Module 4 - Timers

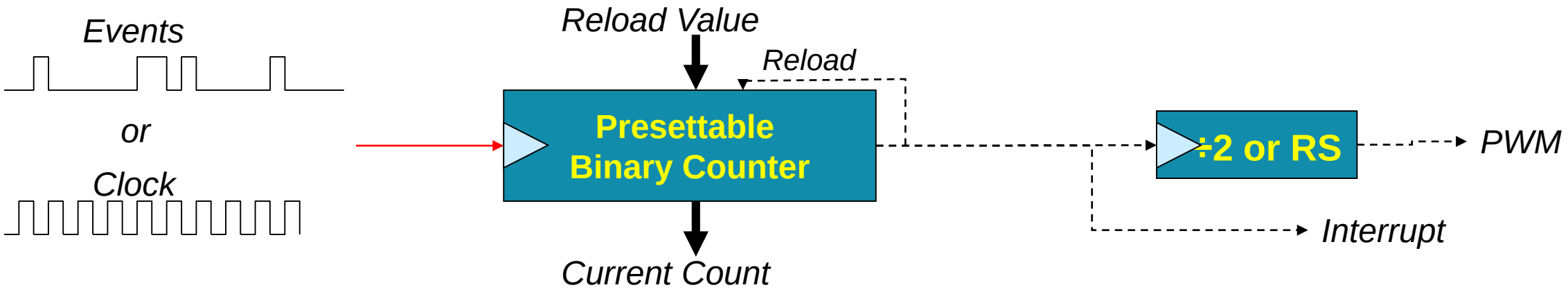
The Architecture for the Digital World®
ARM

KL25 Timer Peripherals

- PIT - Periodic Interrupt Timer
  - Can generate periodically generate interrupts or trigger DMA (direct memory access) transfers
- TPM - Timer/PWM Module
  - Connected to I/O pins, has input capture and output compare support
  - Can generate PWM signals
  - Can generate interrupts and DMA requests
- LPTMR - Low-Power Timer
  - Can operate as timer or counter in all power modes (including low-leakage modes)
  - Can wake up system with interrupt
  - Can trigger hardware
- Real-Time Clock
  - Powered by external 32.768 kHz crystal
  - Tracks elapsed time (seconds) in 32-bit register
  - Can set alarm
  - Can generate 1Hz output signal and/or interrupt
  - Can wake up system with interrupt
- SYSTICK
  - Part of CPU core's peripherals
  - Can generate periodic interrupt

ARM

Timer/Counter Peripheral Introduction

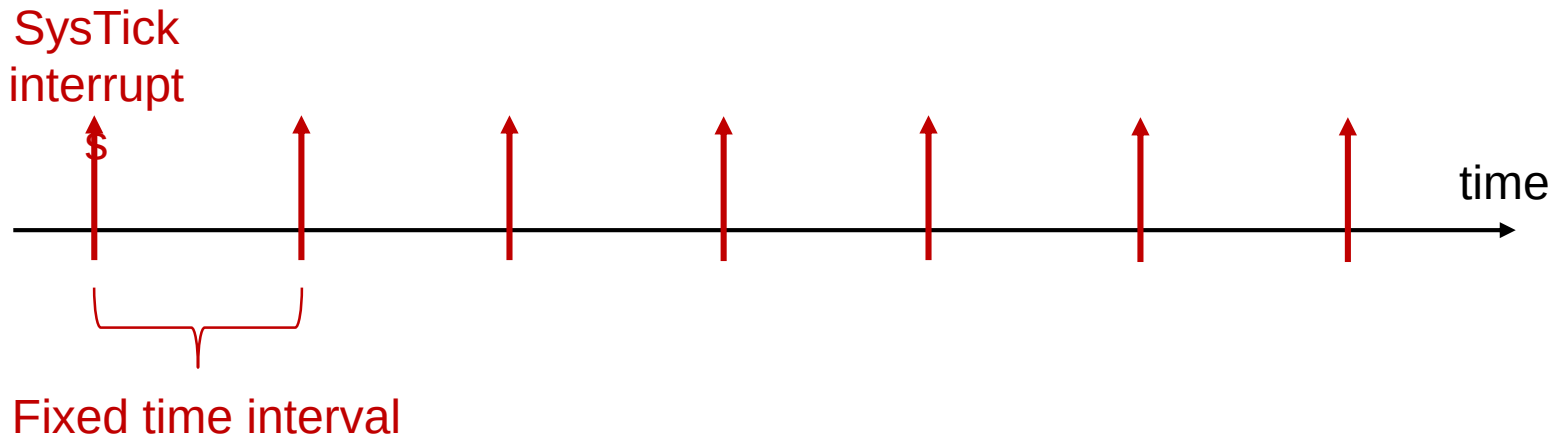


- Common peripheral for microcontrollers
- Based on pre-settable binary counter, enhanced with configurability
  - Count value can be read and written by MCU
  - Count **direction** can often be set to up or down
  - Counter's **clock source** can be selected
    - **Counter mode:** count **pulses** which indicate **events** (e.g. odometer pulses)
    - **Timer mode**: clock source is periodic, so counter value is proportional to **elapsed time** (e.g. stopwatch)
  - Counter's **overflow/underflow action** can be selected
    - Generate interrupt
    - Reload counter with special value and continue counting
    - Toggle hardware output signal
    - Stop!

ARM

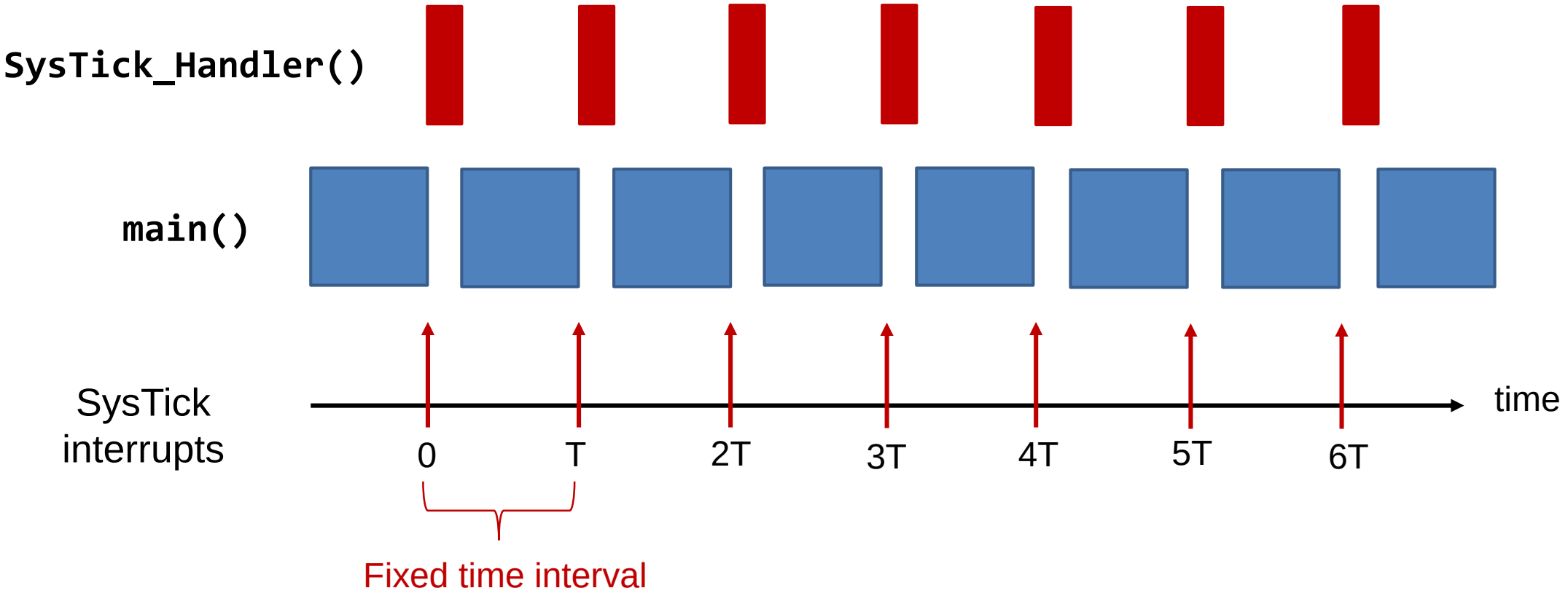
System Timer (SysTick) – *(SysTick Slides are based on the slides of Dr. Yifeng Zhu)*

- Generate SysTick interrupts at a fixed time interval



- Example Usages:
  - Measuring time elapsed, such as time delay function
  - Executing tasks periodically, such as periodic polling, and OS CPU scheduling

System Timer (SysTick)



System Timer (SysTick)

- System timer is a standard hardware component built into ARM Cortex-M.
- This hardware periodically forces the processor to execute the following ISR:

```
void SysTick_Handler(void){
  ...
}
```

