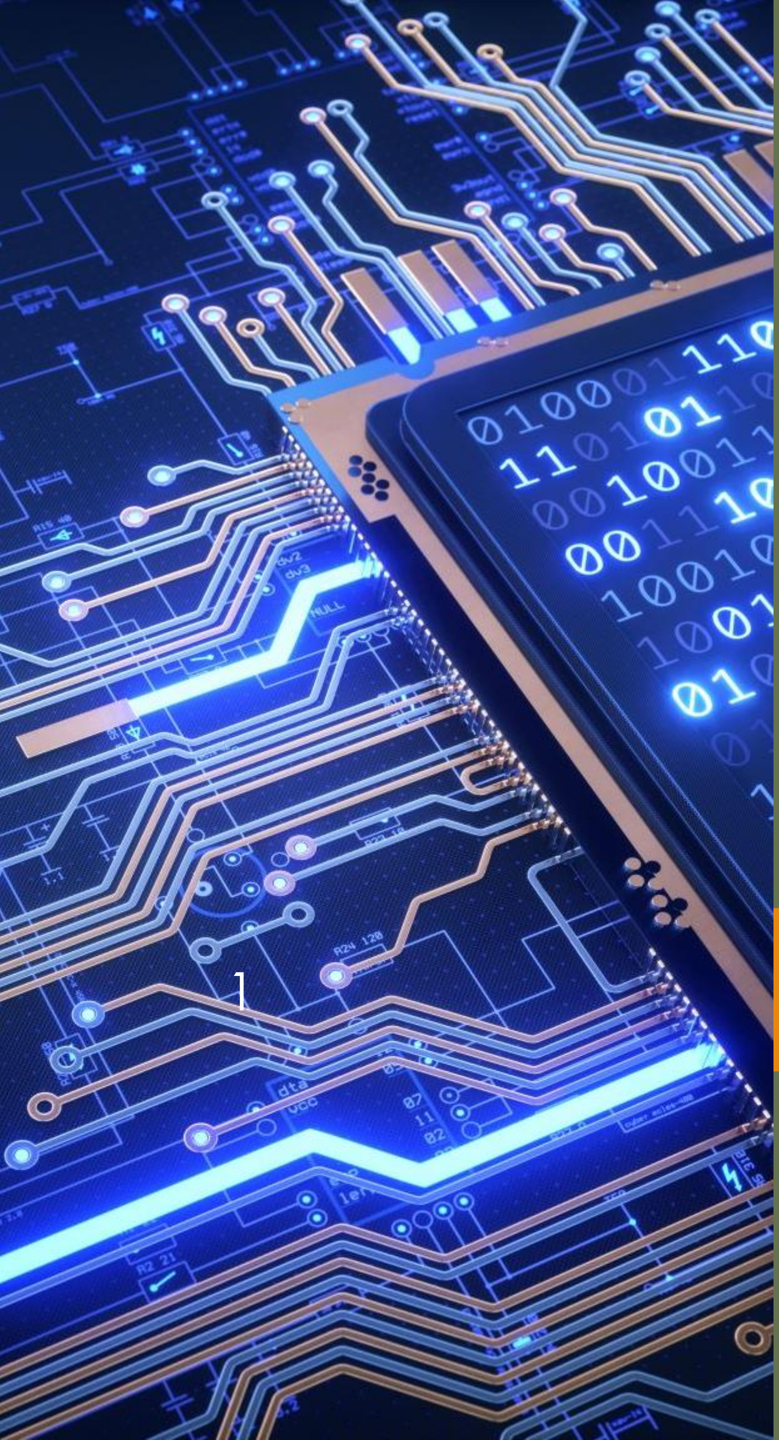


# Embedded Controller programming for Real Time Systems:   ECE-40097

Lesson 7

Vijay Kumar

# Main Topics

TIMERS

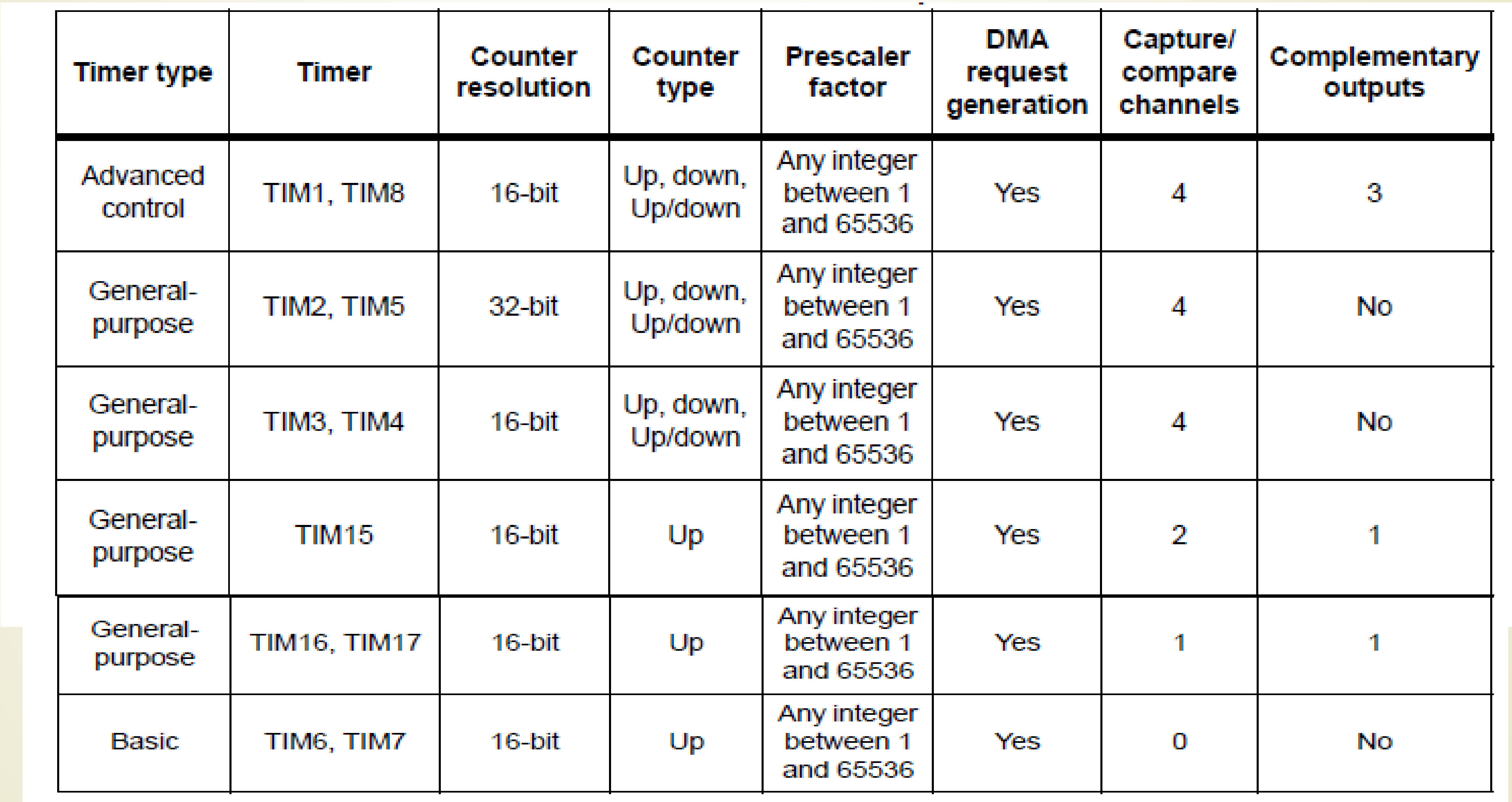
SYSTICK (SYSTEM TIMER)

WATCHDOG TIMERS

# Processor Timers

- 16 timers:
  - 2x 16-bit advanced motor-control,
  - 2x 32-bit **general purpose**
  - *5x 16-bit **general purpose***
  - 2x 16- bit basic
  - 2x low-power 16-bit timers (available in Stop mode)
  - **2x watchdogs**
  - **SysTick timer**

# Timers



| Timer type | Timer | Counter resolution | Counter type | Prescaler factor | DMA request generation | Capture/ compare channels | Complementary outputs |
|---|---|---|---|---|---|---|---|
| Advanced control | TIM1, TIM8 | 16-bit | Up, down, Up/down | Any integer between 1 and 65536 | Yes | 4 | 3 |
| General-purpose | TIM2, TIM5 | 32-bit | Up, down, Up/down | Any integer between 1 and 65536 | Yes | 4 | No |
| General-purpose | TIM3, TIM4 | 16-bit | Up, down, Up/down | Any integer between 1 and 65536 | Yes | 4 | No |
| General-purpose | TIM15 | 16-bit | Up | Any integer between 1 and 65536 | Yes | 2 | 1 |
| General-purpose | TIM16, TIM17 | 16-bit | Up | Any integer between 1 and 65536 | Yes | 1 | 1 |
| Basic | TIM6, TIM7 | 16-bit | Up | Any integer between 1 and 65536 | Yes | 0 | No |

# Timers

- Timers are versatile and provide multiple operating modes to off-load CPU from repetitive and time critical tasks
- Provides timing resources for Software
  - Time-out Event generation
  - Time triggers
- Provides timing resources for Hardware
  - Related to I/O
    - To generate waveforms
    - Measure incoming signal parameters
  - React to external events

