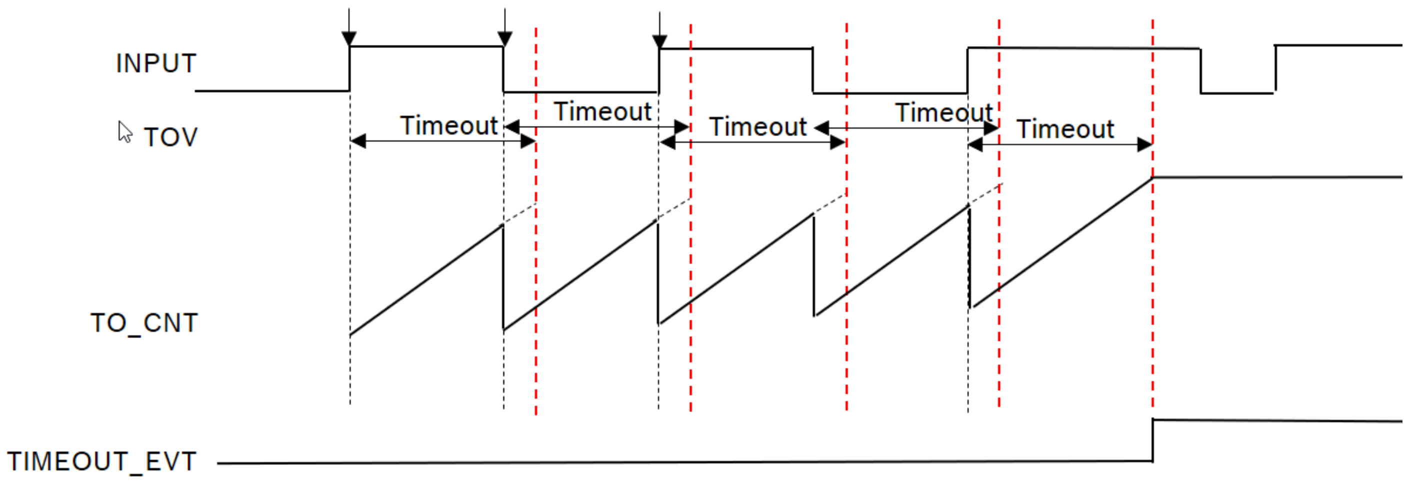

Figure 1.3: Timeout detection for both edges (unique value for TOV)

Section 3.3.1 describes an implementation of the use case shown in Figure 1.3.
Figure 1.4 depicts the timeout detection for both edges and individual timeout threshold values TOV and TOV1.

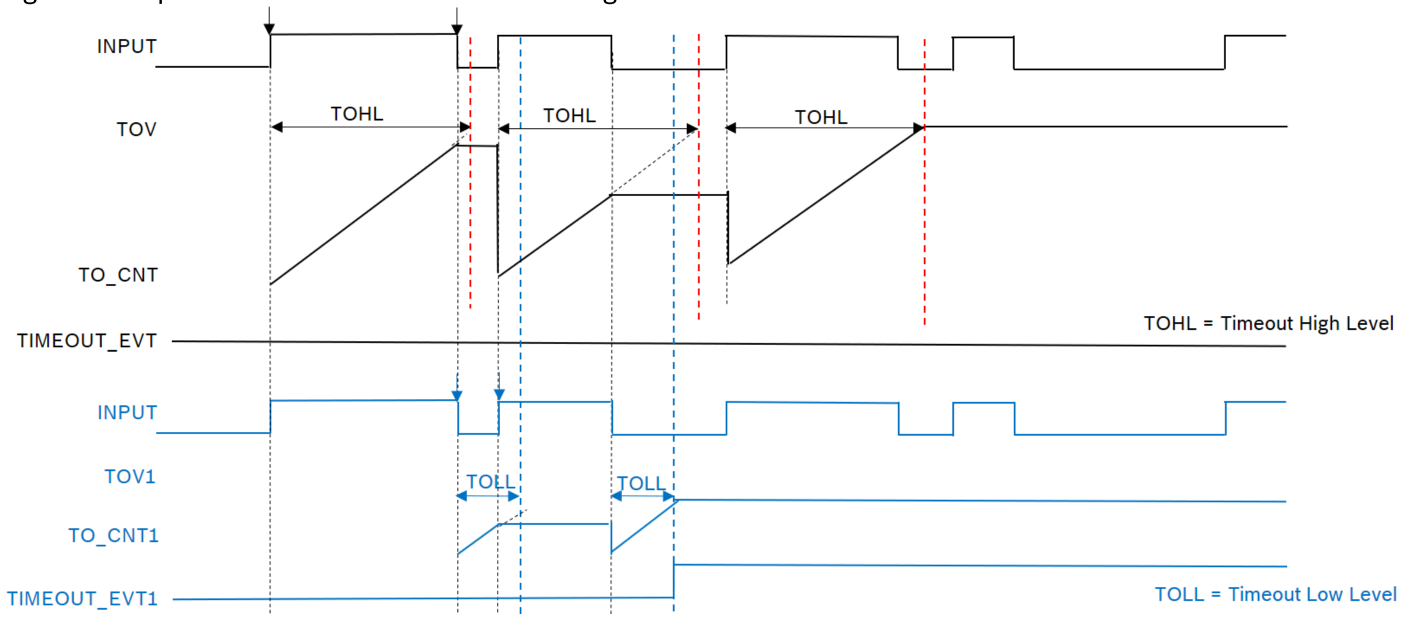

Figure 1.4: Timeout detection for both edges (individual value for TOV and TOV1)

Section 3.3.3 describes how the implementation of such a use case looks like with the TIM module.

1.2 Outlook

Chapter 2 describes the architecture of the TIM TDU in more detail and lists the configuration registers, which are involved when a timeout detection function has to be set up by the user.
Chapter 3 contains implementation examples for the different use cases described above.

2 TIM Timeout detection unit

2.1 Architecture

2.1.1 Overview

The architecture of the TDU is depicted in Figure 2.1. As it can be seen, the TDU is controlled by input signals TDU Enable/Disable, REDGE_DET, FEDGE_DET, EXT_CAPTURE, and clocked with one of the existing CMU_CLKs. The TDU generates the four signals TDU_SAMPLE_EVT, TDU_WORD_EVT, TDU_FRAME_EVT, and TDU_TIMEOUT_EVT. In addition the TDU generates an interrupt signal TO_DET_IRQ. It is configurable, which signal is used to generate the TO_DET_IRQ signal.

