Introduction to Microcontrollers

Michele Magno
IIS Group - ETH Zurich

michele.magno@iis.ee.ethz.ch

Outline

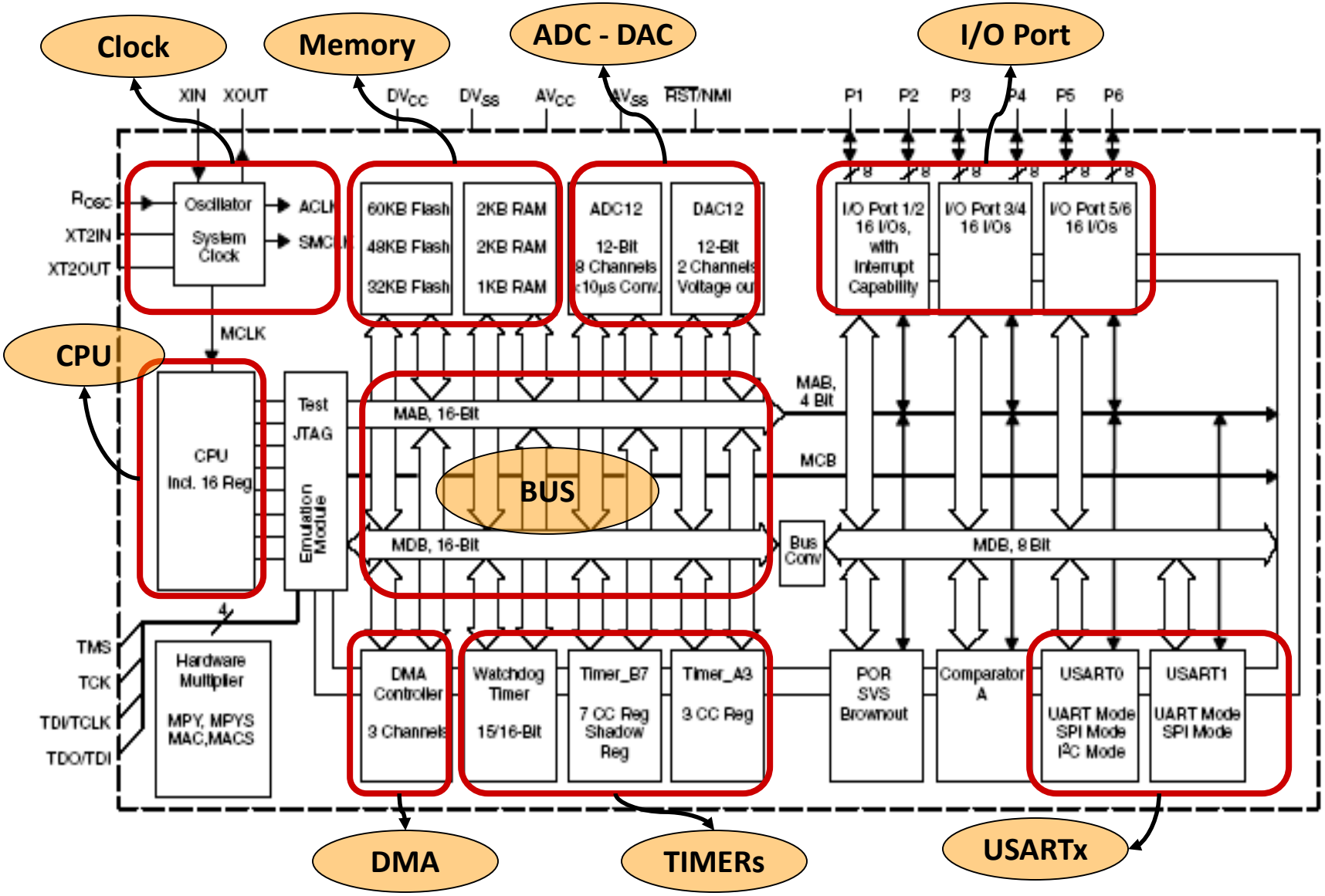
- MCU Architecture
- CPU
- Power Consumption
- MCU Peripherals
- ARM Architecture
  - ARM Coretx
  - ARM Instruction Set
  - STM32 ARM Coretex Mx Family
  - ISA
- Data Sheet Examples

What is a microcontroller ?

A ***Microcontroller*** is a small CPU with many support devices built into the chip

- Self Contained (CPU, Memory, I/O)
- Application or Task Specific (Not a general-purpose computer)
- Appropriately scaled for the job
- Small power consumption
- Low costs ($0.50 to $5.00.)

Example of MCU Architecture



Performance Metrics

- How we compare and classify microcontrollers?
  - Performance Metrics NOT easy to define and mostly application depended.

Eletrical:

- Power Consumptions
- Voltage Supply
- Noise Immunity
- Sensitivity

Goal: best *tradeoff* power consumptions ***Vs*** performances

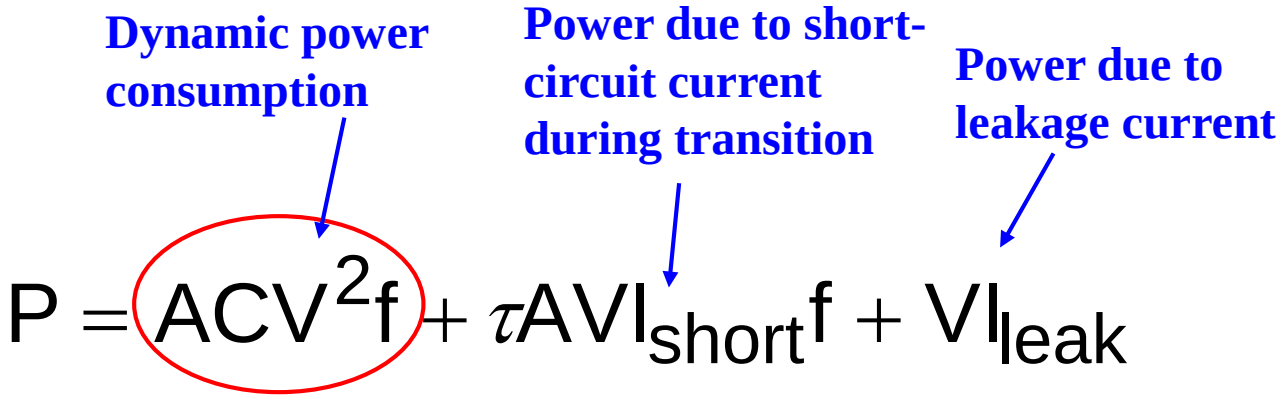
Computation:

- Clock Speed
- MIPS (instructions per sec)
- Latency
  - Lateness of the response
  - Lag between the begin and the end of the computation
- Throughput
  - Tasks per second
  - Byte per second

Power as a Design Constraint

- Why worry about power?
  - Battery life in portable and mobile platforms
  - Power consumption in desktops, server farms
    - Cooling costs, packaging costs, reliability, timing
    - Power density: 30 W/cm2 in Alpha 21364 (3x of typical hot plate)

Where does power go in CMOS?



Dynamic Power Consumption

C – Total capacitance seen by the gate's outputs Function of wire lengths, transistor sizes, ...

V – Supply voltage Trend: has been dropping with each successive fab

$$ACV^2f$$

A - Activity of gates How often on average do wires switch?

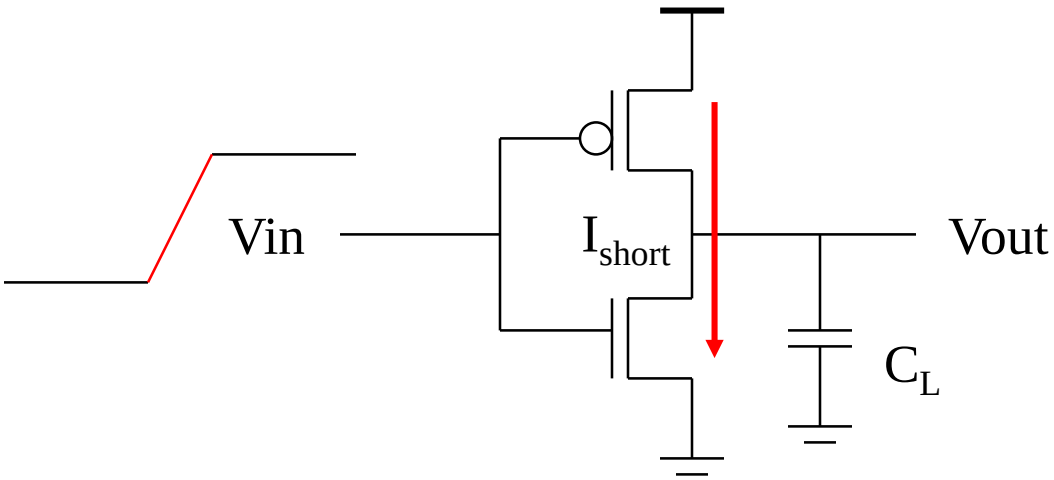
f – clock frequency Trend: increasing ...

Reducing Dynamic Power

1) **Reducing V has quadratic effect; Limits?**
2) **Lower C - shrink structures, shorten wires**
3) **Reduce switching activity - Turn off unused parts or use design techniques to minimize number of transitions**

Short-circuit Power Consumption

$$\tau A V I_{short} f$$

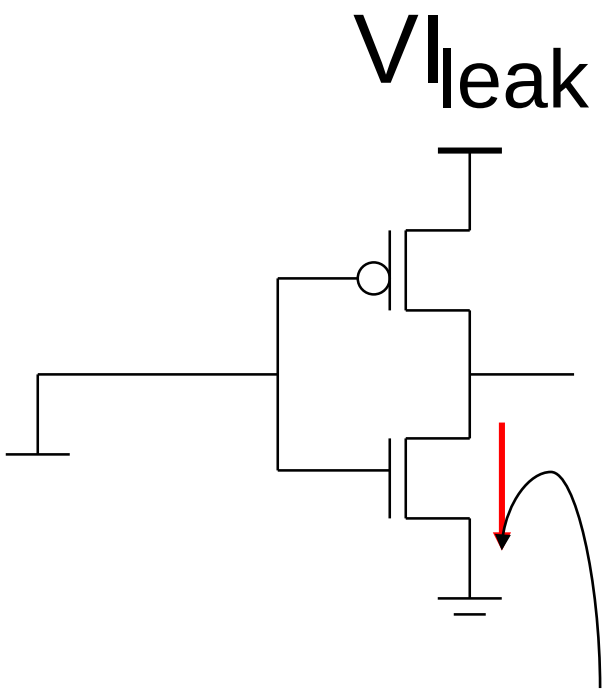


Finite slope of the input signal causes a direct current path between V_{DD} and GND for a short period of time during switching when both the NMOS and PMOS transistors are conducting

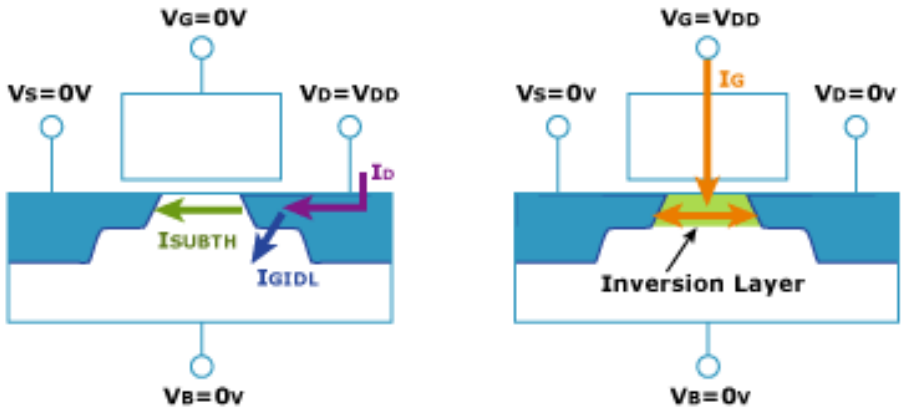
Reducing Short-circuit

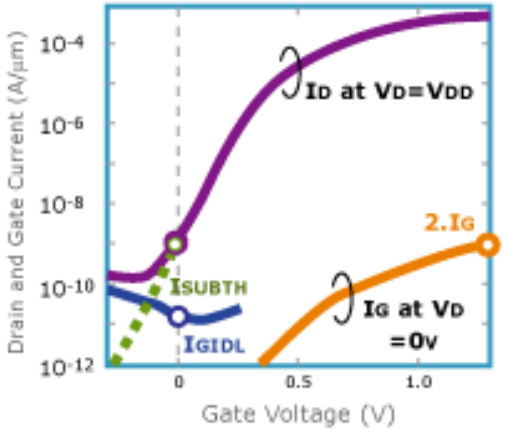
1) **Lower the supply voltage V**
2) **Slope engineering – match the rise/fall time of the input and output signals**

Leakage Power



Sub-threshold current

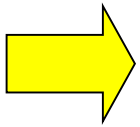




Sub-threshold current grows exponentially with increases in temperature and decreases in Vt

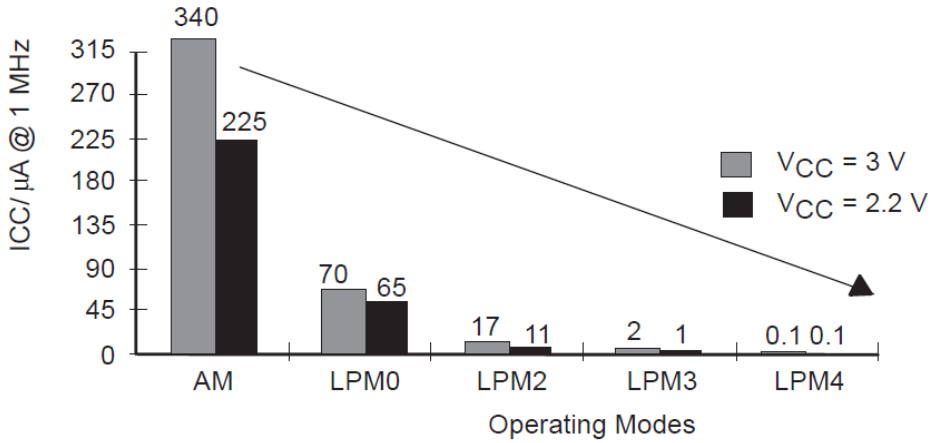
How can we reduce power consumption?

- Dynamic power consumption
  - Reduce the rate of charge/discharge of highly loaded nodes
  - Reduce spurious switching (glitches)
  - Reduce switching in idle states (clock gating)
  - Decrease frequency
  - Decrease voltage (and frequency)
- Static power Consumption
  - Smaller area (!)
  - Reduce device leakage through power gating
  - Reduce device leakage through body biasing
  - Use higher-threshold transistors when possible



Power performance tradeoffs!

Operating Modes



- **Assembler Code Example:**

```
bis.w     #CPUOFF,SR      ; LPM0
```

- **C Code Example:**

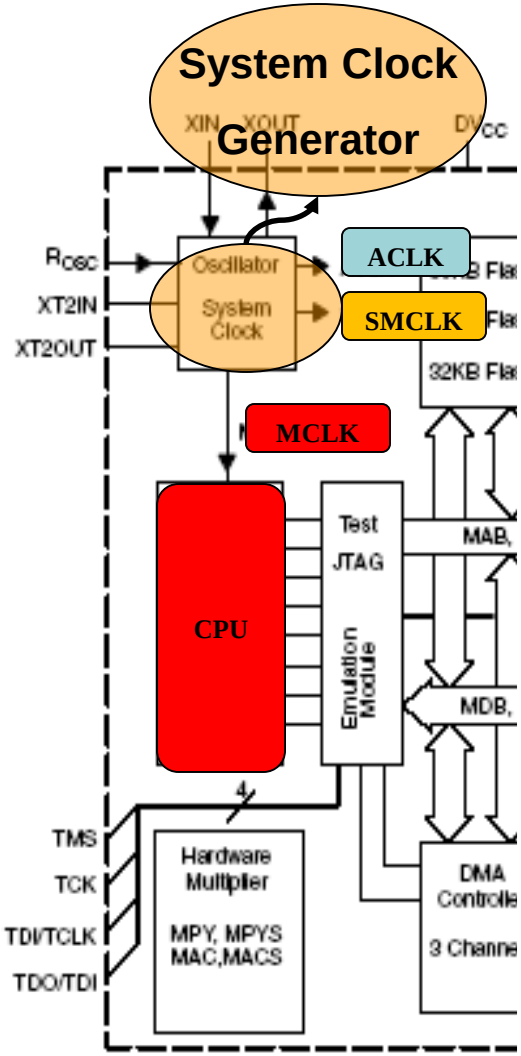
```
_BIS_SR (CPUOFF);         // LPM0
```

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled SMCLK , ACLK are active
0	1	0	1	LPM1	CPU, MCLK, DCO osc. are disabled DC generator is disabled if the DCO is not used for MCLK or SMCLK in active mode SMCLK , ACLK are active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

Why *Ultra-low Power* Is so Important

- Longer battery life
- Smaller products
- Simpler power supplies
- Less EMI simplifies PCB
- *Permanent* battery
- Reduced liability

Clock Distribution

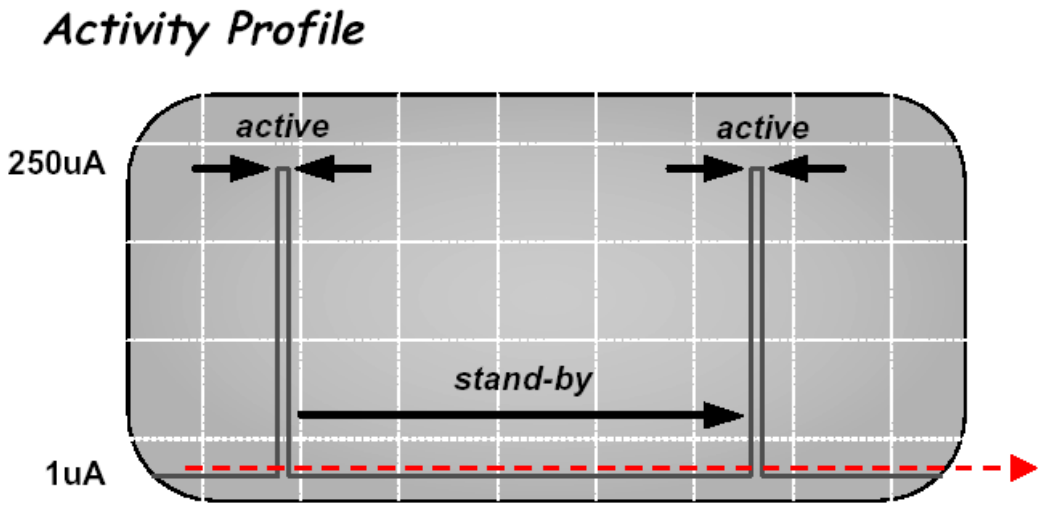


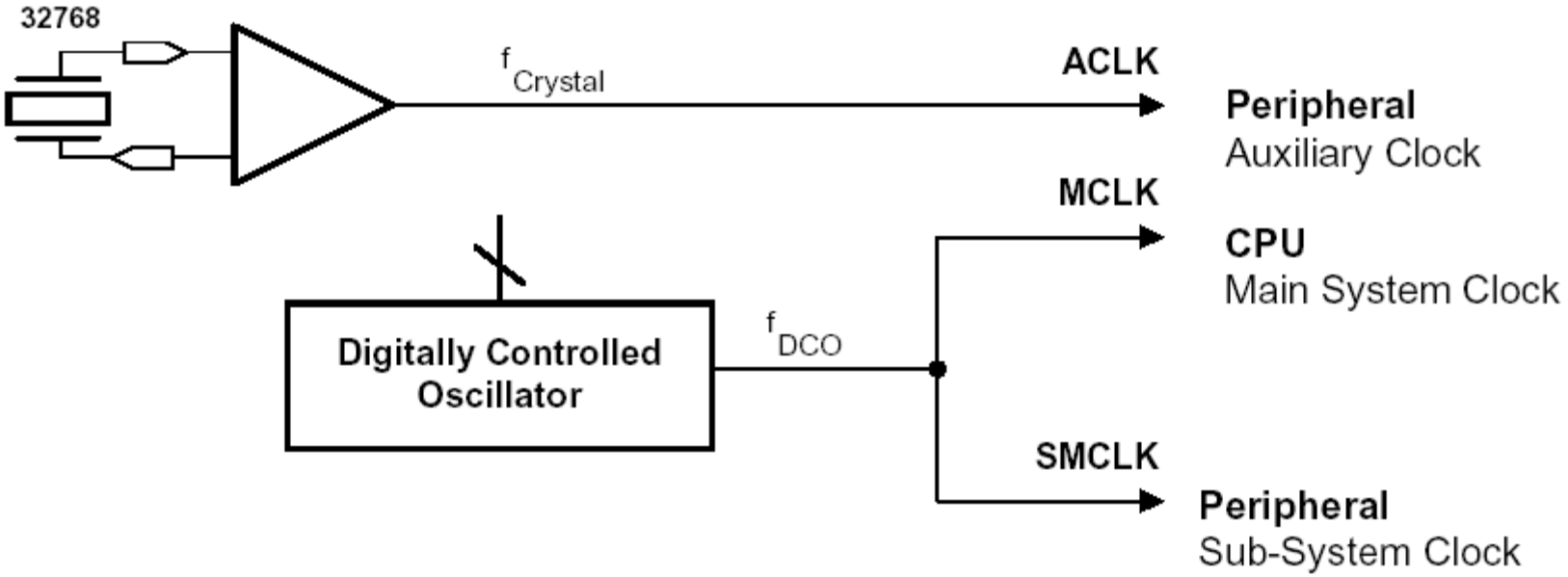
Key Features:

- **MCLK** Main clock provided to the CPU
- **SMCLK** Sub-Main clock provided to the peripherals
- **ACLK** Auxiliary clock at low frequency provided to the peripherals
- Peripherals can work at High and Low frequency
- Each **Clock** can be disabled (**Clock Gating**, reducing dynamic power) by setting the status register SR.
- The **CPU** can be disabled (reducing Leakage power) by setting the SR.

