Peripherals of Freescale Kinetis microcontrolers

Part 1

Outline

- **General Purpose Input Output**
  - Basic Concepts
  - Port Circuitry
  - Control Registers
  - Accessing Hardware Registers in C
  - Clocking and Muxing
  - Port configuration
- **Timer/PWM Module**
  - Structure outline
  - Modes of operation
  - Control and status registers
  - Timer configuration
  - Channels configuration
  - Real life applications

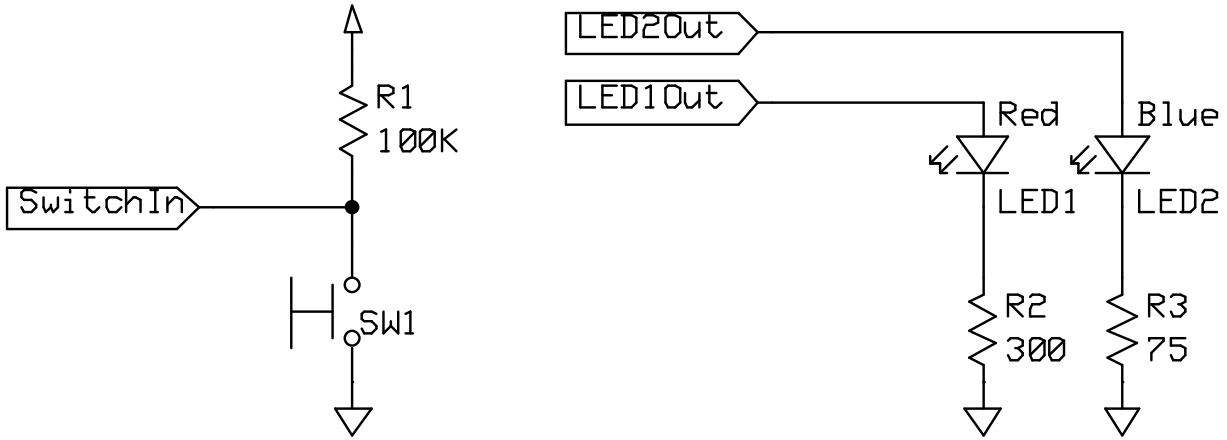
Literature

- ***KL46 Sub-Family Reference Manual*, Freescale Semiconductor**
- ***Kinetis L Peripheral Module Quick Reference*, Freescale Semiconductor**
- ***Mikrokontrolery Kinetis dla początkujących,* A. Gromczyński**
  - CodeWarrior examples !

General Purpose I/O

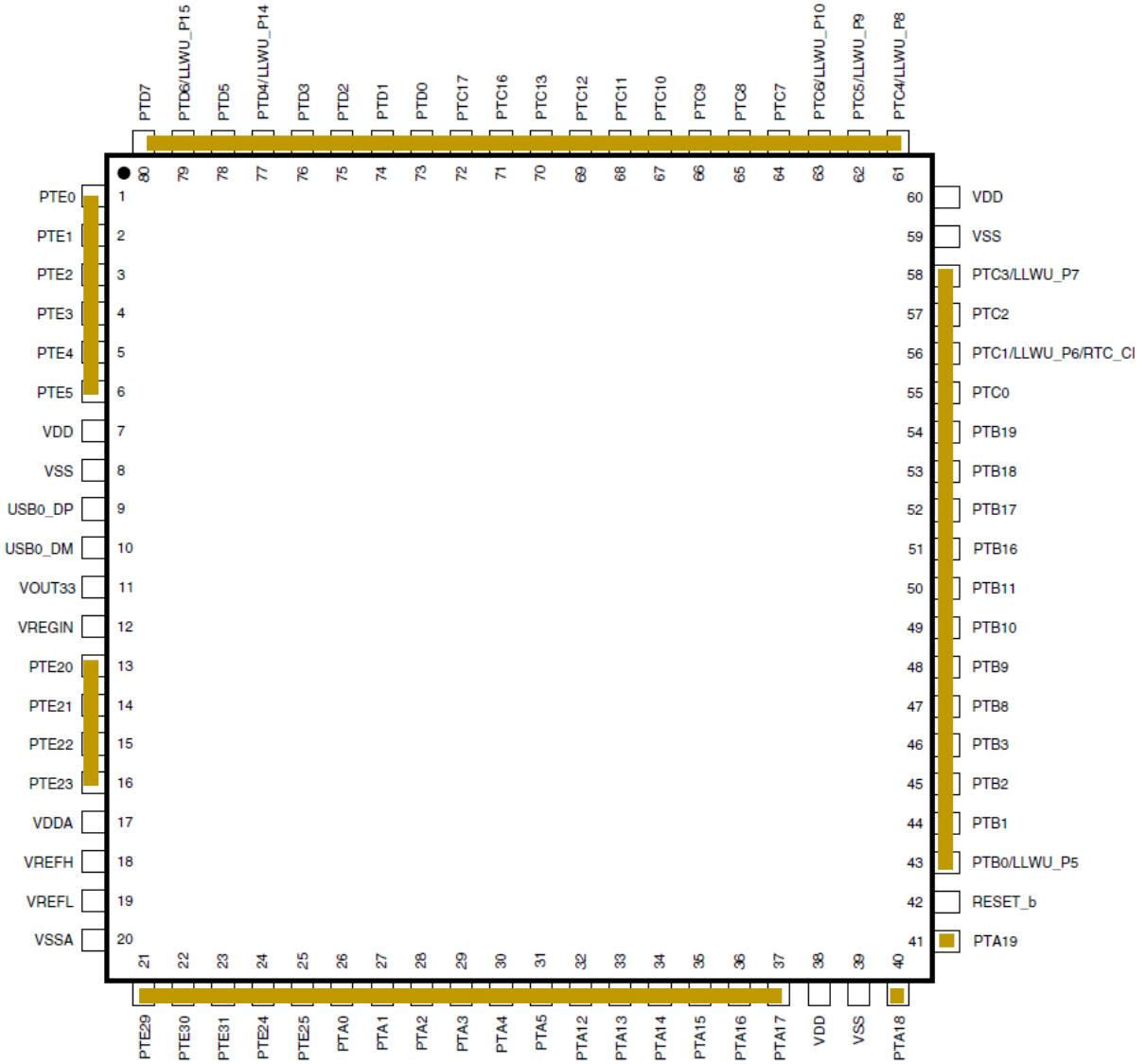
Basic Concepts

- **GPIO = General-purpose (Digital !) Input and Output**
  - Input: program can determine if input signal is a logic 1 or a 0
  - Output: program can set output to logic 1 or 0
- **Provide a basic, direct pin control mechanism**
- **Can use this to interface with external devices**
  - Input: switches, buttons
  - Output: LEDs
- **Example: light either LED1 or LED2 based on switch SW1 position**



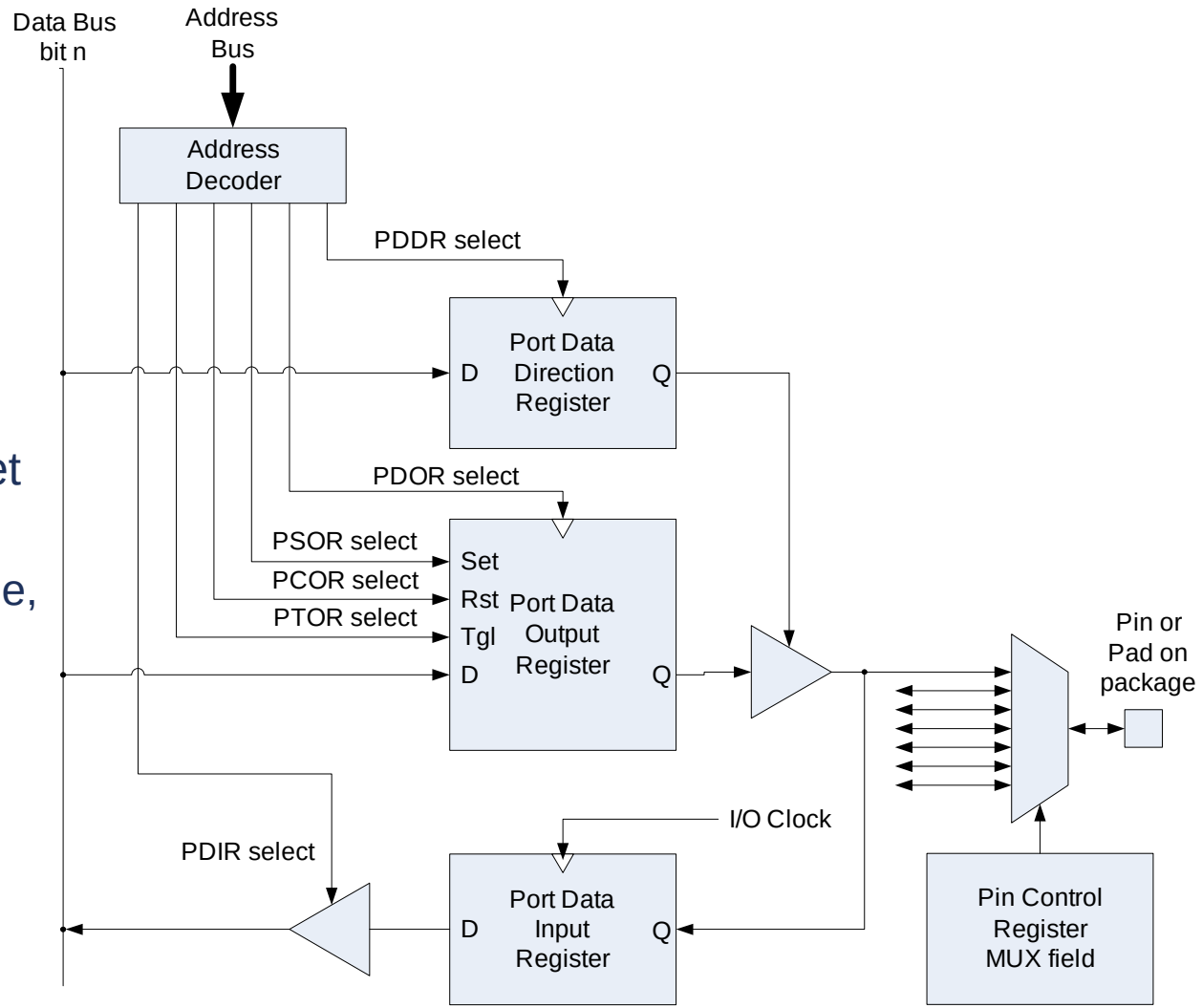
KL25Z GPIO Ports

- Internally 32-bits ports
- Port A (PTA) through Port E (PTE)
- Not all port bits are available
- Quantity depends on package pin count



GPIO Port Bit Circuitry in MCU

- **Configuration**
  - Direction
  - MUX
  - Pull resistor control
- **Data**
  - Different ways to set output data:
    - set, reset, toggle, data
  - Input



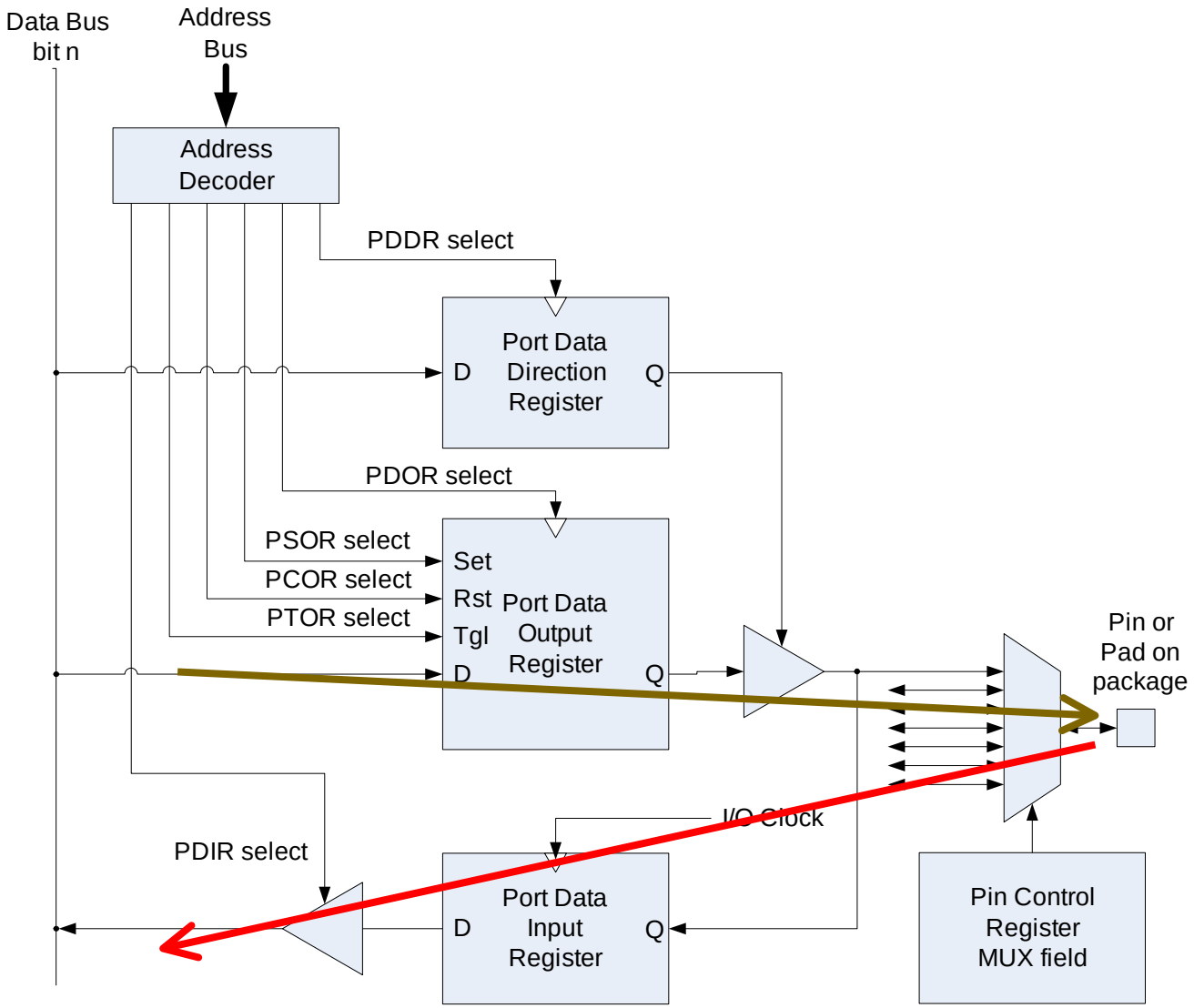
Control Registers

Absolute address (hex)	Register name	Width (in bits)
400F_F000	Port Data Output Register (GPIOA_PDOR)	32
400F_F004	Port Set Output Register (GPIOA_PSOR)	32
400F_F008	Port Clear Output Register (GPIOA_PCOR)	32
400F_F00C	Port Toggle Output Register (GPIOA_PTOR)	32
400F_F010	Port Data Input Register (GPIOA_PDIR)	32
400F_F014	Port Data Direction Register (GPIOA_PDDR)	32