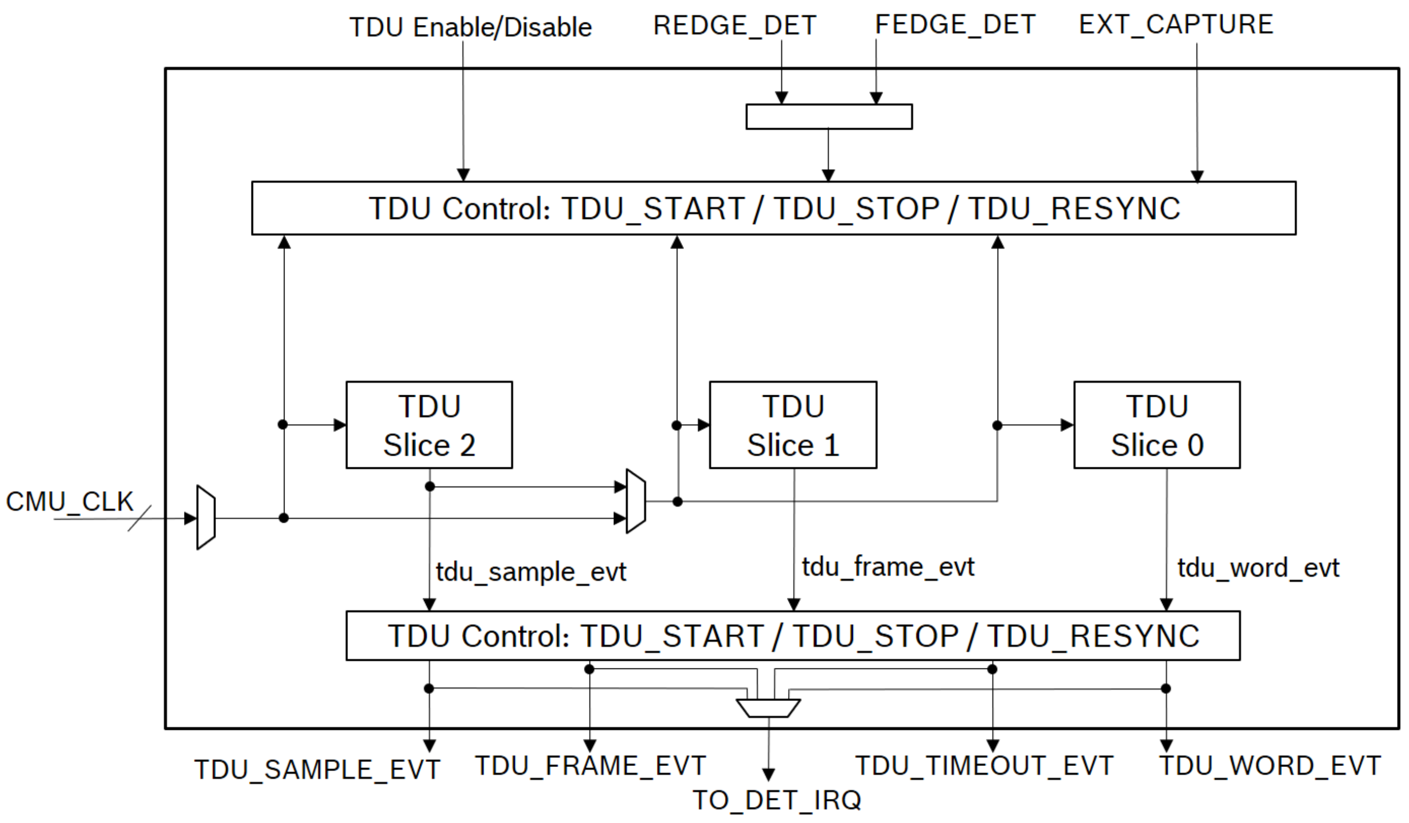

Figure 2.1: TDU architecture

Inside of the TDU, there are three subunits called TDU slices. Each slice has the same amount of resources and architecture. One slice consists of an eight bit counter register TO_CNT, an eight bit counter compare register called TOV and a comparator.
The slices can be cascaded in a flexible manner. Thus, they can form one 24bit counter and compare register or three 8bit counters and compare registers or one 8bit counter and compare register and one 16bit counter and compare register at the same time.
The counters are either driven by a CMU_CLK or by a tdu_sample_evt. The tdu_sample_evt is generated, when the counter of TDU Slice 2 matches or exceeds its individual TOV2 value. When the tdu_sample_evt is used as a clock signal for TDU Slice 0 and 1, the tdu_sample_evt acts as a clock prescaler of the selected CMU_CLK, thus extending the range of timeout measurements. When cascading the Slices 0 and 1, a 16bit timeout value with 8bit clock prescaler is possible. This configuration is discussed in subsection 3.2.2.
Starting, stopping and synchronizing of the counters in the slices can be configured with the three bit fields `tdu_start`, `tdu_stop` and `tdu_resync` of register `TIM[i]_CH[x]_ECTRL`.

2.1.2 TDU_START

The TDU_START configuration is used to start/restart the counters on certain events. Possible events, which start the TDU counters are:

- TDU enable
- 1st CMU_CLK
- First active edge
- External captured event

2.1.3 TDU_STOP

The TDU_STOP configuration is used to stop the counters of the TDU. There is one configuration, where the two counters of slice 0 and 1 can be stopped individually, while the other counter keeps running. The TDU_STOP events are:

- TDU disable
- TDU word event
- TDU frame event
- TDU timeout event
- External capture event

Dependent on the TDU_START configuration, the TDU restarts when the next configured start event occurs, after the TDU is stopped with a TDU_STOP event.

2.1.4 TDU_RESYNC

The TDU_RESYNC feature allows to reset the three different counters in the slices individually or synchronously. The configuration possibilities are shown in the following tables. Please note, that the TDU_RESYNC field is shared and therefore controlling all three counters in parallel.

	TDU_RESYNC					TO_CNT		
SLICING 0b						00	01	10
	0b	0	0	0	0	active edge	active edge	when TO_CTRL = 0b-1: rising input edge
	0b	0	-	-	1	active edge	active edge	when TO_CTRL = 0b-1: rising input edge
	0b	0	-	1	-	tdu_word_evt		
	0b	0	1	-	-		tdu_frame_evt	
	0b	1	0	0	0	EXT_CAP_SRC		
	0b	1	-	-	-			
	0b	1	-	-	1	active edge	active edge	when TO_CTRL = 0b-1: rising input edge
	0b	1	-	1	-	tdu_word_evt		
	0b	1	1	-	-		tdu_frame_evt	

Table 2.1: Resynchronization of TO_CNT

	TDU_RESYNC					TO_CNT1		
SLICING 0b						00	01	10
	0b	0	0	0	0	active edge	active edge	when TO_CTRL = 0b1-: falling input edge
	0b	0	-	-	1	active edge	active edge	when TO_CTRL = 0b1-: falling input edge
	0b	0	-	1	-			
	0b	0	1	-	-	tdu_frame_evt		
	0b	1	0	0	0	EXT_CAP_SRC		
	0b	1	-	-	-			
	0b	1	-	-	1	active edge	active edge	when TO_CTRL = 0b1-: falling input edge
	0b	1	-	1	-			
	0b	1	1	-	-	tdu_frame_evt		

Table 2.2: Resynchronization of TO_CNT1

	TDU_RESYNC					TO_CNT2		
SLICING 0b						00	01	10
	0b	0	0	0	0	active edge or tdu_timeout_evt or TDU enable	active edge or tdu_timeout_evt or TDU enable	active edge or tdu_timeout_evt or TDU enable
	0b	0	-	-	1	active edge		
	0b	0	-	1	-			
	0b	0	1	-	-			
	0b	1	0	0	0	EXT_CAP_SRC		
	0b	1	-	-	-		tdu_sample_evt	
	0b	1	-	-	1	active edge		
	0b	1	-	1	-			
	0b	1	1	-	-			

Table 2.3: Resynchronization of TO_CNT2

2.2 Configuration registers

The behavior of the TDU can be controlled with following TIM channel configuration registers and their bit fields:

Register name	Register field	TIM Timeout functionality configuration
TIM[i]_CH[x]_CTRL	TOCTRL	TDU enable/disable and definition of edge sensitivities
TIM[i]_CH[x]_TDUV	TOVx	Counter compare values (timeout threshold values) for slice x (x:0, 1, 2)
	SLICING	Controls cascading of slices
	USE_SAMPLE_EVT	Configuration of slice counters clock
	SAME_CNT_CLK	Configuration of slice counters clock
	TCS	Configuration of overall used CMU_CLK
TIM[i]_CH[x]_ECTRL	USE_PREV_TDU_IN	Use the previous TDU output signals as input
	TODET_IRQ_SRC	Configures one of the four internal TDU signals (tdu_word_evt, tdu_frame_evt, tdu_sample_evt, tdu_timeout_evt) to serve as TO_DET_IRQ source.
	TDU_START	Start/restart condition for overall TDU
	TDU_STOP	Stop condition for overall TDU
	TDU_RESYNC	Counter resynchronization condition for the three counters
TIM[i]_CH[x]_TDUC	TO_CNTx	Actual counter value of the different slices x (x: 0, 1, 2)

Table 2.4: Registers involved in TIM timeout functionality

3 Timeout detection use cases

3.1 Environmental setup

Table 3.1 shows the environmental setup for the timeout use cases described in this section.

Variable	Configuration
AEI_SYS_CLK	200 MHz
CMU_GCLK	100 MHz
CMU_CLK0	100 MHz
CMU_CLK1	10MHz

Table 3.1: Environmental setup for demonstrating the timeout functionality.

3.2 Timeout detection for individual edge types

3.2.1 Rising to rising edge use case

Figure 3.1 shows the use case for timeout measurement from rising to rising edge.

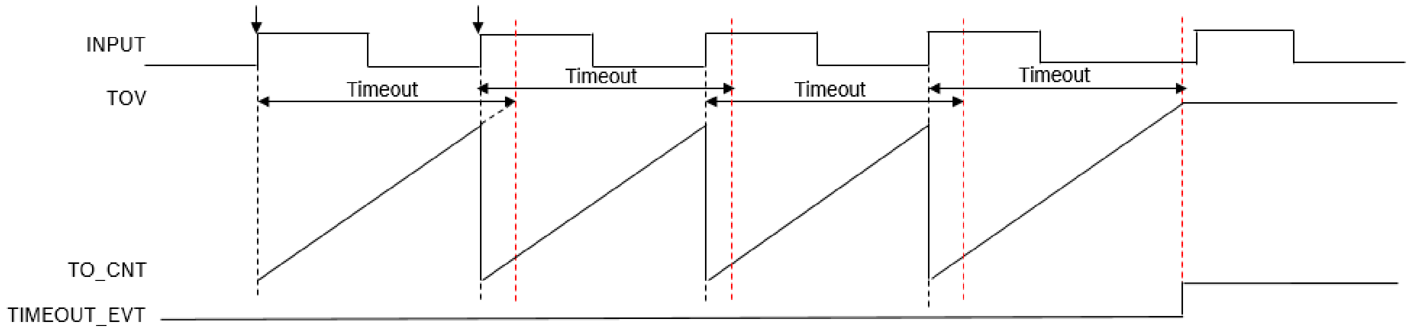

Figure 3.1: Timeout detection for rising to rising edges

Code 1 shows the configuration of TDU registers when a rising to rising edge timeout should be detected.

```
uint32_t tmpRegVal_u32;

/* Configure TIM[0].CH[0].ECTRL */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x3 << GTM_TIM_CH_ECTRL_TDU_STOP_POS)|  // stop on timeout
                 (0x3 << GTM_TIM_CH_ECTRL_TDU_START_POS)| // start act. edge
                 (0x0 << GTM_TIM_CH_ECTRL_TDU_RESYNC_POS);// reset cnt on act.
                                                          // edge
GTM.TIM[0].CH[0].ECTRL = tmpRegVal_u32;

/* Configure GTM.TIM[0].CH[0].TDUV */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (21   << GTM_TIM_CH_TDUV_TOV_POS) |   // timeout @ 2100ns
                 (0x1  << GTM_TIM_CH_TDUV_TCS_POS);    // use CMU_CLK1 = 10MHz
GTM.TIM[0].CH[0].TDUV = tmpRegVal_u32;

/* Enable TDU, sensitive to rising edges and TO_DET_IRQ */
GTM.TIM[0].CH[0].IRQ_EN    = (0x1 << GTM_TIM_CH_IRQ_EN_TODET_IRQ_EN_POS);
GTM.TIM[0].CH[0].CTRL      = 0x40000001;  // Timeout on rising to rising edge
```