Clock System Generator

Clock system Module provides the clocks for the MCU devices



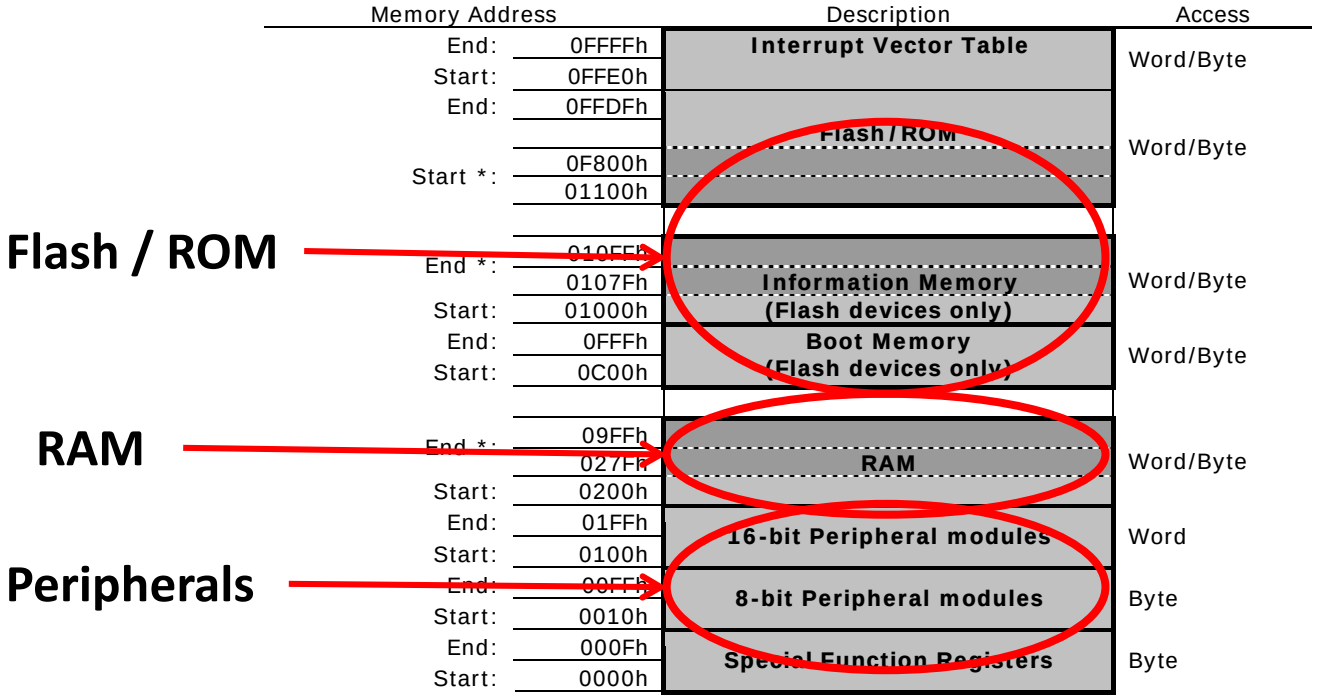


Memory

- RAM (usually SRAM)
  - Volatile memory for runtime execution
  - Fastest access, low amount (<100Kb)
  - Allocates variables
- Flash ROM
  - On-chip non-volatile memory used for code or data storage
  - 8-512Kb, about 10k write cycles
  - Bootloader: protected section to upload code in flash
- **(Ferroelectric Random Access Memory) FRAM**
  - Forefront of next generation non-volatile memory technology
  - On-chip non-volatile memory faster (50ns) and lower power (250x less) than Flash.
- External memory
  - Connected via serial (I2C, SPI) or dedicated (FSMC) interface

Memory - Address Space

- On-Chip FLASH/ROM and RAM memory
- Everything is mapped into a single, contiguous address space:
  - All memory, including RAM, Flash/ROM, information memory, special function registers (SFRs), and peripheral registers.



Interrupts

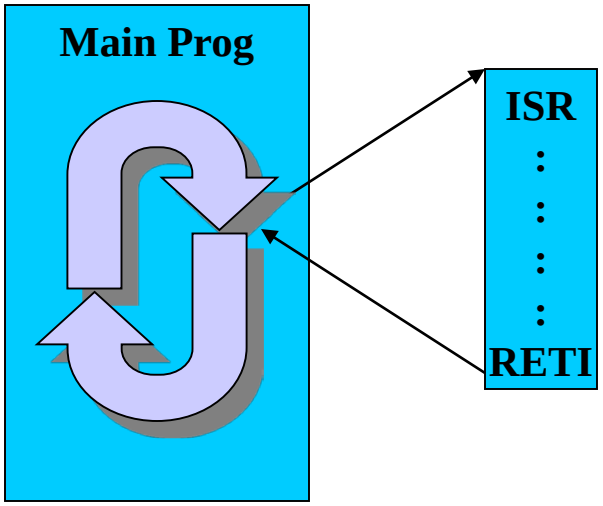
- A way to respond to an external event (i.e., flag being set) without polling

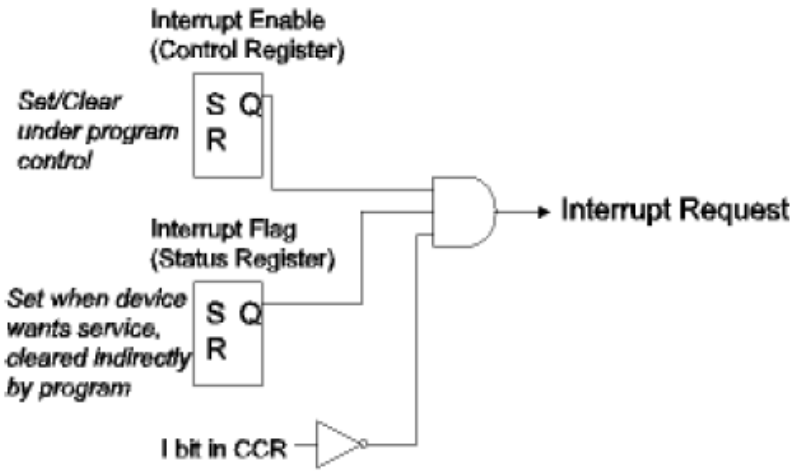
How it works:

- H/W senses flag being set
- Automatically transfers control to s/w that "services" the interrupt
- When done, H/W returns control to wherever it left off

Advantages:

- Transparent to user
- cleaner code
- µC doesn't waste time polling



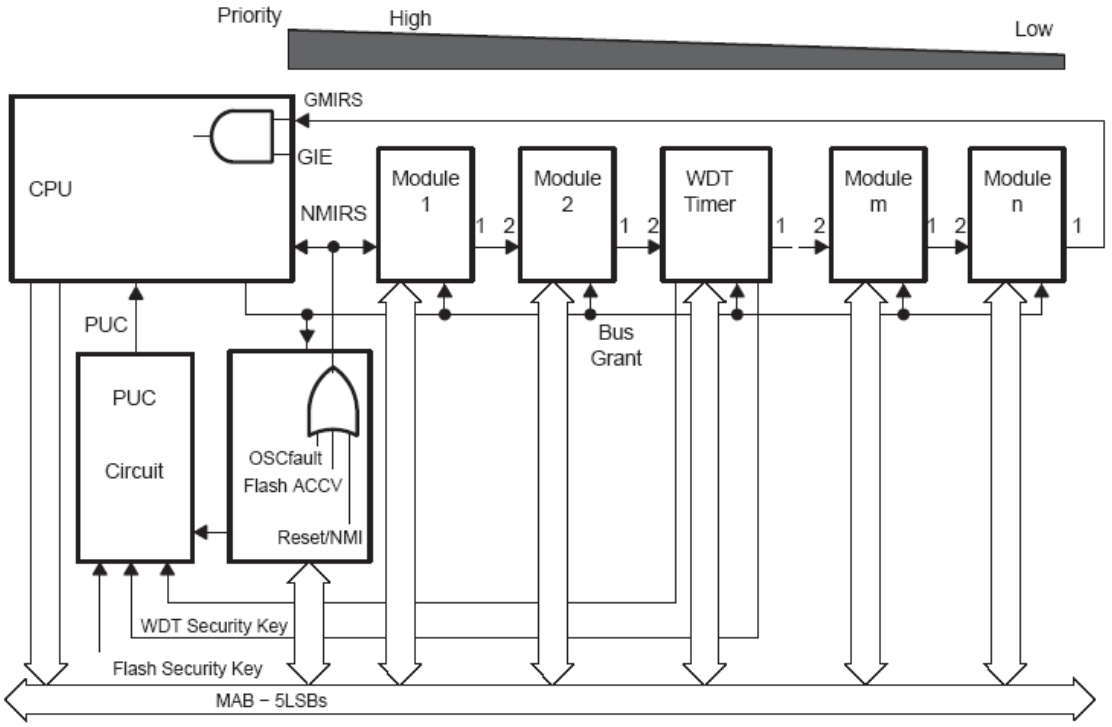


Interrupts: details

- 3 types
  - System reset
  - (Non)-maskable NMI
  - Maskable

- Interrupt priorities could be fixed and defined by the arrangement of modules or set in the interrupt priority register

Figure 2−4. Interrupt Priority



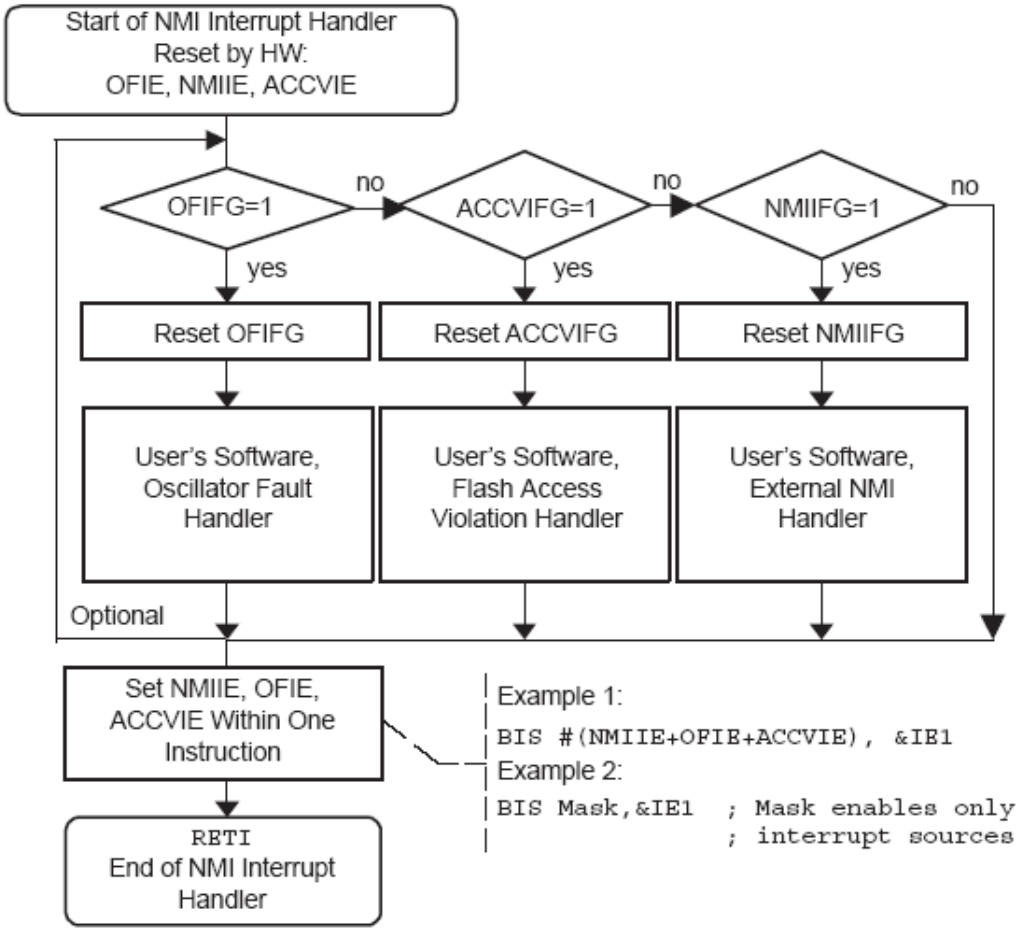
(Non)-Maskable Interrupts

- Sources
  - An edge on the RESET pin when configured in NMI mode
  - An oscillator fault occurs
  - An access violation to the flash memory

- Are not masked by GIE (General Interrupt Enable), but are enabled by individual interrupt enable bits

NMI Interrupt Handler example

Figure 2−6. NMI Interrupt Handler



Maskable Interrupts

- Caused by peripherals with interrupt capability
- Each interrupt can be disabled individually by an interrupt enable bit
- All interrupts can be disabled by GIE bit in the status register

Interrupt acceptance

1) Any currently executing instruction is completed.
2) The ProgramCounter PC, which points to the next instruction, is pushed onto the stack.
3) The StatusRegister SR is pushed onto the stack.
4) The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
5) The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
6) The SR is cleared. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.
7) The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.

Return from Interrupt

RETI - Return from Interrupt Service Routine

1. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
2. The PC pops from the stack and begins execution at the point where it was interrupted.

Timers

- Correct system timing is a fundamental requirement for the proper operation of a real-time application;
  - If the timing is incorrect, the input data may be processed after the output was updated
- The timers may be driven from an internal or external clock;
- Usually timers include multiple independent capture and compare blocks, with interrupt capabilities;
- Main applications:
  - Generate events of fixed-time period;
  - Allow periodic wake-up from sleep;
  - Count external signals/events;
  - Signal generation (Pulse Width Modulation – PWM);
  - Replacing delay loops with timer calls allows the CPU to sleep between operations, thus consuming less power.

24

Timers

- The general-purpose timers consist of a **16-bit auto-reload counter** driven by a programmable prescaler.

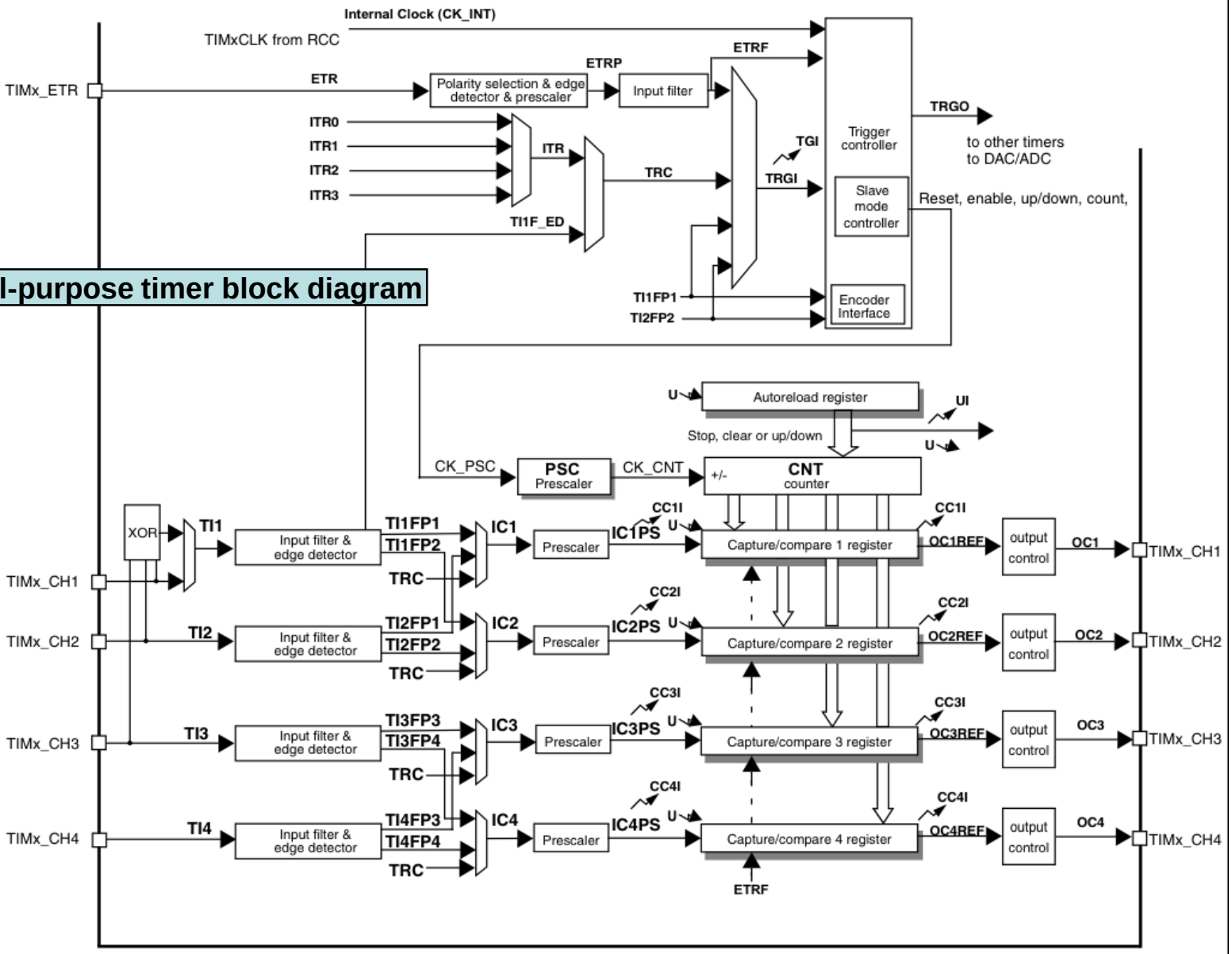
- They may be used for a variety of purposes, including **measuring the pulse lengths of input signals** (input capture) or **generating output waveforms** (output compare and PWM).

- Pulse lengths and waveform periods can be modulated **from a few microseconds to several milliseconds** using the timer prescaler and the RCC clock controller prescalers.

