Tecnológico de Monterrey

Microcontrollers

Professor: Dr. Gilberto Ochoa Ruiz

Topic 3: ARM Basic Programming - Timers

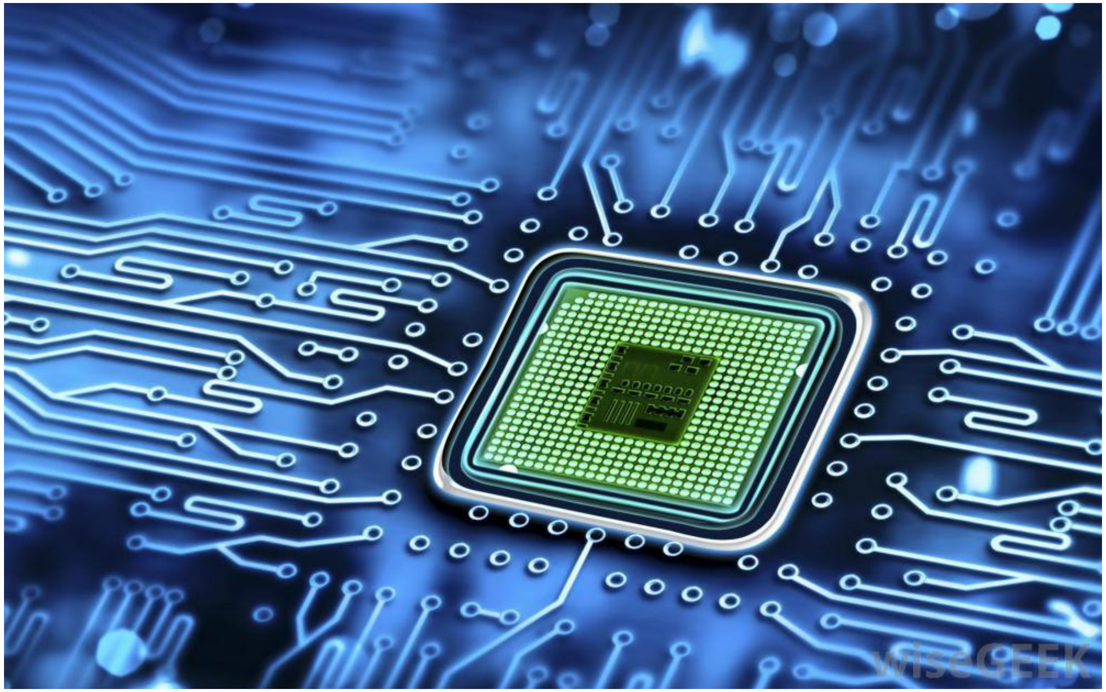
EIAD-204

Wednesday

April 1st 2020

6h30 – 9h30 PM

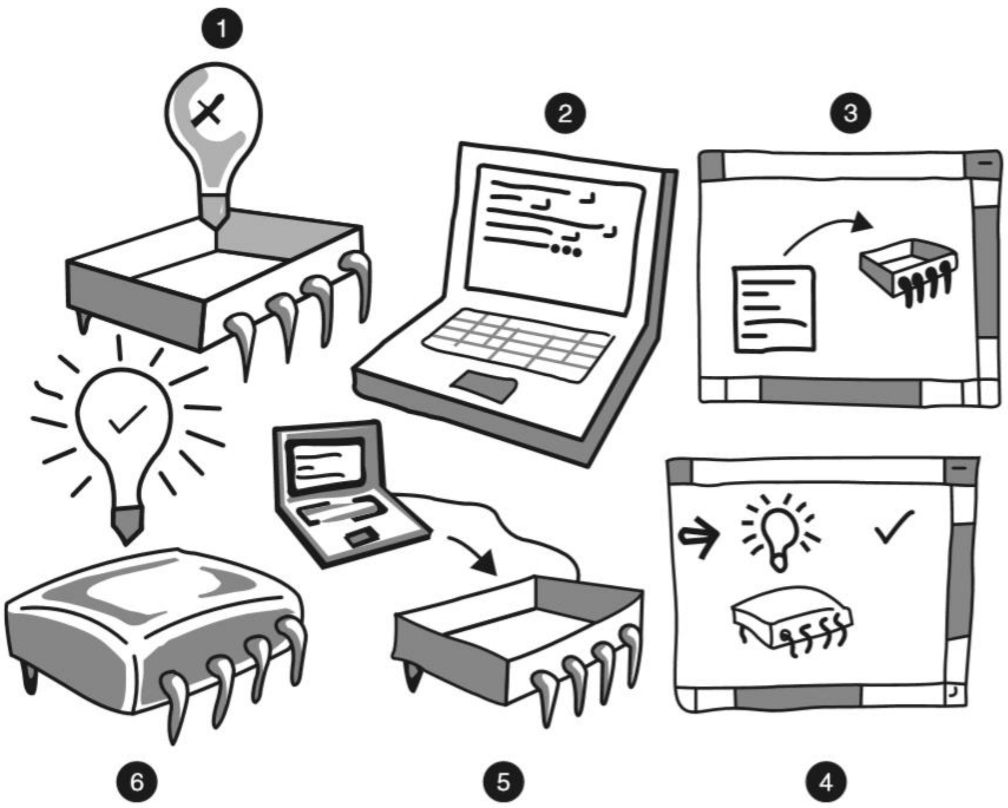
Guadalajara, Mexico



Basic Programming Techniques - Timers

Outline

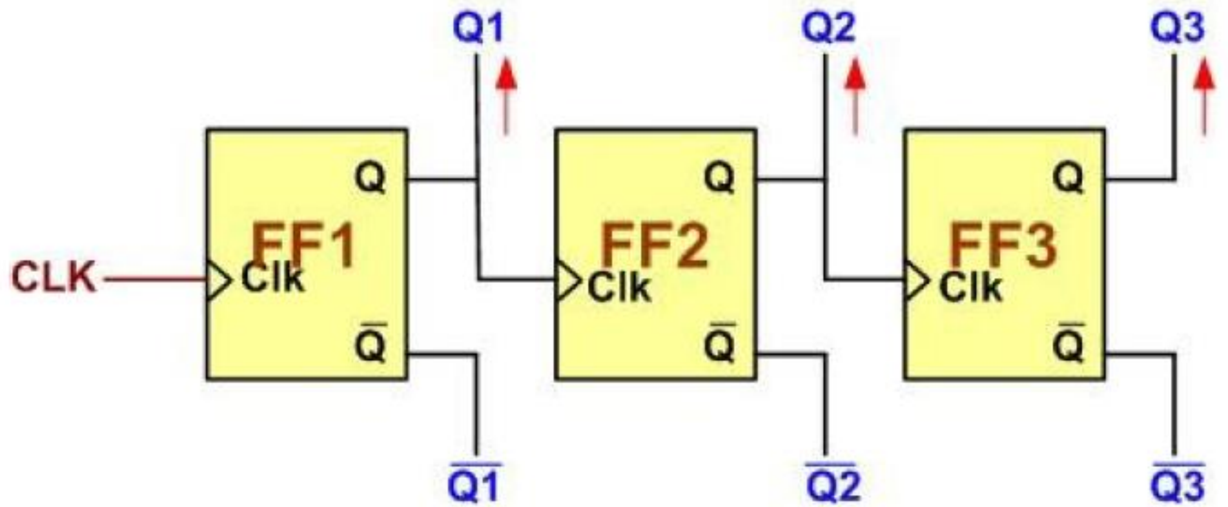
- Introduction to timers
- System Tick Timer
- Delay Generation
- Output Compare
- Edge Time Capturing



Introduction to counters and timers

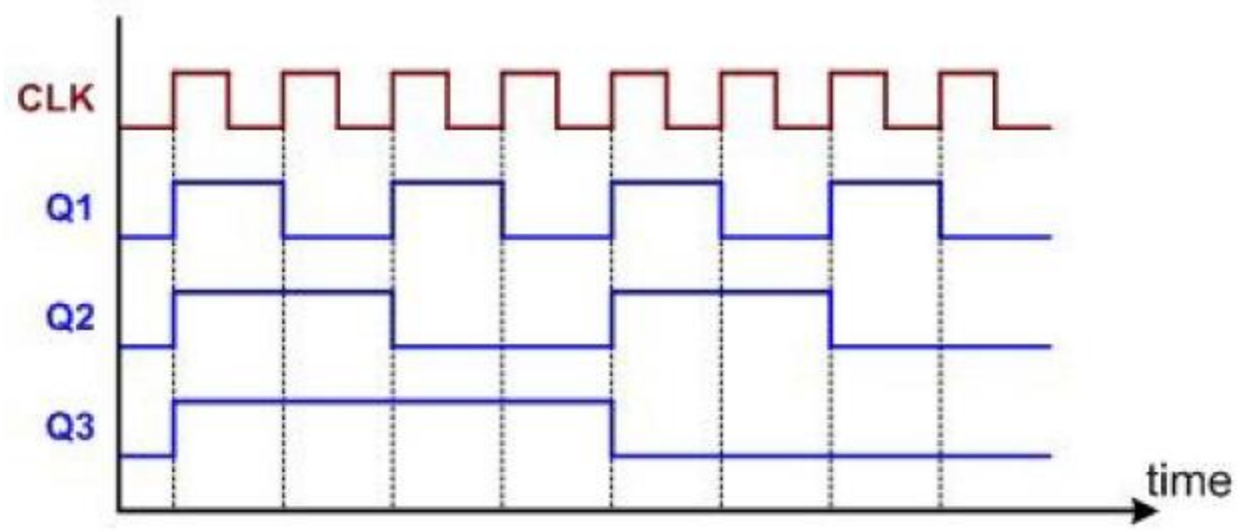
In the digital design course you connected many flip flops (FFs) together to create up counter/down counter.

For example, connecting 3 FFs together we can count up to 7 (000-111 in binary). This is called *3-bit counter*.



Introduction to counters and timers

The same way, to create a 4-bit counter (counting up to 15, or 0000-1111 in binary) we need 4 FFs. For 16-bit counter, we need 16 FFs and it counts up to $2^{16} – 1$



Introduction to counters and timers

Regarding the previous tome diagram, we can notice the following points:

1) The Q outputs give the down counter.

2) The (Q not) outputs give us up counter.

3) The frequency on Q3 is of the Clock fed to FF1.

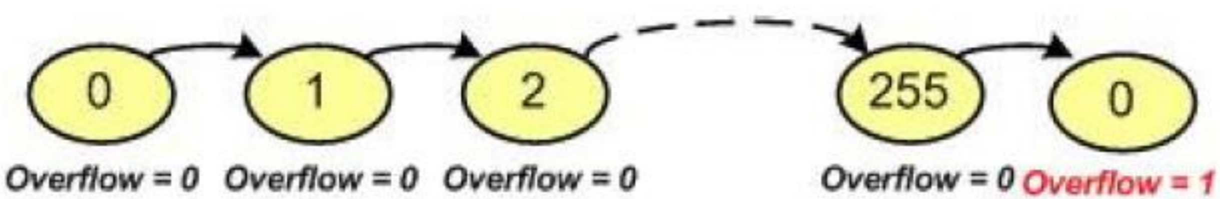
4) We can use the circuit to divide the clock frequency.

5) We can use the circuit in count the number of pulses fed to CLK pin of FF1.

Introduction to counters and timers

An up counter begins counting from 0 and its value increases on each clock until it reaches its maximum value.

Then, it overflows and rolls over to zero in the next clock. The following figure shows the stages which an 8-bit counter goes through.

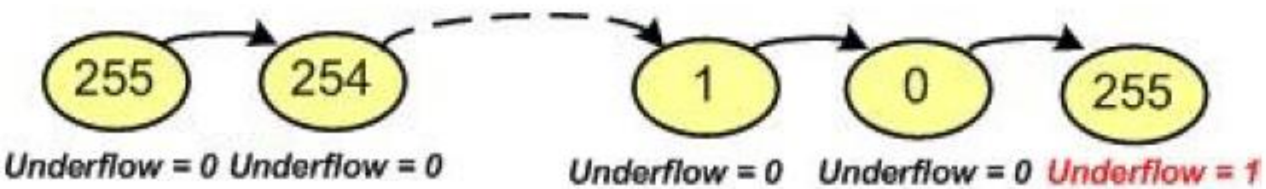


Introduction to counters and timers

A down counter begins counting from its maximum value and decreases on each clock until it reaches to 0.

Then, it underflows and rolls over to its maximum value in the next clock.

The following figure shows the stages which an 8-bit down counter goes through.



Counter Usages

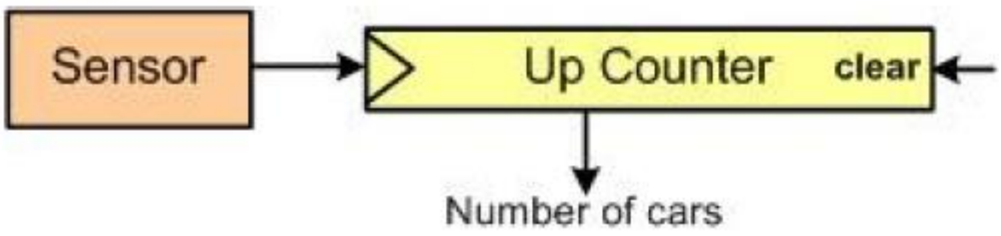
Counters have different usages. Some of them are:

1. Counting events
2. Making delays (Using Counter as a Timer)
3. Measuring the time between 2 events

Counter Usages → Counting Events

You might need to count the number of cars going through a street or the number of spaghetti packages which produced in a factory.

To do so, you can connect the output of a sensor to a counter, as shown in the following figure.

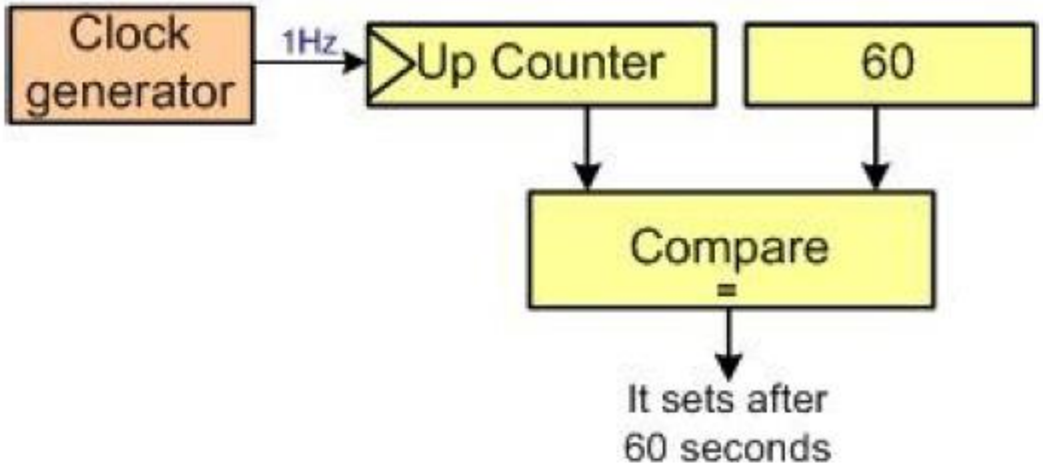


Basic Programming Techniques - Timers

Counter Usages → Counting Delays

While controlling devices, it is a common practice to start or terminate a task when a desired amount of time elapsed.