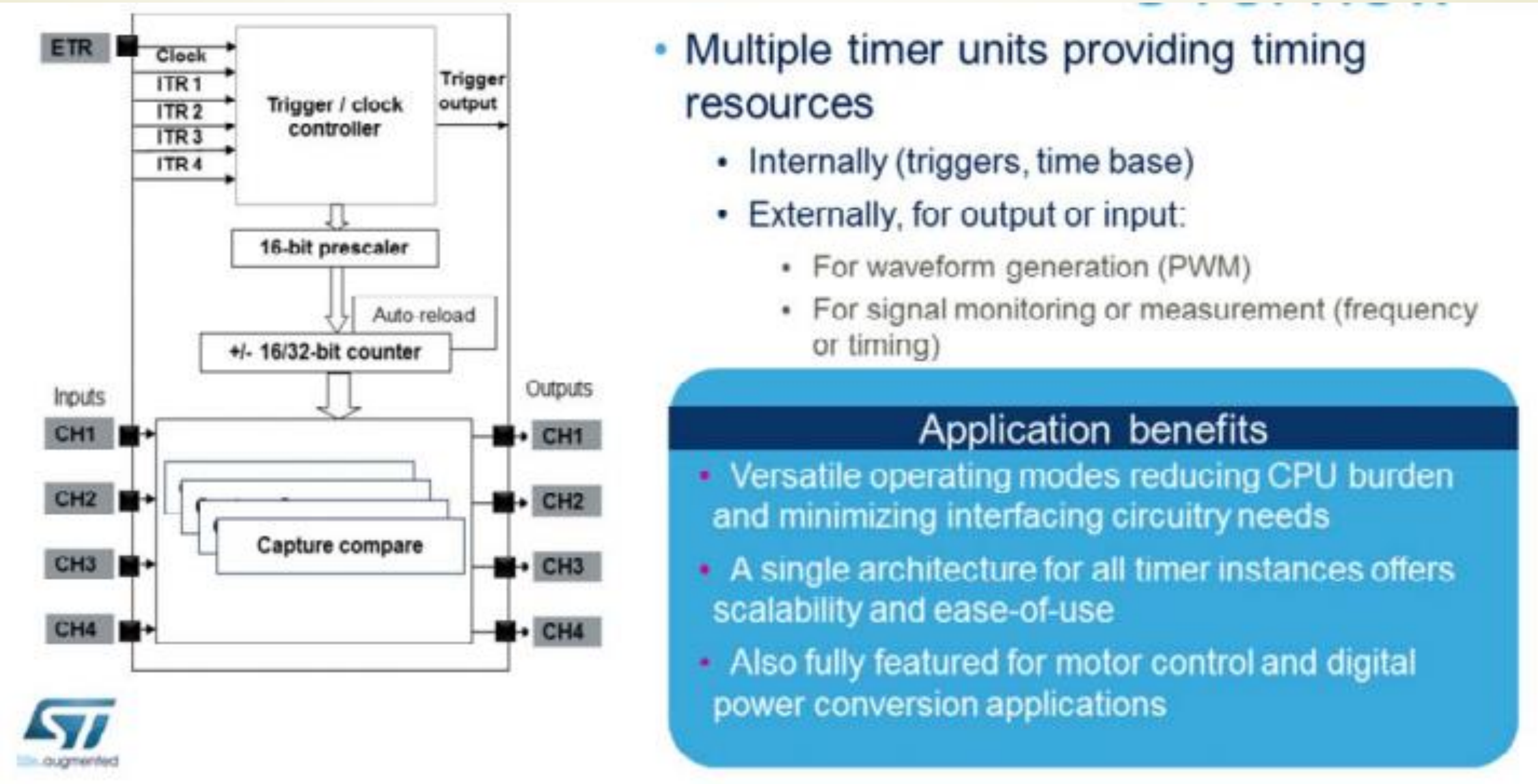
# Processor Timers

- All timers have same architecture
- Each channel could be configured as Input or Output
- Could be time-chained
- Basic timers
    - Used primarily for DAC

# Advanced Timer

- TIM1 timer features include:
- 16-bit up, down, up/down auto-reload counter
- 16-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency either by any factor between 1 and 65536.
- Up to 4 independent channels for
  - Input Capture
  - Output
    - Output Compare
    - PWM generation (Edge and Center-aligned Mode)
  - One-pulse mode output

# Overview



- Multiple timer units providing timing resources
  - Internally (triggers, time base)
  - Externally, for output or input:
    - For waveform generation (PWM)
    - For signal monitoring or measurement (frequency or timing)

**Application benefits**

- Versatile operating modes reducing CPU burden and minimizing interfacing circuitry needs
- A single architecture for all timer instances offers scalability and ease-of-use
- Also fully featured for motor control and digital power conversion applications

# Timer registers

- Four important registers are:
  - Counter register (TIMx_CNT)
  - Prescaler register (TIMx_PSC)
  - Auto-reload register (TIMx_ARR)
  - Repetition counter register (TIMx_RCR)
- Prescaler (PSC) is used to divide the clock
  - Large prescalar value reduces timer resolution
- Auto Reload register (ARR) defines the counting period
- Capture and compare (CCR) holds the initial value
  - Used for input capture Mode

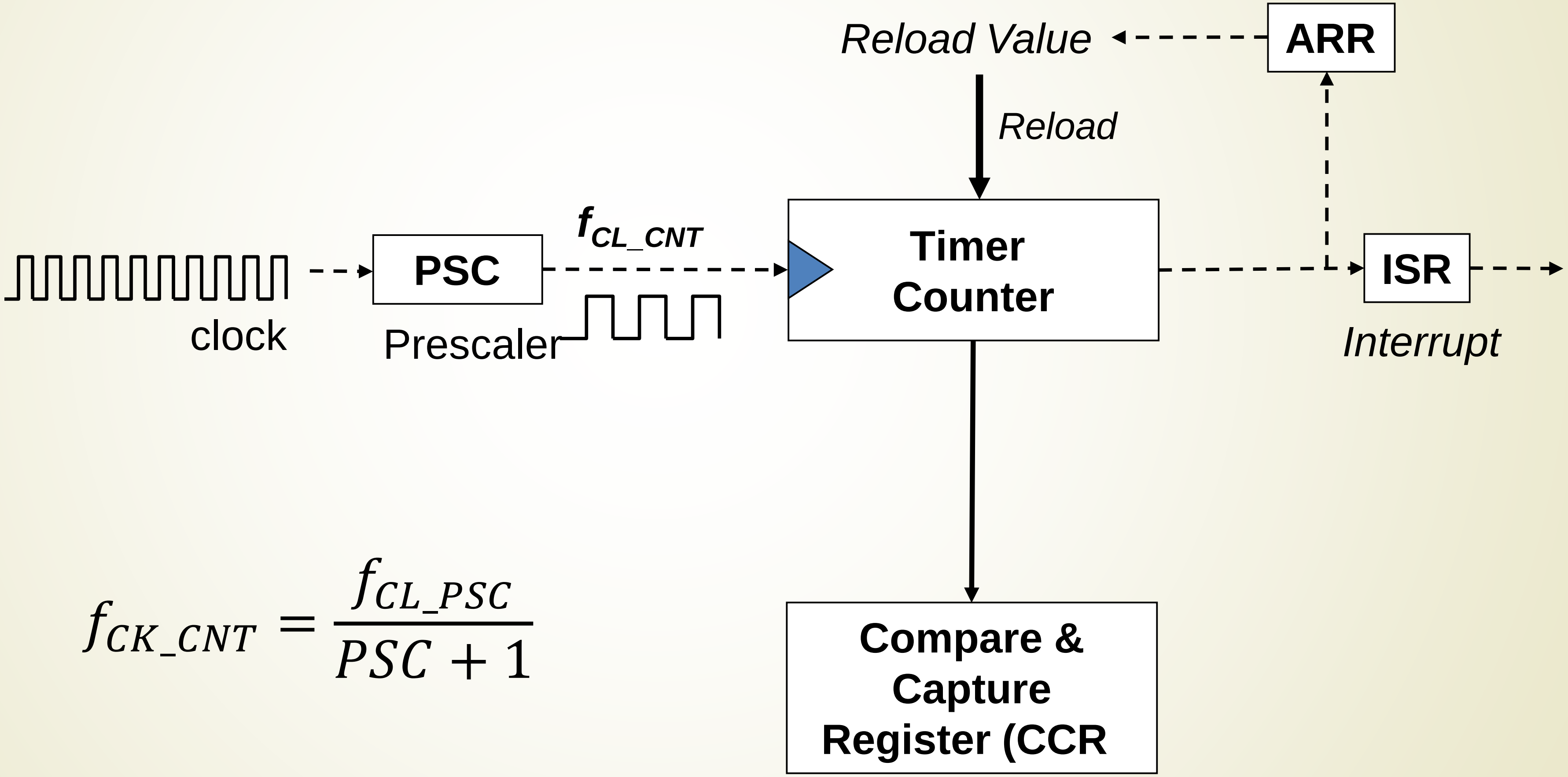
# Low Power Timer

- Lower mode timers
  - Operates in low power mode
  - Have independent clock
  - Run in Stop mode
  - Ability to wakeup the system from Stop mode
  - 16-bit up counter with 16-bit autoreload register
  - 16-bit compare register
  - Configurable output: pulse, PWM
  - Continuous/ one shot mode
  - Selectable clock source
    - Internal clock sources: LSE, LSI, HSI16 or APB clock
    - External clock source over LPTIM input (working eve

# Timer Usage

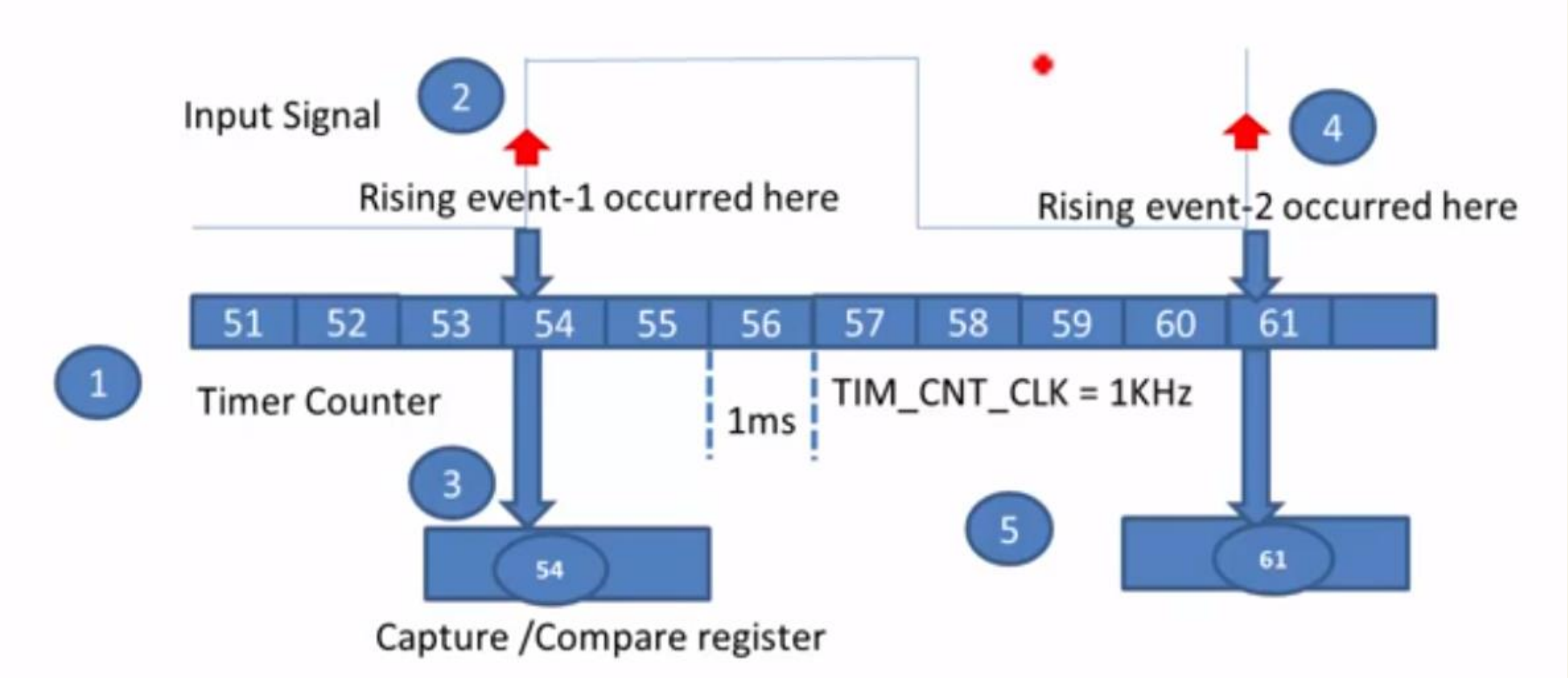
- **Creating delays**
- **Counting events**
- Measuring time between two events
- Measuring the pulse lengths of input signals
- Compare inputs
-  Generating output waveforms
- Multiple timers are used in large systems

