I/O Examples

D.N.Rakhmatov

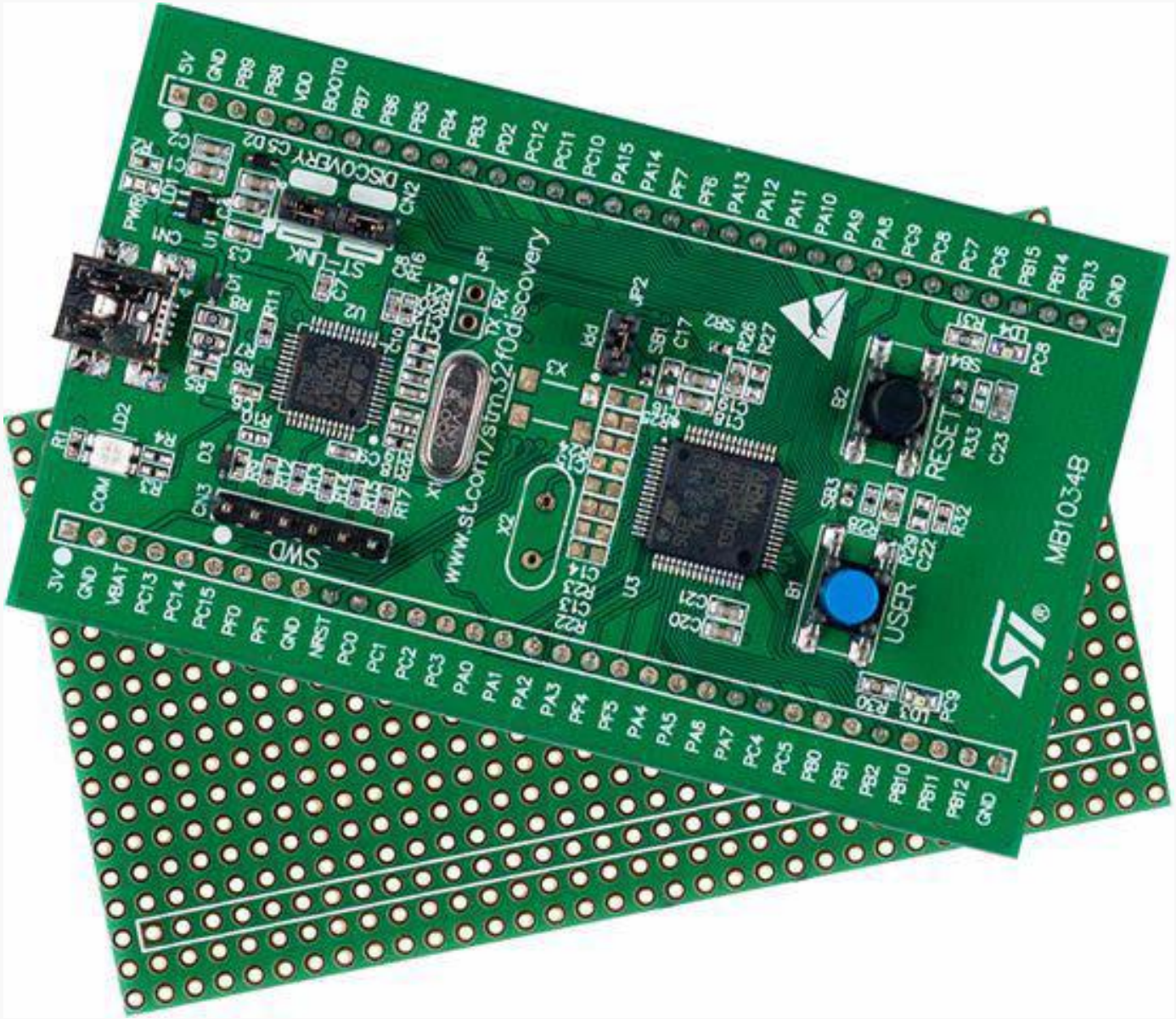
Adopted from:

STM32F0xx Advanced ARM-based 32-bit MCUs Reference Manual,

STM32F051xx ARM® Cortex™-M0 Microcontroller Data Sheet,

STM32F0DISCOVERY STM32F051R8T6 Board User Manual,

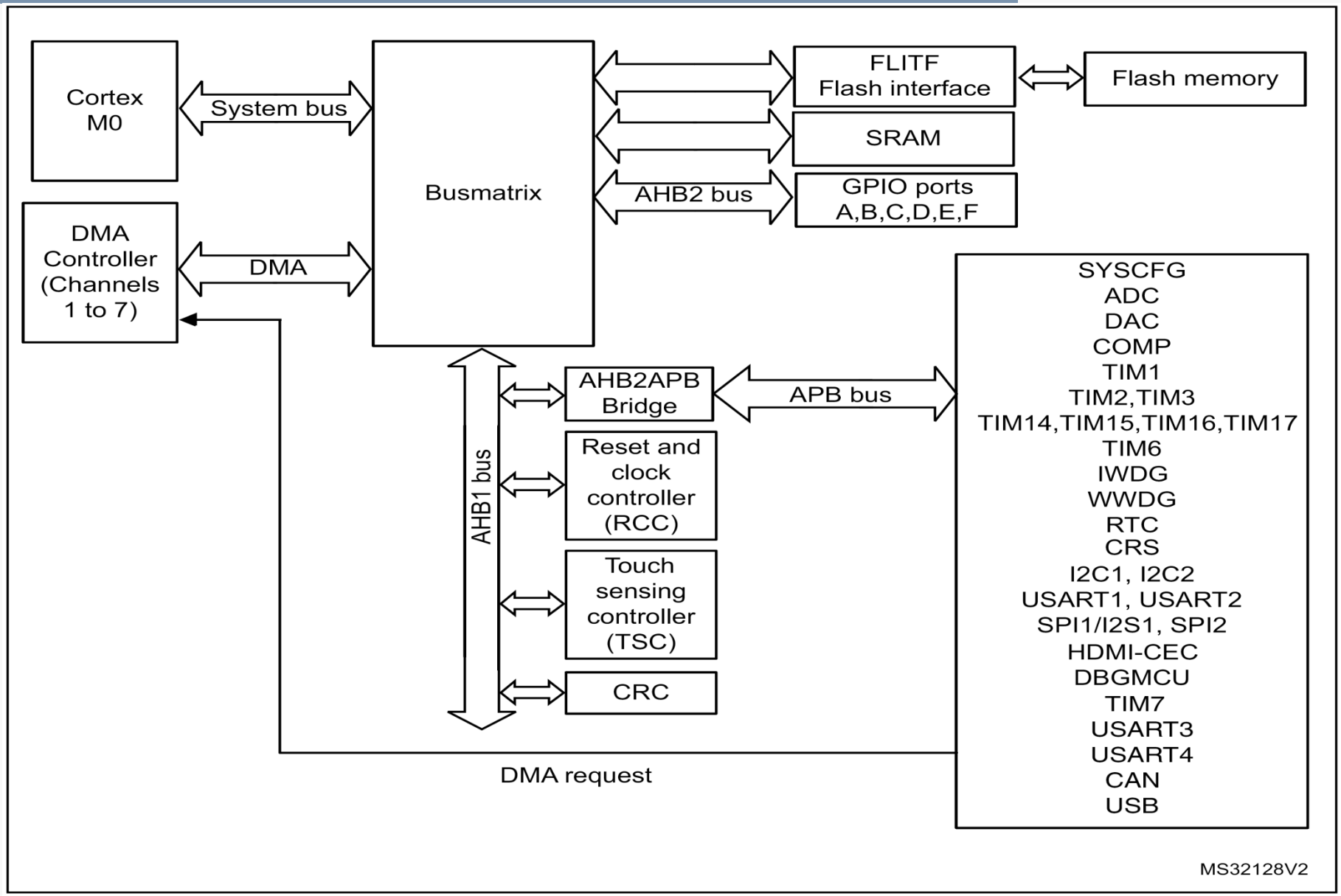
STM32F0 Discovery Board



Preview: Programing Example

- **STM32F0 Discovery** board
  - USER button – I/O pin **PA0** (input)
    - Relevant I/O registers: GPIOA_MODER, GPIOA_PUPDR, GPIOA_IDR
  - Blue and green LEDs – I/O pins **PC8** and **PC9** (output)
    - Relevant I/O registers: GPIOC_MODER, GPIOC_PUPDR, GPIOC_OTYPER, GPIOC_OSPEEDR, GPIOC_ODR, GPIOC_BSRR, GPIOC_BRR
- Application: blinking LED (either blue, or green)
  - Pressing the USER button switches the blinking LED
    - Blinking period = 1/2 s: LED is on for 1/4 s, and then off for 1/4 s
  - **TIM2** is to generate an interrupt every **1/4 s**
    - Relevant timer registers: TIM2_CR1, TIM2_PSC, TIM2_ARR, TIM2_EGR, TIM2_DIER, TIM2_SR