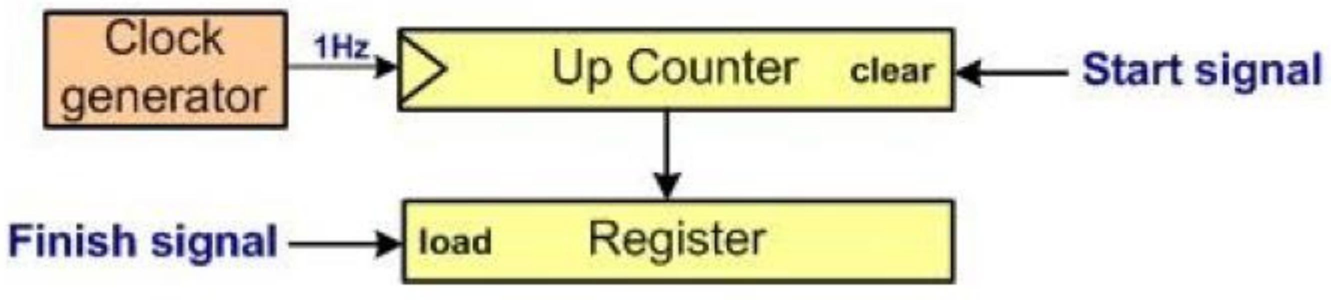
For example, a washing machine or an oven do each task for a determined amount of time. To do timing, we can connect a clock generator to a counter, and wait until a desired amount of time elapses.



Counter Usages → Measuring Time between events

You might need to measure the time between 2 events. For example, the amount of time it takes a marathon runner to go from the start to the finish point.

In such cases we can use a circuit similar to the following:



Time Capturing

Counters and Timers in microcontrollers

Nowadays, all the microcontrollers come with on-chip Timer/Counter.

If the clock to the Timer comes from internal source such as PLL, XTAL, and RC, then it is called a *Timer*.

If the clock source comes from external source, such as pulses fed to the CPU pin, then it is called a *Counter*.

By Counter it is meant event counter since it counts the event happening outside the CPU. In many microcontrollers, the Timers can be used as Timer or Counter.

System Tick Timer

Every ARM Cortex-M comes with a System tick timer.

System tick timer allows the system to initiate an action on a periodic basis. This action is performed internally at a fixed rate without external signal.

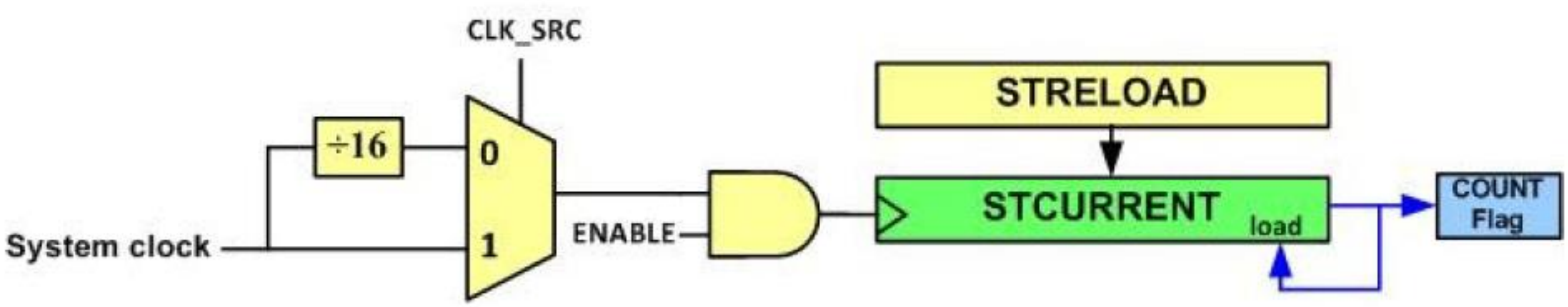
For example, in a given application we can use SysTick to read a sensor every 200 msec.

SysTick is used widely by operating systems so that the system software may interrupt the application software periodically (often 10 ms interval) to monitor and control the system operations.

System Tick Timer

The SysTick is a 24-bit down counter driven by either the system clock or the internal oscillator. It counts down from an initial value to 0.

When it reaches 0, in the next clock, it underflows and it raises a flag called COUNT and reloads the initial value and starts all over.



Basic Programming Techniques - Timers

System Tick Timer

The down counter is named as STCURRENT (SysTick->VAL) in Freescale ARM products.

The counter can receive clock from 2 different sources: the System clock (the clock which the CPU and all the peripherals work with it) or the external clock provided to the PIOSC pin.

The clock source is chosen using the CLK_SRC bit of STCTRL (SysTick->CTRL) register.

The clock is ANDed with the ENABLE bit of STCTRL register. So, it counts down when the ENABLE bit is set.

Basic Programming Techniques - Timers

System Tick Timer

	D31	.....	D17	D16	D15	...	D3	D2	D1	D0	
STCTRL:	Reserved			COUNT	Reserved			CLK_SRC	INTEN	ENABLE	0x010

SysTick->CTRL

bit	Name	Description
0	ENABLE	Enable (0: the counter is disabled, 1: enables SysTick to begin counting down)
1	INTEN	Interrupt Enable 0: Interrupt generation is disabled, 1: when SysTick counts to 0 an interrupt is generated
2	CLK_SRC	Clock Source 0: System clock divided by 16 1: System clock
16	COUNT	Count Flag 0: the SysTick has not counted down to zero since the last time this bit was read 1: the SysTick has counted down to zero *Note: this flag is cleared by reading the STRCTL or writing to STCURRENT register.*

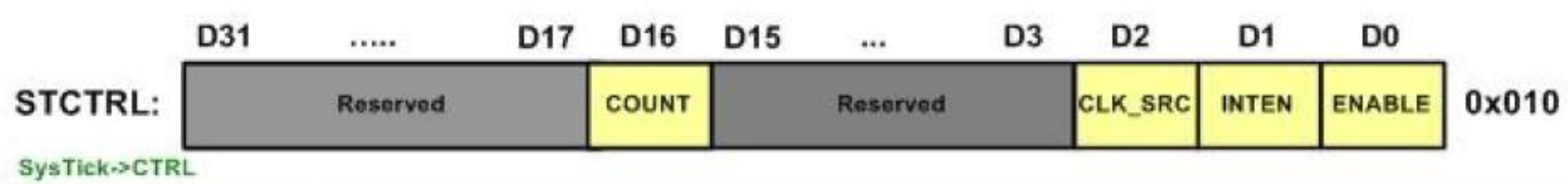
System Tick Timer Control Register

SysTick Registers

Next, we will describe the SysTick registers. There are three registers in the SysTick module:

SysTick Control and Status register, SysTick Reload Value register, and SysTick Current Value register.

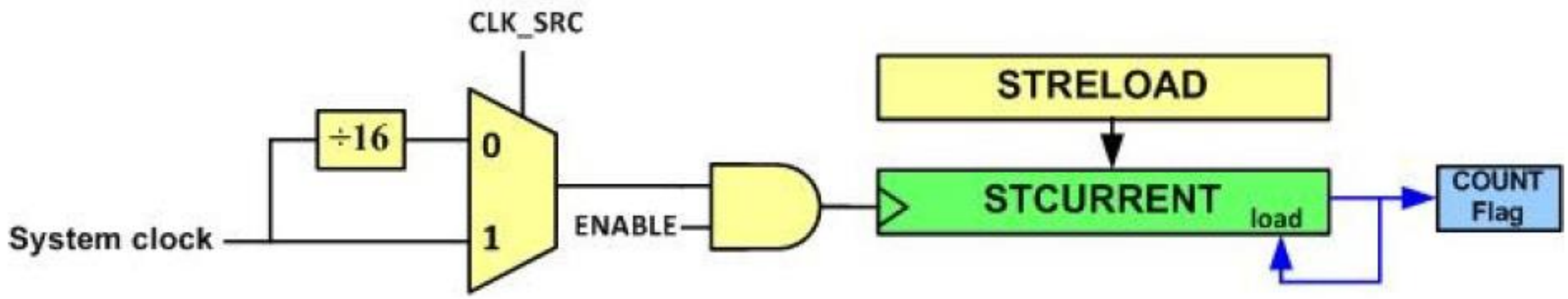
The STCTRL (SysTick Control and Status) register is located at 0xE000E010. We use it to start the SysTick counter among other things.



SysTick Registers → Control Register

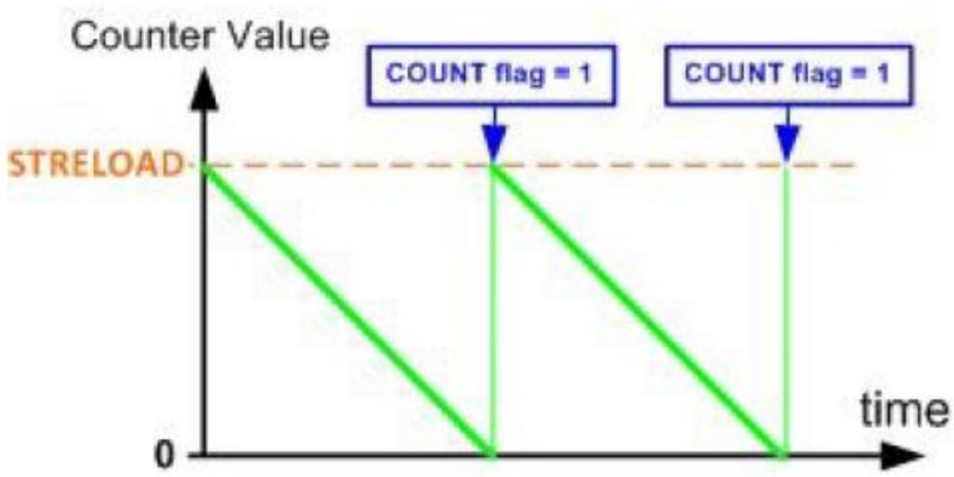
ENABLE (D0): enables or disables the counter.

When the *ENABLE* bit is set the counter initializes the STCURRENT with the value of the *STRELOAD* register and it counts down until it reaches to zero.



SysTick Registers → Control Register

Then, in the next clock, it underflows which sets the *COUNT* Flag to high and the counter reloads the STCURRENT with the value of the *STRELOAD* register and then the process is repeated.



System Tick Counting

Counters and Timers in microcontrollers

INTEN (Interrupt Enable, D1): If INTEN=1, an interrupt occurs when the COUNT flag is set.

CLK_SRC (Clock Source D2): We have the choice of clock coming from System clock or Precision Internal Oscillator (PIOSC). If CLK_SRC=0 then the clock comes from PIOSC/4. If CLK_SRC=1, then the system clock provides the clock source to SysTick down counter.

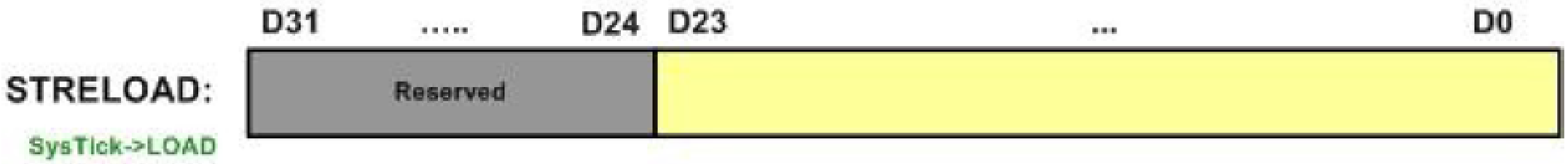
COUNT (D16): Counter counts down from the initial value and when it reaches 0, in the next clock it underflows and the COUNT flag is set high The flag remains high until it is cleared by software.

SysTick Reload Value Register (STRELOAD)

The STRELOAD (SysTick Reload Value) register is located at 0xE000E014.

This is used to program the start value of SysTick down counter, the STCURRENT register.

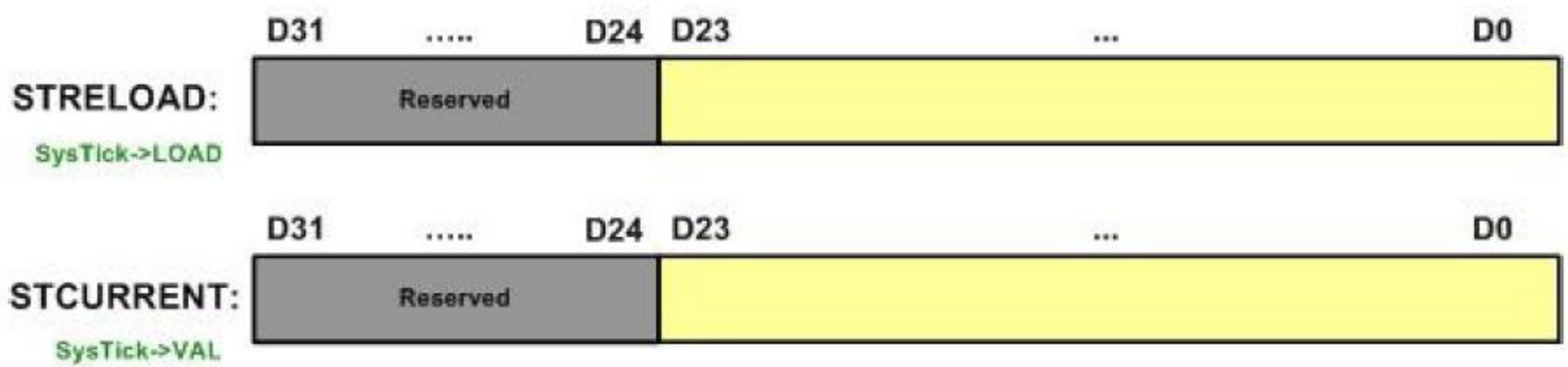
The STRELOAD should contain the value N – 1 for the COUNT to fire every N clock cycles because the counter counts down to 0.



SysTick Reload Value Register (STRELOAD)

For example, if we need 1000 clocks of interval, then we make STRELOAD = 999.

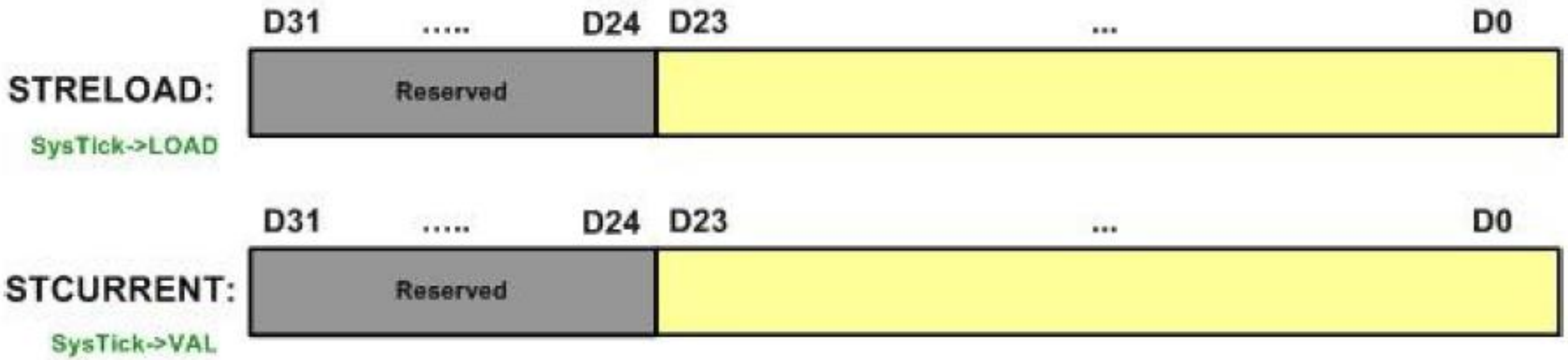
Although this is a 32-bit register, only the lower 24 bits are used. That means the highest value that can be loaded into this register is 0xFFFFFF or 16,777,216 decimal.



