Programming Project (3)

Yesterday, we looked at:

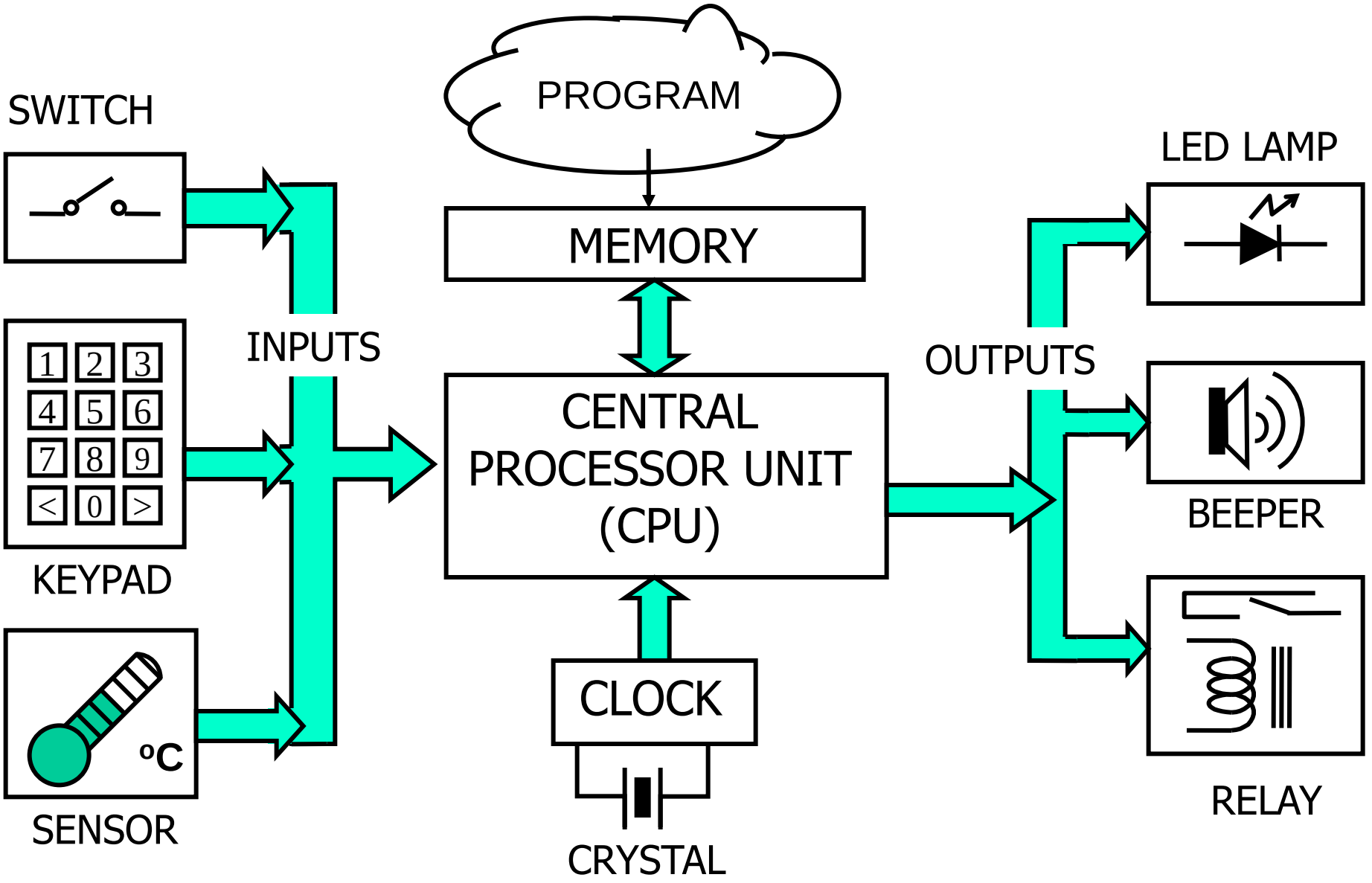
- Control Flow (cont.): if, switch, while, do, for, break & continue, goto
- Range & precedence of operators
- Program structure in C
- SW Architecture & Documentation
- Project Schedule & Block Diagram
- Tables & Arrays. Arrays & Pointers
- Pointers and Function Arguments
- Structures
- Typical errors in C
- Exercise 3: Fixed Point Arithmetic
- Ex. 4: Ball bouncing between walls
- Bit Manipulation Exercise

Today, we will look at:

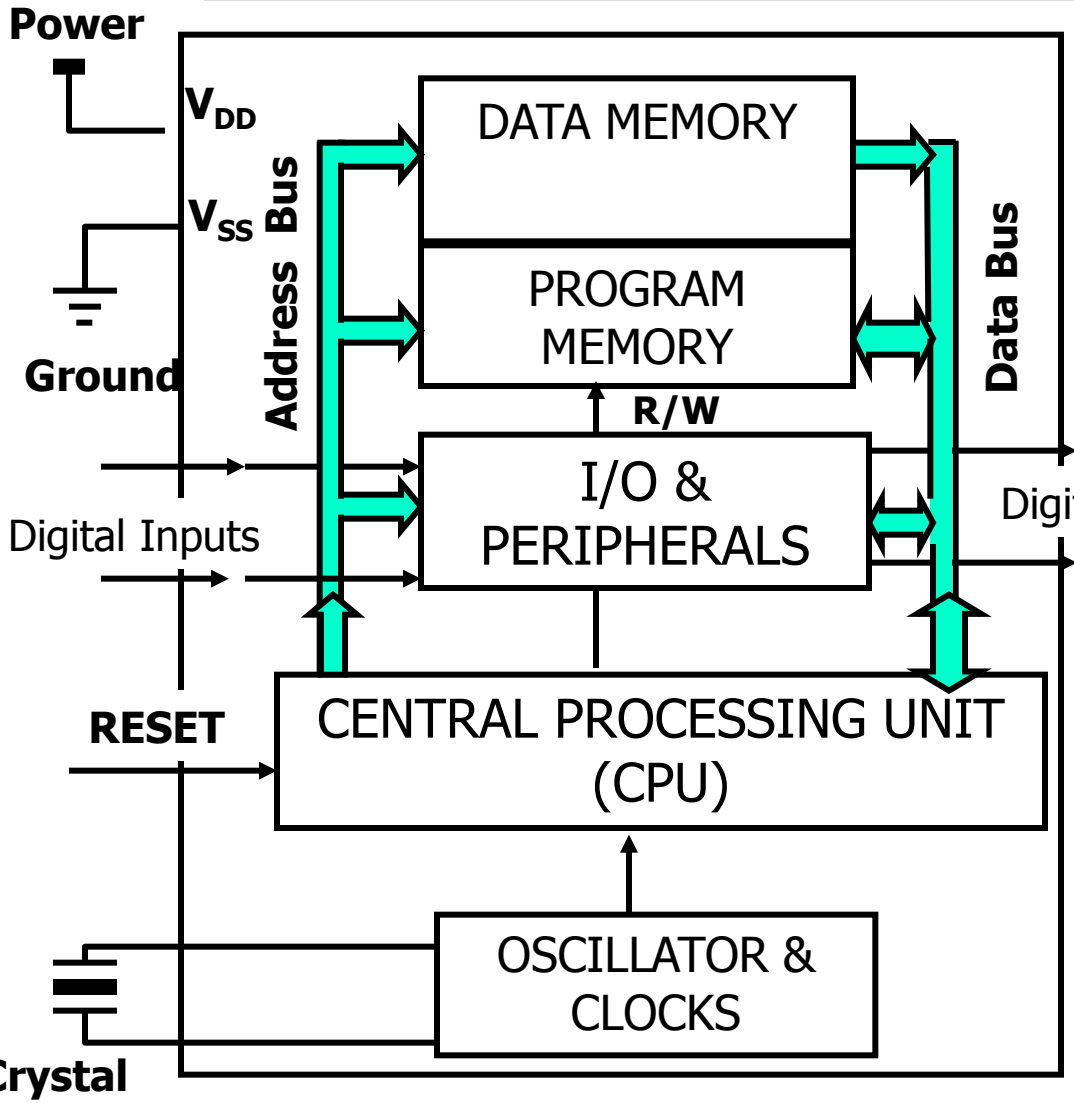
- Computer Systems and MCU
- Central Processor Unit (CPU)
- Memory in a computer
- Computer Number Interpretation
- Conversion: Binary-Hex-Decimal
- ARM Cortex MCU: Block Diagram and General-Purpose I/O (GPIO)
- Relation between Register & Ports
- Configuration of GPIOs