- General-purpose TIMx timer features include:
  - 16-bit up, down, up/down auto-reload counter.
  - 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535.
  - **Up to 4 independent channels** for:
    - Input capture
    - Output compare
    - PWM generation (Edge- and Center-aligned modes)
    - One-pulse mode output

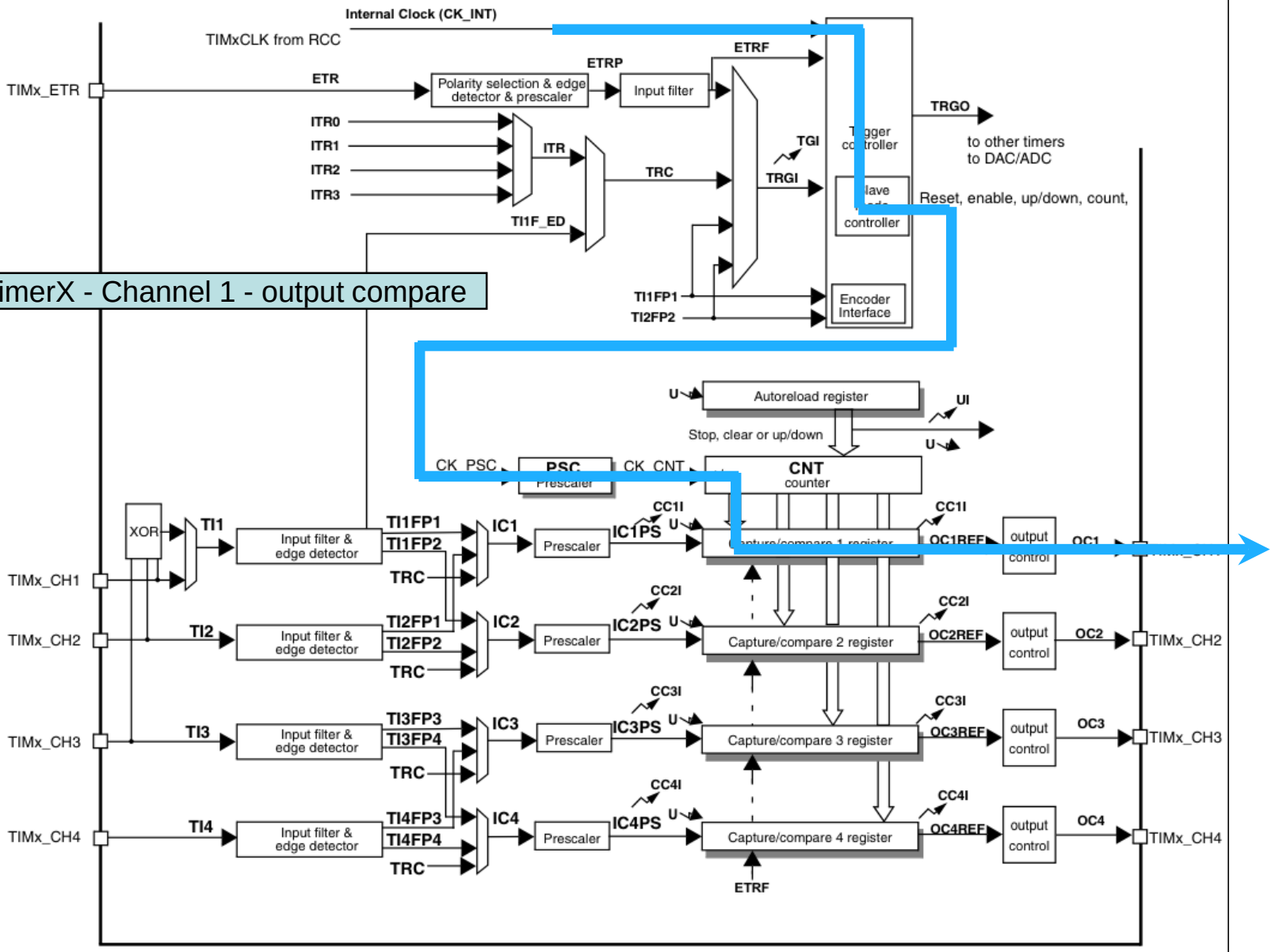
Timers

General-purpose timer block diagram



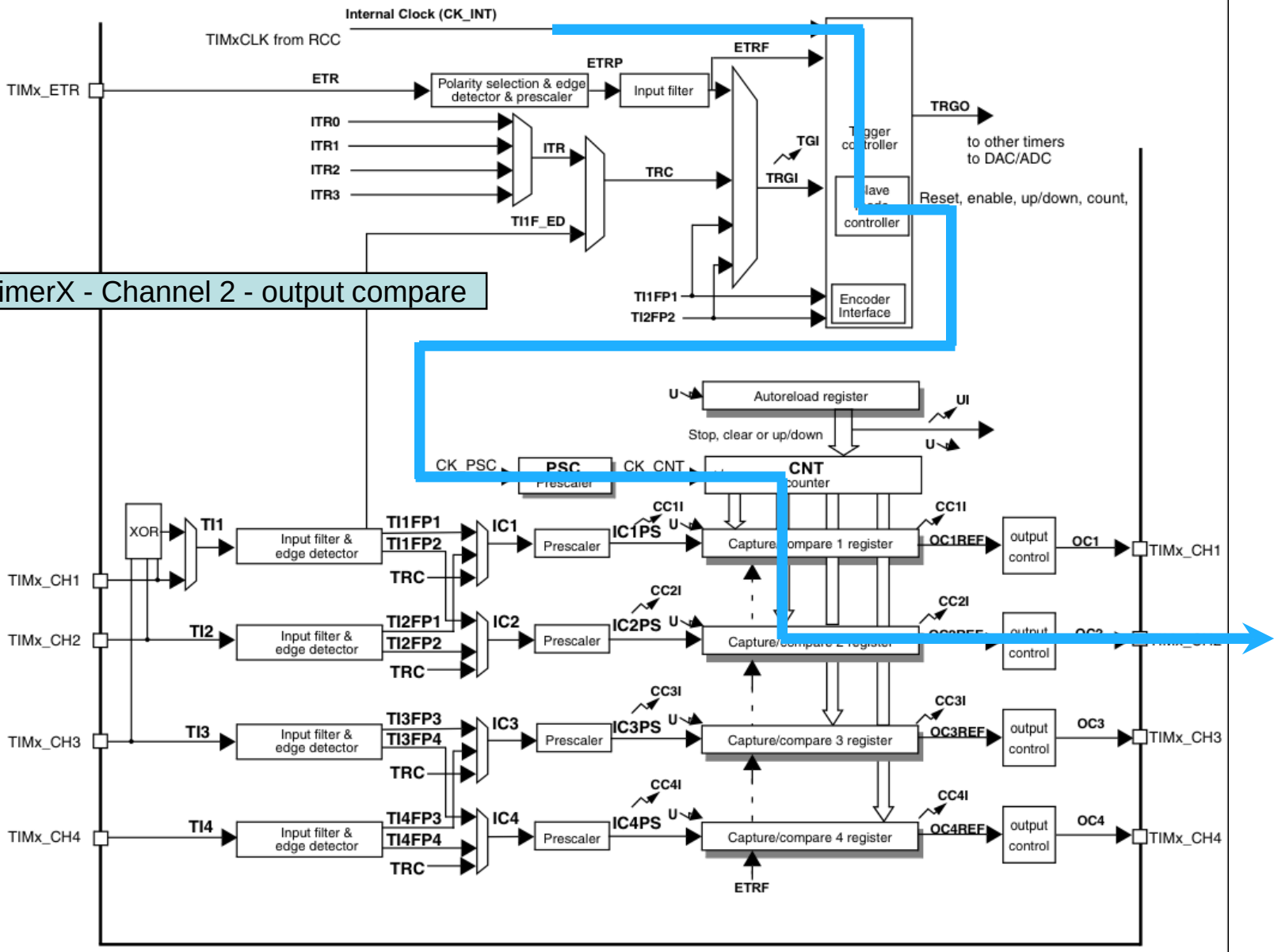
Timers

PATH: TimerX - Channel 1 - output compare



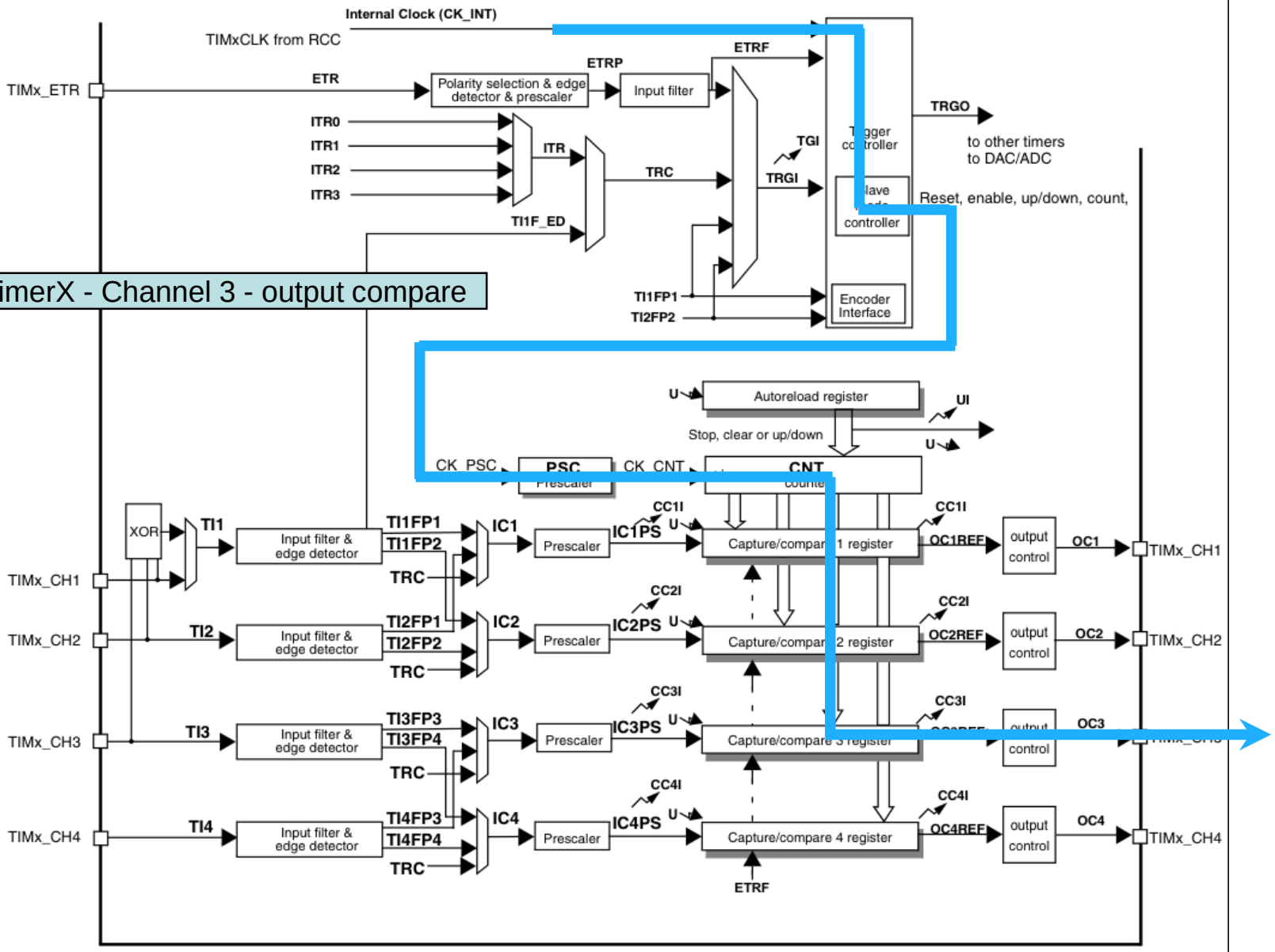
Timers

PATH: TimerX - Channel 2 - output compare



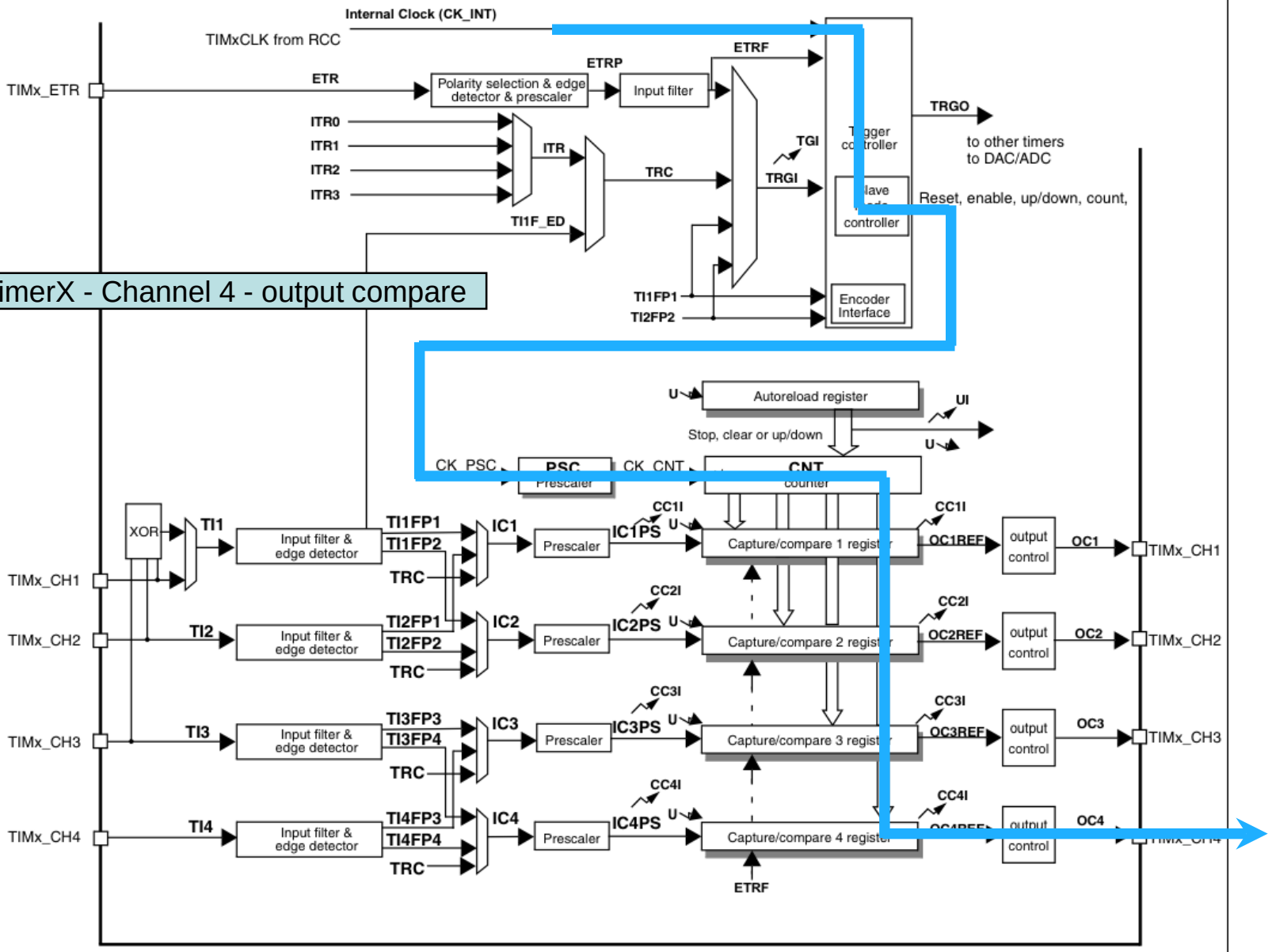
Timers

PATH: TimerX - Channel 3 - output compare

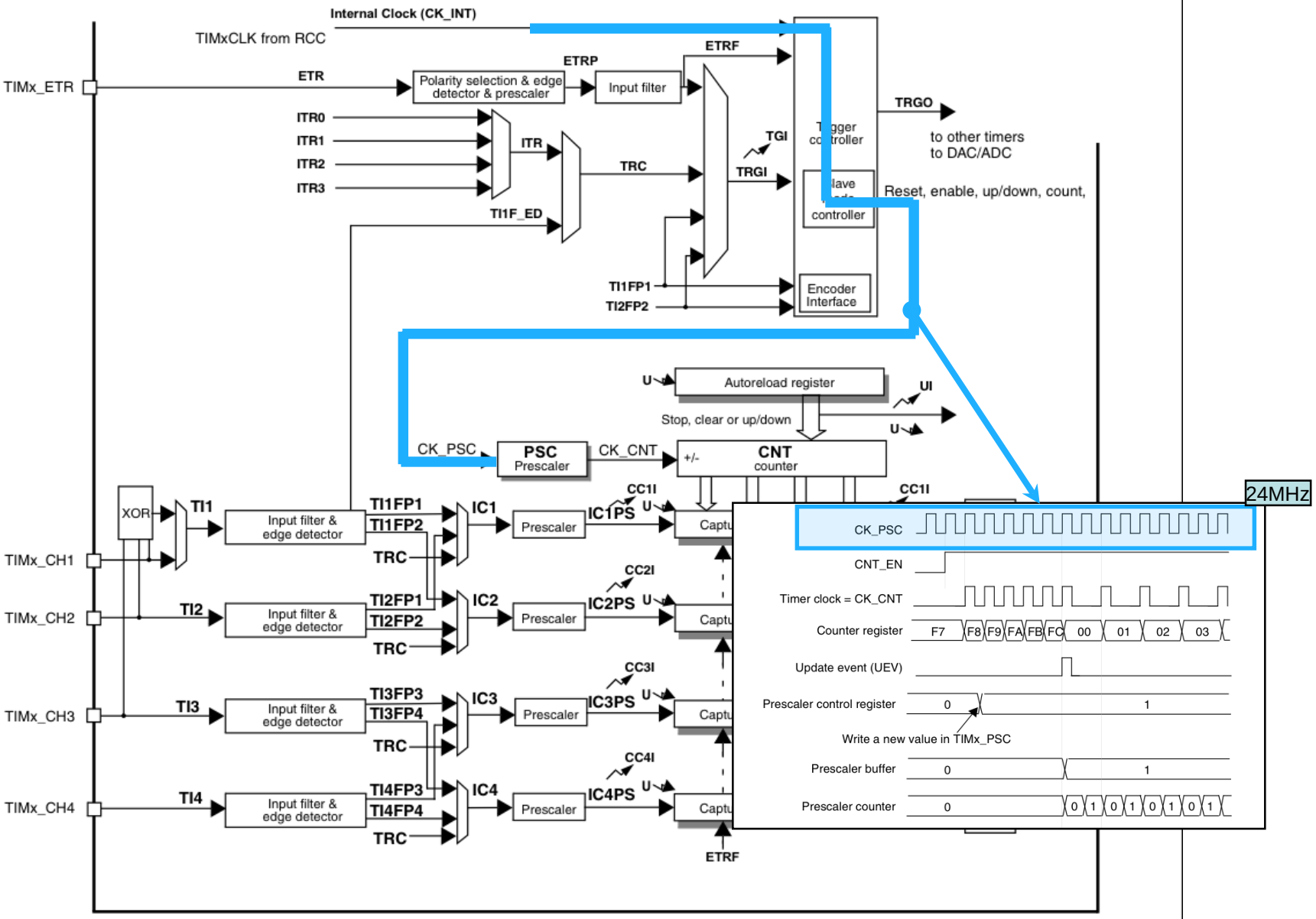


Timers

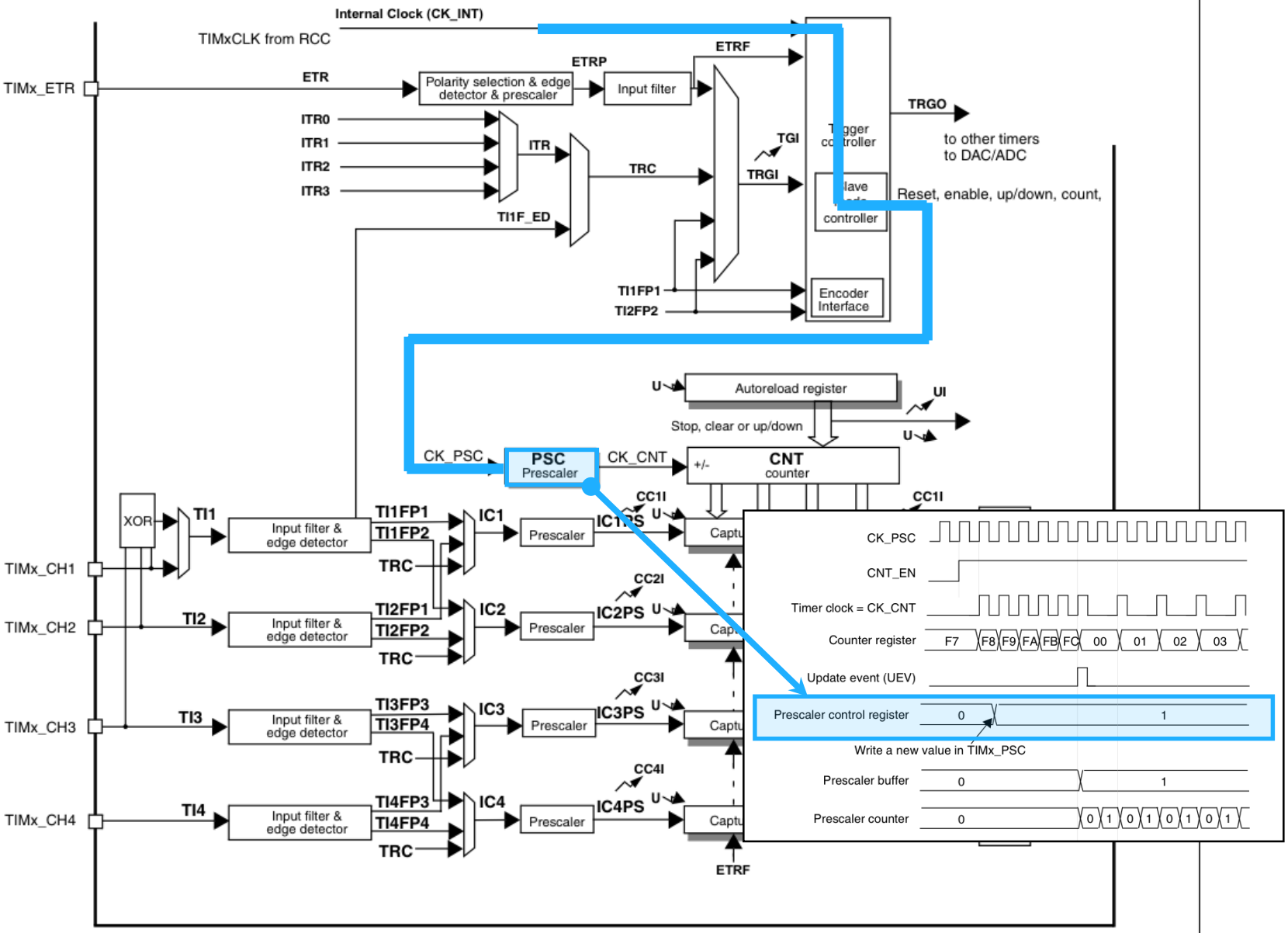
PATH: TimerX - Channel 4 - output compare