# Input Capture

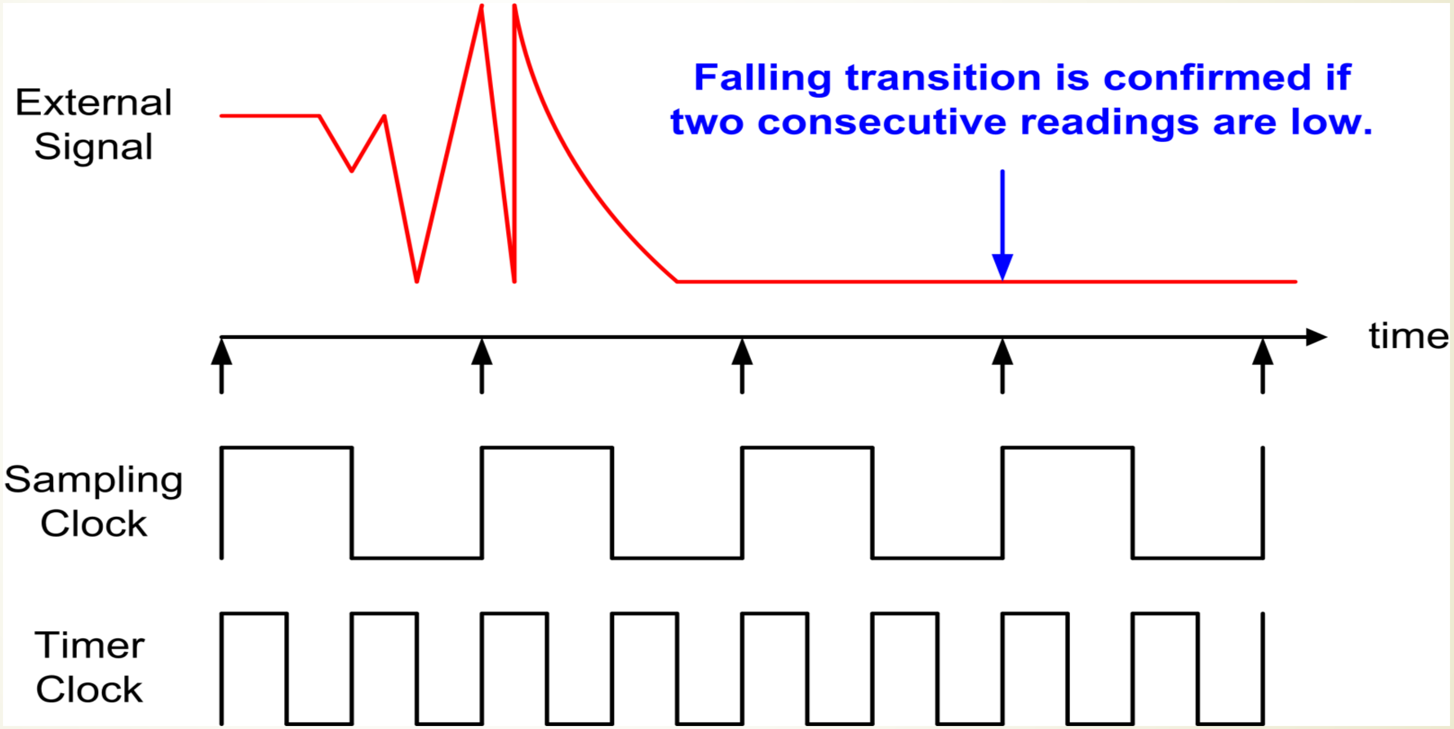


$$f_{CK\_CNT} = \frac{f_{CL\_PSC}}{PSC + 1}$$

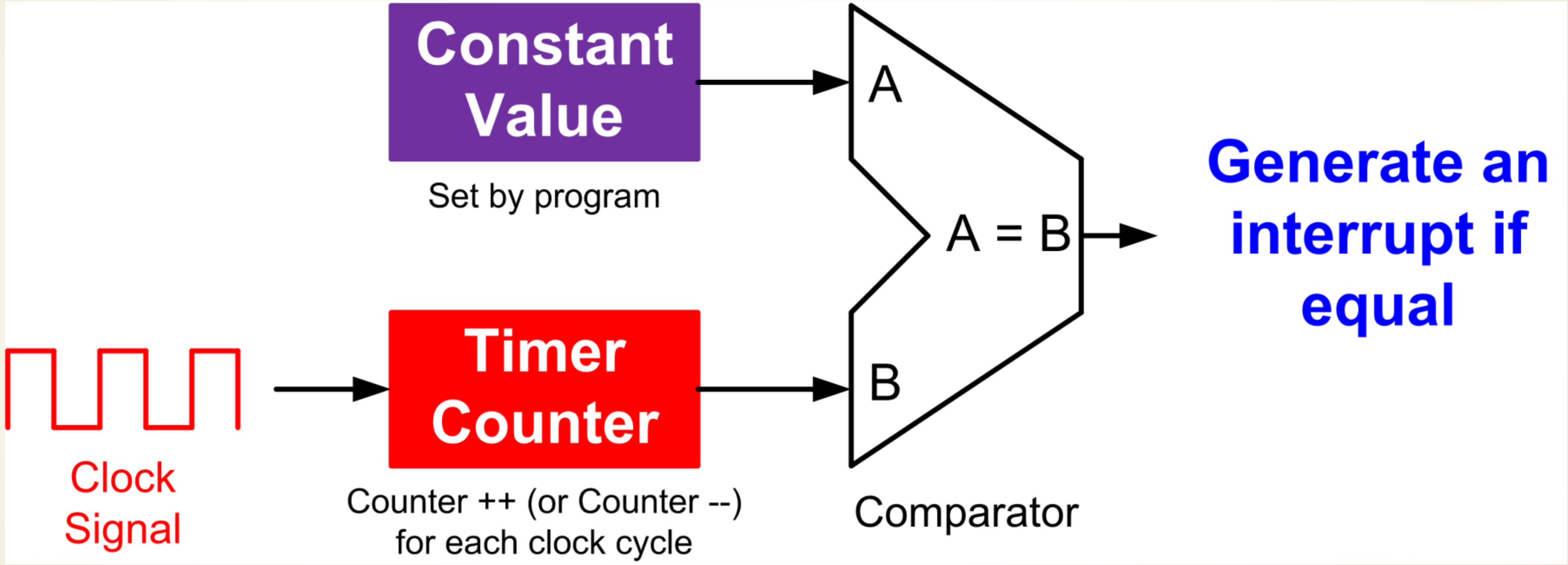
# Pulse width measurement


Input Signal
2
Rising event-1 occurred here
4
Rising event-2 occurred here
51
52
53
54
55
56
57
58
59
60
61
1
Timer Counter
1ms
TIM_CNT_CLK = 1KHz
3
54
5
61
Capture /Compare register

# Input Filtering


External Signal
Falling transition is confirmed if two consecutive readings are low.
time
Sampling Clock
Timer Clock

# Output Compare


Constant Value
Set by program
Clock Signal
Timer Counter
Counter ++ (or Counter --) for each clock cycle
A
B
A = B
Comparator
Generate an interrupt if equal

# Timer counting example

- Let's generate an interrupt every 1/10 sec,
- $f_{CK\_CNT} = \frac{f_{CL\_PSC}}{PSC+1}$
- Period = (1 + ARR) / $f_{CK\_CNT}$
- CL_PSC = 80 Mhz .
- What will be the value of ARR and PSC ?

Let's say PSC (prescalar) = 7999

$f_{CK\_CNT}$ = 80M/8K = 10K

1+ARR = 10k * 1/10 = 1000
ARR = 999

Note: You could generate any interval interrupt by programming ARR and PSC

# PWM

- Pulse width Modulation is digital technique to control the value of an analog variable
- Has many applications
  - DC-DC power conversion
  - Audio amplifications
  - Motor Speed
- Uses rectangular waveform to switch voltage source
- Average output value is proportional to duty cycle
- There are two modes
- Output signal is determined by three factors
  - Comparison between timer counter and CCR
  - PWM output mode
  - The polarity Bit

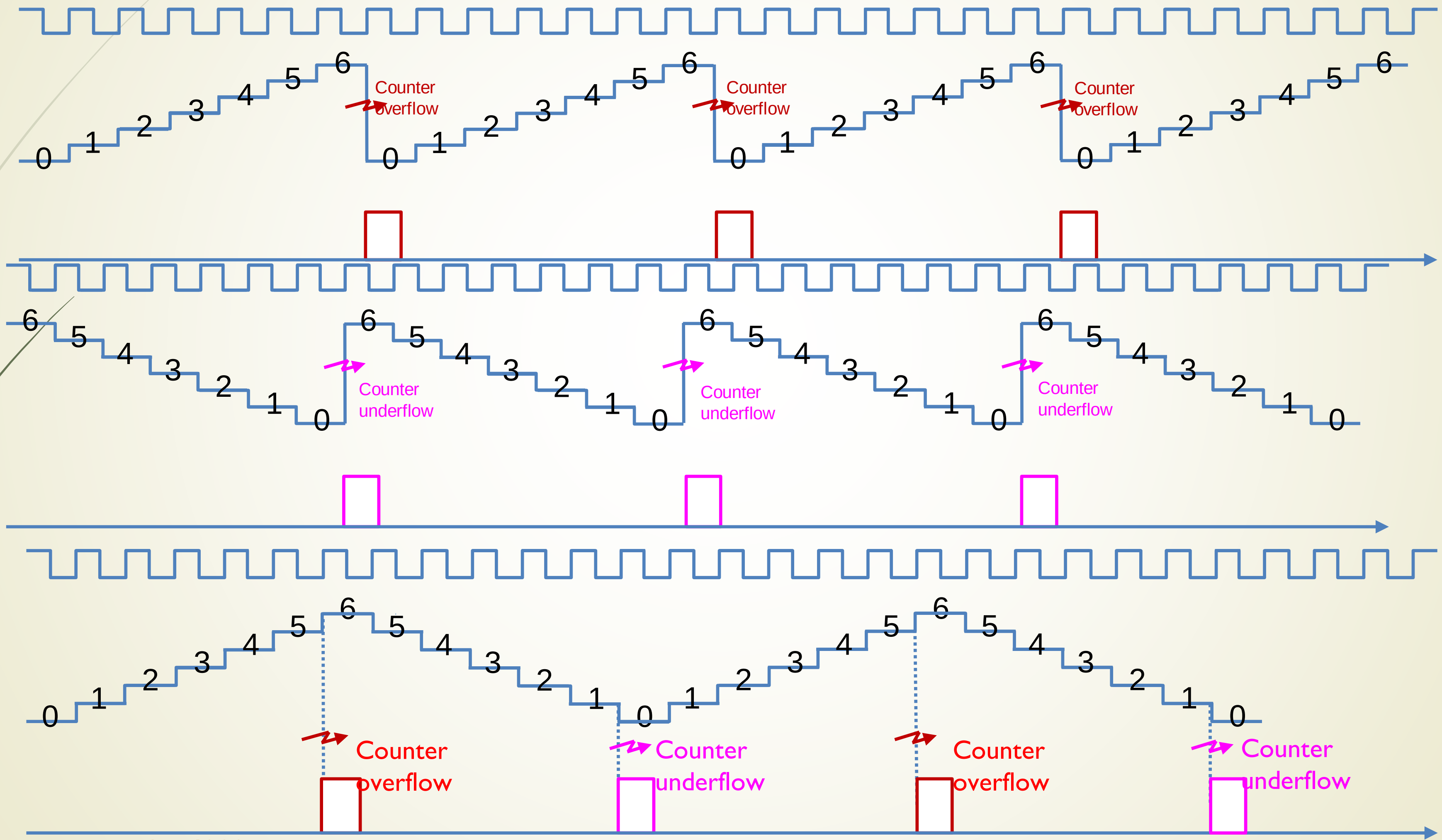
# PWM

- Output signal is determined by three factors
  - Comparison between timer counter and CCR
  - PWM output mode
  - The polarity Bit
- Three counting mode
  - Up-counting
  - Down-counting
  - Center-counting