- Exercise 5: Read & Write from I/Os
- Exercise 6: Stop watch using Timer

Computer Systems



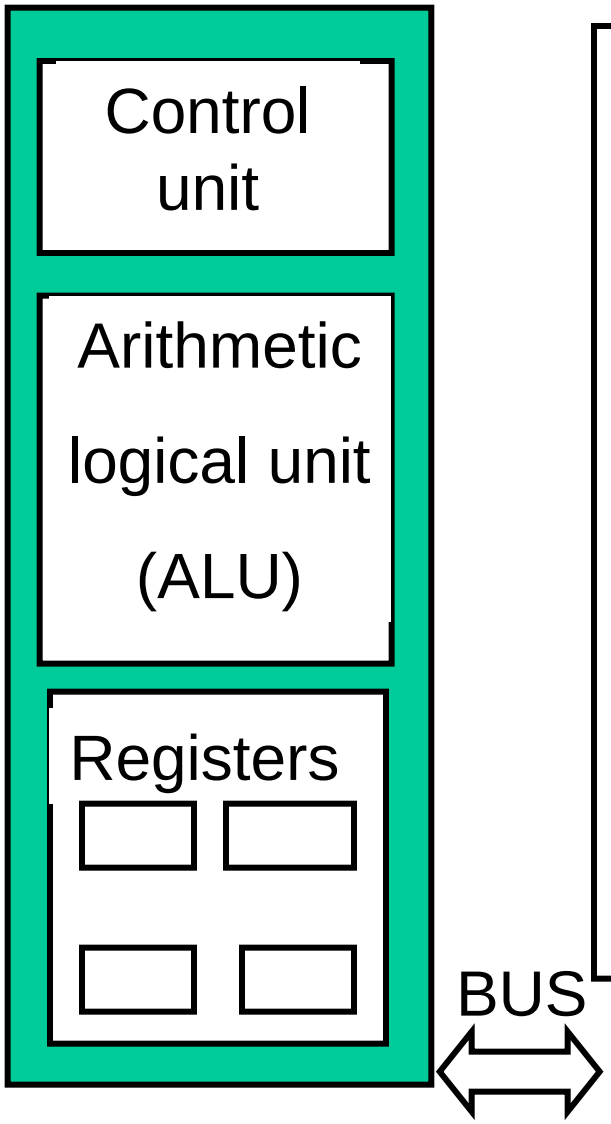
Microcontroller unit (MCU)



A microcontroller is a complete computer system on a chip.

Often the microcontrollers are embedded in a product (e.g. washing machine)

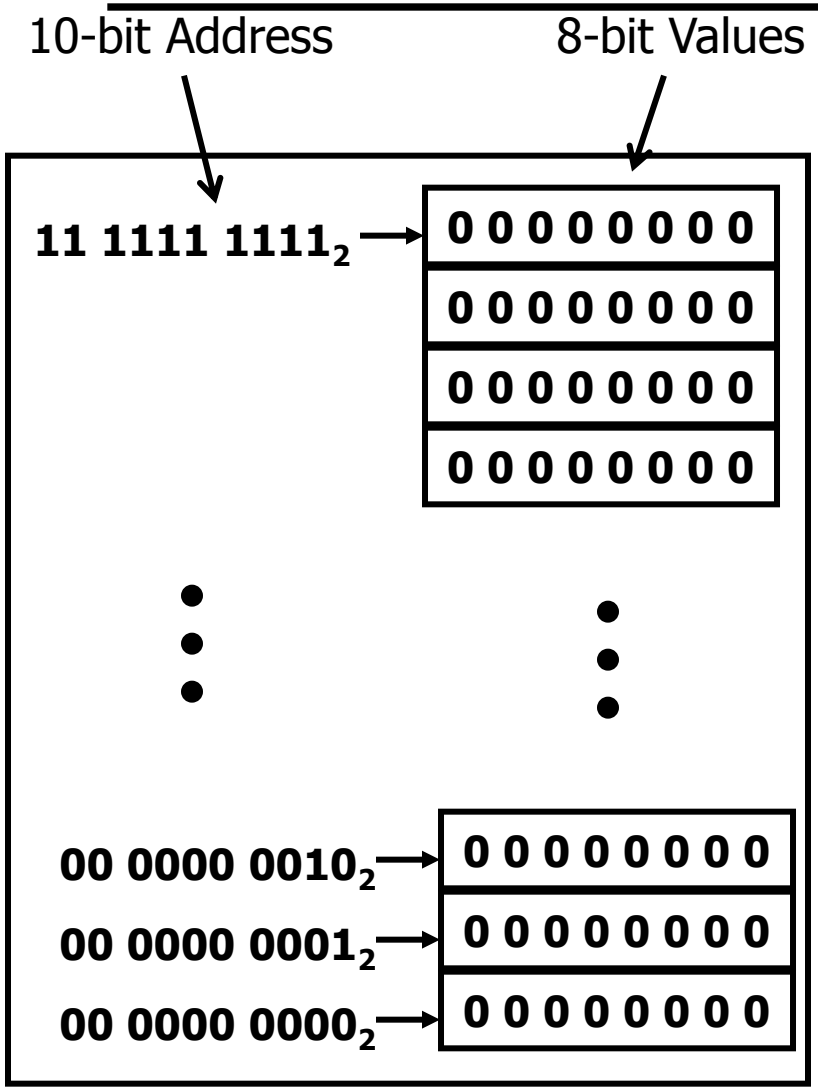
Central Processor Unit (CPU)



CPU executes programs that are stored on the memory.

- **Control unit** is responsible for fetching instructions from memory and interpreting them.
- **ALU** makes operations like addition, subtraction and boolean algebra.
- **Registers** are high speed memory cells which are used to stored temporarily data og control information.

How does a Computer see Memory



- A 8-bit computer with 10 address lines see memory a continuous row of 1024 8-bit values.
- In hexadecimal notation go these address from `$0000` to `$03FF`.

In an 8-bit computer is a good idea to use `char` variables intead of `int`! Type int is either 16 or 32 bit

How a number is interpreted on a computer

Text string: '3~ÛÛÿ¿B<' is a man!!?

Ascii	Decimal	Hexadecimal	Binary	Graphical representation
'3'	51	0x33	00110011	0 0 1 1 0 0 1 1
'~'	126	0x7E	01111110	0 1 1 1 1 1 1 0
'Û'	219	0xDB	11011011	1 1 0 1 1 0 1 1
'Û'	219	0xDB	11011011	1 1 0 1 1 0 1 1
'ÿ'	255	0xFF	11111111	1 1 1 1 1 1 1 1
'¿'	191	0xBF	10111111	1 0 1 1 1 1 1 1
'B'	66	0x42	01000010	0 1 0 0 0 0 1 0
'<'	60	0x3C	00111100	0 0 1 1 1 1 0 0

One number in a computer is what the user interprets it as

Conversion between number systems

Binary to Hexadecimal (8 bit)

```
1001.1101   Split the number in 2x4 bits
8   1 8 4 1
```
1001 **~ 1x8+0x4+0x2+1x1 = 9**
1101 **~ 1x8+1x4+0x2+1x1 = 13 ~ D**

~ 0x9D hexadecimal

Decimal to Hexadecimal

```
09 ~ 0x09
10 ~ 0x0A
15 ~ 0x0F
16 ~ 0x10
31 ~ 0x1F
32 ~ 0x20
```