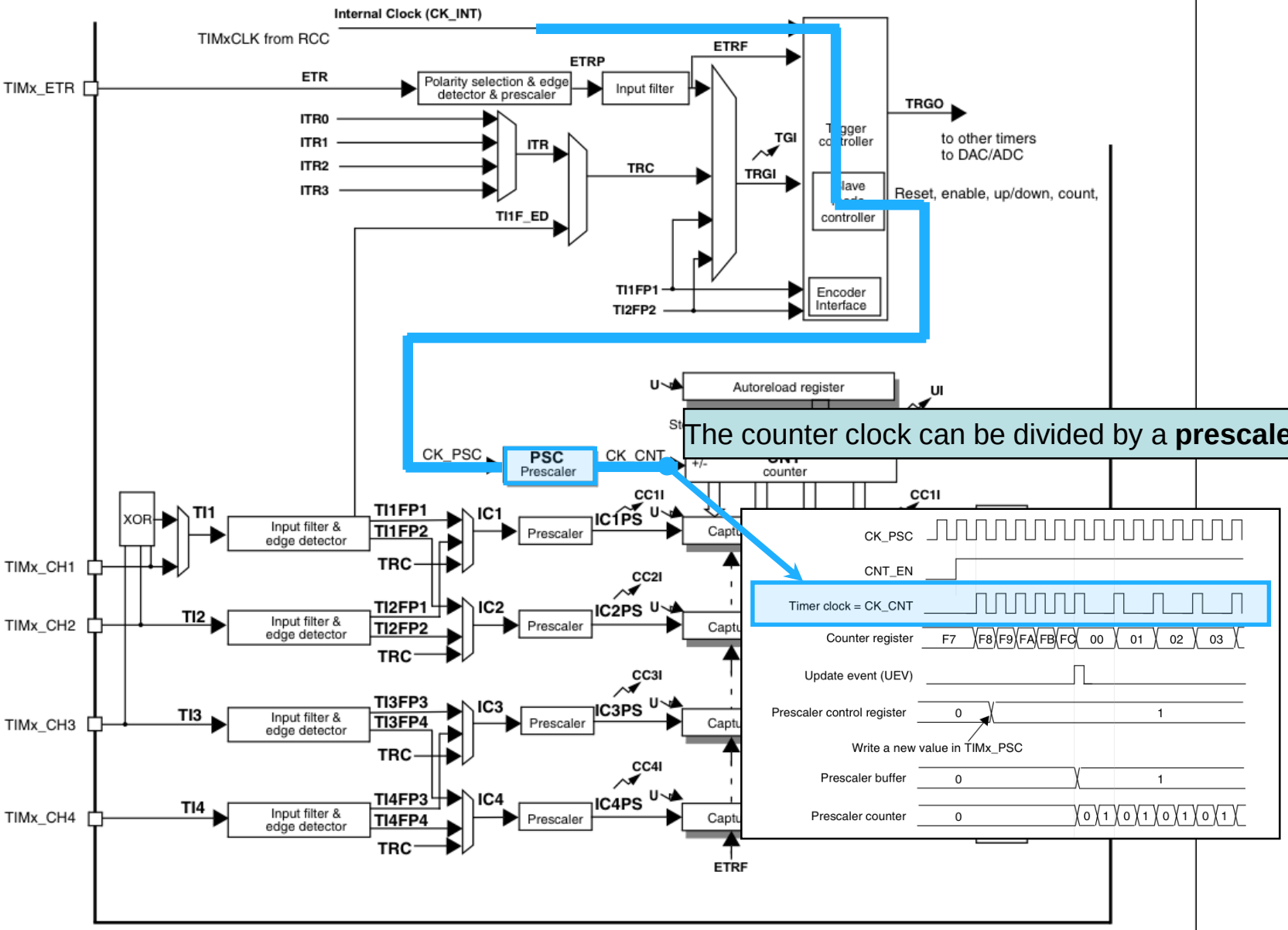
Timers


Internal Clock (CK_INT)
TIMxCLK from RCC
TIMx_ETR
ETR
Polarity selection & edge detector & prescaler
ETRP
Input filter
ETRF
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TI1F_ED
TGI
TRGI
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
controller
Encoder Interface
TI1FP1
TI2FP2
Autoreload register
UI
Stop, clear or up/down
CK_PSC
PSC Prescaler
CK_CNT
+/-
CNT counter
XOR
TI1
TI2
TI3
TI4
Input filter & edge detector
TI1FP1
TI1FP2
TI2FP1
TI2FP2
TI3FP3
TI3FP4
TI4FP3
TI4FP4
IC1
IC2
IC3
IC4
Prescaler
IC1PS
IC2PS
IC3PS
IC4PS
CC1I
CC2I
CC3I
CC4I
TIMx_CH1
TIMx_CH2
TIMx_CH3
TIMx_CH4
24MHz
CK_PSC
CNT_EN
Timer clock = CK_CNT
Counter register
F7
F8
F9
FA
FB
FC
00
01
02
03
Update event (UEV)
Prescaler control register
Write a new value in TIMx_PSC
Prescaler buffer
Prescaler counter

Timers

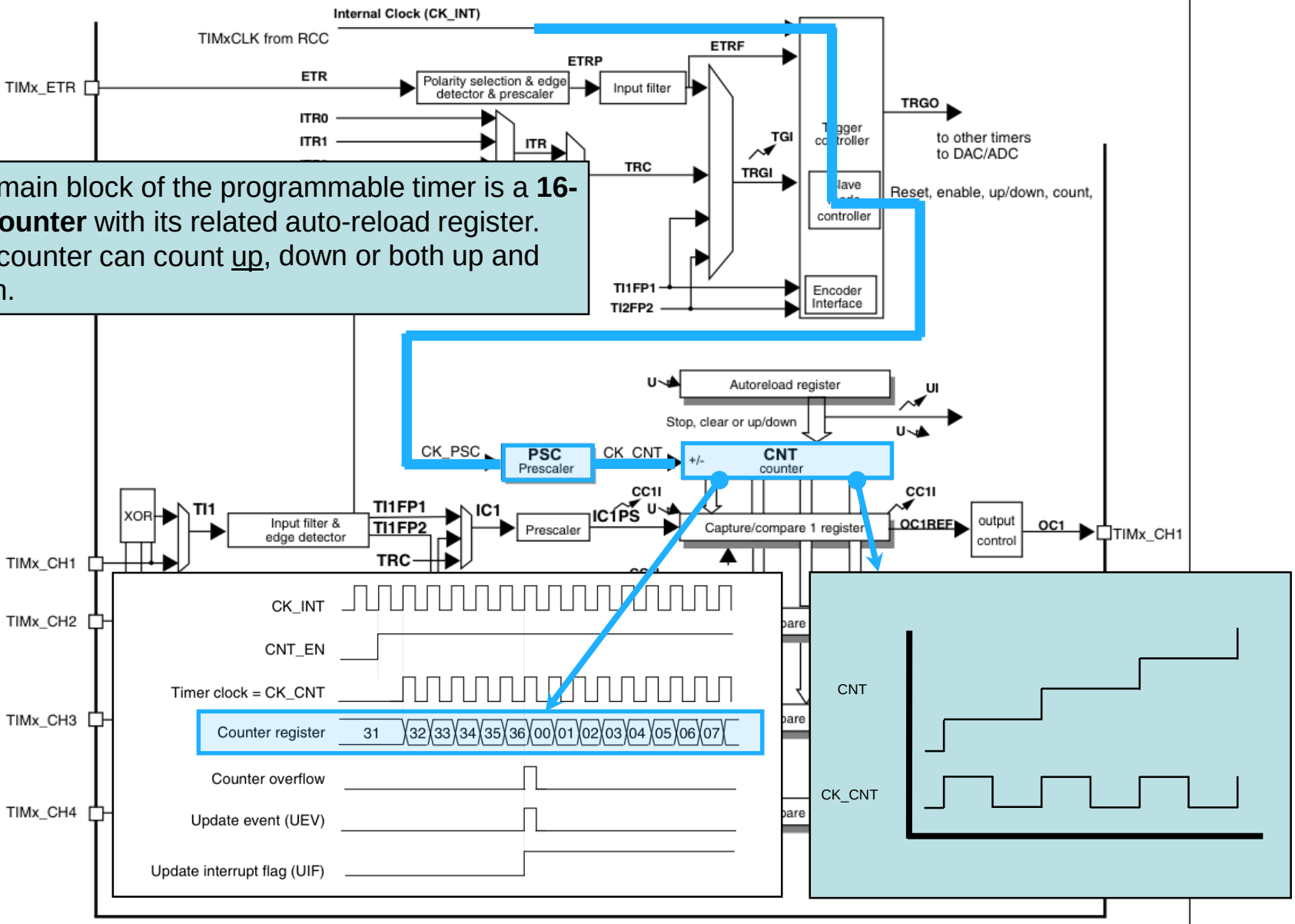

Internal Clock (CK_INT)
TIMxCLK from RCC
TIMx_ETR
ETR
ETRP
ETRF
Polarity selection & edge detector & prescaler
Input filter
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TGI
TRGI
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
controller
TI1F_ED
TI1FP1
TI2FP2
Encoder Interface
Autoreload register
UI
Stop, clear or up/down
CK_PSC
PSC Prescaler
CK_CNT
+/-
CNT counter
CC1I
XOR
TI1
Input filter & edge detector
TI1FP1
TI1FP2
IC1
Prescaler
IC1PS
TRC
TIMx_CH1
CC2I
TI2
TI2FP1
TI2FP2
IC2
IC2PS
TIMx_CH2
CC3I
TI3
TI3FP3
TI3FP4
IC3
IC3PS
TIMx_CH3
CC4I
TI4
TI4FP3
TI4FP4
IC4
IC4PS
TIMx_CH4
CK_PSC
CNT_EN
Timer clock = CK_CNT
Counter register
F7 F8 F9 FA FB FC 00 01 02 03
Update event (UEV)
Prescaler control register
0 1
Write a new value in TIMx_PSC
Prescaler buffer
Prescaler counter
0 1 0 1 0 1 0 1

Timers


Internal Clock (CK_INT)
TIMxCLK from RCC
TIMx_ETR
ETR
ETRP
ETRF
Polarity selection & edge detector & prescaler
Input filter
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TGI
TRGI
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
controller
TI1F_ED
TI1FP1
TI2FP2
Encoder Interface
Autoreload register
UI
CK_PSC
PSC Prescaler
CK_CNT
counter
The counter clock can be divided by a prescaler.
XOR
TI1
TI2
TI3
TI4
Input filter & edge detector
TI1FP1
TI1FP2
TI2FP1
TI2FP2
TI3FP3
TI3FP4
TI4FP3
TI4FP4
IC1
IC2
IC3
IC4
Prescaler
IC1PS
IC2PS
IC3PS
IC4PS
CC1I
CC2I
CC3I
CC4I
TIMx_CH1
TIMx_CH2
TIMx_CH3
TIMx_CH4
ETRF
CK_PSC
CNT_EN
Timer clock = CK_CNT
Counter register
F7 F8 F9 FA FB FC 00 01 02 03
Update event (UEV)
Prescaler control register
0 1
Write a new value in TIMx_PSC
Prescaler buffer
0 1
Prescaler counter
0 0 1 0 1 0 1 0 1

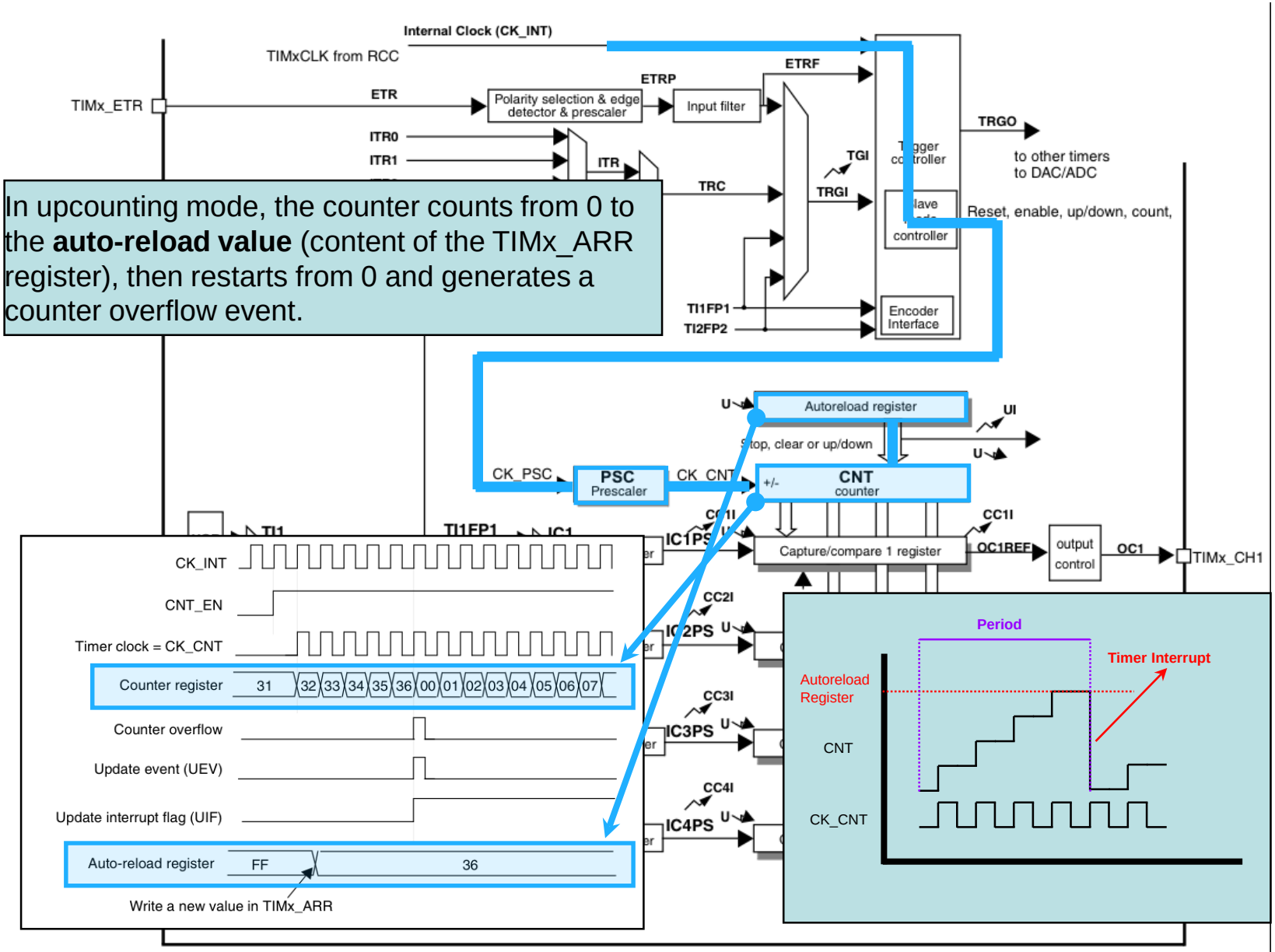
Timers

The main block of the programmable timer is a **16-bit counter** with its related auto-reload register. The counter can count up, down or both up and down.



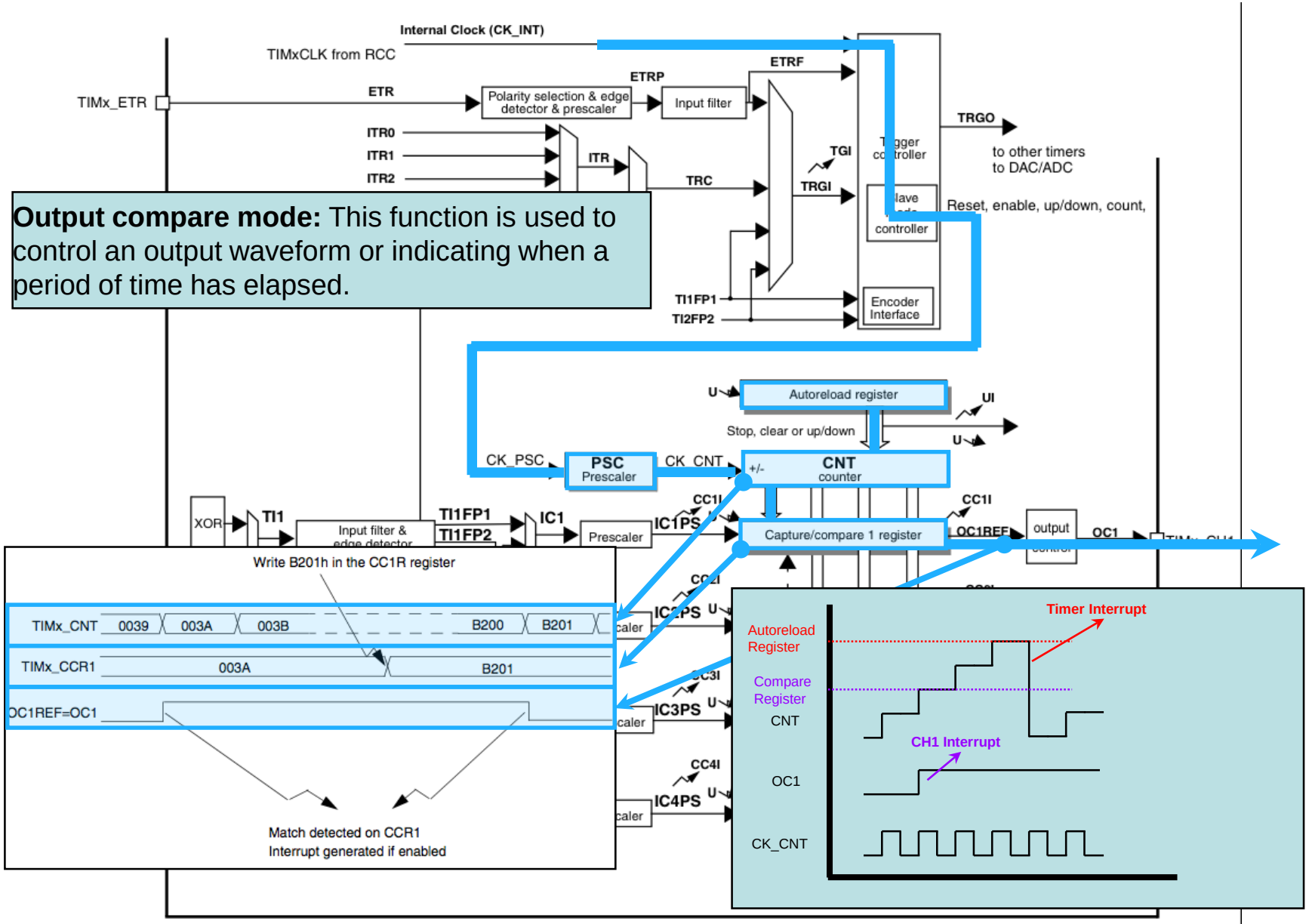
Timers

In upcounting mode, the counter counts from 0 to the **auto-reload value** (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

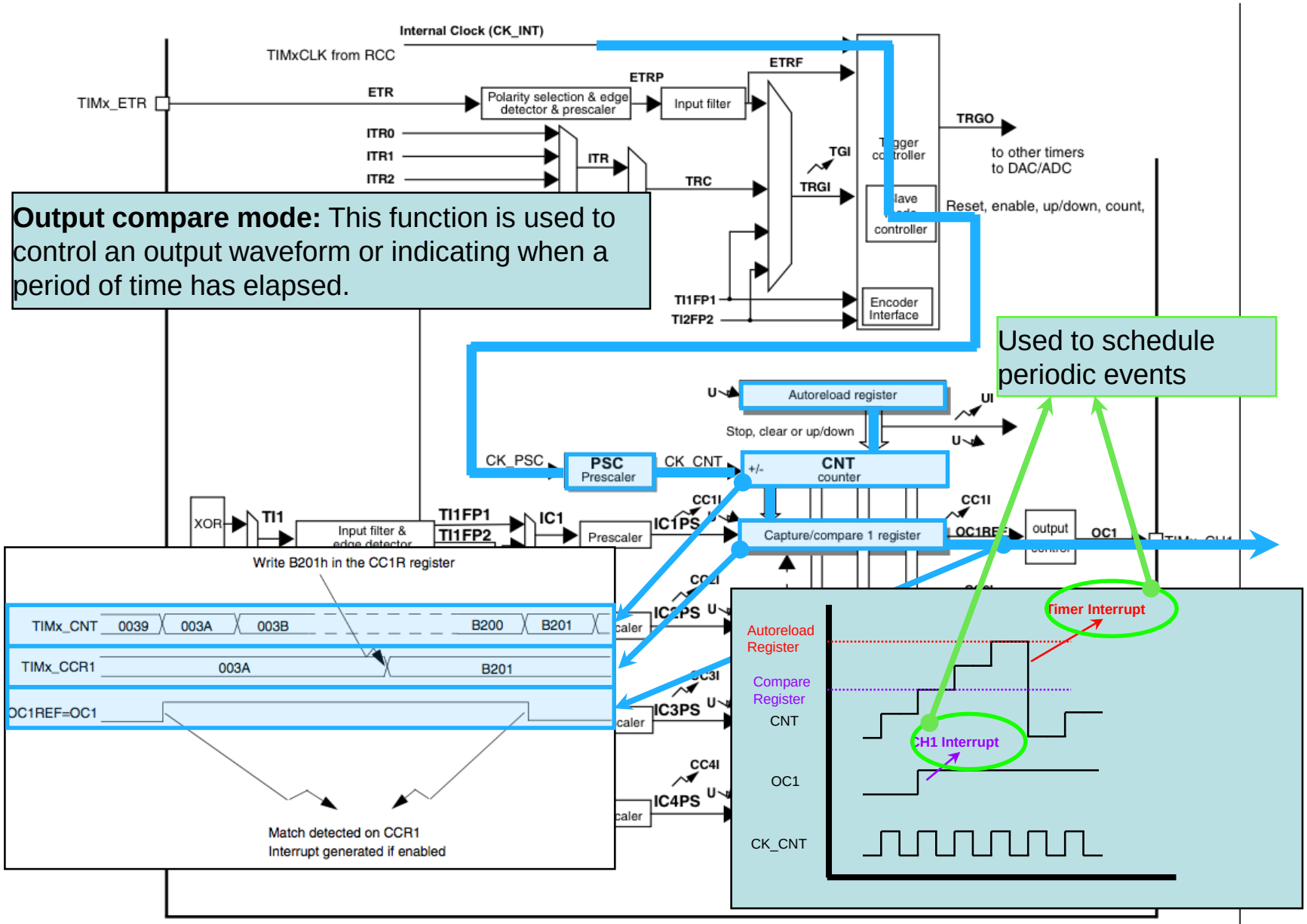


Timers

Output compare mode: This function is used to control an output waveform or indicating when a period of time has elapsed.