Code 1: TDU setup to detect rising to rising edge timeouts

Figure 3.2 shows the simulation result for the detection of a rising to rising edge timeout when configured using Code 1.

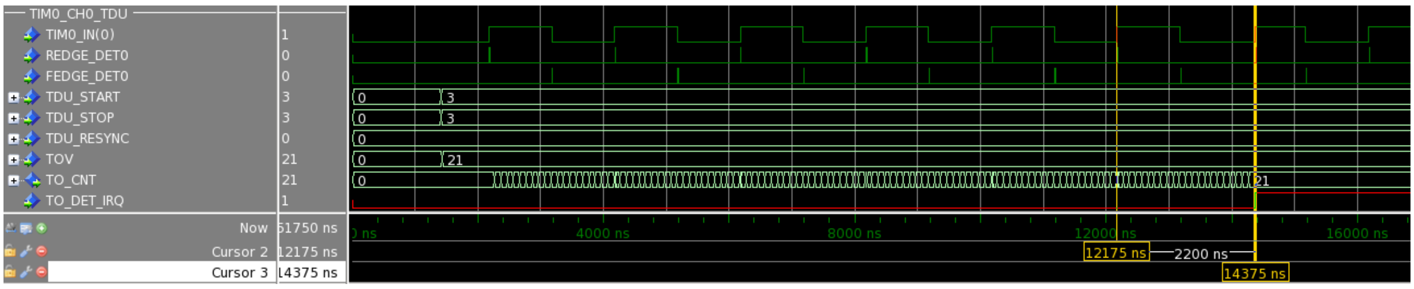

Figure 3.2: Simulation result for timeout on rising to rising edge

3.2.2 Falling to falling edge use case (with prescaler)

For the falling to falling edge use case a timeout detection using a prescaler for the timeout counters should be demonstrated. This configuration can be used, if a combination of the slowest available CMU_CLK frequency of the SW design and a 24bit timeout counter would not be feasible to detect a timeout. Typically, this is a configuration which operates on very long and stable input signals.
The resource allocation in the TDU for a timeout detection with prescaler is shown in Figure 3.3. The TDU slice 2 is used as an 8bit prescaler and the slices 0 and 1 are combined to form a 16bit counter compare value. The slice 2 is configured to use the CMU_CLK as input (bold black clock line) and slices 0 and 1 use the tdu_sample_evt signal as clock input (green clock line). With this configuration, it is important to note, that the reset of the counters for the slices 2, 1 and 0 has to be different. The slice 2 counter should be reset on a tdu_sample_evt, to implement the prescaler, while the combined counter from slice 1 and 0 should be reset on each active edge. This different reset behavior can be configured with the TDU_RESYNC bit field in the register TIM[i]_CH[x]_ECTRL.

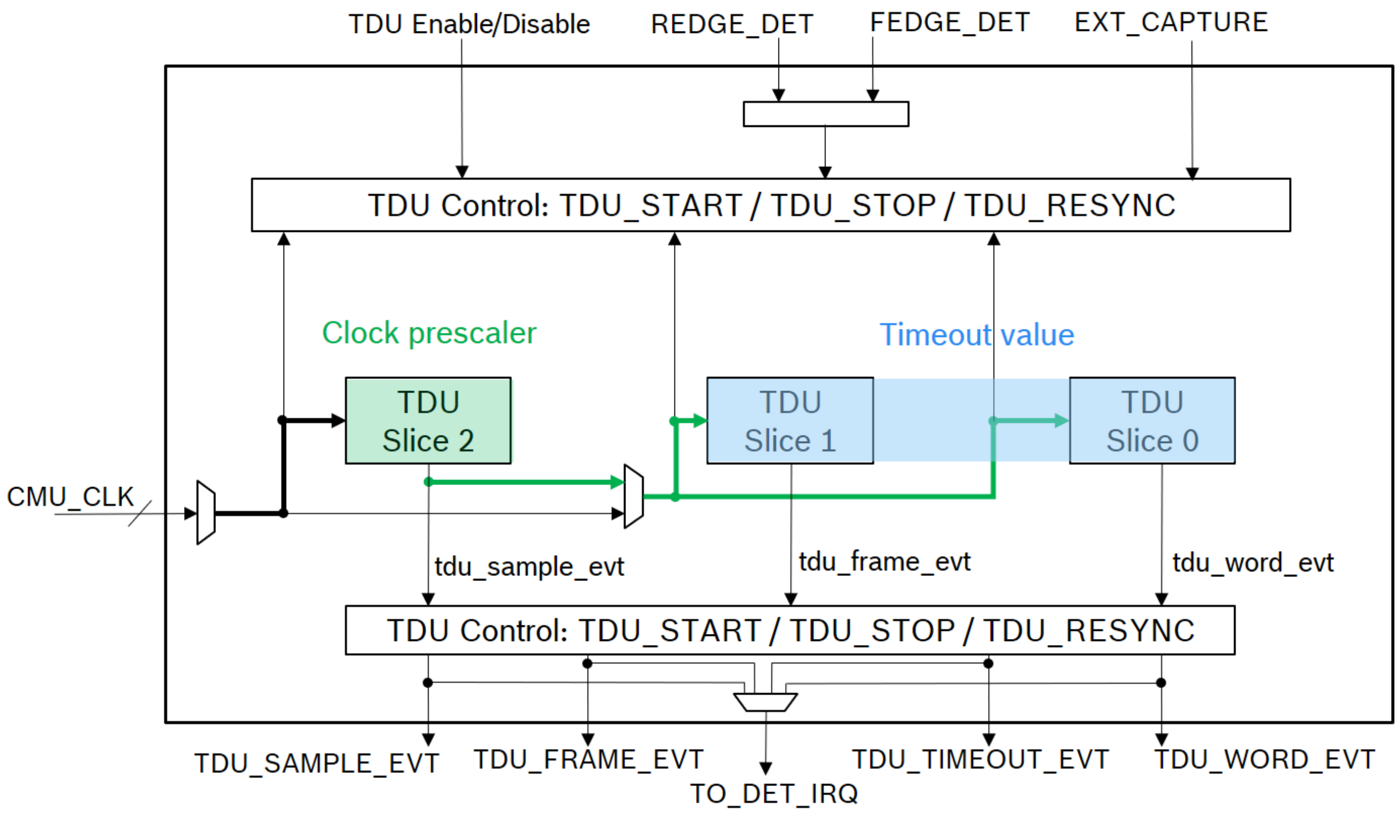

Figure 3.3: TDU resource allocation when using CMU_CLK prescaler for timeout counters

Code 2 shows the implementation of a timeout mechanism for falling to falling edges using an additional clock prescaler mechanism. The corresponding simulation result is depicted in Figure 3.4.

```
uint32_t tmpRegVal_u32;

/* Configure TIM[0].CH[1].ECTRL */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x3 << GTM_TIM_CH_ECTRL_TDU_STOP_POS)  |
                 (0x3 << GTM_TIM_CH_ECTRL_TDU_START_POS) |
                 (0x9 << GTM_TIM_CH_ECTRL_TDU_RESYNC_POS);
GTM.TIM[0].CH[1].ECTRL = tmpRegVal_u32;

/* Configure GTM.TIM[0].CH[1].TDUV */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x0 << GTM_TIM_CH_TDUV_TCS_POS) |      // CMU_CLK0 = 100MHz
                 (0x1 << GTM_TIM_CH_TDUV_SLICING_POS) | // 1x8bit, 1x16bit
                 (0x1 << GTM_TIM_CH_TDUV_TDU_SAME_CNT_CLK_POS) |
                 (0x1 << GTM_TIM_CH_TDUV_TCS_USE_SAMPLE_EVT_POS) |
                 (9 << GTM_TIM_CH_TDUV_TOV2_POS) |       // 10MHz Prescaler
                 (21 << GTM_TIM_CH_TDUV_TOV_POS);        // timeout @ 2100ns
GTM.TIM[0].CH[1].TDUV = tmpRegVal_u32;

/* Enable TDU, sensitive to falling edges and TO_DET_IRQ */
GTM.TIM[0].CH[1].IRQ_EN   = (0x1 << GTM_TIM_CH_IRQ_EN_TODET_IRQ_EN_POS);
GTM.TIM[0].CH[1].CTRL     = 0x80000001;  // Timeout on falling to falling edge
```