STM32F0xx Block Diagram



STM32F051xx Overview I

- **32-bit** ARM® Cortex™-M0 CPU core (48 MHz)
  - **Thread** mode (application), **Handler** mode (exception)
- Memory: 8K SRAM, 64K Flash, 5 DMA channels
- External communication
  - 18-Mbit/s **SPI**, 8-Mbit/s USART, 1-Mbit/s I^2C interface (two of each)
- Two general-purpose analog comparators
- 55 general-purpose I/O (**GPIO**) pins
  - GPIO pins are mapped onto **16-bit** ports
    - Some ports do not use all 16 bits
  - Each GPIO pin can also be configured to serve as an alternate function (AF) I/O supporting built-in peripherals

STM32F051xx Overview II

- ADC (analog-to-digital converter)
  - **12-bit** resolution, 16 analog input channels
- DAC (digital-to-analog converter)
  - **12-bit** resolution, 1 analog output channel
- Timers
  - 24-bit system tick (SysTick) down-counting timer STK
  - 16-bit advanced-control up/down-counting timer TIM1
  - 32-bit general-purpose up/down-counting timer **TIM2**
  - 16-bit general-purpose up/down-counting timer TIM3
  - 16-bit general-purpose up-counting timers TIM14-17
  - 16-bit basic up-counting timer TIM6 (driving DAC)
  - Two watchdog down-counting timers IWDG, WWDG

STM32F051xx Interrupts I

- Nested vectored interrupt controller (**NVIC**)
  - Part of Cortex-M0 to accelerate interrupt processing
  - Up to 32 interrupts (**IRQ**), numbered from **0** to **31**
  - IRQ numbers **0-31** ↔ Exception numbers **16-47**
    - IRQ**0** = Exception **16**, …, IRQ**31** = Exception **47**
    - Exceptions **1-15** are intended for **system use**
    - Exception **0** = **Thread** mode (application software)
  - Programmable priority level for each IRQ
    - 0 (highest priority), 64, 128, or 192 (lowest priority)
  - Fixed secondary prioritization within each priority level
    - If several pending interrupts have the same priority level, the one with the lowest exception number wins
  - Interrupt enable/disable/pending bits for each IRQ

STM32F051xx Interrupts II

- Cortex-M0 internal registers
  - Interrupt program status register (IPSR)
    - Bits **IPSR[5:0]** indicate the exception number being processed
  - Priority mask register (PRIMASK)
    - Letting **PRIMASK[0] = 1** prevents activation of all exceptions with configurable priority
- NVIC memory-mapped registers
  - Interrupt set-enable register (ISER)
  - Interrupt clear-enable register (ICER)
  - Interrupt set-pending register (ISPR)
  - Interrupt clear-pending register (ICPR)
  - Interrupt priority register (IPR0, IPR1, …, IPR7)

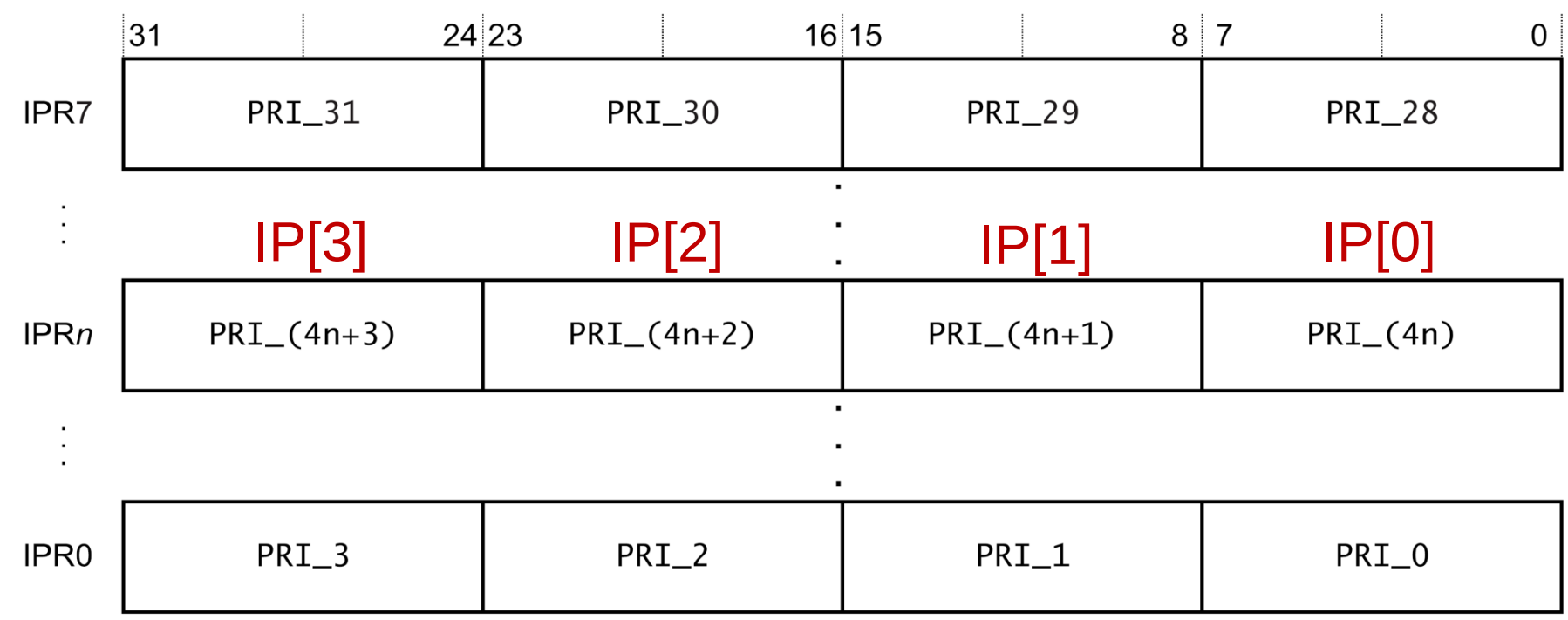
Registers ISER and ICER

- ISER bits **SETENA[31:0]**
  - Write
    - **SETENA[x] = 0:** no effect
    - **SETENA[x] = 1:** enable IRQ**x** interrupt
  - Read
    - **SETENA[x] = 0/1:** IRQ**x** interrupt is disabled/enabled
- ICER bits **CLRENA[31:0]**
  - Write
    - **CLRENA[x] = 0:** no effect
    - **CLRENA[x] = 1:** disable IRQ**x** interrupt
  - Read
    - **CLRENA[x] = 0/1:** IRQ**x** interrupt is disabled/enabled

Registers ISPR and ICPR

- ISPR bits **SETPEND[31:0]**
  - Write
    - **SETPEND[x] = 0:** no effect
    - **SETPEND[x] = 1:** enter the interrupt pending state for IRQ**x**
      - ✓ IRQ**x** interrupt becomes pending (even if disabled)
  - Read
    - **SETPEND[x] = 0/1:** IRQ**x** interrupt is not/is pending
- ICPR bits **CLRPEND[31:0]**
  - Write
    - **CLRPEND[x] = 0:** no effect
    - **CLRPEND[x] = 1:** exit the interrupt pending state for IRQ**x**
      - ✓ IRQ**x** interrupt is no longer pending
  - Read
    - **CLRPEND[x] = 0/1:** IRQ**x** interrupt is not/is pending