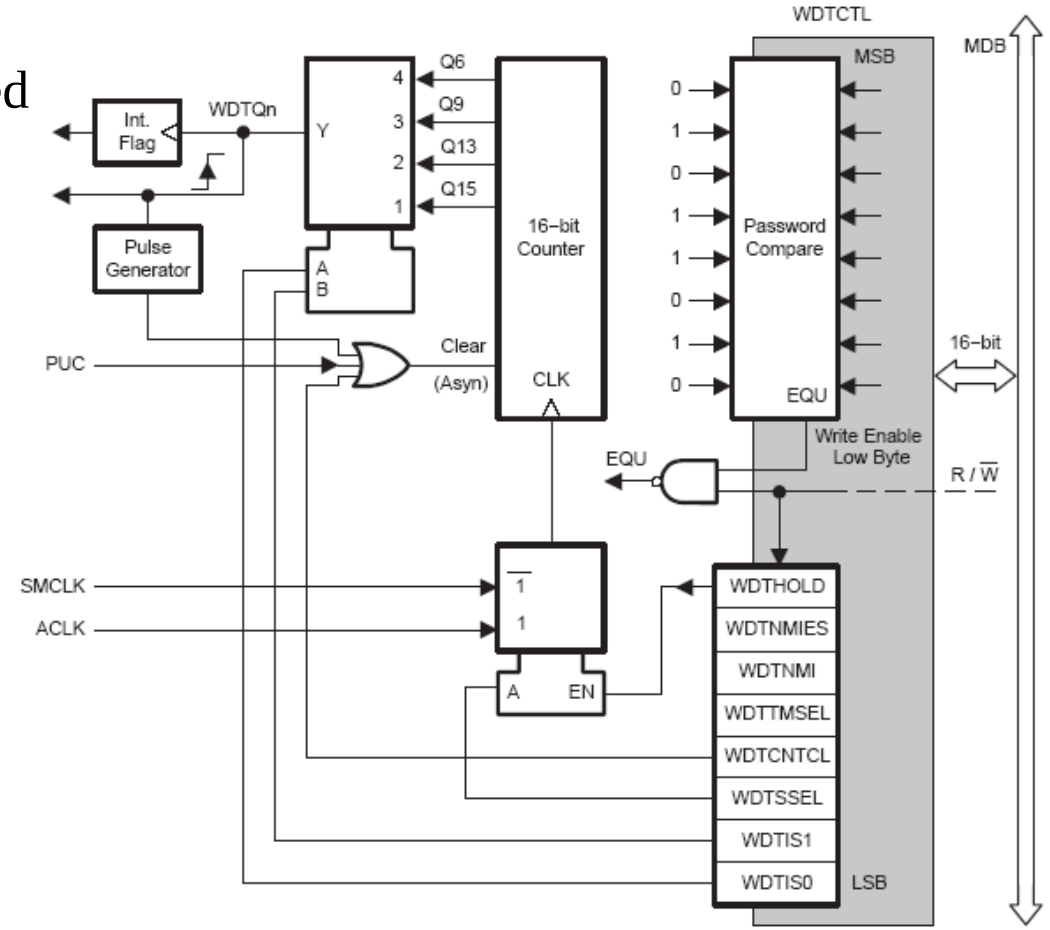


Timers


Internal Clock (CK_INT)
TIMxCLK from RCC
TIMx_ETR
ETR
ETRP
ETRF
Polarity selection & edge detector & prescaler
Input filter
ITR0
ITR1
ITR2
ITR
TRC
TGI
TRGI
TRGO
to other timers
to DAC/ADC
Reset, enable, up/down, count,
controller
TI1FP1
TI2FP2
Encoder Interface
Output compare mode: This function is used to control an output waveform or indicating when a period of time has elapsed.
Used to schedule periodic events
Autoreload register
Stop, clear or up/down
CK_PSC
PSC Prescaler
CK_CNT
CNT counter
XOR
TI1
Input filter & edge detector
TI1FP1
TI1FP2
IC1
Prescaler
IC1PS
CC1I
Capture/compare 1 register
OC1REF
output
OC1
Write B201h in the CC1R register
TIMx_CNT
0039
003A
003B
B200
B201
TIMx_CCR1
003A
B201
OC1REF=OC1
Match detected on CCR1
Interrupt generated if enabled
IC3PS
IC4PS
CC4I
Autoreload Register
Compare Register
CNT
OC1
CK_CNT
Timer Interrupt
CH1 Interrupt

Watchdog Timer (WDT)

- WDT module performs a controlled system restart after a software problem occurs
  - Can serve as an interval timer (generates interrupts)
  - WDT Control register is password protected



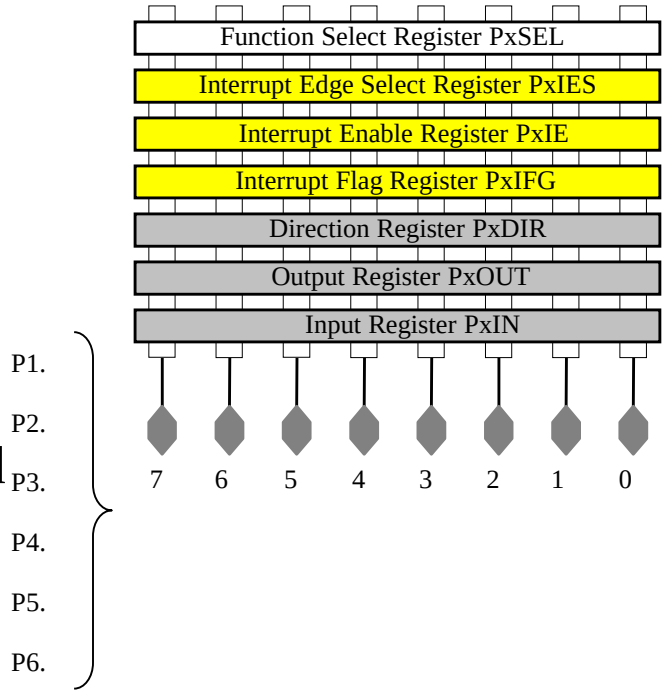
Watchdog timer (WDT)

- The 16-bit WDT module can be used in:
  - **Supervision mode:**
    - Ensure the correct working of the software application;
    - Perform a PUC;
    - Generate an interrupt request after the counter overflows.
  - **Interval timer:**
    - Independent interval timer to perform a “standard” interrupt upon counter overflow periodically;
    - Upper counter (WDTCNT) is not directly accessible by software;
    - Control and the interval time selecting WDTCTL register;

Digital I/O

Independently programmable individual I/Os

- Several ports
- Each has 8 I/O pins
- Each pin can be configured as input or output
- Some pins can be configured to assert an interrupt request

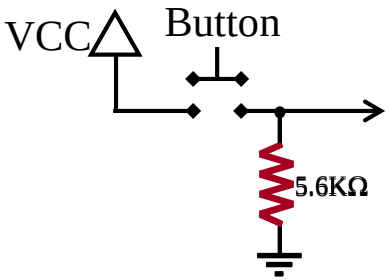


Port1 Port2	Port3 … Port6
yes	yes
yes	no
yes	no
yes	no
yes	yes
yes	yes
yes	yes

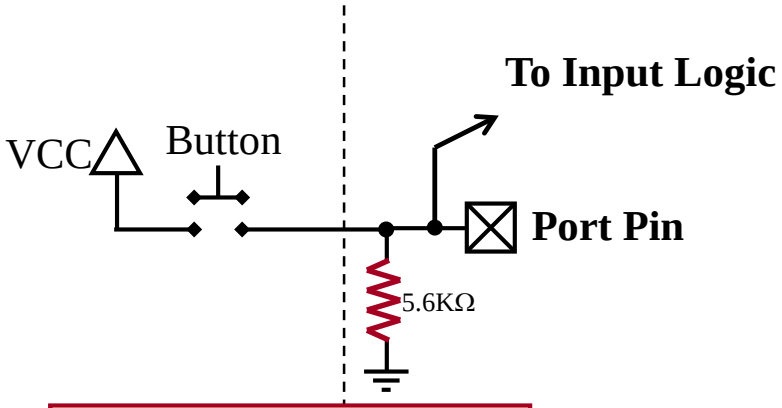
GPIO – General Purpose I/O

- Avoid floating inputs!!!

Use a pull-up/down resistor, GND, or internal programmable logic



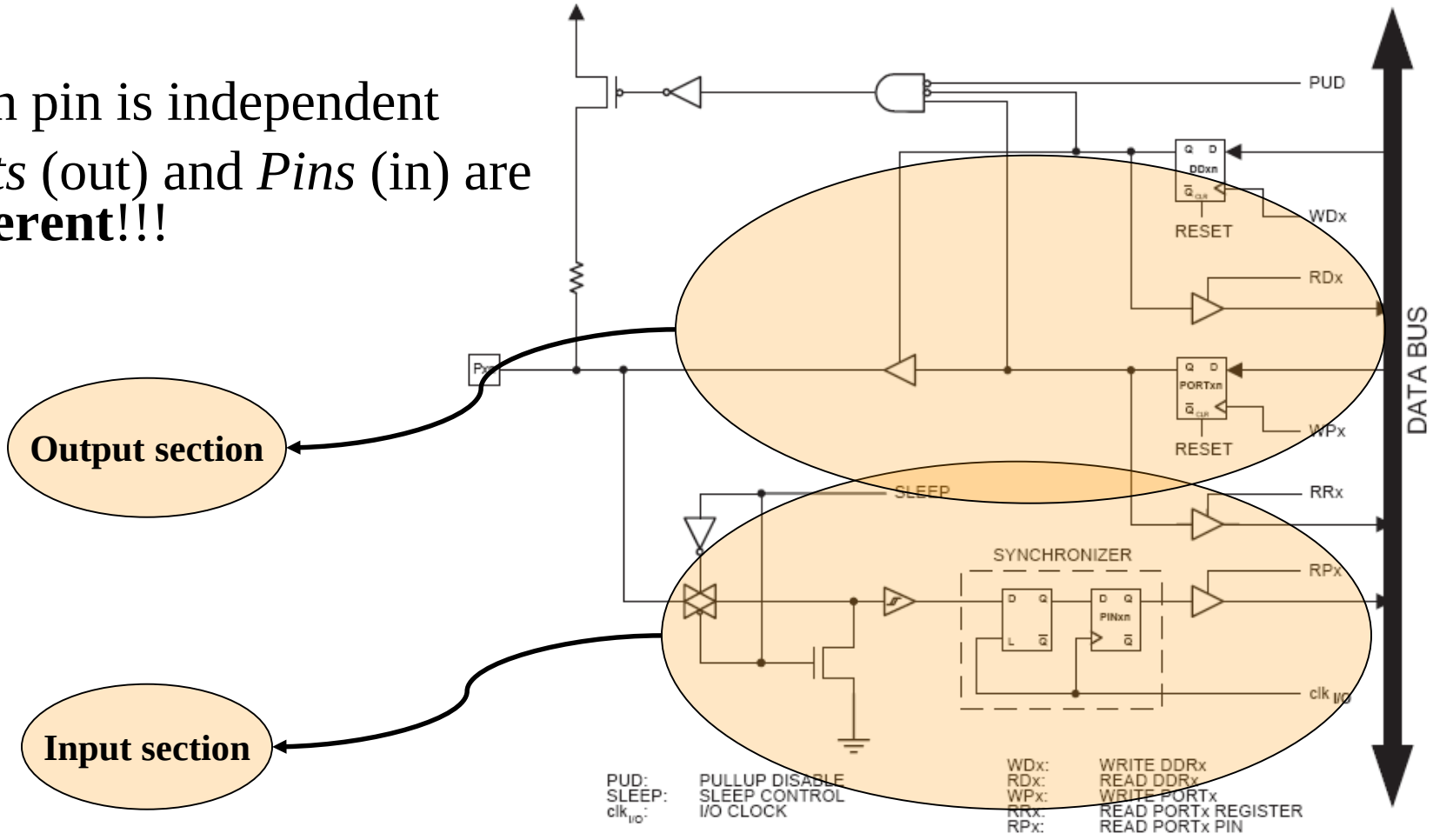
Button produces either Vcc or Floating input. Adding a pull-down resistor fixes it.



Some ports have internal programmable resistors

GPIO - Inside Inputs/Outputs

- Each pin is independent
- *Ports* (out) and *Pins* (in) are **different**!!!



Interfaces

- Several protocols for inter-chip communication

  UART, I^2C, SPI, USB,…

- Serial communication protocols
- Meant for short distances “inside the box”
- Low complexity
- Low cost
- Low speed (a few Mbps at the fastest)
- Serial communication is employed where it is not practical, either in physical or cost terms, to move data in parallel between systems.

I^2C

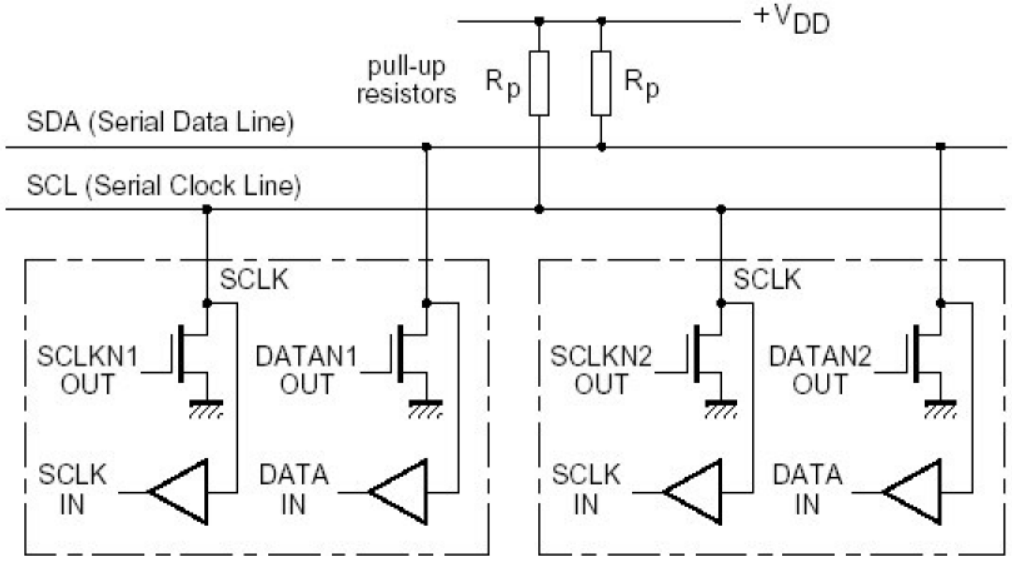
- Shorthand for an “Inter-integrated circuit” bus
- I2C devices include EEPROMs, thermal sensors, and real-time clocks
- Used as a control interface to signal processing devices that have separate data interfaces, e.g. RF tuners, video decoders and encoders, and audio processors.
- I^2C bus has three speeds:
  - Slow (under 100 Kbps)
  - Fast (400 Kbps)
  - High-speed (3.4 Mbps) – I^2C v.2.0
- Limited to about 3 meters for moderate speeds

I^2C (Inter-Integrated Circuit) protocol

- Communications is always initiated and completed by the master, which is responsible for generating the clock signal;
- In more complex applications, I^2C can operate in multi-master mode;
- The slave selection by the master is made using the seven-bit address of the target slave;
- The master (in transmit mode) sends:
  - Start bit;
  - 7-bit address of the slave it wishes to communicate with;
  - A single bit representing whether it wishes to write (0) to or read (1) from the slave;
  - The target slave will acknowledge its address.

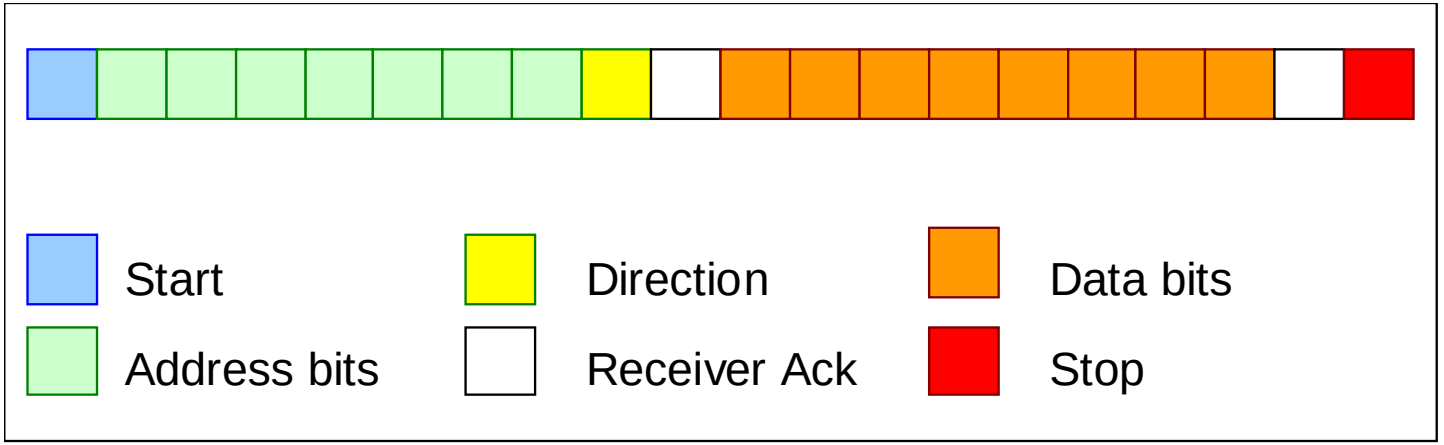
I²C Bus Configuration

- 2-wire serial bus – Serial data (SDA) and Serial clock (SCL)
- Half-duplex, synchronous, multi-master bus
- No chip select or arbitration logic required
- Lines pulled high via resistors, pulled down via open-drain drivers (wired-AND, avoid short circuit among the bus)

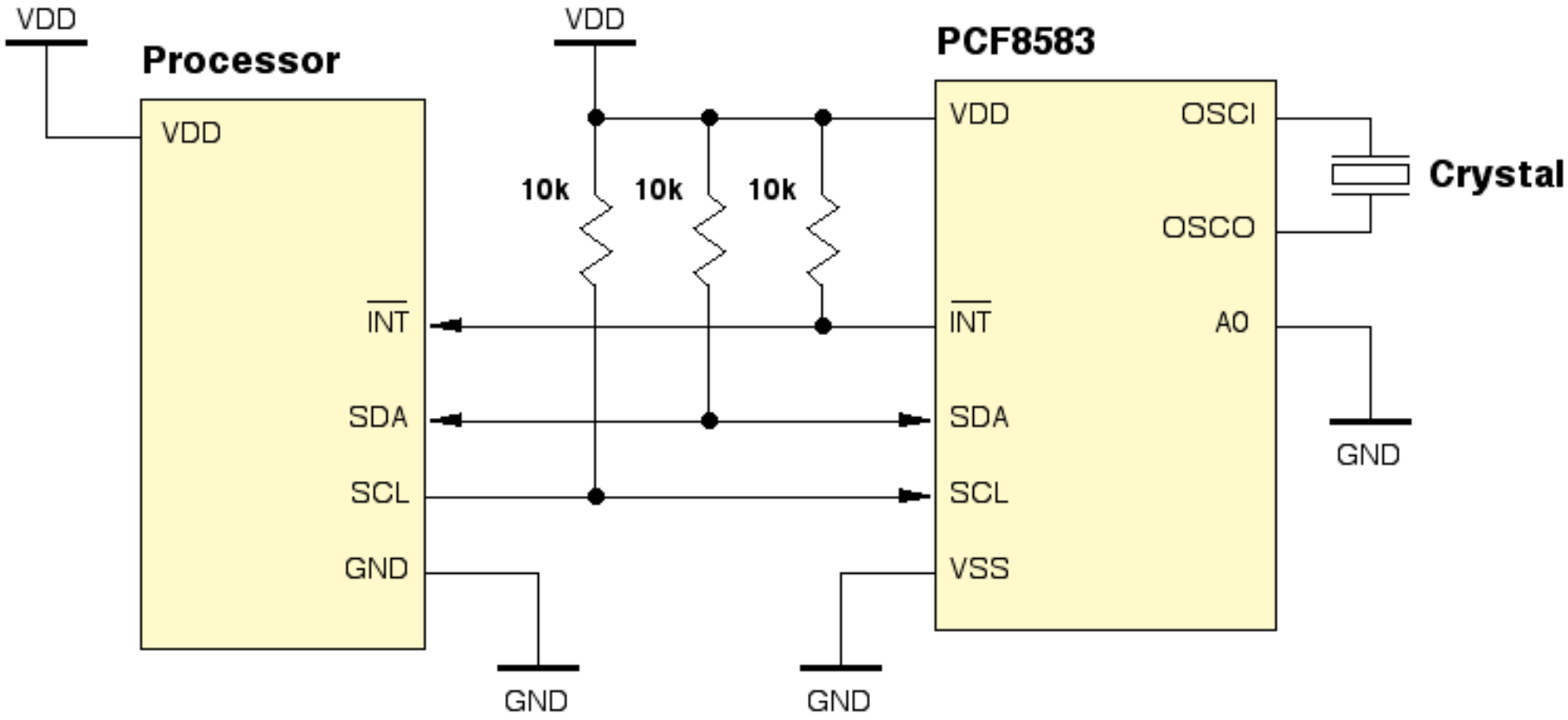


I²C Features

- "Clock stretching" – when the slave (receiver) needs more time to process a bit, it can pull SCL low. The master waits until the slave has released SCL before sending the next bit.
- "General call" broadcast – addresses every device on the bus
- 10-bit extended addressing for new designs. 7-bit addresses all exhausted