A_{dec} ~ $A_{dec}/16$ or $A_{dec}\%16$

$A_{dec}=212$ ~ 212/16 212%16 = 13 4 = 0xD4

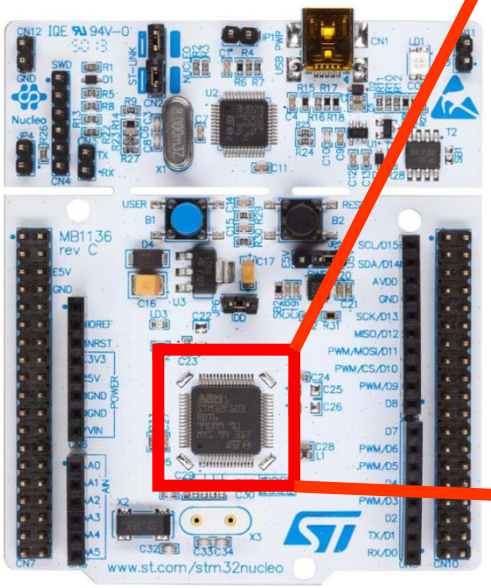
Integer division Remainder

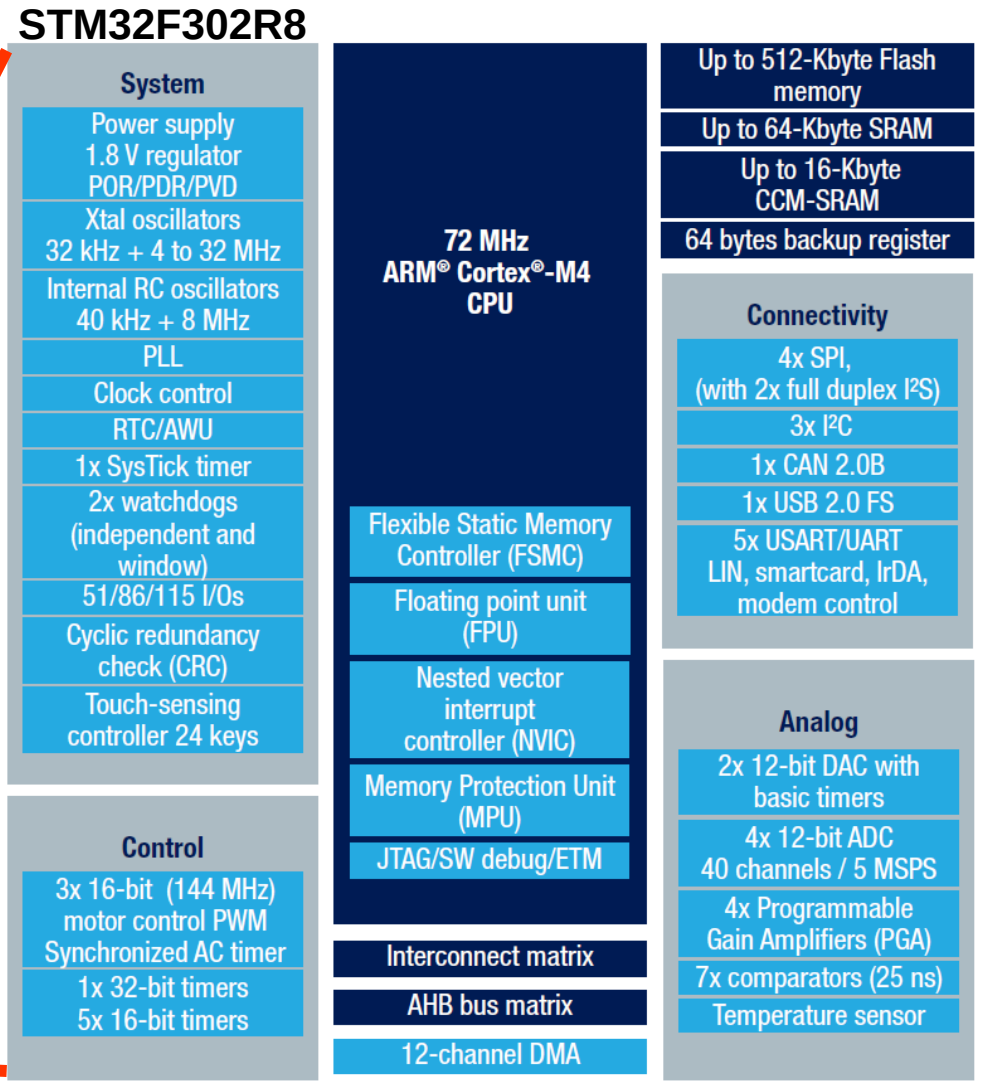
ASCII characters

```
'A','B','C',...
'a','b','c',...
'1','2','3',...
```
~ 0x41, 0x42, 0x43,...
~ 0x61, 0x62, 0x63,...
~ 0x31, 0x32, 0x33,...

ARM Cortex MCU Block Diagram

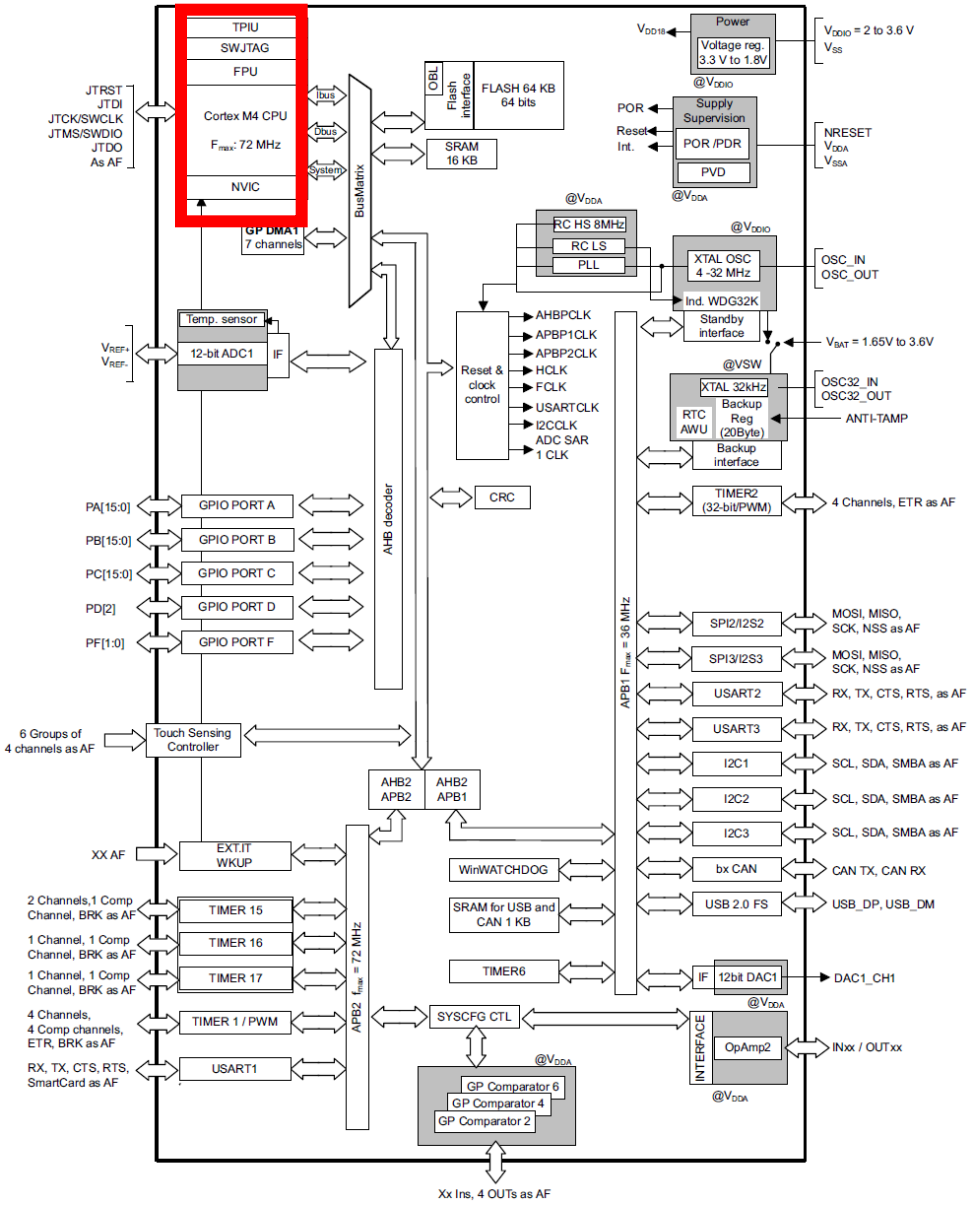
- Core: ARM® 32-bit Cortex®-M4 CPU with FPU (72 MHz max.)
- Memories: 32 to 64 Kb Flash, 16 Kb SRAM on data bus
- CRC calculation unit
- Clock management
- Up to 51 fast I/O ports
- Interconnect matrix
- 1 × ADC 0.20 µs (up to 15 channels)
- Temperature sensor
- 1 x 12-bit DAC channel
- Three fast rail-to-rail analog comparators
- 1 x operational amplifier
- Up to 18 capacitive sensing channels
- Up to 9 timers (PWM, Watchdog,...)
- Calendar RTC with alarm, periodic wakeup from Stop/Standby
- Comm. IF (3x I2Cs, <3 USARTs, <2 SPIs, USB 2.0, CAN, Infrared)



