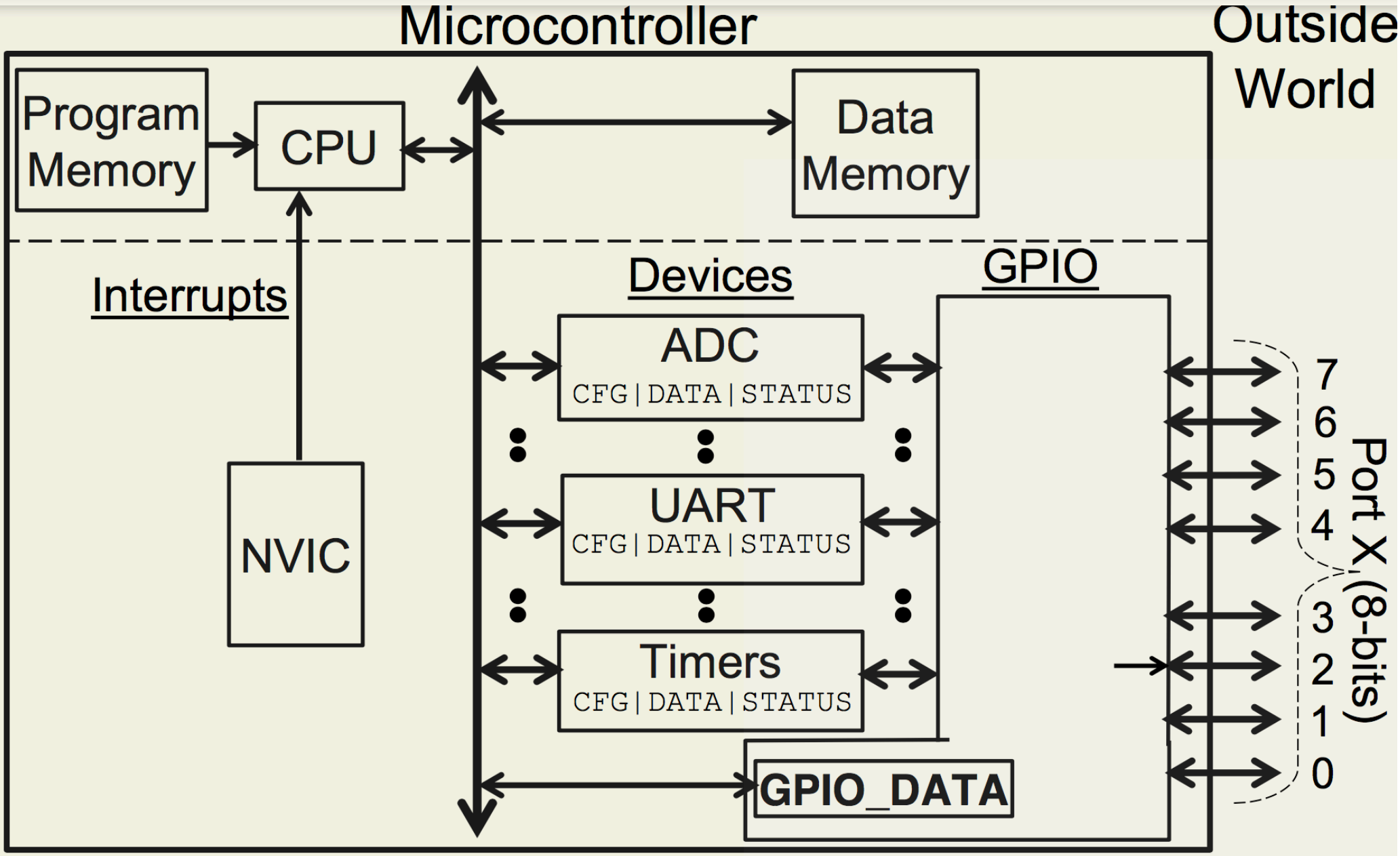
CprE 288 – Introduction to Embedded Systems (Timers, Input Capture, General Purpose Timer Module)

Instructor:
Dr. Diane Rover

TM4C Microcontroller



9.2 General Purpose Timers (Textbook)

- The General-Purpose Timer Module (**GPTM**) contains **six** 16/32-bit GPTM blocks.
- Each **16/32-bit** GPTM block can provide **two** 16-bit (half-width) timers/counters
  - **–Timer A**
  - **–Timer B**
- These timers/counters can be further configured to operate independently as timers or event counters, or concatenated together to operate as one **32-bit** (full-width) timer.
- All timers have interrupt controls and separate interrupt vectors as well as separate interrupt handlers.

9.2 General Purpose Timers

- Each 16/32-bit GPTM block (block0 ~ block5) can:
  - Be set up to run as a **one-shot timer** or a **continuous timer** in either half-width or full-width modes.
    - If configured in one-shot mode, the timer stops counting when it reaches **zero** when counting down or the load value when counting up.
    - If in continuous mode, the timer counts to zero (counting down) or the load value (counting up), then reloads and continues counting.
  - Be configured for **event capture** or as a PWM generator when working in the half-width mode.
    - When working for event capture, the timer acts as a counter. It can be configured to either count the time between events or the events themselves. The type of event being counted can be configured as a positive edge, a negative edge, or both edges.
    - When a timer is configured as a PWM generator, the input signal used to capture events becomes an output signal, and the timer drives an edge-aligned pulse onto that signal.

9.2.2 The General Purpose Timer Module Components

- The **cores** of each GPTM block are two **free-running up/down counters** referred to as **Timer A** and **Timer B**.
- The exact functionality of each GPTM is controlled by software and configured through the register interface.

Table 9.1 Available modes for each GPTM block.

Mode	Timer	Count Mode	Counter Size		Prescaler Size		Prescaler Function
			16/32-bit GPTM	32/64-bit Wide GPTM	16/32-bit GPTM	32/64-bit Wide GPTM	
One-Shot	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Up) Prescaler (Down)
	Concatenated	Up or Down	32-bit	64-bit	-	-	-
Periodic	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Up) Prescaler (Down)
	Concatenated	Up or Down	32-bit	64-bit	-	-	-
RTC	Concatenated	Up	32-bit	64-bit	-	-	-
Edge Count	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Both)
Edge Time	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Both)
PWM	Individual	Down	16-bit	32-bit	8-bit	16-bit	Timer Extension

9.2.3.1 One-Shot and Periodic Timer Mode

- Table 9.2 shows a group of configurations for a 16-bit free-running timer using the prescaler. All values assume an **80-MHz** clock with the clock period **Tc** = 12.5 ns. The prescaler can only be used when a 16/32-bit timer is configured in **16-bit** mode.

Table 9.2 16-bit Timer with prescaler configurations.

Prescale (8-bit value)	Number of Timer Clocks	Max Time	Unit
00000000	1	0.8192	ms
00000001	2	1.6384	ms
00000010	3	2.4576	ms
00000011	4	3.2768	ms
00000100	5	4.0960	Ms
00000101	6	4.9152	Ms
00000110	7	5.7344	Ms
00000111	8	6.5536	ms
--------	-	-	-
11111111	256	209.7152	ms

9.2 General Purpose Timers

- Each 16/32-bit GPTM block (block0 ~ block5) can:
  - –Be set up to run as a **one-shot timer** or a **continuous timer** in either half-width or full-width modes.
    - If configured in one-shot mode, the timer stops counting when it reaches **zero** when counting down or the load value when counting up.
    - If in continuous mode, the timer counts to zero (counting down) or the load value (counting up), then reloads and continues counting.
  - –Be configured for **event capture** or as a PWM generator when working in the half-width mode.
    - When working for event capture, the timer acts as a counter. It can be configured to either count the time between events or the events themselves. The type of event being counted can be configured as a positive edge, a negative edge, or both edges.
    - When a timer is configured as a PWM generator, the input signal used to capture events becomes an output signal, and the timer drives an edge-aligned pulse onto that signal.

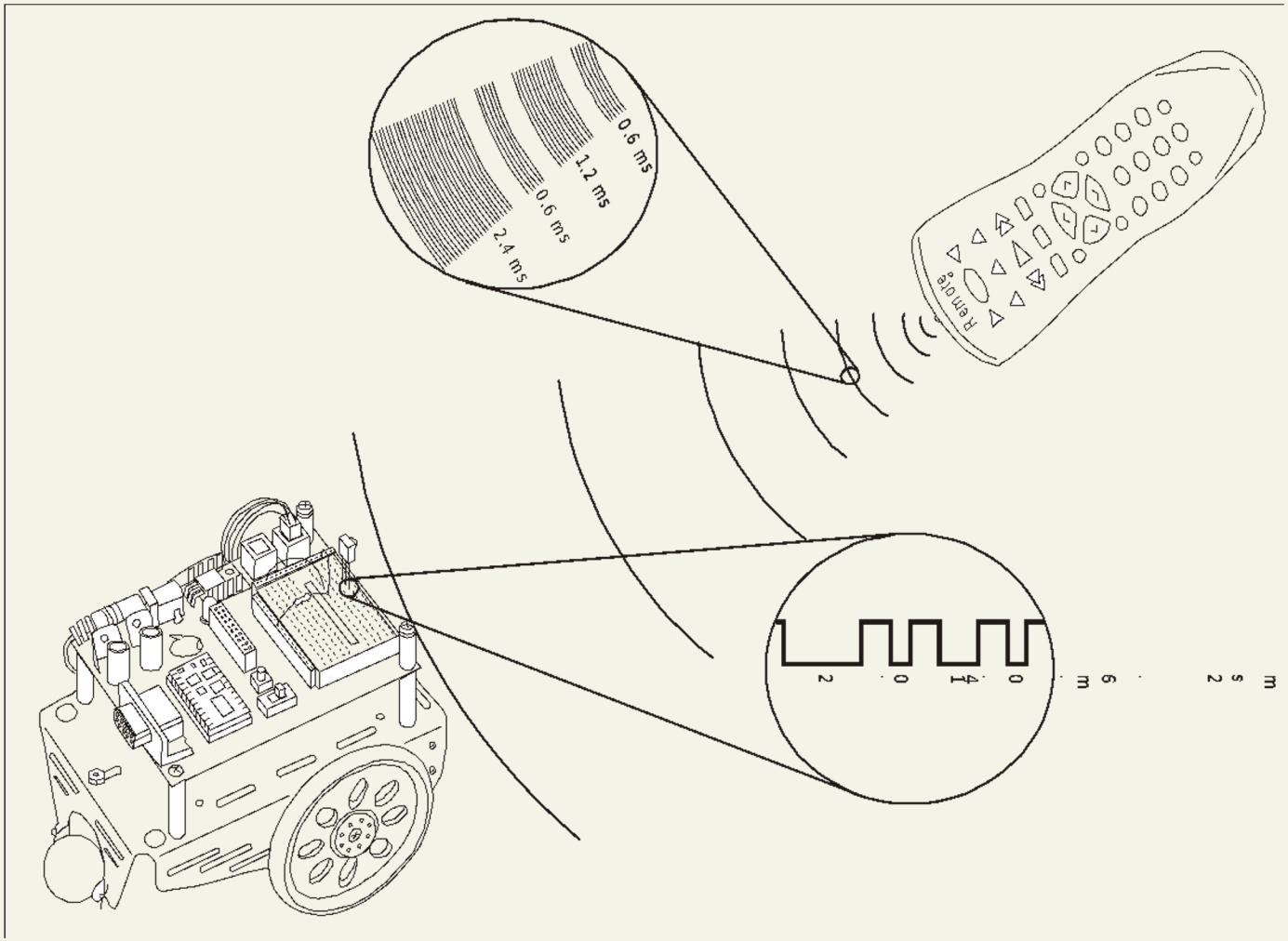
Input Capture

Capture the times of events

Many uses in microcontroller applications

Generally, any input can be treated as a series of events, where the precise measure of event times is important