# Timer

- Three counting mode
  - Up-counting
  - Down-counting
  - Center-counting
- Two update events
  - Overflow
    - When counter is reset o zero in up counting
  - Underflow
    - In down-counting when counter is reset to ARR

# PWM Mode


0
1
2
3
4
5
6
Counter overflow
6
5
4
3
2
1
0
Counter underflow
Counter overflow
Counter underflow
Counter overflow
Counter underflow

# Code Structure

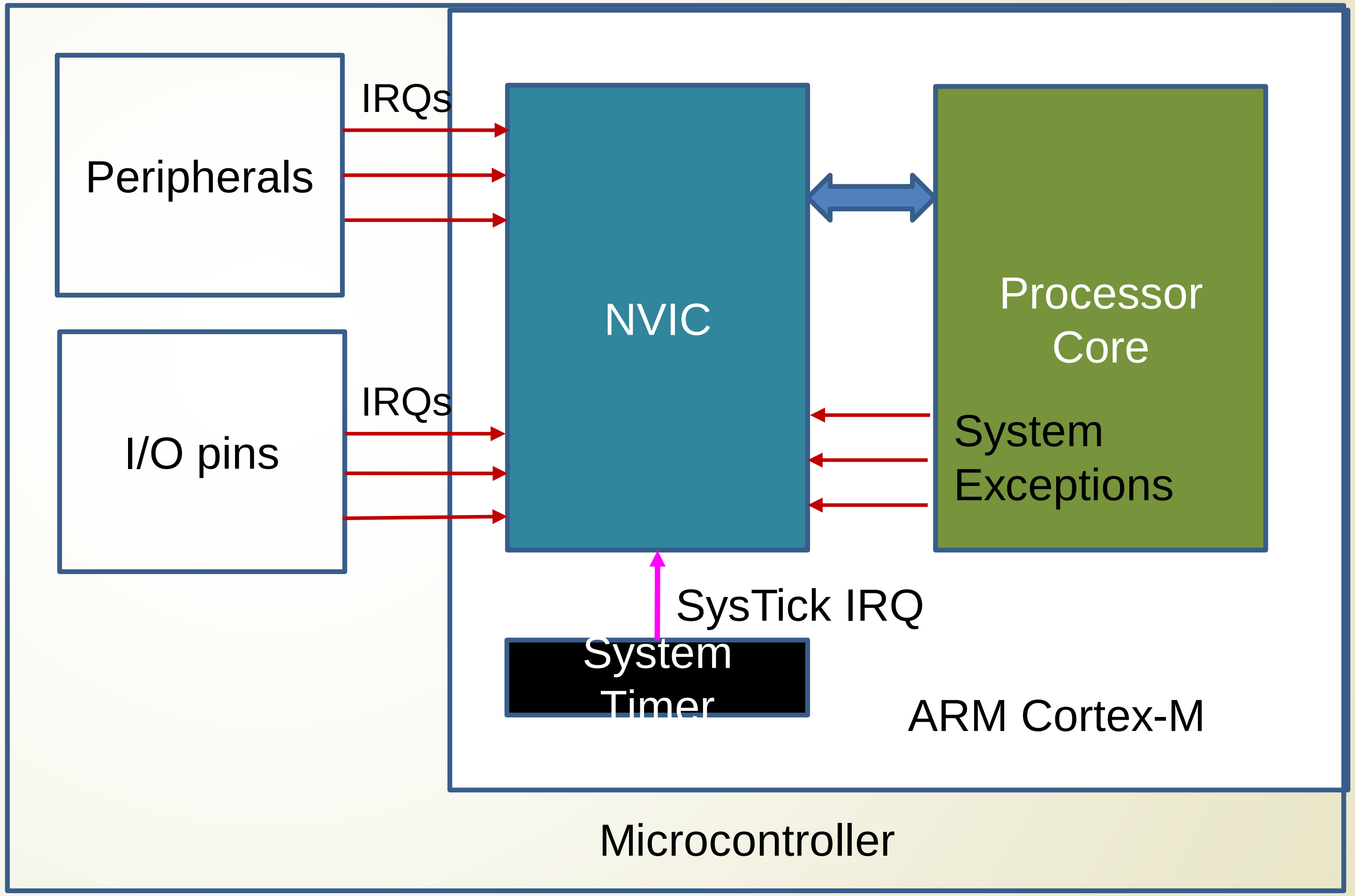
```
*
typedef struct/**
  * @brief  TIM Time base Configuration Structure definition
  */
typedef struct
{
  uint32_t Prescaler;         /*!< Specifies the prescaler value used to divide the TIM clock.
                                   This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF */

  uint32_t CounterMode;       /*!< Specifies the counter mode.
                                   This parameter can be a value of @ref TIM_Counter_Mode */

  uint32_t Period;            /*!< Specifies the period value to be loaded into the active
                                   Auto-Reload Register at the next update event.
                                   This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.  */

  uint32_t ClockDivision;     /*!< Specifies the clock division.
                                   This parameter can be a value of @ref TIM_ClockDivision */

  uint32_t RepetitionCounter;  /*!< Specifies the repetition counter value. Each time the RCR downcounter
                                    reaches zero, an update event is generated and counting restarts
                                    from the RCR value (N).
                                    This means in PWM mode that (N+1) corresponds to:
                                        - the number of PWM periods in edge-aligned mode
                                        - the number of half PWM period in center-aligned mode
                                     GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data =
0xFF.
                                     Advanced timers: this parameter must be a number between Min_Data = 0x0000 and
Max_Data = 0xFFFF. */

  uint32_t AutoReloadPreload;  /*!< Specifies the auto-reload preload.
                                   This parameter can be a value of @ref TIM_AutoReloadPreload */
} TIM_Base_InitTypeDef;
```

# System Timer

- Known as SysTick
- Is 24 bit count down timer
  - Used  for polling
  - Measure time elapsed, time delay function
  - RTOS scheduling
  - Simple counter
- Counts down from reload value to zero and then reloads
- No count down when processor is halted
- Generates SysTick interrupts at fixed interval