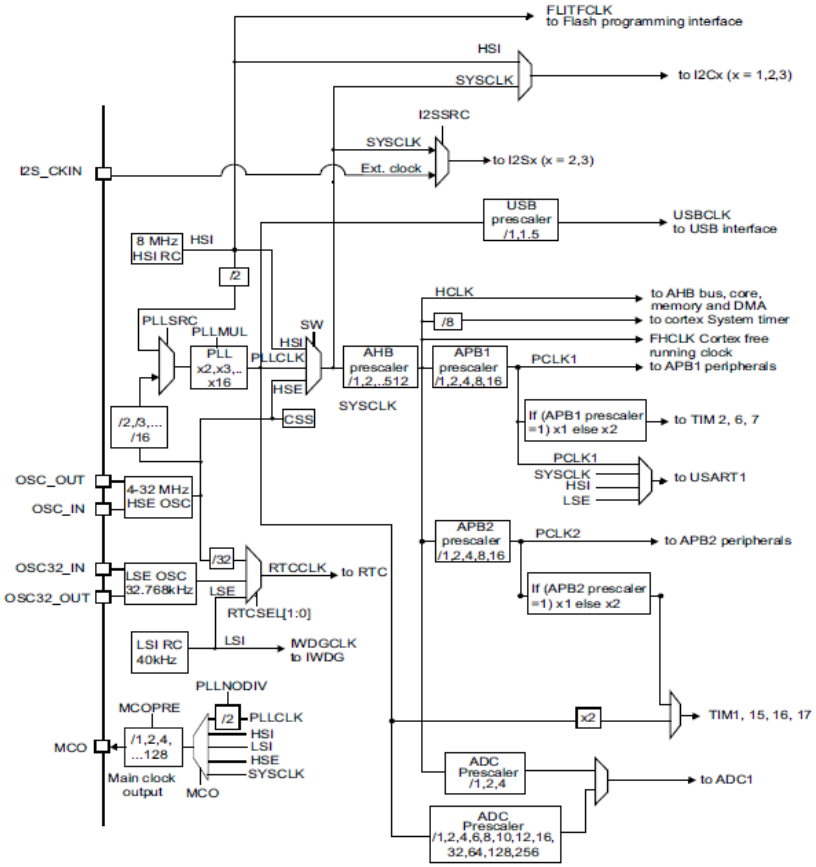


STM32F302R8

Overview



Timing Diagram

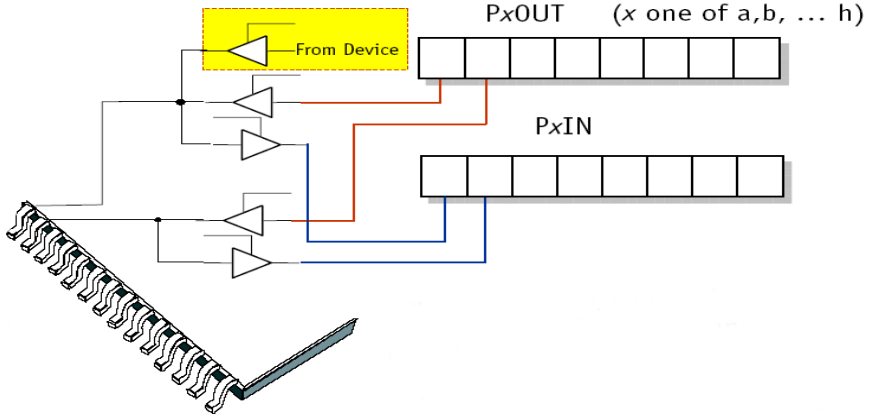


ARM Cortex General-Purpose I/O (GPIO)

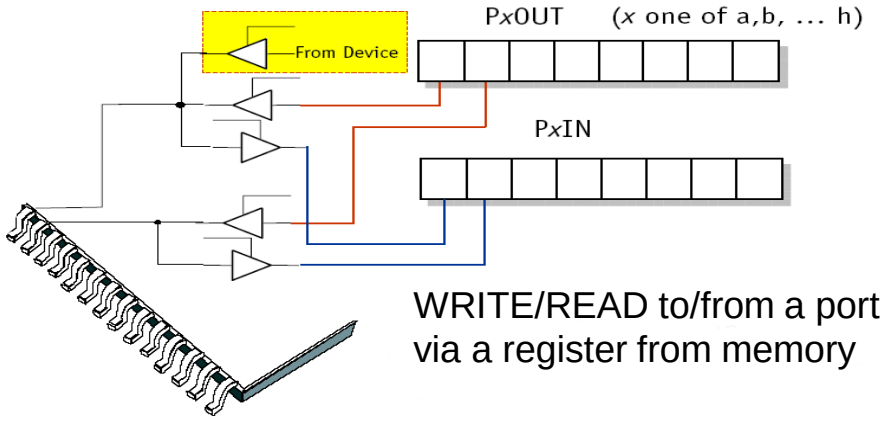
There are 5 ports, each with up to 16 bits available on the MCU (PA[16], PB[16], PC[16], PD[1], PF[2]). On the chip it is possible to get access to the bits on the ports.

Each pin on the port can:

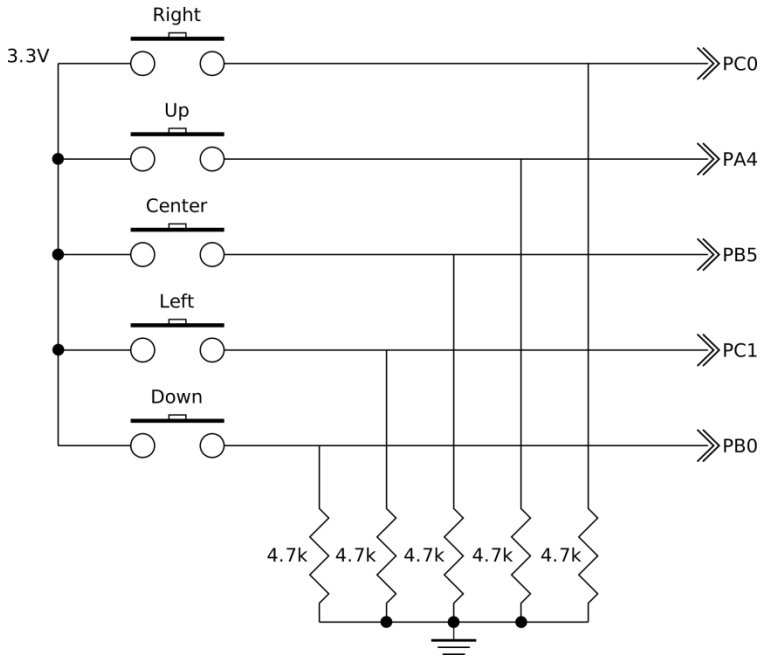
1. Be connected to a register (in the memory) to
   - WRITE to an external unit, or
   - READ from an external unit
2. Be connected to alternative function unit on the chip:
   - Timer
   - A/D converter
   - Communication module
   - Etc.