- **One set of control registers per port**
- **Each bit in a control register corresponds to a port bit**

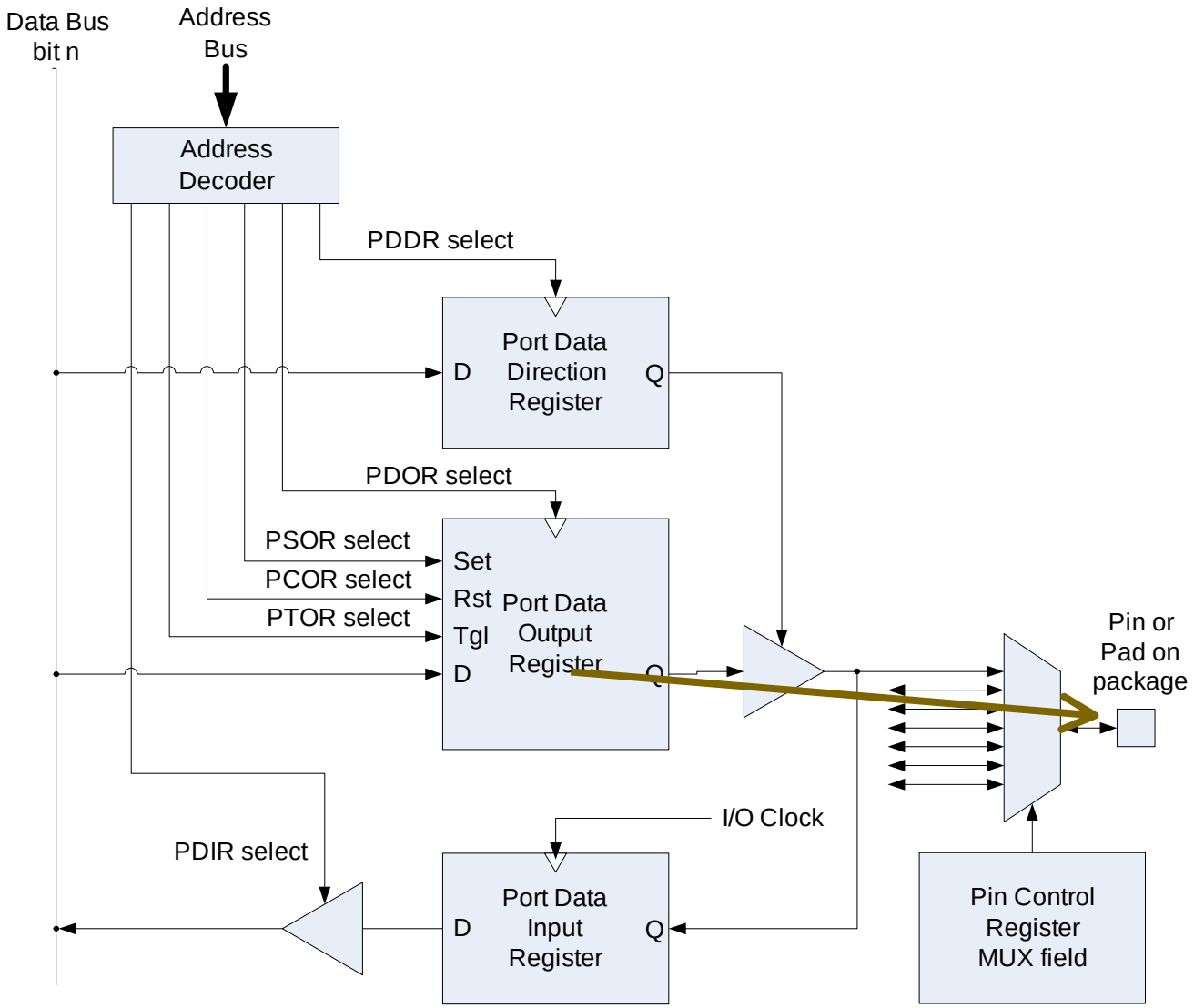
PDDR: Port Data Direction

- Each bit can be configured differently
- Input: 0
- Output: 1
- Reset clears port bit direction to 0



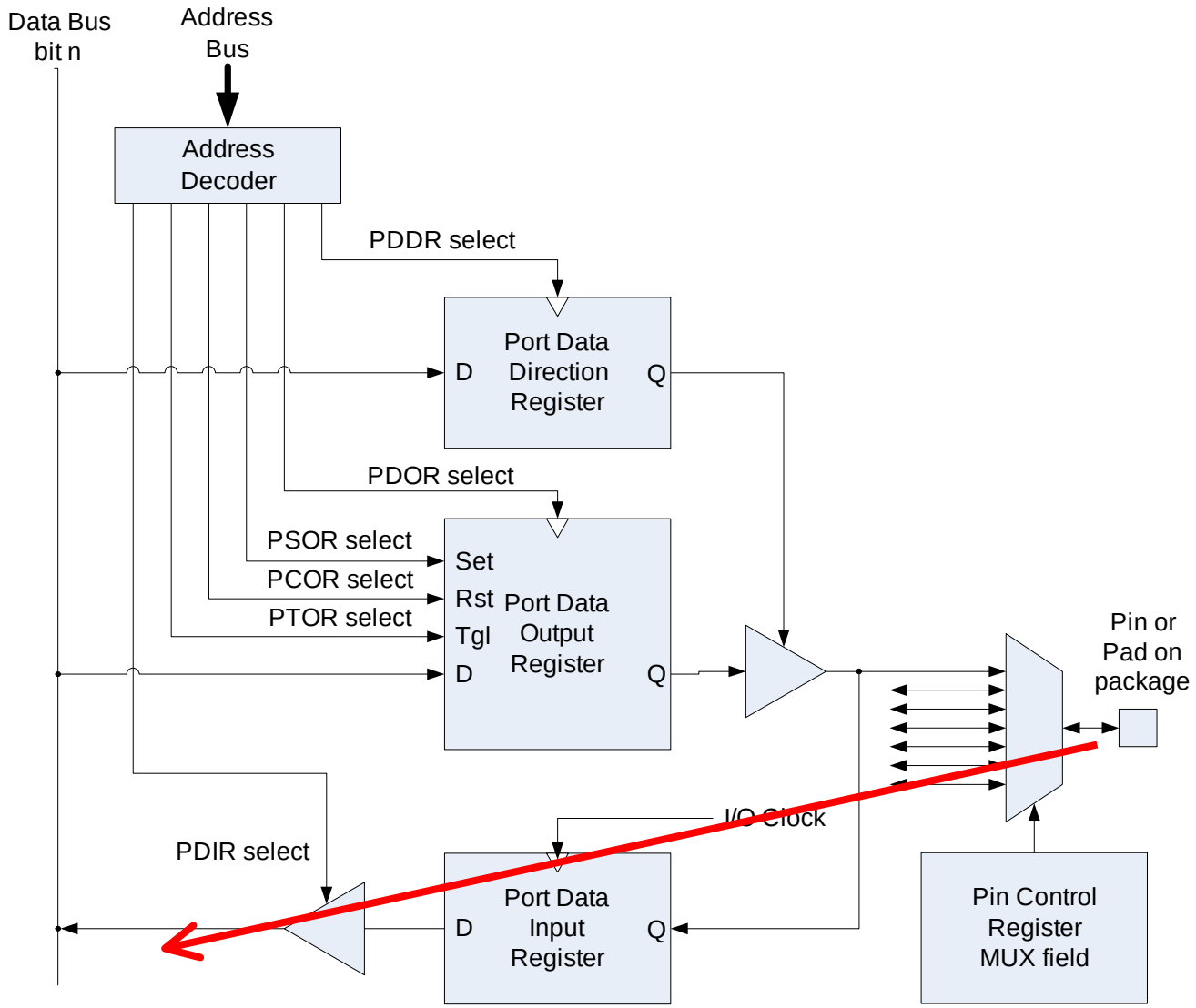
Writing Output Port Data

- **Direct: write value to PDOR**
- **Toggle: write 1 to PTOR**
- **Clear (to 0): Write 1 to PCOR**
- **Set (to 1): write 1 to PSOR**



Reading Input Port Data

- **Read from PDIR**



Pseudocode for Program

Example: light either LED1 or LED2 based on switch SW1 position

```
// Make PTA1 and PTA2 outputs
set bits 1 and 2 of GPIOA_PDDR
// Make PTA5 input
clear bit 5 of GPIOA_PDDR
// Initialize the output data values: LED 1 off, LED 2 on
clear bit 1, set bit 2 of GPIOA_PDOR
// read switch, light LED accordingly
do forever {
      if bit 5 of GPIOA_PDIR is 1 {
            // switch is not pressed, then light LED 2
            set bit 2 of GPIOA_PDOR
            clear bit 1 of GPIO_PDOR
      } else {
            // switch is pressed, so light LED 1
            set bit 1 of GPIOA_PDOR
            clear bit 2 of GPIO_PDOR
      }
}
```

CMSIS - Accessing Hardware Registers in C

- **CMSIS - C**ortex **M**icrocontroller **S**oftware **I**nterface **S**tandard
- **Header file MKL25Z4.h defines C data structure types to represent hardware registers in MCU**

```
#define   __I     volatile const
#define   __O     volatile
#define   __IO    volatile

/** GPIO - Register Layout Typedef */
typedef struct {
  __IO uint32_t PDOR;  /**< Data Output, offset: 0x0      */
  __O  uint32_t PSOR;  /**< Set Output, offset: 0x4       */
  __O  uint32_t PCOR;  /**< Clear Output, offset: 0x8     */
  __O  uint32_t PTOR;  /**< Toggle Output, offset: 0xC    */
  __I  uint32_t PDIR;  /**< Data Input, offset: 0x10      */
  __IO uint32_t PDDR;  /**< Data Direction, offset: 0x14  */
} GPIO_Type;
```

Accessing Hardware Registers in C (2)

- **Header file MKL25Z4.h defines pointers to the registers**

```
/* GPIO - Peripheral instance base addresses */
/** Peripheral PTA base address */
#define PTA_BASE  (0x400FF000u)
/** Peripheral PTA base pointer */
#define PTA       ((GPIO_Type *)PTA_BASE)

PTA->PDOR = …
```

Coding Style and Bit Access

- **Easy to make mistakes dealing with literal binary and hexadecimal values**
  - "To set bits 13 and 19, use 0000 0000 0000 1000 0010 0000 0000 0000 or 0x00082000"

- **Make the literal value from shifted bit positions**

```
n = (1UL << 19) | (1UL << 13);
```

- **Define names for bit positions**

```
#define GREEN_LED_POS (19)
#define YELLOW_LED_POS (13)
n = (1UL << GREEN_LED_POS) | (1UL << YELLOW_LED_POS);
```

- **Create macro to do shifting to create mask**

```
#define MASK(x) (1UL << (x))
n = MASK(GREEN_LED_POS) | MASK(YELLOW_LED_POS);
```

Using Masks

- **Overwrite existing value in n with mask**

```
n = MASK(foo);
```

- **Set in n all the bits which are one in mask, leaving others unchanged**

```
n |= MASK(foo);
```

- **Complement the bit value of the mask**

```
~MASK(foo);
```

- **Clear in n all the bits which are zero in mask, leaving others unchanged**

```
n &= MASK(foo);
```

- **Testing a bit value in register**

```
if ( n & MASK(foo) == 0) ...
if ( n & MASK(foo) == 1) ...
```

C Code

```
#define LED1_POS (1)
#define LED2_POS (2)
#define SW1_POS (5)
#define MASK(x) (1UL << (x))

PTA->PDDR  |= MASK(LED1_POS)
           | MASK (LED2_POS);  // set LED bits to outputs
PTA->PDDR &= ~MASK(SW1_POS);   // clear Switch bit to input

PTA->PDOR = MASK(LED2_POS);    // turn on LED1, turn off LED2

while (1) {
    if (PTA->PDIR & MASK(SW1_POS)) {       //test switch bit
      // switch is not pressed, then light LED 2
      PTA->PDOR = MASK(LED2_POS);
    } else {
      // switch is pressed, so light LED 1
      PTA->PDOR = MASK(LED1_POS);
    }
}
```