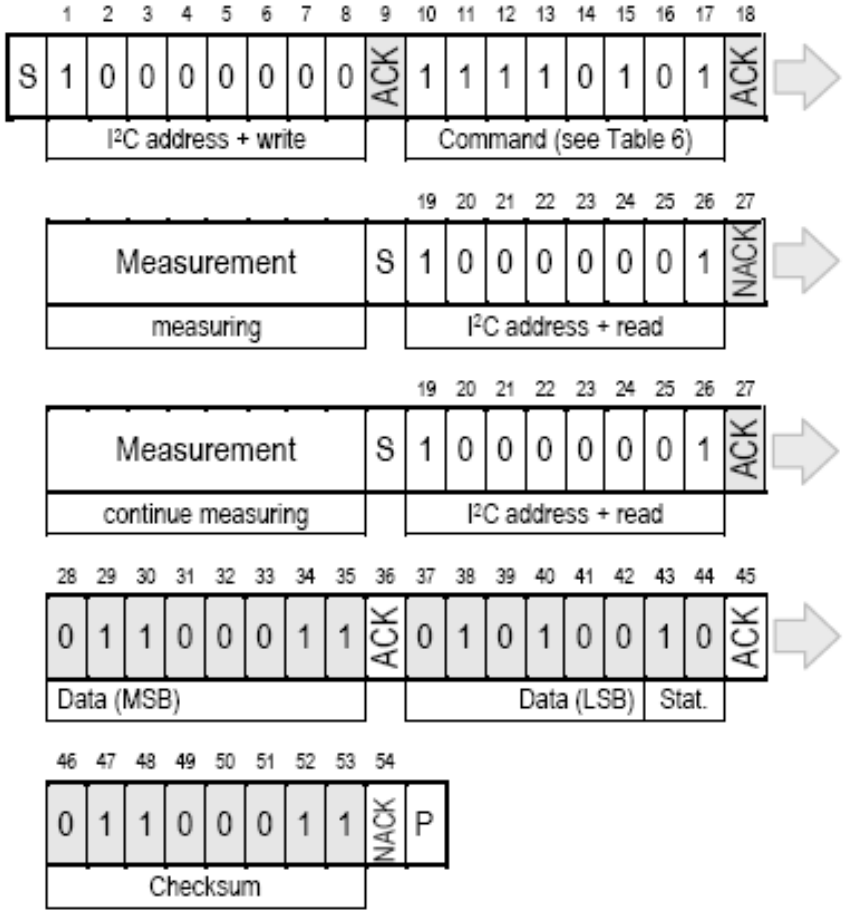
Example
I²C bridge



Sensors data acquisition example

Realization with digital sensor:

- Data acquisition procedure:



SPI

- Shorthand for “Serial Peripheral Interface”
- Defined by Motorola on the MC68HCxx line of microcontrollers
- Generally faster than I^2C, capable of several Mbps

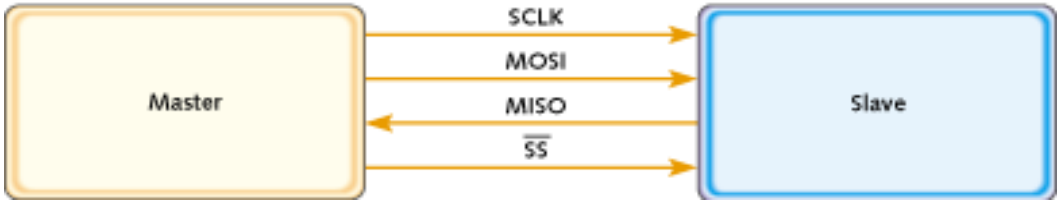
Applications:

- Like I^2C, used in EEPROM, Flash, and real time clocks
- Better suited for “data streams”, i.e. ADC converters
- Full duplex capability, i.e. communication between a codec and digital signal processor

Serial Peripheral Interface (SPI) protocol

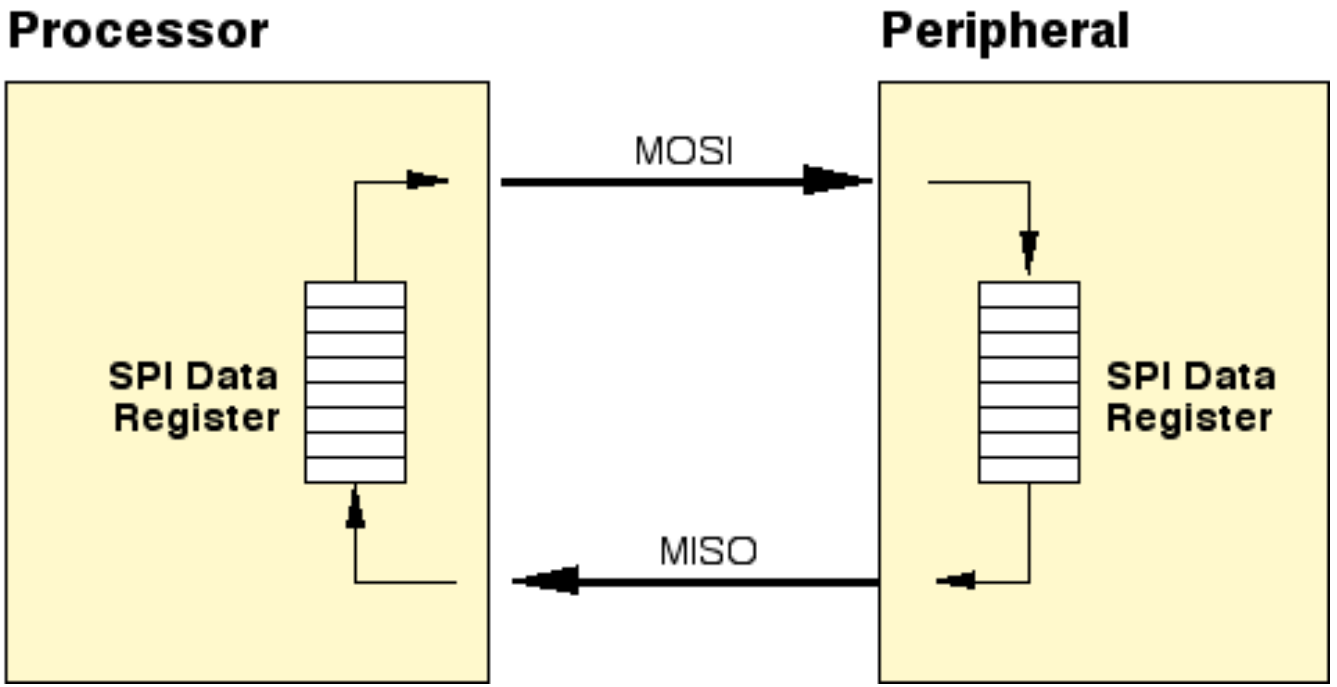
- Supports only one master;
- Can support more than a slave;
- Short distance between devices, e.g. on a printed circuit boards (PCBs);
- Special attention needs to be observed to the polarity and phase of the clock signal;
- The master sends data on one edge of clock and reads data on the other edge. Therefore, it can send/receive at the same time.

SPI Bus Configuration



- Synchronous serial data link operating at full duplex
- Master/slave relationship
- 2 data signals:
  - MOSI – master data output, slave data input
  - MISO – master data input, slave data output
- 2 control signals:
  - SCLK – clock
  - /SS – slave select (no addressing)

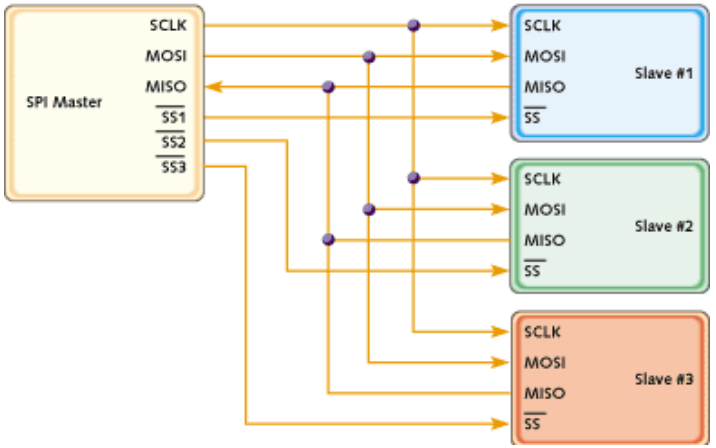
SPI structure



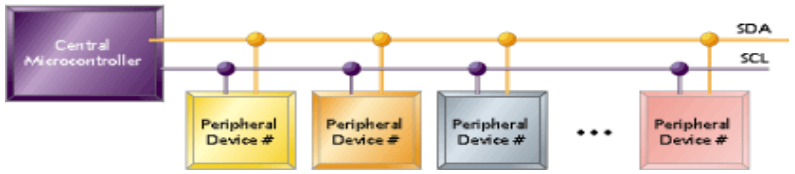
- As the register transmits the byte to the slave on the MOSI signal line, the slave transfers the contents of *its* shift register back to the master on the MISO signal line, exchanging the contents of the two shift registers.

SPI vs. I²C

- For point-to-point, SPI is simple and efficient
  - Less overhead than I²C due to lack of addressing, plus SPI is full duplex.
- For multiple slaves, each slave needs separate slave select signal
  - SPI requires more effort and more hardware than I²C



SPI



I²C

UART

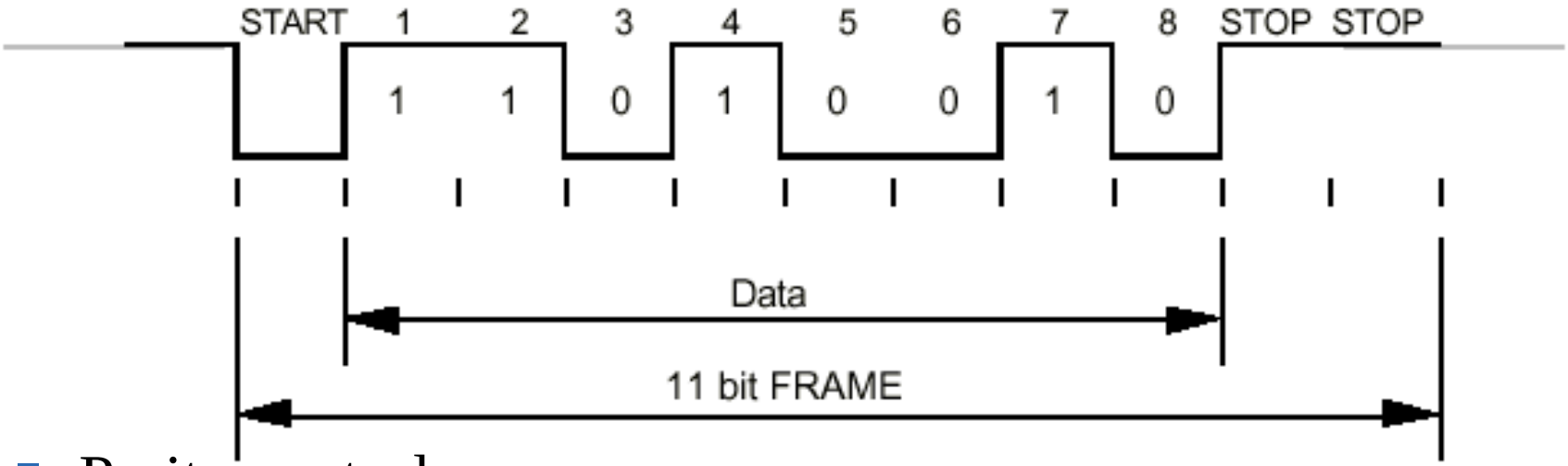
- Shorthand for “Universal Asynchronous Receiver-Transmitter “
- A UART’s transmitter is essentially just a parallel-to-serial converter with extra features.
- The UART bus is a full-duplex bus.
- The essence of the UART transmitter is a shift register that is loaded in parallel, and then each bit is sequentially shifted out of the device on each pulse of the serial clock.
- Application:
  - Communication between microprocessors, pc
  - Used to interface the microcontroller with others transmission bus as: RS232, RS485, USB, CAN BUS, KNX, LonWorks ecc.
  - Used to connect microntroller with modem and transceiver as: telephone modem, Bluetooth, WIFi, GSM/GPRS/HDPSA

UART

- Asynchronous serial devices, such as UARTs, do not share a common clock
- Each device has its own, local clock.
- The devices must operate at exactly the same frequency.
- Logic (within the UART) is required to detect the phase of the transmitted data and *phase lock* the receiver's clock to this.
- Bitrate: 2400, 19200, 57600,115200, 921600…
- One of the problems associated with serial transmission is reconstructing the data at the receiving end, because the clock is not transmitted.
- Difficulties arise in detecting boundaries between bits.

UART

- The transmission format uses:
  - 1 start bit at the beginning
  - Settable 5,6,7,8 data bits string length
  - Settable 1 or 0 even/odd parity bit control
  - settable 1, 1.5, 2 stop bits end of each frame.

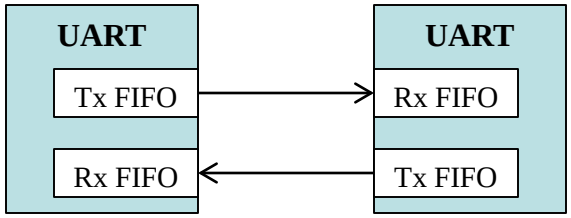


- Parity control
  - The parity bit control is accordingly set to 0 or 1 to have and odd number of frame 1 bits in odd parity either an even number of frame 1 bits in the even parity
  - The control can detect 1 bit error in the frame

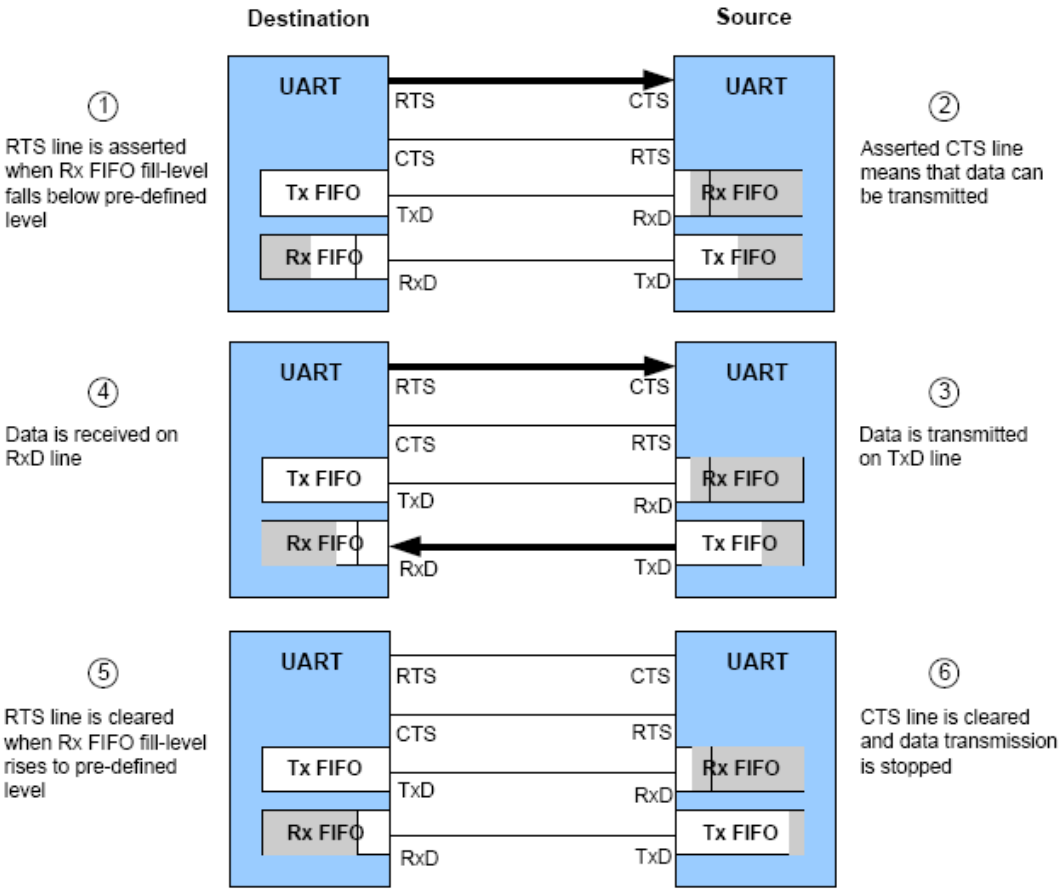
UART transmission

UART can transmit either with 2 or 4 wires

- 2 wires mode has transmit and receive lines



- 4 wires mode has transmit and receive lines plus 2 handshake signals, RTS request to send, CTS clear to send



Analog to Digital Converters

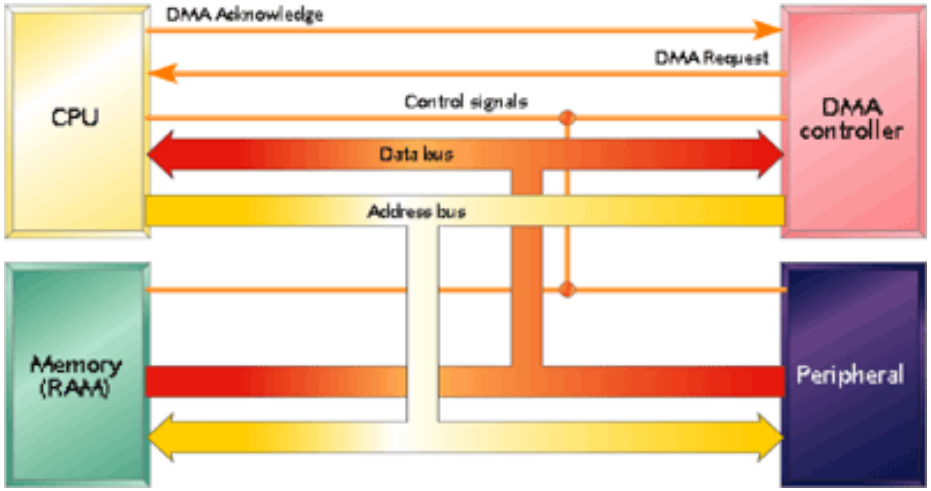
- Most engineering applications require some form of data processing: measurement, control, calculation, communication or data recording;

- These operations, either grouped or isolated, are built into the measuring instruments;

- The measuring equipment must maintain:
  - Compatibility and communication between measuring devices;
  - Acceptable error margin;
  - Noise and interference immunity;
  - Predictable measurement uncertainty;
  - Suitable type of control (analogue/digital);
  - Mathematical processing capacity;
  - …

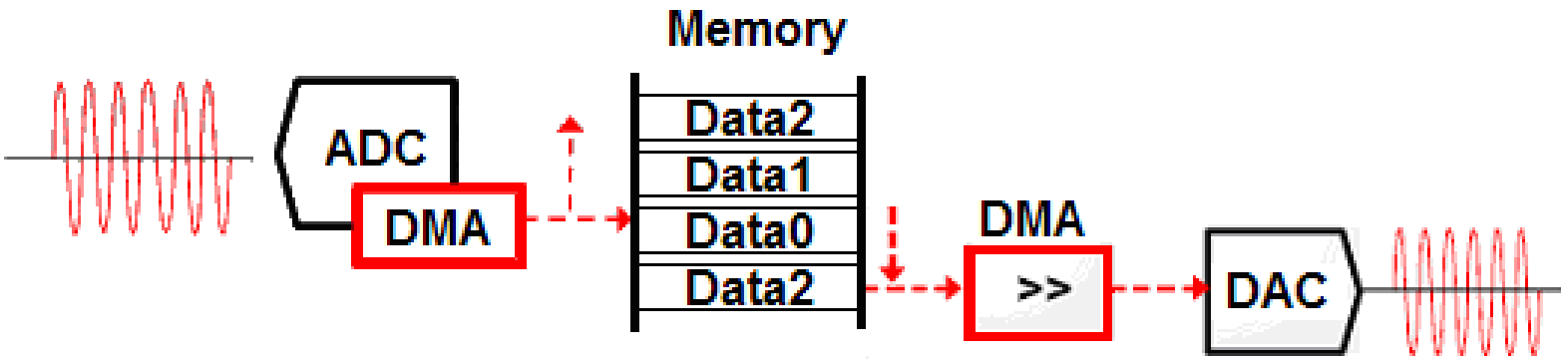
Direct Memory Access

- Direct Memory Access (DMA) allows memory-to-memory or peripheral-to-memory communication without the intervention of the main CPU.
  - The CPU initiates the data transfer and then can do other tasks or go in stand-by
  - The DMA controller handles the actual data stream and sends an interrupt when done