Code 2: TDU setup to detect falling to falling edge timeouts using a prescaler mechanism

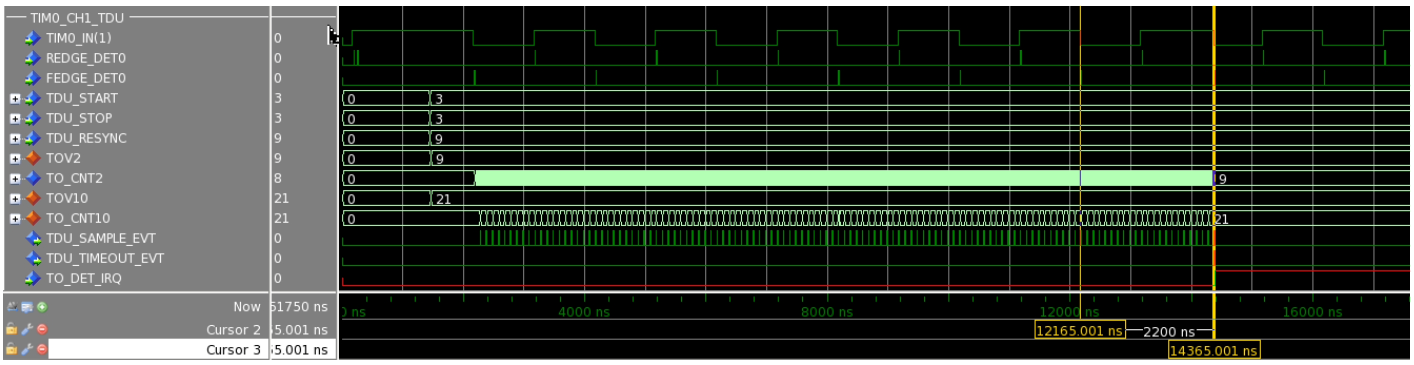

Figure 3.4: Simulation result for timeout on falling to falling edge

3.3 Timeout detection for signal level duration

The TIM TDU subunit offers also the possibility to detect timeouts regarding the duration of the high level as well as the low level at the same time. There are three different use cases for timeout detection on signal levels:

- Timeout detection for both signal levels using one unique timeout threshold value (see subsection 3.3.1)
- Timeout detection on either high or low signal level (see subsection 3.3.2)
- Timeout detection for both signal levels using different timeout threshold values (see subsection 3.3.3)

3.3.1 Timeout detection for both signal levels using one unique timeout threshold value

In Figure 3.5 a timeout measurement is shown for high and low signal levels using the same timeout threshold value for both the signal levels. Since the high phase is too long, the TDU finally signals a timeout event.
When using a unique timeout threshold, the timeout detection is either limited to PWM signals of about 50% duty cycle or the timeout threshold has to be set to the longer signal level duration, since otherwise the TDU would issue a timeout event in case of a regular signal level duration.

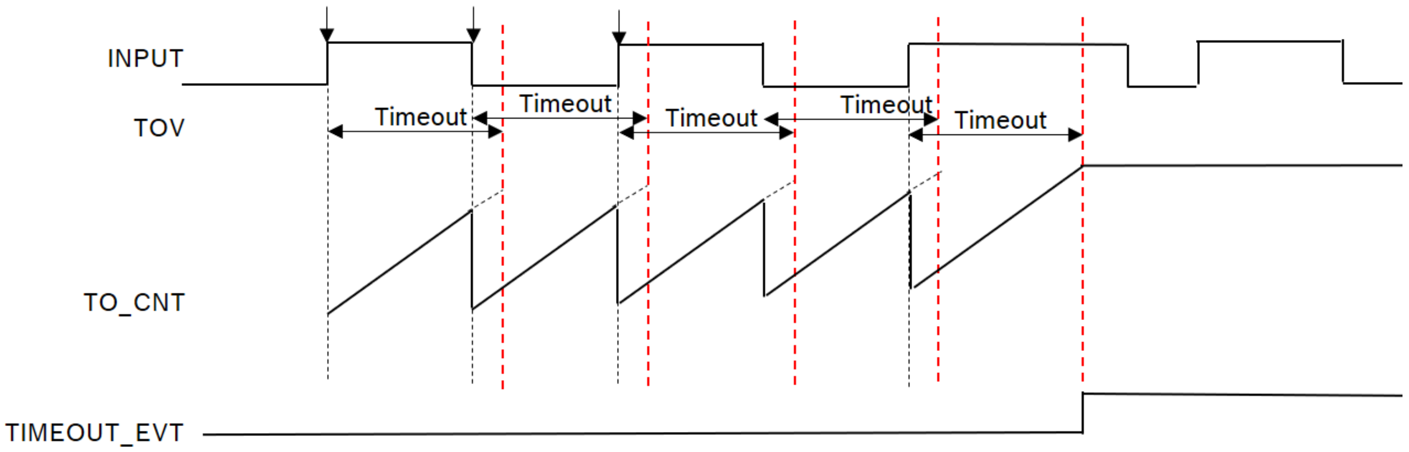

Figure 3.5: Timeout detection on both edges with one unique timeout threshold value (TOV)

Code 3 shows the implementation of the timeout detection functionality for both signal level and one unique timeout threshold value configured in register TIM[i]_CH[x].TDUV as a 24bit value. The corresponding simulation is shown in Figure 3.6.

```
uint32_t tmpRegVal_u32;

/* Configure TIM[2].CH[1].ECTRL */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x3 << GTM_TIM_CH_ECTRL_TDU_STOP_POS)    |
                 (0x3 << GTM_TIM_CH_ECTRL_TDU_START_POS)   |
                 (0x0 << GTM_TIM_CH_ECTRL_TDU_RESYNC_POS);
GTM.TIM[0].CH[2].ECTRL = tmpRegVal_u32;

/* Configure GTM.TIM[2].CH[1].TDUV */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (11   << GTM_TIM_CH_TDUV_TOV_POS) |   // timeout @ 1100ns
                 (0x1  << GTM_TIM_CH_TDUV_TCS_POS);    // CMU_CLK1 = 10MHz
GTM.TIM[0].CH[2].TDUV = tmpRegVal_u32;

/* Enable TDU, sensitive to both edges and TO_DET_IRQ */
GTM.TIM[0].CH[2].IRQ_EN   = (0x1 << GTM_TIM_CH_IRQ_EN_TODET_IRQ_EN_POS);
GTM.TIM[0].CH[2].CTRL = 0xC0000001;  // Timeout for both edges
```

Code 3: High and low signal level timeout detection using one unique timeout threshold

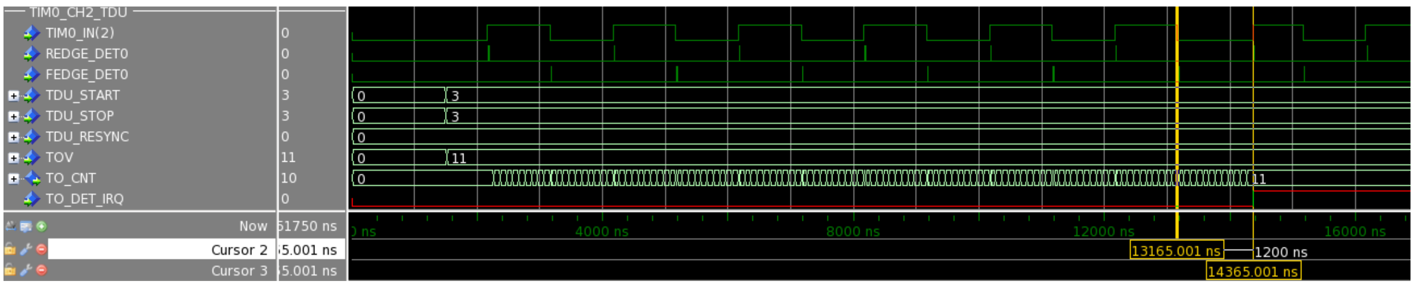

Figure 3.6: Simulation result for timeout detection on signal level duration (unique timeout threshold)

3.3.2 Timeout detection for signal level duration of high phase only

Figure 3.7 shows a scenario, where a timeout should be generated, when the signal high phase exceeds a certain limit while the duration of the low phase doesn't care. This is the situation mentioned above, where the regular low level is far too long and the TDU would raise a timeout for the low level (blue timeline), even though the low level phase is not relevant for the timeout but has a much longer duration than the relevant high level phase.
The TDU has to be configured in a specific manner for this scenario to work, because it should be avoided, that a timeout is signaled, when the low phase exceeds the timeout threshold value. This is for example the case in the application example shown in Figure 3.7.