SysTick Reload Value Register → Example 1

The following example loads the initial value to the maximum and dumps the current value of the SysTick on LEDs of PORTB as it counts down.

The value of STCURRENT (SysTick->VAL) is shifted 4 places to the right so that the most significant bit is aligned with PTB19, which is connected to the green LED.



SysTick Reload Value Register → Example 1

SysTick is configured to use default system clock at 41.94 MHz.

The STRELOAD (SysTick->LOAD) register has 24 bits and is set to the maximal value. So the counter has the frequency of

$$\frac{41{,}940{,}000\ \text{Hz}}{2^{24}} = \frac{41{,}940{,}000\ \text{Hz}}{16{,}777{,}216} \approx 2.5\ \text{Hz}$$

And that is the frequency of the green LED. The red LED is connected to PTB18 therefore runs twice as fast as the green LED at PTB19.

SysTick Reload Value Register → Example 1

```
/* Toggling LEDs using SysTick counter

This program let the SysTick counter run freely and
dumps the counter values to the tri-color LEDs
continuously.

The counter value is shifted 4 places to the right so
that the changes of LEDs will be slow enough to be
visible.

SysTick counter has 24 bits.
The red LED is connected to PTB18.
The green LED is connected to PTB19.
*/
```

SysTick Reload Value Register → Example 1

```
#include <MKL25Z4.H>
int main (void) {
int c;
SIM->SCGC5 |= 0x400; /* enable clock to Port B */
PORTB->PCR[18] = 0x100; /* make PTB18 pin as GPIO */
PORTB->PCR[19] = 0x100; /* make PTB19 pin as GPIO */
PTB->PDDR |= 0xC0000; /* make PTB18, 19 as output pin */
/* Configure SysTick */
SysTick->LOAD = 0xFFFFFF; /* reload reg. with max value */
SysTick->CTRL = 5; /* enable it, no interrupt, use system
clock */
while (1) {
c = SysTick->VAL; /* read current value of down counter */
PTB->PDOR = c >> 4; /* line up counter MSB with LED */
}
```

SysTick Reload Value Register → Example 2

Using the System Tick timer, write a function that makes a delay of 1 ms. Assume sysclk = 41.94 MHz.

Solution:

From the equation delay = (N + 1) / sysclk

(N + 1) = delay × sysclk = 0.001 sec × 41.94 MHz = 41,940 ==> N = 41,940 – 1 = 41939

```
void delay1ms(void) {
SysTick->LOAD = 41939;
SysTick->CTRL = 0x5; /* Enable the timer and choose sysclk as the clock source */
while((SysTick->CTRL & 0x10000) == 0) /* wait until the COUNT flag is set */
{ }
SysTick->CTRL = 0; /* Stop the timer (Enable = 0) */
}
```

SysTick Reload Value Register → Example 2

The next program uses the SysTick to toggle the PTB18 every 200 milliseconds.

We need the RELOAD value of 8,387,999 since 0.200 sec * 41.94 MHz = 8,388,000.

We assume the system clock is 41.94 MHz. Notice, every 8,388,000 clocks the down counter reaches 0, and COUNT flag is raised.

Then the RELOAD register is loaded with 8,388,000 automatically. The COUNT flag is clear when the STCTRL (SysTick->CTRL) register is read.

SysTick Reload Value Register → Example 2

```
#include <MKL25Z4.H>
int main (void) {
SIM->SCGC5 |= 0x0400; /* enable clock to Port B */
PORTB->PCR[18] = 0x100; /* make PTB18 pin as GPIO */
PTB->PDDR |= 0x040000; /* make PTB18 as output pin */
/* Configure SysTick */
SysTick->LOAD = 8388000 - 1; /* reload with number of
clocks per 200 ms */
SysTick->CTRL = 5; /* enable it, no interrupt, use system
clock */
while (1)
{
if (SysTick->CTRL & 0x10000) /* if COUNT flag is set */
PTB->PTOR = 0x040000; /* toggle red LED */
}
}
```

SysTick Reload Value Register → Example 2

```
/* Toggling green LED using SysTick delay

* This program uses SysTick to generate one second delay to toggle the green LED.

* System clock is running at 41.94 MHz. SysTick is configure to count down from 41939 to give a 1 ms delay.

• For every 1000 delays (1 ms * 1000 = 1 sec), toggle the green LED once. The green LED is connected to PTB19.

*/
```

SysTick Reload Value Register → Example 2

```
include <MKL25Z4.H>

int main (void) {

void delayMs(int n);

SIM->SCGC5 |= 0x400; /* enable clock to Port B */
PORTB->PCR[19] = 0x100; /* make PTB19 pin as GPIO */
PTB->PDDR |= 0x080000; /* make PTB19 as output pin */

while (1) {
delayMs(1000); /* delay 1000 ms */
PTB->PTOR = 0x080000; /* toggle green LED */
}
}
```

SysTick Reload Value Register → Example 2

```
void delayMs(int n) {
int i;
SysTick->LOAD = 41940 - 1;
SysTick->CTRL = 0x5; /* Enable the timer and
choose sysclk as the clock source */

for(i = 0; i < n; i++) {
while((SysTick->CTRL & 0x10000) == 0) /* wait
until the COUTN flag is set */
{ }
}
SysTick->CTRL = 0; /* Stop the timer (Enable =
0) */
}
```

Delay Generation with Timers

In this section we examine the timers for KL25Z ARM chip.

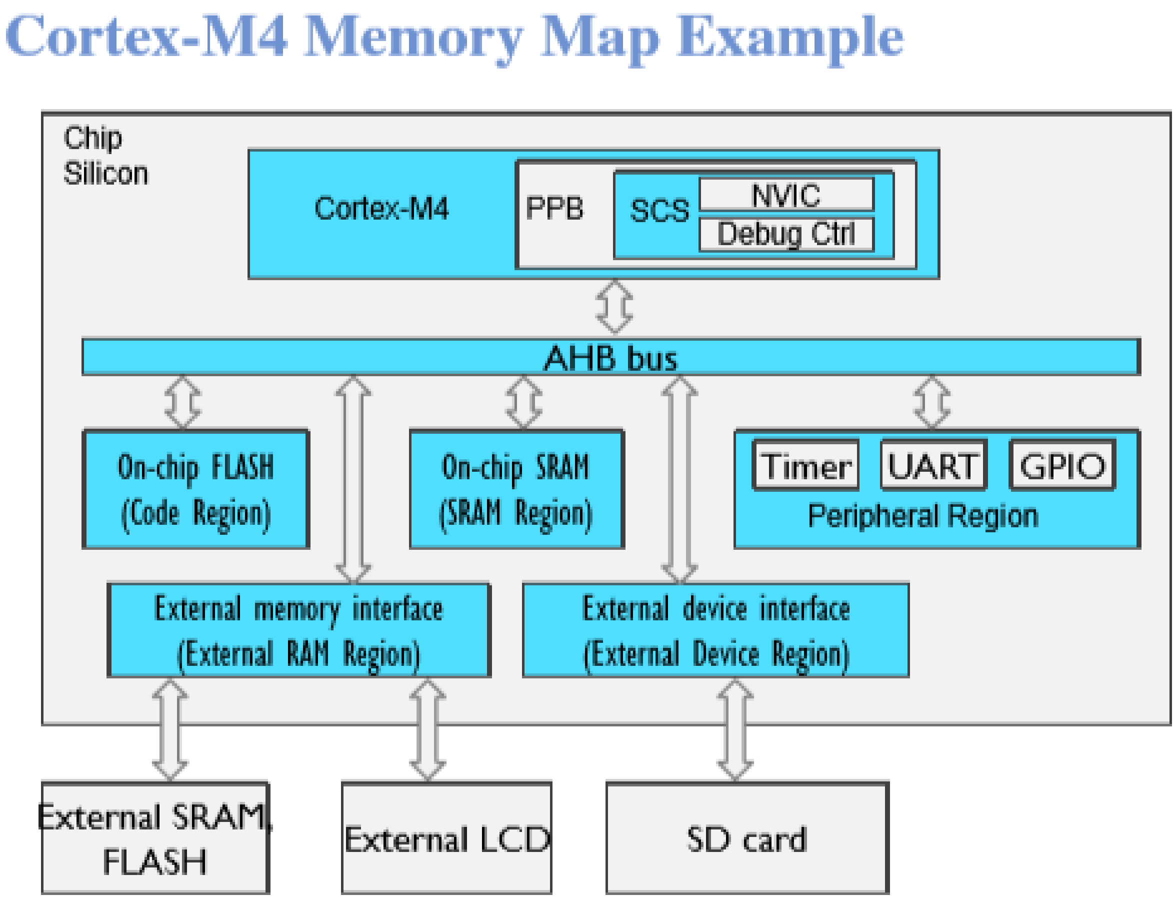
We will use KL25Z timer to create time delay on FRDM board.

Timer Registers

In KL25Z microcontrollers, the timers are called *Timer/ PWM Module (TPM)*.

There are 3 Timer Modules in the KL25Z, each supporting up to 6 channels. The Timer modules support Output Compare, Input capture, and PWM.

Delay Generation with Timers



Delay Generation with Timers

The Timer Modules in KL25Z are designated as TPMx in which x = 0, 1, or 2.

In other words, there are TPM0, TPM1, and TPM2.

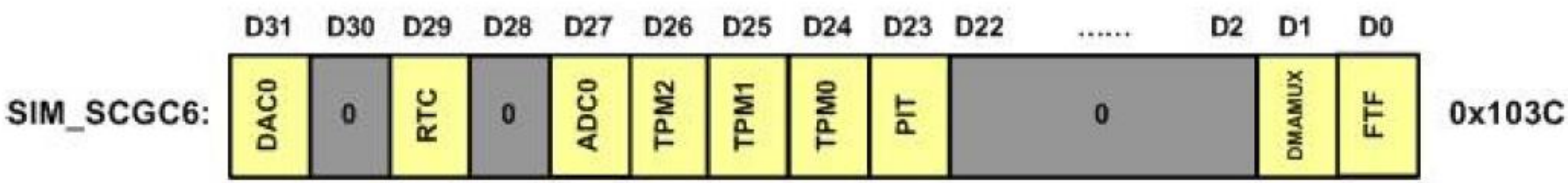
The following shows the base addresses for the Timer modules:

- TPM0 base: 0x4003 8000
- TPM1 base: 0x4003 9000
- TPM2 base: 0x4003 A000

Enabling Clock to TMPx

Before we can use any of the Timer Modules, we must enable the clock to it.

This is done with the System Clock Gating Register 6 (SIM_SCGC6 register), as shown below:



bit	Name	Description
24	TPM0	TPM0 clock gate control (0: clock disabled, 1: clock enabled)
25	TPM1	TPM1 clock gate control (0: clock disabled, 1: clock enabled)
26	TPM2	TPM2 clock gate control (0: clock disabled, 1: clock enabled)

Enabling Clock to TMPx

The SIM_SCGC6 is part of the SIM (System Integration Module). The details of SIM are shown in Chapter 12 of KL25Z reference manual.

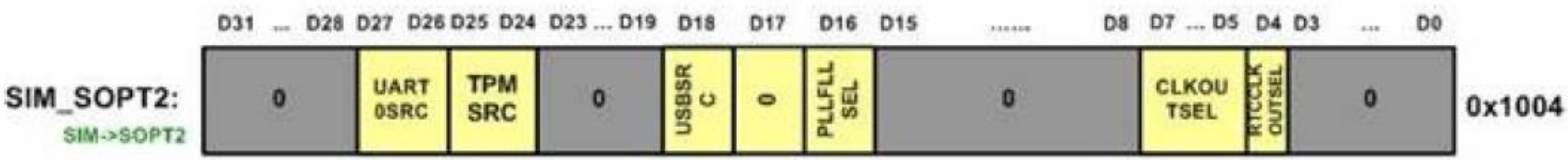
Just like GPIO and UART, we must enable the clock to TPM0 –TPM2 modules before we can use them.

Notice, in SIM_SCGC6, bit D24 is for TPM0 module, bit D25 is for TPM1 module, and bit D26 is for TPM2 module.

This clock is used to operate the timer module circuit. The core of the timer is a counter, which receives a different clock.

Enable Timer Counter Clock

The clock that drives the timer counter is selected by the TPMSRC bits and the PLLFLLSEL bit of SIM_SOPT2 register in System Integration Module.



TPM clock source select selects the clock source for the TPM counter clock

TPMSRC	Selected Clock
00	Clock disabled
01	MCGFLLCLK clock or MCGFLLCLK/2
10	OSCERCLK clock
11	MCGIRCLK clock

Basic Programming Techniques - Timers

Enable Timer Counter Clock

Upon reset, the timer counter clock is disabled. The possible clock sources are MCGFLLCLK, MCGPLLCLK/2, OSCERCLK and MCGIRCLK (KL25Z Ref Man Section 5.7.5).

The clock source availability depends on the configuration of the MCG (Multiple Clock Generation) Module.

The Keil MDK-ARM v5 supports three MCG configurations in the startup code of system_MKL25Z4.c file and is default to Mode 0.

The available clock sources and frequency for FRDM-MKL25Z are shown in the table next

Enable Timer Counter Clock

The available clock sources and frequency for FRDM-MKL25Z are shown in the table next

MCG Mode	MCGFLLCLK	MCGPLLCLK/2	OSCERCLK	MCGIRCLK
0	41.94 MHz	N.A.	N.A.	32.768 kHz
1	N.A.	48.0 MHz	8.0 MHz	N.A.
2	N.A.		8.0 MHz	N.A.
Note: *N.A.: Not Applicable*				

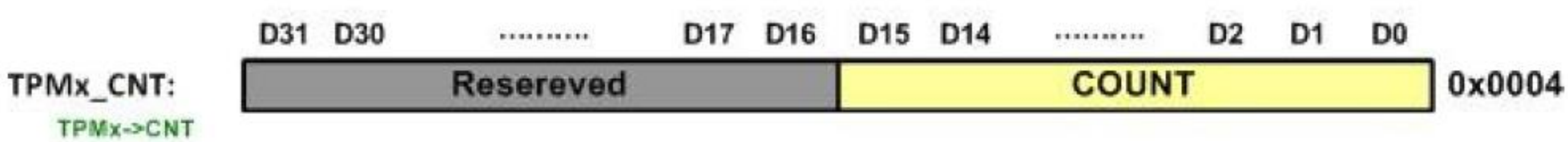
Clock Sources in FRDM-MKL25Z

TPM COUNT Register (TPMx_CNT)

Each of the Timer modules has a 16-bit counter. It is called TPMx_CNT in which x = 0, 1, or 2.