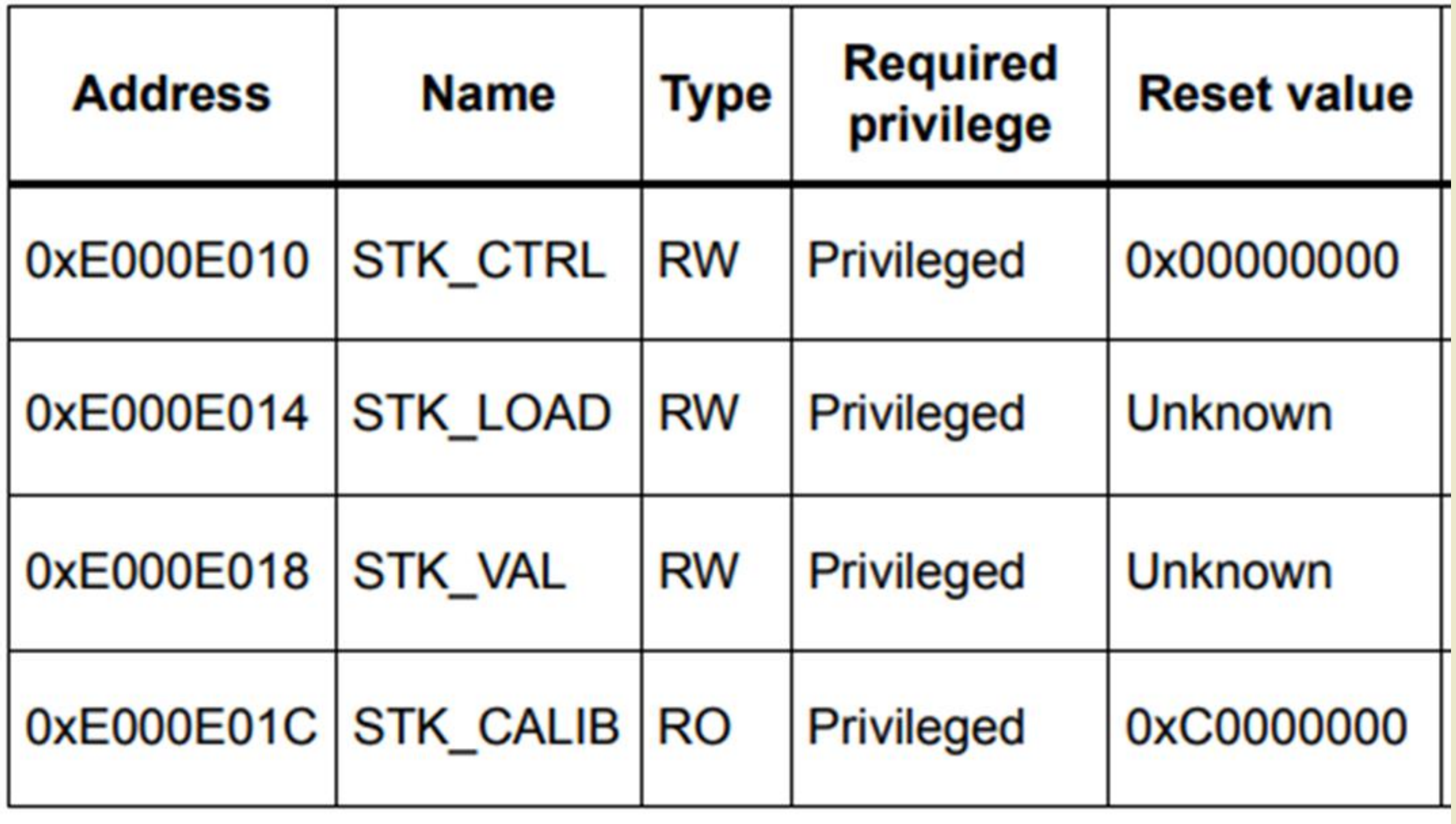
# Enable Interrupt

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



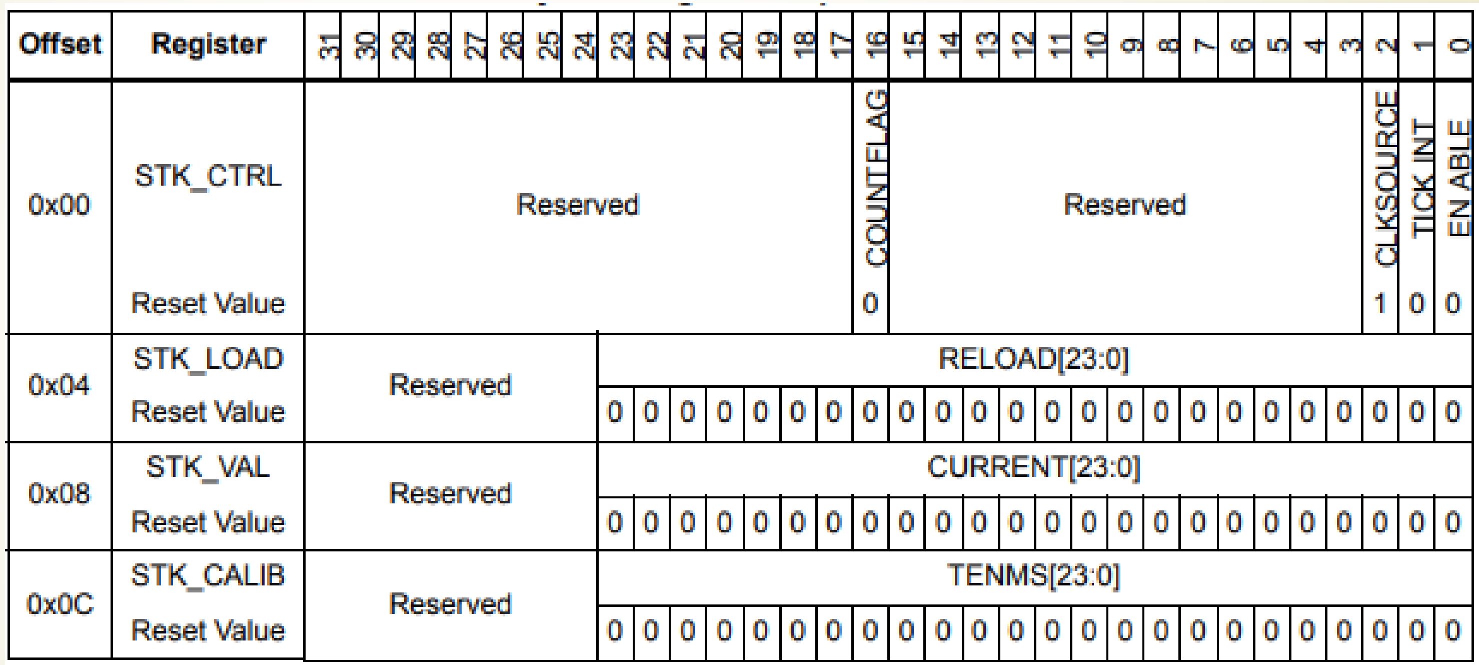
# Systick Registers

- Control and Status Register
- Reload value Register
- Current value Register
- Calibration value Register



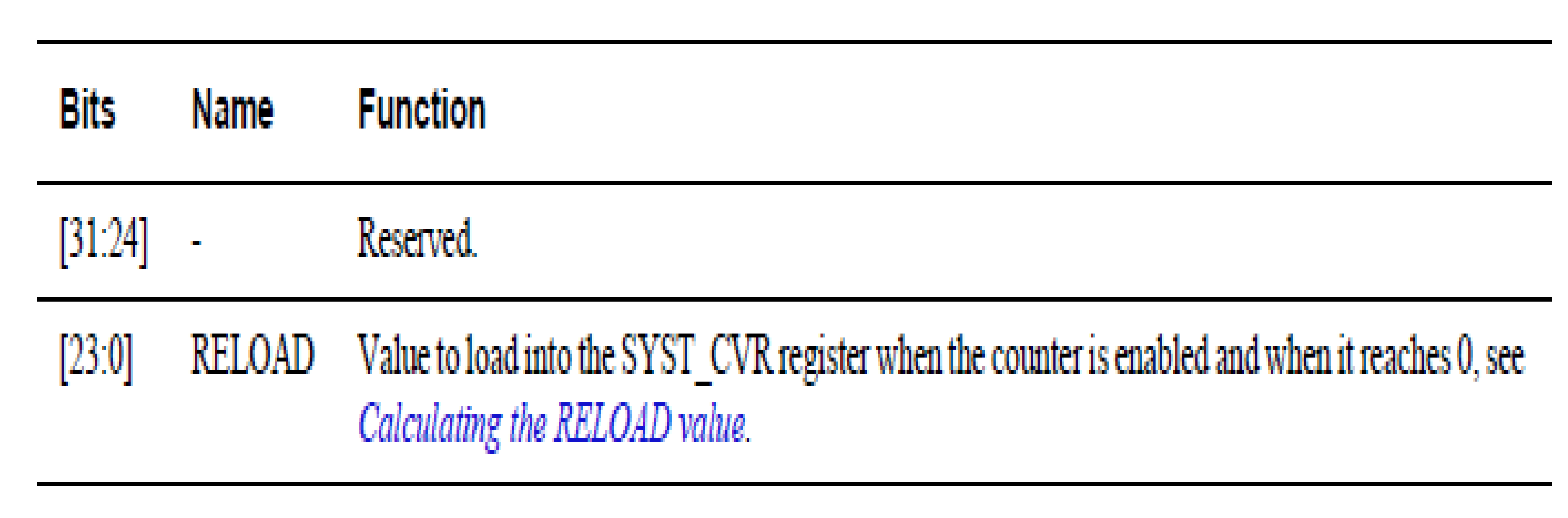
| Address | Name | Type | Required privilege | Reset value |
|---|---|---|---|---|
| 0xE000E010 | STK_CTRL | RW | Privileged | 0x00000000 |
| 0xE000E014 | STK_LOAD | RW | Privileged | Unknown |
| 0xE000E018 | STK_VAL | RW | Privileged | Unknown |
| 0xE000E01C | STK_CALIB | RO | Privileged | 0xC0000000 |

# Register Map



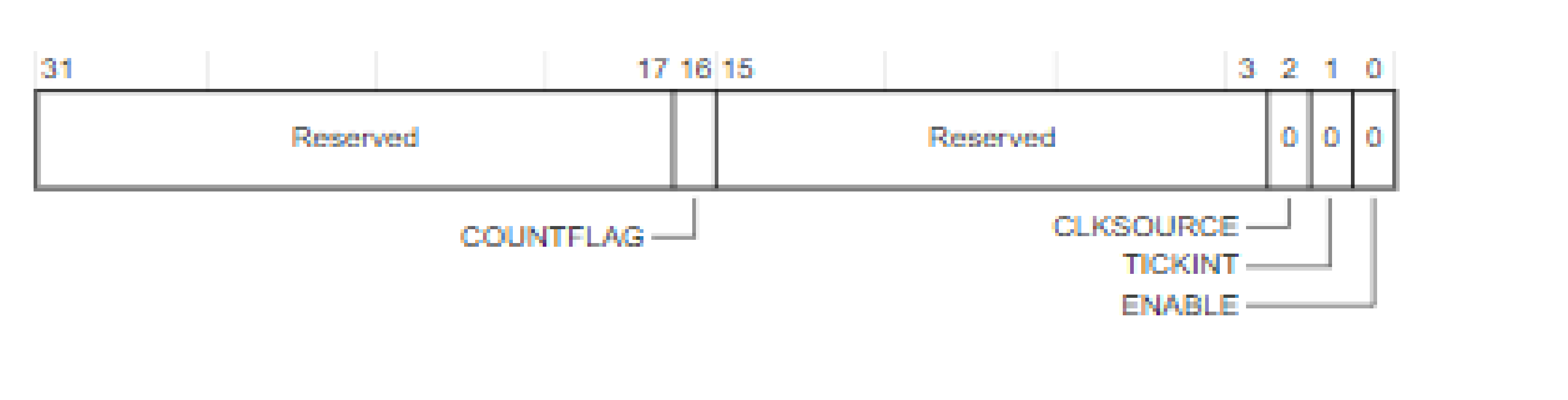
| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | STK_CTRL | Reserved | | | | | | | | | | | | | | | COUNTFLAG | Reserved | | | | | | | | | | | | | CLKSOURCE | TICK INT | EN ABLE |
| | Reset Value | | | | | | | | | | | | | | | | 0 | | | | | | | | | | | | | | 1 | 0 | 0 |
| 0x04 | STK_LOAD | Reserved | | | | | | | | RELOAD[23:0] | | | | | | | | | | | | | | | | | | | | | | | |
| | Reset Value | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x08 | STK_VAL | Reserved | | | | | | | | CURRENT[23:0] | | | | | | | | | | | | | | | | | | | | | | | |
| | Reset Value | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x0C | STK_CALIB | Reserved | | | | | | | | TENMS[23:0] | | | | | | | | | | | | | | | | | | | | | | | |
| | Reset Value | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Reload Register



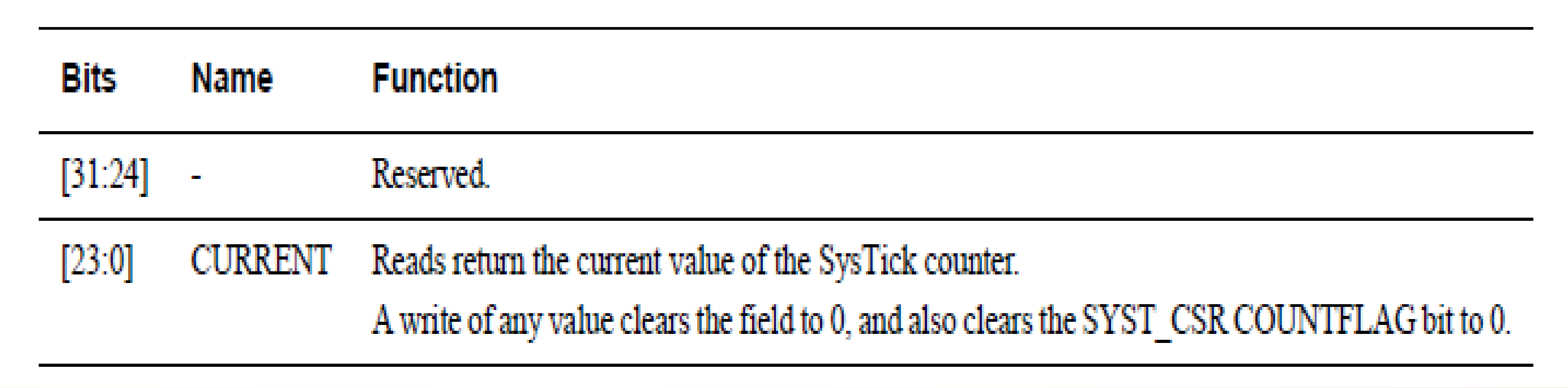
| Bits | Name | Function |
|---|---|---|
| [31:24] | - | Reserved. |
| [23:0] | RELOAD | Value to load into the SYST_CVR register when the counter is enabled and when it reaches 0, see *Calculating the RELOAD value.* |

# Control Register



| | | |
|---|---|---|
| [0] | ENABLE | Enables the counter:<br>0 = counter disabled<br>1 = counter enabled. |
| [2] | CLKSOURCE | Indicates the clock source:<br>0 = external clock<br>1 - processor clock. |
| [16] | COUNTFLAG | Returns 1 if timer counted to 0 since last time this was read. |

# Current value Register



| Bits | Name | Function |
|---|---|---|
| [31:24] | - | Reserved. |
| [23:0] | CURRENT | Reads return the current value of the SysTick counter.<br>A write of any value clears the field to 0, and also clears the SYST_CSR COUNTFLAG bit to 0. |

# Code Structure

```
/**
  \brief  Structure type to access the System Timer (SysTick).
 */
typedef struct
{
  __IOM uint32_t CTRL;                    /*!< Offset: 0x000 (R/W)  SysTick Control and
Status Register */
  __IOM uint32_t LOAD;                    /*!< Offset: 0x004 (R/W)  SysTick Reload
Value Register */
  __IOM uint32_t VAL;                    /*!< Offset: 0x008 (R/W)  SysTick Current Value
Register */
  __IM  uint32_t CALIB;                  /*!< Offset: 0x00C (R/ )  SysTick Calibration
Register */
} SysTick_Type;
```

# Usage

- To calculate Delay
  - Program the control register to enable the ounter
  - Put the delay value in value register
  - Wait in loop and keep checking the COUNTGLAG bit in control register
- Reset the control register