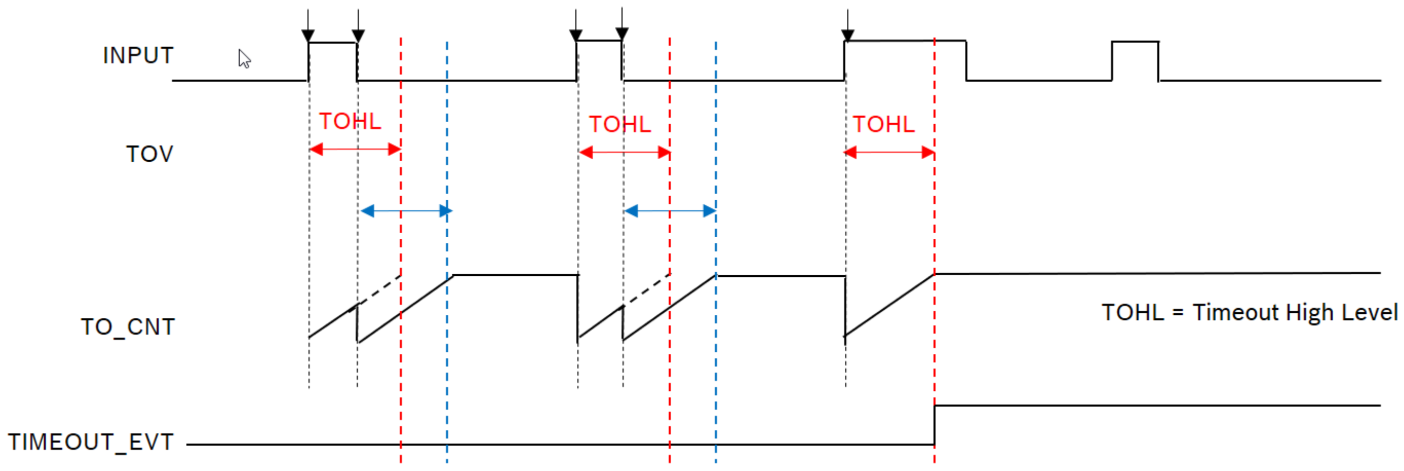

Figure 3.7: Use case timeout detection for high phase signal level

Therefore, the timeout counter has to be started with the edge, that determines the signal level for which the timeout has to be measured, and the timeout counter has to be stopped with the edge, with which the signal level is left. When a timeout for the high signal level should be determined, the starting edge is the rising one and the stopping edge is the falling one.
For such a timeout measurement, two neighboured TIM channels have to be used, where one TIM TDU determines the starting edge and the succeeding TIM channel determines the falling edge. The signal flow and resource usage is shown in Figure 3.8.

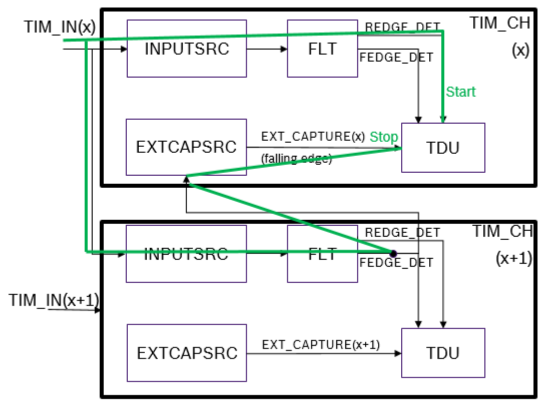

Figure 3.8: Resource allocation of neighboured TIM channels for signal high level timeout measurement

As shown in Figure 3.8 the timeout measurement should be done for signal TIM_IN(x) (signal flow in green). Therefore, the TIM_IN(x) is feed into TIM channel x and via CICTRL=1 bit into TIM channel x+1. The TDU of TIM channel x has to be configured to start the timeout measurement with the rising edge and to stop the measurement with an external capture event EXT_CAPTURE(x). Since stopping of the TDU has to be done with a falling edge, the EXTCAPSRC(x) subunit of TIM channel x has to be configured to route through the falling edge detected with TIM channel x+1 (see green path). The architecture of the subunit EXTCAPSRC(x) is shown in Figure 3.9.

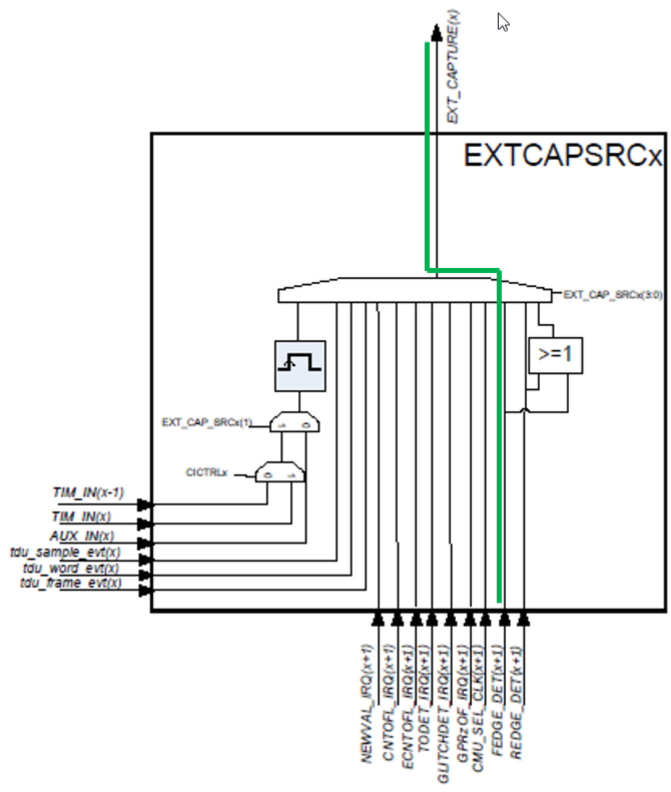

Figure 3.9: EXTCAPSRC configuration for TIM channel x

By using the path FEDGE_DET(x+1), a falling edge event can be signalled to the TIM TDU unit, thus stopping the timeout measurement. To generate the FEDGE_DET(x+1) signal, the TIM channel x+1 has to be used. There, the INPUTSRC subunit has to be configured to feed the TIM_IN(x) signal into the channel. The falling edge will then be detected by the TIM channel x+1 and feed back to TIM channel x via FEDGE_DET(x+1) signal. The corresponding implementation for the two neighboured TIM channels three and four is shown in Code 4.

```
uint32_t tmpRegVal_u32;

/*
 * Configure TIM channel 3
 */
/* Configure ECTRL */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x4 << GTM_TIM_CH_ECTRL_TDU_STOP_POS)   |
                 (0x7 << GTM_TIM_CH_ECTRL_TDU_START_POS)  |
                 (0x9 << GTM_TIM_CH_ECTRL_TDU_RESYNC_POS) |
                 (0xa << GTM_TIM_CH_ECTRL_EXT_CAP_SRC_POS); // use falling
                                                            // edge as ext cap
GTM.TIM[0].CH[3].ECTRL = tmpRegVal_u32;
/* Configure TDUV */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x0 << GTM_TIM_CH_TDUV_TCS_POS) |      // CMU_CLK0 = 100MHz
                 (0x1 << GTM_TIM_CH_TDUV_SLICING_POS) | // 1x8bit, 2x16bit
                 (0x0 << GTM_TIM_CH_TDUV_TDU_SAME_CNT_CLK_POS) |
                 (0x1 << GTM_TIM_CH_TDUV_TCS_USE_SAMPLE_EVT_POS) |
                 (9 << GTM_TIM_CH_TDUV_TOV2_POS) |       // 10MHz Prescaler
                 (6 << GTM_TIM_CH_TDUV_TOV_POS);         // timeout @ 6ns
GTM.TIM[0].CH[3].TDUV = tmpRegVal_u32;
/* Enable TDU, sensitive to rising edge and TO_DET_IRQ */
GTM.TIM[0].CH[3].IRQ_EN   = (0x1 << GTM_TIM_CH_IRQ_EN_TODET_IRQ_EN_POS);
GTM.TIM[0].CH[3].CTRL = 0x40080001;  // EXT_CAP_EN = 1

/*
 * Configure TIM channel 4
 */
GTM.TIM[0].CH[4].CTRL = 0x00000041;     // CICTRL = 1 → use TIM_IN(x-1)
```