Application: Remote Control (Decoding)

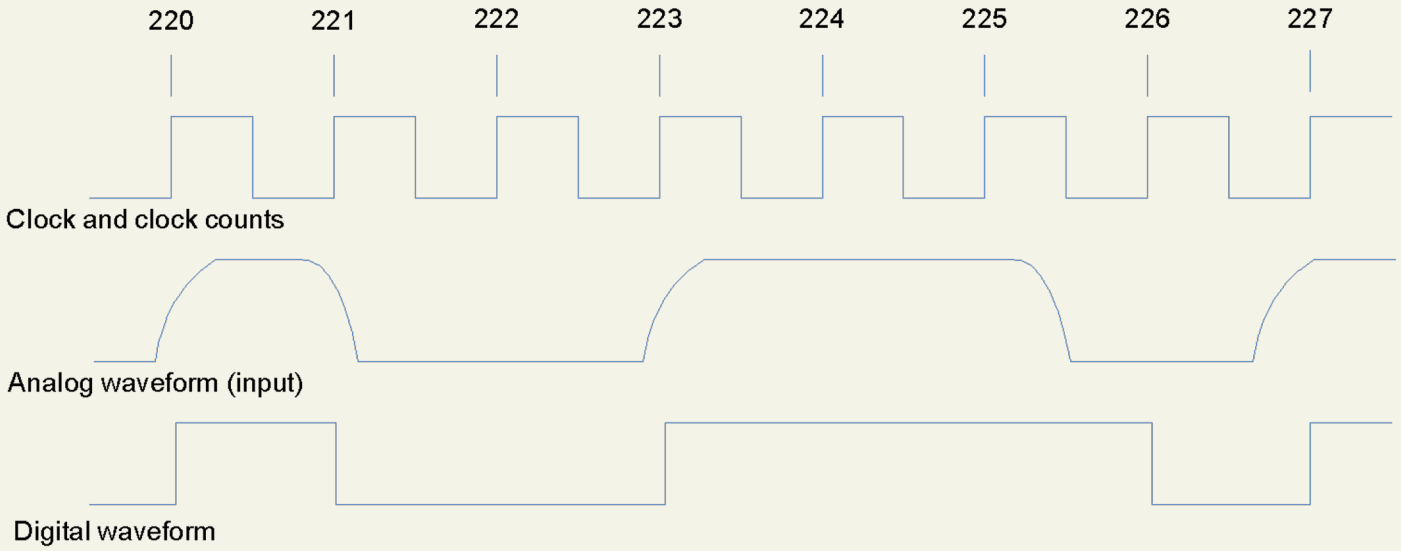


Input Capture

An event is a transition of binary signal

Example:

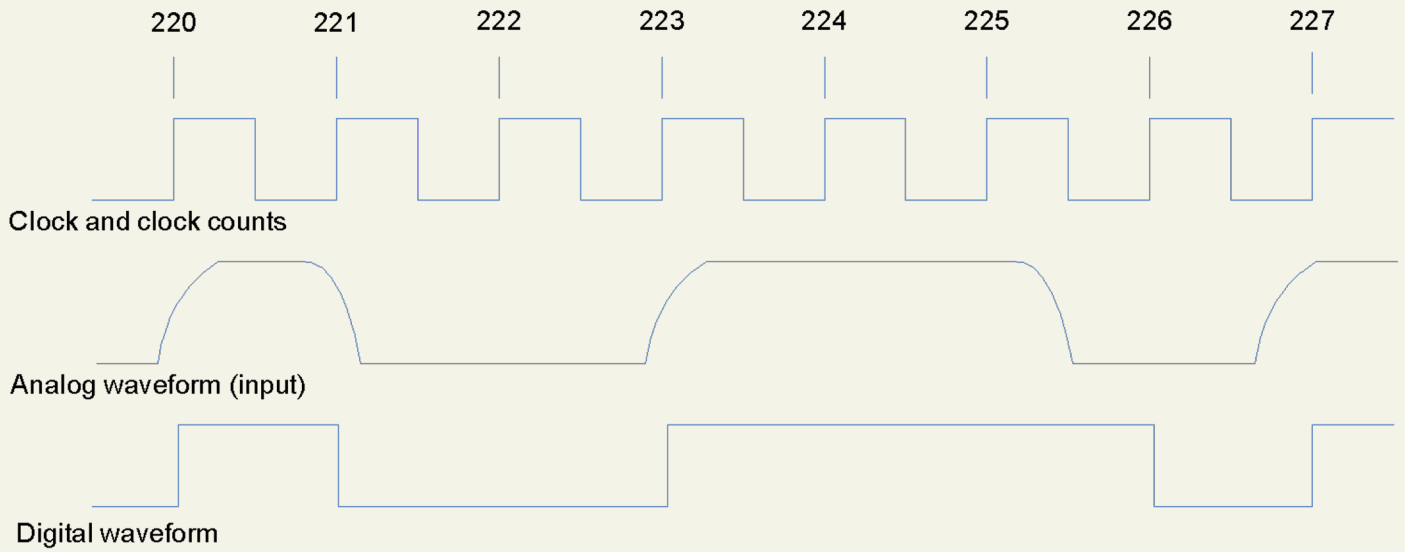
How many events make up the following waveform?



Input Capture

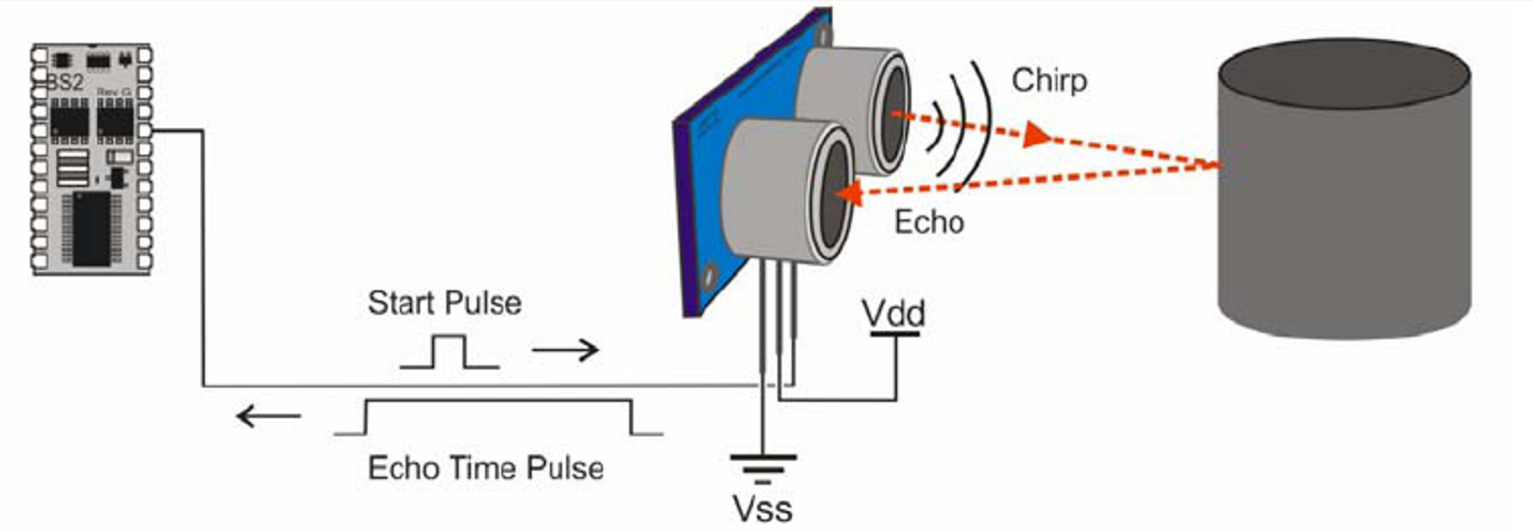
An input is digitized and then times are captured

Example: The input is understood as events occurring at the following times: 220, 221, 223, 226, and 227 with initial state as low

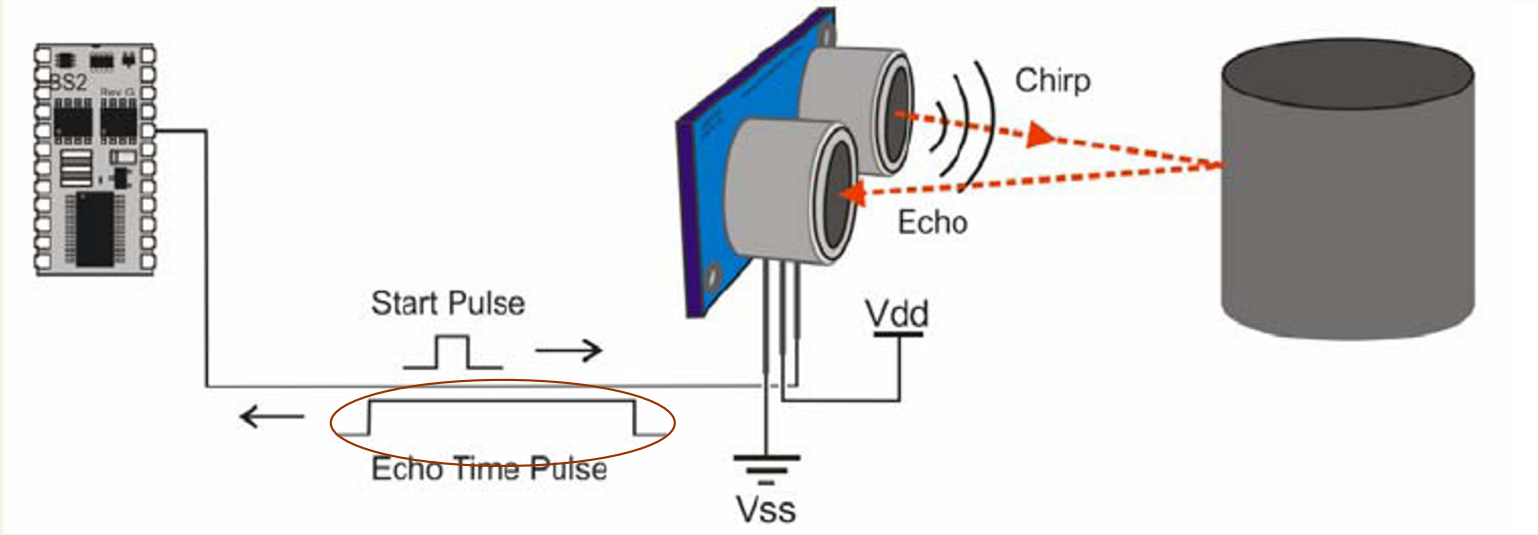


Application: Sonar Device

PING))) sensor: ultrasound distance detection device



Sonar Principle



Speed of sound in lab room temperature: about 340 m/s
Pulse width proportional to round-trip distance

* Temperature affects sound speed

Input Capture: Design Principle

Time is important!

How could a microcontroller capture the time of an event, assuming a clock count can be read?