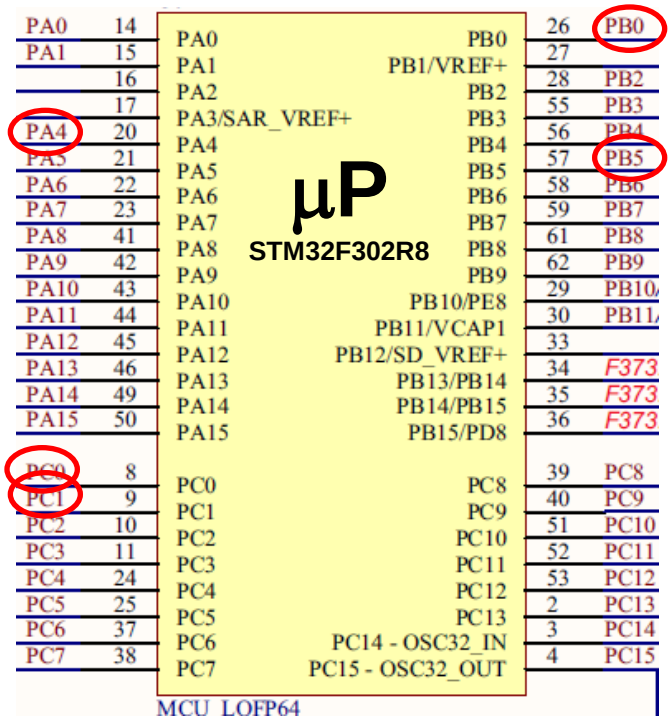
Relation between register & port



WRITE/READ to/from a port via a register from memory



Schematics of ARM Cortex with GPIOs



General Purpose In/Out (GPIO)

Each GPIO port is associated with the following registers:

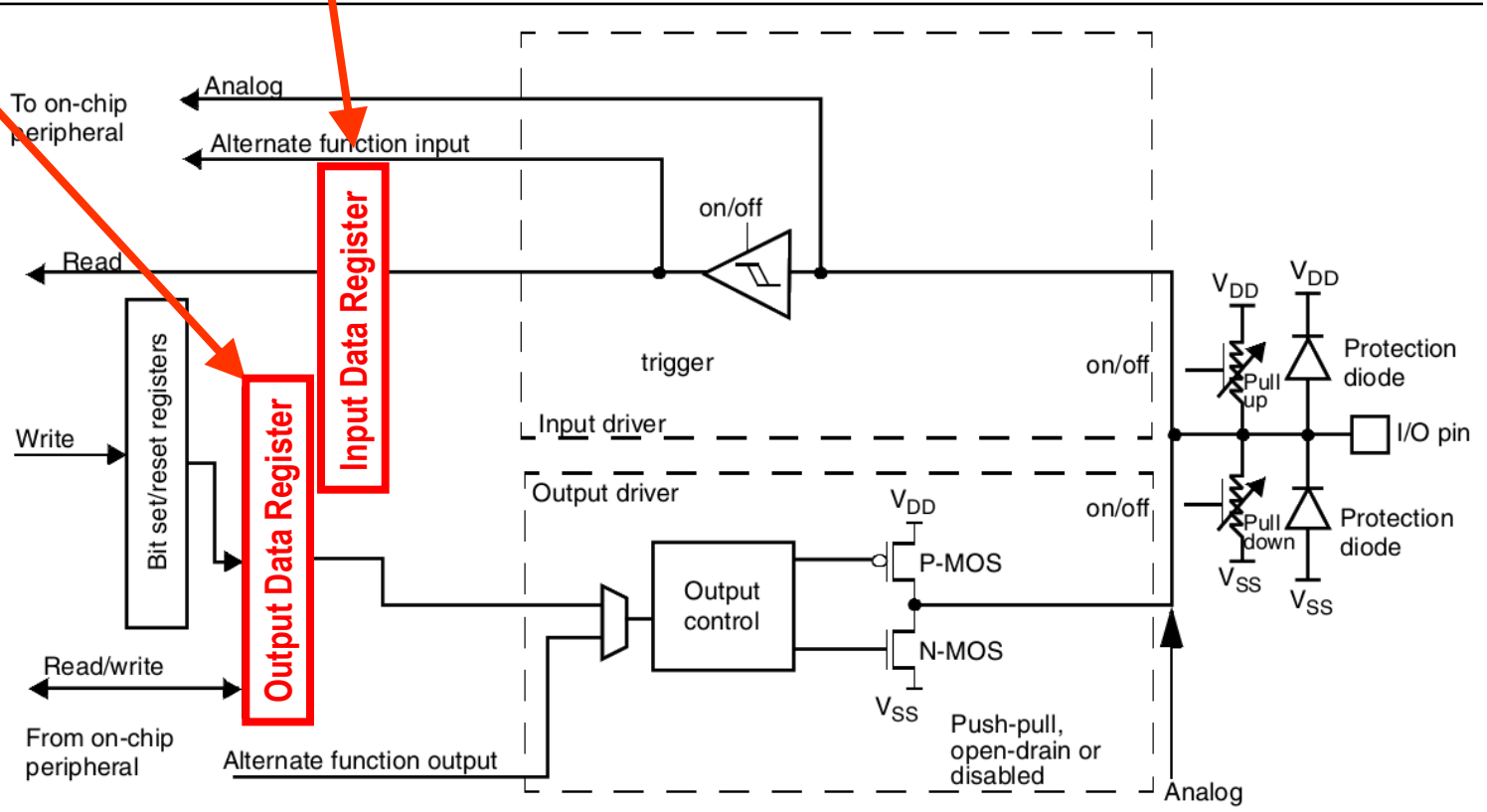
- 4 x 32-bit configuration registers:
  - GPIOx_MODER -> input, output, AF, analog
  - GPIOx_OTYPER -> push-pull or open-drain
  - GPIOx_OSPEEDR -> speed
  - GPIOx_PUPDR -> pull-up/pull-down whatever the I/O direction
- 2x 32-bit data registers:
  - GPIOx_IDR -> stores the data to be input, it is a read-only register
  - GPIOx_ODR -> stores the data to be output, it can be read/write
- 1x 32-bit set/reset register :
  - GPIOx_BSRR
- 1x 32-bit locking register:
  - GPIOx_LCKR
- 2x 32-bit alternate function selection registers:
  - GPIOx_AFRH
  - GPIOx_AFRL

General Purpose In/Out - Data

GPIO registers for 32-bit data (only low 16 bit are valid) are called

- GPIOx_**I**DR (in)
- GPIOx_**O**DR (out)

Figure 17. Basic structure of an I/O port bit



General Purpose In/Out - Conf

GPIOx_MODER (x =A..F)

Bits 2y+1:2y **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O mode.
00: Input mode (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	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]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_OTYPER (x = A..F)

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

Bits 15:0 **OTy:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O output type.
0: Output push-pull (reset state)
1: Output open-drain

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
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

General Purpose In/Out – Conf(2)

GPIOx_OSPEEDR (x =A..F)

Bits 2y+1:2y **OSPEEDRy[1:0]**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O output speed.
x0: Low speed
01: Medium speed
11: High speed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_PUPDR (x = A..F)

Bits 2y+1:2y **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Configuration example

```
/*****************************************************/

// set pin PA0 as input //

/*****************************************************/

GPIOA->MODER &= ~(0x00000003 << (0 * 2)); // Clear mode register

GPIOA->MODER |=  (0x00000000 << (0 * 2)); // Set mode register
(0x00 - Input, 0x01 - Output, 0x02 - Alternate, 0x03 - Analog)

GPIOA->PUPDR &= ~(0x00000003 << (0 * 2)); // Clear push/pull reg.

GPIOA->PUPDR |=  (0x00000002 << (0 * 2)); // Set push/pull reg.
(0x00 - No pull, 0x01 - Pull-up, 0x02 - Pull-down)

uint16_t val = GPIOA->IDR & (0x0001 << 0); // Read from PA0
```

Configuration example (2)

```
/*****************************************************/

// Set pin PA1 to output //

/*****************************************************/

GPIOA->OSPEEDR &= ~(0x00000003 << (1 * 2)); // Clear speed register
GPIOA->OSPEEDR |=  (0x00000002 << (1 * 2)); // set speed register
(0x01 - 10 MHz, 0x02 - 2 MHz, 0x03 - 50 MHz)

GPIOA->OTYPER &= ~(0x0001 << (1)); // Clear output type register
GPIOA->OTYPER |=  (0x0000 << (1)); // Set output type register
                                   (0x00 - Push pull, 0x01 - Open drain)

GPIOA->MODER &= ~(0x00000003 << (1 * 2)); // Clear mode register
GPIOA->MODER |=  (0x00000001 << (1 * 2)); // Set mode register
(0x00 - In, 0x01 - Out, 0x02 - Alternate, 0x03 - Analog in/out)

GPIOA->ODR |= (0x0001 << 1); //Set pin PA1 to high
```

Comments/Feedback to Exercise 3-4 (Friday)

- **Just in case … re-check which COMx is installed!**
- **Periodicity in sinus and cosinus**
  - **LUT: 0 degree is x0000, 360 degrees is x0200**
  - **236 degrees is in binary 0001.0100.1111 (x014F)**
  - **236+360 degrees is in binary 0011.0100.1111 (x034F)**
  - **236+2x360 degrees is in binary 0101.0100.1111 (x054F)**
- **`signed short sin(int a) {return SIN[a & 0x1FF];}`**
- **Use of Pointers when calling a function**

```
void swap (int *px, int *py)
     {int temp; temp=*px; *px=*py; *py=temp;}
```

- **Structure and function to Rotate**

```
void RotVector (struct vector_t *v, int angle) {
// call with RotVector (&vec,angle)//
// variable declarations://  long temp;
    (*v).x = // v->x = // Udtryk //;
    (*v).y = // v->y = // Udtryk //;}
```

Exercise 5

- **Learn how to use the General Purpose In/Out (GPIO) of the μP**
- **By detecting that a button has been pressed, a message is written**
- **Parts of the "STM32F302x" literature containing the chapters "General-Purpose I/O", "Interrupt Controller", and "Timers" should be used as reference.**
- **You control each I/O register (A to H) through a number of configuration registers:**
  - GPIOx_MODER -> input, output, AF, analog
  - GPIOx_OTYPER -> push-pull or open-drain
  - GPIOx_OSPEEDR -> speed
  - GPIOx_PUPDR -> pull-up/pull-down whatever the I/O direction
- **Display a color on the RGB LED**
- **Note:** The scope of a variable is the function where it was created. Variables declared outside of functions can not be accessed from other functions unless they're declared to be `static`. Variables declared in functions can preserve their value between calls if they are defined as `static`, otherwise they're automaticly getting recreated for each function call.