Register IPR0, IPR1, …, IPR7

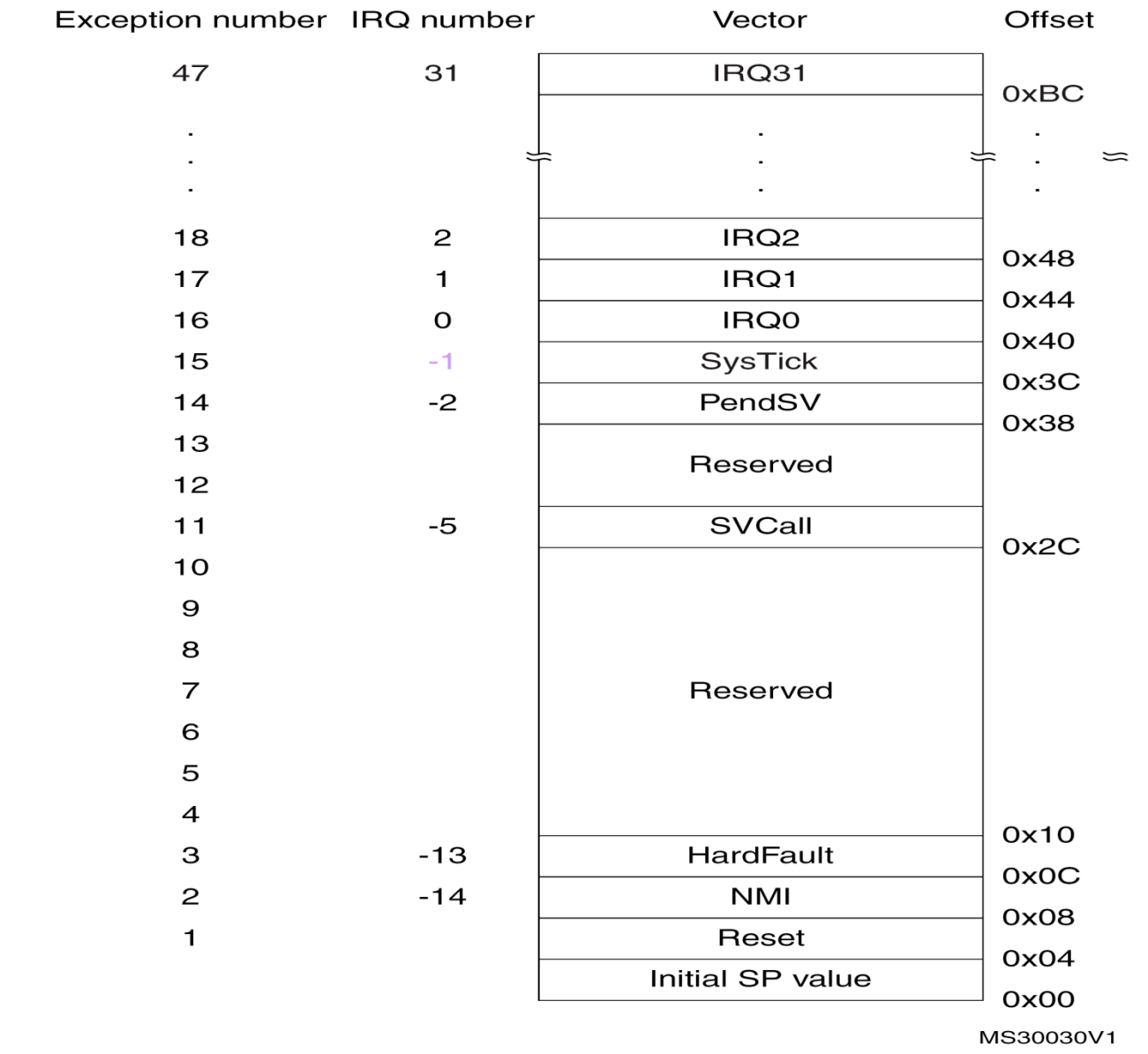


- Four 8-bit sections per IPR*n* register: IP[0]-IP[3]
  - 32 sections in total: **PRI_0**, **PRI_1**, …, **PRI_31**
  - For each IRQ**x**, only bits **PRI_x[7:6]** are in effect
    - Bits **PRI_x[5:0]** are 0's (regardless of what we write to them)
    - Bits **PRI_x[7:6]** = one of 4 priority levels: 0, 64, 128, 192

Exception States and Types

- Exception states
  - **Active:** the exception is being handled by the processor
    - If the current exception handler is interrupted by another exception, both exceptions are in the active state
  - **Pending:** the exception is waiting to be handled by the processor
  - **Active and Pending:** the exception is being handled by the processor, and there is a pending exception from the same source
  - **Inactive:** the exception is neither active, nor pending
- Exception types
  - System exceptions: Reset, SysTick, calls to OS, etc.
  - Interrupts: IRQ**0**, IRQ**1**, …, IRQ**31**

Vector Table



Exception Processing

- Exception entry
  - The processor enters the handler for a pending exception when
    - The processor is in the **Thread** mode (**IPSR[5:0] = 0**), or
    - The processor is the **Handler** mode (**IPSR[5:0] ≠ 0**) handling some other exception of lower priority
  - Once the processor has entered the exception handler, the exception state is changed from **pending** to **active**
    - If another exception (from the same source) becomes pending, the exception state changes to **active and pending**
- Exception return
  - When the processor returns from the exception handler, the exception state is changed to **inactive** (if **active**) or to **pending** (if **active and pending**)

Pending Interrupts

- IRQ**x** interrupt <u>becomes pending when</u>:
  - NVIC detects an interrupt signal from the interrupt source, which sets **SETPEND[x]** in ISPR
    - If the IRQ**x** interrupt state was **active** (i.e., the IRQ**x** handler is already running), it changes to **active and pending**
  - Or: software writes **SETPEND[x] = 1** to ISPR
- IRQ**x** interrupt <u>remains pending until</u>:
  - The processor enters the IRQ**x** handler (provided that **SETENA[x] = 1** in ISER)
    - The IRQ**x** interrupt state changes to **active**, which automatically clears **SETPEND[x]**
  - Or: software writes **CLRPEND[x] = 1** to ICPR
    - The IRQ**x** interrupt state changes to **inactive** (if **pending**) or to **active** (if **active and pending**)

CMSIS Support I

- What is **CMSIS**?
  - **C**ortex **M**icrocontroller **S**oftware **I**nterface **S**tandard
    - Part of the driver library to ease software development
  - Device-independent interface for OS kernels
    - Named definitions of peripheral registers/bits, exception vectors
    - Access functions to peripheral and internal registers
- Accessing processor's PRIMASK using CMSIS:
  - **`void __disable_irq(void)`** sets **PRIMASK[0]**
  - **`void __enable_irq(void)`** clears **PRIMASK[0]**
  - **`uint32_t __get_PRIMASK(void)`** reads PRIMASK
  - **`void __set_PRIMASK(uint32_t value)`** writes **`value`** to PRIMASK

CMSIS Support II

- Accessing NVIC using CMSIS:

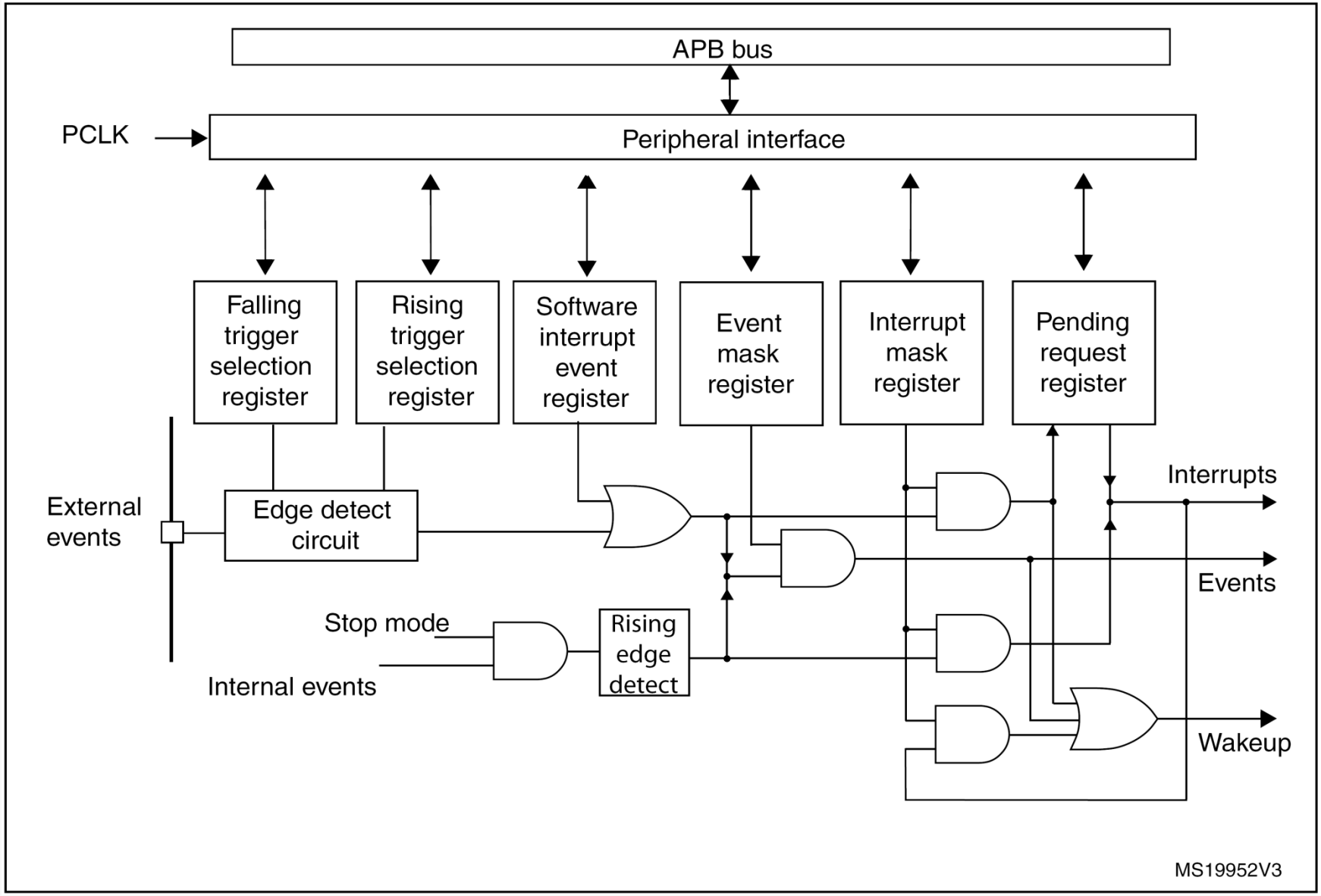
CMSIS interrupt control function	Description
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true (1) if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

Note: `IRQn_t` corresponds to `typedef enum { ... } IRQn_Type` that we will see later.

External Events/Interrupts

- Extended interrupts and events controller (**EXTI**)
  - Up to 23 external + 9 internal event/interrupt lines
  - External event/interrupt lines
    - Configurable detection: rising/falling/either edge
    - Detection status flag (pending bit) recorded for each line
    - **16** out of 23 lines connect to selectable **GPIO** pins
  - Internal events/interrupts
    - Fixed detection: rising edge
    - No detection status flags recorded in **EXTI**
  - Interrupt generation
    - Must unmask interrupt requests
  - Event (pulse) generation
    - Must unmask event requests

EXTI Block Diagram



Some of **EXTI** Registers

- Interrupt mask register (EXTI_IMR): bits **MR[31:0]**
  - **MR[x] = 0/1:** interrupt request from line **x** is/is not masked
- Pending register (EXTI_PR): bits **PR[22:0]**
  - **PR[x] = 0/1:** triggering edge has not/has been detected on line **x**
  - To clear **PR[x]**, software must write **1** to it
- Rising and falling trigger selection registers (EXTI_RTSR, EXTI_FTSR): bits **TR[22:0]**
  - EXTI_RTSR – **TR[x] = 0/1:** rising-edge trigger for line **x** is disabled/enabled
  - EXTI_FTSR – **TR[x] = 0/1:** falling-edge trigger for line **x** is disabled/enabled

GPIO Module

- 55 external I/O pins are mapped onto 5 ports
  - **PA[15:0]**, **PB[15:0]**, **PC[15:0]** – 48 pins
  - **PF[7:4]**, **PF[1:0]** – 6 pins
  - **PD[2]** – 1 pin
- Selectable GPIO pins connect to EXTI lines **0-15**
  - System configuration controller (**SYSCFG**) contains four external interrupt configuration registers
    - SYSCFG_EXTICR1: four **EXTIx[3:0]** fields, **x = 0, 1, 2, 3**
    - SYSCFG_EXTICR2: four **EXTIx[3:0]** fields, **x = 4, 5, 6, 7**
    - SYSCFG_EXTICR3: four **EXTIx[3:0]** fields, **x = 8, 9, 10, 11**
    - SYSCFG_EXTICR4: four **EXTIx[3:0]** fields, **x = 12, 13, 14, 15**
  - Each **EXTIx[3:0]** indicates which port's pin **x** becomes external event/interrupt line **x**

Example: SYSCFG_EXTICR1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration bits (x = 0 to 3)

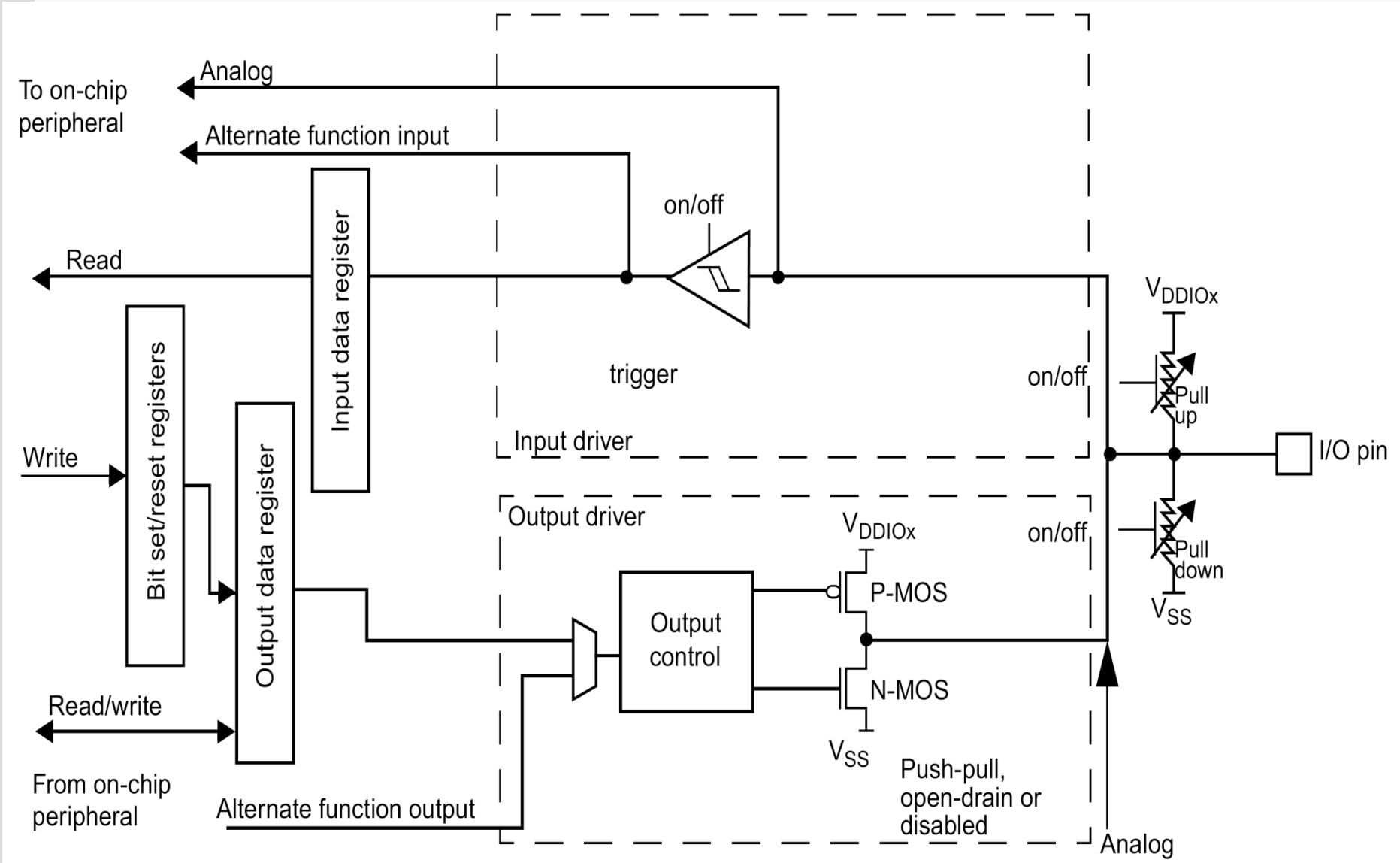
These bits are written by software to select the source input for the EXTIx external interrupt.