That means we have TPM0_CNT, TPM1_CNT, and TPM2_CNT. When the clock is fed to TPMx_CNT, it keeps counting up (or counting down).

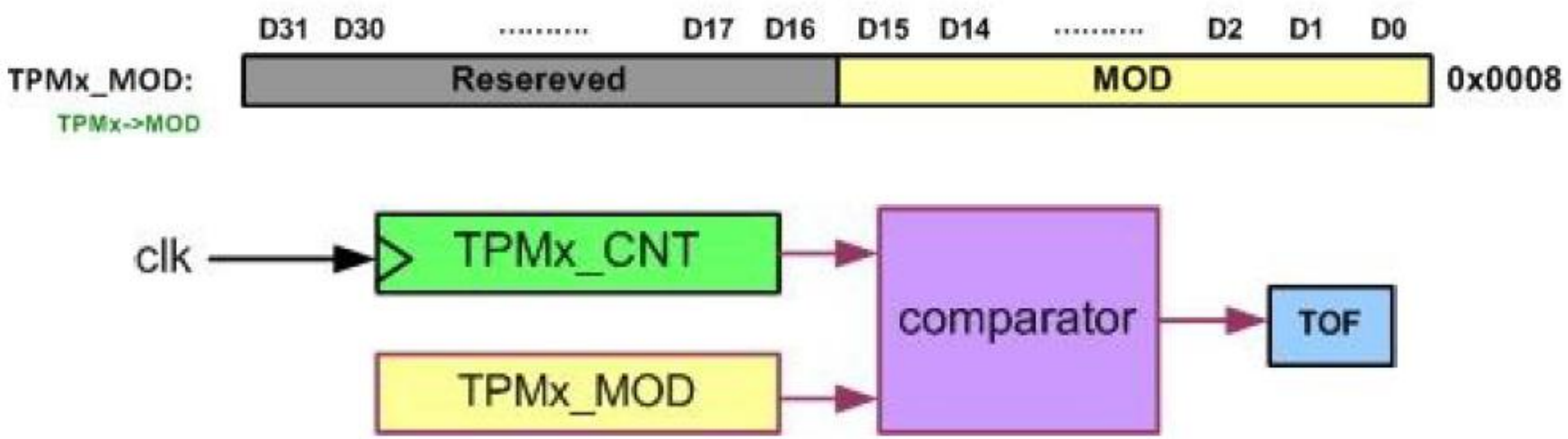
TPMx_CNT is a 16-bit counter register. Although the TPMx_CNT is a 32-bit register only 16-bits are used. We can read its content as it counts. Upon Reset TPMx_CNT=0000.



TPM Modulo Register (TPMx_MOD)

Each TPM has a Modulo (TPMx_MOD) register.

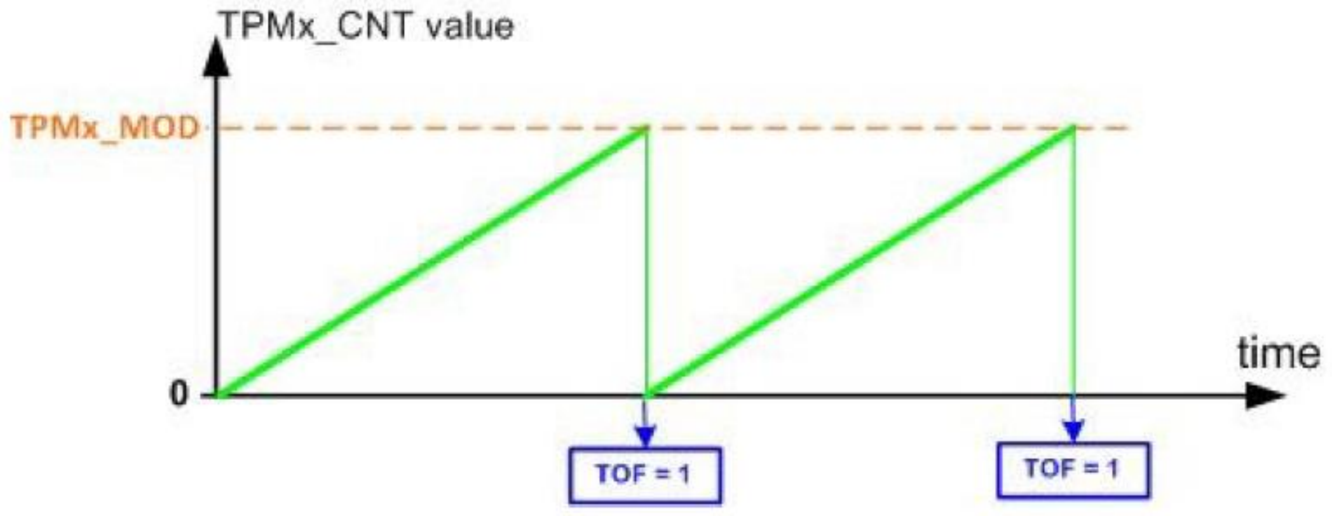
It is a 16-bit register whose
value is continuously compared with the TPMx_CNT register.



TPM Modulo Register (TPMx_MOD)

When TPMx_CNT counter register is counting up, it is compared with the contents of this register.

When the contents of TPMx_CNT counter and TPMx_MOD register are equal, the TOF flag (Timer Over Flow flag) goes up indicating there is a match



TPM Modulo Register (TPMx_MOD)

The D7 bit of TPMx_SC (TPMx Status Control) register belongs to the TOF flag, as we will see soon.

Although the Timer Modulo is a 32-bit register, only the lower 16 bits are used. We can initialize this register with values ranging from 0x0000 to 0xFFFF.

It must be noted that upon Reset TPMx_MOD=0xFFFF.

That means, if we do not initialize the TPMx_MOD register, the TPMx_CNT keeps counting up to 0xFFFF and rolls over to zero when it reaches 0xFFFF.

TPMx Status Control (TPMx_SC) register

Each of the TPMx has its own Status Control register. It is called TPMx_SC in which x = 0, 1, or 2.

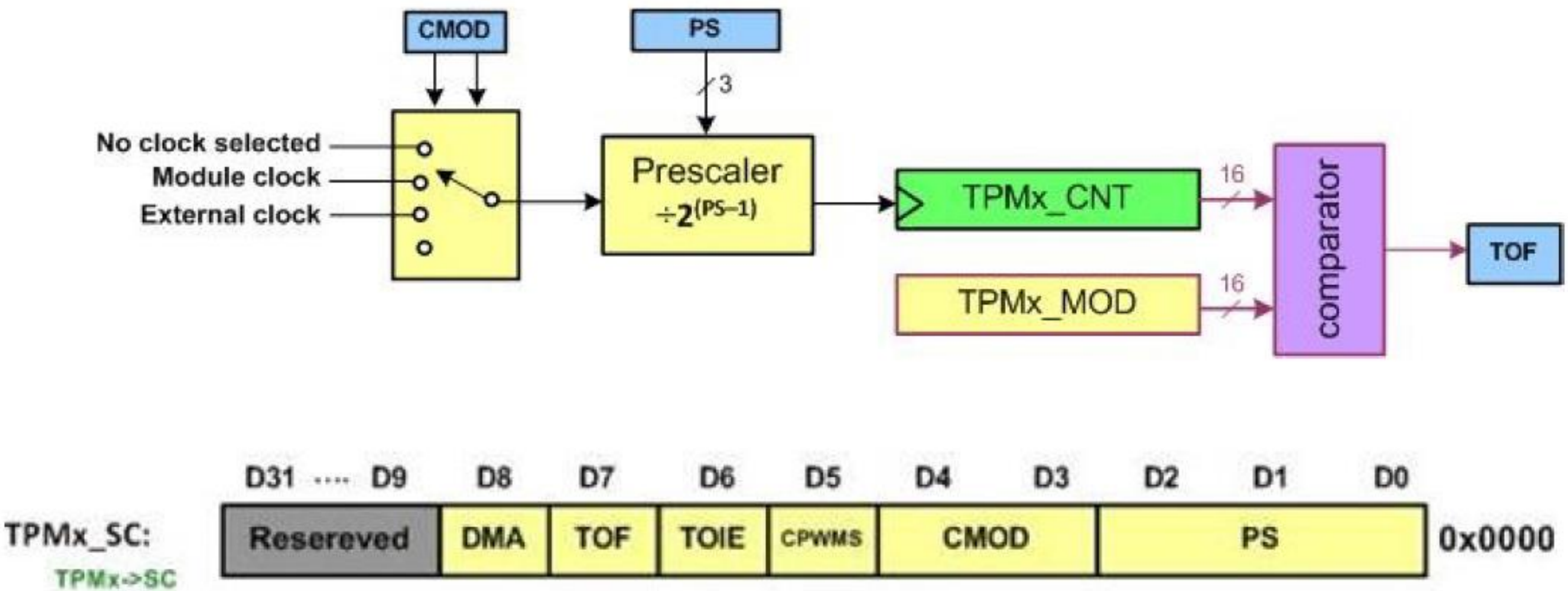
During the initialization of the timers we must disable them. Modifying the configurations of a running timer may cause unpredictable results.

We use D4:D3 (CMOD) bits of TPMx_SC (TPM Status Control) register to disable or enable the Counter.

This must be done in addition to allowing clock to the TPMx module using the SIM_SCGC6 register and selecting the clock source for timer counter using SIM_SOPT2 register.

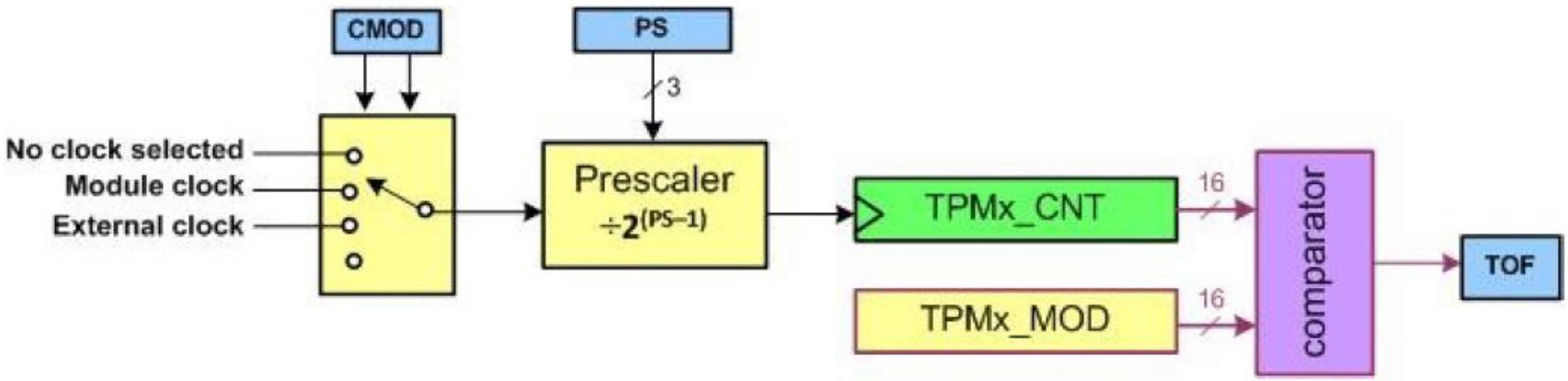
TPMx Status Control (TPMx_SC) register

Among other important bits of this register are TOF (Timer Over Flow flag), PS (Prescaler), and TOIE. The TOIE (Timer Overflow Interrupt Enable) is covered when we see interrupts



TPMx Status Control (TPMx_SC) register

<table>
<tr><th>Field</th><th>Bits</th><th>Description</th></tr>
<tr><td>PS</td><td>0–2</td><td>In the prescaler, the clock is divided by 2^{PS}.
<table><tr><th>PS value</th><th>000</th><th>001</th><th>010</th><th>011</th><th>100</th><th>101</th><th>110</th><th>111</th></tr><tr><td>Division</td><td>1</td><td>2</td><td>4</td><td>8</td><td>16</td><td>32</td><td>64</td><td>128</td></tr></table></td></tr>
</table>



Basic Programming Techniques - Timers

TPMx Status Control (TPMx_SC) register

Field	Bits	Description
CMOD	3–4	Clock Mode Selection **CMOD value** / **Selected clock** **00**: Timer stopped (No clock selected): In the mode, the TPM_CNT register receives no clock and it is stopped. **01**: Timer mode (clock selected at SIM_SOPT2): This mode can be used to generate delays, periodic interrupts, or PWM. **10**: Counter mode (clocked by LPTPM_EXTCLK pin): This mode is used to count an external event. **11**: Reserved

CMOD value	Selected clock
00	Timer stopped (No clock selected): In the mode, the TPM_CNT register receives no clock and it is stopped.
01	Timer mode (clock selected at SIM_SOPT2): This mode can be used to generate delays, periodic interrupts, or PWM.
10	Counter mode (clocked by LPTPM_EXTCLK pin): This mode is used to count an external event.
11	Reserved

Basic Programming Techniques - Timers

TPMx Status Control (TPMx_SC) register

Field	Bits	Description
CPWMS	5	Center-aligned PWM select (0: Up counter mode, 1: up-down counter mode). For generating delays use the Up counter mode.
TOIE	6	Time Overflow Interrupt Enable (0: Disabled, 1: Enabled). It is discussed in Chapter 6.
TOF	7	Timer Overflow Flag
DMA	8	DMA Enable (0: Disabled, 1: Enabled)

TOF flag bit

The TOF (Timer Overflow Flag) is bit D7 of the TPM_SC register. When the CNT register counts up and matches the value in TPMx_MOD, TOF is set to 1.

Making delays using the TPM timer

The steps to program the timer for TPMx_CNT are:

1) enable the clock to TPMx module in SIM_SCGC6,
2) select the clock source for timer counter in SIM_SOPT2,
3) disable timer while the configuration is being modified,
4) set the mode as up-counter timer mode with TPMx_SC register,
5) load TPMx_MOD register with proper value,
6) clear TOF flag,
7) enable timer,
8) wait for TOF flag to go HIGH.

Basic Programming Techniques - Timers

Making delays using the TPM timer

Example: Toggle blue LED (PTD1 pin) every 320 times TPM0_CNT matches the TPM0_MOD.

```
/* This program uses TPM0 to generate maximal delay to
toggle the blue LED.
MCGFLLCLK (41.94 MHz) is used as timer counter clock.
The Modulo register is set to 65,535. The timer counter
overflows at
41.94 MHz / 65,536 = 640 Hz
We put the time out delay in a for loop and repeat it
for 320 times before we
toggle the LED. This results in the LED flashing at half
second on and half
second off.
The blue LED is connected to PTD1. */
```

Making delays using the TPM timer

```
#include <MKL25Z4.H>

int main (void) {
int i;
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
PTD->PDDR |= 0x02; /* make PTD1 as output pin */

SIM->SCGC6 |= 0x01000000; /* enable clock to TPM0 */
SIM->SOPT2 |= 0x01000000; /* use MCGFLLCLK as CNT clock */
TPM0->SC = 0; /* disable timer while configuring */
TPM0->MOD = 0xFFFF; /* max modulo value */
TPM0->SC |= 0x80; /* clear TOF */
TPM0->SC |= 0x08; /* enable timer free-running mode */
```

Making delays using the TPM timer

```
while (1) {

for(i = 0; i < 320; i++) {

/* repeat timeout for 320 times */

while((TPM0->SC & 0x80) == 0) {}

/* wait until the TOF is set */

TPM0->SC |= 0x80; /* clear TOF */

PTD->PTOR = 0x02; /* toggle blue LED */
}
}
```

Making delays using the TPM timer

Exercise

(a) Show time delay calculation for the previous program
(b) Find the TPMx_MOD value to make a delay of 142 ms.