Programming Project (3)

Afternoon

- **Exercise 6:**
  - Timer 2 Control Registers
  - Ex.: Timer 2 in ”continuous mode”
  - Interrupt Registers
  - TIM2 InterruptEnable
  - Application of **onboard Timer** + **IRQ** for implementing a stop watch.
  - Serial read, implement the function **`uart_getc`**
  - OBS! - Keep a good program structure!
    Fx.: only ansi functions in the library Ansi.c / Ansi.h
    - Keep a good formatting in file editing!
    Fx.: Indent in loops, blocks, etc.

Timer 2 Control Registers

Register	Address Offset	Size [Bits]	Description
TIM2			Port base address
TIM2->CR1	0x00	16	Primary Configuration
TIM2->CR2	0x04	32	Secondary Configuration
TIM2->SMCR	0x08	32	Slave Mode Control
TIM2->DIER	0x0C	32	DMA/Interrupt Enable
TIM2->SR	0x10	32	Status
TIM2->EGR	0x14	32	Event Generation
TIM2->CCMR[1,2]	0x18 - 0x1C	32 + 32	Compare Mode
TIM2->CCER	0x20	32	Compare Enable
TIM2->CNT	0x24	32	Counter
TIM2->PSC	0x28	16	Prescaler
TIM2->ARR	0x2C	32	Auto-reload
TIM2->CCR[1,..,4]	0x34 - 0x40	32 + 32 + 32 + 32	Capture/Compare
TIM2->DCR	0x48	16	DMA Control
TIM2->DMAR	0x4C	16	DMA Address

Timer 2 Control Registers

Address offset: 0x00

Reset value: 0x0000

TIMx control register 1 (TIMx_CR1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIF RE-MAP	Res.	CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

UIFREMAP:—UIF status bit remapping

0: No remapping.

1: No remapping.

CKD:—Clock Division

00: tDTS = tCK_INT

01: tDTS = 2 × tCK_INT

10: tDTS = 4 × tCK_INT

11: Reserved

ARPE:—Auto-reload preload enable

0: TIMx_ARR register is not buffered

1: TIMx_ARR register is buffered

CMS:—Center-aligned mode selection

00: Edge-aligned mode.

01: Center-aligned mode 1.

10: Center-aligned mode 2.

11: Center-aligned mode 3.

Dir:—Direction

0: Up-counting.

1: Down-counter.

OPM:—One-pulse mode

0: Do not stop at update event.

1: Stop at update event.

URS:—Update request source

0: Any event.

1: Counter over-/underflow

UDIS:—Update disable

0: Update event enabled.

1: Update event disabled.

CEN:—Counter enable

0: Counter disabled.

1: Counter enabled.

Timer 2 Counter/Reload Values

TIM2->CNT: Timer 2 Counter Value

TIM2->ARR : Timer 2 Reload Value

$$Upcounting\ Mode\ Timeout\ Period = \frac{(1 + Reload\ Value) \cdot (1 + Prescale)}{System\ Clock\ Frequency}$$

$$Reload\ Value\ for\ 1\ MHz = \frac{Timeout\ Period \cdot System\ Clock\ Frequency}{1 + Prescale} - 1$$
$$= \frac{1 \times 10^{-6}\text{s} \cdot 64 \times 10^{6}\text{Hz}}{1 + 0} - 1 = 63$$

```
//Reload Value
 TIM2->ARR = 63; // Set auto reload value
//Start timer
 TIM2->CR1  |= 0x0001; // Enable timer
```

21.4.10 TIMx counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNT[31] or UIFCPY	CNT[30:16] (depending on timers)														
rw or r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 Value depends on IUFREMAP in TIMx_CR1.
If UIFREMAP = 0
CNT[31]: Most significant bit of counter value (on TIM2)
Reserved on other timers
If UIFREMAP = 1
UIFCPY: UIF Copy
This bit is a read-only copy of the UIF bit of the TIMx_ISR register

Bits 30:16 **CNT[30:16]**: Most significant part counter value (on TIM2)

Bits 15:0 **CNT[15:0]**: Least significant part of counter value

21.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

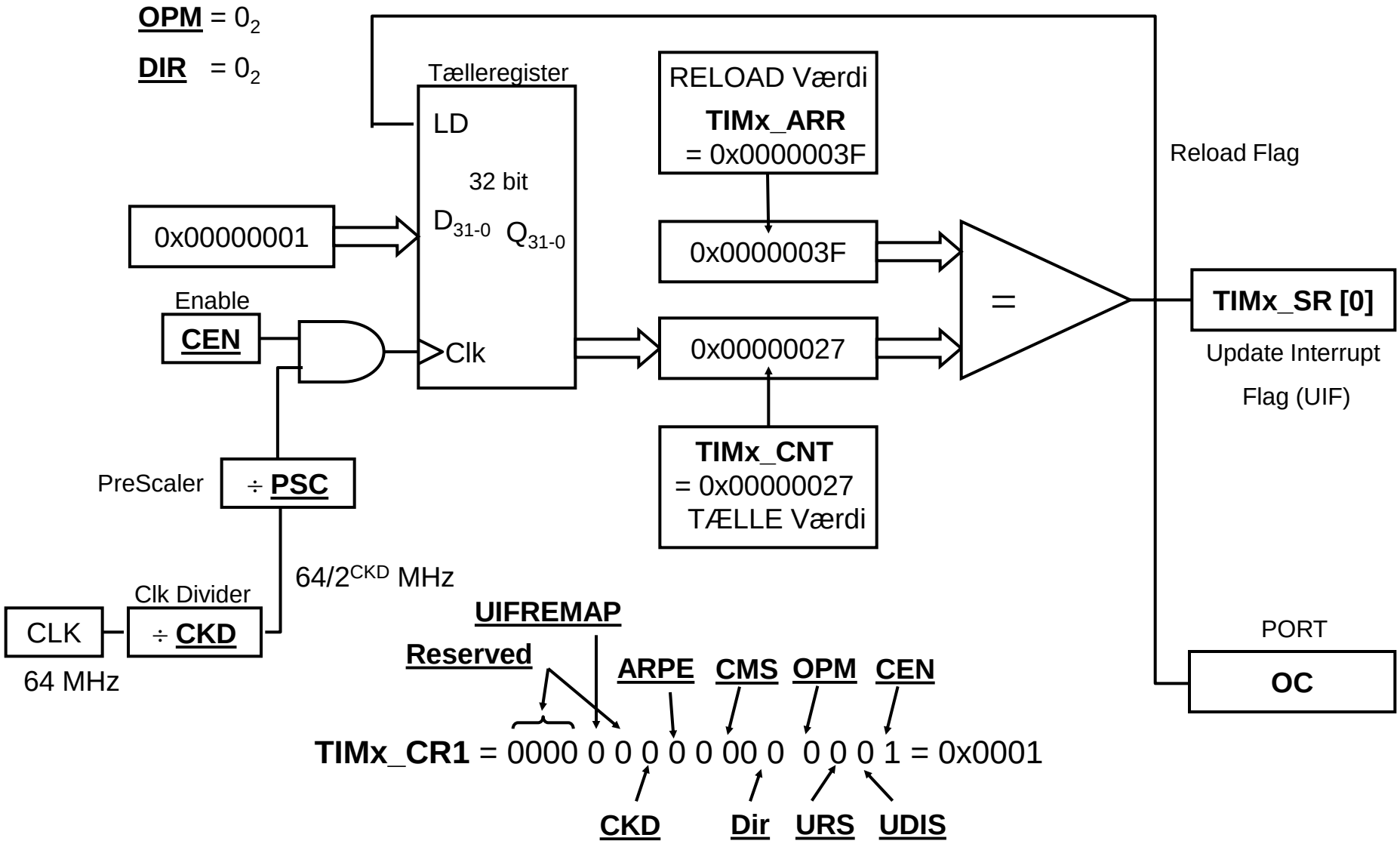
Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **ARR[31:16]**: High auto-reload value (on TIM2)