x000: PA[x] pin
x001: PB[x] pin
x010: PC[x] pin
x011: PD[x] pin
x100: PE[x] pin ⟵ Not applicable
x101: PF[x] pin
other configurations: reserved

GPIO Clocking

- Internal AHB2 bus provides a clock signal for GPIO ports, but its use must be enabled
  - Reset and clock control (**RCC**) module contains AHB peripheral clock enable register (RCC_AHBENR) that includes 5 clock-enable bits for 5 GPIO ports
    - **RCC_AHBENR[17] = 0/1:** PA clock is disabled/enabled
    - **RCC_AHBENR[18] = 0/1:** PB clock is disabled/enabled
    - **RCC_AHBENR[19] = 0/1:** PC clock is disabled/enabled
    - **RCC_AHBENR[20] = 0/1:** PD clock is disabled/enabled
    - **RCC_AHBENR[22] = 0/1:** PF clock is disabled/enabled

I/O Port Structure



GPIO Registers I

- GPIO port mode register
  - GPIOx_MODER, x = A, B, …F
  - 16 fields **MODERy[1:0]**, **y = 0, 1, …, 15**
    - **00:** input mode
    - **01:** general purpose output mode
    - **10:** alternate function (AF) mode
    - **11:** analog mode
- GPIO port output speed register
  - GPIOx_OSPEEDR, x = A, B, …F
  - 16 fields **OSPEEDRy[1:0]**, **y = 0, 1, …, 15**
    - **00/10:** low speed
    - **01:** medium speed
    - **11:** high speed

GPIO Registers II

- GPIO port output type register
  - GPIOx_TYPER, x = A, B, …F
  - 16 bits **OTy**, **y = 0, 1, …, 15**
    - **0:** push-pull output
    - **1:** open-drain output
- GPIO port pull-up/pull-down register
  - GPIOx_PUPDR, x = A, B, …F
  - 16 fields **PUPDRy[1:0]**, **y = 0, 1, …, 15**
    - **00:** no pull-up/pull-down
    - **01:** pull-up
    - **10:** pull-down
    - **11:** (reserved)

GPIO Registers III

- GPIO port input data register (**read-only**)
  - GPIOx_IDR, x = A, B, …F
  - 16 bits **IDRy**, **y = 0, 1, …, 15**
- GPIO port output data register
  - GPIOx_ODR, x = A, B, …F
  - 16 bits **ODRy**, **y = 0, 1, …, 15**
  - Individual **ODRy** bits can be set/reset **atomically** using special registers GPIOx_BSRR and GPIOx_BRR

GPIO Registers IV

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
GPIOx_IDR (where x = **A..F**)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
GPIOx_ODR (where x = **A..F**)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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
GPIOx_BRR (where x = **A..F**)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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
GPIOx_BSRR (where x = **A..F**)	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
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	0	0	0	0	0	0	0	0

GPIO Registers V

- GPIO port bit reset register (**write-only**)
  - GPIOx_BRR, x = A, B, …F
  - 16 bits **BRy**, **y = 0, 1, …, 15**
    - Writing **0** to **BRy** has no effect on **ODRy**
    - Writing **1** to **BRy** makes **ODRy = 0**
- GPIO port bit set/reset register (**write-only**)
  - GPIOx_BSRR, x = A, B, …F
  - Low half [15:0]: 16 bits **BSy**, **y = 0, 1, …, 15**
    - Writing **0** to **BSy** has no effect on **ODRy**
    - Writing **1** to **BSy** makes **ODRy = 1**
  - High half [31:16]: 16 bits **BRy**, **y = 0, 1, …, 15**
    - Writing **0** to **BRy** has no effect on **ODRy**
    - Writing **1** to **BRy** makes **ODRy = 0** (unless **BSy = 1**)

GPIO Registers VI

- If **MODERy[1:0] = 10** in GPIOx_MODER, pin **y** of port x is configured as an alternate function I/O
  - Alternate function (AF) specifications and numbering are provided in the device datasheets
- GPIO AF low register (GPIOx_AFRL)
  - Eight **AFRy[3:0]** fields, pins **y = 0, 1, …, 7**
- GPIO AF high register (GPIOx_AFRH)
  - Eight **AFRy[3:0]** fields, pins **y = 8, 9, …, 15**
- **AFRy[3:0] = 0000/0001/…/0111** → AF 0/1/…/7

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
GPIOx_AFRL (where x = **A..E**)	AFR7[3:0]				AFR6[3:0]				AFR5[3:0]				AFR4[3:0]				AFR3[3:0]				AFR2[3:0]				AFR1[3:0]				AFR0[3:0]			
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	0	0	0	0	0	0	0	0
GPIOx_AFRH (where x = **A..E**)	AFR15[3:0]				AFR14[3:0]				AFR13[3:0]				AFR12[3:0]				AFR11[3:0]				AFR10[3:0]				AFR9[3:0]				AFR8[3:0]			
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	0	0	0	0	0	0	0	0

Example: **PB[15:0]** AFs

	Pin name	AF0	AF1	AF2	AF3
AFRL	PB0	EVENTOUT	TIM3_CH3	TIM1_CH2N	TSC_G3_IO2
	PB1	TIM14_CH1	TIM3_CH4	TIM1_CH3N	TSC_G3_IO3
	PB2				TSC_G3_IO4
	PB3	SPI1_SCK, I2S1_CK	EVENTOUT	TIM2_CH2	TSC_G5_IO1
	PB4	SPI1_MISO, I2S1_MCK	TIM3_CH1	EVENTOUT	TSC_G5_IO2
	PB5	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM16_BKIN	I2C1_SMBA
	PB6	USART1_TX	I2C1_SCL	TIM16_CH1N	TSC_G5_IO3
	PB7	USART1_RX	I2C1_SDA	TIM17_CH1N	TSC_G5_IO4
AFRH	PB8	CEC	I2C1_SCL	TIM16_CH1	TSC_SYNC
	PB9	IR_OUT	I2C1_SDA	TIM17_CH1	EVENTOUT
	PB10	CEC	I2C2_SCL	TIM2_CH3	TSC_SYNC
	PB11	EVENTOUT	I2C2_SDA	TIM2_CH4	TSC_G6_IO1
	PB12	SPI2_NSS	EVENTOUT	TIM1_BKIN	TSC_G6_IO2
	PB13	SPI2_SCK		TIM1_CH1N	TSC_G6_IO3
	PB14	SPI2_MISO	TIM15_CH1	TIM1_CH2N	TSC_G6_IO4
	PB15	SPI2_MOSI	TIM15_CH2	TIM1_CH3N	TIM15_CH1N

STM32F051xx Timers

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary outputs
Advanced control	TIM1	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	Yes
General purpose	TIM2	32-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No
	TIM3	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No
	TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No
	TIM15	16-bit	Up	Any integer between 1 and 65536	Yes	2	Yes
	TIM16, TIM17	16-bit	Up	Any integer between 1 and 65536	Yes	1	Yes
Basic	TIM6	16-bit	Up	Any integer between 1 and 65536	Yes	0	No

TIM2 Timer

- Internal APB bus provides a clock signal for TIM2, but its use must be enabled
  - Reset and clock control (**RCC**) module contains APB peripheral clock enable register 1 (RCC_APB1ENR) that includes a clock-enable bit for TIM2
    - **RCC_APB1ENR[0] = 0/1:** TIM2 clock is disabled/enabled
- Input clock (CK_PSC) frequency is divided by a programmable prescaler value (ranging from 1 to $2^{16} - 1 = 65{,}535$) to produce timer clock CK_CNT
- TIM2 can generate interrupts on update events (initialization, overflow/underflow), trigger events, input capture events, and output compare events