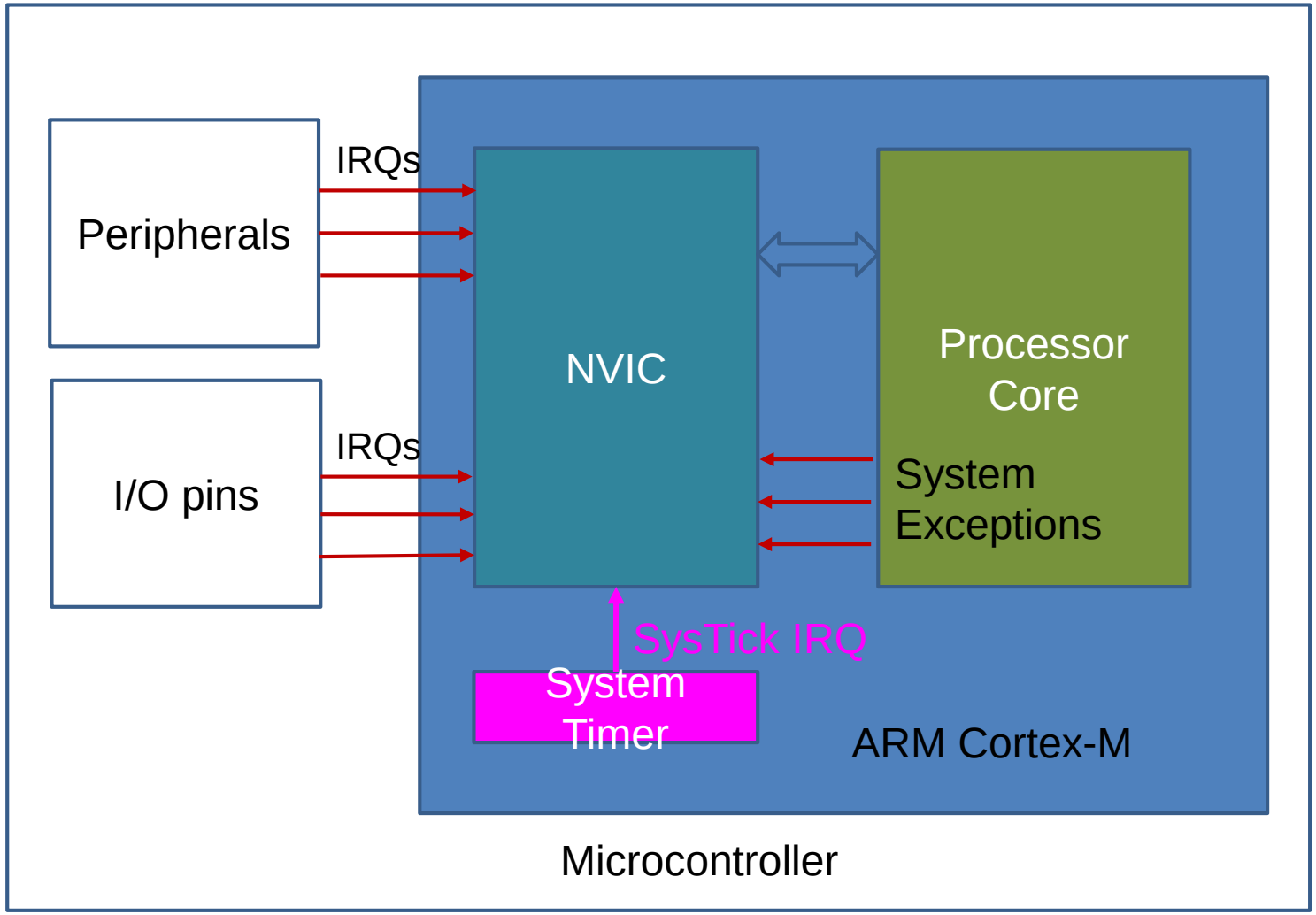
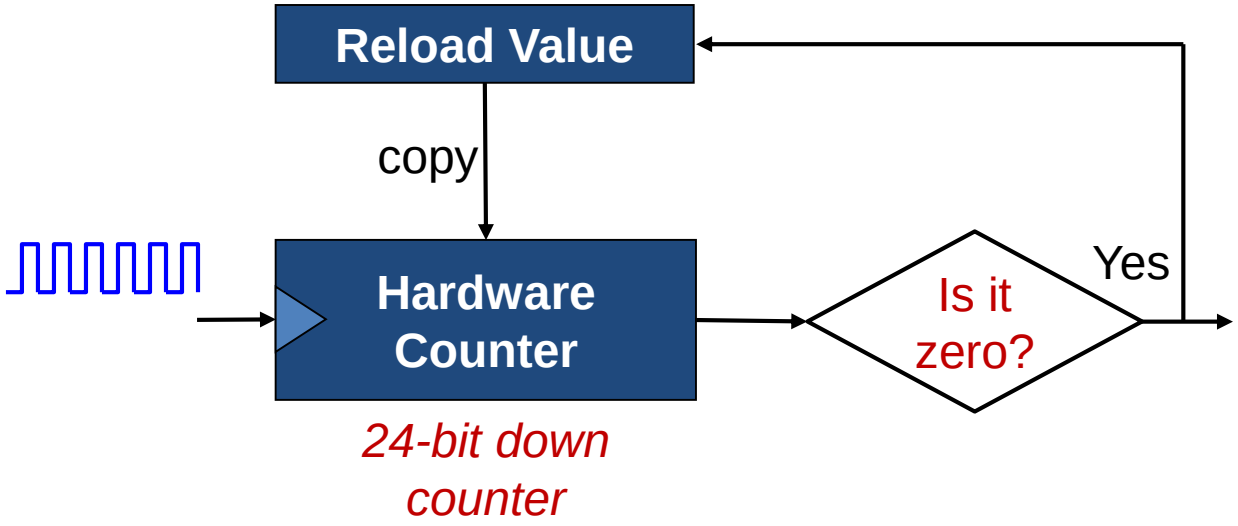
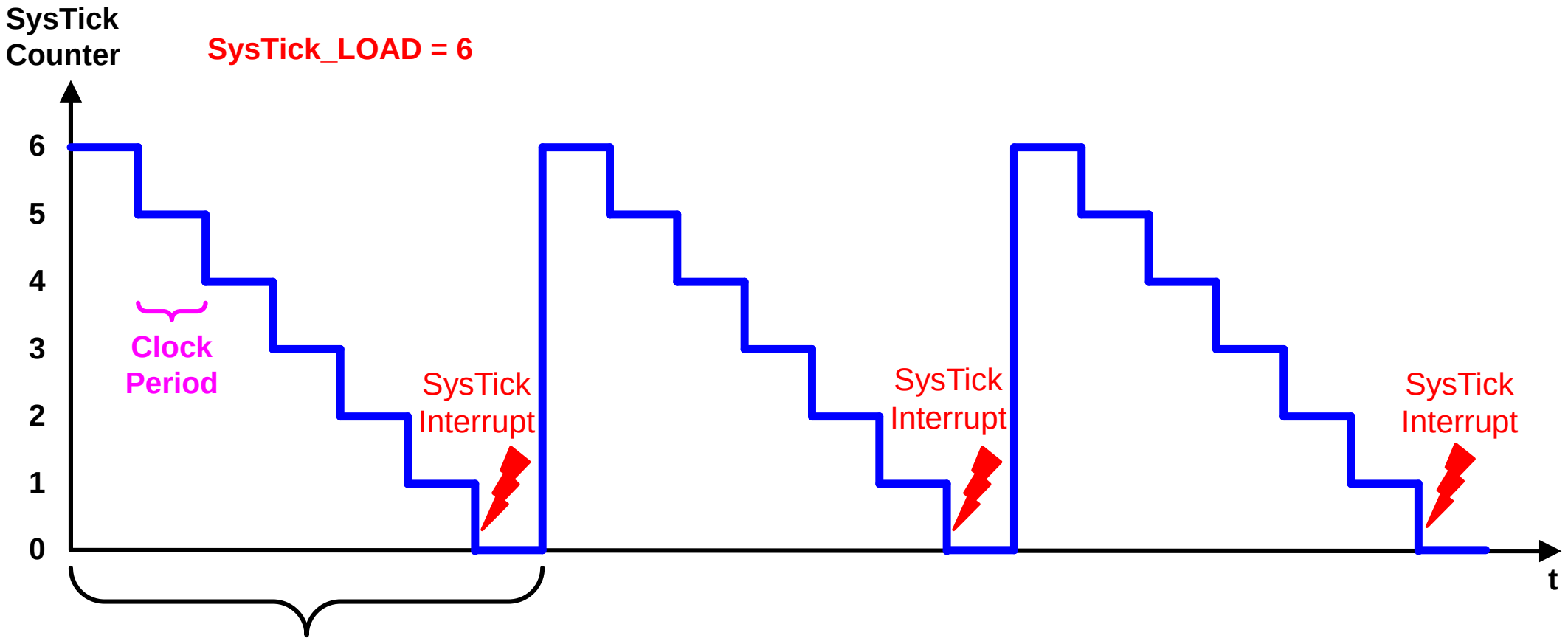


Diagram of System Timer (SysTick)

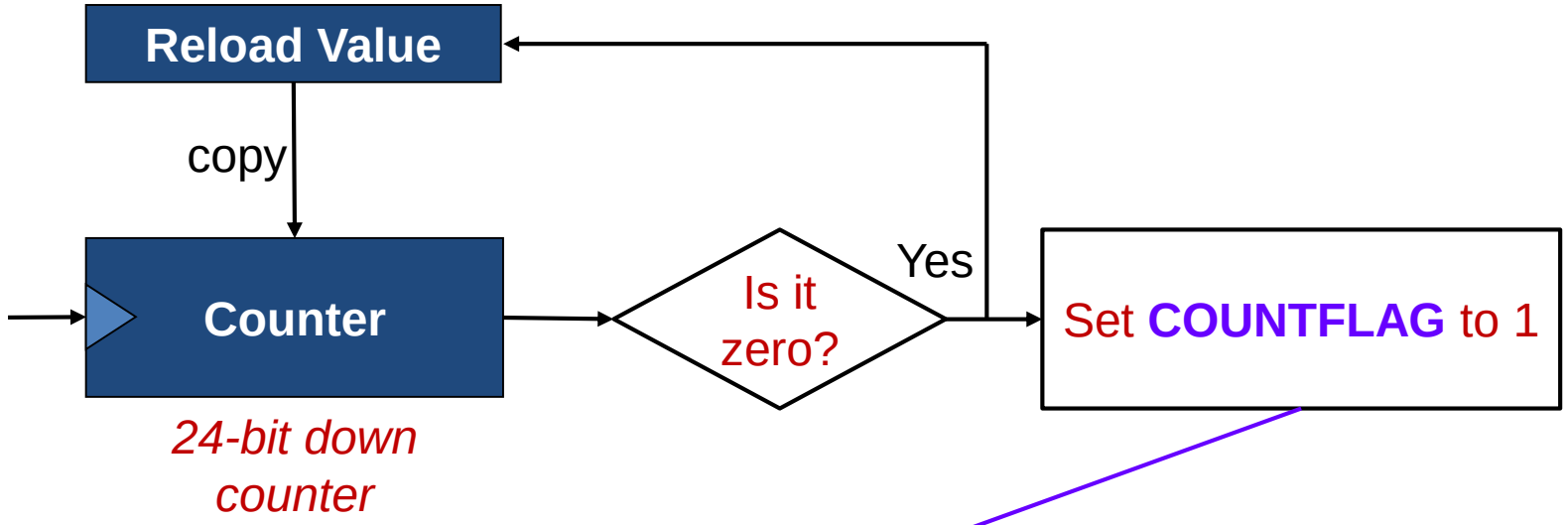


System Timer



SysTick Interrupt Time Period = (SysTick_LOAD + 1) × Clock Period = 7 × Clock Period

Diagram of System Timer (SysTick)



SysTick control and status register (SysTick_CTRL)