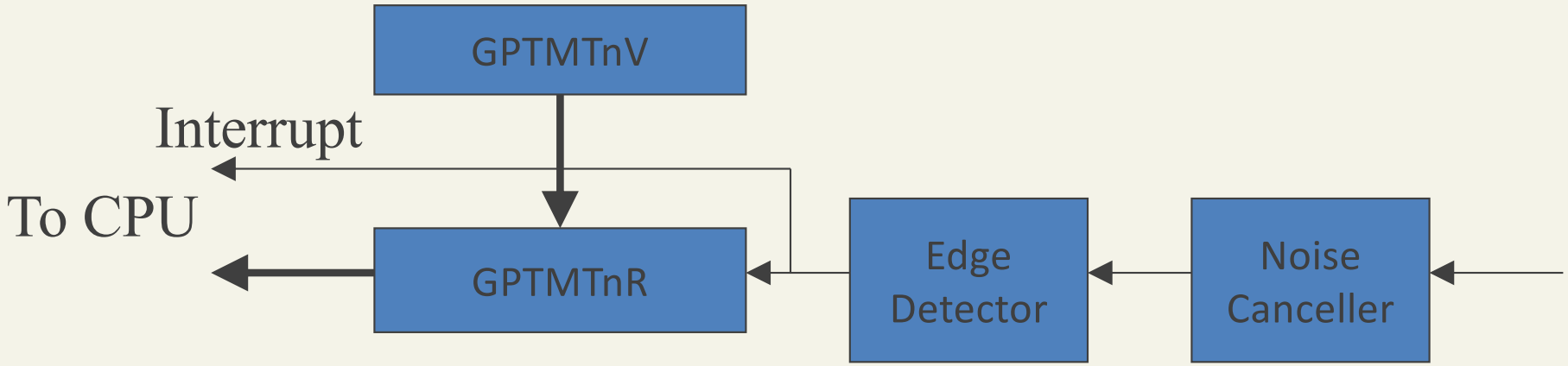
- Keep polling the input pin?
- Use an interrupt?
- ???

Precise timing is needed!

Input Capture: Design Principle

Time value (clock count) is captured first by timer hardware, then read by the CPU in software

Note that if the edge detect input is not set up to get the event, the timer register TnR will NOT be loaded with the value register TnV.



GPTMTnV: Timer n Value Register (n is A or B)
GPTMTnR: Timer Register (in Edge-Time mode, this register is loaded with the value in GPTMTnV at the last input edge event)

9.2.3.6 Input Edge-Time Mode

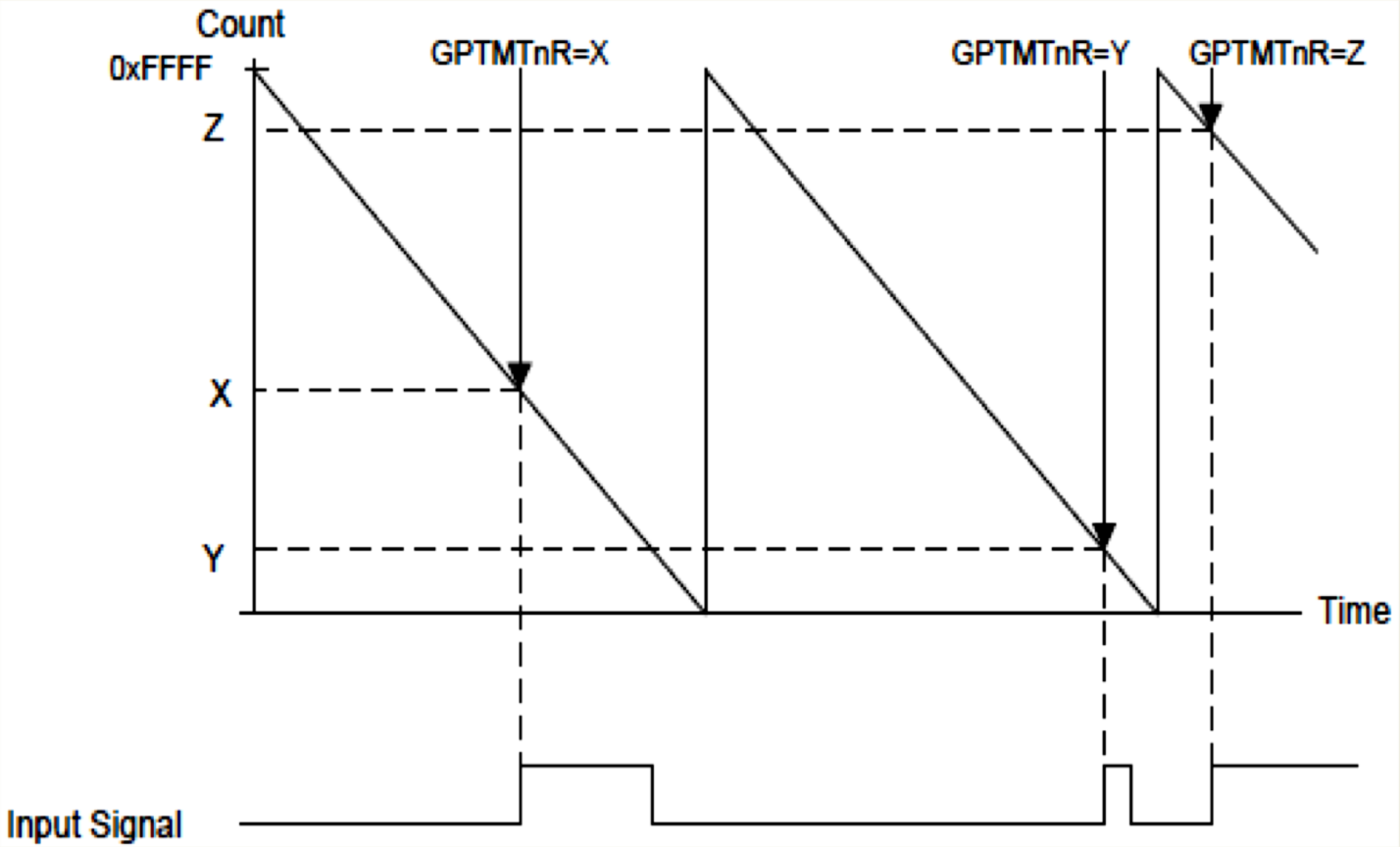


Figure 9.4 An example of using the count-down mode timer to detect the edge time.

Input Capture: Design Principle

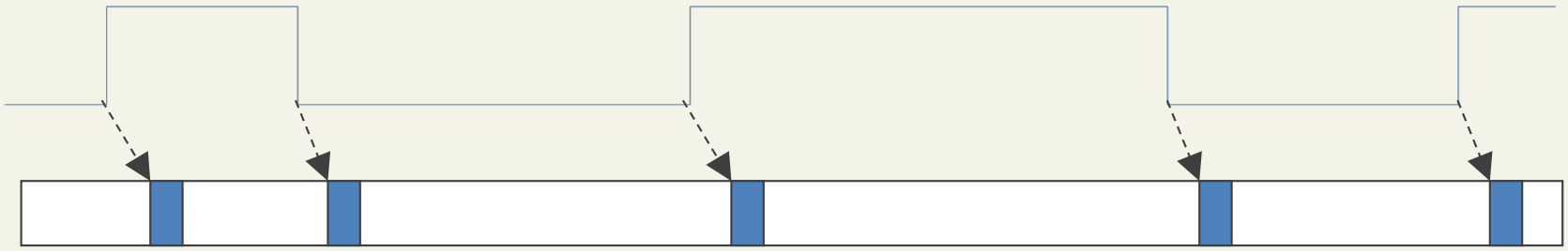
What happens in hardware and software when and after an event occurs:

- The event's time is *captured* in the GPTMTnR (timer register)
- An interrupt is raised to the CPU
- CPU executes the input capture ISR, which reads the timer register and completes the related processing

The captured time is *precise* because it's captured immediately when the event occurs.

The ISR should read the timer register and complete its processing fast enough to avoid loss of events.

Input Capture: Design Principle

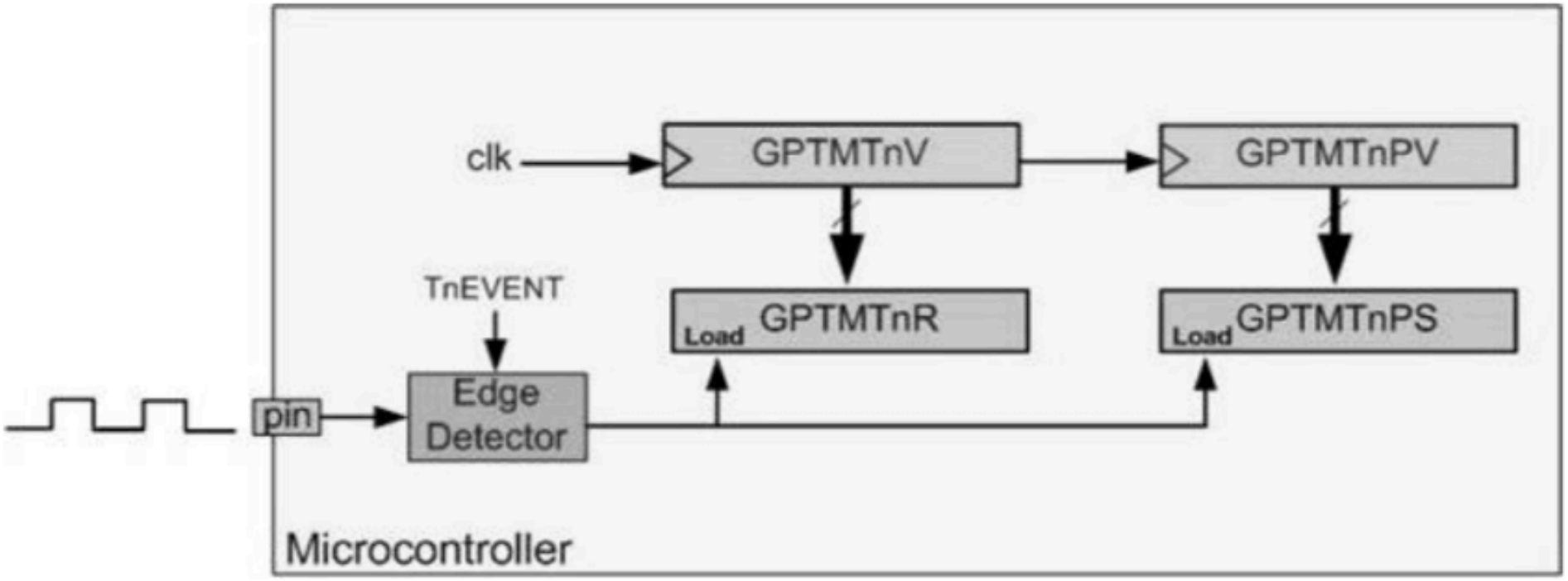


Interrupt

CPU interrupt processing

CPU foreground computation

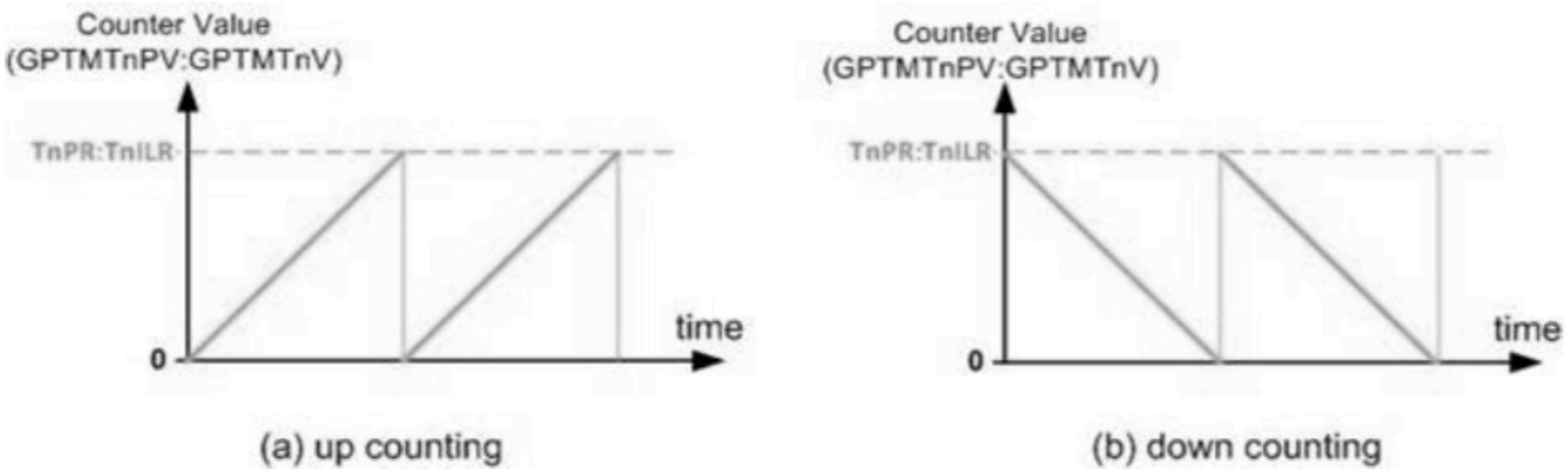
Input Capture: Design Principle (24-bit)