# Design hints

- Used for delay
- Runs on processor clock
- Counter stops, if clock signal is stopped
- Correct initialization sequences
  - Program reload value
  - Enable the timer
  - Wait for the COUNTFLAG

# Reload value

- Suppose clock source = 80 Mhz
- Needed SysTick interval = 10 ms

$$\textbf{Reload} = \frac{\textbf{Interval}}{\textbf{Clock}\ \boldsymbol{Period}} - \mathbf{1}$$

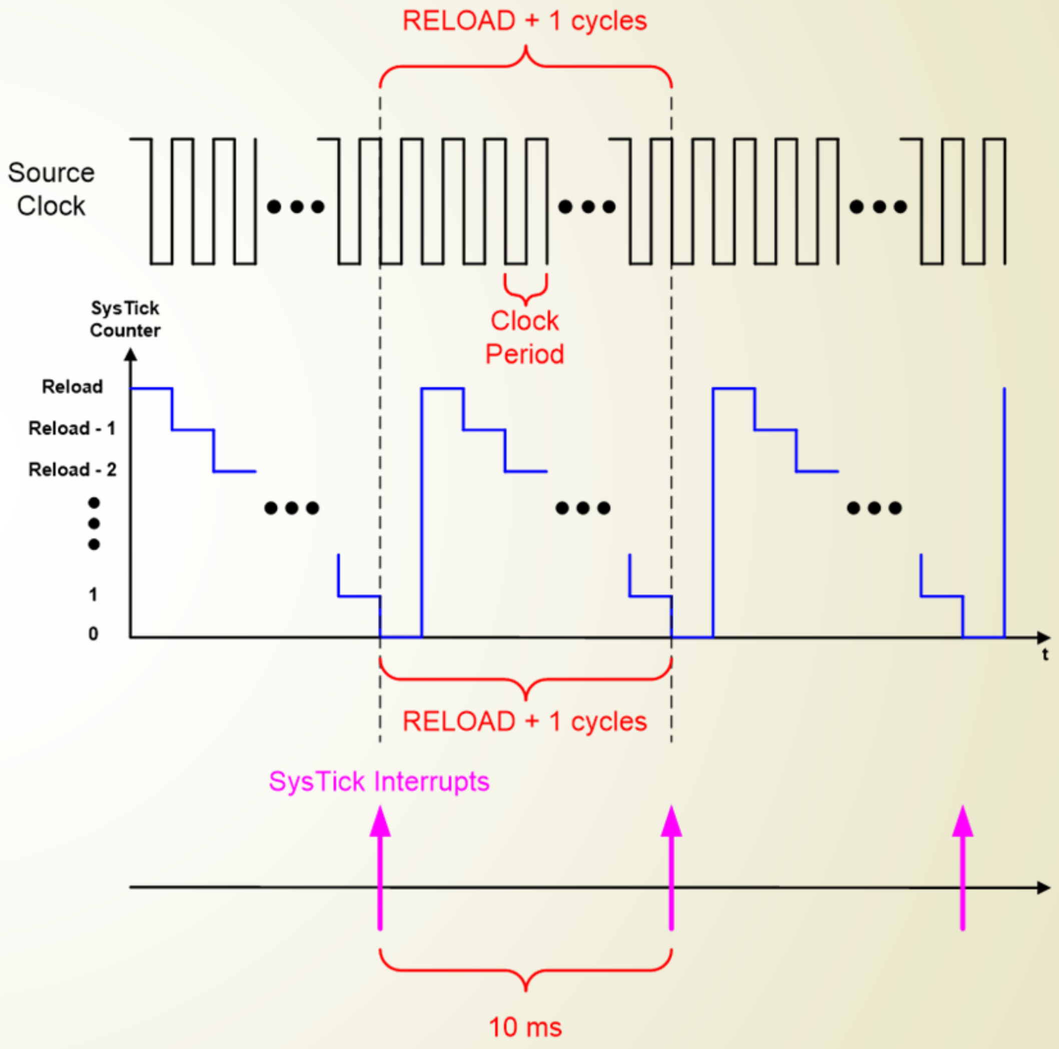
$$= 10\text{ms} \times \text{Clock Frequency } - 1$$

$$= 10\text{ms} \times 80\text{MHz} - 1$$

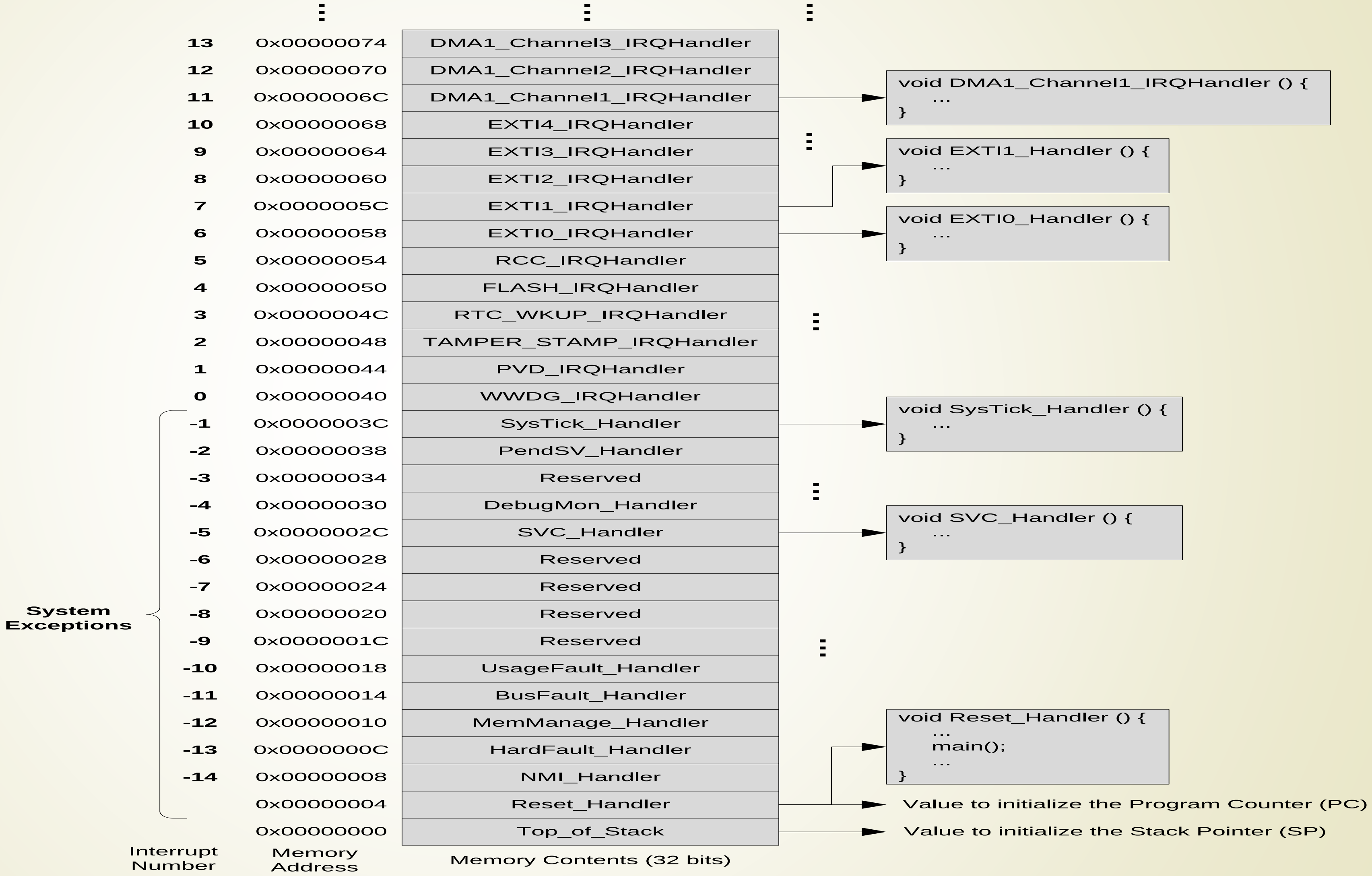
$$= 10 \times 10^{-3} \times 80 \times 10^{6} - 1$$

$$= 800000 - 1$$

$$= 799999$$



# Vector Table



| Interrupt Number | Memory Address | Memory Contents (32 bits) |
|---|---|---|
| ⋮ | ⋮ | ⋮ |
| 13 | 0x00000074 | DMA1_Channel3_IRQHandler |
| 12 | 0x00000070 | DMA1_Channel2_IRQHandler |
| 11 | 0x0000006C | DMA1_Channel1_IRQHandler |
| 10 | 0x00000068 | EXTI4_IRQHandler |
| 9 | 0x00000064 | EXTI3_IRQHandler |
| 8 | 0x00000060 | EXTI2_IRQHandler |
| 7 | 0x0000005C | EXTI1_IRQHandler |
| 6 | 0x00000058 | EXTI0_IRQHandler |
| 5 | 0x00000054 | RCC_IRQHandler |
| 4 | 0x00000050 | FLASH_IRQHandler |
| 3 | 0x0000004C | RTC_WKUP_IRQHandler |
| 2 | 0x00000048 | TAMPER_STAMP_IRQHandler |
| 1 | 0x00000044 | PVD_IRQHandler |
| 0 | 0x00000040 | WWDG_IRQHandler |
| -1 | 0x0000003C | SysTick_Handler |
| -2 | 0x00000038 | PendSV_Handler |
| -3 | 0x00000034 | Reserved |
| -4 | 0x00000030 | DebugMon_Handler |
| -5 | 0x0000002C | SVC_Handler |
| -6 | 0x00000028 | Reserved |
| -7 | 0x00000024 | Reserved |
| -8 | 0x00000020 | Reserved |
| -9 | 0x0000001C | Reserved |
| -10 | 0x00000018 | UsageFault_Handler |
| -11 | 0x00000014 | BusFault_Handler |
| -12 | 0x00000010 | MemManage_Handler |
| -13 | 0x0000000C | HardFault_Handler |
| -14 | 0x00000008 | NMI_Handler |
|  | 0x00000004 | Reset_Handler |
|  | 0x00000000 | Top_of_Stack |

```
void DMA1_Channel1_IRQHandler () {
    ...
}
```

```
void EXTI1_Handler () {
    ...
}
```

```
void EXTI0_Handler () {
    ...
}
```

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

```
void SVC_Handler () {
    ...
}
```

```
void Reset_Handler () {
    ...
    main();
    ...
}
```

Value to initialize the Program Counter (PC)

Value to initialize the Stack Pointer (SP)

# Using the Interrupt

```
volatile int32_t TimeDelay;

int main (void {
HAL_Delay(50);   // Using HAL
MyDelay(50);                  // using myDelay
  ...
}

// SysTick interrupt service routine

void MyDelay (uint32_t nTime) {
  // Program the register

SysTick->LOAD    ; Load the correct   value
SysTick->VAL
SysTick->CTRL  // Program Control register to select clock and enable
timer
}
```