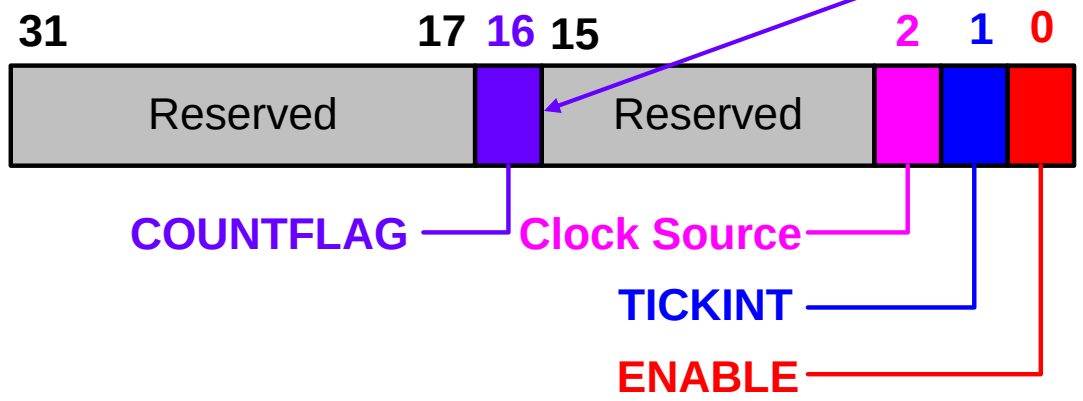
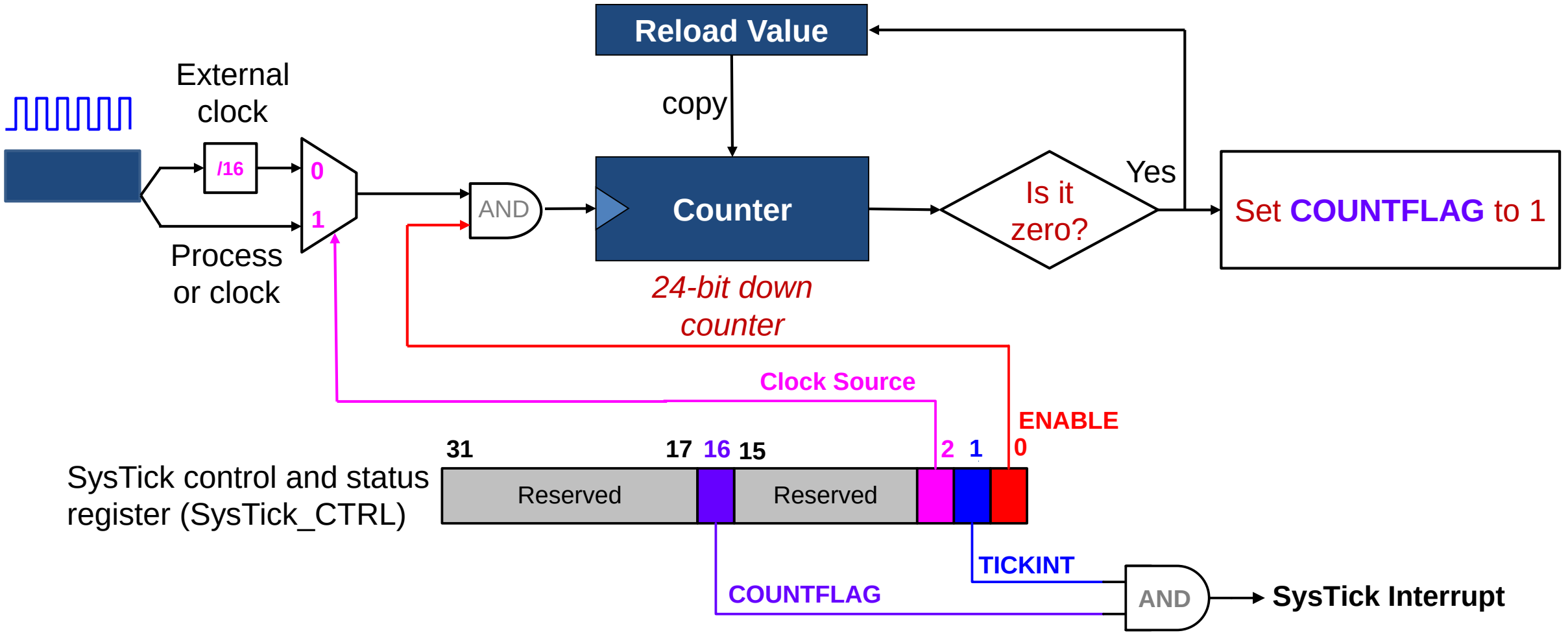
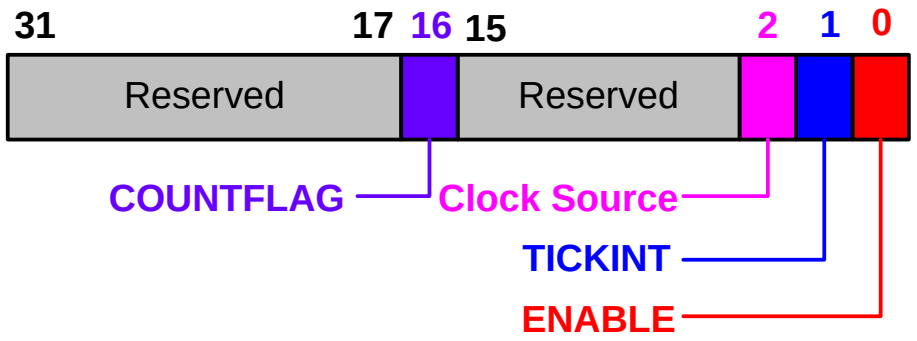


Diagram of System Timer (SysTick)



Registers of System Timer

SysTick control and status register (SysTick_CTRL)



SysTick reload value register (SysTick_LOAD)

SysTick current value register (SysTick_VAL)

Registers of System Timer

SysTick reload value register (SysTick_LOAD)

- 24 bits, maximum value `0x00FF.FFFF` (16,777,215)
- Counter counts down from RELOAD value to 0.
- Writing RELOAD to 0 disables SysTick, independently of TICKINT
- Time interval between two SysTick interrupts

$$Interval = (RELOAD + 1) \times Source\_Clock\_Period$$

- If `100` clock periods between two SysTick interrupts

$$RELOAD = 99$$

Registers of System Timer

SysTick current value register (SysTick_VAL)

- Reading it returns the current value of the counter
- When it transits from `1` to `0`, it generates an interrupt
- Writing to SysTick_VAL clears the counter and COUNTFLAG to zero
  - Cause the counter to reload on the next timer clock
  - But, does not trigger an SysTick interrupt
- It has random value on reset.
  - Always clear it before enabling the timer

Calculating Reload Value

- Suppose clock source = `80MHz`
- Goal: SysTick Interval = `10ms`
- What is RELOAD value?

$$Reload = \frac{10\ ms}{Clock\ Period} - 1$$

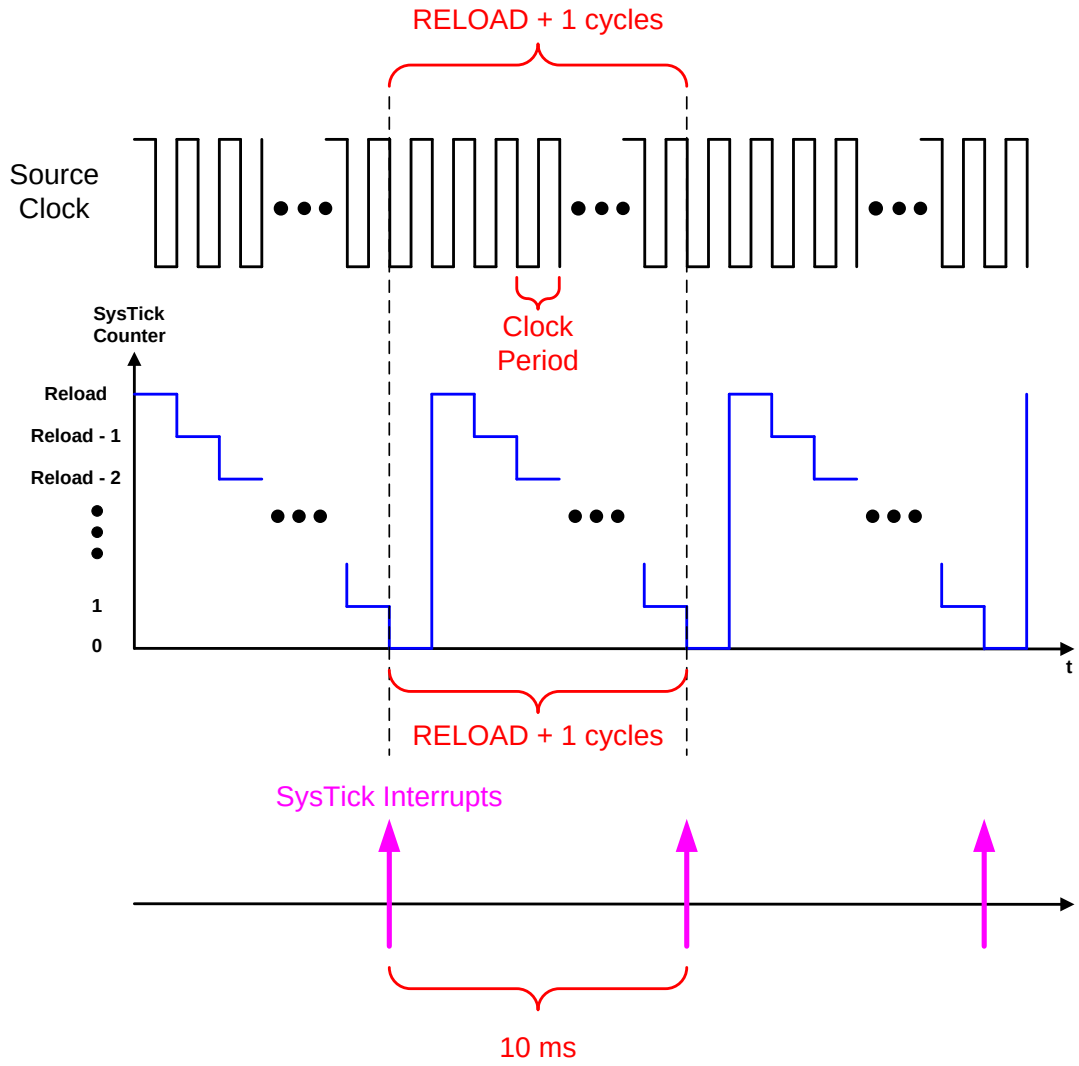
$$= 10ms \times Clock\ Frequency\ - 1$$

$$= 10ms \times 80MHz - 1$$

$$= 10 \times 10^{-3} \times 80 \times 10^{6} - 1$$

$$= 800000 - 1$$

$$= 799999$$



Example Code (Textbook page 189)

```
void Init_SysTick (void) {
    SysTick->CTRL = 0;              // Disable SysTick
    SysTick->LOAD = 0x13FFFF;     // Set reload register to get 1s interrupts clk=20971520
    NVIC_SetPriority(SysTick_IRQn, 3);
    SysTick->VAL = 0;               // Reset the SysTick counter value
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk;
}

void SysTick_Handler() {
static int n=0;
Control_RGB_LEDs(n&1,n&1,n&1);
n++;
}
```