Bits 15:0 **ARR[15:0]**: Low Auto-reload Prescaler value
ARR is the value to be loaded in the actual auto-reload register.
Refer to the *Section 21.3.1: Time-base unit on page 552* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

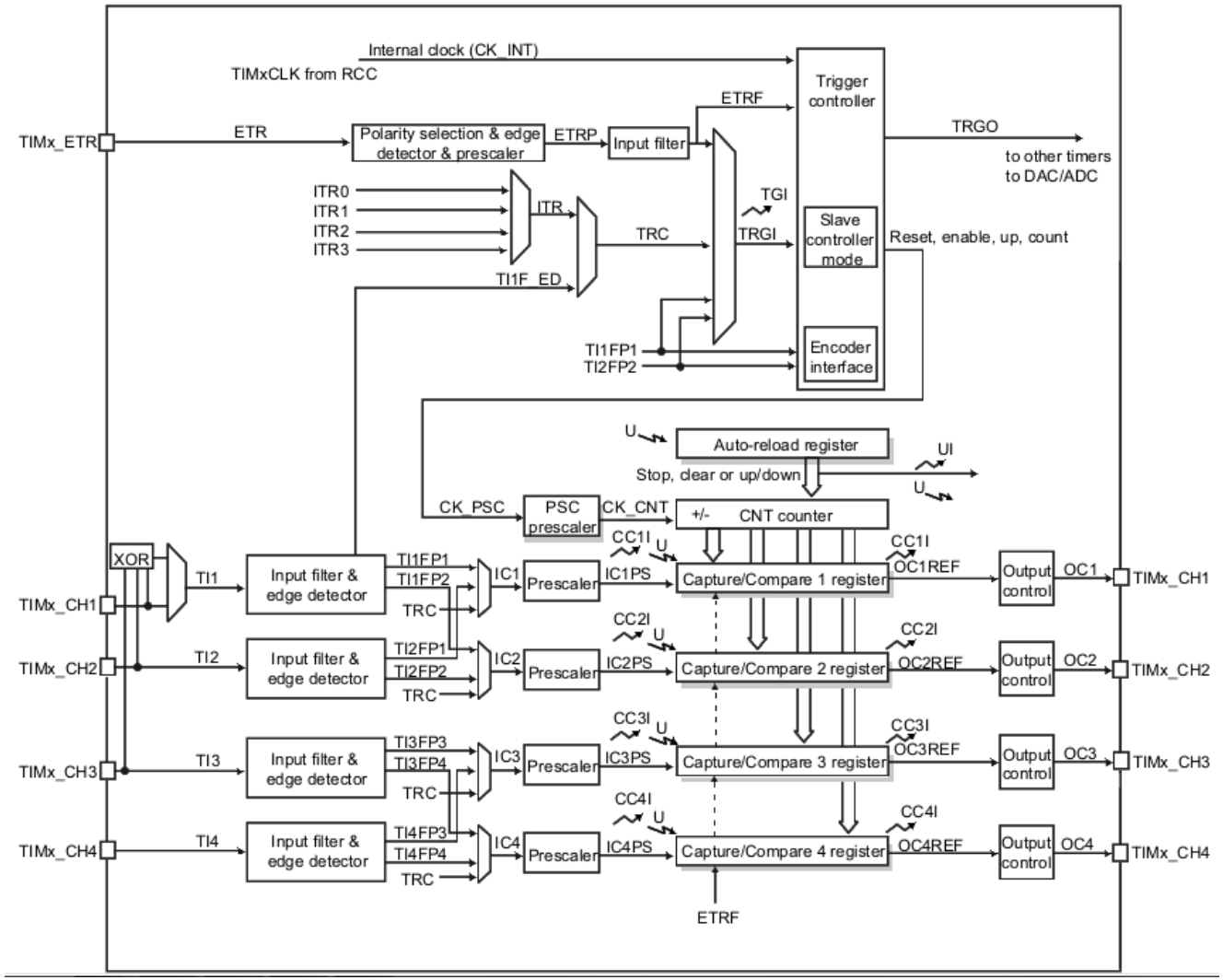
Timer 2 i ”continuous mode”



Timer 2 i ”continuous mode”