Some of **TIM2** Registers I

- TIM2 counter register (TIM2_CNT)
  - Bits **CNT[31:0]**
  - Incremented/decremented every clock cycle (if enabled)
- TIM2 auto-reload register (TIM2_ARR)
  - Bits **ARR[31:0]**
  - Update event → **CNT[31:0]** reloaded with **ARR[31:0]**
- TIM2 event generation register (TIM2_EGR)
  - Bit **[0]: UG** (update generation)
    - **UG = 0:** no action
    - **UG = 1:** re-initialize TIM2 and generate update of its registers
    - Set by software (write-only), cleared by hardware
  - Other bits are for trigger/capture/compare events

Some of **TIM2** Registers II

- TIM2 status register (TIM2_SR)
  - Bit **[0]: UIF** (update interrupt flag)
    - **UIF = 0/1:** update event has not/has occurred
    - Set by hardware, cleared by software
  - Other bits are for trigger/capture/compare events
- TIM2 DMA/interrupt enable register (TIM2_DIER)
  - Bit **[0]: UIE** (update interrupt enable)
    - **UIE = 0/1:** update interrupts are disabled/enabled
  - Other bits are for trigger/capture/compare interrupts
- TIM2 prescaler register (TIM2_PSC)
  - Bits **PSC[15:0]**
  - CK_CNT = CK_PSC / (**PSC[15:0]** + 1)

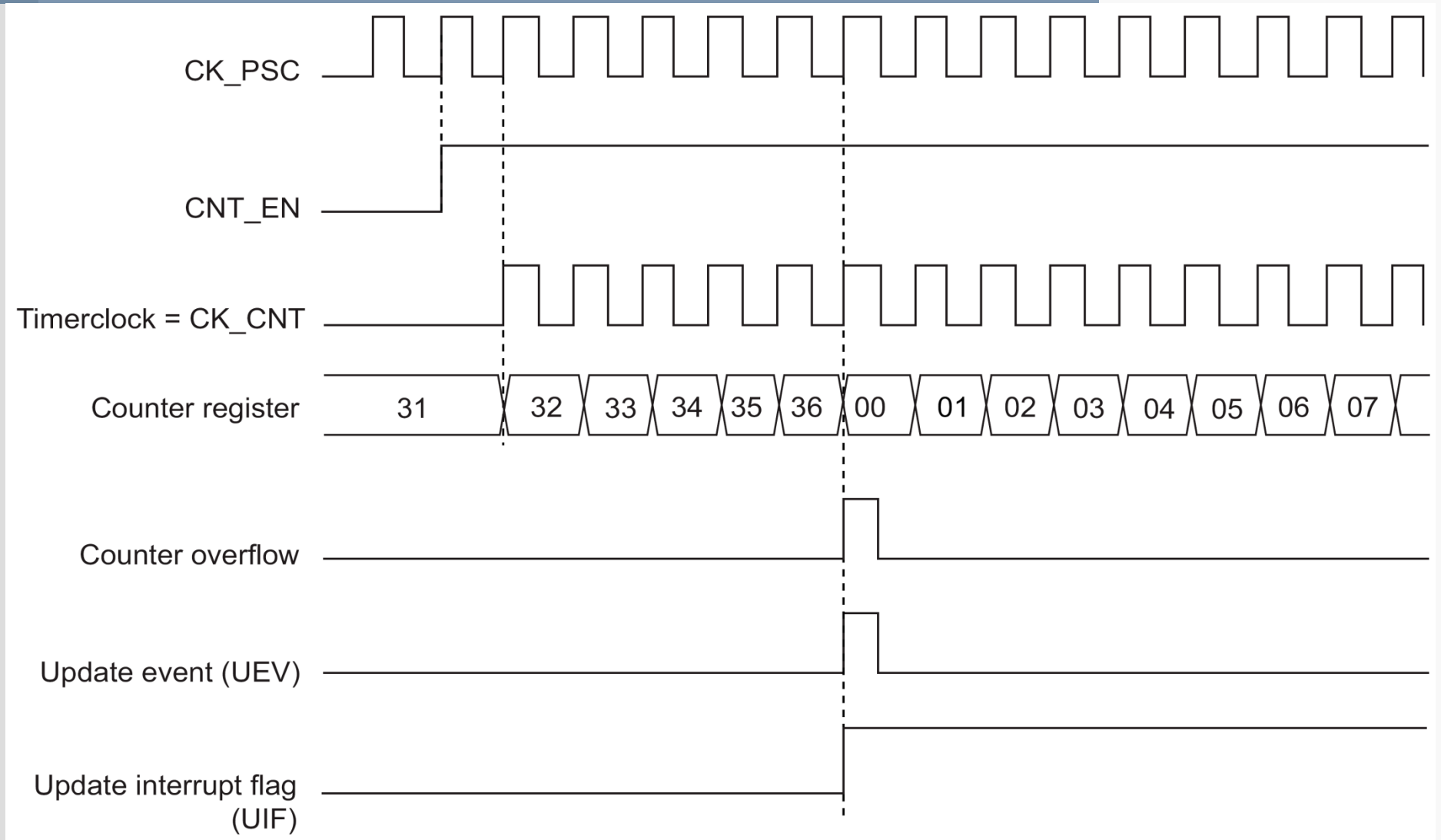
Some of **TIM2** Registers III

- TIM2 control register 1 (TIM2_CR1)
  - Bit **[0]: CEN** (counter enable)
    - **CEN = 0/1:** counter is disabled/enabled
  - Bit **[1]: UDIS** (update disable)
    - **UDIS = 0/1:** update event requests are enabled/disabled
  - Bit **[2]: URS** (update request source)
    - **URS = 1:** update event = counter overflow/underflow only
    - **URS = 0:** update event = counter overflow/underflow, or setting **UG** in TIM2_EGR (by software), or slave-mode update
  - Bit **[3]: OPM** (one-pulse mode)
    - **OPM = 0:** counter does not stop at the next update event
    - **OPM = 1:** counter stops counting at the next update event, automatically clearing **CEN**