Making delays using the TPM timer

Solution:

a) 1 / 41.94MHz = 23.84ns since the FRDM board working clock is 41.94MHz. 23.84ns x 65,536 = 1.56 msec

The timer overflows every 1.56 msec. The delay contains 320 timer overflows in the for-loop: 1.56msec x 320 = 0.5 second

b) 1 / 41.94MHz = 23.84ns since the FRDM board working clock is 41.94MHz.

142 ms / 23.84ns = 5956. Thus TPMx_MOD = 5955

Prescaler options of timer

The clock source of the timer counter is selected in SIM_SOPT2 register.

The prescaler sits between the clock source and the timer counter. It can be configured to divide the clock source by a number before feeding it to the timer counter.

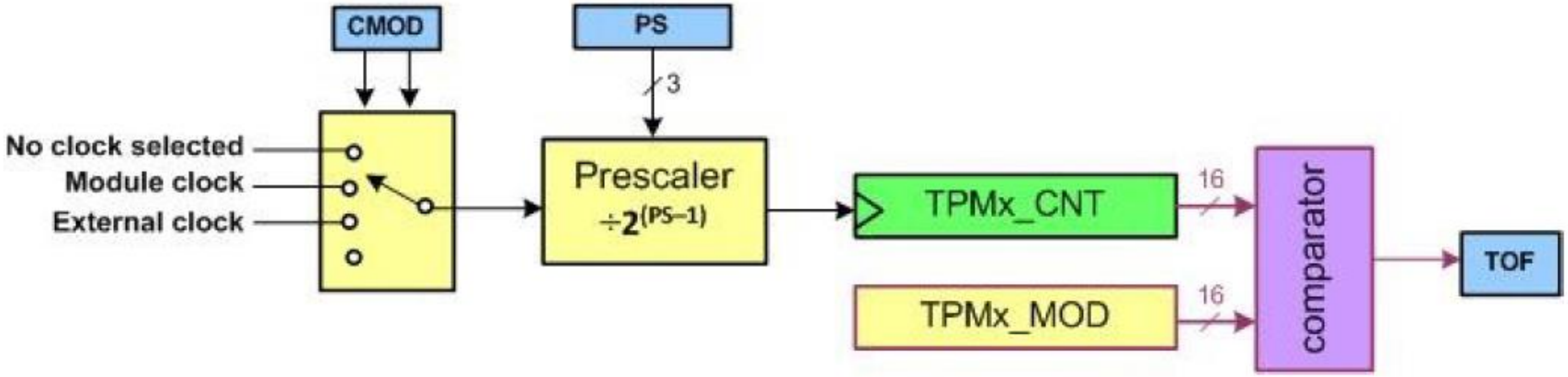
The lowest 3 bits of the TPMx_SC register give the options of the number we can divide by.

Next, we will examine how the prescaler options are programmed.

Prescaler register for TPMx

Because TPM Modulo register has only 16 bits, the time interval is limited to 1.56 ms with 41.94 MHz clock as seen before.

For longer delay, we will need to incorporate prescaler. The next program sets the prescalers to divide by 128 that will extend the period to 200 ms.



Prescaler register for TPMx

```
/* Toggling blue LED using TPM0 delay (prescaler)
* This program uses TPM0 to generate maximal delay
to
* toggle the blue LED.
* MCGFLLCLK (41.94 MHz) is used as timer counter
clock.
* Prescaler is set to divided by 128 and the
Modulo register
* is set to 65,535. The timer counter overflows at
* 41.94 MHz / 128 / 65,536 = 5.0 Hz
*
* The blue LED is connected to PTD1.
*/
```

Prescaler register for TPMx

```
#include <MKL25Z4.H>

int main (void) {

SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
PTD->PDDR |= 0x02; /* make PTD1 as output pin */
SIM->SCGC6 |= 0x01000000; /* enable clock to TPM0 */
SIM->SOPT2 |= 0x01000000; /* use MCGFLLCLK as timer counter
clock */
TPM0->SC = 0; /* disable timer while configuring */
TPM0->SC = 0x07; /* prescaler /128 */
TPM0->MOD = 0xFFFF; /* max modulo value */
TPM0->SC |= 0x80; /* clear TOF */
TPM0->SC |= 0x08; /* enable timer free-running mode */
```

Prescaler register for TPMx

```
while (1) {
while((TPM0->SC & 0x80) == 0) { }
/* wait until the TOF is set */
TPM0->SC |= 0x80; /* clear TOF */
PTD->PTOR = 0x02; /* toggle blue LED */
}
}
```

Prescaler register for TPMx

Exercise

(a) Show time delay calculation for the previous program
(b) calculate the largest delay size without prescaler
(c) Find the TPMx_MOD value to generate a delay of 0.1 second. Use the prescaler of 128.

Prescaler register for TPMx

Solution:

(a) 41.94 MHz / 128 = 327,656 Hz with prescaler of 128.
1 / 327,656 Hz = 3.05 μsec , 3.05 μsec x 65,535 = 200 ms

(b) 41.94 MHz / 1 = 41.94 MHz with no prescaler.
1 / 41.94 MHz = 23.84 ns.
The largest possible delay is TPMx_MOD=65,535=0xFFFF.
Now, 65,536 × 23.84 ns = 1,562,613 ns = 1.56 ms = 0.00156 sec.

(c) 41.94 MHz / 128 = 327,656 Hz with prescaler of 128.
1 / 327,656 Hz = 3.05 μsec
0.1 sec / 3.05 μsec = 32766. TPMx_MOD is 32,766 – 1 = 32,765.

TPMx Registers and their Addresses

Table below shows the addresses of major registers for TPM0, TPM1, and TPM2 modules.

Absolute address	Register Name
4003 8000	Status and Control (TPM0_SC)
4003 8004	Counter (TPM0_CNT)
4003 8008	Modulo (TPM0_MOD)
4003 800C	Channel 0 Status and Control (TPM0_C0SC)
4003 8010	Channel 0 Value (TPM0_C0V)

Basic Programming Techniques - Timers

TPMx Registers and their Addresses

4003 8014	Channel 1 Status and Control (TPM0_C1SC)
4003 8018	Channel 1 Value (TPM0_C1V)
4003 801C	Channel 2 Status and Control (TPM0_C2SC)
4003 8020	Channel 2 Value (TPM0_C2V)

4003 8050	Capture and Compare Status (TPM0_STATUS)
4003 9000	Status and Control (TPM1_SC)

Longer Time Interval

As shown in previous examples, with 41.94 MHz system clock the longest time interval we could get was 200 ms.

To achieve a longer time interval, we may repeat the short time interval multiple times

An alternative is to drive the timer with a slower clock. The benefit of slow clock is that the circuit consumes much less power when it is switching fewer times.

This is important if it is used in mobile device when battery charge is precious.

Longer Time Interval

The Freescale ARM KL25Z has an internal reference clock at 32.768 kHz that may be used as the clock source for the timers.

Recall the timer clock source selection is made in SIM->SOPT2 register.

The next example uses the 32.768 kHz to generate 5 second timeout interval.

The longest timeout interval from the timers with the 32.768 kHz clock is 32.768 kHz / 128 / 65536 = 0.0039 Hz or 256 second.

Longer Time Interval

```
/* Toggling blue LED using TPM0 delay
* This program uses TPM0 to generate long delay to
* toggle the blue LED.

* MCGIRCLK (32.768 kHz) is used as timer counter clock.

* Prescaler is set to divided by 4 and the Modulo
register
* is set to 40,959. The timer counter overflows at
* 32,768 Hz / 40,960 / 4 = 0.2 Hz
*
* The blue LED is connected to PTD1.
*/
```

Longer Time Interval

```
#include <MKL25Z4.H>

int main (void) {

SIM->SCGC5 |= 0x1000; /* enable clock to Port D */

PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */

PTD->PDDR |= 0x02; /* make PTD1 as output pin */

SIM->SCGC6 |= 0x01000000; /* enable clock to TPM0 */

SIM->SOPT2 |= 0x03000000;

/* use MCGIRCLK as timer counter clock */
```

Longer Time Interval

```
TPM0->SC = 0; /* disable timer while configuring */
TPM0->SC = 0x02; /* prescaler /4 */
TPM0->MOD = 40960 - 1; /* modulo value */
TPM0->SC |= 0x80; /* clear TOF */
TPM0->SC |= 0x08; /* enable timer free-running mode */

while (1) {

while((TPM0->SC & 0x80) == 0) { }
/* wait until the TOF is set */

TPM0->SC |= 0x80; /* clear TOF */
PTD->PTOR = 0x02; /* toggle blue LED */
}
}
```

Counters and Timers in microcontrollers

For even longer timeout interval, the Freescale ARM KL25Z has a low power timer (LPTMR) that may use the 32.768 kHz clock or a 1 kHz clock.

The LPTMR
has a prescaler that will divide up to 65,536 and a 16-bit counter that will count up to 65,536.

The longest timeout interval for LPTMR is 4,294,967 seconds or about 50 days.

We will leave the programing of the LPTMR to you ☺

Output Compare and TPM Channels

In the last section, we showed how to use timers to generate time delay.

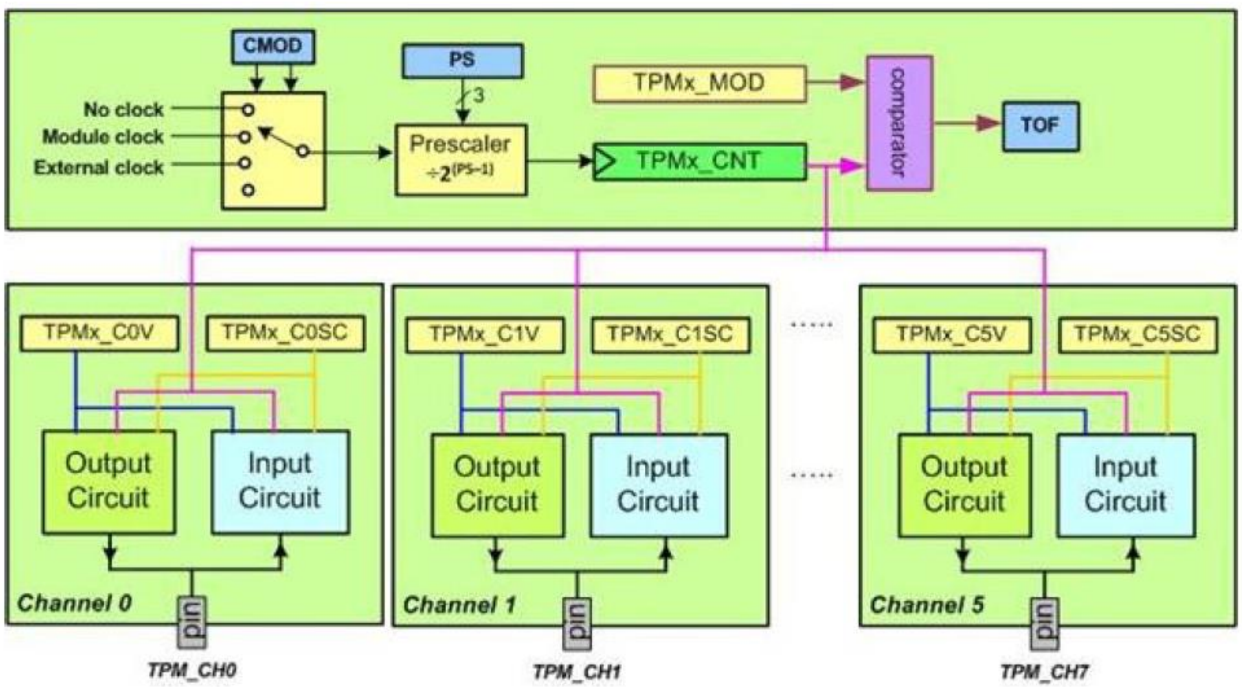
In this and following sections, we will examine the use of timers with the I/O pins.

In this section, we will study the Output Compare feature of the KL25Z Timers.

We examine the channels of TPMs, as well.

Programming the Output Compare Option

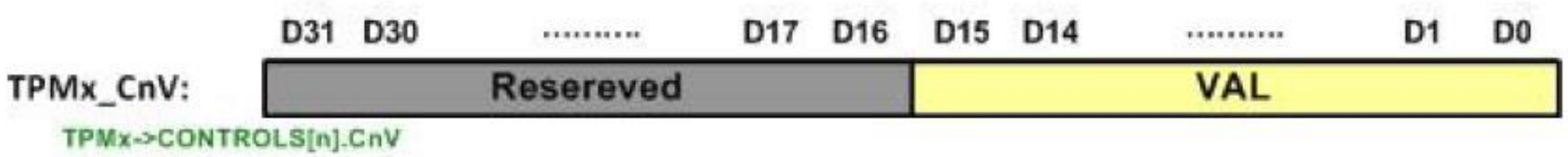
In some applications we need to control the digital pin output transition with precision timing. To do that, we use the Output Compare function of the timer. In the MCU, we have 6 channels



Programming the Output Compare Option

Each channel has its own 16-bit register for the compare purpose.

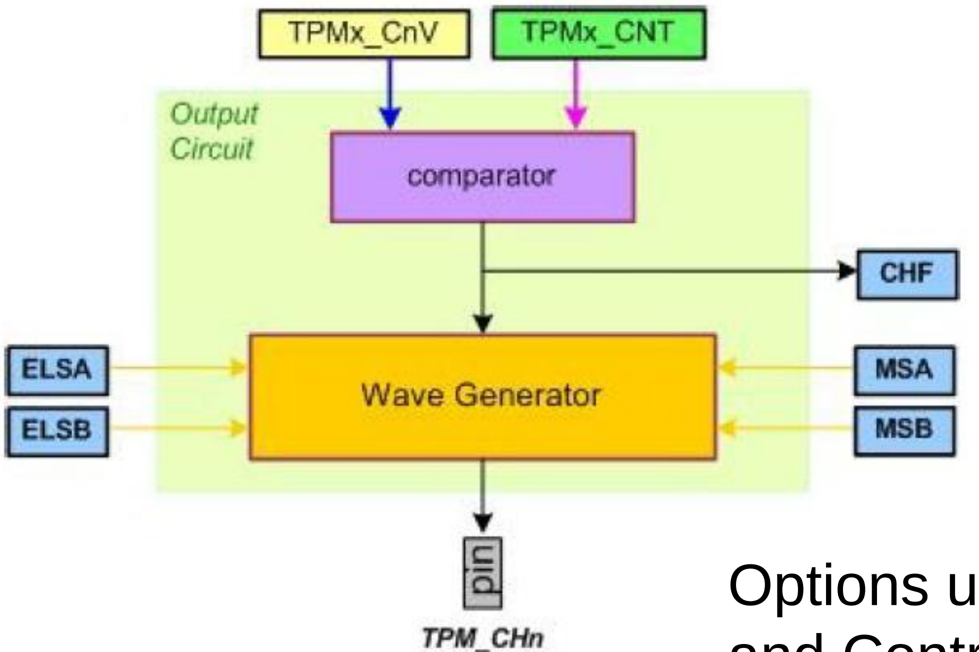
The registers are called TPMx Channel Value (TPMx_CnV) and are designated as TPMxC0V to TPMxC5V.