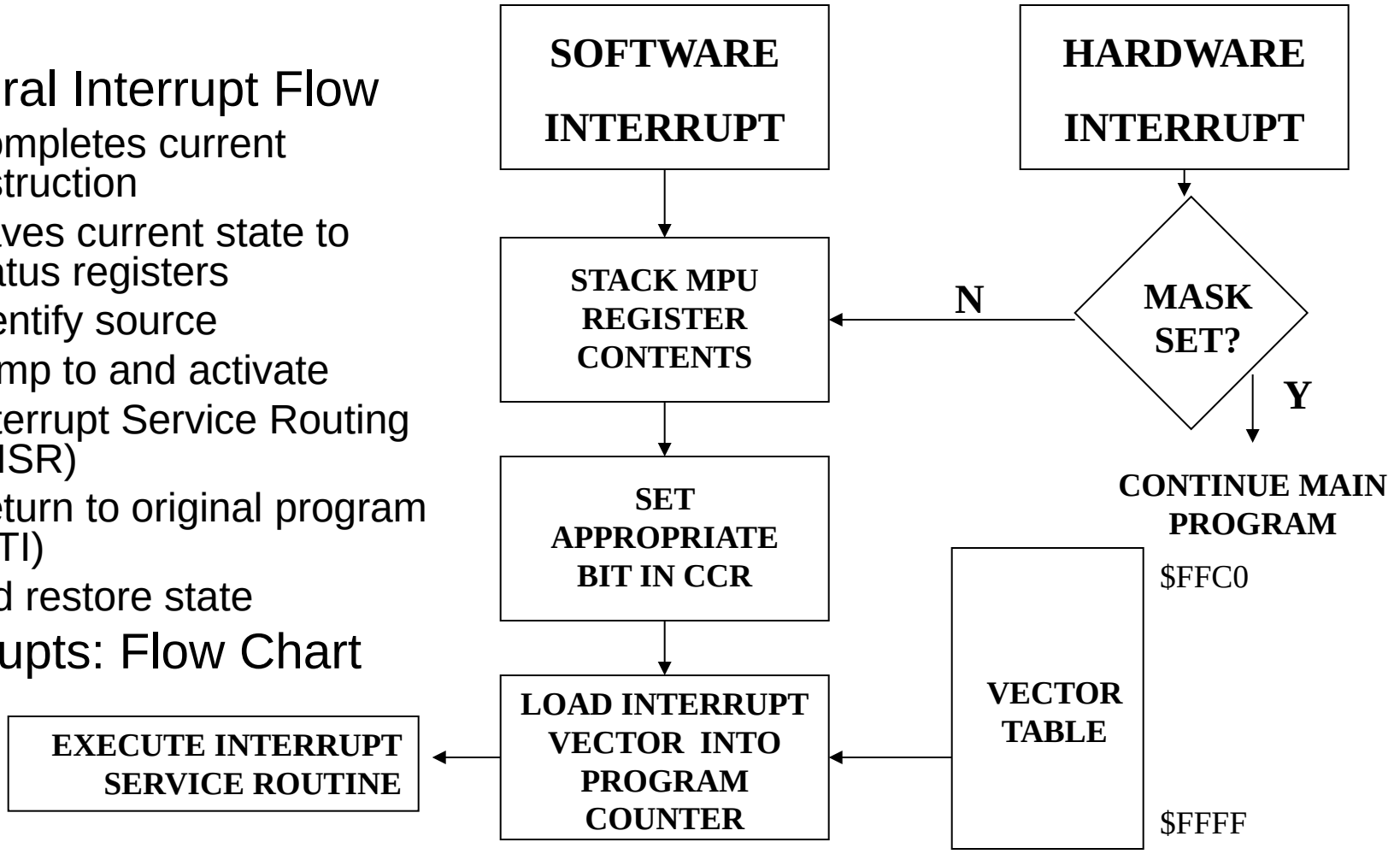
STM32F302x8_Reference_Manual, page 551

Figure 189. General-purpose timer block diagram



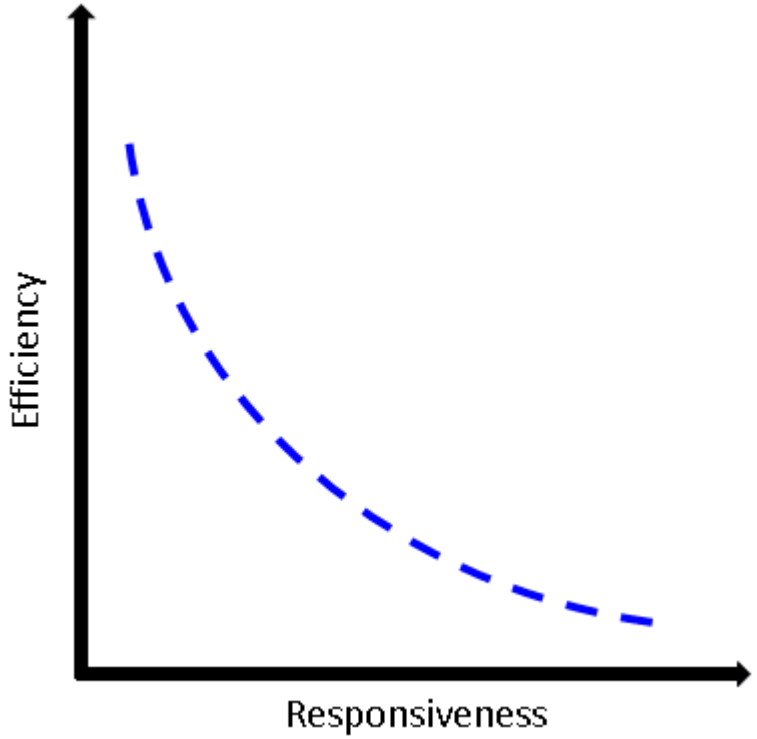
Interrupts

- Interrupt vs Pooling
- General Interrupt Flow
  - Completes current instruction
  - Saves current state to status registers
  - Identify source
  - Jump to and activate Interrupt Service Routing (ISR)
  - Return to original program (RTI) and restore state
- Interrupts: Flow Chart



Interrupt advantages

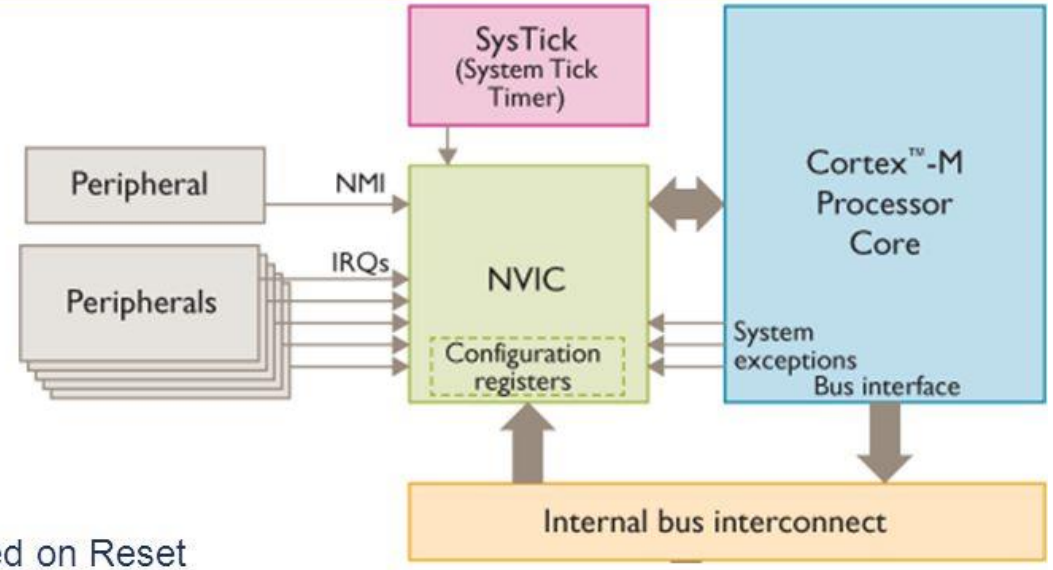
- Increases efficiency
  - Minimizing useless work
  - Maximizing useful work
  - Saving cycles & energy
- Increases responsiveness
  - Minimizing latency
  - Tight event-action coupling



IRQ Structure for AMR Cortex M4

Nested Vectored Interrupt Controller

- **NVIC manages and prioritizes external interrupts for Cortex-M4**
- **Interrupts are types of exceptions**
  - CM4 supports up to 240 IRQs
- **Modes**
  - Thread Mode: entered on Reset
  - Handler Mode: entered on executing an exception
- **Privilege level**
- **Stack pointers**
  - Main Stack Pointer, MSP
  - Process Stack Pointer, PSP
- **Exception states: Inactive, Pending, Active, A&P**



From ARM Cortex M4

Types of Interrupt

- Over 100 interrupt sources
  - Power on reset, bus errors, I/O pins changing state, data in on a serial bus etc.
- Need a great deal of control
  - Ability to enable and disable interrupt sources
  - Ability to control where to branch to for each interrupt
  - Ability to set interrupt priorities
    - Who wins in case of a tie
    - Can interrupt A interrupt the ISR for interrupt B?
    - If so, A can “preempt” B.
- All that control will involve memory mapped I/O.
  - And given the number of interrupts that’s going to be a pain

Table 40. STM32F302xB/C/D/E vector table

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	fixed	Reset	Reset	0x0000 0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-	-1	fixed	HardFault	All class of fault	0x0000 000C
-	0	settable	MemManage	Memory management	0x0000 0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	-	Reserved	0x0000 001C - 0x0000 0028
■	■	■	■	■	■
28	35	settable	TIM2	TIM2 global interrupt	0x0000 00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000 00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000 00B8
31	38	settable	I2C1_EV	I2C1 event interrupt & EXTI Line23 interrupt	0x0000 00BC
32	39	settable	I2C1_ER	I2C1 error interrupt	0x0000 00C0
33	40	settable	I2C2_EV	I2C2 event interrupt & EXTI Line24 interrupt	0x0000 00C4
34	41	settable	I2C2_ER	I2C2 error interrupt	0x0000 00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000 00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000 00D0
37	44	settable	USART1	USART1 global interrupt & EXTI Line 25	0x0000 00D4
38	45	settable	USART2	USART2 global interrupt & EXTI Line 26	0x0000 00D8
39	46	settable	USART3	USART3 global interrupt & EXTI Line 28	0x0000 00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000 00E0
41	48	settable	RTC_Alarm	RTC alarm interrupt	0x0000 00E4
42(1)	49	settable	USBWakeUp	USB wakeup from Suspend (EXTI line 18)	0x0000 00E8
43	50	settable	Reserved		0x0000 00EC
44	51	settable	Reserved		0x0000 00F0

Interrupt for Timer 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res.	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

- **TIMx_DIER**
  - TDE: Trigger DMA request enable
  - CCxDE: Capture/Compare x DMA request enable
  - UDE: Update DMA request enable
  - TIE: Trigger interrupt enable
  - CCxIE: Capture/Compare x interrupt enable
  - **UIE: Update interrupt enable**

STM32F302x8_Reference_Manual, page 602

void TIM2_IRQHandler(void)

TIM2_IRQHandler: Every peripheral on the STM32 chip that is capable of generating an interrupt has a built-in interrupt function predefined in memory. After an interrupt is enabled in both the peripheral itself and in the Nested vectored interrupt controller (NVIC) this function will be called whenever the interrupt is triggered. By defining a function in your code with the exact same name the compiler will automatically know to use your function whenever an interrupt is triggered. For timer 2 the function name definition is:

```
void TIM2_IRQHandler(void)
```

Forbidden Pins

Avoid the following pins:

- PA2, PA3, PA13, PA14
- PB3, PB13
- PC13, PC14, PC15
- PF0

- Otherwise, the ARM will stall