Some of **TIM2** Registers IV

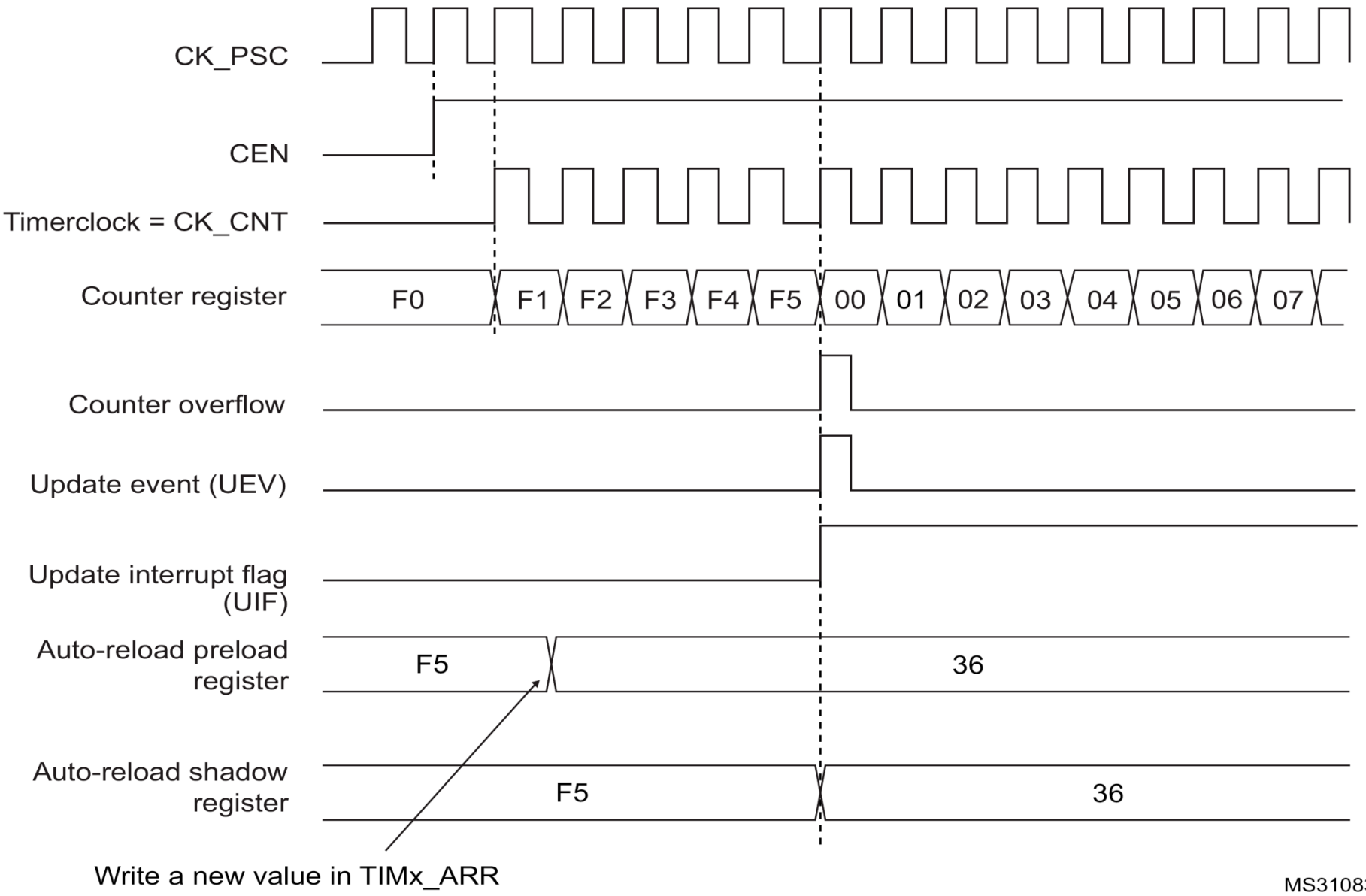
- TIM2 control register 1 (TIM2_CR1) – continued
  - Bit **[4]: DIR** (direction)
    - **DIR = 0/1:** counter is to be counting up/down
  - Bits **[6:5]: CMS** (center-aligned mode selection)
    - **CMS = 00:** counter counts up or down, depending on **DIR** (edge-aligned mode)
    - **CMS = 01/10/11:** counter counts up and down alternatively (center-aligned modes 1, 2, 3)
  - Bit **[7]: ARPE** (auto-reload preload enable)
    - **ARPE = 0/1:** TIM2_ARR is not/is buffered
  - Bit **[9:8]: CKD** (clock division)
    - **CKD = 00/01/10:**
      input sampling frequency = (internal clock CK_INT) / 2^{CKD}
    - **CKD = 11:** (reserved)

Example: Upcounting

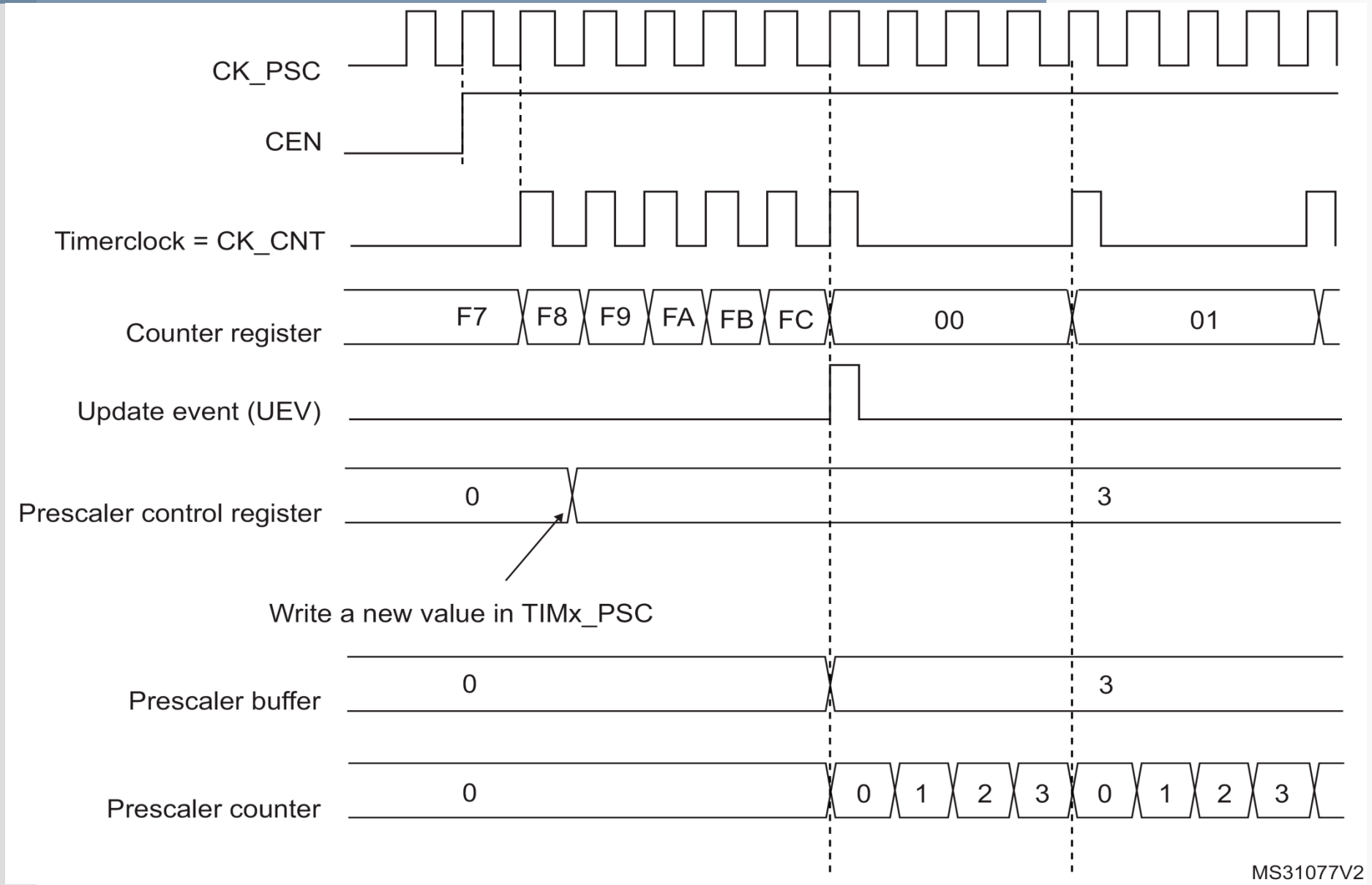


CNT[31:0] = 0x31 **ARR[31:0] = 0x36** **PSC[15:0] = 0x0**

Example: Buffered Auto-Reload



Example: Prescaler Change



Programing Example

- **STM32F0 Discovery** board
  - USER button – I/O pin **PA0** (input)
    - Relevant I/O registers: GPIOA_MODER, GPIOA_PUPDR, GPIOA_IDR
  - Blue and green LEDs – I/O pins **PC8** and **PC9** (output)
    - Relevant I/O registers: GPIOC_MODER, GPIOC_PUPDR, GPIOC_OTYPER, GPIOC_OSPEEDR, GPIOC_ODR, GPIOC_BSRR, GPIOC_BRR
- Application: blinking LED (either blue, or green)
  - Pressing the USER button switches the blinking LED
    - Blinking period = 1/2 s: LED is on for 1/4 s, and then off for 1/4 s
  - **TIM2** is to generate an interrupt every **1/4 s**
    - Relevant timer registers: TIM2_CR1, TIM2_PSC, TIM2_ARR, TIM2_EGR, TIM2_DIER, TIM2_SR

Accessing Relevant Registers I

- Inside our C program: `#include "cmsis/cmsis_device.h"`
- Inside **`cmsis_device.h`**: `#include "stm32f0xx.h"`
- Inside **`stm32f0xx.h`**: `#include "stm32f051x8.h"`

- Inside **`stm32f051x8.h`** (key reference file):

```
...
#define PERIPH_BASE  0x40000000UL
...
#define APBPERIPH_BASE  PERIPH_BASE
...
#define TIM2_BASE  (APBPERIPH_BASE + 0x00000000UL)
...
#define TIM2  ((TIM_TypeDef *) TIM2_BASE)
...
```