The 16-bit registers of TPMx_CnV are readable and writable, which means we can initialize them to a desired value.

Programming the Output Compare Option

After the initialization, the TPMx_CnV content is compared with the value in TPMx_CNT after each clock cycle as TPM_CNT is counting up.



When the value of the CNT register and CnV register match

CHF flag is set high

Options using the Channel Status and Control Register(TPMx_CnSC)

Channel Status and Control Selection (TPMx_CnSC) register

As we just stated, each TPM module has six channels.

There are two registers associated with each channel.

They are the Channel Value register (TPMxCnV) and Channel Status and Control register (TPMxCnSC).

Notice the x can be 0, 1, or 2 for Timer modules of 0, 1, and 2.

The n can be 0 to 5 for one of the six channels inside each Timer module.

Channel Status and Control Selection (TPMx_CnSC) register

Now, the mode and edge selections for Output Compare of a given channel are done with TPMx Channel Status and Control (TPMxCnSC).

Bits D5:D4 is used to choose the Output Compare option of the timer.

	D31 ..... D8	D7	D6	D5	D4	D3	D2	D1	D0
TPMx_CnSC:	Reserved	CHF	CHIE	MSB	MSA	ELSB	ELSA	0	DMA

TPMx->CONTROLS[n].CnSC

Channel Status and Control Selection (TPMx_CnSC) register

Field	Bit	Description
CHF	7	Channel Flag
CHIE	6	Channel interrupt enable
MSB and MSA	5-4	Channel mode select
ELSB and ELSA	3-2	Edge or Level Select
DMA	0	DMA enable (0: Disabled, 1: Enabled)

Channel mode select (MSB and MSA, bits 5-4):

D5:D4 (MSB:MSA)	Output mode
00	Channel disabled
01	Output compare
10	PWM
11	Output compare

Channel Status and Control Selection (TPMx_CnSC) register

After selecting the Output Compare with D5:D4=01, we use the D3:D2 bits to choose the following action for a given channel

D5:D4 (MSB:MSA)	D3 (ELSB)	D2 (ELSA)	Output Action
01	0	1	Toggle Output on Match
01	1	0	Clear Output on Match (make it Low)
01	1	1	Set Output on Match (make output High)
11	1	0	Pulse Output Low on Match
11	X	1	Pulse Output High on Match

Output Compare Mode

If a timer channel is in the output compare mode →

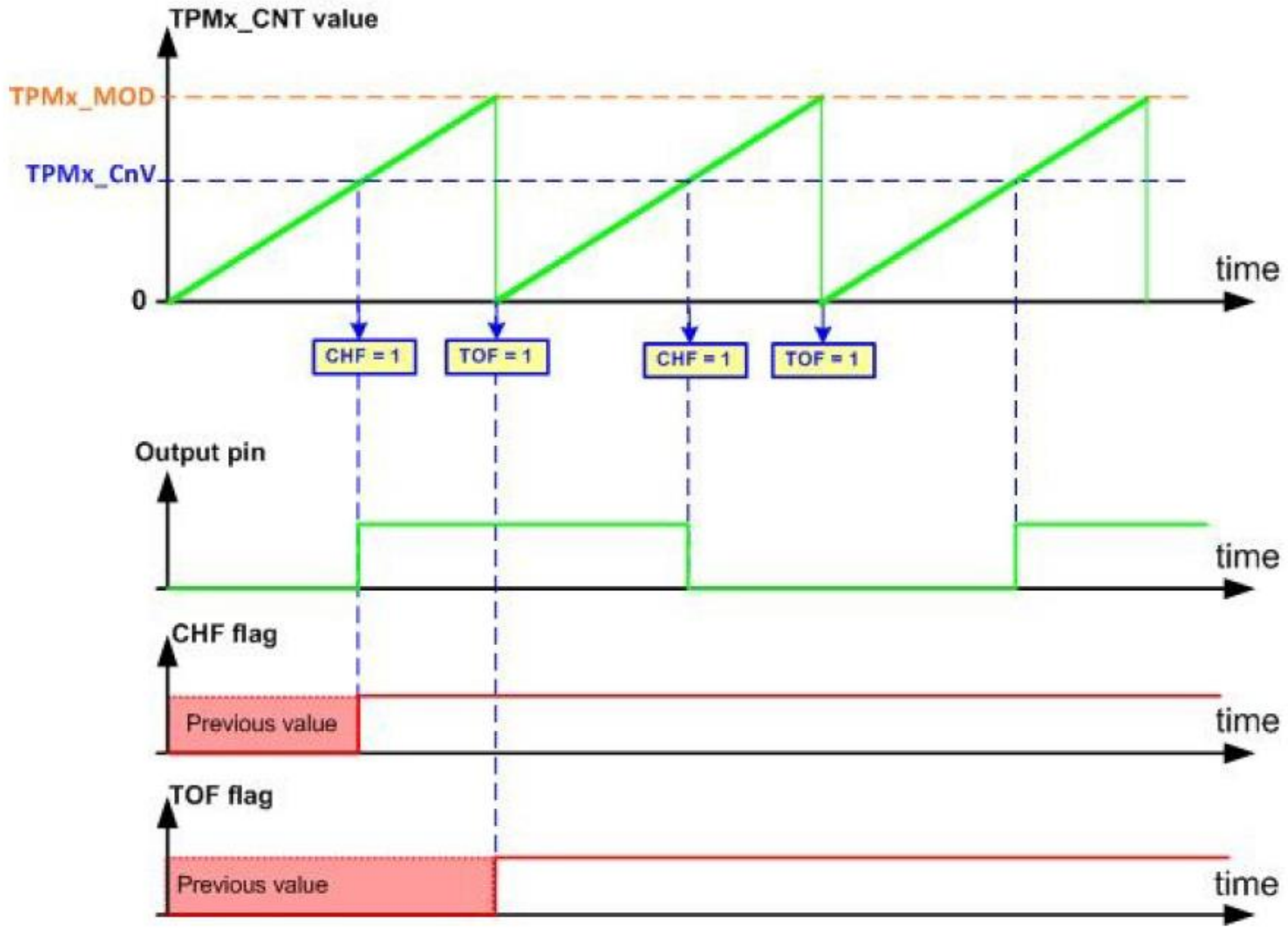
When the timer is counting up, the TPMx_CNT counter begins counting from 0 and goes up until it reaches the TPMx_CnV value.

Then, the Channel Flag (CHF) for that channel is set and the channel output is changed.

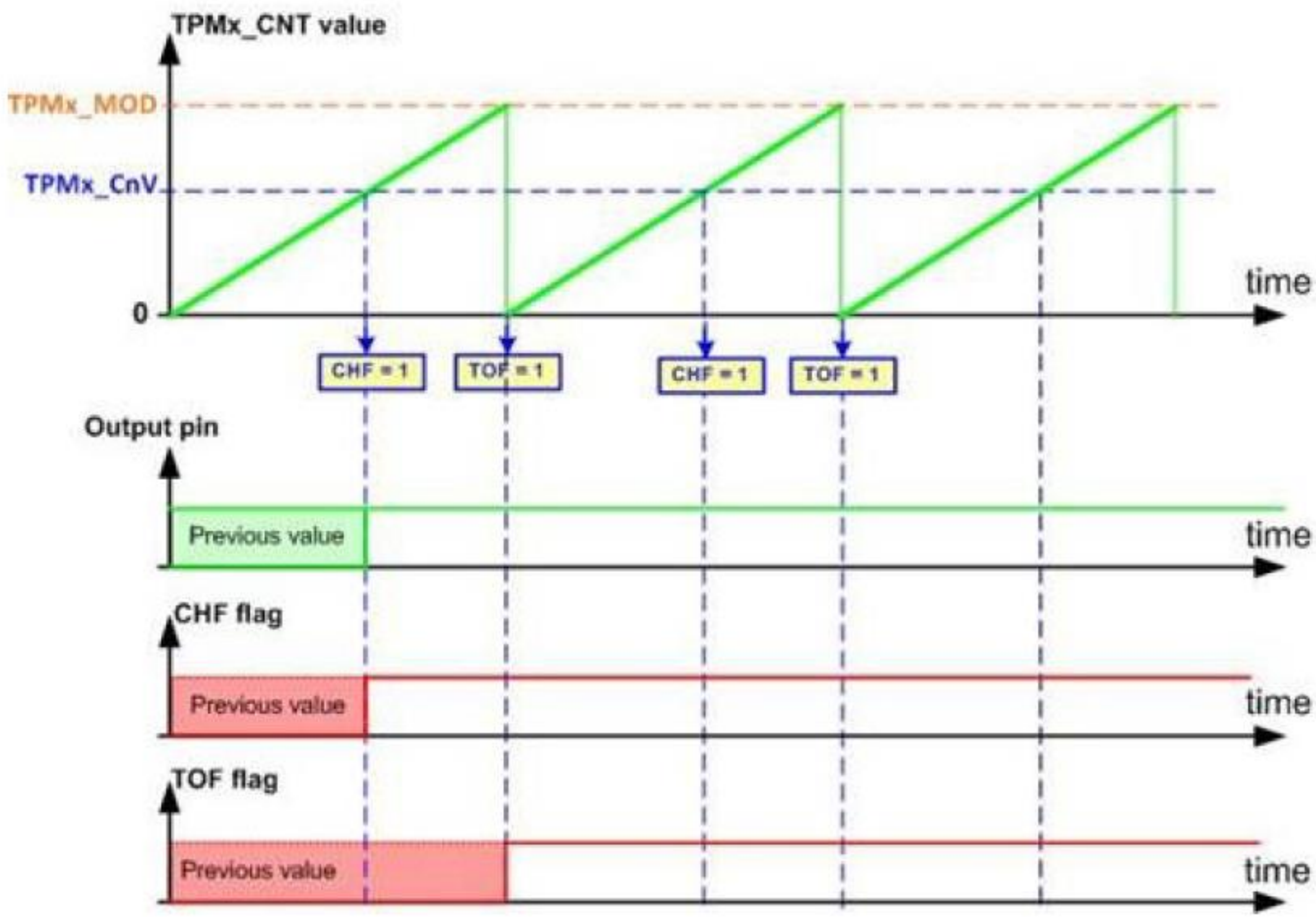
The timer continues counting until it reaches to the TPMx_MOD value and rolls over.

See some examples

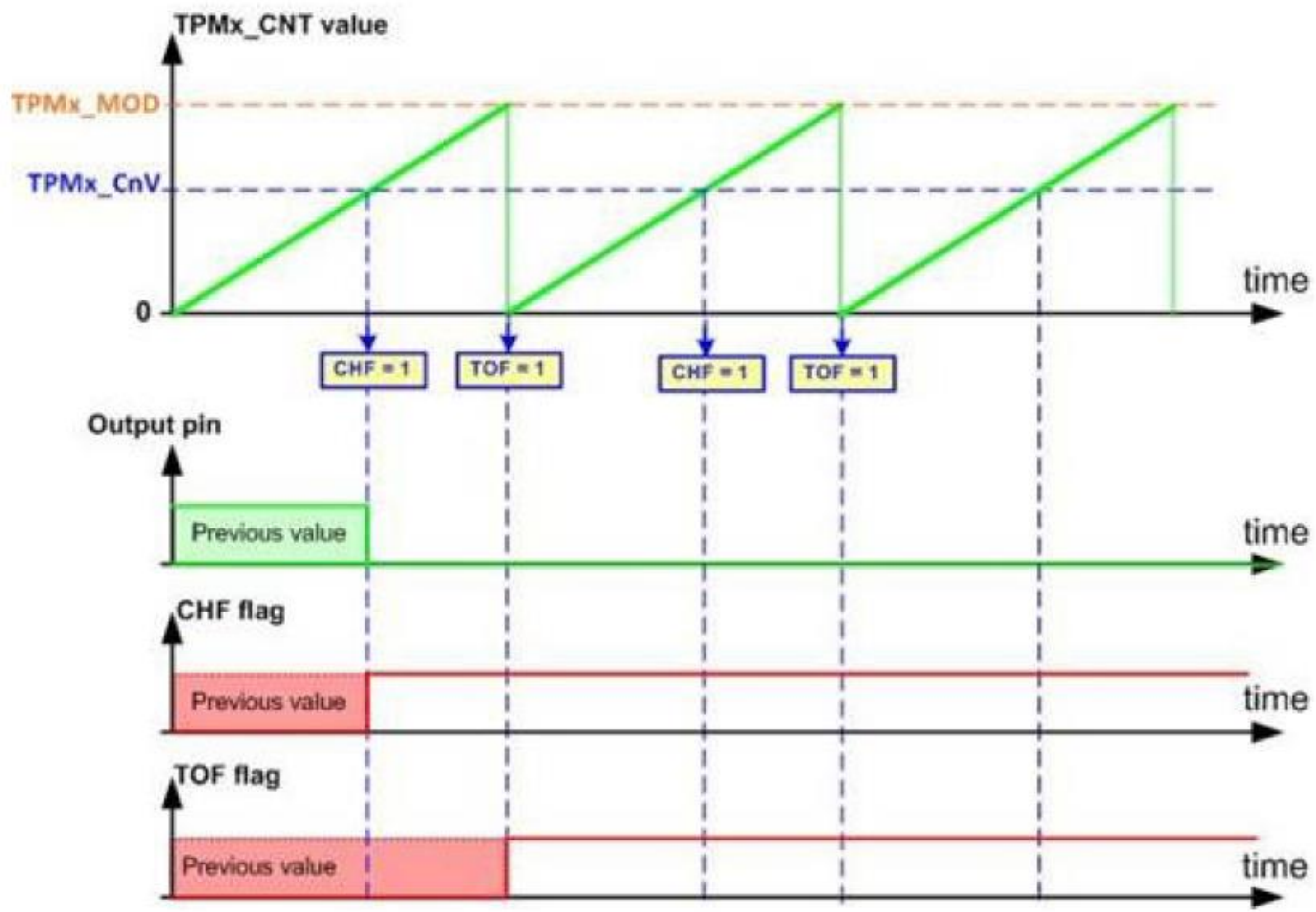
Output Compare Module → Toggle Mode



Output Compare Module → Set Mode



Output Compare Module → Clear Mode



Channel Pins

There are CH0 to CH5 for each TPMx. Each channel has its own designated output pins.

For example, Channel 0 of TPM0 may be connected to PTA3, PTC1, PTD0, or PTE24 for output and Channel 1 of TPM0 may use PTA4, PTC2, PTD1 or PTE25 as output pin.