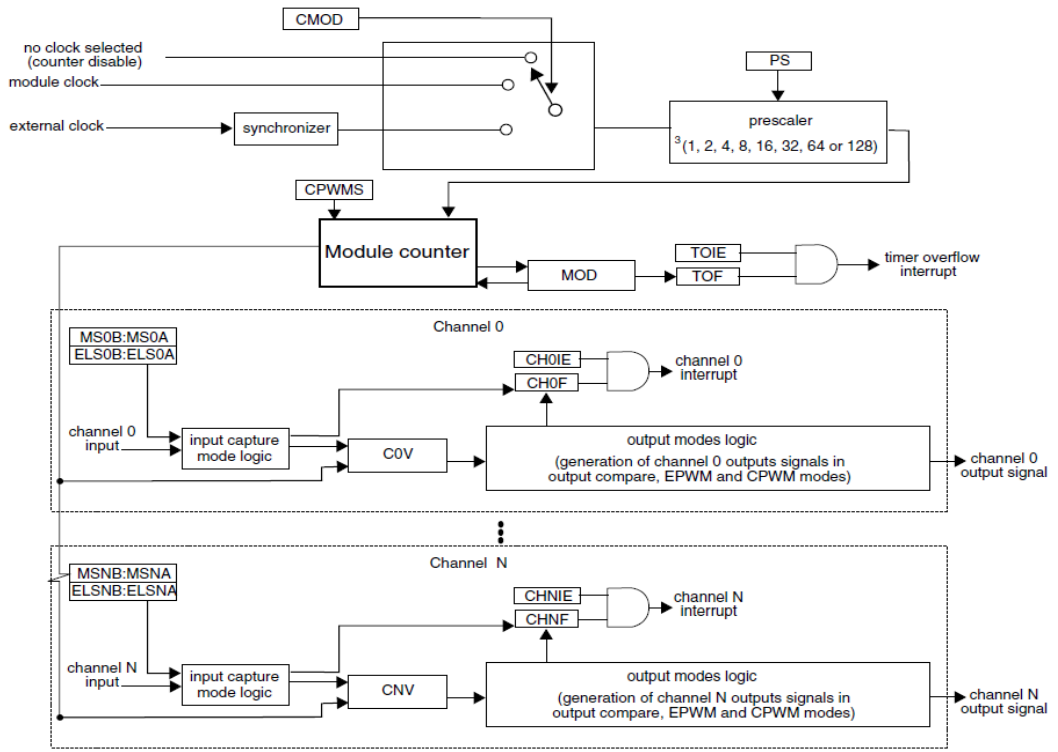

ARM

TIMER/PWM MODULE (TPM)

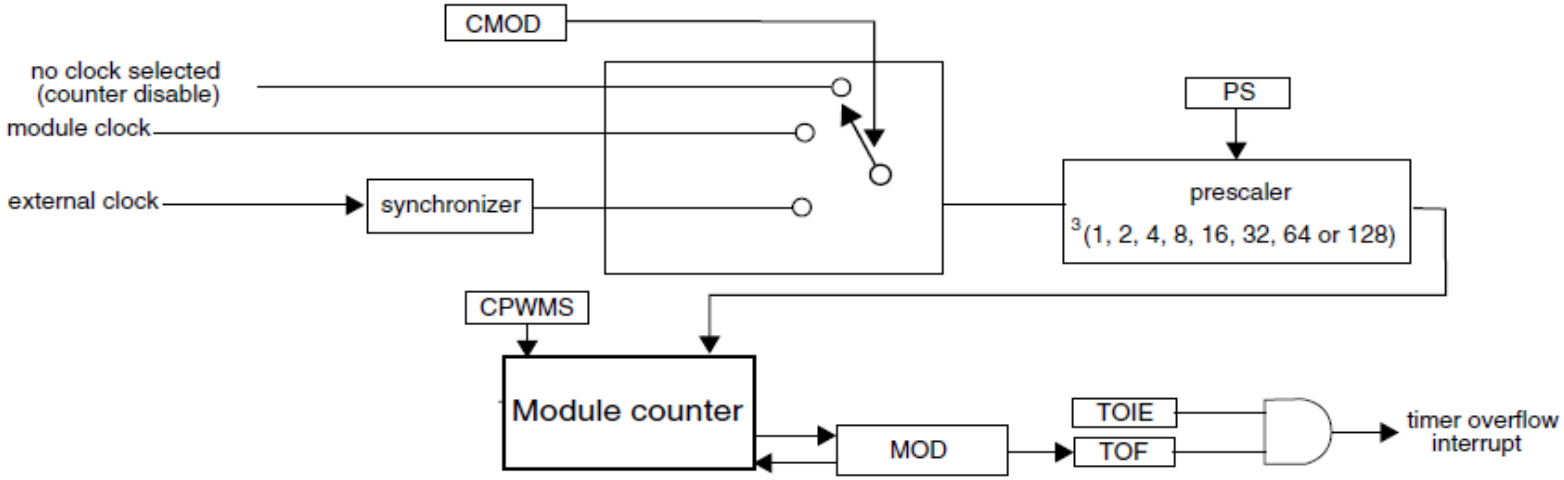
ARM

TPM - Timer/PWM Module

- Core: **Module counter**
  - Two clock options - external or internal
  - Prescaler to divide clock by 1 to 128
  - 16-bit counter
    - Can count up or up/down
    - Can reload with set load value or wrap around (to FFFF or 0000)
- Six **channels**
  - 3 modes
    - Capture Mode: capture timer's value when input signal changes
    - Output Compare: Change output signal when timer reaches certain value
    - PWM: Generate pulse-width-modulated signal. Width of pulse is proportional to specified value.
  - Each channel can generate interrupt, DMA request, hardware trigger on overflow
  - One I/O pin per channel TPM_CHn
  - TPM0 has six channels, TPM1 and TPM2 each have two channels



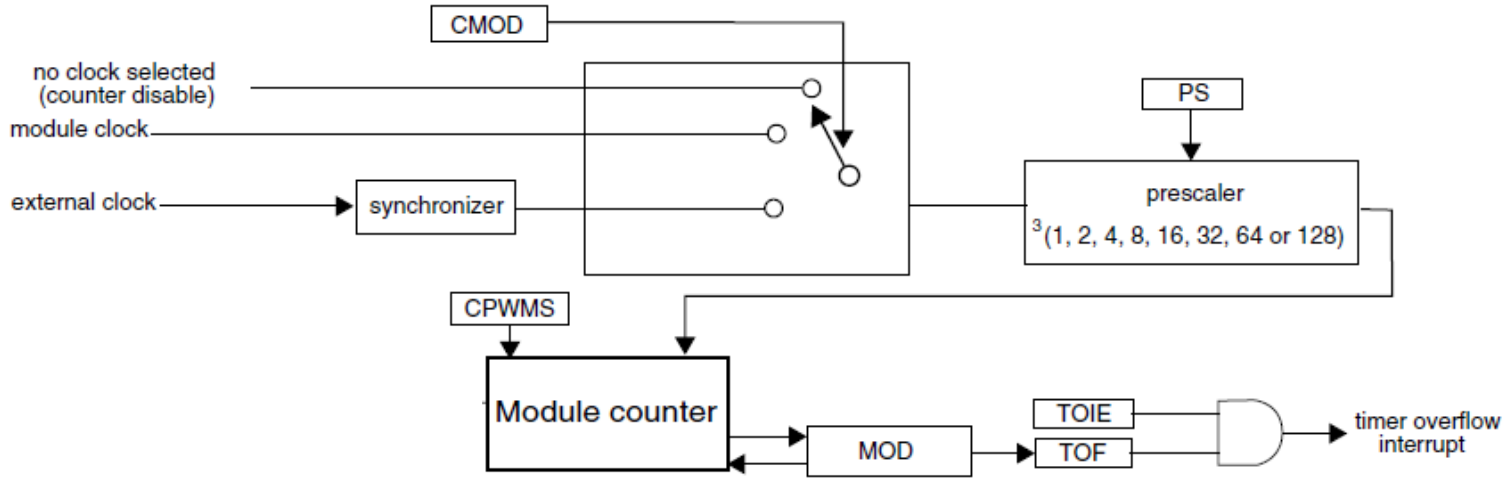
Timer Configuration



- Clock source
  - CMOD: selects internal or external clock
- Prescaler
  - PS: divide selected clock by 1, 2, 4, 8, 16, 32,64, 128
- Count Mode and Modulo
  - CPWMS: count up (0) or up and down (1)
  - MOD: 16-bit value up to which the counter counts
    - Up counting: 0, 1, 2, … MOD, 0/Overflow, 1, 2, … MOD
    - Up/down counting: 0, 1, 2, … MOD, MOD-1/Interrupt, MOD-2, … 2, 1, 0, 1, 2, …
- Timer overflows when counter goes 1 beyond MOD value
- DMA: Enable DMA transfer on overflow
- TOF: Flag indicating timer has overflowed

ARM

Basic Counter Mode



- Count external events applied on input pin
  - Set CMOD = 01 to select external clock
  - Set PS = 000 (unless division needed)
- Timer overflow flag TOF set to 1 upon receiving MOD * prescaler pulses
- Can generate interrupt if TOIE is set

2-0 PS	Prescaler Factor
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

ARM

Count Mode and Modulo - Counting Up

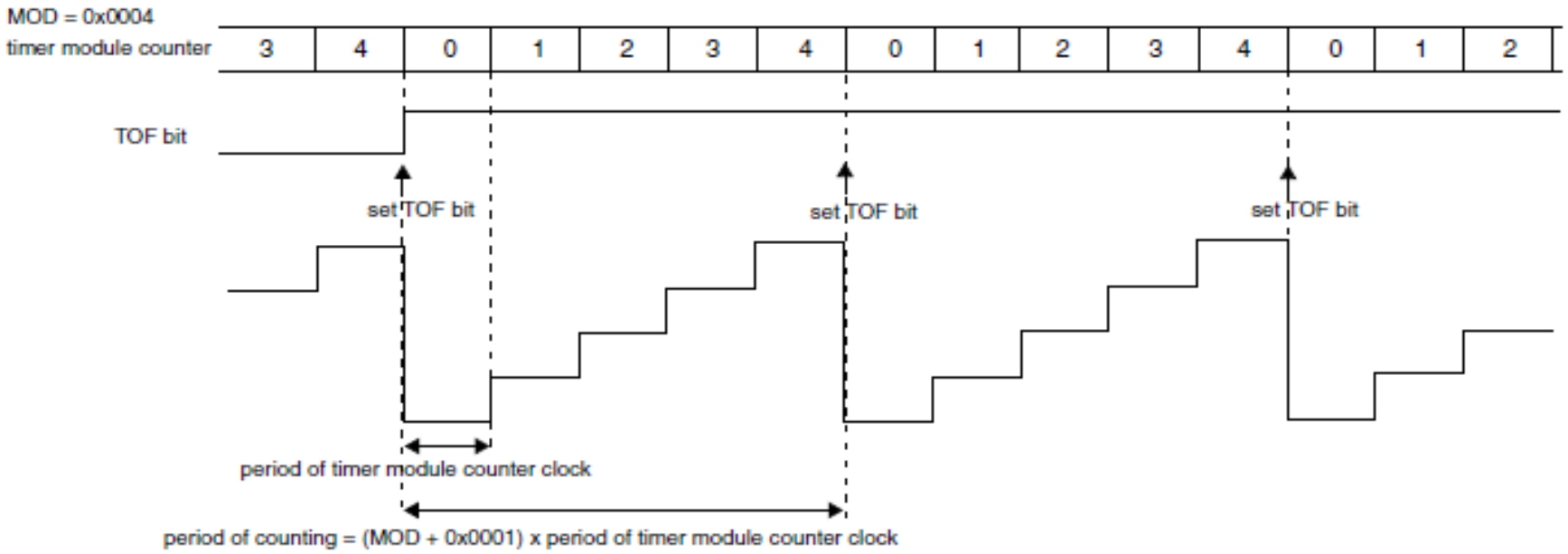


Figure 31-79. Example of TPM Up Counting

- Counter increments with each clock tick
- When counter reaches MOD,
  - set TOF bit (timer overflow)
  - reset counter value to 0
- Frequency of overflows is timer clock frequency / (1 + MOD)