UL: unsigned long int, same as **uint32_t** (32-bit positive)

Accessing Relevant Registers II

- E.g., inside **stm32f051x8.h** (continued):

```
...
typedef struct {
  __IO uint32_t CR1;     /* ... Address offset: 0x00 */
  ...
  __IO uint32_t SR;      /* ... Address offset: 0x10 */
  ...
  __IO uint32_t CNT;     /* ... Address offset: 0x24 */
  ...
} TIM_TypeDef;
...
```

- E.g., inside our C program: **TIM2->CR1 = ((uint16_t)0x008C);**

0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0

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	TIM2_CR1 and TIM3_CR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CKD [1:0]		ARPE	CMS [1:0]		DIR	OPM	URS	UDIS	CEN
	Reset value																							0	0	0	0	0	0	0	0	0	0

Initializing GPIOA I

```
void myGPIOA_Init() {
  /* Enable clock for GPIOA peripheral */
  RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
  /* Configure PA0 as input */
  GPIOA->MODER &= ~(GPIO_MODER_MODER0);
  /* Ensure no pull-up/pull-down for PA0 */
  GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR0);
}
```

```
r &= m: same as r = r & m (bitwise AND)
r |= m: same as r = r | m (bitwise OR)
r ^= m: same as r = r ^ m (bitwise XOR)
```

m	r	r & m
0	0	0
0	1	0
1	0	0 (r)
1	1	1 (r)

m	r	r | m
0	0	0 (r)
0	1	1 (r)
1	0	1
1	1	1

m	r	r ^ m
0	0	0 (r)
0	1	1 (r)
1	0	1 (~r)
1	1	0 (~r)

Initializing GPIOA II

- From **stm32f051x8.h**:

```
#define RCC_AHBENR_GPIOAEN_Pos   (17U)

#define RCC_AHBENR_GPIOAEN_Msk
  (0x1UL << RCC_AHBENR_GPIOAEN_Pos) /*!< 0x00020000 */

#define RCC_AHBENR_GPIOAEN  RCC_AHBENR_GPIOAEN_Msk
```

U: unsigned int, same as **uint32_t** (32-bit positive)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSCEN	Res.	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.
							rw		rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CRC EN	Res.	FLITF EN	Res.	SRAM EN	Res.	DMA EN
									rw		rw		rw		rw

```
RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
```

Initializing GPIOA III

- From **`stm32f051x8.h`**:

```
#define GPIO_MODER_MODER0_Pos   (0U)

#define GPIO_MODER_MODER0_Msk
  (0x3UL << GPIO_MODER_MODER0_Pos) /*!< 0x00000003 */

#define GPIO_MODER_MODER0  GPIO_MODER_MODER0_Msk
```

~(0x00000003) = 11111111...111111 0 0

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	**GPIOx_MODER** **x = A**	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
	Reset value	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

`GPIOA->MODER &= ~(GPIO_MODER_MODER0)`

Initializing GPIOC I

```
void myGPIOC_Init() {
  /* Enable clock for GPIOC peripheral */
  RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
  /* Configure PC8, PC9 as outputs:
     assuming that Port C's MODER8 = MODER9 = 00 */
  GPIOC->MODER |= (GPIO_MODER_MODER8_0 |
                   GPIO_MODER_MODER9_0);
  /* Ensure no pull-up/pull-down for PC8, PC9 */
  GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPDR8 |
                   GPIO_PUPDR_PUPDR9);
  /* Ensure push-pull mode for PC8, PC9 */
  GPIOC->OTYPER &= ~(GPIO_OTYPER_OT_8 |
                   GPIO_OTYPER_OT_9);
  /* Ensure high-speed mode for PC8, PC9 */
  GPIOC->OSPEEDR |= (GPIO_OSPEEDER_OSPEEDR8 |
                   GPIO_OSPEEDER_OSPEEDR9); }
```

Initializing GPIOC II

- From **stm32f051x8.h**:

```
#define GPIO_MODER_MODER8_Pos   (16U)
#define GPIO_MODER_MODER8_0
   (0x1UL << GPIO_MODER_MODER8_Pos) /*!< 0x00010000 */

#define GPIO_MODER_MODER9_Pos   (18U)
#define GPIO_MODER_MODER9_0
   (0x1UL << GPIO_MODER_MODER9_Pos) /*!< 0x00040000 */
```

(0x00010000 | 0x00040000) = 0000... 0 1 0 1 ...0000

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	GPIOx_MODER x = C	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
	Reset value	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
GPIOC->MODER |= (GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0)
```

Initializing TIM2 I

```
void myTIM2_Init() {
  /* Enable clock for TIM2 peripheral */
  RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

  /* Configure TIM2:
     buffer auto-reload, count up, stop on overflow,
     enable UEV, interrupt on overflow only */
  TIM2->CR1 = ((uint16_t)0x008C);

  /* Set clock prescaler value */
  TIM2->PSC = ((uint16_t)0x0000);

  /* Set auto-reloaded delay: 1/4 s at 48 MHz*/
  TIM2->ARR = ((uint32_t)12000000);
```

Initializing TIM2 II

```
  /* Update timer registers */
  TIM2->EGR = ((uint16_t)0x0001);

  /* Assign TIM2 interrupt priority = 0 in NVIC */
  NVIC_SetPriority(TIM2_IRQn, 0);

  /* Enable TIM2 interrupts in NVIC */
  NVIC_EnableIRQ(TIM2_IRQn);

  /* Enable update interrupt generation */
  TIM2->DIER |= TIM_DIER_UIE;

  /* Start counting timer pulses */
  TIM2->CR1 |= TIM_CR1_CEN;
}
```

STM32F0 IRQ Numbers

- From **`stm32f051x8.h`**:

```
typedef enum {
  ...
  EXTI0_1_IRQn = 5,
  EXTI2_3_IRQn = 6,
  ...
  TIM2_IRQn = 15,
  TIM3_IRQn = 16,
  ...
  SPI1_IRQn = 25,
  ...
} IRQn_Type;
```

← **IRQ5** shared by **EXTI** lines **0** and **1** interrupts

← **IRQ6** shared by **EXTI** lines **2** and **3** interrupts

← **IRQ15** used for **TIM2** global interrupts

← **IRQ16** used for **TIM3** global interrupts

← **IRQ25** used for **SPI1** global interrupts

Main Program

```
/* Global variable indicating which LED is blinking */
volatile uint16_t blinkingLED = ((uint16_t)0x0100);

int main (int argc, char* argv[]) {
  ...
  myGPIOA_Init();          /* Initialize I/O port PA */
  myGPIOC_Init();          /* Initialize I/O port PC */
  myTIM2_Init();           /* Initialize timer TIM2 */

  while (1) {
    - See next slide...
  }

  return 0;
}
```

Inside **while (1)**

GPIO_IDR_0 = 0x00000001

```
/* Check if button is pressed (PA0 = 1) */
if((GPIOA->IDR & GPIO_IDR_0) != 0) {

  /* Wait for button to be released (PA0 = 0) */
  while((GPIOA->IDR & GPIO_IDR_0) != 0){}

  /* Turn off currently blinking LED */
  GPIOC->BRR = blinkingLED;
  /* Switch blinking LED */
  blinkingLED ^= ((uint16_t)0x0300);
  /* Turn on switched LED */
  GPIOC->BSRR = blinkingLED;

  trace_printf("Switching the blinking LED...\n");
}
```

IRQ Handler for TIM2

TIM_SR_UIF = 0x00000001

```
void TIM2_IRQHandler() {
  uint16_t LEDstate;
  /* Check if update interrupt flag is indeed set */
  if ((TIM2->SR & TIM_SR_UIF) != 0) {
    /* Read current PC and isolate PC8 and PC9 bits */
    LEDstate = GPIOC->ODR & ((uint16_t)0x0300);
    if (LEDstate == 0) {              /* If LED is off… */
      GPIOC->BSRR = blinkingLED;       /* Set PC8/PC9 */
    } else {                        /* Else (LED's on)… */
      GPIOC->BRR = blinkingLED;      /* Reset PC8/PC9 */
    }
    TIM2->SR &= ~(TIM_SR_UIF);         /* Clear UIF */
    TIM2->CR1 |= TIM_CR1_CEN;          /* Restart TIM2 */
  }
}
```