Clocking Logic

Bit	Port
13	PORTE
12	PORTD
11	PORTC
10	PORTB
9	PORTA

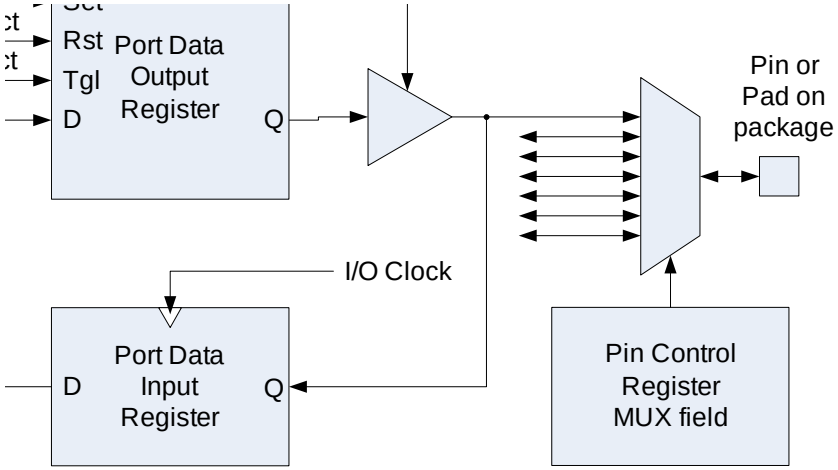
- Need to enable clock to GPIO module
- By default, GPIO modules are disabled to save power
- Writing to an unclocked module triggers a hardware fault!
- Control register SIM_SCGC5 gates clocks to GPIO ports
- Enable clock to Port A

```
SIM->SCGC5 |= (1UL <<  9);
```

- Header file MKL25Z4.h has definitions

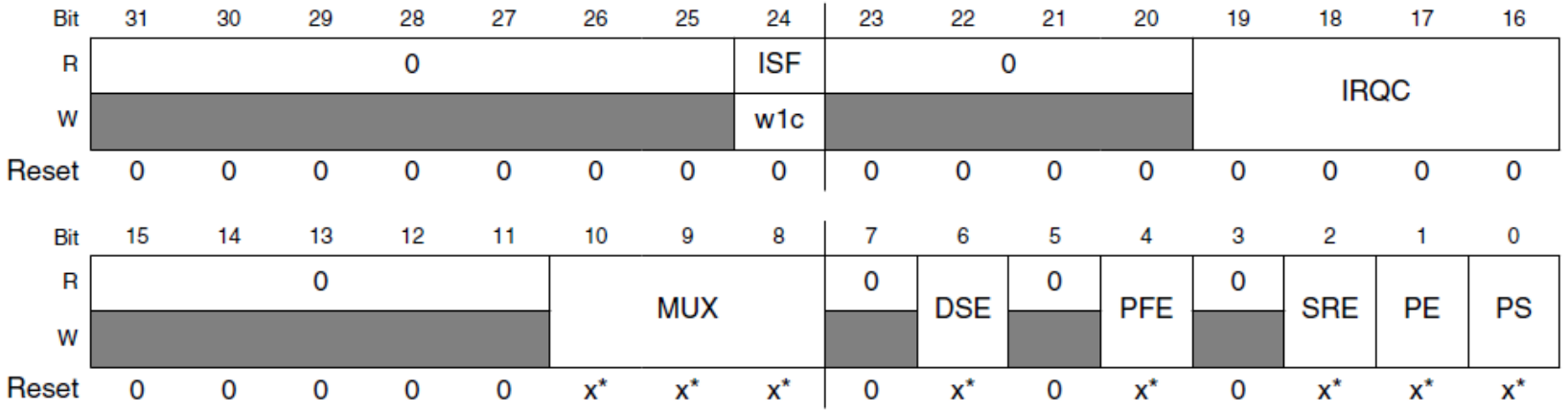
```
SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
```

Connecting a GPIO Signal to a Pin



- **Multiplexer used to increase configurability - what should pin be connected with internally?**
- **Each configurable pin has a 32-bits Pin Control Register**

Pin Control Register (PCR)



80 LQFP	64 LQFP	48 QFN	32 QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
64	52	40	28	PTC7	CMP0_IN1	CMP0_IN1	PTC7	SPI0_MISO			SPI0_MOSI		
65	53	—	—	PTC8	CMP0_IN2	CMP0_IN2	PTC8	I2C0_SCL	TPM0_CH4				

- **MUX field of PCR defines connections**

MUX (bits 10-8)	Configuration
000	Pin disabled (analog)
001	Alternative 1 – GPIO
010	Alternative 2
011	Alternative 3
100	Alternative 4
101	Alternative 5
110	Alternative 6
111	Alternative 7

CMSIS C Support for PCR

- **MKL25Z4.h defines PORT_Type structure with a PCR field (array of 32 integers)**

```
/** PORT - Register Layout Typedef */
typedef struct {
  __IO uint32_t PCR[32];  /** Pin Control Register n, array offset: 0x0, array step: 0x4 */
  __O  uint32_t GPCLR;    /** Global Pin Control Low Register, offset: 0x80 */
  __O  uint32_t GPCHR;    /** Global Pin Control High Register, offset: 0x84 */
       uint8_t RESERVED_0[24];
  __IO uint32_t ISFR;/** Interrupt Status Flag Register, offset: 0xA0 */
} PORT_Type;
```

CMSIS C Support for PCR

- **Header file defines pointers to PORT_Type registers**

```
/* PORT - Peripheral instance base addresses */
/** Peripheral PORTA base address */
#define PORTA_BASE   (0x40049000u)
/** Peripheral PORTA base pointer */
#define PORTA        ((PORT_Type *)PORTA_BASE)
```

- **Also defines macros and constants**

```
#define PORT_PCR_MUX_MASK 0x700u
#define PORT_PCR_MUX_SHIFT     8
#define PORT_PCR_MUX(x)
  (((uint32_t)(((uint32_t)(x))<<PORT_PCR_MUX_SHIFT))
  &PORT_PCR_MUX_MASK)
```

Resulting C Code for Clock Control and Mux

```
// Enable Clock to Port A
SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

// Make 3 pins GPIO
PORTA->PCR[LED1_POS] &= ~PORT_PCR_MUX_MASK;
PORTA->PCR[LED1_POS] |= PORT_PCR_MUX(1);
PORTA->PCR[LED2_POS] &= ~PORT_PCR_MUX_MASK;
PORTA->PCR[LED2_POS] |= PORT_PCR_MUX(1);
PORTA->PCR[SW1_POS] &= ~PORT_PCR_MUX_MASK;
PORTA->PCR[SW1_POS] |= PORT_PCR_MUX(1);
```

IOPORT module (Fast GPIO)

- The IOPORT registers are at a different address than the GPIO registers but they point to the same register in the control.
- Any writes to the IOPORT register will result in a change to the the corresponding GPIO register.
- Clocking is controlled to allow the core to do single cycle access to the GPIO register through the IOPORT mapped registers.
- Normal accesses to the GPIO registers take several cycles because it is internally connected to the peripheral bus.

FGPIO: Single Cycle I/O Port

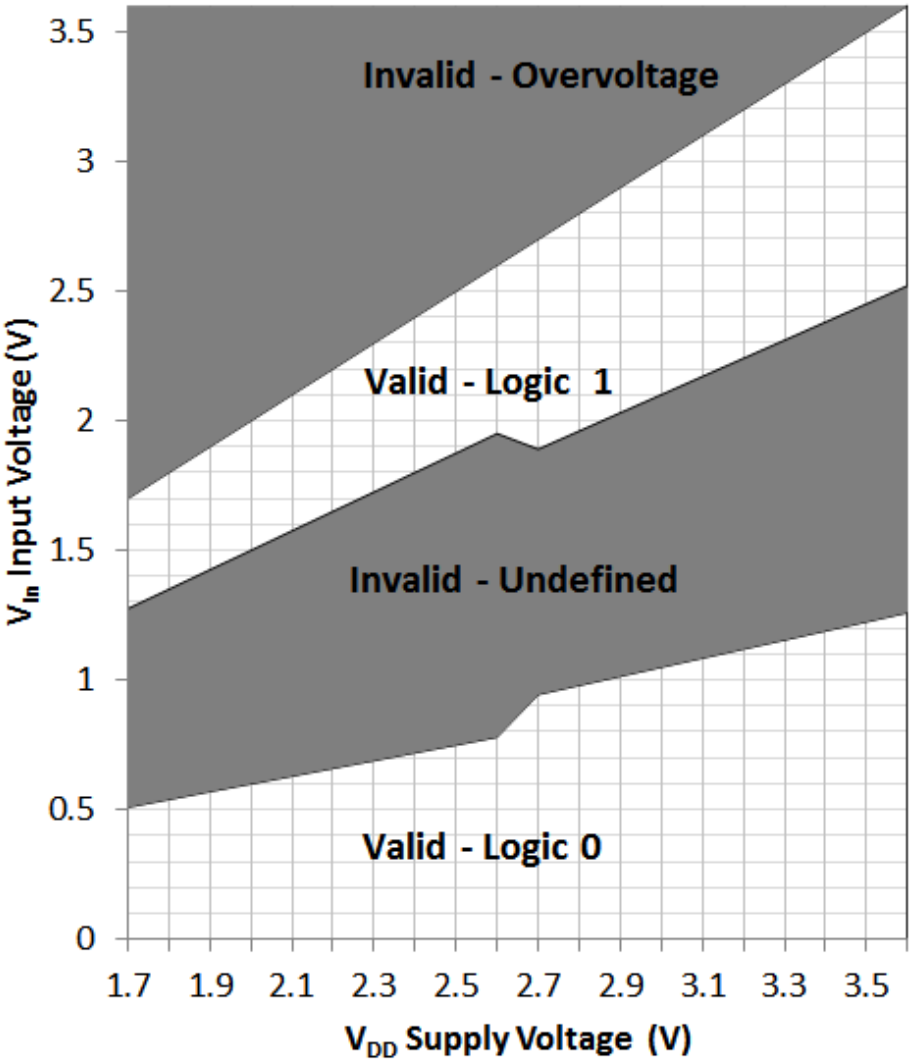
```
typedef struct {
    __IO uint32_t PDOR;  /**< Port Data Output Register, offset: 0x0 */
    __O  uint32_t PSOR;  /**< Port Set Output Register, offset: 0x4 */
    __O  uint32_t PCOR;  /**< Port Clear Output Register, offset: 0x8 */
    __O  uint32_t PTOR; /**< Port Toggle Output Register, offset: 0xC */
    __I  uint32_t PDIR;    /**< Port Data Input Register, offset: 0x10 */
    __IO uint32_t PDDR; /**< Port Data Direction Register, offset: 0x14 */
} FGPIO_Type;

#define FPTA_BASE  (0xF80FF000u)
/** Peripheral FPTA base pointer */
#define FPTA  ((FGPIO_Type *)FPTA_BASE)

FPTA->PDOR = MASK(LED2_POS);
```

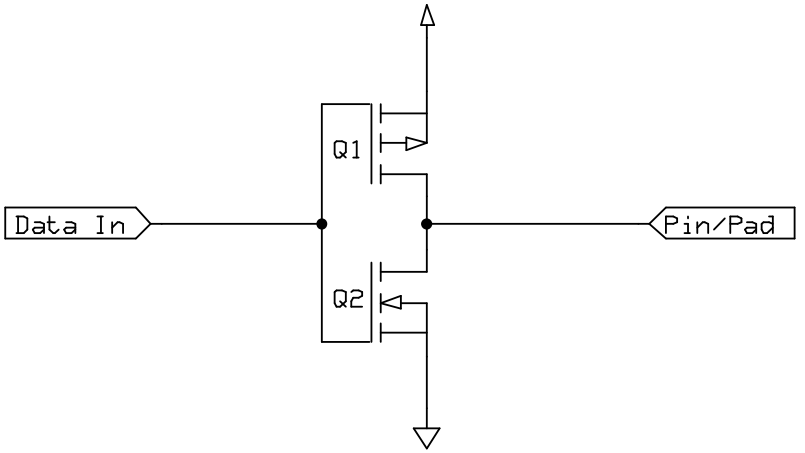
Inputs: What's a One? A Zero?

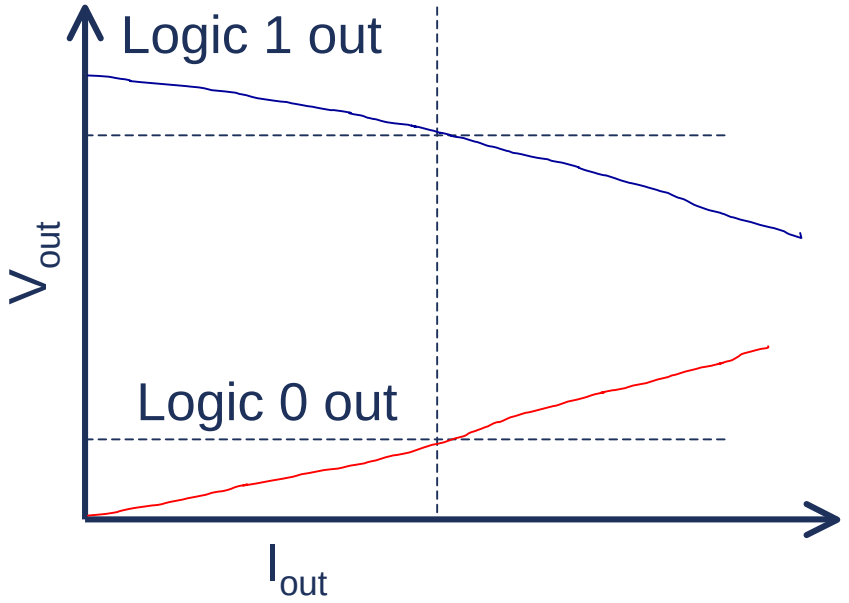
- Input signal's value is determined by voltage
- Input threshold voltages depend on supply voltage V_{DD}
- Exceeding V_{DD} or GND may damage chip



Outputs: What's a One? A Zero?