The choices of the output pin designations are made in the alternate function of the pin control register (PORTx_PCR) of each port.

It is possible to have multiple output pins connected to a single channel at the same time.

Selecting alternate function for Timers pin

Upon Reset, the PORTx_PCRn register has all 0s meaning the I/O pins are not defined yet.

To use an alternate function, we first must configure the bits in the PORTx_PCRn register for that pin.

As we mentioned in previous classes, each pin has its own PORTx_PCRn register. For example, for the PTB18 to be used by TPM2_CH1, we need to write 0x0300 (0000 0011 0000 0000 in binary) to PORTB_PCR18 register

To do that, we need to use the information in Section 10.3.1 of Freescale KL25Z Ref. Manual

Selecting alternate function for Timers pin

Using ALT3 pin options for TPM0 Channel Output		
TPM0 Channels	Pins	Pin Control Register
TPM0 CH0 Output Pins	PTA3	PORTA_PCR3=0x0300
	PTE24	PORTE_PCR24=0x0300
TPM0 CH1 Output Pins	PTA4	PORTA_PCR4=0x0300
	PTE25	PORTE_PCR25=0x0300
TPM0 CH2 Output Pins	PTA5	PORTA_PCR5=0x0300
	PTE29	PORTE_PCR29=0x0300
TPM0 CH3 Output Pins	PTE30	PORTE_PCR30=0x0300

Selecting alternate function for Timers pin

Using ALT4 pin options for TPM0 Channel Output		
TPM0 CH0 Output Pins	PTC1	PORTC_PCR1=0x0400
	PTD0	PORTD_PCR0=0x0400
TPM0 CH1 Output Pins	PTC2	PORTC_PCR2=0x0400
	PTD1	PORTD_PCR1=0x0400
TPM0 CH2 Output Pins	PTC3	PORTC_PCR3=0x0400
	PTD2	PORTD_PCR2=0x0400
TPM0 CH3 Output Pins	PTC4	PORTC_PCR4=0x0400
	PTD3	PORTD_PCR3=0x0400

Selecting alternate function for Timers pin

Using ALT3 pin options for TPM1 Channel Output		
TPM1 Channels	Pins	Pin Control Register
TPM1 CH0 Output Pins	PTA12	PORTA_PCR12=0x0300
	PTB0	PORTB_PCR0=0x0300
	PTE20	PORTE_PCR20=0x0300
TPM1 CH1 Output Pins	PTA13	PORTA_PCR13=0x0300
	PTB1	PORTB_PCR1=0x0300
	PTE21	PORTE_PCR21=0x0300

Selecting alternate function for Timers pin

Using ALT3 pin options for TPM2 Channel Output		
TPM2 Channels	Pins	Pin Control Register
TPM2 CH0 Output Pins	PTA1	PORTA_PCR1=0x0300
	PTB18	PORTB_PCR18=0x0300
	PTE22	PORTE_PCR22=0x0300
TPM2 CH1 Output Pins	PTA2	PORTA_PCR2=0x0300
	PTB3	PORTB_PCR3=0x0300
	PTB19	PORTB_PCR19=0x0300
	PTE23	PORTE_PCR23=0x0300

Basic Programming Techniques - Timers

Toggling a pin using output compare

The steps to program the timer for Output Compare are:

Enable the clock to the output pin GPIO port

Select the alternate function for the output pin

Enable the clock to TPMx module

Select the clock source for timer counter

Disable timer while the configuration is being modified

Toggling a pin using output compare

Select prescaler value with TPMx_SC register

Set modulo value in TPMx_MOD register,set the CnSC register to toggle mode

(MSA = 1, MSB = 0, ELSA = 1, ELSB = 0),

Clear CHFn (channel n flag) flag

set TPMx_CnV register based on its current value with the interval count added

Enable timer

Basic Programming Techniques - Timers

Toggling a pin using output compare

The next program uses output compare mode to toggle thePTD1 pin (blue led). Every time there is match between TPM0_CNT and TPM0_C1V registers, the output is toggled.

The program reads the TPM0_C1V value and adds 32766 to it that scheduled the next match to be 32,766 clock cycles later.

The timer counter clock is running at 41.94 MHz and the prescaler to 128 so the timer counter is counting at 41.94 MHz / 128 = 367,656 Hz and the period is 3.05 μs.

To schedule next output compare match for 32,766 clock cycles results in 3.05 μs × 32,766 = 0.1 sec.

Toggling a pin using output compare

```
#include <MKL25Z4.H>

int main (void) {

SIM->SCGC5 |= 0x1000; /* enable clock to Port D */

PORTD->PCR[1] = 0x400; /* set PTD1 pin for TPM0CH1 */

SIM->SCGC6 |= 0x01000000; /* enable clock to TPM0 */

SIM->SOPT2 |= 0x01000000;

/* use MCGFLLCLK as timer counter clock */
```

Basic Programming Techniques - Timers

Toggling a pin using output compare

```
TPM0->SC = 0; /* disable timer while configuring */
TPM0->SC = 0x07; /* prescaler /128 */
TPM0->MOD = 0xFFFF; /* max modulo value */
TPM0->CONTROLS[1].CnSC = 0x14; /* OC toggle mode */
TPM0->CONTROLS[1].CnSC |= 0x80; /* clear CHF */
TPM0->CONTROLS[1].CnV = TPM0->CNT + 32766; /* schedule
next transition */
TPM0->SC |= 0x08; /* enable timer */

while (1) {
while(!(TPM0->CONTROLS[1].CnSC & 0x80)) { } /* wait until
the CHF is set */
TPM0->CONTROLS[1].CnSC |= 0x80; /* clear CHF */
TPM0->CONTROLS[1].CnV = TPM0->CNT + 32766; /* schedule
next transition */
}
```

Channel Status for each timer module (TPMx_STATUS) register

As we just stated that, each TPM module has six channels.

There are two registers associated with each channel.

They are the Channel Value register (TPMxCnV) and Channel Status and Control register (TPMxCnSC).

However, we also have a single status register for all the channels.

This is called TPMx_STATUS.

Channel Status for each timer module (TPMx_STATUS) register

This allows us to monitor the status of all the 6 channels with a single read of the register to see whether any given CHF flag has been raised.

Notice that we have D5 bit for Channel 5 flag and D0 bit for Channel 0 flag. Also notice that, in addition to the CHF (Channel flag) for each channel, we also have the TOF belonging to the TPMx_CNT and TPMx_MOD all of them in one register.

	D31	......	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
TPMx_STATUS:	Reserved			TOF	0		CH5F	CH4F	CH3F	CH2F	CH1F	CH0F
TPMx->STATUS												

Using Timer for Input Edge-time Capturing

Input edge-time mode

In input edge-time mode, an I/O pin is used to capture the signal transition events.

When an event occurs, the content of the TPMx_CNT timer counter is captured and saved in a register while the counter keeps counting.

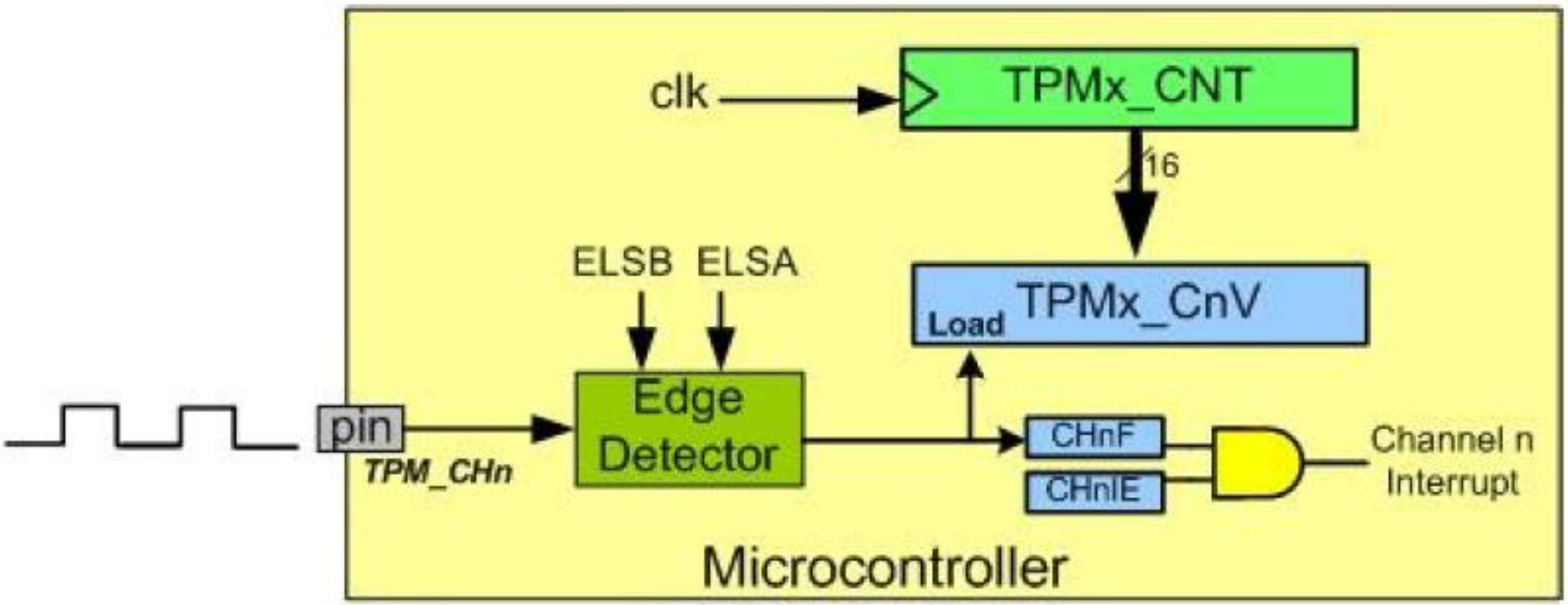
To configure TPM as Input Capture mode, bits MSnB:MSnA of the TPMx_CnSC should be 00 (binary). We use ELSnB:ELSnA bits to choose the rising or falling edge.

	D31 ..... D8	D7	D6	D5	D4	D3	D2	D1	D0
TPMx_CnSC:	Reserved	CHF	CHIE	MSB	MSA	ELSB	ELSA	0	DMA

TPMx->CONTROLS[n].CnSC

Using Timer for Input Edge-time Capturing

In this mode, the counter value is stored in the Channel register (TPMx_CnV) whenever the input pin is triggered by an external event (falling or rising edge-triggered) fed to the TPM_CHn pin.



Contact Bounce and Debounce

Notice that the Channel can be configured to capture on the falling edge, rising edge, or both. To determine the type of edge that is captured, the ELSn:BLELSnA bits of the TPMx_CnSC register should be initialized

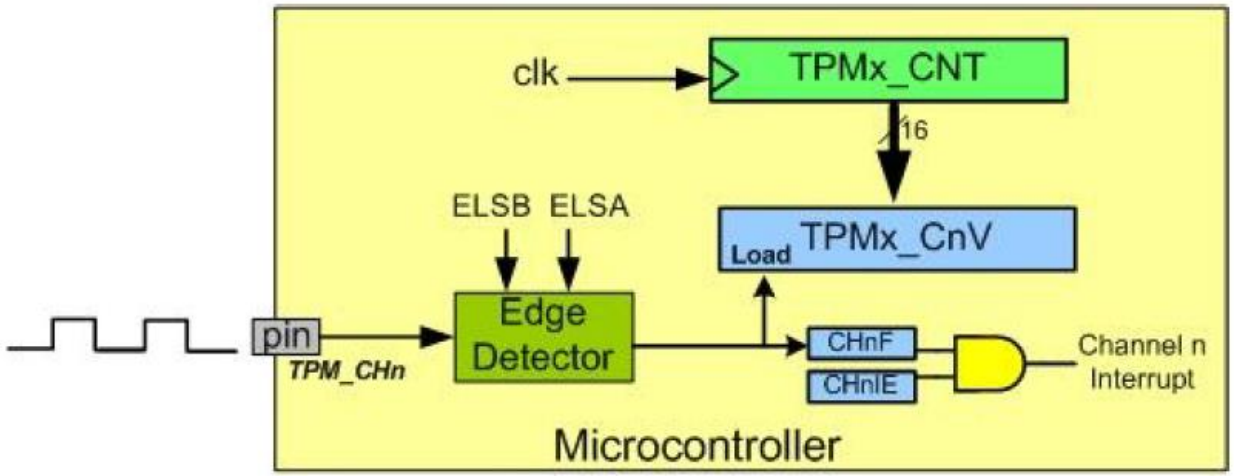
ELSB	ELSA	Capture mode
0	0	Channel disabled
0	1	Capture on rising edge
1	0	Capture on falling edge
1	1	Capture on both edges

Also notice that capturing has no effect on counting and the timer counter continues counting when the capture event takes place

Pin Selection for Input Capture

To measure the edge time we must feed the pulse into the TPM_CHn pin.

The input capture timer channel-pin designation is identical to the output compare timer channel-pin designation in tables in slides 84-87. So we will not repeat it here.



Input edge-time mode usages

The input edge time capturing can be used for many applications; e.g. recording the arrival time of an event, measuring the frequency and pulse width of a signal.

Steps to program the Input Capture function

Perform the following steps to measure the period of a periodic waveform based on the edge arrival time of the Input Capture function.

1) Enable the clock to the input pin GPIO port,

2) select the alt. function for the input pin at the PORTX_PCR register,

3) enable the clock to TPMx module,

4) select the clock source for timer counter,

5) disable timer while the configuration is being modified,

6) select prescaler value with TPMx_SC register,

Steps to program the Input Capture function

7) set modulo value in TPMx_MOD register,

8) set the CnSC register to capture rising edge,

9) enable timer,

10) wait until the CHF bit is set in CnSC register,

11) read the current counter value captured,

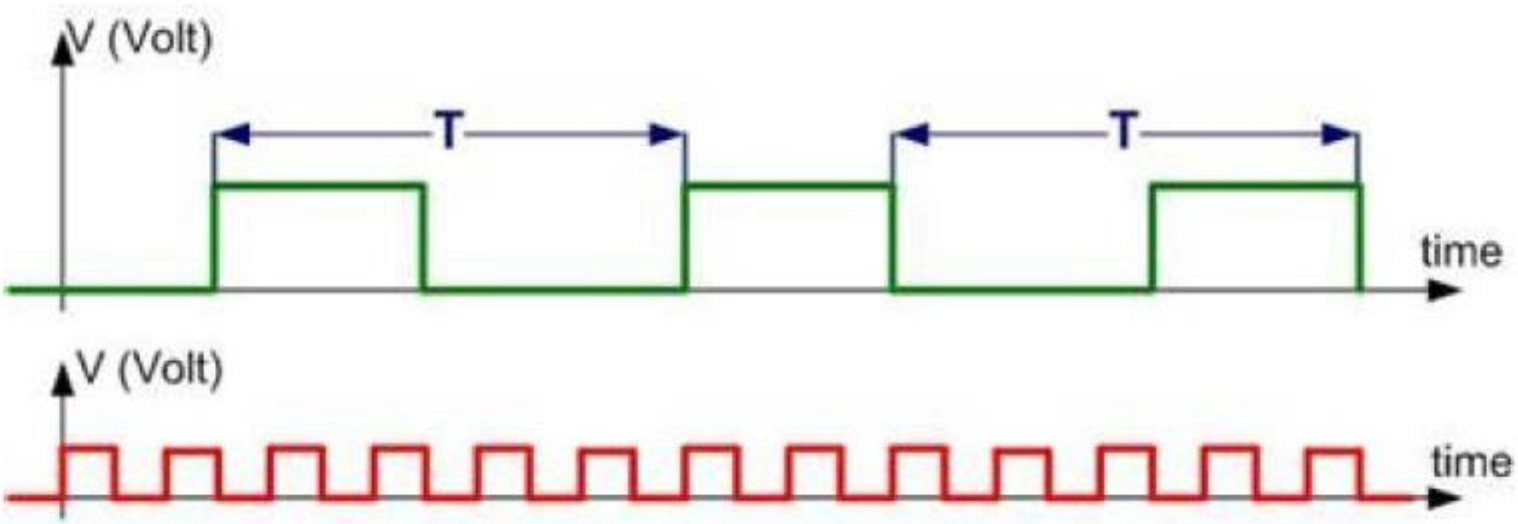
12) calculate the current counter value difference from the last value,

13) save the current value for next calculation,

14) repeat from step 10.