(/edwiki/File:Tm4c_input_edge_time_capture.png)

Figure 9.1: Input Edge Time Capturing

Input Capture: Design Principle (24-bit)

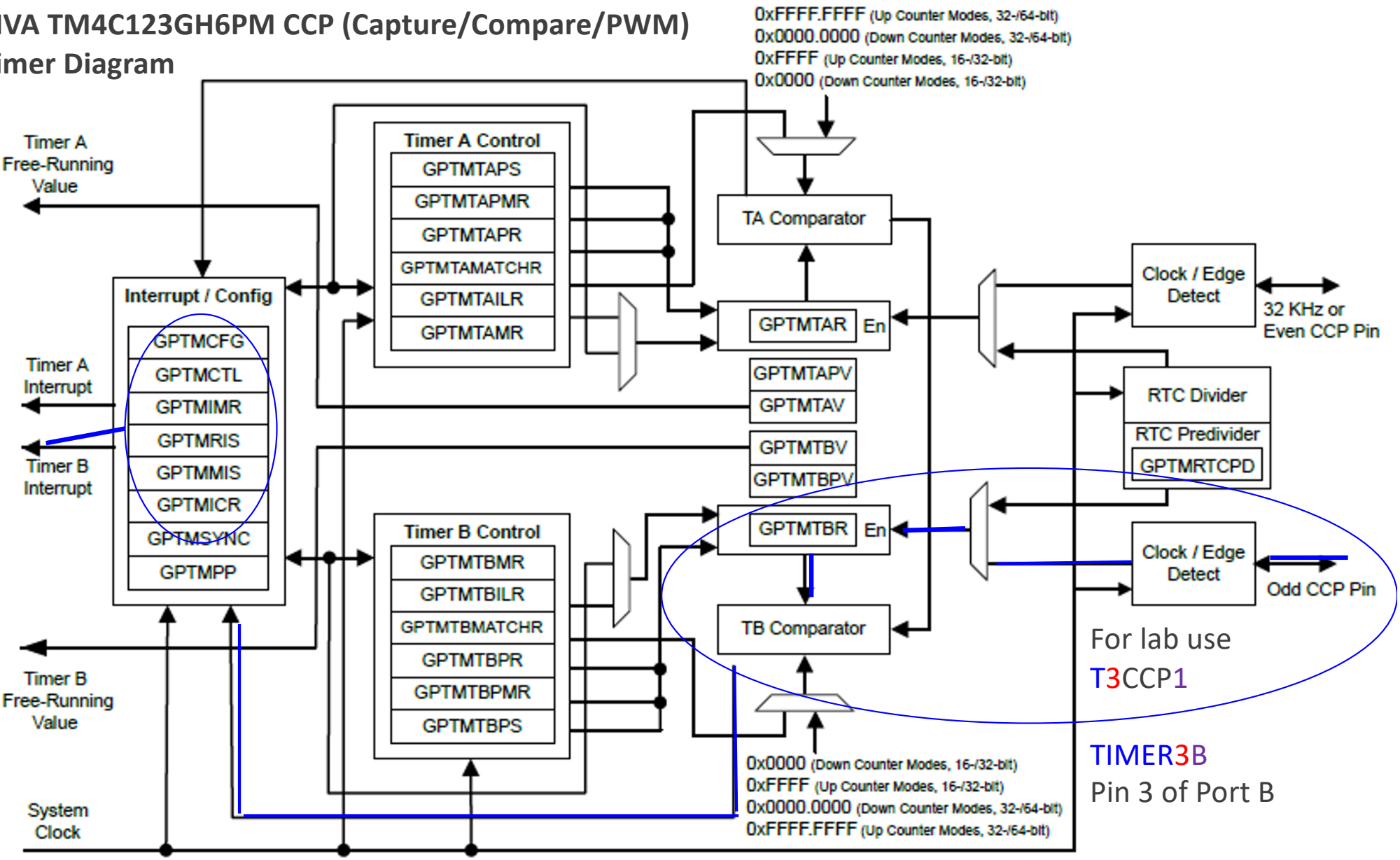


(/edwiki/File:Tm4c_counting_in_input_edge_time_mode.png)

Figure 9.2: Counting in Input Edge-Time Mode

Figure 11-1. GPTM Module Block Diagram

TIVA TM4C123GH6PM CCP (Capture/Compare/PWM) Timer Diagram



(More on pages 704-707 of datasheet)

9.2.5 General Purpose Timer Module GPIO Related Control Signals

Table 9.7 General Purpose Timers CCP pins distributions

Timers	Up/Down Counter	Even CCP Pins	Odd CCP Pins
16/32-bit Timer 0	**Timer A**	T0CCP0	-
	Timer B	-	T0CCP1
16/32-bit Timer 1	**Timer A**	T1CCP0	-
	Timer B	-	T1CCP1
16/32-bit Timer 2	**Timer A**	T2CCP0	-
	Timer B	-	T2CCP1
16/32-bit Timer 3	**Timer A**	T3CCP0	-
	Timer B	-	T3CCP1
16/32-bit Timer 4	**Timer A**	T4CCP0	-
	Timer B	-	T4CCP1
16/32-bit Timer 5	**Timer A**	T5CCP0	-
	Timer B	-	T5CCP1
32/64-bit Wide Timer 0	**Timer A**	WT0CCP0	-
	Timer B	-	WT0CCP1
32/64-bit Wide Timer 1	**Timer A**	WT1CCP0	-
	Timer B	-	WT1CCP1
32/64-bit Wide Timer 2	**Timer A**	WT2CCP0	-
	Timer B	-	WT2CCP1
32/64-bit Wide Timer 3	**Timer A**	WT3CCP0	-
	Timer B	-	WT3CCP1
32/64-bit Wide Timer 4	**Timer A**	WT4CCP0	-
	Timer B	-	WT4CCP1
32/64-bit Wide Timer 5	**Timer A**	WT5CCP0	-
	Timer B	-	WT5CCP1

9.2.5 General Purpose Timer Module GPIO Related Control Signals

Table 9.8 General Purpose Timers signals and GPIO pins distributions (Part I).

GPTM Pin	GPIO Pin	Pin Type	Pin Function
T0CCP0	PB6 (7) PF0 (7)	I/O	16/32-Bit Timer 0 Capture/Compare/PWM 0.
T0CCP1	PB7 (7) PF1 (7)	I/O	16/32-Bit Timer 0 Capture/Compare/PWM 1.
T1CCP0	PB4 (7) PF2 (7)	I/O	16/32-Bit Timer 1 Capture/Compare/PWM 0.
T1CCP1	PB5 (7) PF3 (7)	I/O	16/32-Bit Timer 1 Capture/Compare/PWM 1.
T2CCP0	PB0 (7) PF4 (7)	I/O	16/32-Bit Timer 2 Capture/Compare/PWM 0.
T2CCP1	PB1 (7)	I/O	16/32-Bit Timer 2 Capture/Compare/PWM 1.
T3CCP0	PB2 (7)	I/O	16/32-Bit Timer 3 Capture/Compare/PWM 0.
T3CCP1	PB3 (7)	I/O	16/32-Bit Timer 3 Capture/Compare/PWM 1.
T4CCP0	PC0 (7)	I/O	16/32-Bit Timer 4 Capture/Compare/PWM 0.
T4CCP1	PC1 (7)	I/O	16/32-Bit Timer 4 Capture/Compare/PWM 1.
T5CCP0	PC2 (7)	I/O	16/32-Bit Timer 5 Capture/Compare/PWM 0.
T5CCP1	PC3 (7)	I/O	16/32-Bit Timer 5 Capture/Compare/PWM 1.

GPIO Alternate Function Setup: PMCx Values

- Refer to the Tiva datasheet, Table 23-5

Table 23-5. GPIO Pins and Alternate Functions

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PA0	17	-	U0Rx	-	-	-	-	-	-	CAN1Rx	-	-	-
PA1	18	-	U0Tx	-	-	-	-	-	-	CAN1Tx	-	-	-
PA2	19	-	-	SSI0Clk	-	-	-	-	-	-	-	-	-
PA3	20	-	-	SSI0Fss	-	-	-	-	-	-	-	-	-
PA4	21	-	-	SSI0Rx	-	-	-	-	-	-	-	-	-
PA5	22	-	-	SSI0Tx	-	-	-	-	-	-	-	-	-
PA6	23	-	-	-	I2C1SCL	-	M1PWM2	-	-	-	-	-	-
PA7	24	-	-	-	I2C1SDA	-	M1PWM3	-	-	-	-	-	-
PB0	45	USB0ID	U1Rx	-	-	-	-	-	T2CCP0	-	-	-	-
PB1	46	USB0VBUS	U1Tx	-	-	-	-	-	T2CCP1	-	-	-	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-	-	-	-
PB3	48	-	-	-	I2C0SDA	-	-	-	T3CCP1	-	-	-	-
PB4	58	AIN10	-	SSI2Clk	-	M0PWM2	-	-	T1CCP0	CAN0Rx	-	-	-
PB5	57	AIN11	-	SSI2Fss	-	M0PWM3	-	-	T1CCP1	CAN0Tx	-	-	-
PB6	1	-	-	SSI2Rx	-	M0PWM0	-	-	T0CCP0	-	-	-	-
PB7	4	-	-	SSI2Tx	-	M0PWM1	-	-	T0CCP1	-	-	-	-