Code 4: Signal high level timeout detection using two TIM channels

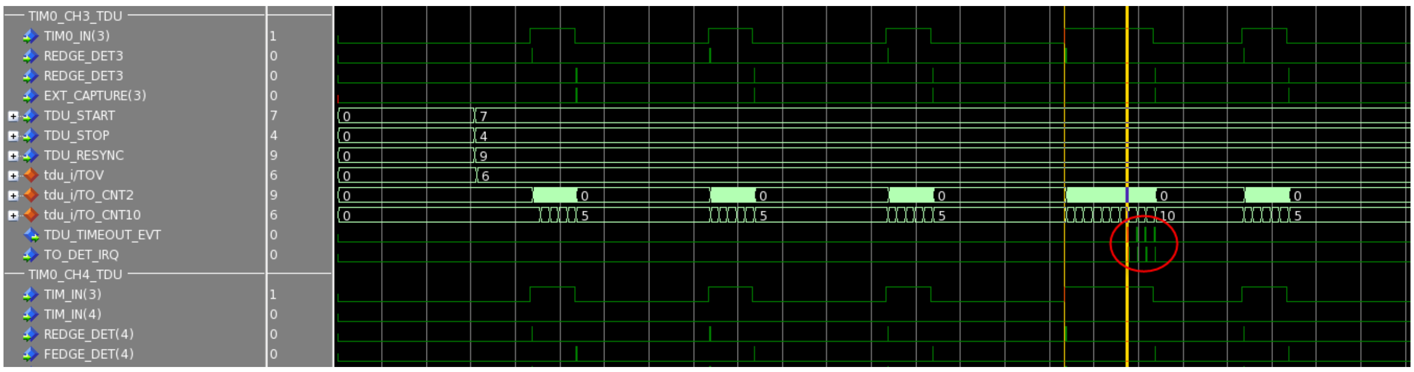

Figure 3.10: Simulation result for signal high level timeout measurement

Figure 3.10 shows the simulation result for signal high level timeout measurement. It is important to note, that TDU_TIMEOUT_EVT is signalled as long as the input signal stays on high level, since the counters are stopped with the external capture event signaling a falling edge. This behaviour leads to multiple TO_DET_IRQs for one detected timeout. Thus, the SW has to stop the TDU by SW and reenable the TDU immediately afterwards to omit subsequent TO_DET_IRQ events. The TDU will then restart the counters with the next active edge. Code 5 shows a code fragment implementing this behaviour. The corresponding simulation result is shown in Figure 3.11.

```
…
GTM.TIM[0].CH[3].CTRL       = 0x00080001; // Stopp TDU
GTM.TIM[0].CH[3].IRQ_NOTIFY = 0x10;       // Clear TO_DET_IRQ
GTM.TIM[0].CH[3].CTRL       = 0x40080001; // Reenable TDU
…
```

Code 5: Avoidance of multiple TO_DET_IRQ events

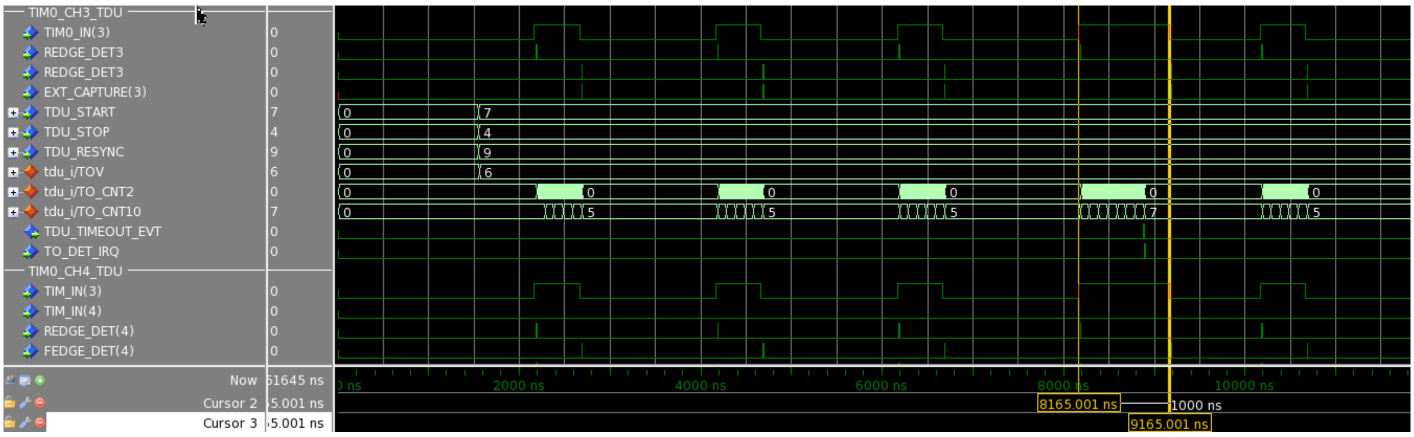

Figure 3.11: Simulation result for signal high level timeout measurement, single TO_DET_IRQ

3.3.3 Timeout detection for both signal levels using individual timeout threshold values

The timeout measurement for both signal levels and with different timeout thresholds is shown in Figure 3.12. The problem is, that the counters inside of the TDU have to be started and stopped individually, dependent on the measured signal level. Since the TOLL value is shorter, than the TOHL value, it has to be ensured, that TO_CNT is not running, while the signal low level has to be measured. Otherwise, the comparator for TO_CNT would issue a timeout event during low level signal phase in any case.

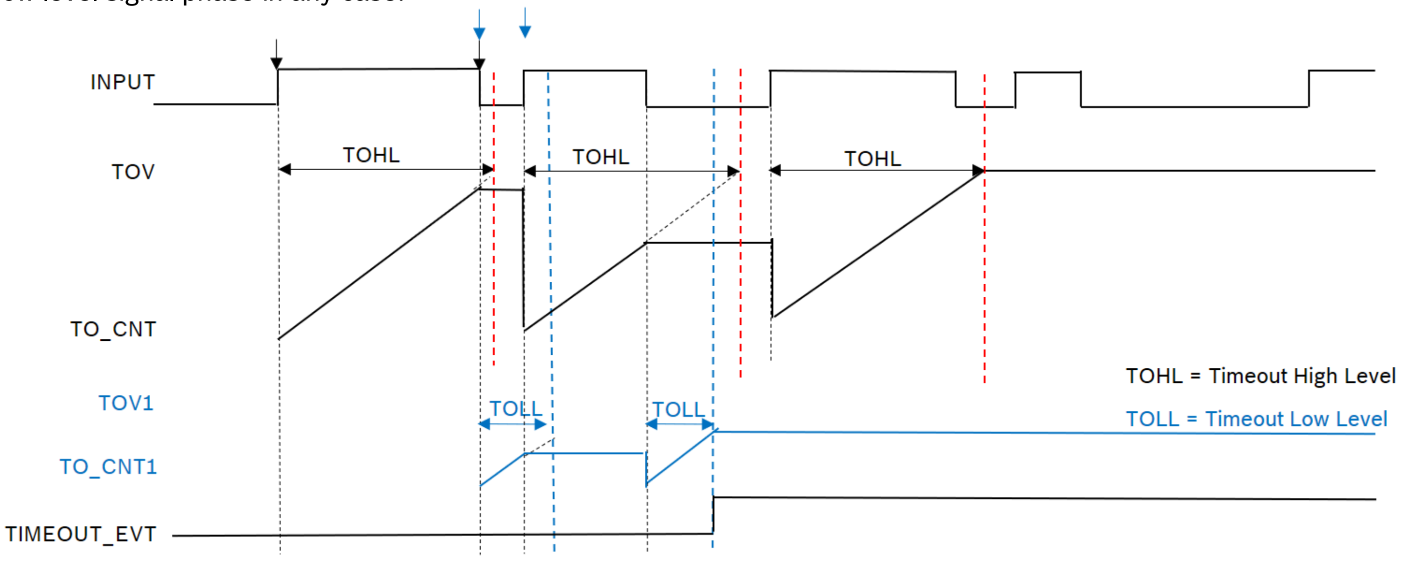

Figure 3.12: Timeout detection for both signal levels using individual timeout threshold values

Figure 3.13 shows the resources of TIM channels and their subunits needed to measure timeout values on two signal levels with different timeout threshold values. The green and blue lines mark the input signal flow for the two measurement units. In the use case shown in the figure, the TIM channel x measures the signal level from rising to falling edge (blue signal flow). The TIM channel x+1 measures the signal level from falling to rising edge (green signal flow). Both TDU subunits use the EXT_CAPTURE signal coming from their EXTCAPSRC subunit to stop the counter on the edge which should end the timeout measurement.