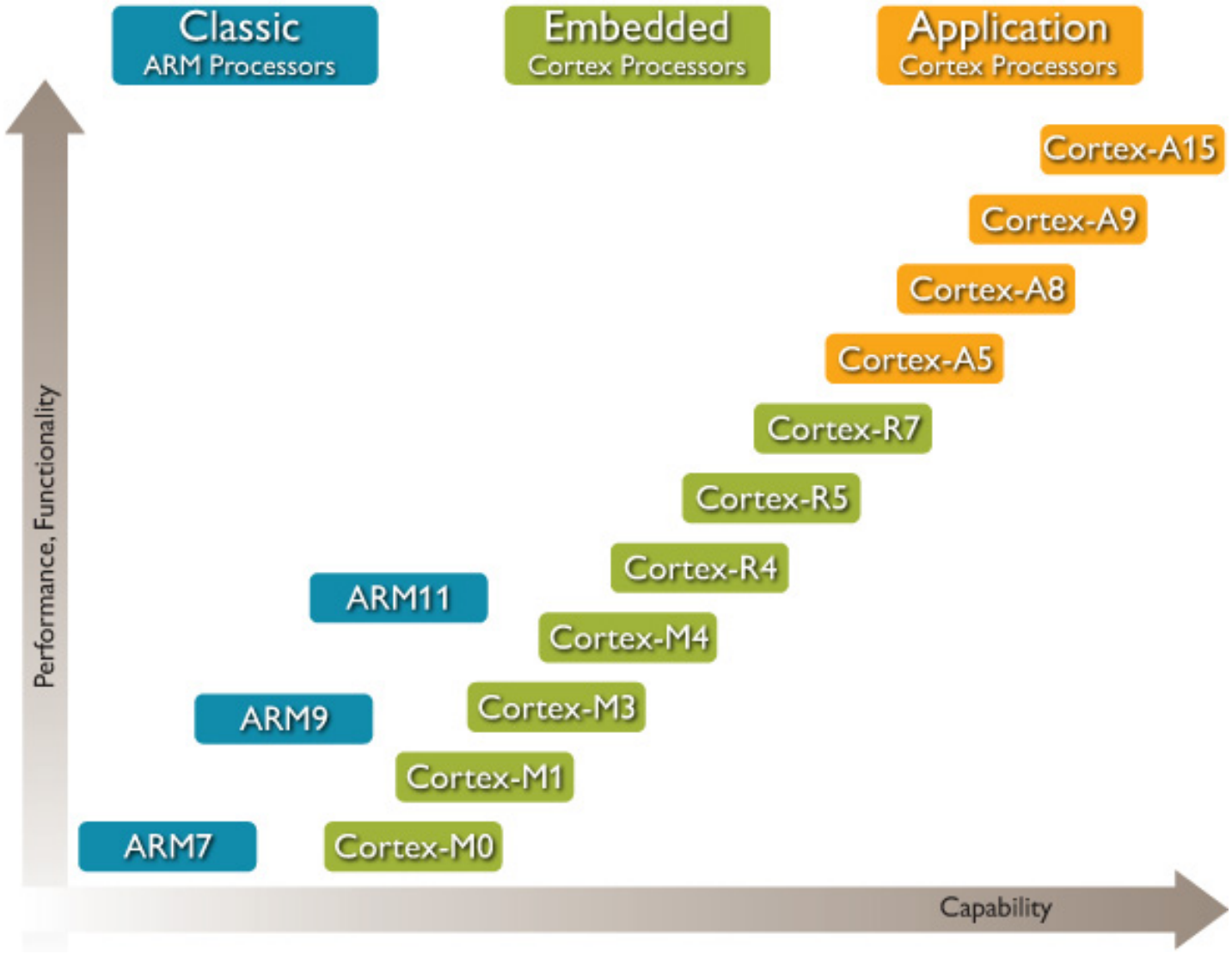


Direct Memory Access

- Concept of DMA: move functionality to peripherals
  - Peripherals use less current than the CPU;
  - Delegating control to peripherals allows the CPU to shut down (saves power) or perform other tasks (increase processing capabilities);
  - “Intelligent” peripherals are more capable, providing a better opportunity for CPU shutoff;
  - DMA can be enabled for repetitive data handling, increasing the throughput of peripheral modules;
  - Minimal software requirements and CPU cycles.



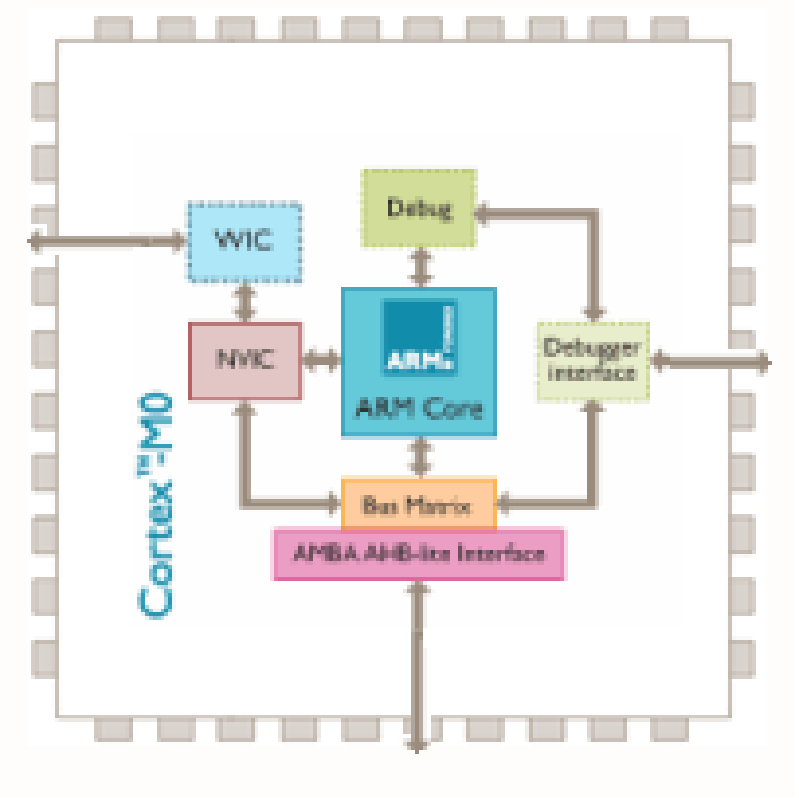
ARM Processors Families



STM32 ARM® Cortex™-M Family

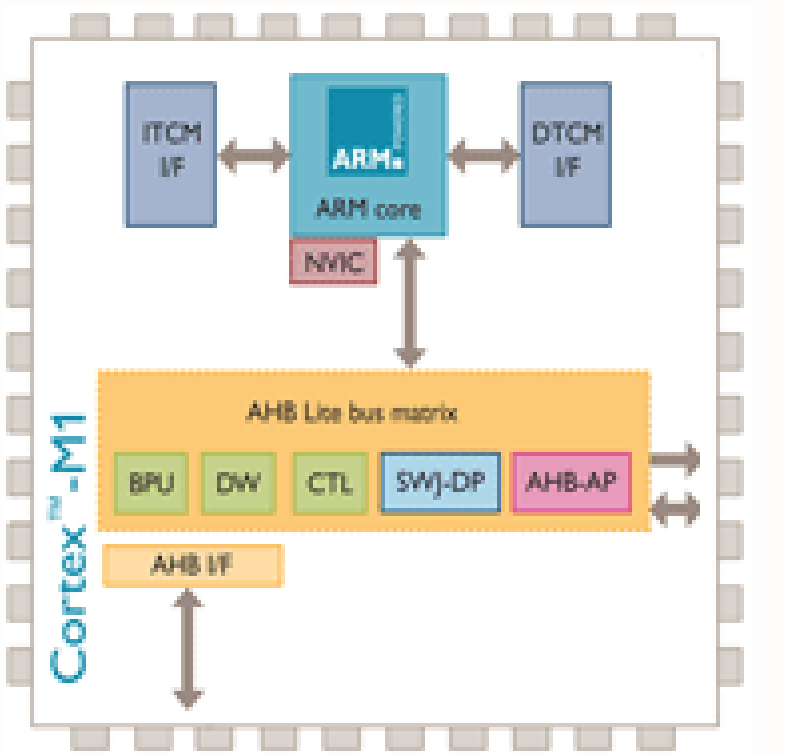
Embedded ARM Cortex Processors

- Cortex M0:
  - Ultra low gate count (less that 12 K gates).
  - Ultra low-power (3 µW/MHz).
  - 32-bit processor.



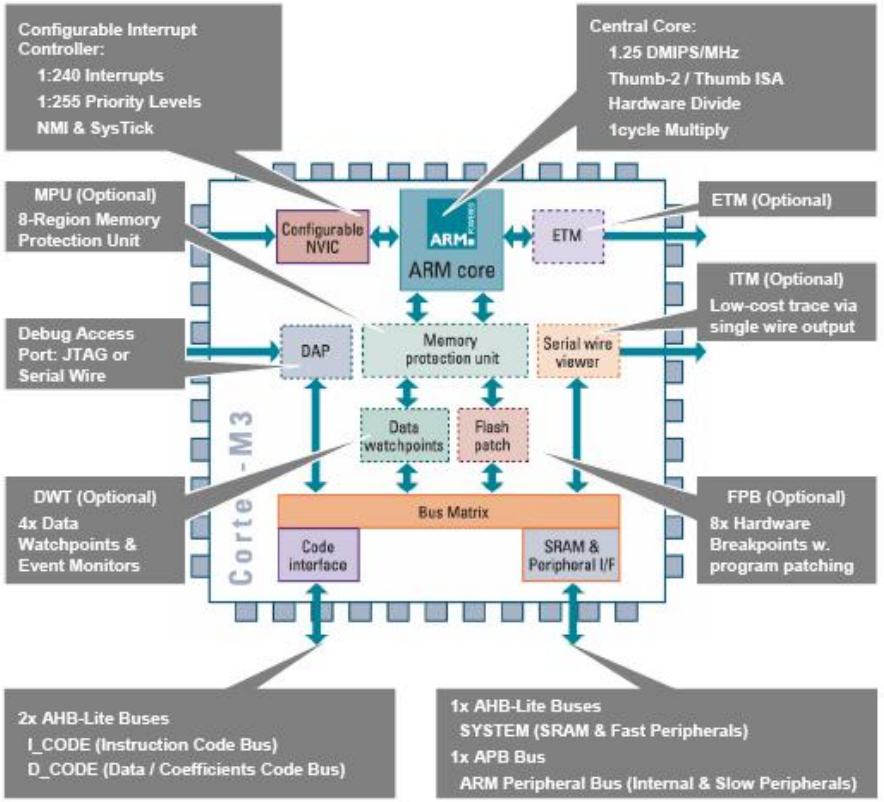
Embedded ARM Cortex Processors (2)

- Cortex M1:
  - The first ARM processor designed specifically for implementation in FPGAs.
  - Supports all major FPGA vendors.
  - Easy migration path from FPGA to ASIC.



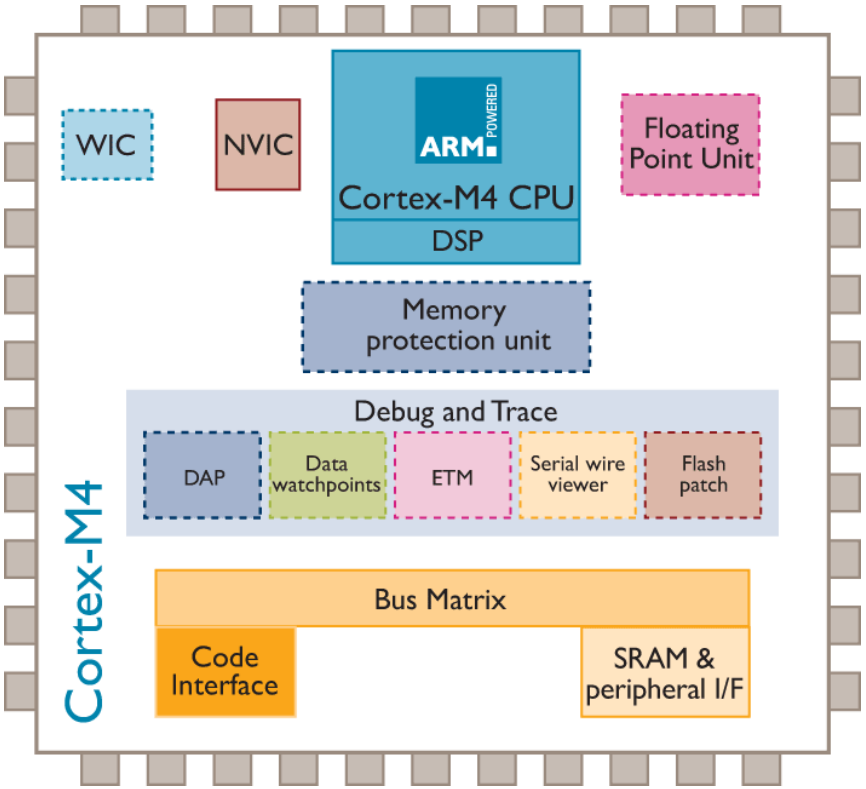
Embedded ARM Cortex Processors (3)

- Cortex M3:
  - The mainstream ARM processor for microcontroller applications.
  - High performance and energy efficiency.

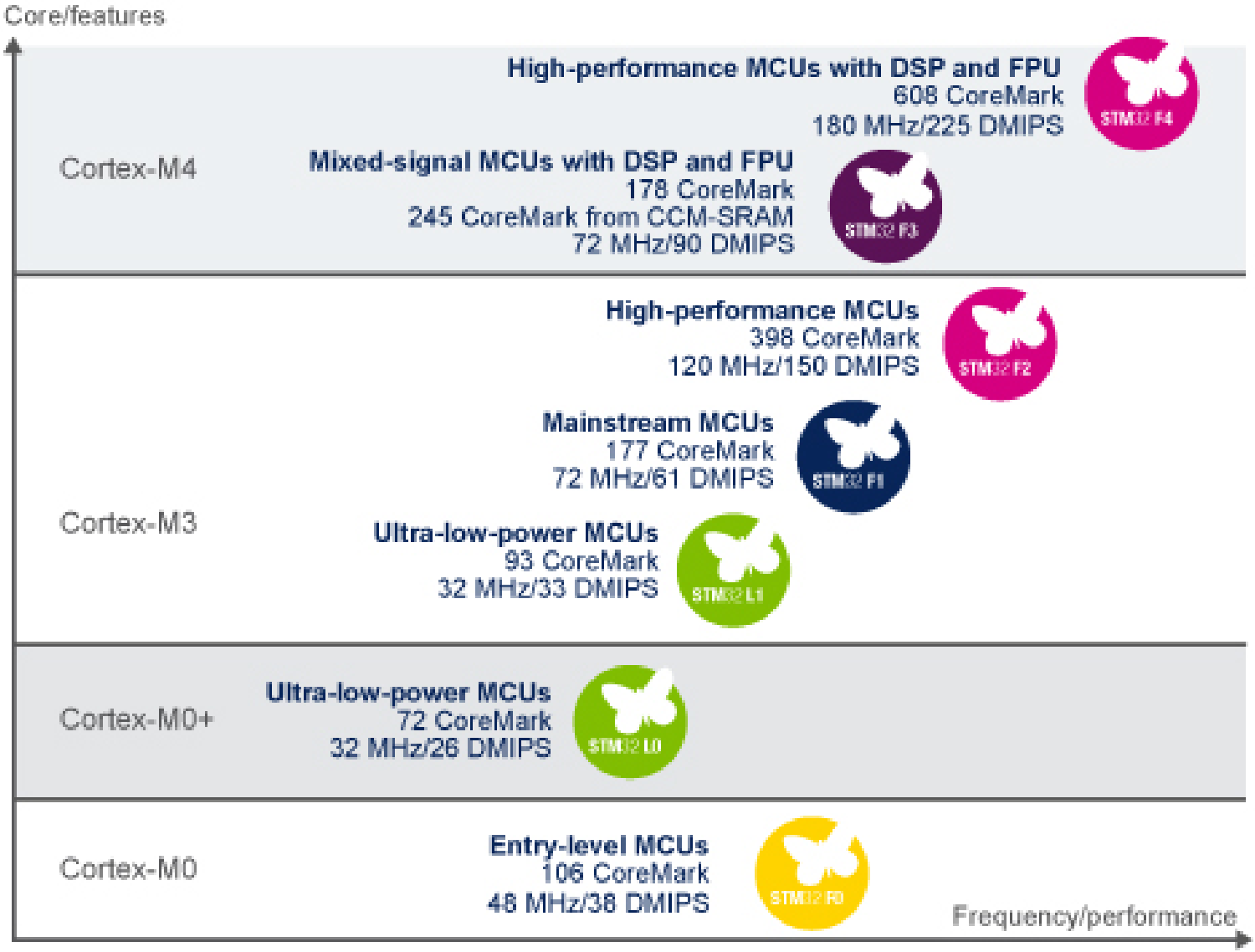


Embedded ARM Cortex Processors (4)

- Cortex M4:
  - The latest embedded processor for **DSP**.



STM32 ARM® Cortex™-M Family



STM32L1x - Block Diagram

System
- Power supply Internal regulator POR/PDR/PVD/BOR
- Xtal oscillators 32 kHz + 1 ~24 MHz
- Internal RC oscillators 37 kHz + 16 MHz
- Internal multispeed ULP RC oscillator 64 kHz to 4 MHz
- PLL
- Clock control
- RTC/AWU
- 2x watchdogs (independent and window)
- 36/50/82/108/114 I/Os
- Cyclic redundancy check (CRC)
- Voltage scaling 3 modes
- Touch sensing

Control
- 8x 16-bit timer
- 1x 32-bit timer

ARM Cortex-M3 CPU 32 MHz
- Nested vector interrupt controller (NVIC)
- JTAG/SW debug
- Embedded Trace Macrocell (ETM)
- Memory protection unit (MPU)

AHB bus matrix

7-channel DMA

Display
- LCD driver (8x40)

Encryption
- AES (128 bits)*

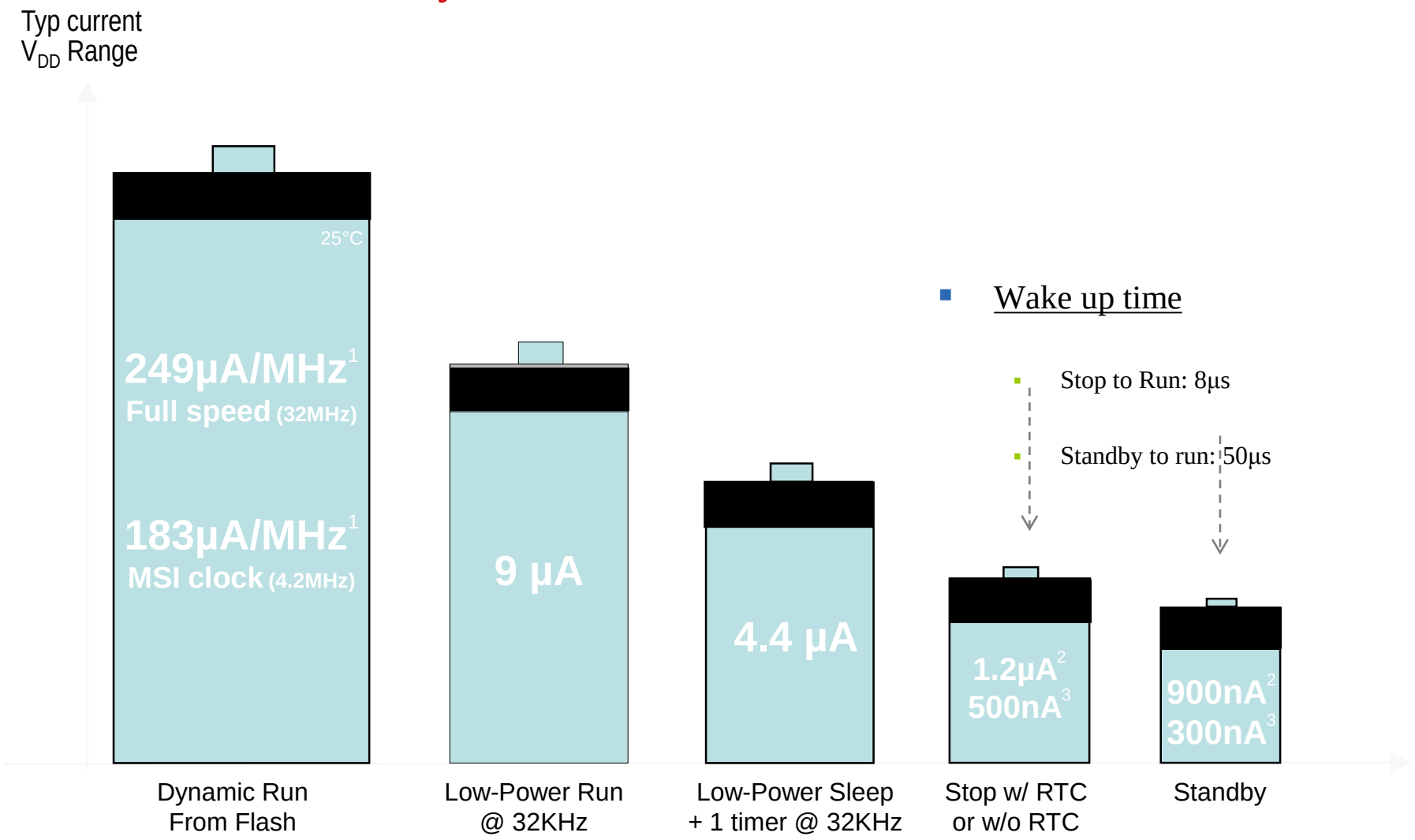
- 32- to 384-Kbyte Flash memory / dual bank
- Up to 48-Kbyte SRAM
- 128-byte backup data
- Up to 12-Kbyte EEPROM
- Boot ROM