Timer3B IC GPIO Alternate Function Setup

- Enable the alternate function using the GPIOAFSEL register

// set bit 3 to enable PB3 alternate function

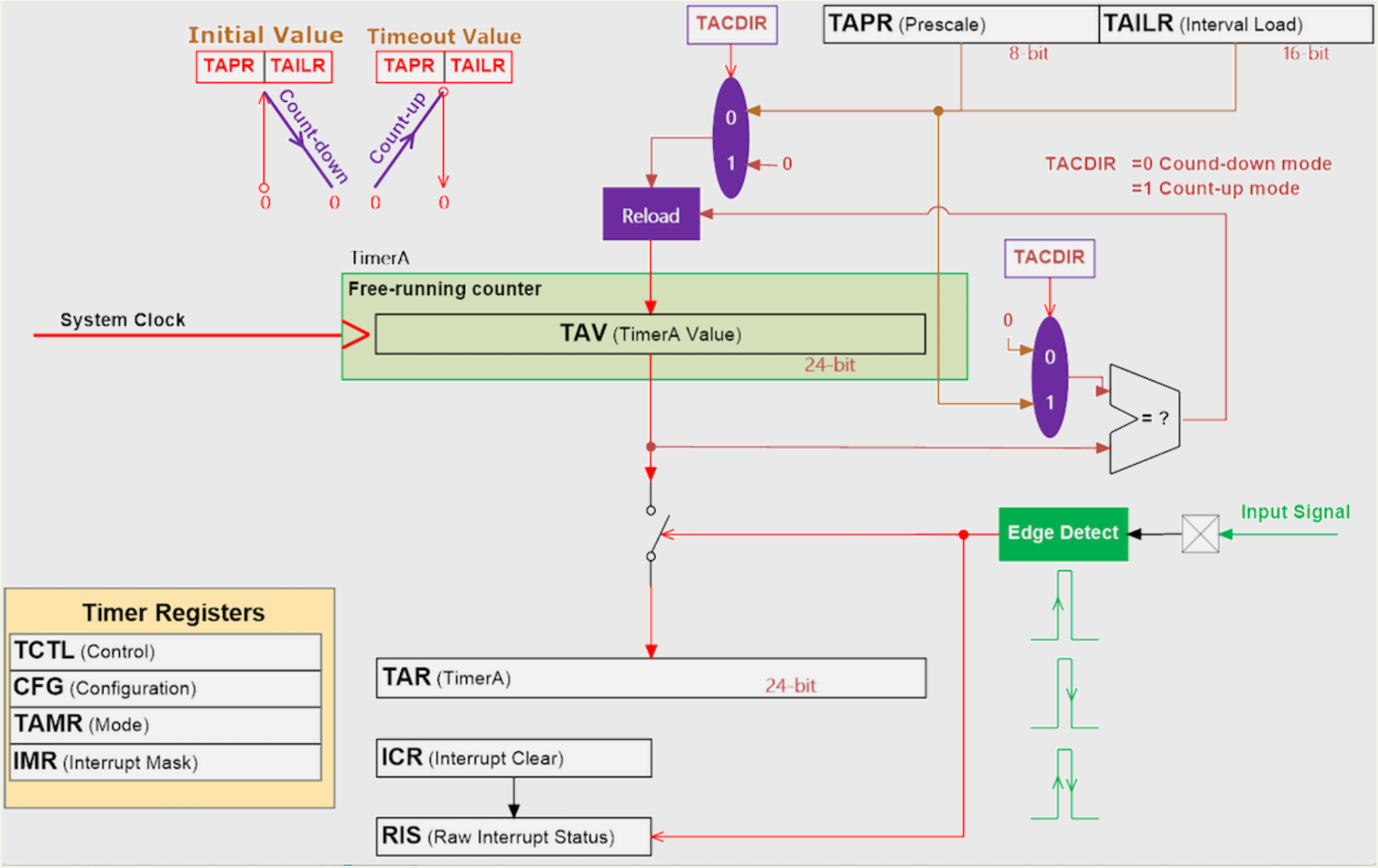
GPIO_PORTB_AFSEL_R = _____;

- Set up the specific function (select the peripheral) using the GPIOPCTL register
  - Function is T3CCP1 for Timer3B input capture mode

// set PB3 to T3CCP1 function using PMC3 = 7

GPIO_PORTB_PCTL_R = _____;

GPTM Module in Input Edge-Time Mode



Input Edge-Time Mode Initialization

11.4.4 Input Edge Time Mode

A timer is configured to Input Edge Time mode by the following sequence:

1. Ensure the timer is disabled (the `TnEN` bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTMCFG)** register with a value of 0x0000.0004.
3. In the **GPTM Timer Mode (GPTMTnMR)** register, write the `TnCMR` field to 0x1 and the `TnMR` field to 0x3 and select a count direction by programming the `TnCDIR` bit.
4. Configure the type of event that the timer captures by writing the `TnEVENT` field of the **GPTM Control (GPTMCTL)** register.
5. If a prescaler is to be used, write the prescale value to the **GPTM Timer n Prescale Register (GPTMTnPR)**.
6. Load the timer start value into the **GPTM Timer n Interval Load (GPTMTnILR)** register.
7. If interrupts are required, set the `CnEIM` bit in the **GPTM Interrupt Mask (GPTMIMR)** register.
8. Set the `TnEN` bit in the **GPTM Control (GPTMCTL)** register to enable the timer and start counting.
9. Poll the `CnERIS` bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the `CnECINT` bit of the **GPTM Interrupt Clear (GPTMICR)** register. The time at which the event happened can be obtained by reading the **GPTM Timer n (GPTMTnR)** register.

9.2.4.2.1 GPTM Timer A Register (GPTMTAR)

- This is a 32-bit register and all 32 bits are used to indicate the current value of the **16-bit free-running counter** or **Timer A**.
- This register shows the current value of the Timer A counter in all cases **except for Input Edge Time and Count modes**. **In Input Edge Time mode, this register contains the time at which the last edge event took place.**
- **In 16-bit mode:**
  - **Bits 15:0 contain the value of the counter and bits 23:16 contain the value of the prescaler in Input Edge Count, Input Edge Time, and PWM modes, which is the upper 8 bits of the count.**
  - **In input edge time mode, the count is a 24-bit value, as the 8-bit prescaler is an extension to the 16-bit timer.**
- Read-only register

9.2.4.2.2 GPTM Timer A Value Register (GPTMTAV)

- This is a **32-bit** read/write register and is used to store the current value of the Timer A free running counter.
- When read, this register shows the current, free-running value of Timer A in all modes.
- When written, the value written into this register is loaded into the **GPTMTAR** register on the next clock cycle.
- The difference between the **GPTMTAR** and **GPTMTAV** is that the former is a **read only** register, but the latter is a **read/write** register.
  - GPTMTnR: read only
  - GPTMTnV: read/write

9.2.4.2.2 GPTM Timer A Value Register (GPTMTAV)

- In **16-bit** mode, bits **15:0** contain the value of the counter and bits **23:16** contain the current, free-running value of the prescaler.
- In one-shot or periodic down count modes, the prescaler stored in **23:16** is a true prescaler, meaning bits **23:16** count down before decrementing the value in bits **15:0**. The prescaler in bits **31:24** always reads as **0**.

9.2.4.1 Timer A Control Register Group

- Eight (8) registers are used to configure and control the operations of the Timer A:
  - GPTM Configuration Register (**GPTMCFG**)
  - GPTM Control Register (**GPTMCTL**)
  - GPTM Timer A Mode Register (**GPTMTAMR**)
  - GPTM Timer A Interval Load Register (**GPTMTAILR**)
  - GPTM Timer A Prescale Register (**GPTMTAPR**)
  - GPTM Timer A Match Register (**GPTMTAMATCHR**)
  - GPTM Timer A Prescale Match Register (**GPTMTAPMR**)
  - GPTM Timer A Prescale Snapshot Register (**GPTMTAPS**)

Timer Programming Interface

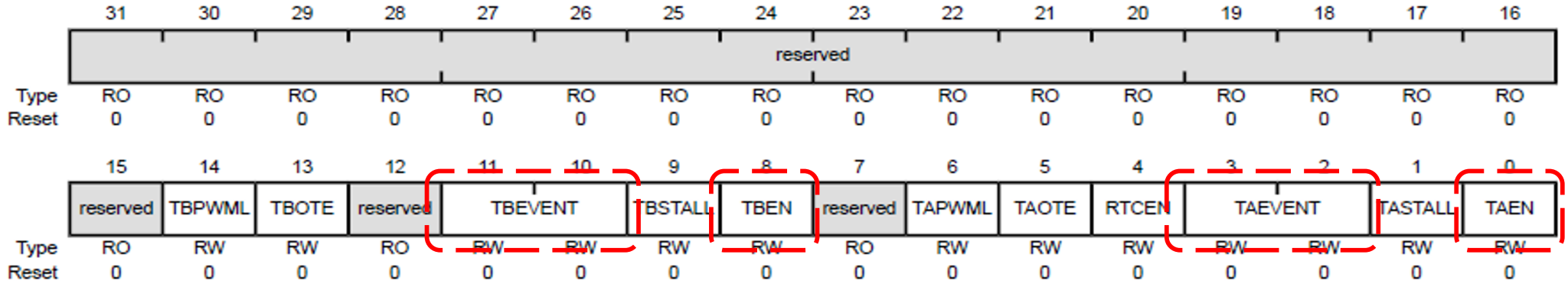
GPTMCTL: GPTM Control

GPTMCFG: GPTM Configuration

GPTMTnMR: GPTM Timer n Mode (n is A or B)

GPTMTnILR: GPTM Timer n Interval Load

GPTMCTL (TIMERx_CTL_R)



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved	TBPWML	TBOTE	reserved	TBEVENT		TBSTALL	TBEN	reserved	TAPWML	TAOTE	RTCEN	TAEVENT		TASTALL	TAEN
Type	RO	RW	RW	RO	RW	RW	RW	RW	RO	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TnEN: GPTM **Timer n Enable Bit** – Set this bit to enable Timer n.

Make sure a timer is *disabled* before trying to change its settings.

TnEVENT: GPTM **Timer n Event Mode** – Which edge will trigger an interrupt?

00: Positive (rising) edge

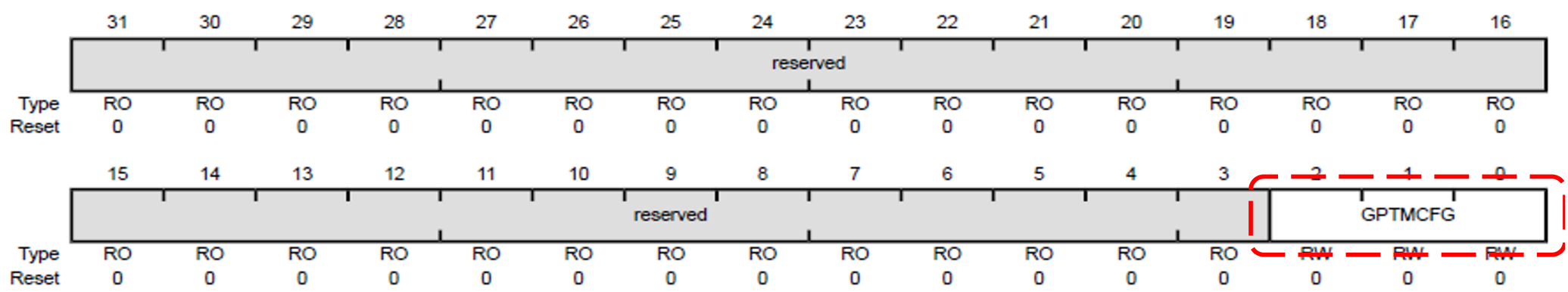
01: Negative (falling) edge

10: Reserved

11: Both

(from p. 737 of datasheet)

GPTMCFG (TIMERx_CFG_R)



GPTMCFG: GPTM Configuration

0x0: "Concatenated" mode (16/32 bit timers use 32 bits, 32/64 bit timers use 64 bits.)