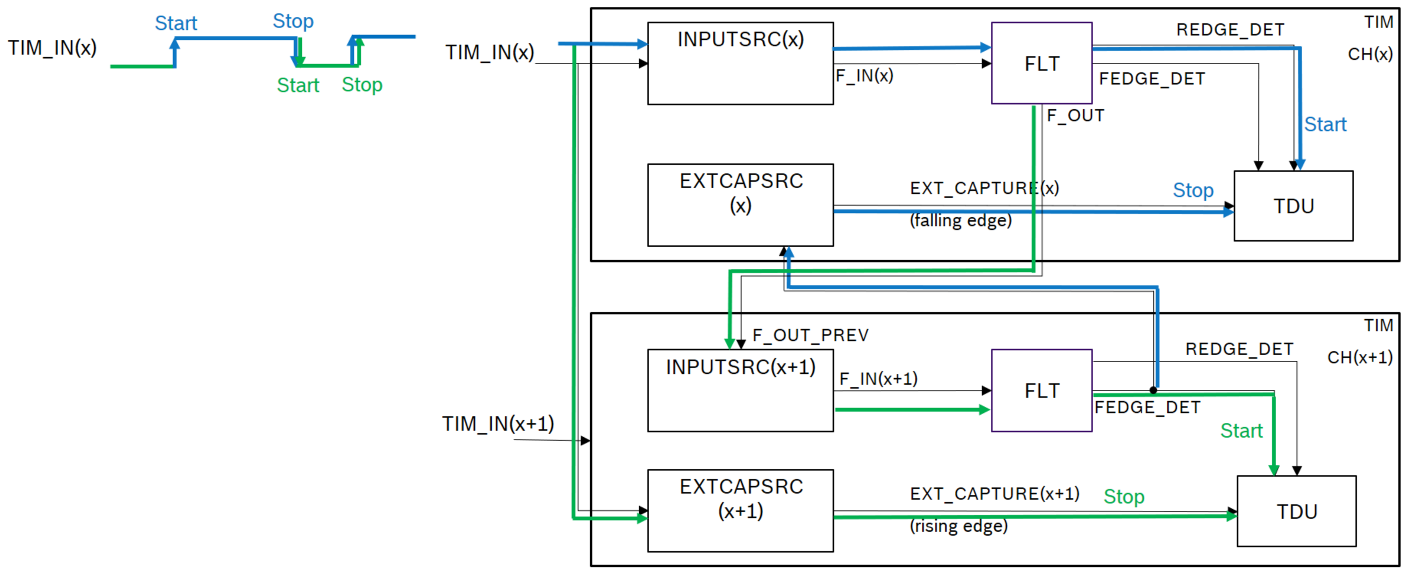

Figure 3.13: Configuration of TIM channels for timeout measurement on both signal levels

3.3.3.1 Signal flow configuration for TIM_CH(x) (high signal level measurement)

For the rising to falling edge timeout measurement (high signal level), the INPUTSRC(x) configuration can stay on default, since the signal which should be measured is provided at the input pin TIM_IN(x) of this channel. The EXTCAPSRC(x) subunit has to be configured to route the signal FEDGE_DET(x+1) through the subunit. The TDU is then stopped with the falling edge. The path through the EXTCAPSRC(x) subunit for TIM channel (x) is shown in Figure 3.14.

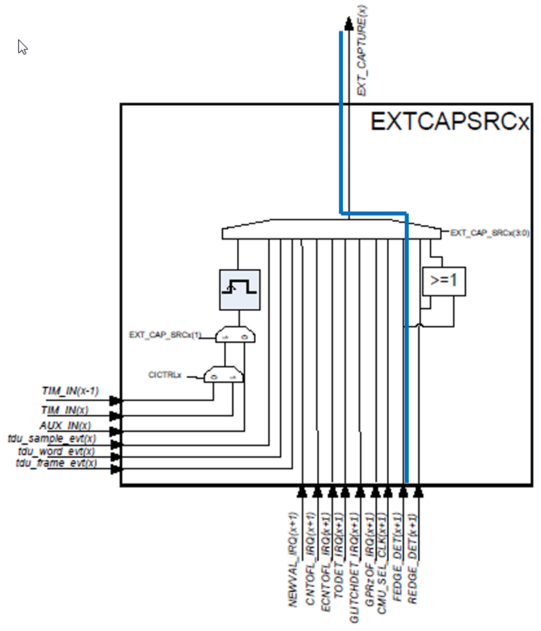

Figure 3.14: Signal path through EXTCAPSRC for falling edge propagation

3.3.3.2 Signal flow configuration for TIM_CH(x+1) (low signal level measurement)

For TIM channel x+1, the rising edge is needed as TDU stop signal, since this channel is used to measure the signal low level timeout. A rising edge event can be generated by the EXTCAPSRC(x+1) subunit, by using the build in rising edge generator. The signal path is shown in Figure 3.15.

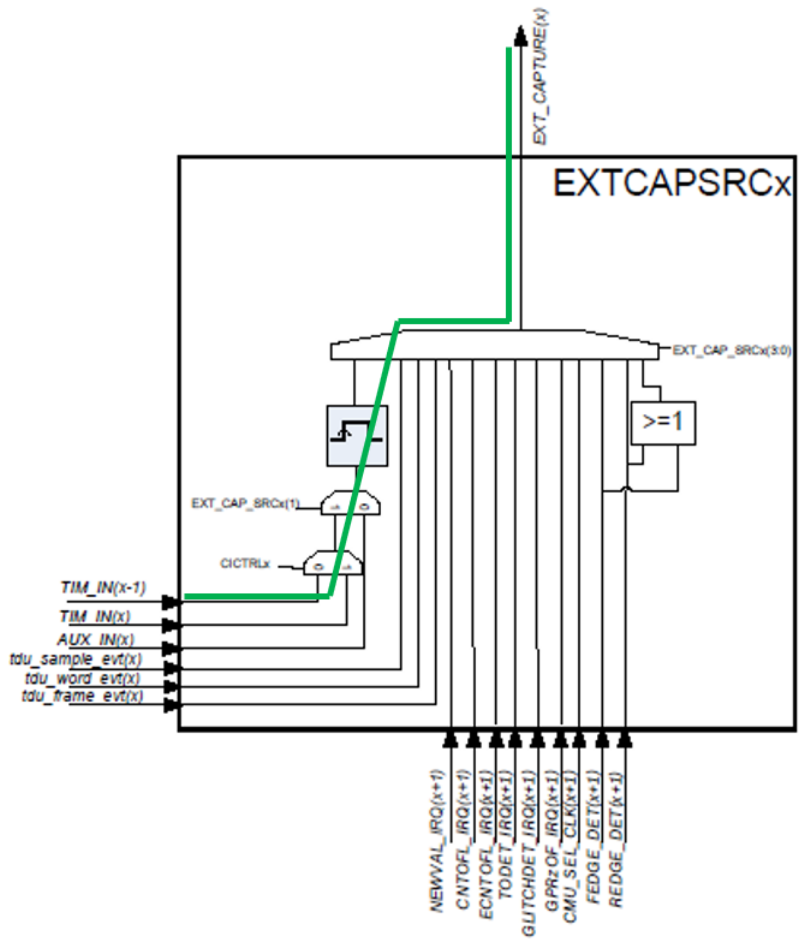

Figure 3.15: Signal path through EXTCAPSRC for rising edge propagation of the channels input

Because of this EXTCAPSRC configuration with CICTRLx=0, the TIM_IN(x) signal is not available on the F_IN(x+1) signal inside TIM channel x+1. This is because CICTRLx=0 multiplexes the TIM_IN(x) signal, which is not the signal that has to be measured (see Figure 3.16, please note: here TIM_IN(x) corresponds to TIM_IN(x+1) in this example). Therefore, the signal F_OUT_PREV is used, which is routed from TIM channel x FLT output to TIM channel x+1 INPUTSRC input.

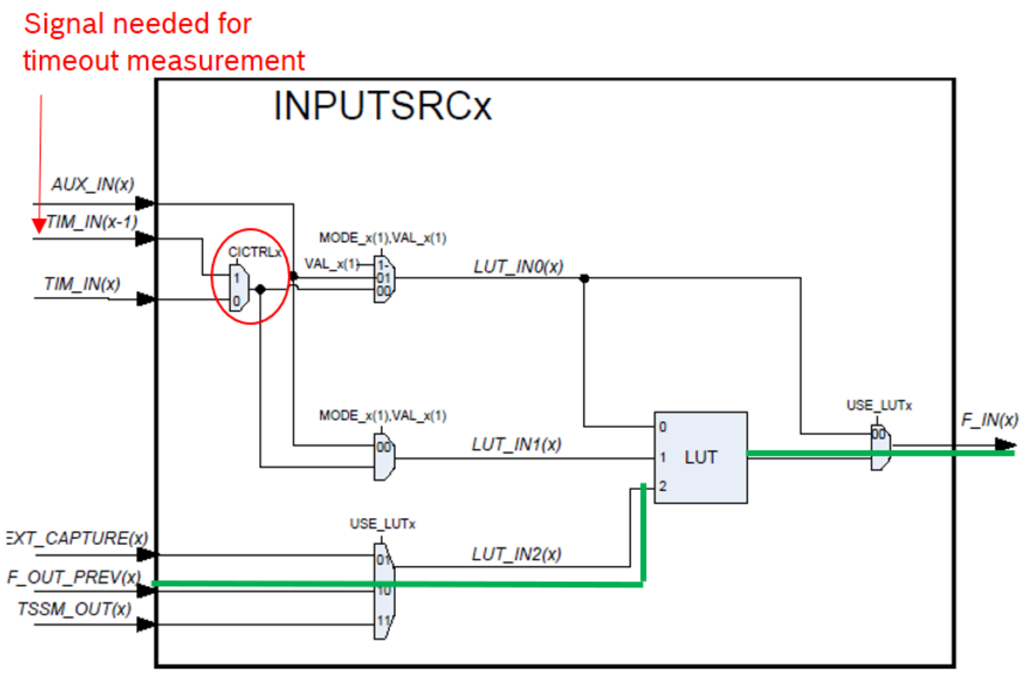

Figure 3.16: INPUTSRC configuration for TIM channel x+1 to route through the TIM channel x input signal

The LUT has to be programmed in a manner, that it routes through the F_OUT_PREV (LUT_IN2) signal. The logical function for LUT is depicted in Table 3.2.