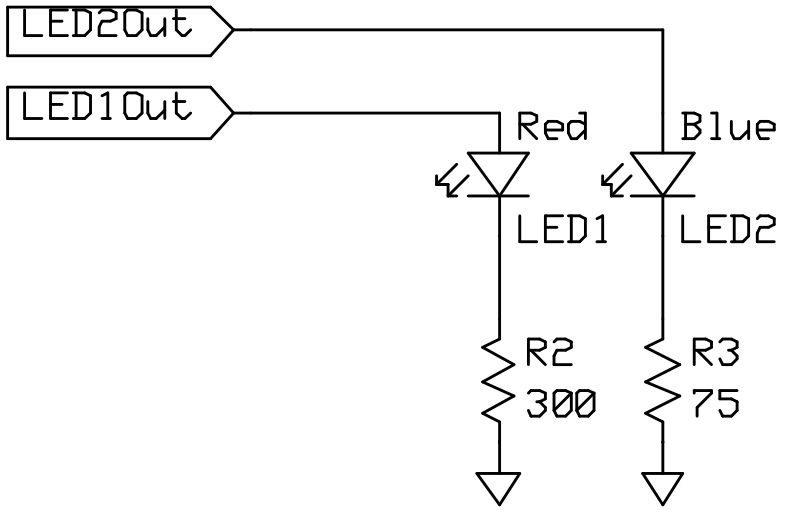
- **Nominal output voltages**
  - 1: V_{DD}-0.5 V to V_{DD}
  - 0: 0 to 0.5 V
- **Note: Output voltage depends on current drawn by load on pin**
  - Need to consider source-to-drain resistance in the transistor
  - Above values only specified when current < 5 mA (18 mA for high-drive pads) and V_{DD} > 2.7 V



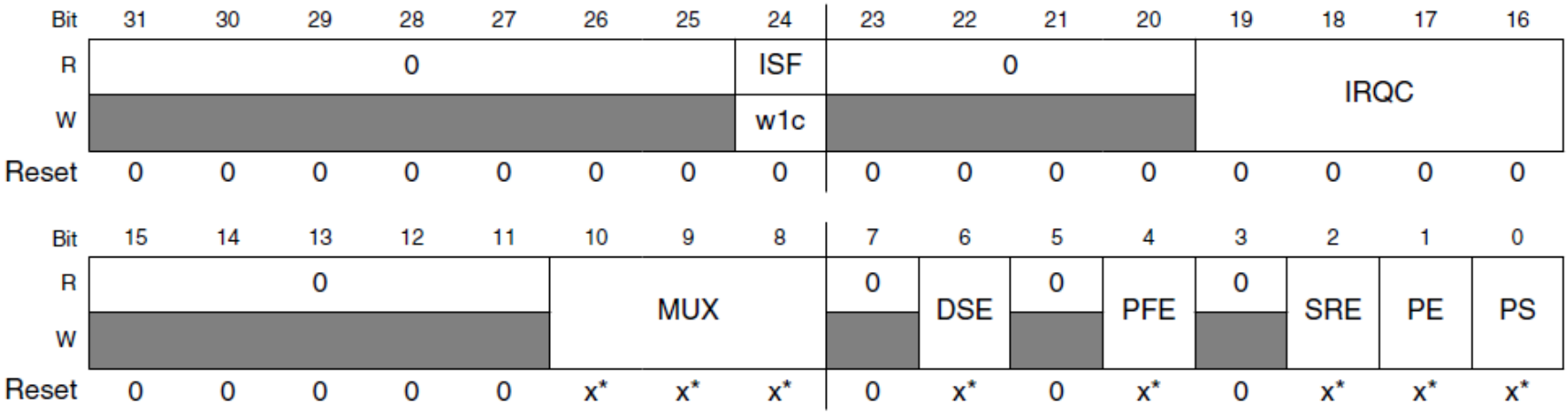


Output Example: Driving LEDs

- **Need to limit current to a value which is safe for both LED and MCU port driver**
- **Use current-limiting resistor**
  - $R = (V_{DD} – V_{LED})/I_{LED}$
- **Set I_{LED} = 4 mA**
- **V_{LED} depends on type of LED (mainly color)**
  - Red: ~1.8V
  - Blue: ~2.7 V
- **Solve for R given VDD = ~3.0 V**
  - Red: 300 Ω
  - Blue: 75 Ω



Additional Configuration in PCR



Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

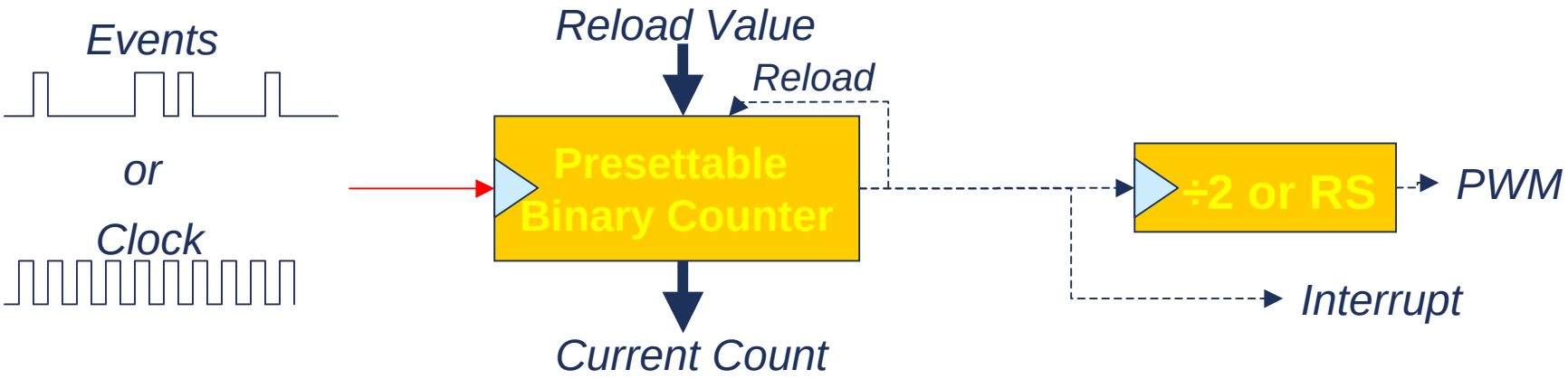
- **Pull-up and pull-down resistors**
  - Used to ensure input signal voltage is pulled to correct value when high-impedance
  - PE: Pull Enable. 1 enables the pull resistor
  - PS: Pull Select. 1 pulls up, 0 pulls down.
- **High current drive strength**
  - DSE: Set to 1 to drive more current (e.g. 18 mA vs. 5 mA @ > 2.7 V, or 6 mA vs. 1.5 mA @ <2.7 V)
  - Available on some pins - MCU dependent

Timer Peripherals

KL25 Timer Peripherals

- **PIT - Periodic Interrupt Timer**
  - Can generate periodically generate interrupts
- **TPM - Timer/PWM Module**
  - Connected to I/O pins, has input capture and output compare support
  - Can generate PWM signals
  - Can generate interrupts
- **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

Timer/Counter Peripheral Introduction

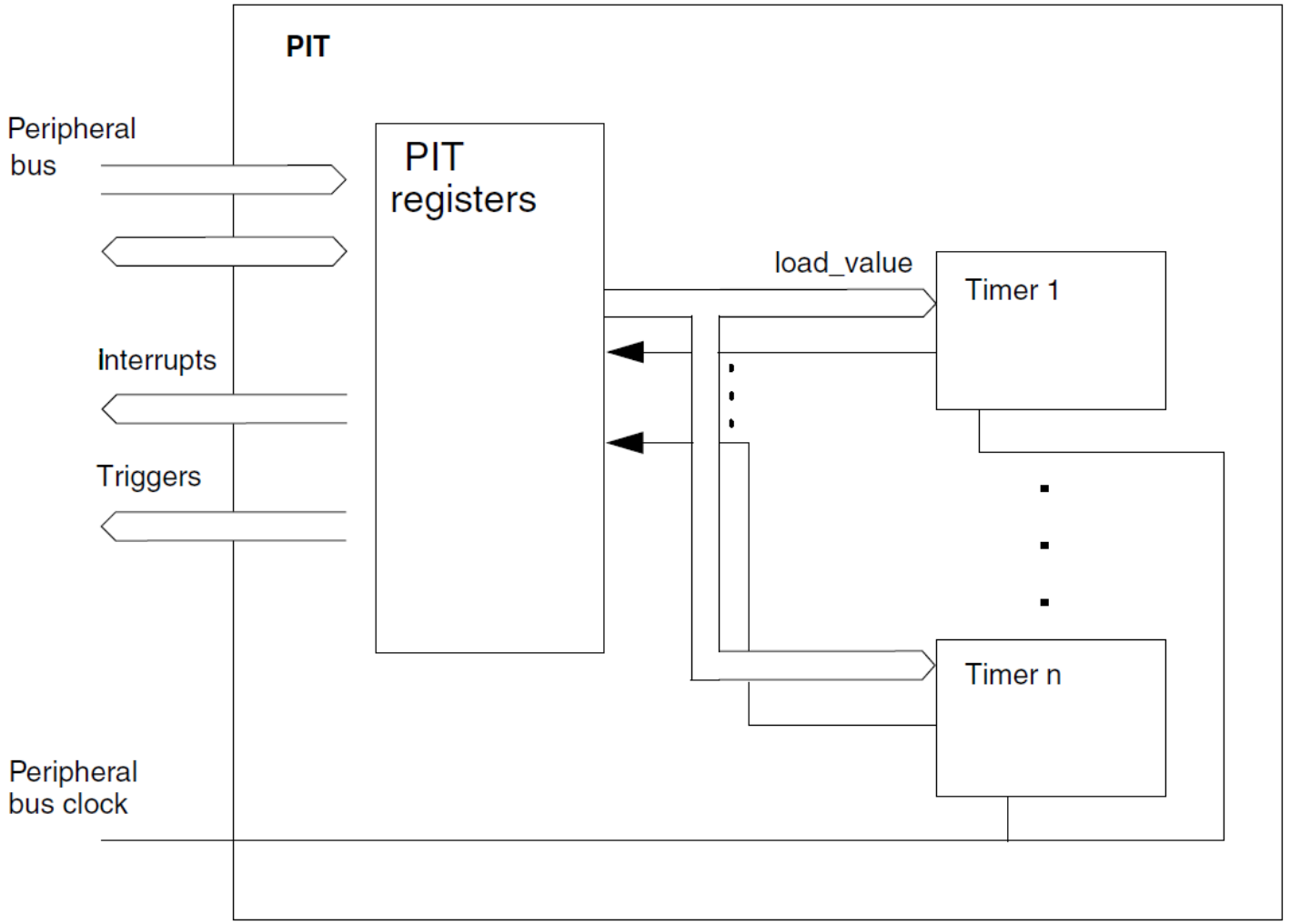


- **Common peripheral for microcontrollers**
- **Based on presettable binary counter, enhanced with configurability**
  - Count value can be read and written by MCU
  - Count **direction** can often be set to up or down
  - Counter's **clock source** can be selected
    - **Counter mode:** count **pulses** 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!