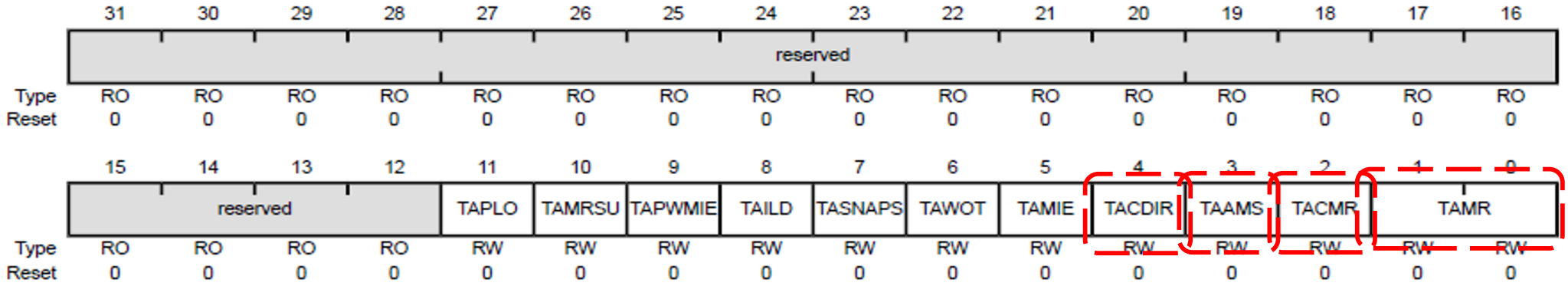
0x1: Concatenated mode, and timers are set to RTC (real-time clock) counter configuration

0x4: 16/32 bit timers are split into two 16-bit timers, Timer A and Timer B. 32/64 bit timers are split into two 32-bit timers.

Other values for GPTMCFG: reserved

(p. 727 of datasheet)

GPTMTnMR (TIMERx_TnMR_R)



GPTMTnMR: GPTM Timer n Mode

TnMR: Timer n Mode

- **0x0**: Reserved
- **0x1**: One-Shot Timer Mode
- **0x2**: Periodic Timer Mode
- **0x3**: Capture Mode

(p. 729 (Timer A) & p. 733 (Timer B) of datasheet)

TnCMR: Capture Mode

GPTM Timer A Capture Mode

The TACMR values are defined as follows:

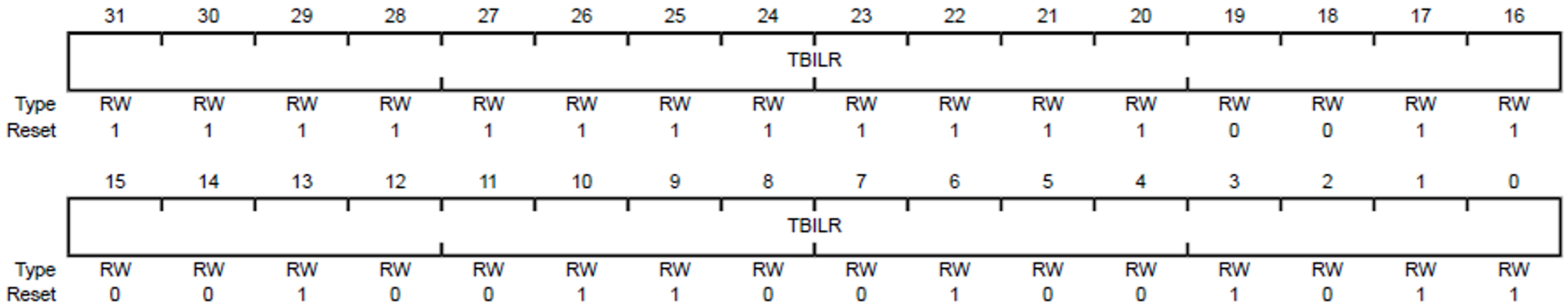
Value	Description
0	Edge-Count mode
1	Edge-Time mode

TnCDIR: Count DIRection

GPTM Timer A Count Direction

Value	Description
0	The timer counts down.
1	The timer counts up. When counting up, the timer starts from a value of 0x0.

GPTMTnILR (TIMERx_TnILR_R)



GPTMTnILR: GPTM Timer n Interval (Initial) Load value

When the timer is counting down to 0, this register is used to load the starting count value. When the timer is counting up from 0, this register sets the upper bound for the timeout event.

For input capture mode, to capture a 24-bit timer (counter) in the TnR register, both TnILR and its corresponding prescaler load register TnPR should be initialized with their max values.

For example:

```
TIMER3_TBPR_R = 0xFF;
TIMER3_TBILR_R = 0xFFFF;
```

Timer3B Configuration for Lab

TIMER3_CTL_R: Enable TB, Both edges for TB event
TIMER3_CFG_R: 16-bit mode
TIMER3_TBMR_R: Capture mode, Edge-time mode, Count down
TIMER3_TBILR_R: 0xFFFF (16-bit max value)
TIMER3_TBPR_R: 0xFF (8-bit max value)
TIMER3_IMR_R: Enable TB capture interrupt

GPIO Port B pin 3 (PB3)
→ Timer3B Capture/Compare/PWM (CCP) pin
→ Connects to the input/output pin of the Ping))) sensor

9.2.4.3 Timers A and B Interrupt and Configuration Register Group

- Six registers are used to control and handle interrupts of the Timers A and B:
  - GPTM Interrupt Mask Register (**GPTMIMR**)
  - GPTM Raw Interrupt Status Register (**GPTMRIS**)
  - GPTM Masked Interrupt Status Register (**GPTMMIS**)
  - GPTM Interrupt Clear Register (**GPTMICR**)
  - GPTM Synchronize Register (**GPTMSYNC**)
  - GPTM Peripheral Properties Register (**GPTMPP**)

Timer Programming Interface - Interrupts

GPTMCTL: GPTM Control

GPTMCFG: GPTM Configuration

GPTMTnMR: GPTM Timer n Mode (n is A or B)

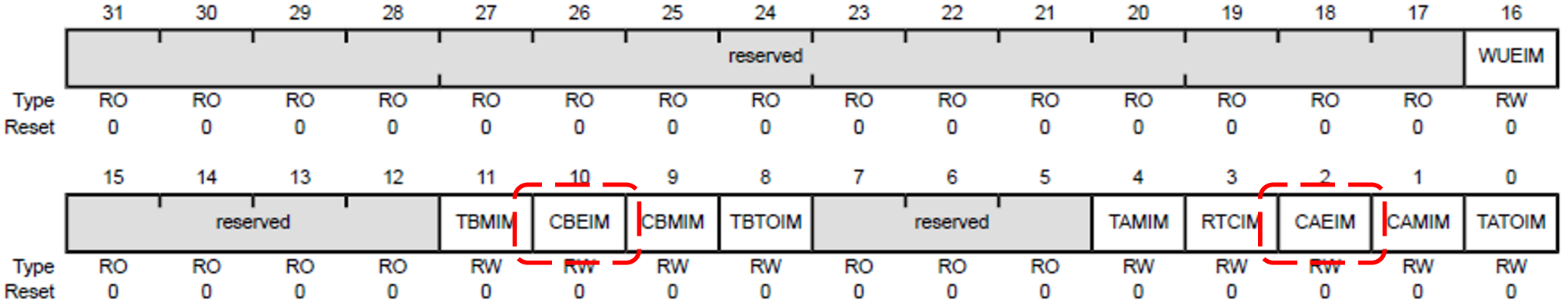
GPTMTnILR: GPTM Timer n Interval Load

GPTMIMR: GPTM Interrupt Mask Register

GPTMMIS: GPTM Masked Interrupt Status

GPTMICR: GPTM Interrupt Clear Register

GPTMIMR (TIMERx_IMR_R)



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															WUEIM
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved				TBMIM	CBEIM	CBMIM	TBTOIM	reserved			TAMIM	RTCIM	CAEIM	CAMIM	TATOIM
Type	RO	RO	RO	RO	RW	RW	RW	RW	RO	RO	RO	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

For any interrupt in GPTMIMR write to the corresponding bit:

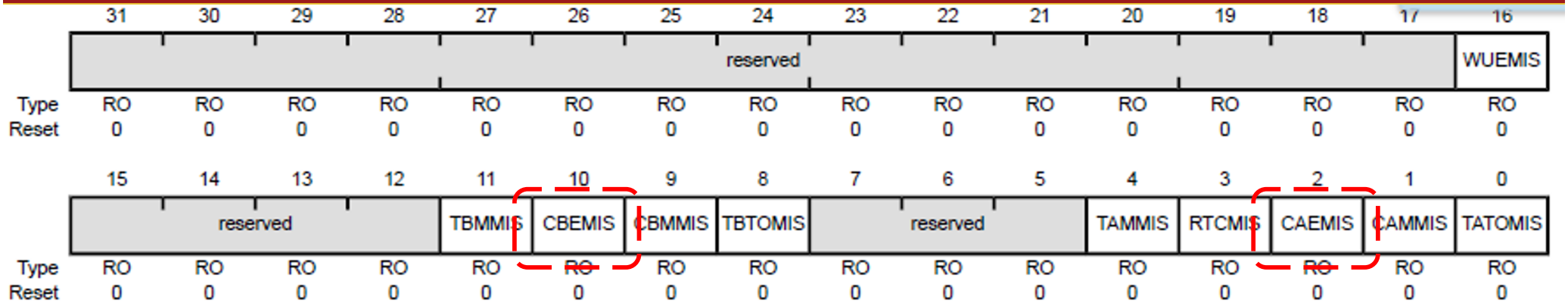
0 to disable the interrupt

1 to enable the interrupt

CnEIM: Timer n Capture Mode Event Interrupt Mask

(p. 745 of datasheet)

GPTMMIS (TIMERx_MIS_R)



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															WUEMIS
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

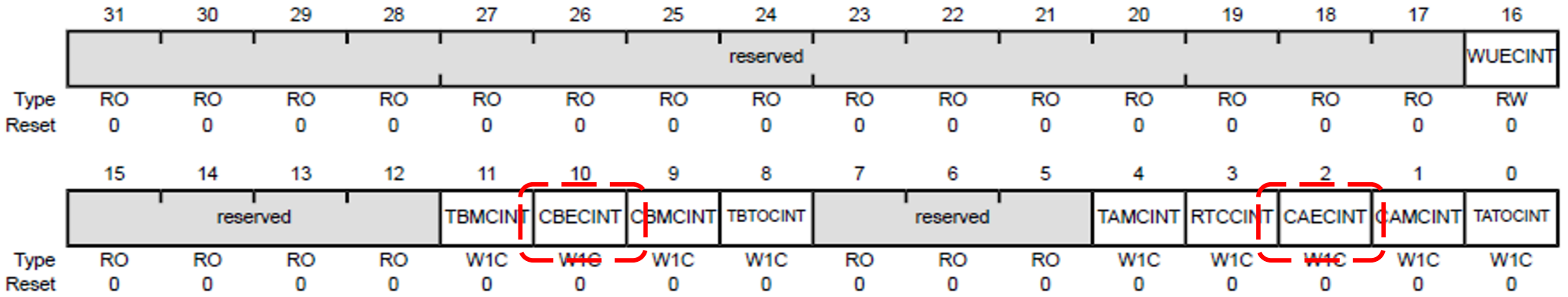
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved				TBMMIS	CBEMIS	CBMMIS	TBTOMIS	reserved			TAMMIS	RTCMIS	CAEMIS	CAMMIS	TATOMIS
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The **GPTMMIS** register shows the status of *unmasked (enabled)* interrupts. Each timer x has 1 ISR vector that all of its interrupts trigger, so it is necessary to check the GPTMMIS register to see which specific interrupt occurred (i.e., polled interrupts).

CnEMIS: Timer n Capture Mode Event Flag

(p. 751 of datasheet)

GPTMICR (TIMERx_ICR_R)



To clear an interrupt flag write a 1 to the corresponding bit in GPTMICR.