ARM

Count Mode and Modulo - Counting Up and Down

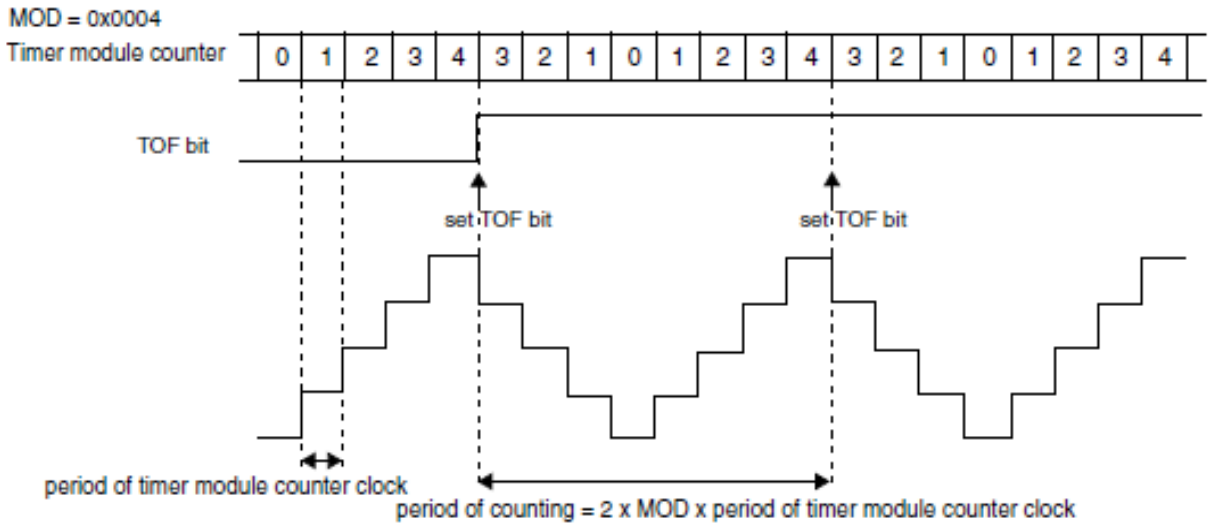
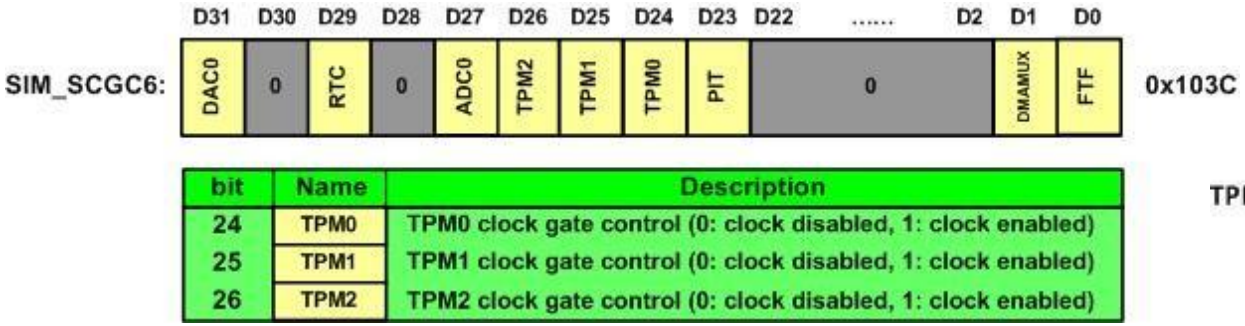


Figure 31-80. Example of Up-Down Counting

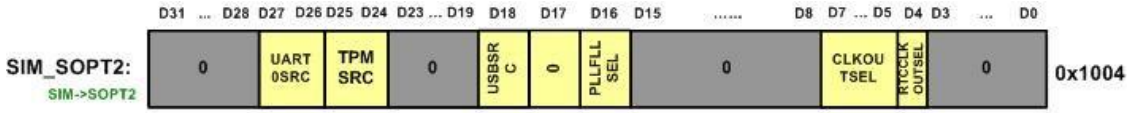
- Two modes
  - Up-counting
  - Counter increments with each clock tick
  - When counter reaches MOD, set TOF bit (timer overflow), set to down-count mode
  - Down-counting
  - Counter decrements with each clock tick
  - When counter reaches 0, set to up-count mode
- Frequency of overflows is timer clock frequency / (2 * MOD)

ARM

Registers

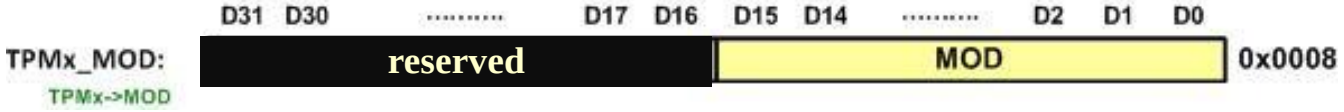


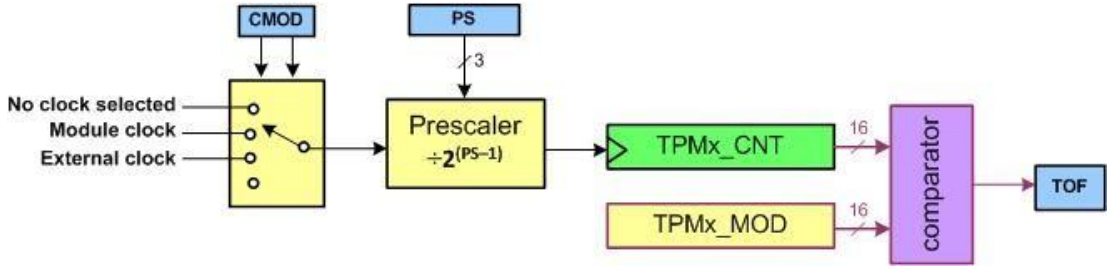
bit	Name	Description
24	TPM0	TPM0 clock gate control (0: clock disabled, 1: clock enabled)
25	TPM1	TPM1 clock gate control (0: clock disabled, 1: clock enabled)
26	TPM2	TPM2 clock gate control (0: clock disabled, 1: clock enabled)

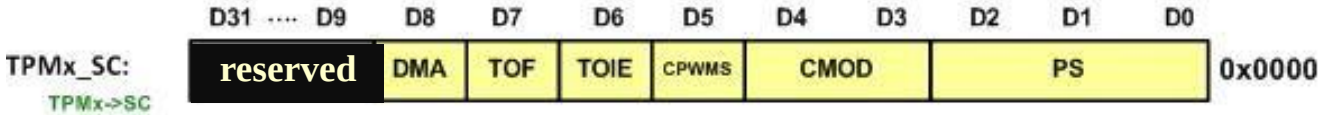


TPM clock source select selects the clock source for the TPM counter clock

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







Field	Bits	Description
PS	0–2	In the prescaler, the clock is divided by 2^PS.
CMOD	3–4	Clock Mode Selection
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	Timer Overflow Interrupt Enable (0: Disabled, 1: Enabled).
TOF	7	Timer Overflow Flag
DMA	8	DMA Enable (0: Disabled, 1: Enabled)

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

Program 5.4 from Mazidi etal

```
/* p5_4.c Toggling blue LED using TPM0 delay
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. */
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 timer counter 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 */
    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 */
    }
}
```


ARM

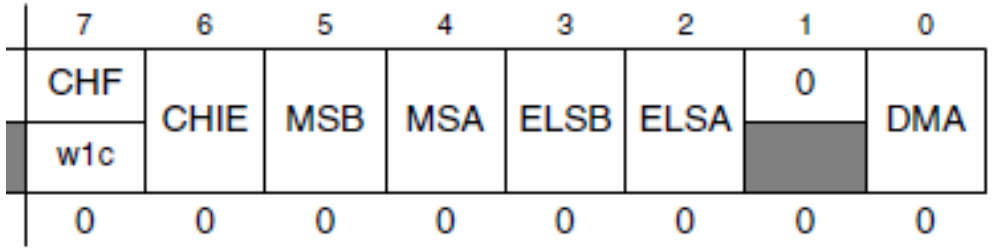
Major Channel Modes

- Input Capture Mode
  - Capture timer's value when input signal changes
    - Rising edge, falling edge, both
  - How long after I started the timer did the input change?
    - Measure time delay
- Output Compare Mode
  - Modify output signal when timer reaches specified value
    - Set, clear, pulse, toggle (invert)
  - Make a pulse of specified width
  - Make a pulse after specified delay
- **Pulse Width Modulation**
  - **Make a series of pulses of specified width and frequency**

ARM

Channel Configuration and Value

- Configuration: TPMx_CnSC



7	6	5	4	3	2	1	0
CHF	CHIE	MSB	MSA	ELSB	ELSA	0	DMA
w1c							
0	0	0	0	0	0	0	0

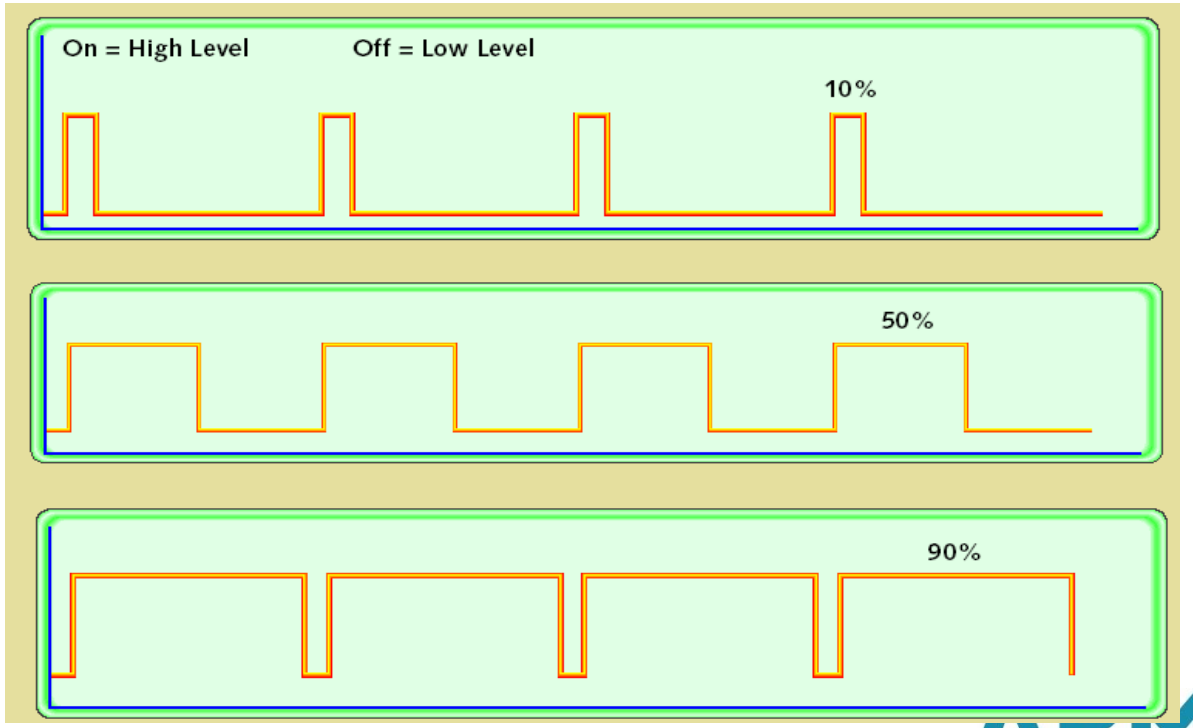
  - CHF - set when event occurs
  - CHIE - enable channel to generate an interrupt
  - MSB:MSA - mode select
  - ELSB:ELSA - edge or level select
  - DMA - enable DMA transfers