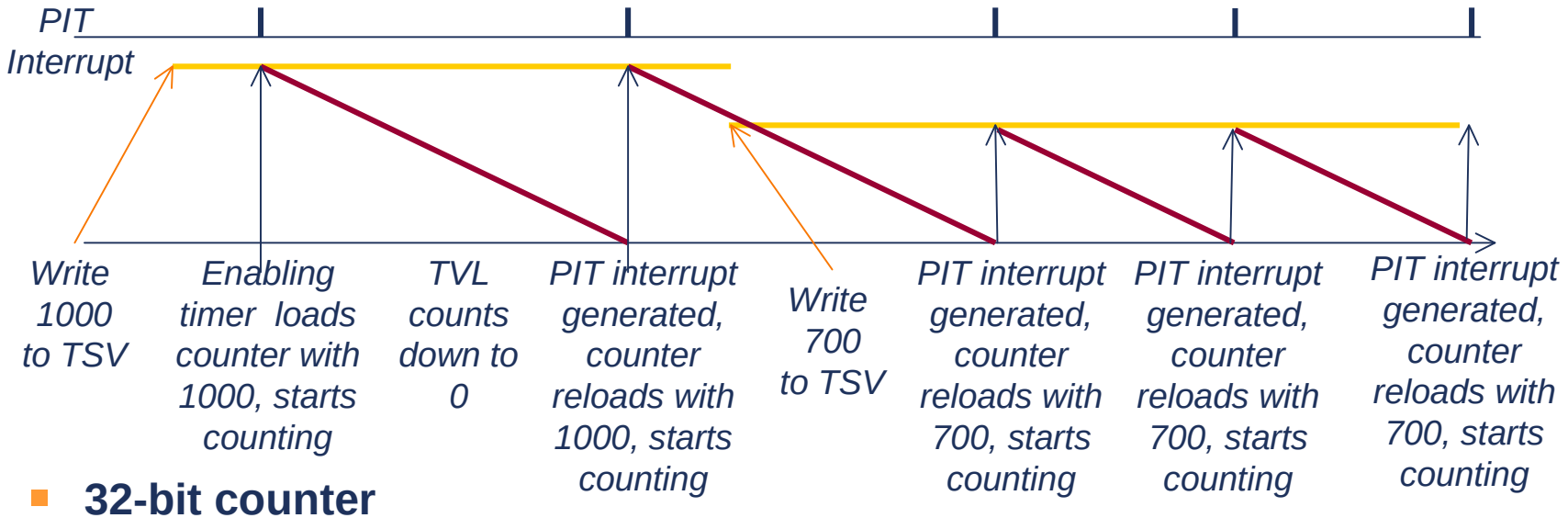
PERIODIC INTERRUPT TIMER

PIT - Periodic Interrupt Timer

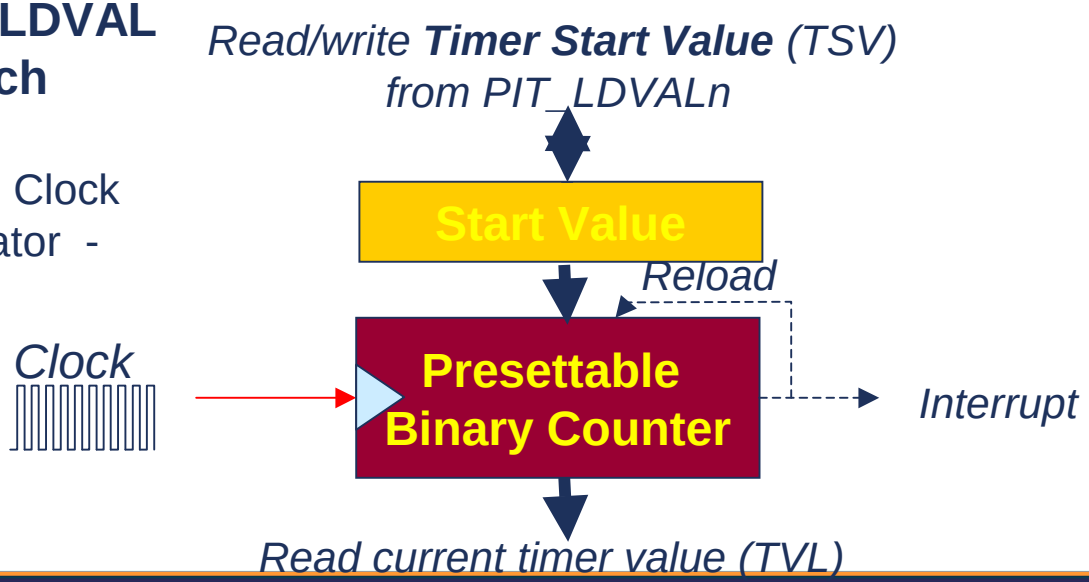
- Generates periodic interrupts with specified period



Periodic Interrupt Timer

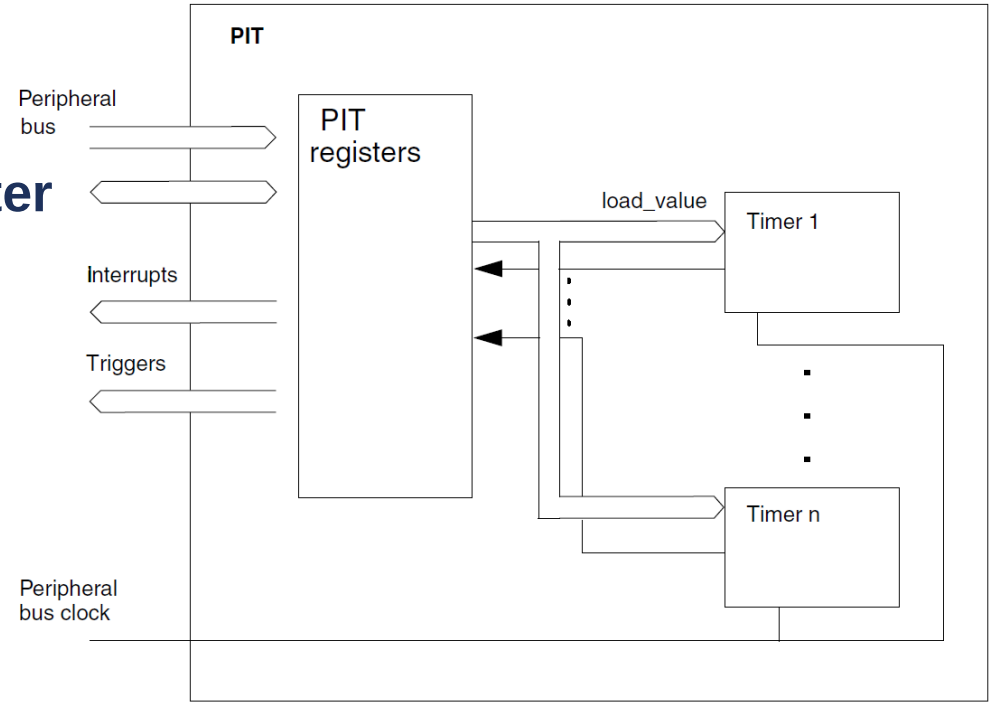


- **32-bit counter**
- **Load start value (32-bit) from LDVAL**
- **Counter counts down with each clock pulse**
  - Fixed clock source for PIT - Bus Clock from Multipurpose Clock Generator - e.g. 24 MHz
- **When timer value (CVAL) reaches zero**
  - Generates interrupt
  - Reloads timer with start value



PIT Configuration

- **First: Clock gating**
  - SIMCGC6 PIT
- **Second: Module Control Register (PIT->MCR)**
  - MDIS - Module disable
    - 0: module enabled
    - 1: module disabled (clock disabled)
  - FRZ - Freeze - stops timers in debug mode
    - 0: timers run in debug mode
    - 1: timers are frozen (don't run) in debug mode



- **Multiple timer channels within PIT**
  - KL25Z has two channels
- **Can chain timers together to create 64-bit timer**

Control of Each Timer Channel n

- **CMSIS Interface:**
  - General PIT settings accessed as struct: **PIT->MCR**, etc.
  - Channels are accessed as an array of structs: PIT->CHANNEL[n].LDVAL, etc
- **PIT_LDVALn: Load value (PIT->CHANNEL[n].LDVAL)**
- **PIT_CVALn: Current value (PIT->CHANNEL[n].CVAL)**
- **PIT_TCTRLn: Timer control (PIT->CHANNEL[n].TCTRL)**
  - CHN: Chain
    - 0: independent timer operation, uses own clock source
    - 1: timer n is clocked by underflow of timer n-1
  - TIE: Timer interrupt enable
    - 0: Timer will not generate interrupts
    - 1: Interrupt will be requested on underflow (i.e. when TIF is set)
  - TEN: Timer enable
    - 0: Timer will not count
    - 1: Timer is enabled, will count
- **PIT_TFLG0: Timer flags**
  - TIF: Timer interrupt flag
    - 1: Timeout has occurred

Configuring the PIT

- **Enable clock to PIT module**

```
SIM->SCGC6 |= SIM_SCGC6_PIT_MASK;
```

- **Enable module, freeze timers in debug mode**

```
PIT->MCR &= ~PIT_MCR_MDIS_MASK;
PIT->MCR |= PIT_MCR_FRZ_MASK;
```

- **Initialize PIT0 to count down from starting_value**

```
PIT->CHANNEL[0].LDVAL = PIT_LDVAL_TSV(starting_value);
```

- **No chaining of timers**

```
PIT->CHANNEL[0].TCTRL &= PIT_TCTRL_CHN_MASK;
```

PIT_MCR field descriptions

Field	Description
31–3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
2 Reserved	This field is reserved.
1 MDIS	Module Disable - (PIT section) Disables the standard timers. This field must be enabled before any other setup is done. 0 Clock for standard PIT timers is enabled. 1 Clock for standard PIT timers is disabled.
0 FRZ	Freeze Allows the timers to be stopped when the device enters the Debug mode. 0 Timers continue to run in Debug mode. 1 Timers are stopped in Debug mode.

Calculating Load Value

- **Goal: generate an interrupt every T seconds**
- **LDV = round($T*f_{count}$ - 1)**
  - -1 since the counter counts to 0
  - Round since LDV register is an integer, not a real number
    - Rounding provides closest integer to desired value, resulting in minimum timing error
- **Example: Interrupt every 137.41 ms**
  - LDV = 137.41 ms * 24 MHz - 1 = 3297839
- **Example: Interrupt with a frequency of 91 Hz**
  - LDV = (1/91 Hz)*24 MHz - 1 = round (263735.2637-1) = 263734

Configuring the PIT and NVIC for Interrupts

- **Configure PIT**
  - Let the PIT channel generate interrupt requests

```
PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TIE_MASK;
```

- **Configure NVIC**
  - Set PIT IRQ priority

```
NVIC_SetPriority(PIT_IRQn, 2); // 0, 1, 2 or 3
```

  - Clear any pending IRQ from PIT

```
NVIC_ClearPendingIRQ(PIT_IRQn);
```

  - Enable the PIT interrupt in the NVIC

```
NVIC_EnableIRQ(PIT_IRQn);
```

- **Make sure interrupts are not masked globally**

```
__enable_irq();
```

Table 3-7. Interrupt vector assignments (continued)

Address	Vector	IRQ[1]	NVIC IPR register number[2]	Source module	Source description
0x0000_0098	38	22	5	PIT	Single interrupt vector for all channels
0x0000_009C	39	23	5	I²S0	Single interrupt vector for all sources

PIT initialize code sample

```
void Init_PIT(unsigned period) {
    /* Enable clock to PIT module */
    SIM->SCGC6 |= SIM_SCGC6_PIT_MASK;
    /* Enable module, freeze timers in debug mode */
    PIT->MCR &= ~PIT_MCR_MDIS_MASK;
    PIT->MCR |= PIT_MCR_FRZ_MASK;
    /* Initialize PIT0 to count down from argument */
    PIT->CHANNEL[0].LDVAL = PIT_LDVAL_TSV(period)
    /* No chaining */
    PIT->CHANNEL[0].TCTRL &= PIT_TCTRL_CHN_MASK;
    /* Generate interrupts */
    PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TIE_MASK;
    /* Enable Interrupts */
    NVIC_SetPriority(PIT_IRQn, 128); // 0, 64, 128 or 192
    NVIC_ClearPendingIRQ(PIT_IRQn);
    NVIC_EnableIRQ(PIT_IRQn);
}
```

Interrupt Handler

- **One interrupt for entire PIT**

- **CMSIS ISR name: PIT_IRQHandler**
  - (can be found in startup_MKL46Z4.s)

- **ISR activities**
  - Clear pending (waiting) IRQ

    ```
    NVIC_ClearPendingIRQ(PIT_IRQn);
    ```
  - Determine which channel triggered interrupt

    ```
    if (PIT->CHANNEL[n].TFLG & PIT_TFLG_TIF_MASK) {
    ```
  - Clear interrupt request flag for channel

    ```
    PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK;
    ```
  - Do the ISR's work

ISR code sample

```
void PIT_IRQHandler() {
    /* clear pending IRQ */
    NVIC_ClearPendingIRQ(PIT_IRQn);
    /* check to see which channel triggered interrupt */
    if (PIT->CHANNEL[0].TFLG & PIT_TFLG_TIF_MASK) {
        /* clear status flag for timer channel */
        PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK;
        /* Do ISR work for channel 0*/
        .....                              }
    } else if (PIT->CHANNEL[1].TFLG & PIT_TFLG_TIF_MASK) {
        /* clear status flag for timer channel 1 */
        PIT->CHANNEL[1].TFLG &= PIT_TFLG_TIF_MASK;
        /* Do ISR work for channel 0*/
        .....
    }
}
```

Starting and Stopping the Timer Channel

- **Start the timer channel**

```
PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TEN_MASK;
```

- **Stop the timer channel**

```
PIT->CHANNEL[0].TCTRL &= ~PIT_TCTRL_TEN_MASK;
```

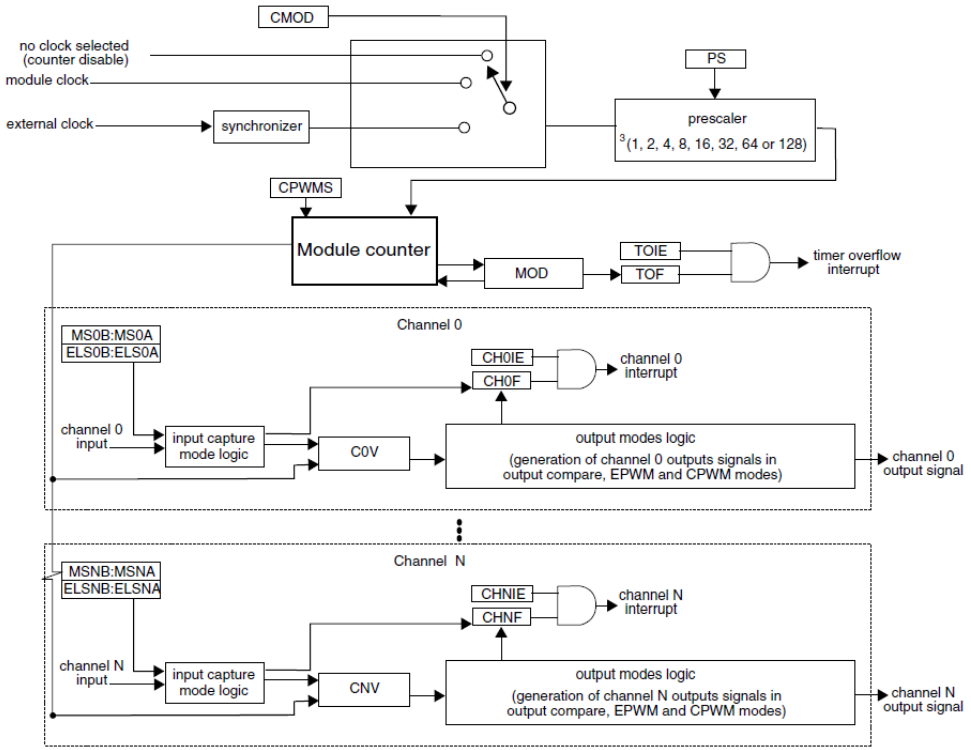
PIT_TCTRL*n* field descriptions

Field	Description
31–3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
2 CHN	Chain Mode When activated, Timer n-1 needs to expire before timer n can decrement by 1. Timer 0 cannot be chained. 0 Timer is not chained. 1 Timer is chained to previous timer. For example, for Channel 2, if this field is set, Timer 2 is chained to Timer 1.
1 TIE	Timer Interrupt Enable When an interrupt is pending, or, TFLGn[TIF] is set, enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TFLGn[TIF] must be cleared first. 0 Interrupt requests from Timer n are disabled. 1 Interrupt will be requested whenever TIF is set.
0 TEN	Timer Enable Enables or disables the timer. 0 Timer n is disabled. 1 Timer n is enabled.