CnECINT: Clears the Timer n Capture Mode Event Flag

(p. 754 of datasheet)

Timer3B IC Interrupt Setup

```
// Disable Timer3B while setting it up…

// Assign various configuration settings...

// Clear capture interrupt flag

TIMER3_ICR_R |= 0x400; // CBECINT = bit 10, 0b100_0000_0000

// Enable event capture interrupts

TIMER3_IMR_R |= 0x400; // CBEIM = bit 10, 0b100_0000_0000

// Set up NVIC for Timer3B IC interrupts…

//Bind Timer3B interrupt requests to your interrupt handler

IntRegister(52, TIMER3B_Handler);

// Re-enable Timer3B after setting it up…

// Enable global interrupts…
```

5.3.2.1 The NVIC Interrupt Priority-Level Registers

- Table 5.10 shows relationships between interrupt, its handler and its priority bits.

Table 5.10 The relationship between vectors and NVIC definitions.

Vector Address	Exception Number	IRQ Number	ISR Name in Startup_TM4C123.s	NVIC Macros for Priority Register	Priority Bits
0x00000038	14	-2	PendSV_Handler	NVIC_SYS_PRI3_R	23 - 21
0x0000003C	15	-1	SysTick_Handler	NVIC_SYS_PRI3_R	31 - 29
0x00000040	16	0	GPIOA_Handler	NVIC_PRI0_R	7 - 5
0x00000044	17	1	GPIOB_Handler	NVIC_PRI0_R	15 - 13
0x00000048	18	2	GPIOC_Handler	NVIC_PRI0_R	23 - 21
0x0000004C	19	3	GPIOD_Handler	NVIC_PRI0_R	31 - 29
0x00000050	20	4	GPIOE_Handler	NVIC_PRI1_R	7 – 5
0x00000054	21	5	UART0_Handler	NVIC_PRI1_R	15 – 13
0x00000058	22	6	UART1_Handler	NVIC_PRI1_R	23 – 21
0x0000005C	23	7	SSI0_Handler	NVIC_PRI1_R	31 – 29
0x00000060	24	8	I2C0_Handler	NVIC_PRI2_R	7 – 5
0x00000064	25	9	PWM0_Fault_Handler	NVIC_PRI2_R	15 – 13
0x00000068	26	10	PWM0_0_Handler	NVIC_PRI2_R	23 – 21
0x0000006C	27	11	PWM0_1_Handler	NVIC_PRI2_R	31 – 29
0x00000070	28	12	PWM0_2_Handler	NVIC_PRI3_R	7 – 5
0x00000074	29	13	QEI0_Handler	NVIC_PRI3_R	15 – 13
0x00000078	30	14	ADC0SS0_Handler	NVIC_PRI3_R	23 – 21
0x0000007C	31	15	ADC0SS1_Handler	NVIC_PRI3_R	31 – 29
0x00000080	32	16	ADC0SS2_Handler	NVIC_PRI4_R	7 – 5
0x00000084	33	17	ADC0SS3_Handler	NVIC_PRI4_R	15 – 13
0x00000088	34	18	WDT0_Handler	NVIC_PRI4_R	23 – 21
0x0000008C	35	19	TIMER0A_Handler	NVIC_PRI4_R	31 – 29
0x00000090	36	20	TIMER0B_Handler	NVIC_PRI5_R	7 – 5
0x00000094	37	21	TIMER1A_Handler	NVIC_PRI5_R	15 – 13
0x00000098	38	22	TIMER1B_Handler	NVIC_PRI5_R	23 – 21
0x0000009C	39	23	TIMER2A_Handler	NVIC_PRI5_R	31 – 29

Interrupt Vector Number (to identify interrupt source and access vector table)

Interrupt Number (bit in interrupt registers)

See Table 2-9 in Tiva datasheet.

5.3.2.1 The NVIC Interrupt Priority-Level Registers

- Table 5.10 shows relationships between interrupt, its handler and its priority bits.

Priority notes:
When the processor is executing an ISR, the ISR is preempted if a higher priority interrupt occurs. If an interrupt occurs with the same priority as the interrupt being handled, the ISR is not preempted.

If multiple pending interrupts have the same priority, the pending interrupt with the lowest exception/vector number takes precedence.

Table 5.10 The relationship between vectors and NVIC definitions.

Vector Address	Exception Number	IRQ Number	ISR Name in Startup_TM4C123.s	NVIC Macros for Priority Register	Priority Bits
0x000000A0	40	24	TIMER2B_Handler	NVIC_PRI6_R	7 – 5
0x000000A4	41	25	COMP0_Handler	NVIC_PRI6_R	15 – 13
0x000000A8	42	26	COMP1_Handler	NVIC_PRI6_R	23 – 21
0x000000AC	43	27	COMP2_Handler	NVIC_PRI6_R	31 – 29
0x000000B0	44	28	SYSCTL_Handler	NVIC_PRI7_R	7 – 5
0x000000B4	45	29	FLASH_Handler	NVIC_PRI7_R	15 – 13
0x000000B8	46	30	GPIOF_Handler	NVIC_PRI7_R	23 – 21
0x000000BC	47	31	GPIOG_Handler	NVIC_PRI7_R	31 – 29
0x000000C0	48	32	GPIOH_Handler	NVIC_PRI8_R	7 – 5
0x000000C4	49	33	UART2_Handler	NVIC_PRI8_R	15 – 13
0x000000C8	50	34	SSI1_Handler	NVIC_PRI8_R	23 – 21
0x000000CC	51	35	TIMER3A_Handler	NVIC_PRI8_R	31 – 29
0x000000D0	52	36	TIMER3B_Handler	NVIC_PRI9_R	7 – 5
0x000000D4	53	37	I2C1_Handler	NVIC_PRI9_R	15 – 13
0x000000D8	54	38	QEI1_Handler	NVIC_PRI9_R	23 – 21
0x000000DC	55	39	CAN0_Handler	NVIC_PRI9_R	31 – 29
0x000000E0	56	40	CAN1_Handler	NVIC_PRI10_R	7 – 5
0x000000E4	57	41	CAN2_Handler	NVIC_PRI10_R	15 – 13
0x000000E8	58	42	0 (Reserved)	NVIC_PRI10_R	23 – 21
0x000000EC	59	43	HIB_Handler	NVIC_PRI10_R	31 – 29
0x000000F0	60	44	USB0_Handler	NVIC_PRI11_R	7 – 5
0x000000F4	61	45	PWM0_3_Handler	NVIC_PRI11_R	15 – 13
0x000000F8	62	46	UDMA_Handler	NVIC_PRI11_R	23 – 21
0x000000FC	63	47	UDMAERR_Handler	NVIC_PRI11_R	31 – 29

5.3.2.1 The NVIC Interrupt Priority-Level Registers

Table 5.11 The bit filed of priority levels and related interrupt priority group.

PRIn Register Bit Field	Interrupt Source	Priority Register Macros
Bits 31:29	Interrupt[IRQ] = Interrupt[4n + 3]	NVIC_PRI**n**_R NVIC→IP[4**n**] ~ NVIC→IP[4**n** + 3]
Bits 23:21	Interrupt[IRQ] = Interrupt[4n + 2]	
Bits 15:13	Interrupt[IRQ] = Interrupt[4n + 1]	
Bits 7:5	Interrupt[IRQ] = Interrupt[4n]	

Table 5.12 Most popular Priority Registers used in the TM4C123GH6PM NVIC.

Priority Register	Priority Bits 31 - 29	23 - 21	15 - 13	7 - 5	Address
NVIC_PRI0_R	GPIO Port D	GPIO Port C	GPIO Port B	GPIO Port A	0xE000E400
NVIC_PRI1_R	SSI0, Rx Tx	UART1, Rx Tx	UART0, Rx Tx	GPIO Port E	0xE000E404
NVIC_PRI2_R	PWM Gen 1	PWM Gen 0	PWM Fault	I2C0	0xE000E408
NVIC_PRI3_R	ADC Seq 1	ADC Seq 0	Quad Encoder	PWM Gen 2	0xE000E40C
NVIC_PRI4_R	Timer 0A	Watchdog	ADC Seq 3	ADC Seq 2	0xE000E410
NVIC_PRI5_R	Timer 2A	Timer 1B	Timer 1A	Timer 0B	0xE000E414
NVIC_PRI6_R	Comp 2	Comp 1	Comp 0	Timer 2B	0xE000E418
NVIC_PRI7_R	GPIO Port G	GPIO Port F	Flash Control	System Control	0xE000E41C
NVIC_PRI8_R	Timer 3A	SSI1, Rx Tx	UART2, Rx Tx	GPIO Port H	0xE000E420
NVIC_PRI9_R	CAN0	Quad Encoder 1	I2C1	Timer 3B	0xE000E424
NVIC_PRI10_R	Hibernate	Ethernet	CAN2	CAN1	0xE000E428
NVIC_PRI11_R	uDMA Error	uDMA Soft Tfr	PWM Gen 3	USB0	0xE000E42C
NVIC_SYS_PRI3_R	SysTick	PendSV	--	Debug	0xE000ED20

5.3.2.2 The NVIC Interrupt Set Enable Registers

- Table 5.13 lists most popular used peripherals and their bit numbers in the **NVIC Set Enable Registers**. Each bit is associated with a peripheral and a setting to a bit enable the selected peripheral. Each 32-bit Set Enable Register can be used enable 32 peripherals.
- During the programming process, users can directly use various macros defined for all Set Enable Registers, **NVIC_EN0_R** ~ **NVIC_EN4_R**, in their program to access them to perform the enable configurations for selected peripherals.
- Find the **interrupt number** (Tiva datasheet, Table 2-9, Interrupts) and set the bit in the NVIC_ENn_R register, where n is 0-4 and indicates a group of 32 interrupts (i.e., 0 → 0-31, 1 → 32-63, etc.). The bit set is: (interrupt number - (32*n))

Table 5.13 Relationship between each bit on interrupt enable register and related peripheral.