- Value: TPMx_CnV
  - 16-bit value for output compare or input capture

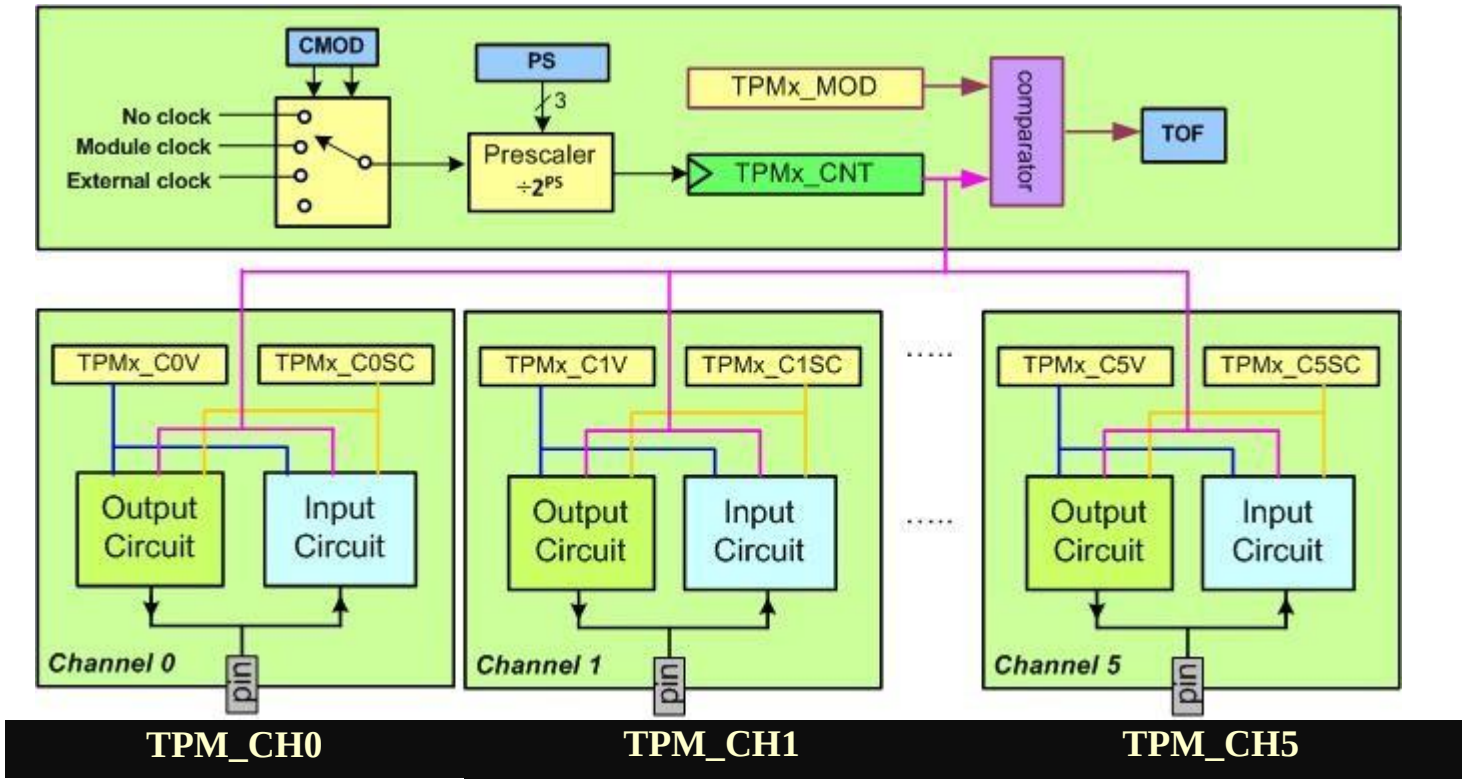
ARM

Pulse-Width Modulation

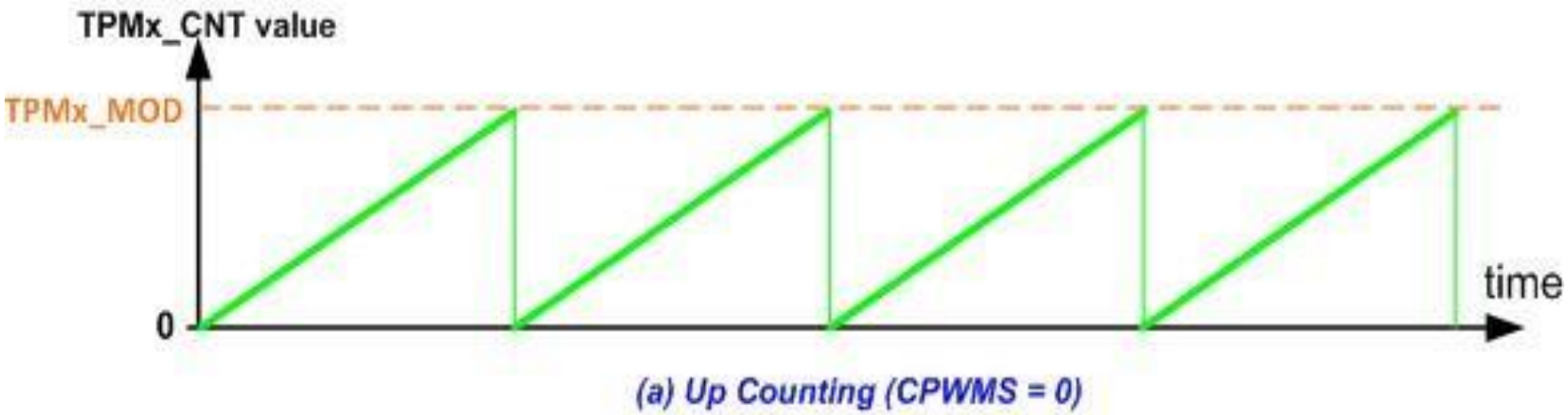
- Uses of PWM
  - **Digital power amplifiers** are more efficient and less expensive than analog power amplifiers
    - Applications: motor speed control, light dimmer, switch-mode power conversion
    - Load (motor, light, etc.) responds slowly, averages PWM signal
  - **Digital communication** is less sensitive to noise than analog methods
    - PWM provides a *digital encoding* of an *analog* value
    - Much less vulnerable to noise
- PWM signal characteristics
  - Modulation frequency – how many pulses occur per second (fixed)
  - Period – 1/(modulation frequency)
  - On-time – amount of time that each pulse is on (asserted)
  - Duty-cycle – on-time/period
  - Adjust *on-time* (hence *duty cycle*) to represent the analog value



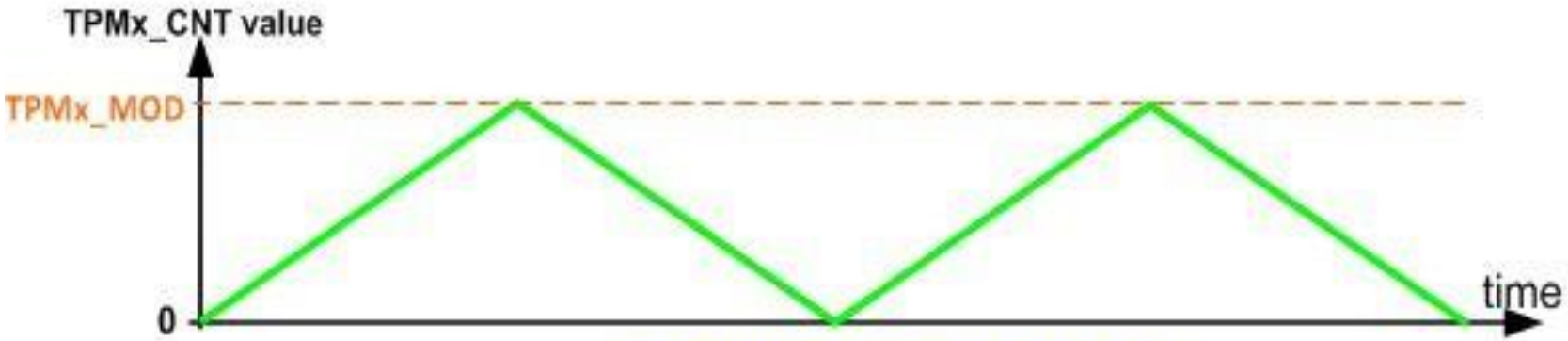
CMOD and PS (Prescaler) bits



Up/Down-Counter and Up-Counter



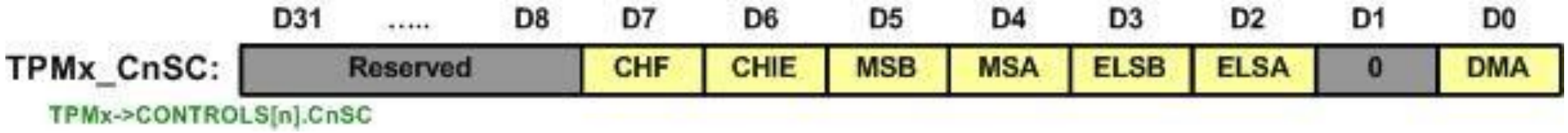
(a) Up Counting (CPWMS = 0)



(b) Up-down Counting (CPWMS = 1)

ARM

TPMxCnSC (TPMx Channel Status and Control)