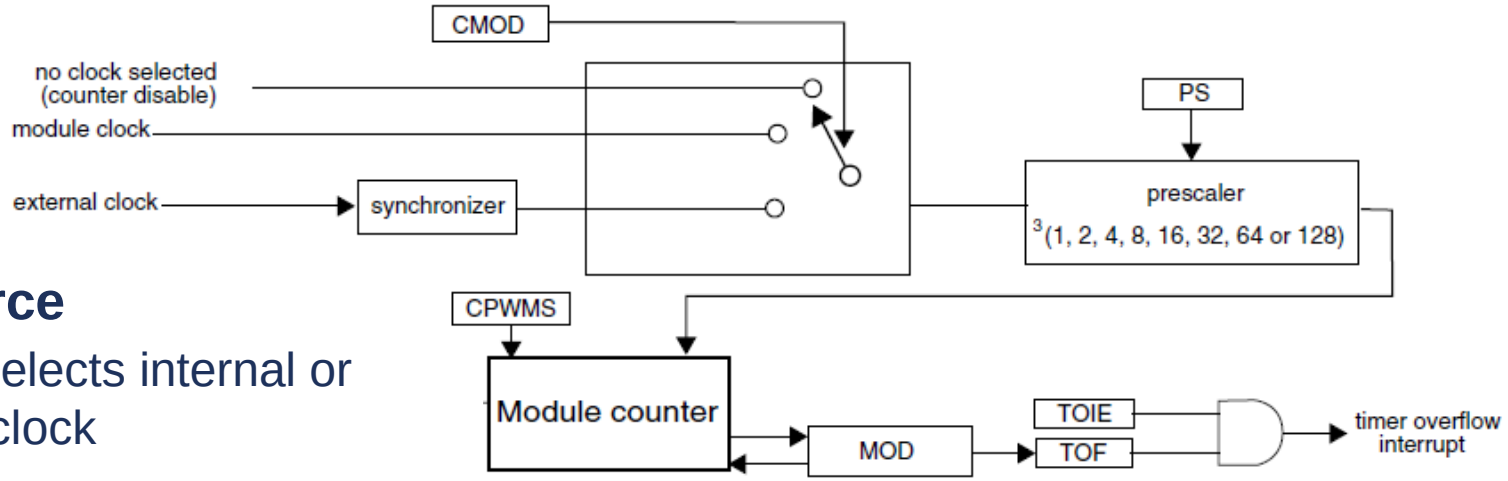
TIMER/PWM MODULE (TPM)

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.
  - One I/O pin per channel TPM_CHn
  - Each channel can generate interrupt, hardware trigger on overflow

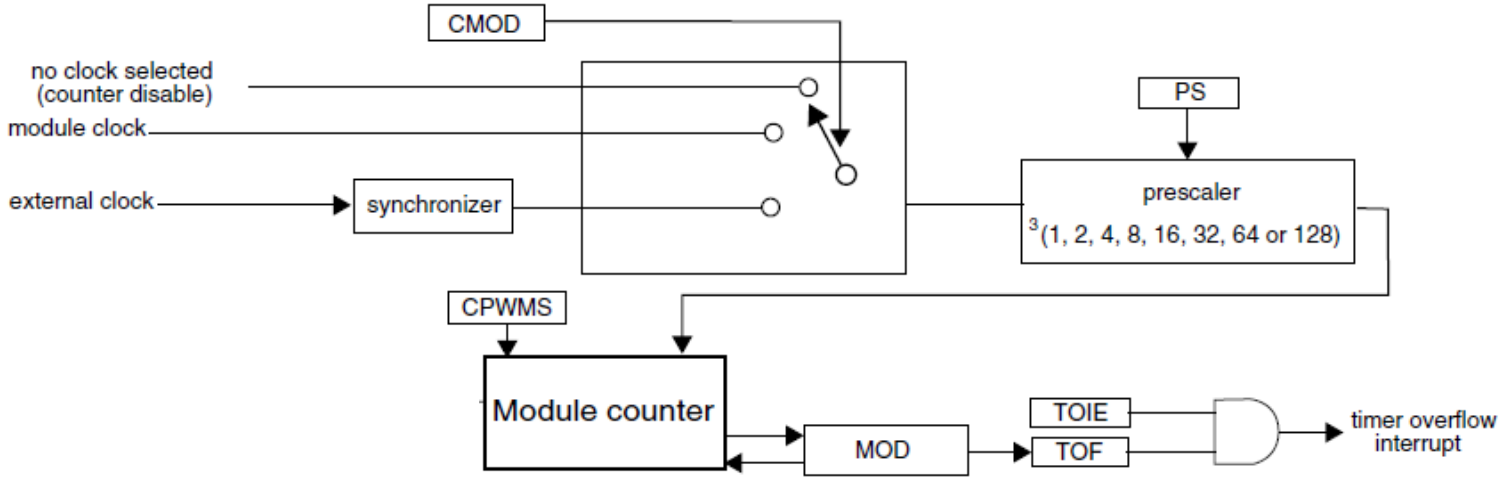


Timer Configuration



- **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**
- **TOF: Flag indicating timer has overflowed**

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

Count Mode and Modulo - Counting Up

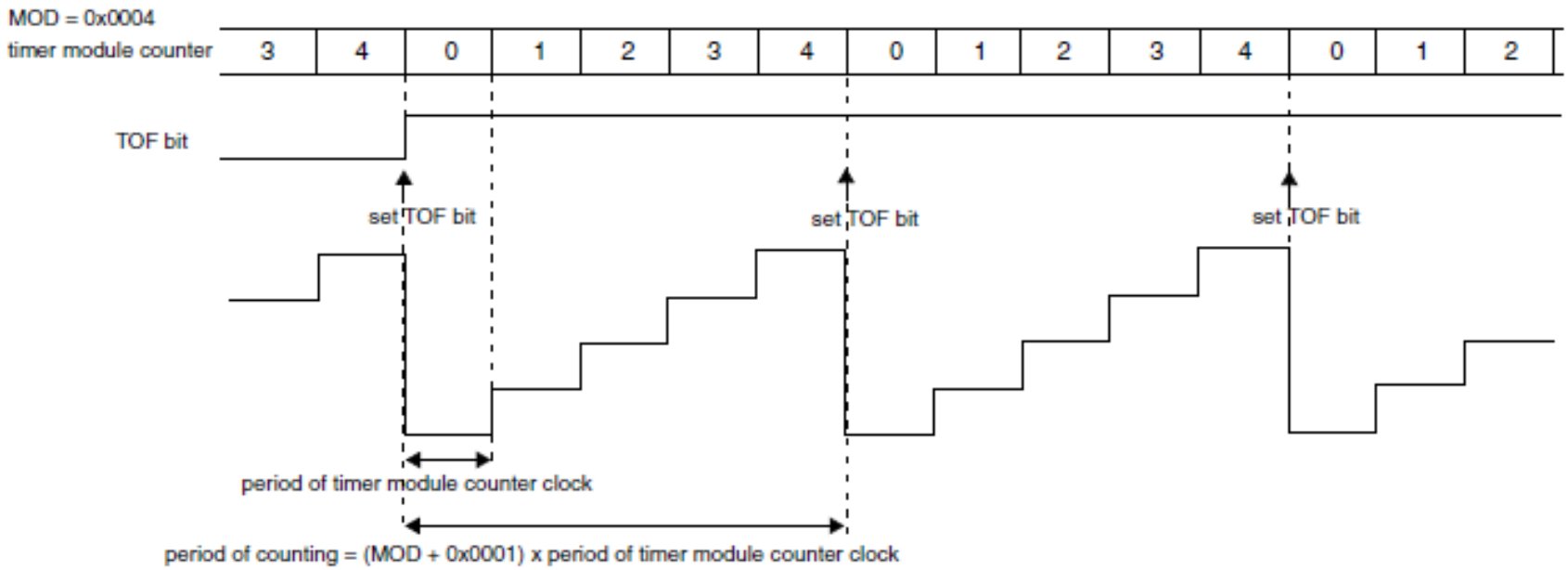


Figure 31-79. Example of TPM Up Counting

Count Mode and Modulo - Counting Up and Down

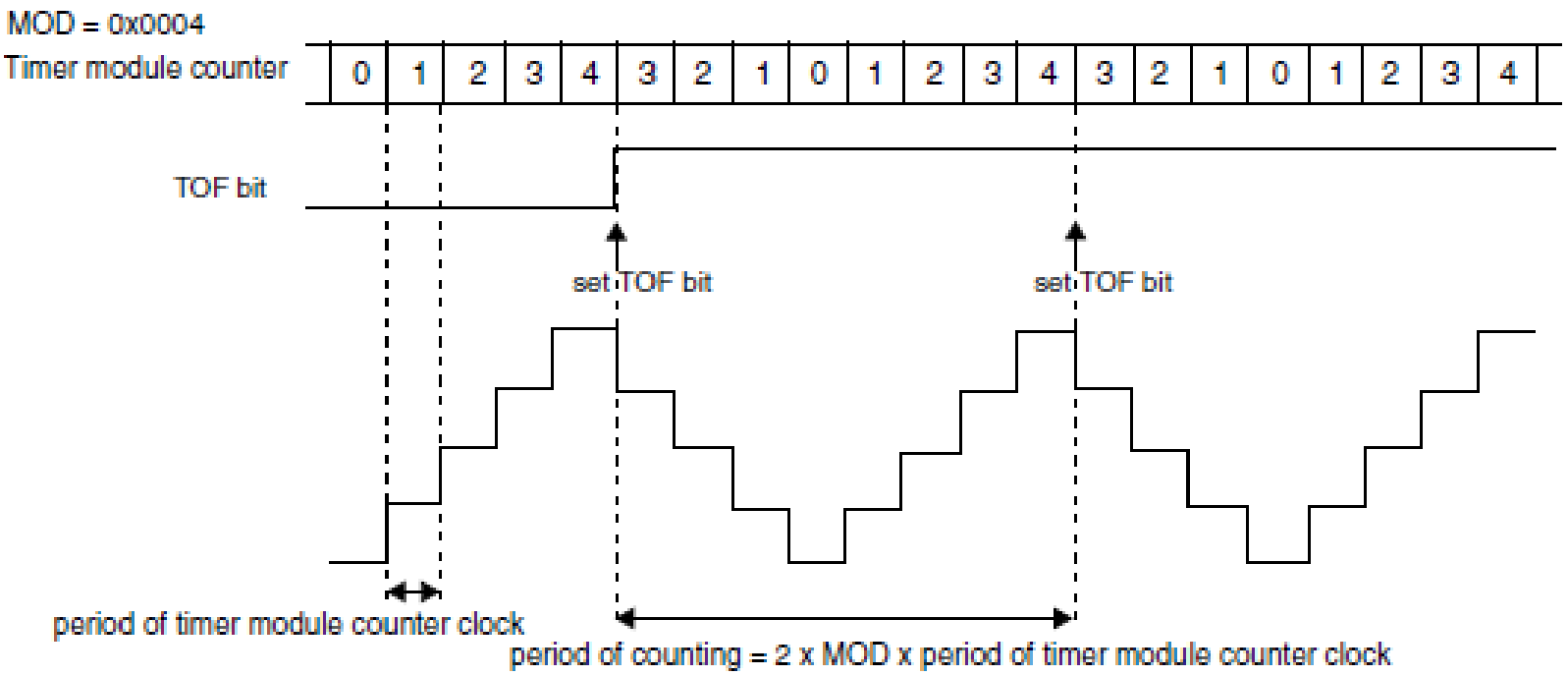


Figure 31-80. Example of Up-Down Counting

TPM Status (TPMx_STATUS)