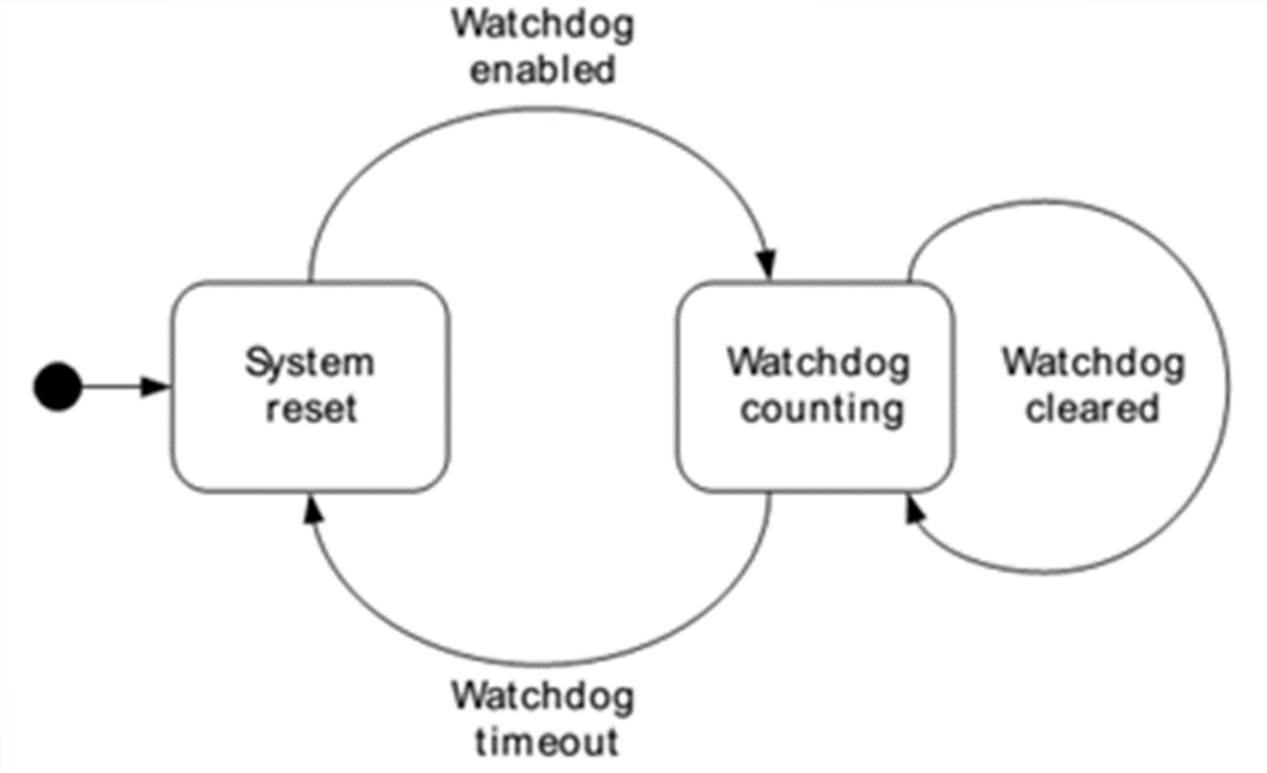
# Watchdog Timers

- As the name suggests, it is to watch the system
- Detect and recover from Software/Hardware malfunctions
- Is free running timer which gets reset/refresh/punch during normal operations
    - Action varies based on design and products
        - Do cold/hard reboot of the board
        - Try 2-3 times before rebooting it
        - Detect the stuck tasks/threads and try to restart it
        - Visual indication (only applicable with human intervention)
- Must have in embedded systems for healthy operation/recovery of systems
- What happens when the thread monitoring watchdog dies ??

# Watchdog Timers

- Programmable free running down counter
- Conditional reset
  - Resets when value is below the threshold
  - Reset if value is loaded from outside
- Early wake up interrupt
  - Used for early detection
  - Could be used for safety operations
  - Data logging
  - Protecting file system
  - Or continue with degraded performance (load the value in ISR)

# Use of Watchdog



# Types of Watchdog

- Independent watchdog
- Window watchdog

# Independent Watchdog

- The independent watchdog is based on a 12-bit down counter and 8-bit prescaler.
- It is clocked from an independent 32 kHz internal RC (LSI) and as it operates independently from the main clock
- It can operate in Stop and Standby modes.
- It can be used either as a watchdog to reset the device when a problem occurs, or as a free running timer for application timeout management.

# Independent Watchdog

- 7 prescalers – 4 to 256
- Programmable Timeout Range is from 125 micro sec to 32.8 sec
- Generates reset when:
  - Timeout value is reached
  - Refresh occurs outside the window
- Could be configured to enable automatically after system reset

# Independent Watchdog

- Equation to calculate the time out value

TimeOut in seconds = (Reload * Prescaler) / Freq.

Min Prescaler = 4, clock = 32000

Min Reload = 1

Min Timeout value = (4*1)/32000 = **125 microsec.**

- Could you calculate max value – 32.768 sec ?

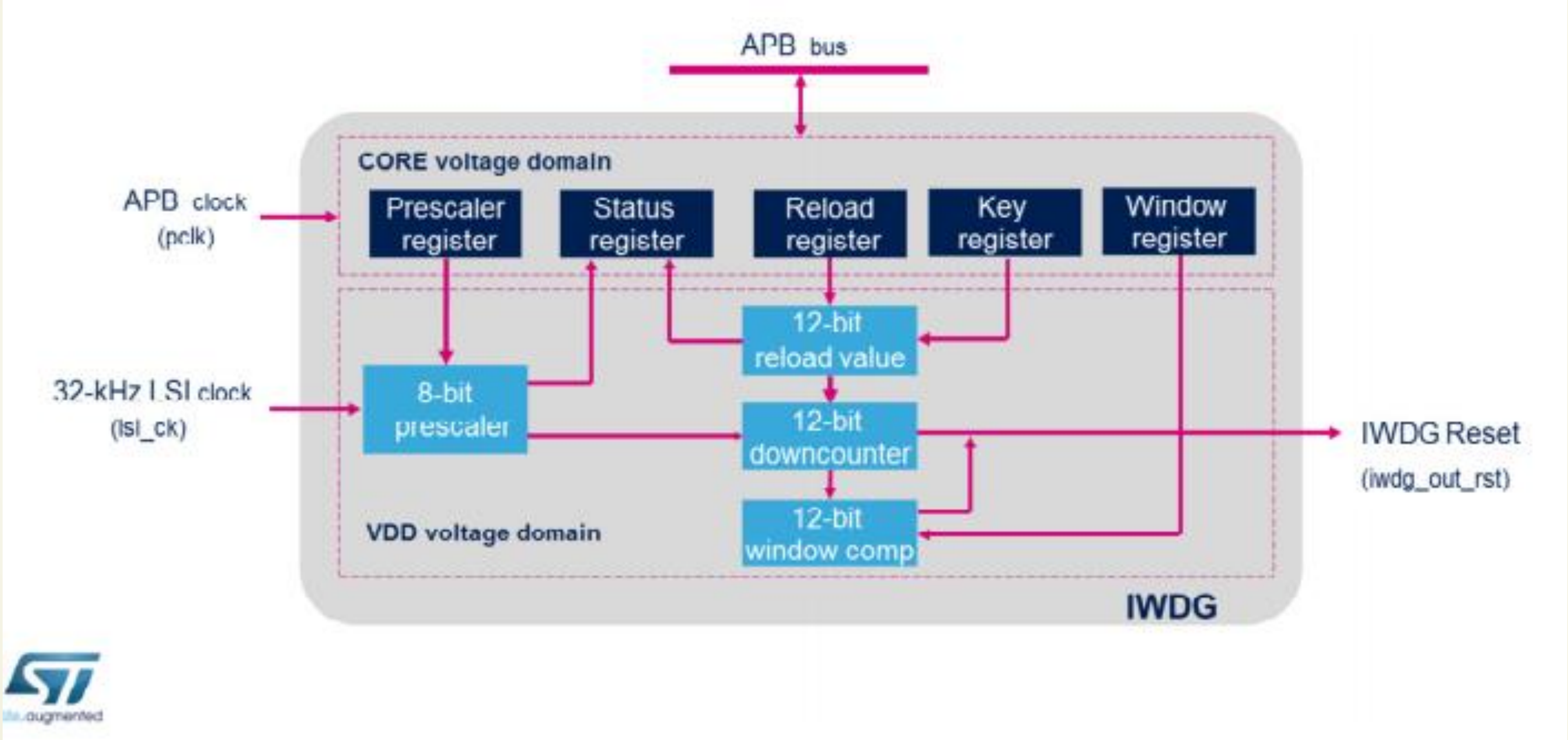
# Registers

```
**
  * @brief  IWDG Init structure definition
  */
typedef struct
{
  uint32_t Prescaler;  /*!< Select the prescaler of the IWDG.
                            This parameter can be a value of @ref IWDG_Prescaler */

  uint32_t Reload;     /*!< Specifies the IWDG down-counter reload value.
                            This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF */

  uint32_t Window;     /*!< Specifies the window value to be compared to the down-counter.
                            This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF */

} IWDG_InitTypeDef;
```

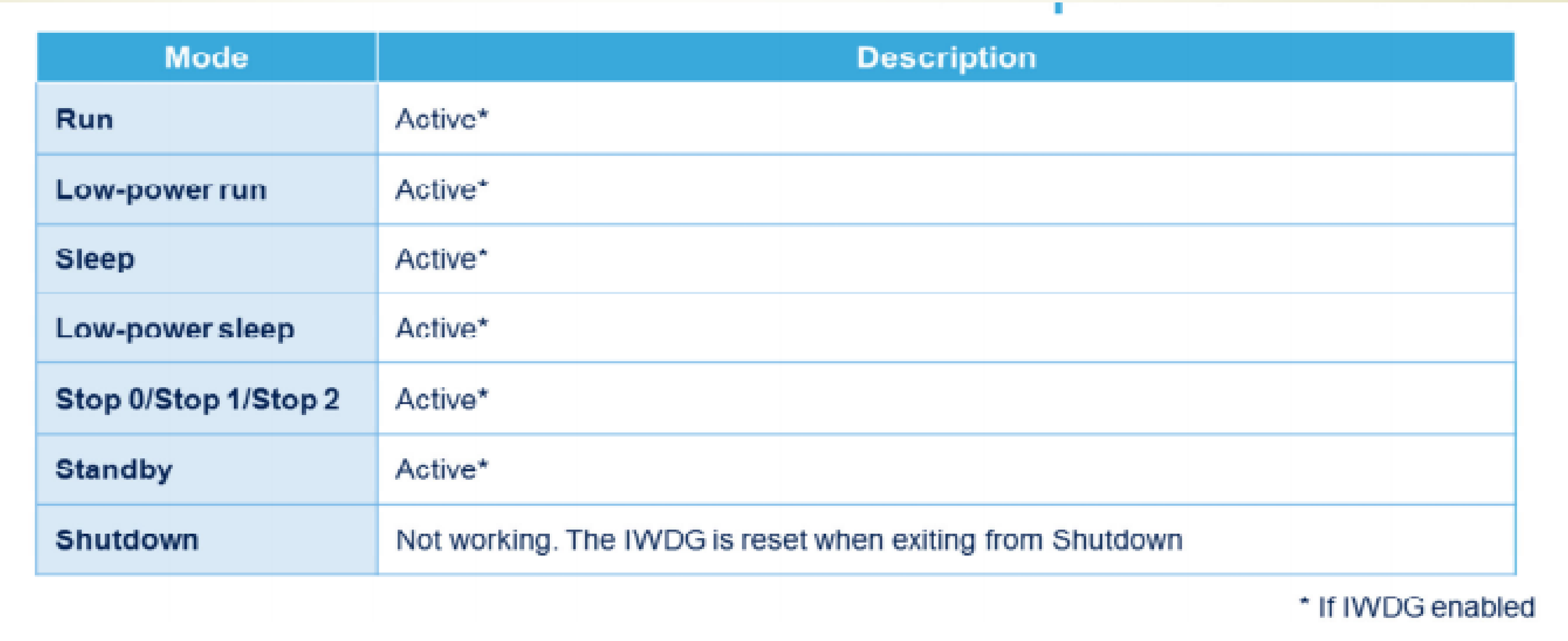
# Independent Watchdog

```
/** @defgroup IWDG_Prescaler IWDG Prescaler
  * @{
  */
#define IWDG_PRESCALER_4                        0x00000000u
/*!< IWDG prescaler set to 4   */
#define IWDG_PRESCALER_8                        IWDG_PR_PR_0
/*!< IWDG prescaler set to 8   */
#define IWDG_PRESCALER_16                       IWDG_PR_PR_1
/*!< IWDG prescaler set to 16  */
#define IWDG_PRESCALER_32                       (IWDG_PR_PR_1 | IWDG_PR_PR_0)
/*!< IWDG prescaler set to 32  */
#define IWDG_PRESCALER_64                       IWDG_PR_PR_2
/*!< IWDG prescaler set to 64  */
#define IWDG_PRESCALER_128                      (IWDG_PR_PR_2 | IWDG_PR_PR_0)
/*!< IWDG prescaler set to 128 */
#define IWDG_PRESCALER_256                      (IWDG_PR_PR_2 | IWDG_PR_PR_1)
/*!< IWDG prescaler set to 256*/


/**
```