Input edge-time mode usages

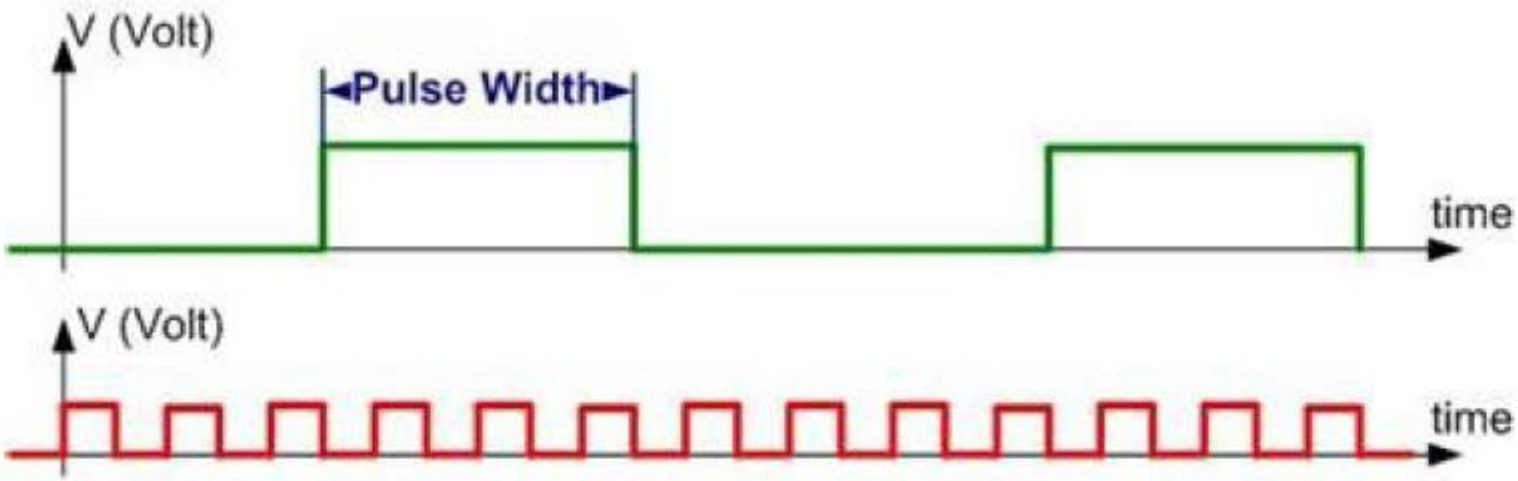
As shown in the figure to measure the period of a signal we must measure the time between two falling edges or two rising edges.



(a) Measuring Period in Terms of the Number of Clocks Counted by the Timer

Input edge-time mode usages

The next program measures the period of the square wave.



(b) Measuring Pulse Width in Terms of the Number of Clocks Counted by the Timer

Input edge-time mode usages

```
/* Using TPM2 Channel 0 to measure input period.
* This program uses TPM2 CH1 Input Edge-time Capture to
measure
* the period of a periodic waveform.
* MCGFLLCLK (41.94 MHz) is used as timer counter clock.
* Prescaler is set to divided by 128. So the timer
counter is
* counting at 41.94 MHz / 128 = 327,656 Hz.
* Timer 2 Channel 0 is configured as Input Edge-time
Capture mode.
* and the input is using PTA1.
```

Input edge-time mode usages

```
* When a rising edge occurs at PTA1, the timer counter
value is
* copied to TPM2_C0V and the CHF is set.
* The program waits for CHF flag to set then calculates
the
* difference of the current value to the previous
recorded value.
* Bit 11-9 are used to control the tri-color LEDs.
* The LED should change color when the input frequency is
changing
* below 642 Hz. Above 642 Hz, the number of clock cycles
between
* rising edges is too small to reach bit 9.
*/
```

Input edge-time mode usages

```
#include <MKL25Z4.H>
int main (void) {
unsigned short then = 0;
unsigned short now = 0;
unsigned short diff;
/* Initialize GPIO pins for tri-color LEDs */
SIM->SCGC5 |= 0x400; /* enable clock to Port B */
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTB->PCR[18] = 0x100; /* make PTB18 pin as GPIO */
PTB->PDDR |= 0x40000; /* make PTB18 as output pin */
PORTB->PCR[19] = 0x100; /* make PTB19 pin as GPIO */
PTB->PDDR |= 0x80000; /* make PTB19 as output pin */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
PTD->PDDR |= 0x02; /* make PTD1 as output pin */
/* end GPIO pin initialization for LEDs */
```

Input edge-time mode usages

```
/* Start of Timer code */

SIM->SCGC5 |= 0x0200; /* enable clock to Port A */
PORTA->PCR[1] = 0x0300; /* set PTA1 pin for TPM2CH0 */
SIM->SCGC6 |= 0x04000000; /* enable clock to TPM2 */
SIM->SOPT2 |= 0x01000000; /* use MCGFLLCLK as timer
counter clock */
TPM2->SC = 0; /* disable timer while configuring */
TPM2->SC = 0x07; /* prescaler /128 */
TPM2->MOD = 0xFFFF; /* max modulo value */
TPM2->CONTROLS[0].CnSC = 0x04; /* IC rising edge */
TPM2->SC |= 0x08; /* enable timer */
```

Input edge-time mode usages

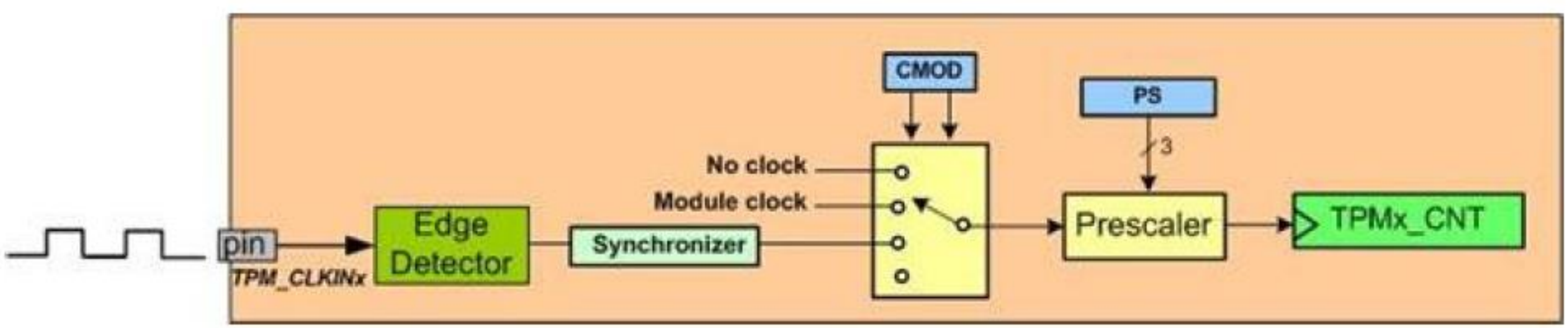
```
while (1) {
while(!(TPM2->CONTROLS[0].CnSC & 0x80)) { }

/* wait until the CHF is set */

TPM2->CONTROLS[0].CnSC |= 0x80; /* clear CHF */
now = TPM2->CONTROLS[0].CnV;

diff = now - then;

then = now;

/* save the current counter value for next calculation */
```

Input edge-time mode usages

```
/* change LEDs according to bit 11-9 of the value of diff */
diff = diff >> 9;
if (diff & 1) /* use bit 0 of diff to control red LED */
PTB->PCOR = 0x40000; /* turn on red LED */
else
PTB->PSOR = 0x40000; /* turn off red LED */
if (diff & 2) /* use bit 1 of diff to control green LED */
PTB->PCOR = 0x80000; /* turn on green LED */
else
PTB->PSOR = 0x80000; /* turn off green LED */
if (diff & 4) /* use bit 2 of diff to control blue LED */
PTD->PCOR = 0x02; /* turn on blue LED */
else
PTD->PSOR = 0x02; /* turn off blue LED */
/* end of LED code */
}
}
```

Using Timer as an Event Counter

TPMx works as a counter when the CMOD field of TPMx_SC is set to 10 (binary). In this mode, the timer counts the rising edges at the input pin synchronized to the timer counter clock.



For the timer to count the external edges the timer counter clock must be present and the external signal at the input pin should have the frequency half of the timer counter clock or lower

Using Timer as an Event Counter

The timer counter in event counter mode operates the same as the counter described before .

When the counter value reaches the Modulo register value, the TOF bit is set in TPMx_SC register and the TPMx_CNT value restarts from zero again.

As shown in the previous block diagram, the external clock signal also passes through the prescaler.

If the prescaler is set to divide by a number greater than 1, the external pulses are divided by the prescaler before incrementing the counter.

Basic Programming Techniques - Timers

Pin Selection for Event Counter

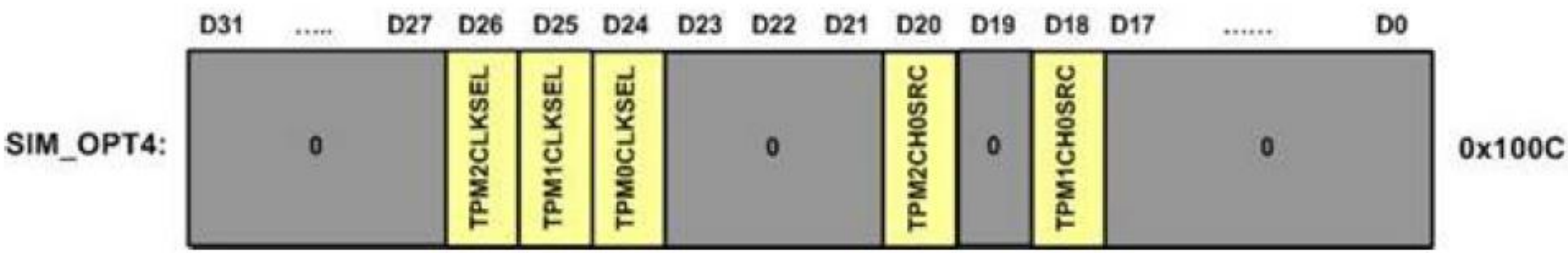
There are eight pins available to be used for external event counter. These pins are grouped as TPM_CLKIN0 and TPM_CLKIN1. The available pins for each group are listed in the following table

TPM_CLKIN0 Pins	PTA18	PORTA_PCR18=0x0400
	PTB16	PORTB_PCR16=0x0400
	PTC12	PORTC_PCR12=0x0400
	PTE29	PORTE_PCR29=0x0400
TPM_CLKIN1 Pins	PTA19	PORTA_PCR19=0x0400
	PTB17	PORTB_PCR17=0x0400
	PTC13	PORTC_PCR13=0x0400
	PTE30	PORTE_PCR30=0x0400

Pin Selection for Event Counter

The selected pin should have the clock enabled and the alternate port pin function set to 4 in PORTx_PCR register. Each timer module in event counter mode may select one input pin from either TPM_CLKIN0 or TPM_CLKIN1.

The selection is made in SIM_SOPT4 register. Bit 26 of SIM_SOPT4 is used for TPM2, bit 25 is used for TPM1, and bit 24 is used for TPM0. When the bit is 0, TPM_CLKIN0 is used. When the bit is 1, TPM_CLKIN1 is used.



Pin Selection for Event Counter

```
/¨* Counting pulses from PTC12.
* This is used as the base for P5_10.
* This program uses TPM0 to count the number of
pulses
* from PTC12.
* The tri-color LEDs are used to display bit2-0 of
* the counter. At low frequency input, the change of
* LED color should be visible.
* Although the counter is counting pulses from PTC12,
* timer counter clock must be present.
*/
```

Pin Selection for Event Counter

```
#include <MKL25Z4.H>
#include <stdio.h>
int main (void) {
unsigned short count;
/* Initialize GPIO pins for tri-color LEDs */
SIM->SCGC5 |= 0x400; /* enable clock to Port B */
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTB->PCR[18] = 0x100; /* make PTB18 pin as GPIO */
PTB->PDDR |= 0x40000; /* make PTB18 as output pin */
PORTB->PCR[19] = 0x100; /* make PTB19 pin as GPIO */
PTB->PDDR |= 0x80000; /* make PTB19 as output pin */
PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
PTD->PDDR |= 0x02; /* make PTD1 as output pin */
/* end GPIO pin initialization for LEDs */
```

Pin Selection for Event Counter

```
/* Start of Timer code */
SIM->SCGC5 |= 0x0800; /* enable clock to Port C */
PORTC->PCR[12] = 0x400; /* set PTC12 pin for TPM0 */
SIM->SOPT4 &= ~0x01000000; /* use TPM_CLKIN0 as timer counter
clock */
SIM->SCGC6 |= 0x01000000; /* enable clock to TPM0 */
SIM->SOPT2 |= 0x01000000; /* counter clock must be present */
TPM0->SC = 0; /* disable timer while configuring */
TPM0->SC = 0x80; /* prescaler /1 and clear TOF */
TPM0->MOD = 0xFFFF; /* max modulo value */
TPM0->CNT = 0; /* clear counter */
TPM0->SC |= 0x10; /* enable timer and use LPTPM_EXTCLK */
```

Pin Selection for Event Counter

```
while (1) {
count = TPM0->CNT;
/* change LEDs according to bit 2-0 of the value of count */
if (count & 1) /* use bit 0 of count to control red LED */
PTB->PCOR = 0x40000; /* turn on red LED */
else
PTB->PSOR = 0x40000; /* turn off red LED */
if (count & 2) /* use bit 1 of count to control green LED */
PTB->PCOR = 0x80000; /* turn on green LED */
else
PTB->PSOR = 0x80000; /* turn off green LED */
if (count & 4) /* use bit 2 of count to control blue LED */
PTD->PCOR = 0x02; /* turn on blue LED */
else
PTD->PSOR = 0x02; /* turn off blue LED */
/* end of LED code */
}
}
```

Contact Bounce and Debounce