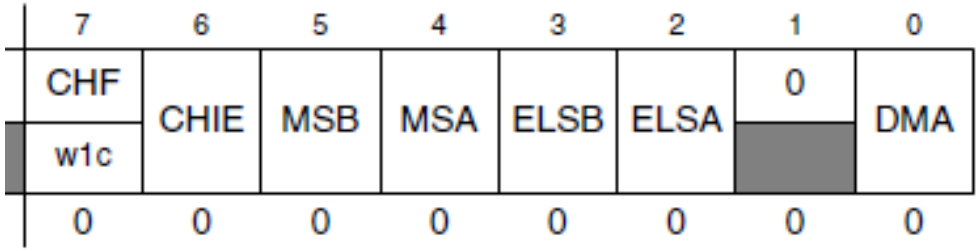
- TOF - LPTPM counter has overflowed
- CHxF - Channel event has occurred (event depends on mode)

Major Channel Modes

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

Channel registers

- **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 by hardware when event occurs on channel
  - CHIE - enable channel to generate an interrupt
  - MSB:MSA - mode select
  - ELSB:ELSA - edge or level select

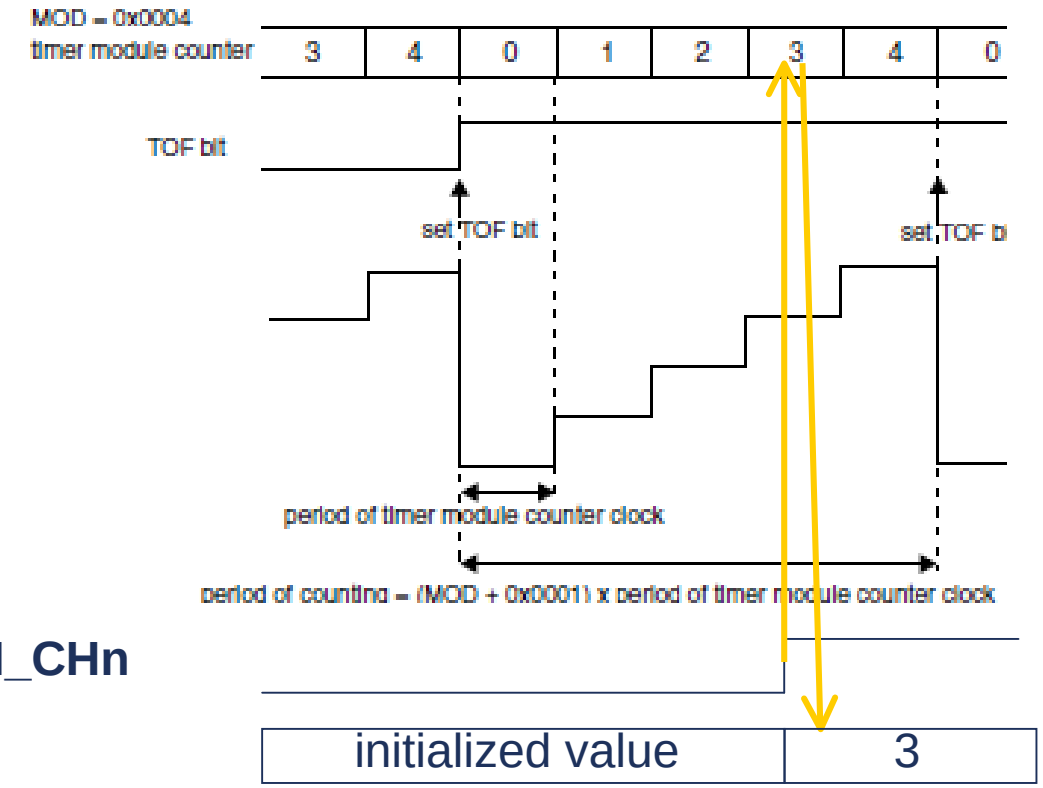
- **Value: TPMx_CnV**
  - 16-bit value for output compare or input capture

Channel Configuration and Value

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	00	00	None	Channel disabled
X	01	00	Software compare	Pin not used for TPM
0	00	01	Input capture	Capture on Rising Edge Only
		10		Capture on Falling Edge Only
		11		Capture on Rising or Falling Edge
	01	01	Output compare	Toggle Output on match
		10		Clear Output on match
		11		Set Output on match
	10	10	Edge-aligned PWM	High-true pulses (clear Output on match, set Output on reload)
		X1		Low-true pulses (set Output on match, clear Output on reload)
	11	10	Output compare	Pulse Output low on match
		01		Pulse Output high on match
1	10	10	Center-aligned PWM	High-true pulses (clear Output on match-up, set Output on match-down)
		01		Low-true pulses (set Output on match-up, clear Output on match-down)

Input Capture Mode

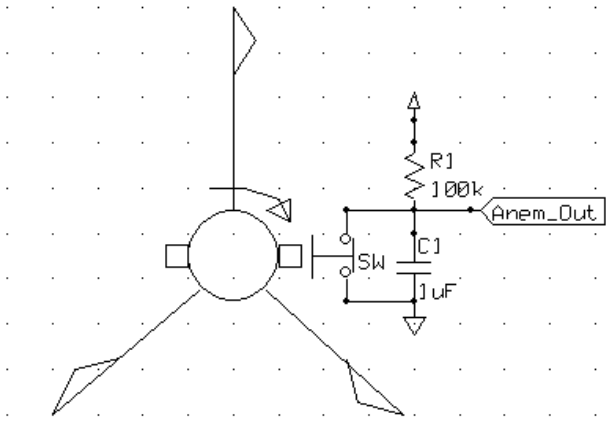
- **Select mode with CPWMS = 0, MSnB:MSnA = 00**
- **TPM_CHn I/O pin operates as edge-sensitive input**
  - ELSnB:ELSnA select rising (01) or falling edge (10) or both (11)



- **When valid edge is detected on TPM_CHn…**
  - Current value of counter is stored in CnV
  - Interrupt is enabled (if CHnIE = 1)
  - CHnF flag is set (after 3 clock delay)

Wind Speed Indicator (Anemometer)

- **How can we use the TPM for this?**
  - Use Input Capture Mode to measure period of input signal
- **Rotational speed (and pulse frequency) is proportional to wind velocity**
- **Two measurement options:**
  - Frequency (best for high speeds)
  - Width (best for low speeds)
- **Can solve for wind velocity v**



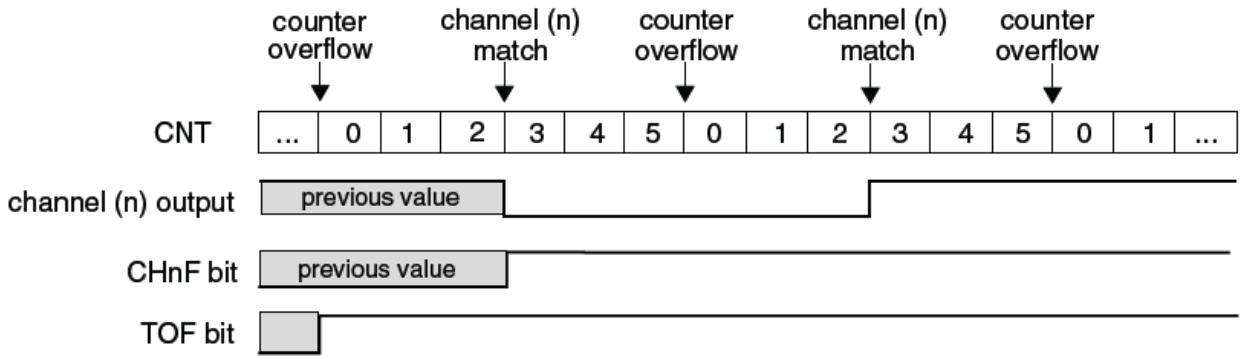
TPM Capture Mode for Anemometer

- **Configuration to measure a pulse width**
  - Set up TPM to count at given speed from internal clock
  - Set up TPM channel for input capture on rising edge

- **Operation: Repeat**
  - First TPM interrupt - on rising edge
    - Reconfigure channel for input capture on falling edge
    - Clear TPM counter, start it counting
  - Second TPM interrupt - on falling edge
    - Read capture value from CnV, save for later use in wind speed calculation
    - Reconfigure channel for input capture on rising edge
    - Clear TPM counter, start it counting

Output Compare Mode

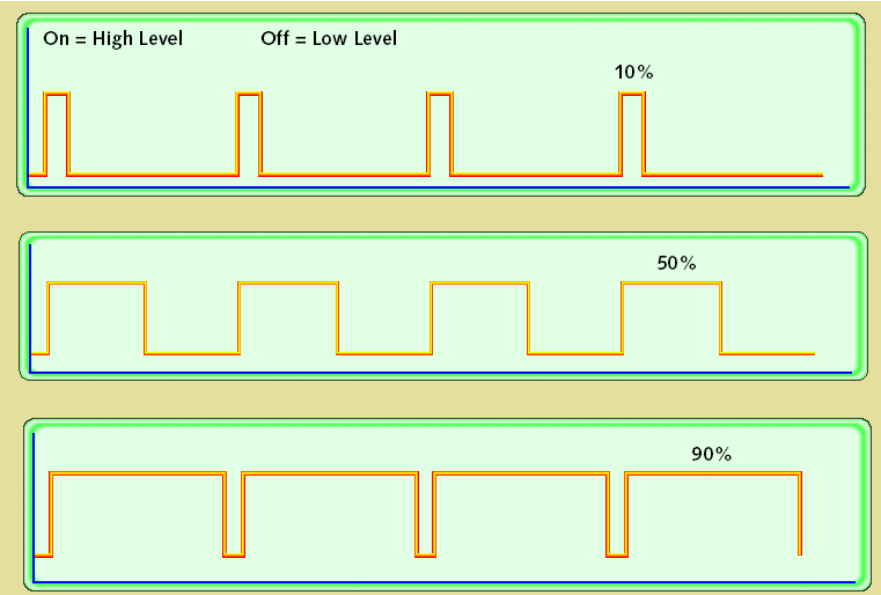
- **Select mode with CPWMS = 0, MSnA = 1**
- **TPM_CHn I/O pin operates as output, MSnB and ELSnB:ELSnA select action on match**
  - If MSnB = 0
    - Toggle (01)
    - Clear (00)
    - Set (11)
  - If MSnB = 1
    - Pulse low (10)
    - Pulse high (x1)



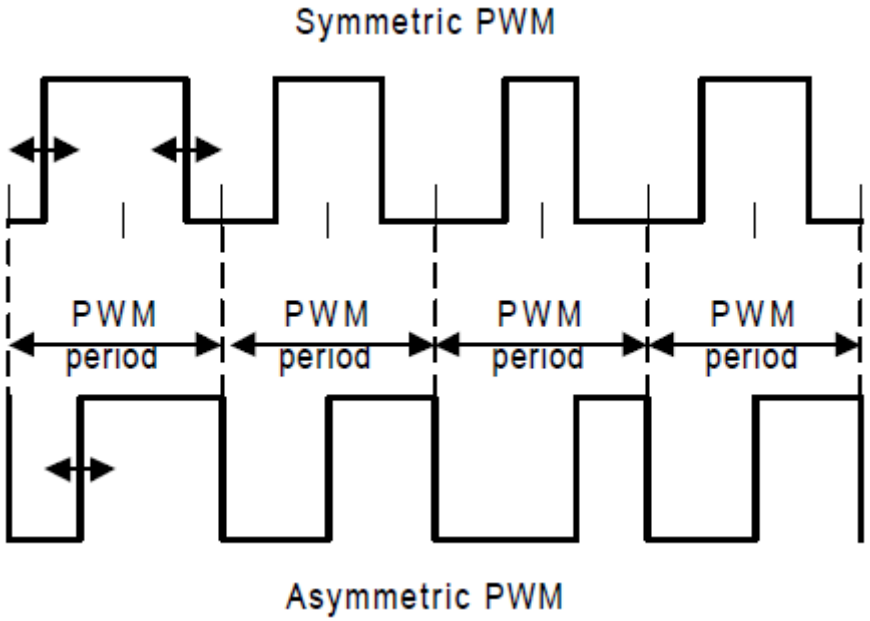
- **When CNT matches CnV …**
  - Output signal is generated
  - CHnF flag is set
  - CHnI Interrupt is enabled (if CHnIE = 1)

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**



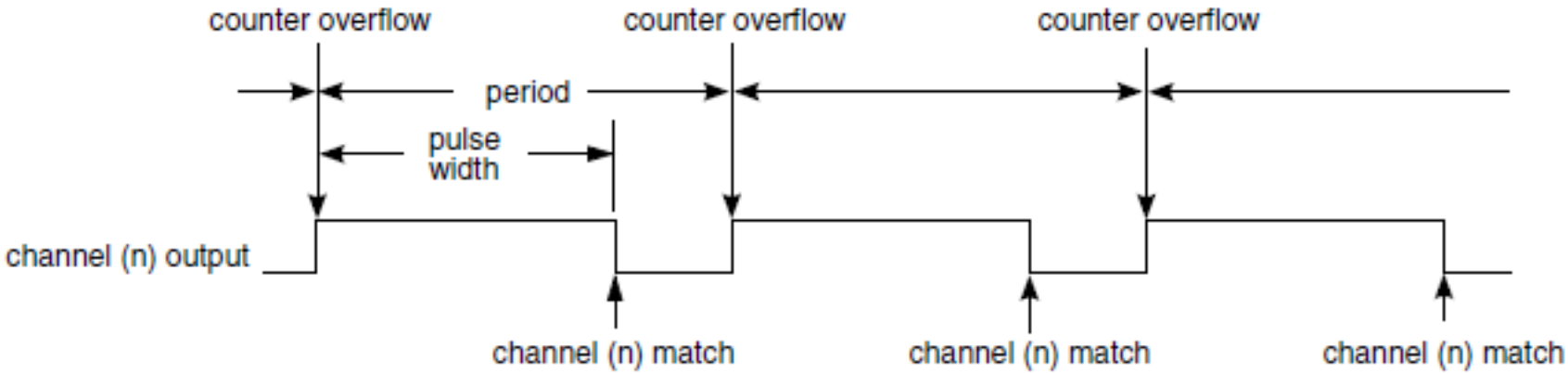
Center Aligned vs Edge Aligned PWM



- The pulses of a **symmetric PWM** signal are always symmetric with respect to the center of each PWM period.
- The pulses of an **asymmetric PWM** signal always have the same side aligned with one end of each PWM period.