LUT_IN2	LUT_IN1	LUT_IN0	F_IN
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.2: LUT logical function to route through F_OUT_PREV (LUT_IN2)

The overall configuration of the two TIM channels to measure timeouts on high and low signal levels and with two different timeout threshold values is shown in Code 6.

```
GTM.TIM[0].CH[1].ECTRL = tmpRegVal_u32;

/*
 * Configure TIM channel 6
 */
/* Configure ECTRL */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x4 << GTM_TIM_CH_ECTRL_TDU_STOP_POS)  |  // stop on ext cap
       (0x7 << GTM_TIM_CH_ECTRL_TDU_START_POS) |  // start/restart on act edge
       (0x0 << GTM_TIM_CH_ECTRL_TDU_RESYNC_POS) |
       (0xa << GTM_TIM_CH_ECTRL_EXT_CAP_SRC_POS); // use FEDGE_DET(x+1);
GTM.TIM[0].CH[5].ECTRL = tmpRegVal_u32;

/* Configure TDUV */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x0 << GTM_TIM_CH_TDUV_TCS_POS) |     // CMU_CLK0 = 100MHz
                 (0x0 << GTM_TIM_CH_TDUV_SLICING_POS) | // 24bit counter
                 (0x0 << GTM_TIM_CH_TDUV_TDU_SAME_CNT_CLK_POS) |
                 (0x0 << GTM_TIM_CH_TDUV_TCS_USE_SAMPLE_EVT_POS) |
                 (60  << GTM_TIM_CH_TDUV_TOV_POS);      // timeout @ 600ns
GTM.TIM[0].CH[5].TDUV = tmpRegVal_u32;

/* Enable TDU, timeout on rising to falling edge and TO_DET_IRQ */
GTM.TIM[0].CH[5].IRQ_EN = (0x1 << GTM_TIM_CH_IRQ_EN_TODET_IRQ_EN_POS);
GTM.TIM[0].CH[5].CTRL   = 0x40080001;   // CICTRL = 1

/*
 * Configure TIM channel 5
 */
/* Configure ECTRL */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x4 << GTM_TIM_CH_ECTRL_TDU_STOP_POS) |  // stop on ext cap
        (0x7 << GTM_TIM_CH_ECTRL_TDU_START_POS)| // start/restart on act edge
        (0x0 << GTM_TIM_CH_ECTRL_TDU_RESYNC_POS) |
        (0x2 << GTM_TIM_CH_ECTRL_USE_LUT_POS)    |
        (0x3 << GTM_TIM_CH_ECTRL_EXT_CAP_SRC_POS); // rising edge TIM_IN(x-1)
GTM.TIM[0].CH[6].ECTRL = tmpRegVal_u32;

/* Configure TDUV */
tmpRegVal_u32  = 0;
tmpRegVal_u32  = (0x0 << GTM_TIM_CH_TDUV_TCS_POS) |     // CMU_CLK0 = 100MHz
                 (0x1 << GTM_TIM_CH_TDUV_SLICING_POS) | // 2x16bit
                 (0x0 << GTM_TIM_CH_TDUV_TDU_SAME_CNT_CLK_POS) |
                 (0x0 << GTM_TIM_CH_TDUV_TCS_USE_SAMPLE_EVT_POS) |
                 (160 << GTM_TIM_CH_TDUV_TOV_POS);       // timeout @ 1600ns
GTM.TIM[0].CH[6].TDUV = tmpRegVal_u32;

/* Configure INPUTSRC LUT functionality to reflect F_OUT_PREXV */
GTM.TIM[0].CH[6].TDUC  = 0x00F00000;

/* Enable TDU, sensitive to falling to rising edges and TO_DET_IRQ */
GTM.TIM[0].CH[6].IRQ_EN = (0x1 << GTM_TIM_CH_IRQ_EN_TODET_IRQ_EN_POS);
GTM.TIM[0].CH[6].CTRL   = 0x80080001;  // EXT_CAP_EN
```

Code 6: Timeout measurement for high and low signal level using different timeout threshold values

Please note, that due to the fact, that the TO_CNT2 value of TIM channel x+1 is used for the definition of the INPUTSRC LUT functionality, the TDU counters TO_CNT 0 and 1 can only be used for timeout measurement. Thus, the timeout threshold for channel x+1 is limited to a 16 bit value. The LUT functionality is defined by setting the content of register `GTM.TIM[0].CH[6].TDUC`. The value for the register is determined by Table 3.2.
Figure 3.17 shows the simulation result for the example Code 6.

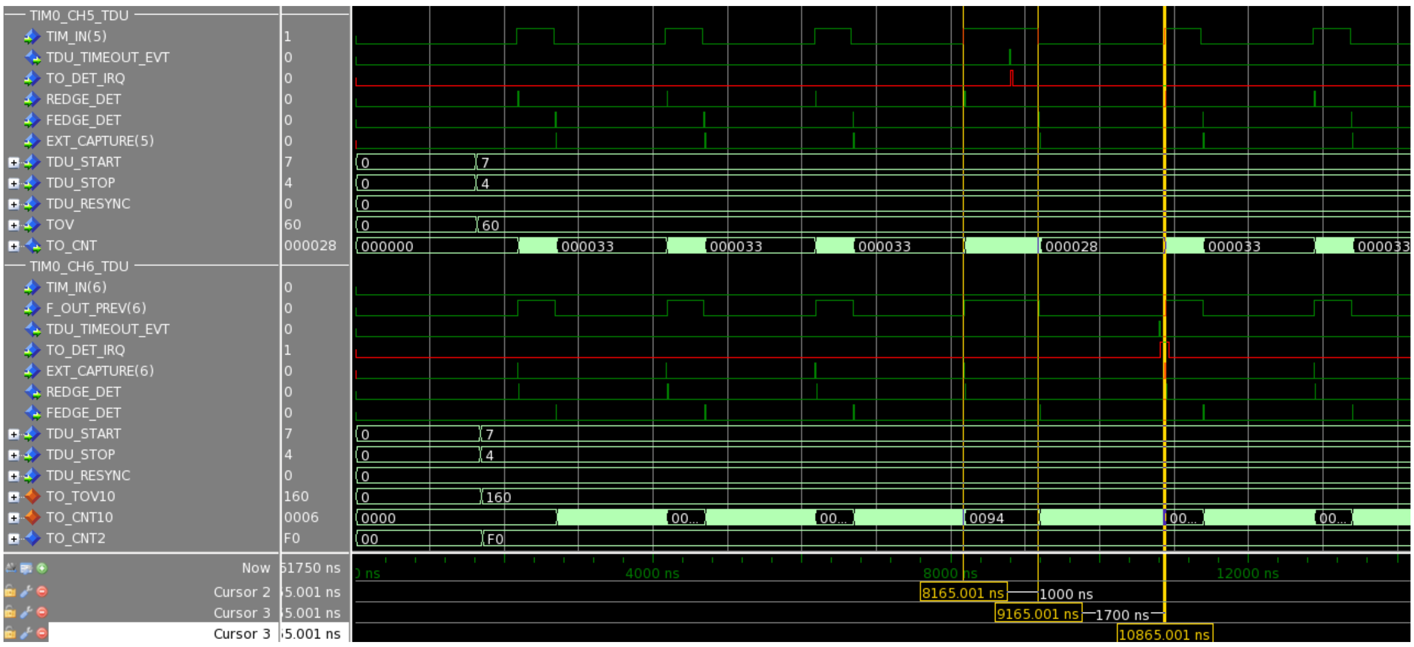

Figure 3.17: Simulation result for timeout measurement on both signal levels

4 References

[1] Robert Bosch GmbH, GTM-IP Specification, Revision 3.1.5.1, 24.03.2016

5 General Information about this document

5.1 LEGAL NOTICE

5.2 Revision History

Version	Date	Description
0.1	10.10.2018	Initial version
1.0	29.10.2018	Released

Robert Bosch GmbH
AE/EID5
Postfach 13 42
72703 Reutlingen
Germany

bosch.semiconductors@de.bosch.com