# Block Diagram



# Low Power Modes



| Mode | Description |
|---|---|
| **Run** | Active* |
| **Low-power run** | Active* |
| **Sleep** | Active* |
| **Low-power sleep** | Active* |
| **Stop 0/Stop 1/Stop 2** | Active* |
| **Standby** | Active* |
| **Shutdown** | Not working. The IWDG is reset when exiting from Shutdown |

* If IWDG enabled

# Window Watchdog

- The window watchdog is based on a 7-bit down counter that can be set as free running.
- It can be used as a watchdog to reset the device when a problem occurs.
- It is clocked from the main clock.
- It has an early warning interrupt capability before rest happens
- Best suited for application required to react within an accurate timing window

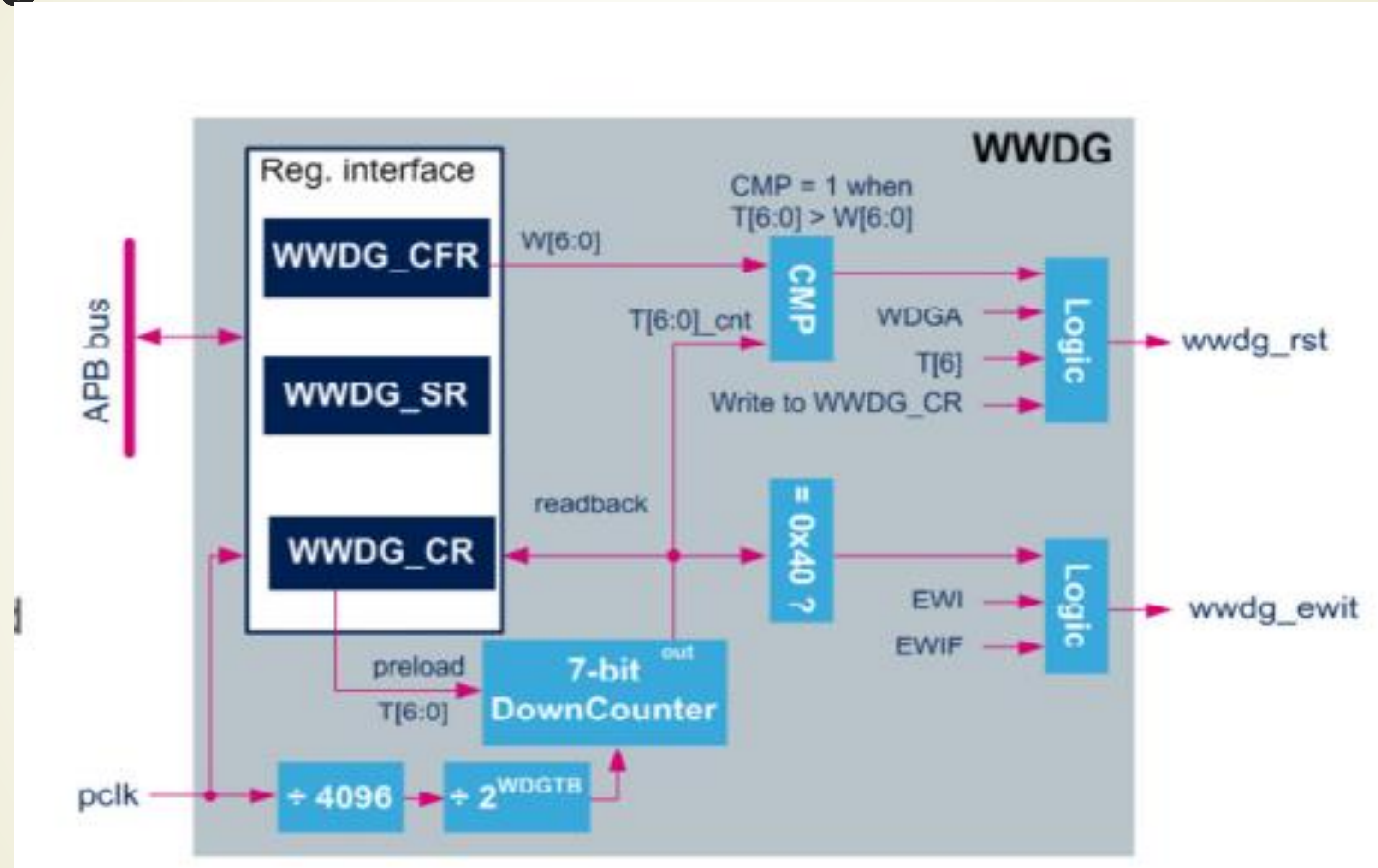
# Usage

- The user can program the timeout value and the window width according to application needs.
- It can generate a reset under two conditions:
  - when the downcounter value becomes less or equal to 0x3F, or
  - when the watchdog is refreshed outside the timewindow.
- It can generate an early wakeup interrupt when the downcounter reaches 0x40.
- The early wakeup interrupt can be used to reload the downcounter in order to avoid a reset generation, or to manage system recovery and context backup operations.

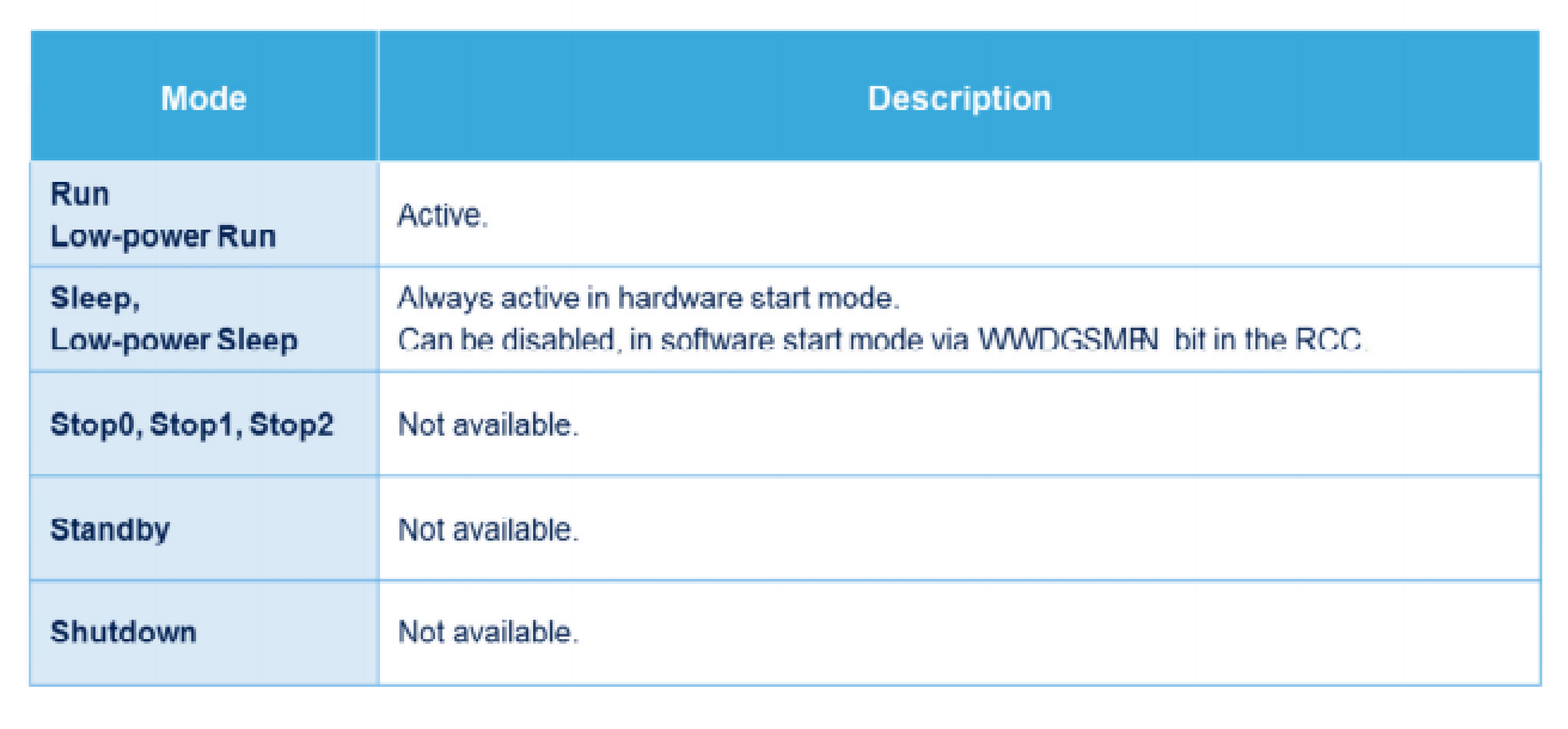
# Independent Watchdog

```
/**
  * @brief  WWDG Init structure definition
  */
typedef struct
{
  uint32_t Prescaler;     /*!< Specifies the prescaler value of the WWDG.
                               This parameter can be a value of @ref WWDG_Prescaler */

  uint32_t Window;        /*!< Specifies the WWDG window value to be compared to the downcounter.
                               This parameter must be a number Min_Data = 0x40 and Max_Data = 0x7F */

  uint32_t Counter;       /*!< Specifies the WWDG free-running downcounter  value.
                               This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F */

  uint32_t EWIMode ;      /*!< Specifies if WWDG Early Wakeup Interupt is enable or not.
                               This parameter can be a value of @ref WWDG_EWI_Mode */

} WWDG_InitTypeDef;
```

# Block Diagram



# Low Power Modes



| Mode | Description |
|---|---|
| **Run**<br>**Low-power Run** | Active. |
| **Sleep,**<br>**Low-power Sleep** | Always active in hardware start mode.<br>Can be disabled, in software start mode via WWDGSMEN bit in the RCC. |
| **Stop0, Stop1, Stop2** | Not available. |
| **Standby** | Not available. |
| **Shutdown** | Not available. |

# Next Lesson Topic

- RTC
- DMA
- Power modes