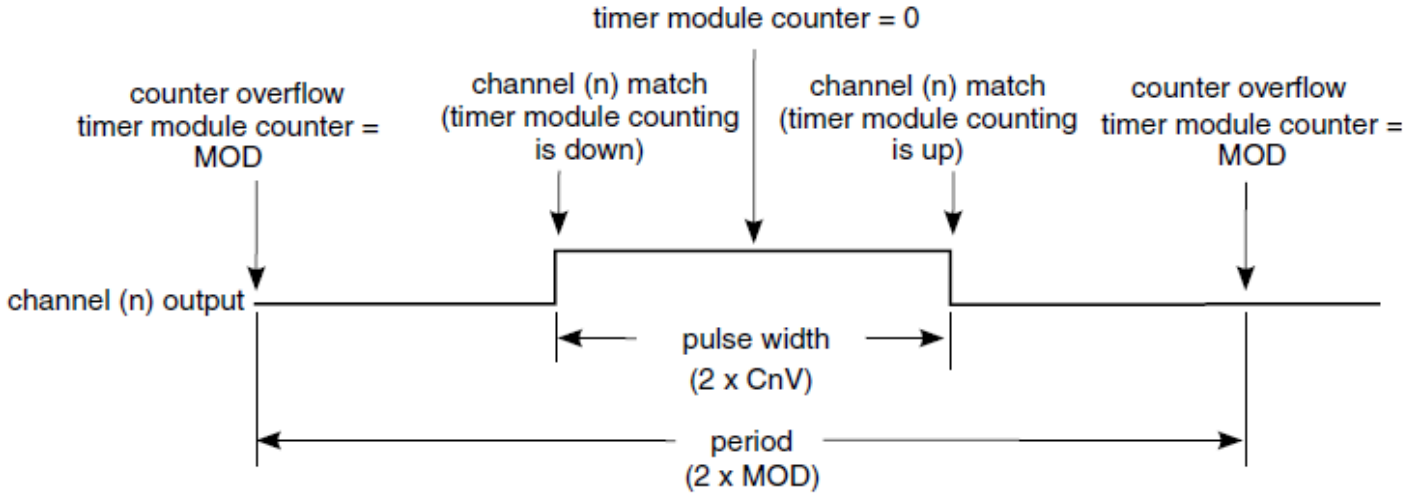
- **Also known as: Symmetric and Asymmetric PWM Signals**
- **Symmetric PWM:**
  - More complicated hardware,
  - lower maximum operation frequency,
  - generate fewer harmonics,
  - preferred for motor control applications

TPM Channel configuration registers



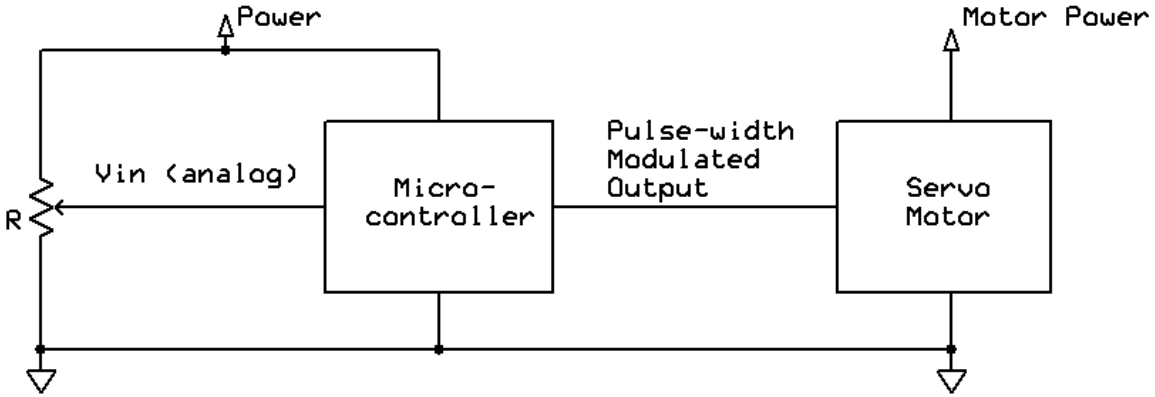
- **Edge-aligned - leading edges of signals from all PWM channels are aligned**
  - Uses count up mode
  - Period = (MOD + 1) cycles
  - Pulse width = (CnV) cycles
- **MSnB:MSnA = 01, CPWMS = 0**
  - ELSnB:ELSnA = 10 - high-true pulses
  - ELSnB:ELSnA = x1 - low-true pulses

TPM Channel for PWM Mode

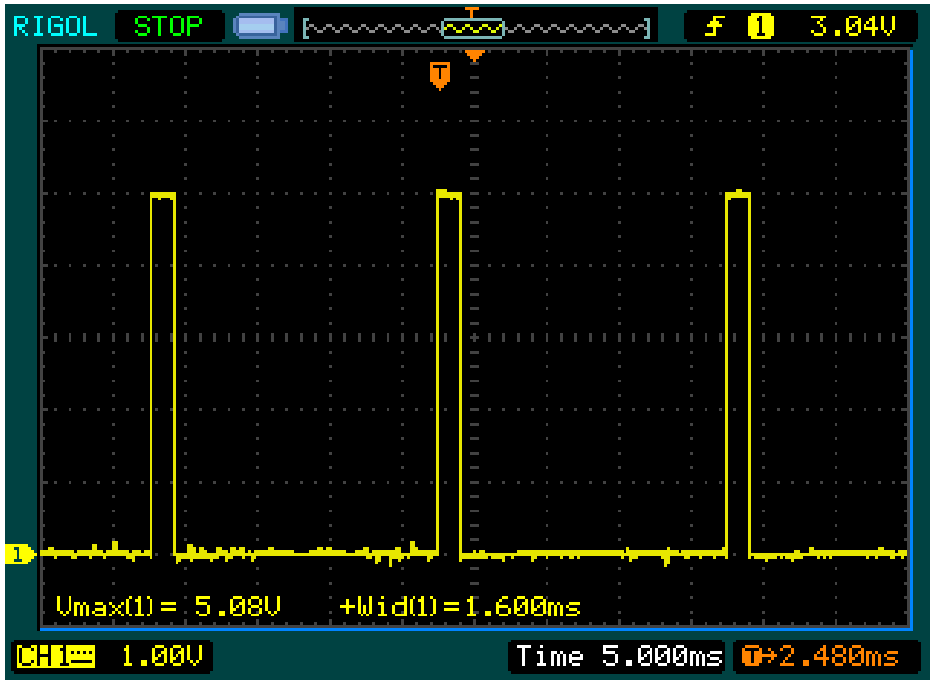


- **Center-aligned - centers of signals from all PWM channels are aligned**
  - Uses count up/down mode
  - Period = 2*MOD cycles. 0x0001 <= MOD <= 0x7FFF
  - Pulse width = 2*CnV cycles
- **MSnB:MSnA = 10, CPWMS = 1**
  - ELSnB:ELSnA = 10 - high-true pulses
  - ELSnB:ELSnA = x1 - low-true pulses

PWM to Drive Servo Motor



- **Servo PWM signal**
  - 20 ms period
  - 1 to 2 ms pulse width



TPM Triggers

- **Support selectable trigger input**
- **Trigger can:**
  - Start TPM counter
  - Reload TPM counter with zero
- **Trigger can be triggered by:**
  - External Pin
  - PITx
  - TPMx
  - RTC
  - LPTMR

TPM Configuration (TPMx_CONF)

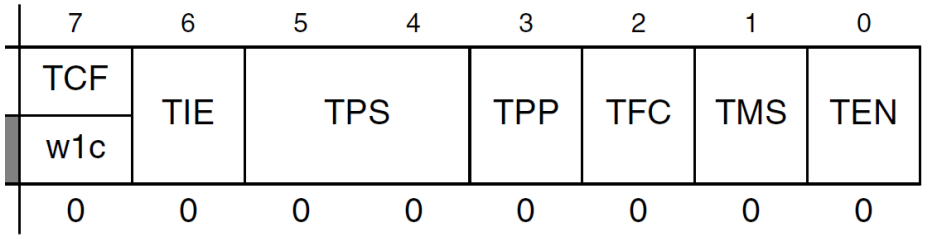
- TRGSEL - input trigger select
- CROT - counter reload on trigger
- CSOO - counter stop on overflow
- CSOT - counter start on trigger
- GTBEEN - external global time base enable (rather than LPTPM counter)
- DBGMODE - let LPTPM counter increment during debug mode
- DOZEEN - pause LPTPM when in doze mode

LOW POWER TIMER (LPTMR)

LPTMR Overview

- **Features**
  - 16 bit counter
  - Can count time or external pulses
  - Can generate interrupt when counter matches compare value
  - Interrupt wakes MCU from any low power mode
- **Registers**
  - Control Status register LPTMRx_CSR
  - Prescale register LPTMRx_PSR
  - Counter register LPTMRx_CNR
  - Compare register LPTRMx_CMR

Control Status Register

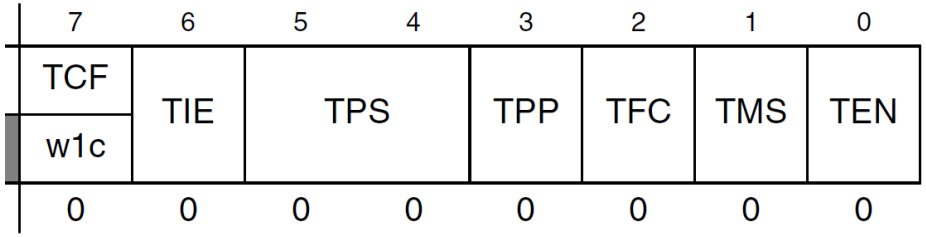


7	6	5	4	3	2	1	0
TCF	TIE	TPS		TPP	TFC	TMS	TEN
w1c							
0	0	0	0	0	0	0	0

- **TCF: Timer Compare Flag**
  - 1 if CNR matches CMR and increments
- **TIE: Timer Interrupt Enable**
  - Set to 1 to enable interrupt when TCF == 1
- **TPS: Timer Pin Select for pulse counter mode**
  - Inputs available depend on chip configuration, see KL25 SRM Chapter 3: Chip Configuration

LPTMR_CSR[TPS]	Pulse counter input number	Chip input
00	0	CMP0 output
01	1	LPTMR_ALT1 pin
10	2	LPTMR_ALT2 pin
11	3	LPTMR_ALT3 pin

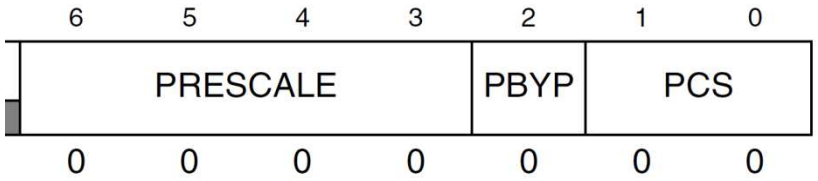
Control Status Register



7	6	5	4	3	2	1	0
TCF	TIE	TPS		TPP	TFC	TMS	TEN
w1c							
0	0	0	0	0	0	0	0

- **TPP: Timer Pin Polarity**
  - 0: input is active high, increments CNR on rising edge
  - 1: input is active low, increments CNR on falling edge
- **TFC: Timer Free-running Counter**
  - 0: Reset CNR whenever TCF is set (on match)
  - 1: Reset CNR on overflow (wrap around)
- **TMS: Timer Mode Select**
  - 0: Time counter
  - 1: Pulse counter
- **TEN: Timer Enable**
  - 1: Enable LPTMR operation

Prescale Register



- **PRESCALE: divide by 2 to 65536**
  - Time counter mode: Divide input clock by $2^{PRESCALE+1}$
  - Pulse counter mode: Is glitch filter which recognizes input signal change after $2^{PRESCALE}$ rising clock cycles
- **PBYP: Prescaler Bypass**
  - 0: use prescaler
  - 1: bypass prescaler
- **PCS: Prescaler Count Select**
  - Inputs available depend on chip configuration, see KL25 SRM Chapter 3: Chip Configuration

LPTMR0_PSR[PCS]	Prescaler/glitch filter clock number	Chip clock
00	0	MCGIRCLK — internal reference clock (not available in LLS and VLLS modes)
01	1	LPO — 1 kHz clock (not available in VLLS0 mode)
10	2	ERCLK32K (not available in VLLS0 mode when using 32 kHz oscillator)
11	3	OSCERCLK — external reference clock (not available in VLLS0 mode)