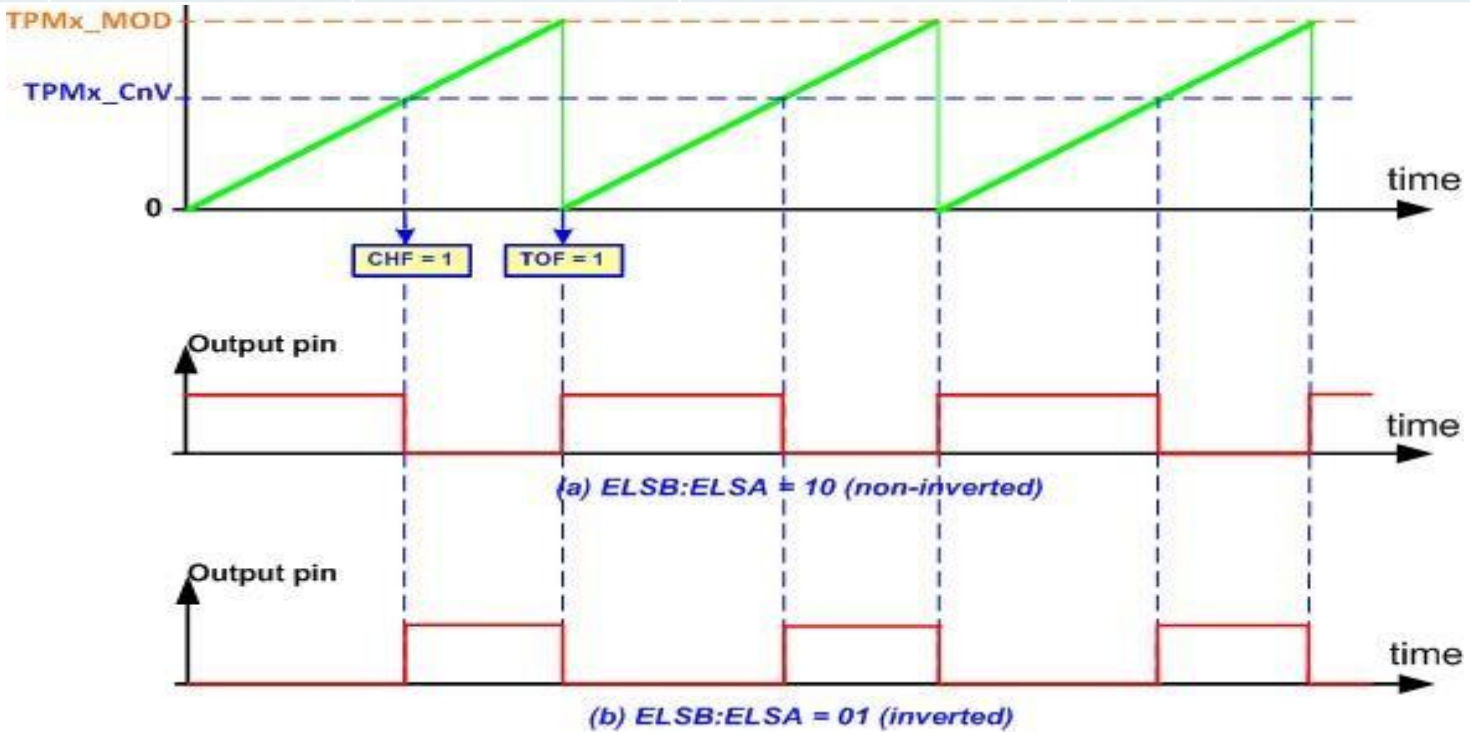


Field	Bit	Description
CHF	7	Channel Flag
CHIE	6	Channel interrupt enable
MSB and MSA	5-4	Channel mode select
ELSB and ELSA	3-2	Edge or Level Select
DMA	0	DMA enable (0: Disabled, 1: Enabled)

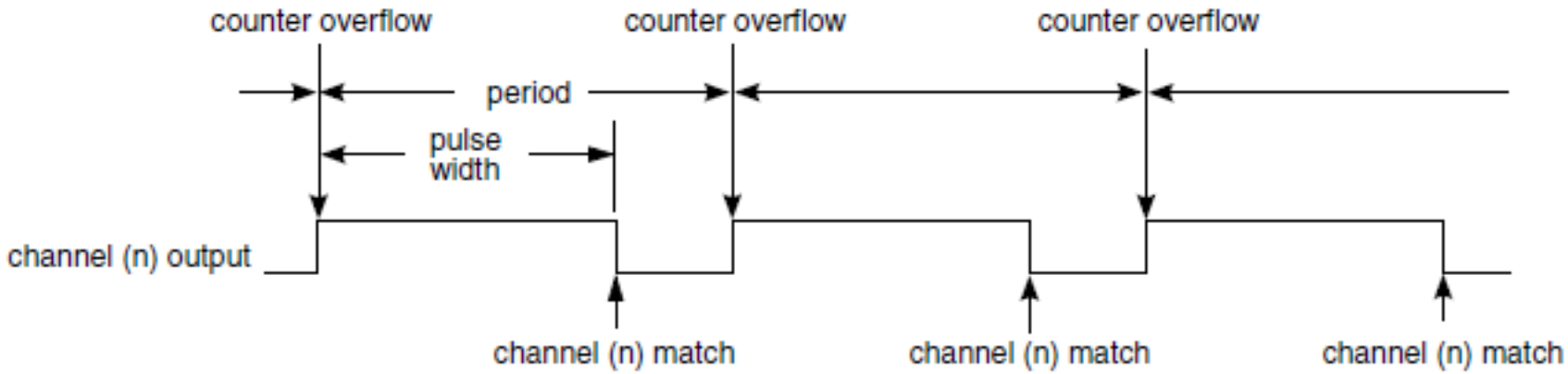
ARM

Edge-Aligned PWM

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
0	10	10	Edge-Aligned PWM (non-inverted)	Set output on reload, clear output on match
0	10	01 or 11	Edge-Aligned PWM (inverted)	Clear output on reload, set output on match



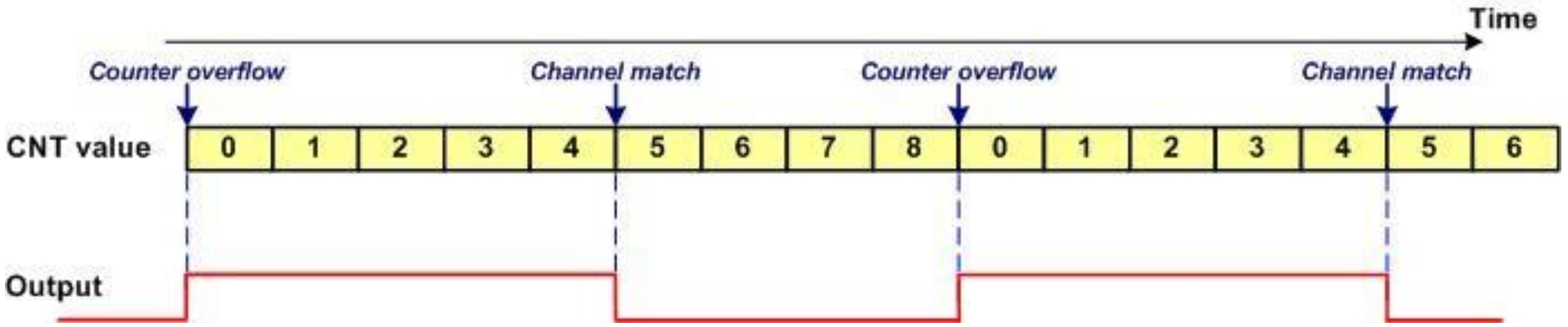
TPM Channel for PWM Mode



- Edge-aligned - leading edges of signals from all PWM channels are aligned
  - Uses count up mode
  - Period = (MOD + 1) cycles
  - Pulse width = (CnV) cycles
- MSnB:MSnA = 01, CPWMS = 0
  - ELSnB:ELSnA = 10 - high-true pulses
  - ELSnB:ELSnA = x1 - low-true pulses

ARM

The PWM output for MOD = 8, CnV = 5, ELSB:ELSA = 10 (non-inverted)



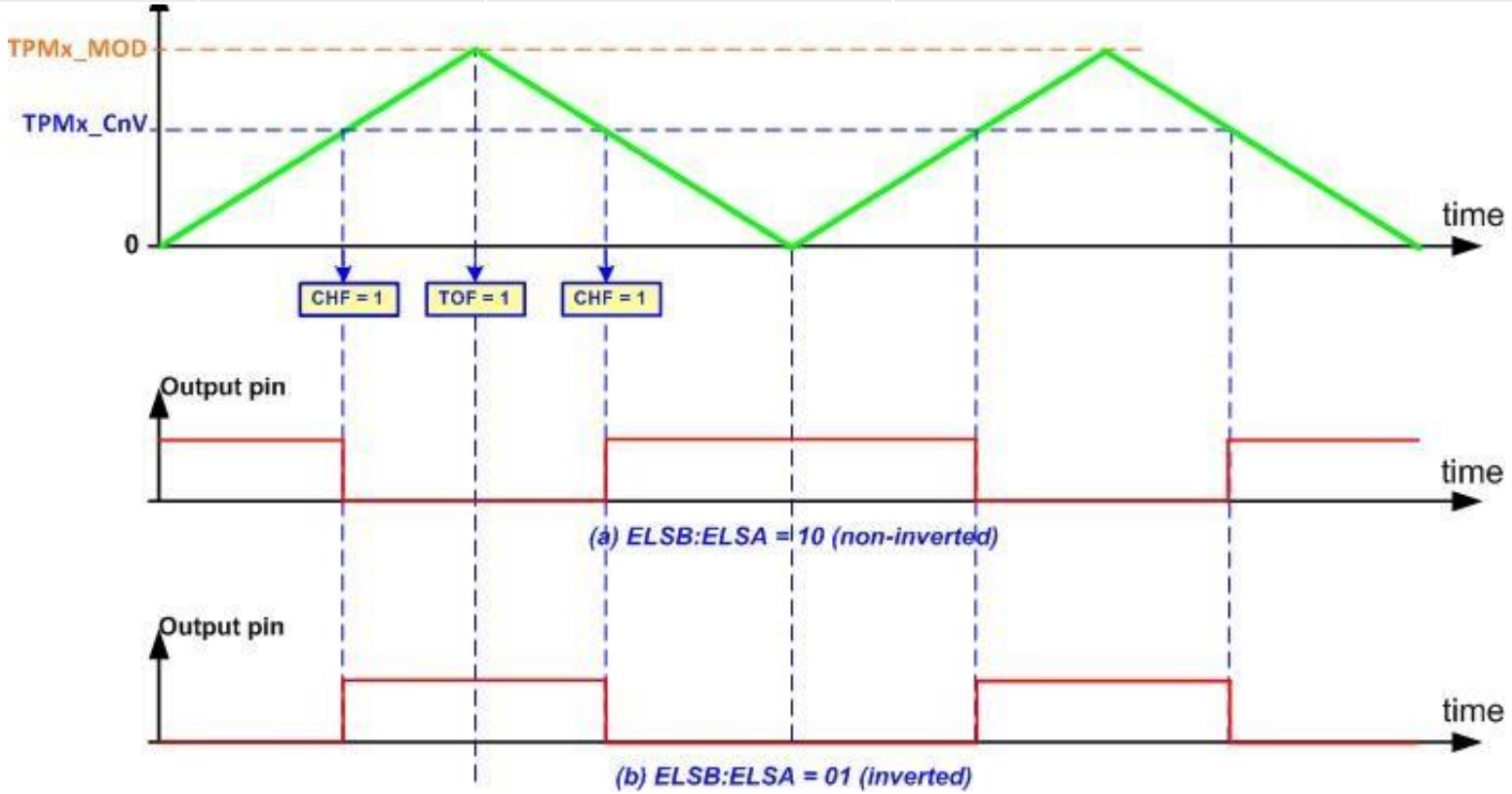
$$\text{Duty Cycle} = \frac{\text{CnV}}{\text{MOD} + 1} \times 100$$

$$F_{generated\ wave} = \frac{F_{timer\ clock} / \text{prescaler}}{\text{MOD+1}} = \frac{F_{timer\ clock}}{(\text{MOD} + 1) \times 2^{PS}}$$