Enable Register	32 Enable Bits									Address
	0	1	2	3	4	5	6 - 29	30	31	
NVIC_EN0_R	PORTA	PORTB	PORTC	PORTD	PORTE	UART0	.....	PORTF	PORTG	0xE000E100
NVIC_EN1_R	PORTH	UART2	SSI1	Timer3A	Timer3B	I2C1	.....	UART6	UART7	0xE000E104
NVIC_EN2_R	...	...	...	...	I2C2	I2C3	.....	WTimer0A	WTimer0B	0xE000E108
NVIC_EN3_R	WT1A	WT1B	WT2A	WT2B	WT3A	WT3B	.....	GPIOQ2	GPIOQ3	0xE000E10C

Timer3B IC NVIC Setup

- Set up the priority for T3CCP1 interrupts using the appropriate NVIC_PRIx_R register (e.g., use priority = 1)

NVIC_PRI9_R = _____ ; // set bits 7:5 to 0b001

- Enable the interrupt at the NVIC level for T3CCP1 interrupts

NVIC_EN1_R = _____ ; // set bit 4

5.3.4 The Vector Table and Vectors Used in the TM4C123GH6PM MCU

```
__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     MemManage_Handler         ; MPU Fault Handler
                DCD     BusFault_Handler          ; Bus Fault Handler
                DCD     UsageFault_Handler        ; Usage Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     SVC_Handler               ; SVCall Handler
                DCD     DebugMon_Handler          ; Debug Monitor Handler
                DCD     0                         ; Reserved
                DCD     PendSV_Handler            ; PendSV Handler
                DCD     SysTick_Handler           ; SysTick Handler
; External Interrupts   Vector or Handler         ; IRQ#   Peripheral
                DCD     GPIOA_Handler             ;   0:    GPIO Port A
                DCD     GPIOB_Handler             ;   1:    GPIO Port B
                DCD     GPIOC_Handler             ;   2:    GPIO Port C
                DCD     GPIOD_Handler             ;   3:    GPIO Port D
                DCD     GPIOE_Handler             ;   4:    GPIO Port E
                DCD     UART0_Handler             ;   5:    UART0 Rx and Tx
                DCD     UART1_Handler             ;   6:    UART1 Rx and Tx
                DCD     SSI0_Handler              ;   7:    SSI0 Rx and Tx
                DCD     I2C0_Handler              ;   8:    I2C0 Master and Slave
                DCD     PMW0_FAULT_Handler        ;   9:    PWM Fault
                DCD     PWM0_0_Handler            ;  10:    PWM Generator 0
                DCD     PWM0_1_Handler            ;  11:    PWM Generator 1
                DCD     PWM0_2_Handler            ;  12:    PWM Generator 2
                DCD     QEI0_Handler              ;  13:    Quadrature Encoder 0
                DCD     ADC0SS0_Handler           ;  14:    ADC Sequence 0
```

Figure 5.7 The vector definitions in the Cortex-M4 microcontroller

5.3.4 The Vector Table and Vectors Used in the TM4C123GH6PM MCU

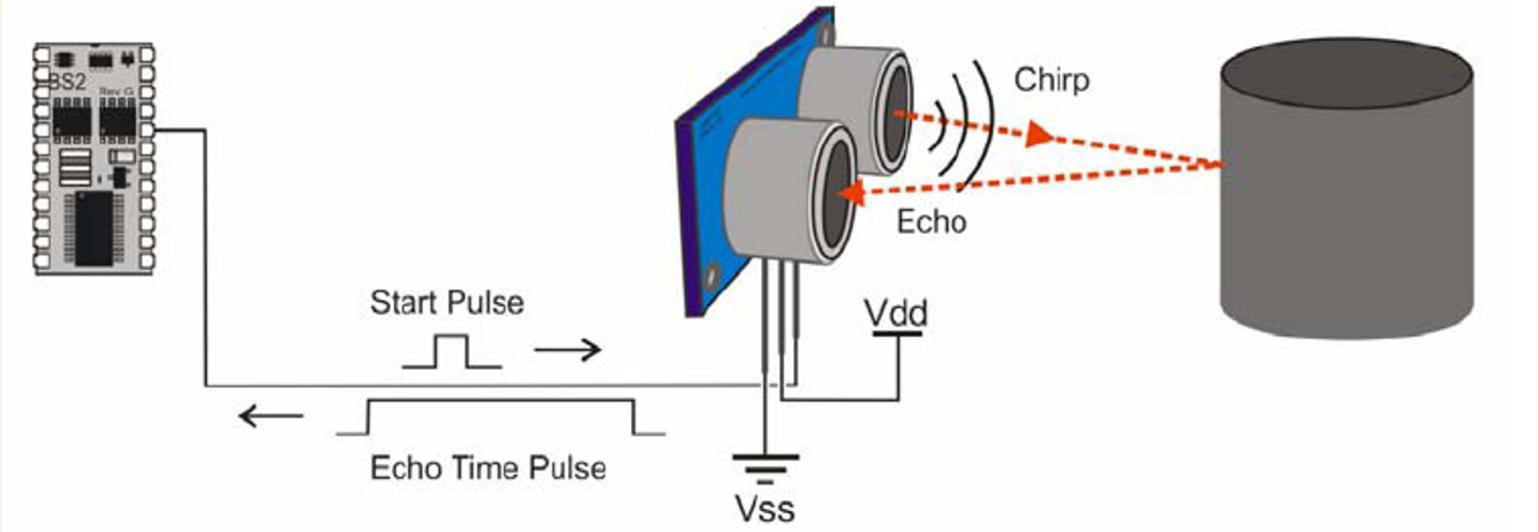
```
; External Interrupts      Vector or Handler          ; IRQ#    Peripheral
                DCD     ADC0SS1_Handler          ;  15:    ADC Sequence 1
                DCD     ADC0SS2_Handler          ;  16:    ADC Sequence 2
                DCD     ADC0SS3_Handler          ;  17:    ADC Sequence 3
                DCD     WDT0_Handler             ;  18:    Watchdog timer
                DCD     TIMER0A_Handler          ;  19:    Timer 0 subtimer A
                DCD     TIMER0B_Handler          ;  20:    Timer 0 subtimer B
                DCD     TIMER1A_Handler          ;  21:    Timer 1 subtimer A
                DCD     TIMER1B_Handler          ;  22:    Timer 1 subtimer B
                DCD     TIMER2A_Handler          ;  23:    Timer 2 subtimer A
                DCD     TIMER2B_Handler          ;  24:    Timer 2 subtimer B
                DCD     COMP0_Handler            ;  25:    Analog Comparator 0
                DCD     COMP1_Handler            ;  26:    Analog Comparator 1
                DCD     COMP2_Handler            ;  27:    Analog Comparator 2
                DCD     SYSCTL_Handler           ;  28:    System Control (PLL, OSC, BO)
                DCD     FLASH_Handler            ;  29:    FLASH Control
                DCD     GPIOF_Handler            ;  30:    GPIO Port F
                DCD     GPIOG_Handler            ;  31:    GPIO Port G
                DCD     GPIOH_Handler            ;  32:    GPIO Port H
                DCD     UART2_Handler            ;  33:    UART2 Rx and Tx
                DCD     SSI1_Handler             ;  34:    SSI1 Rx and Tx
                DCD     TIMER3A_Handler          ;  35:    Timer 3 subtimer A
                DCD     TIMER3B_Handler          ;  36:    Timer 3 subtimer B
                DCD     I2C1_Handler             ;  37:    I2C1 Master and Slave
                DCD     QEI1_Handler             ;  38:    Quadrature Encoder 1
                DCD     CAN0_Handler             ;  39:    CAN0
                DCD     CAN1_Handler             ;  40:    CAN1
                DCD     CAN2_Handler             ;  41:    CAN2
                DCD     0                        ;  42:    Reserved
                DCD     HIB_Handler              ;  43:    Hibernate
                DCD     USB0_Handler             ;  44:    USB0
```

Figure 5.7 The vector definitions in the Cortex-M4 microcontroller

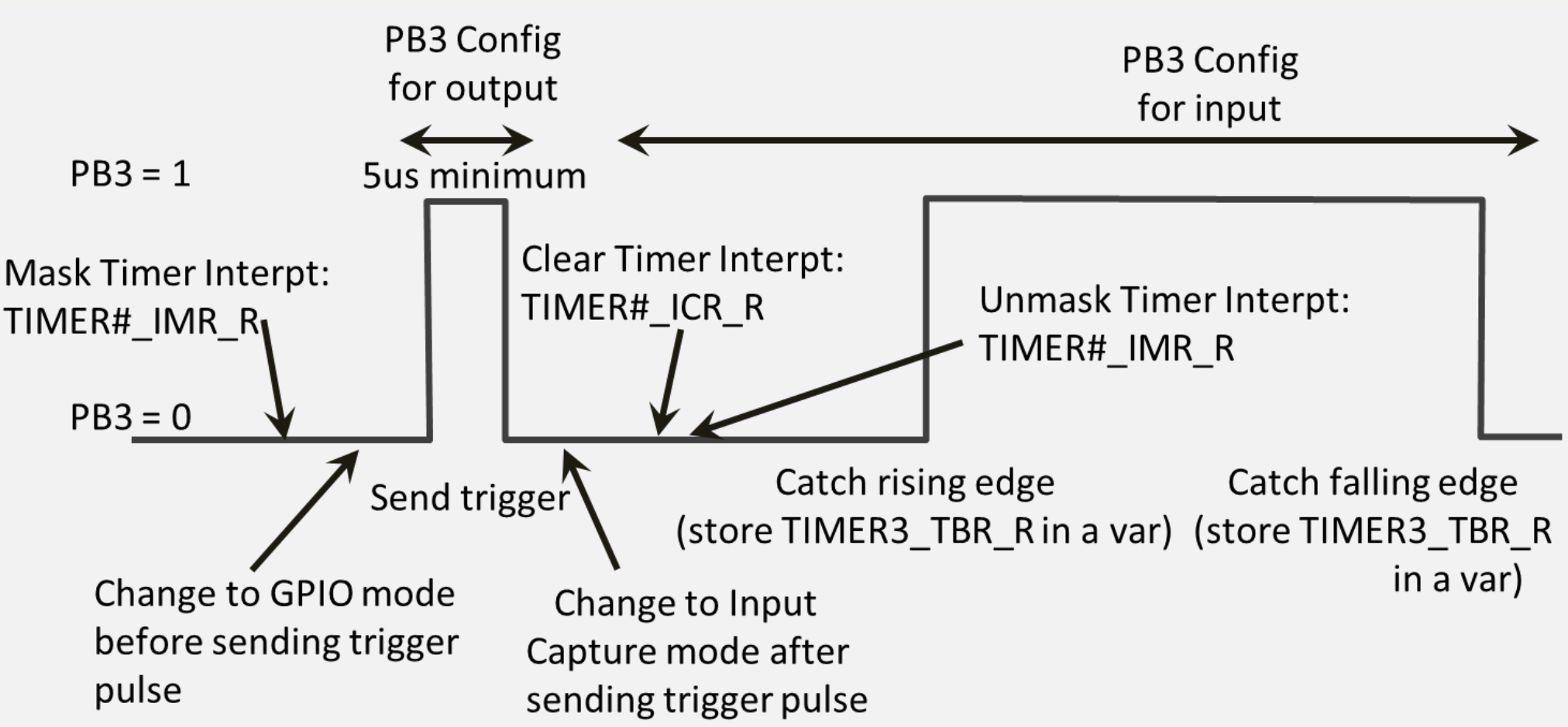
PING Lab General Idea of Programming

General idea:

- Generate a pulse to activate the PING))) sensor
- Configure Timer3B for input capture
- Capture the time of rising edge event
- Capture the time of falling edge event
- Calculate time difference and then distance to object



PING Lab General Idea of Programming



Remember only one pin (i.e PB3) used to communicate with the PING))) sensor

Input Capture ISR Example

```
volatile unsigned long last_time = 0;
volatile unsigned long current_time = 0;
volatile int update_flag = 0;

// ISR: Record the current event time
void TIMER1A_Handler(void)
{
    last_time = current_time;
    current_time = TIMER1_TAR_R;
    update_flag = 1;
}
```

Recall: We have to declare “volatile” for global variables changed by ISRs.

Overflow (Counting Up)

Are we concerned with overflow in the calculation?

```
time_diff = current_time - last_time;
```

What happens if `current_time` is ***less*** than `last_time`?

If we use the prescaler register as an extension with the 16-bit timer register to make a 24-bit timer register:

Overflow: Change from 0xFFFFFF to 0x000000

0xFFFFFF is 16777216 in decimal, it takes over 1 second for 16MHz clock to overflow the timer. Considering the time scale of the PING))) sensor readings, we would never have more than 1 overflow at a time. One overflow can be accounted for easily.

Overflow (Counting Up)

```
unsigned long time_diff;

overflow = (current_time < last_time);
time_diff = ((unsigned long)overflow<<24)
          + current_time - last_time;
update_flag = 0;
```

- Overflow occurred if current_time < last_time
- For each overflow, increase time_diff by 16,777,216 (2^{24})
- You have to use long integer which is 32-bit (0 to 2^{32}-1)