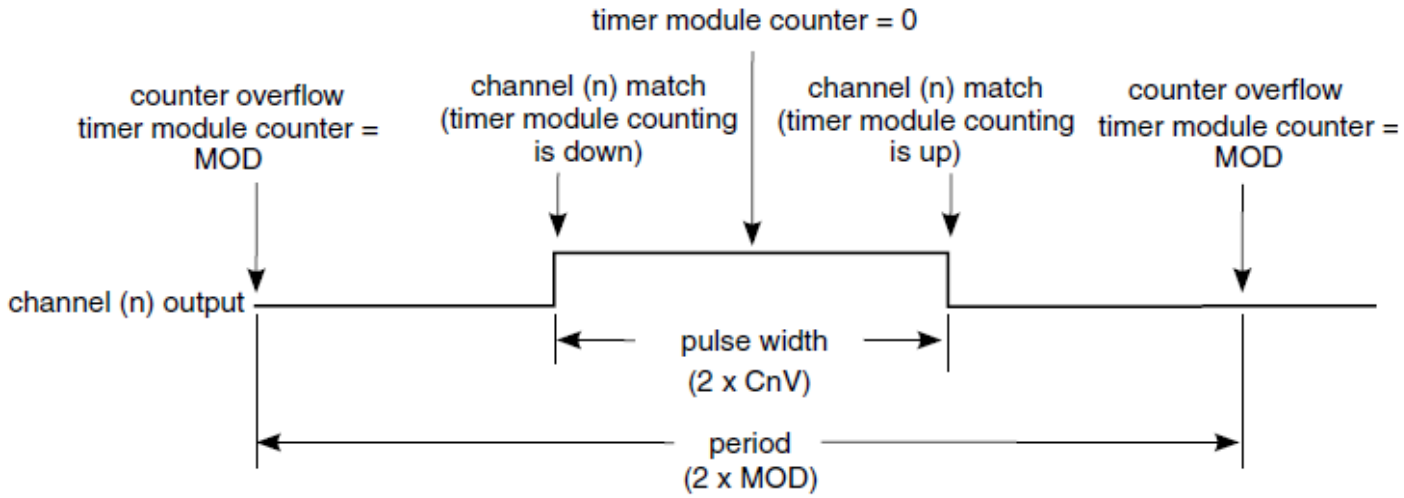
ARM

Center-Aligned PWM

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
1	10	10	Center-Aligned PWM	Clear output on match-up, set output on match-down
1	10	X1	Center-Aligned PWM	Set output on match-up, clear output on match-down



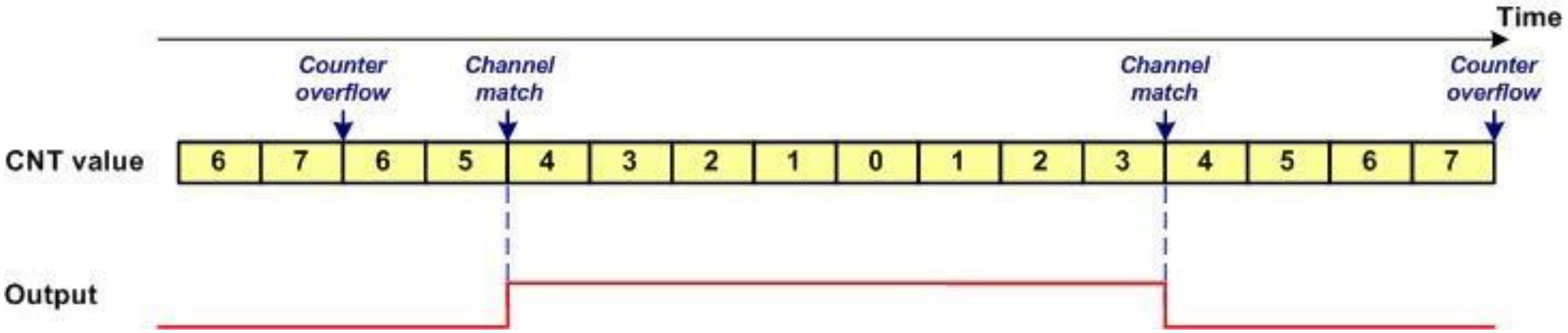
TPM Channel for PWM Mode



- Center-aligned - centers of signals from all PWM channels are aligned
  - Uses count up/down mode
  - Period = 2*MOD cycles.
  - Pulse width = 2*CnV cycles
- MSnB:MSnA = 10, CPWMS = 1
  - ELSnB:ELSnA = 10 - high-true pulses
  - ELSnB:ELSnA = x1 - low-true pulses

ARM

The PWM output for MOD = 7, CnV = 4, ELSB:ELSA = 10 (non-inverted)

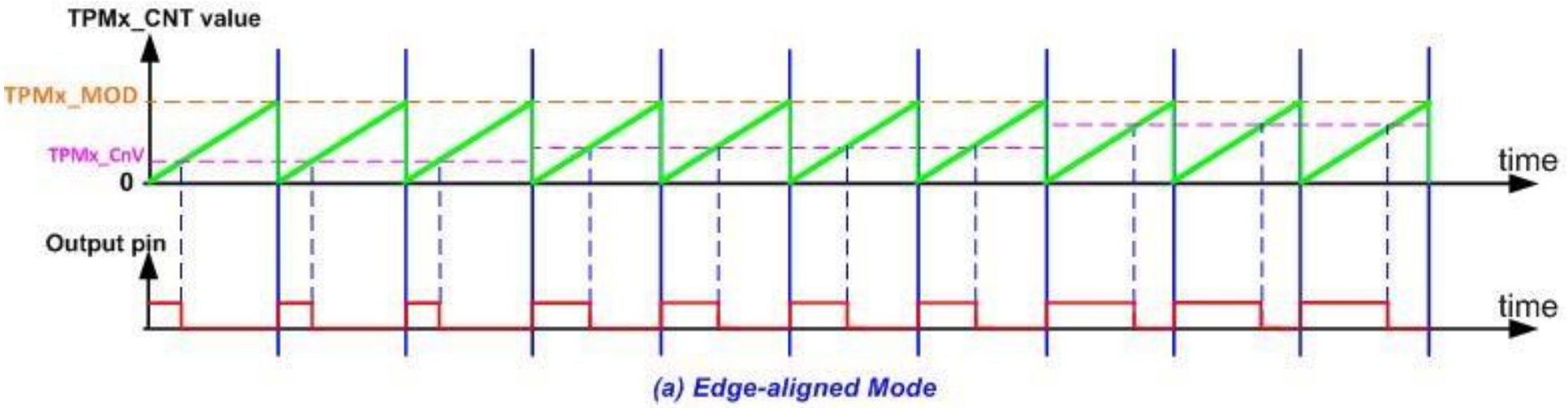


$$\text{Duty Cycle} = \frac{\text{CnV} \times 2}{\text{MOD} \times 2} \times 100 = \frac{\text{CnV}}{\text{MOD}} \times 100$$

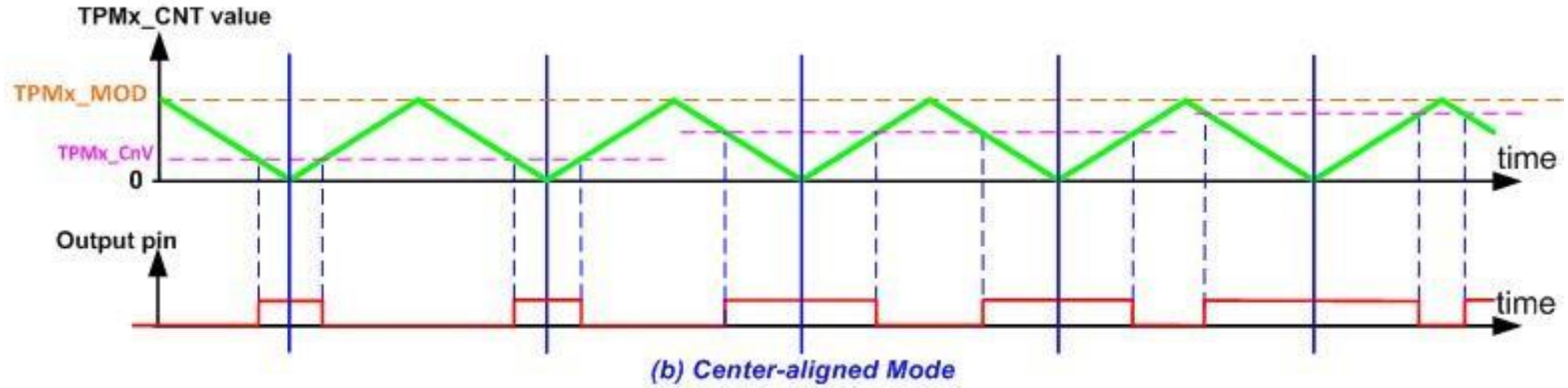
$$F_{generated\ wave} = \frac{F_{timer\ clock} / \text{prescaler}}{\text{MOD} \times 2} = \frac{F_{timer\ clock}}{\text{MOD} \times 2^{PS+1}}$$

ARM

Edge-aligned vs. Center-aligned Mode



(a) Edge-aligned Mode



(b) Center-aligned Mode

ARM

Program 11.4 from Mazidi etal

```
/* Generate 30Hz 40% center-aligned PWM
* TPM0 uses MCGFLLCLK which is 41.94 MHz.
* The prescaler is set to divide by 16.
* The modulo register is set to 43703 and the CnV
* register is set to 17481. TPM0 channel 1 is
* configured to be center-aligned pulse high. */
#include <MKL25Z4.H>
int main (void) {
SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
PORTD->PCR[1] = 0x0400; /* PTD1 used by TPM0 */
SIM->SCGC6 |= 0x01000000; /* enable clock to TPM0 */
SIM->SOPT2 |= 0x01000000; /* use MCGFLLCLK as timer counter clock */
TPM0->SC = 0; /* disable timer */
TPM0->CONTROLS[1].CnSC = 0x20 | 0x08; /* center-aligned, pulse high */
TPM0->MOD = 43703; /* Set up modulo register for 1 kHz */
TPM0->CONTROLS[1].CnV = 17481; /* Set up channel value for 40% duty cycle */
TPM0->SC = 0x0C | 0x20; /* enable TPM0 with prescaler /16, center-aligned */
while (1) {
}
}
```

73	PTD0	DISABLED		PTD0	SPI0_PCS0		TPM0_CH0			
74	PTD1	ADC0_SE5b	ADC0_SE5b	PTD1	SPI0_SCK		TPM0_CH1			

ARM