Connectivity
- FSMC
- USB 2.0 FS
- 5x USART
- 3x SPI
- 2x I²C
- SDIO

Analog
- 2x 12-bit DAC
- 12-bit ADC 24 channels
- 2x comparators
- 3x op-amps
- Temperature sensor

STM32L1x – Power profile

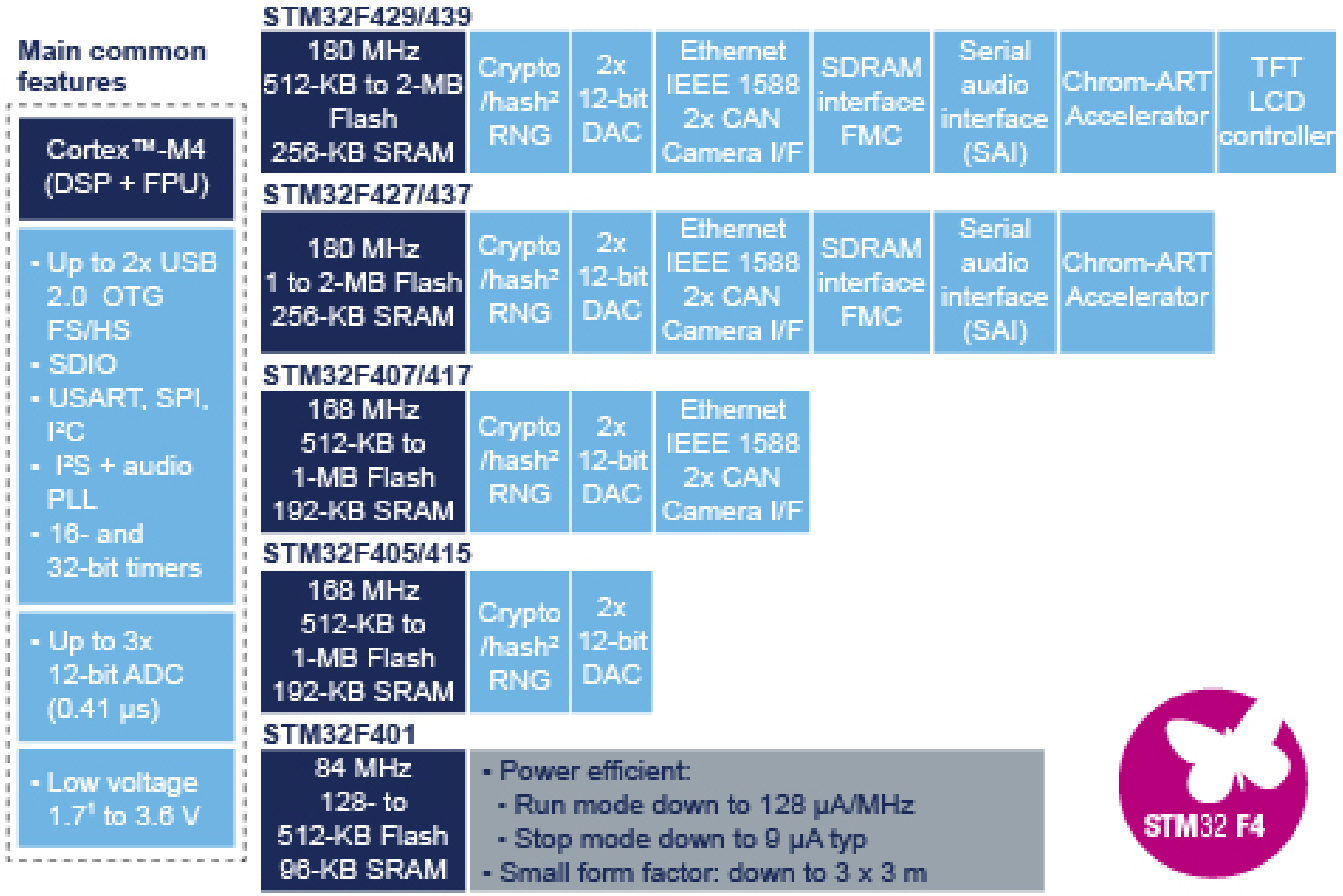
128 Kbytes Flash die



1/ Dhrystone power consumption value executed from Flash with V_{DD}=3V
2/ Stop and standby with RTC given with V_{DD}=1.8V
3/ Stop and standby without RTC given with V_{DD}=3V

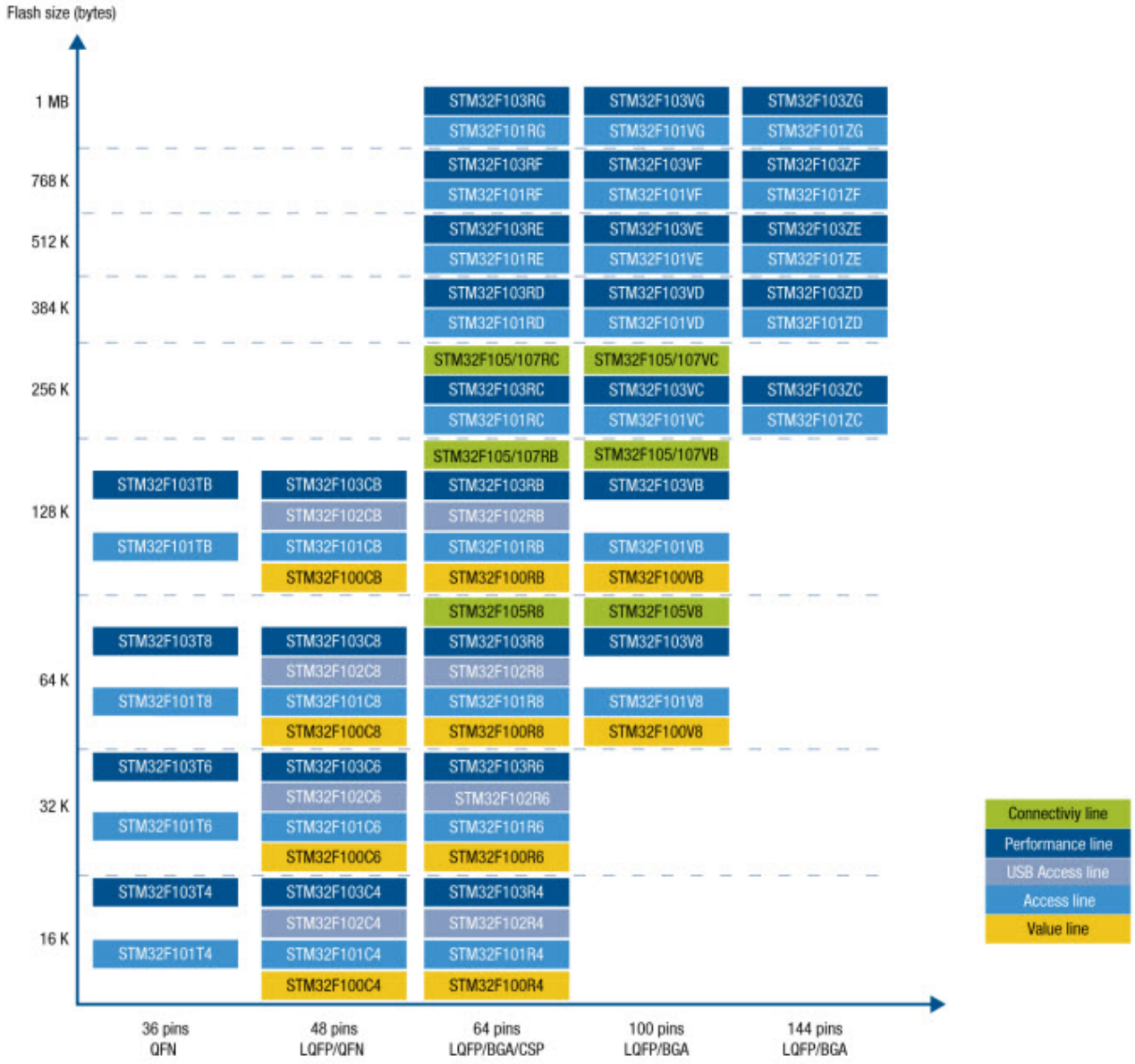
STM32F4 – CORTEX M4

STM32F401 – 84 MHz, the smallest, cost-effective solution with outstanding power efficiency
STM32F405/415 – 168 MHz up to 1 Mbyte of Flash with advanced connectivity and encryption
STM32F407/417 – 168 MHz, up to 1 Mbyte of Flash adding Ethernet MAC and camera interface
STM32F427/437 – 180 MHz, up to 2 Mbytes of dual-bank Flash with SDRAM interface, Chrom-ART Accelerator™, serial audio interface, more performance and lower static power consumption
STM32F429/439 – 180 MHz CPU/225 DMIPS, up to 2 Mbytes of dual-bank flash adding an LCD-TFT controller



Notes:
1. 1.7 V min on specific packages
2. Hardware crypto/hash on F415/417 and F437/439 only

STM32F101 Product Lines



STM32F10x Product Lines (2)

All lines include:

- Multiple communication peripherals Up to 5 x USART, 3xSPI, 2xI²C
- ETM*
- FSMC**
- Dual 12-bit DAC***
- Multiple 16-bit Timers
- Main Osc 4-16MHz (25MHz on 105/107)
- Internal 8 MHz RC and 40 kHz RC
- Real Time Clock with Battery domain & 32KHz ext osc
- 2 x Watchdogs
- Reset circuitry and Brown Out Warning
- Up to 12 DMA channels

\+

Connectivity Line: STM32F107

72MHz CPU	Up to 256 KB Flash / 64KB SRAM	2x12-bit ADC (1µs) TempSensor	USB 2.0 OTG (FS)	2 x Audio Class I2S	2 x CAN	PWM timer	Ethernet IEEE1588

Connectivity Line: STM32F105

72MHz CPU	Up to 256 KB Flash / 64KB SRAM	2x12-bit ADC (1µs) TempSensor	USB 2.0 OTG (FS)	2 x Audio Class I2S	2 x CAN	PWM timer

Performance Line: STM32F103

72MHz CPU	Up to 1MB Flash / 96KB SRAM	2/3x12-bit ADC (1µs) TempSensor	USB–FS Device	SDIO*	I2S*	CAN	PWM timer

USB Access Line: STM32F102

48MHz CPU	Up to 128KB Flash / 16KB SRAM	1x12-bit ADC (1µs) Temp sensor	USB–FS Device

Access Line: STM32F101

36MHz CPU	Up to 1MB Flash / 80KB SRAM	1x12-bit ADC (1µs) Temp sensor

Value Line: STM32F100

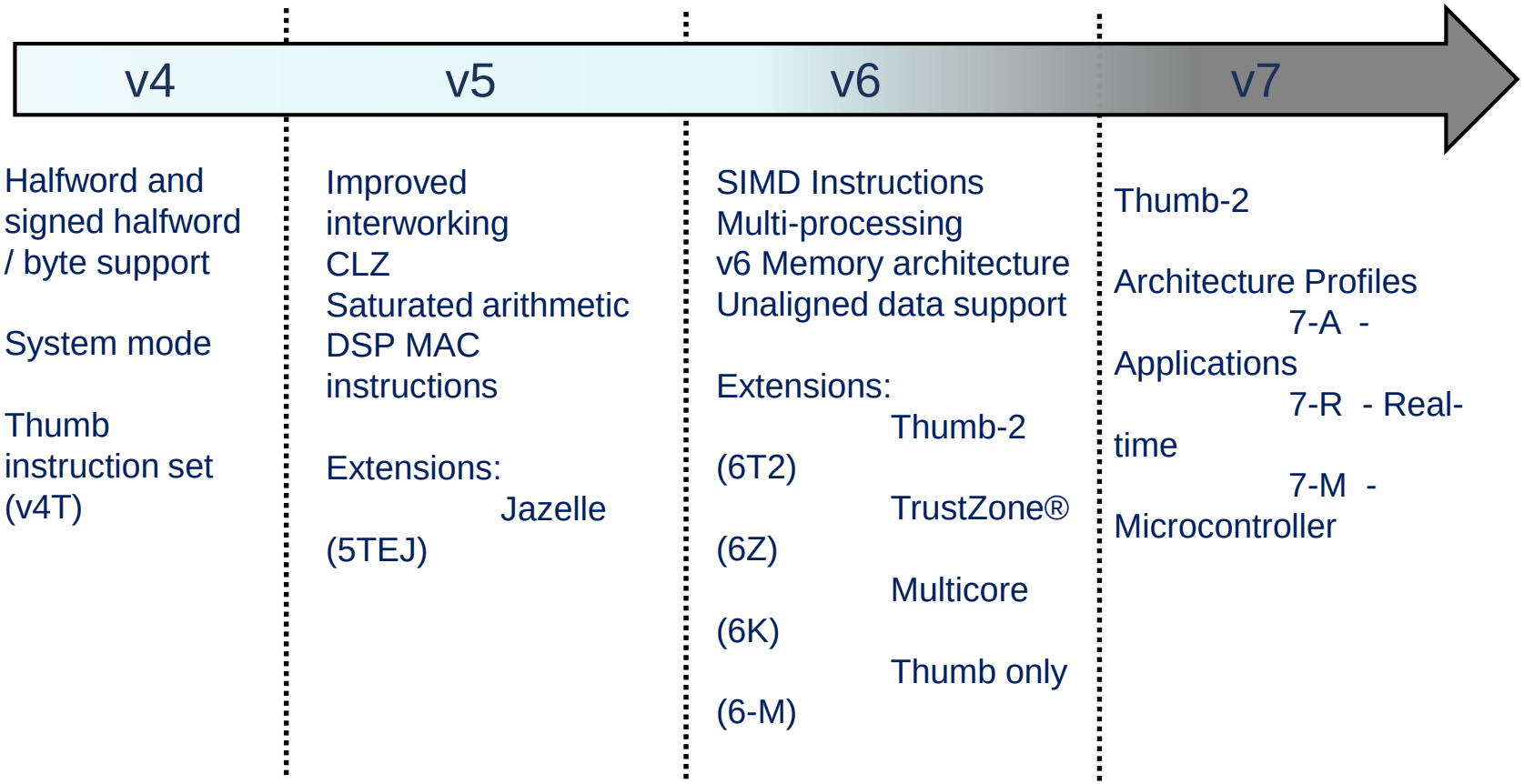
24MHz CPU	Up to 512KB Flash / 32KB SRAM	1x12-bit ADC (1.2µs) Temp sensor	HDMI-CEC	PWM timer

* Performance/Access Lines 256KB Flash or more, Value Line with 100+pins and ALL Connectivity devices

** Performance and Access and Value devices with 256KB Flash or more.

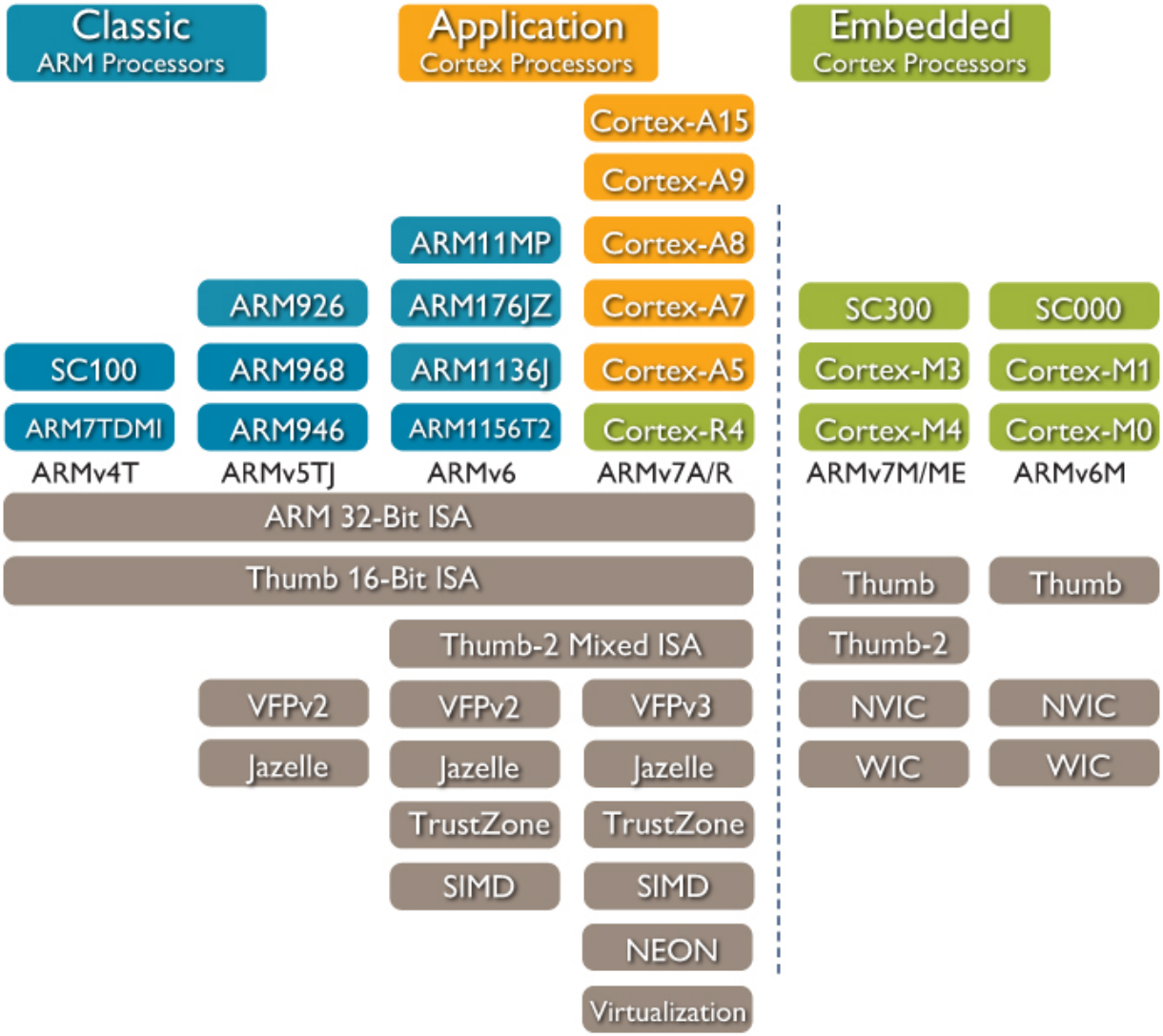
*** ALL Value line devices and Performance/Access devices with 256KB Flash or more

Development of the ARM Architecture



- **Note that implementations of the same architecture can be different**
  - Cortex-A8 - architecture v7-A, with a 13-stage pipeline
  - Cortex-A9 - architecture v7-A, with an 8-stage pipeline

Which architecture is my processor?



Data Sizes and Instruction Sets

- ARM is a 32-bit load / store RISC architecture
  - The only memory accesses allowed are loads and stores
  - Most internal registers are 32 bits wide
  - Most instructions execute in a single cycle
- When used in relation to ARM cores
  - **Halfword** means 16 bits (two bytes)
  - **Word** means 32 bits (four bytes)
  - **Doubleword** means 64 bits (eight bytes)
- ARM cores implement two basic instruction sets
    - ARM instruction set – instructions are all 32 bits long
    - Thumb instruction set – instructions are a mix of 16 and 32 bits
    - Thumb-2 technology added many extra 32- and 16-bit instructions to the original 16-bit Thumb instruction set
- Depending on the core, may also implement other instruction sets
  - **VFP** instruction set – 32 bit (vector) floating point instructions
  - **NEON** instruction set – 32 bit SIMD instructions
  - **Jazelle-DBX** - provides acceleration for Java VMs (with additional software support)
  - **Jazelle-RCT** - provides support for interpreted languages

Datasheet example : system architecture

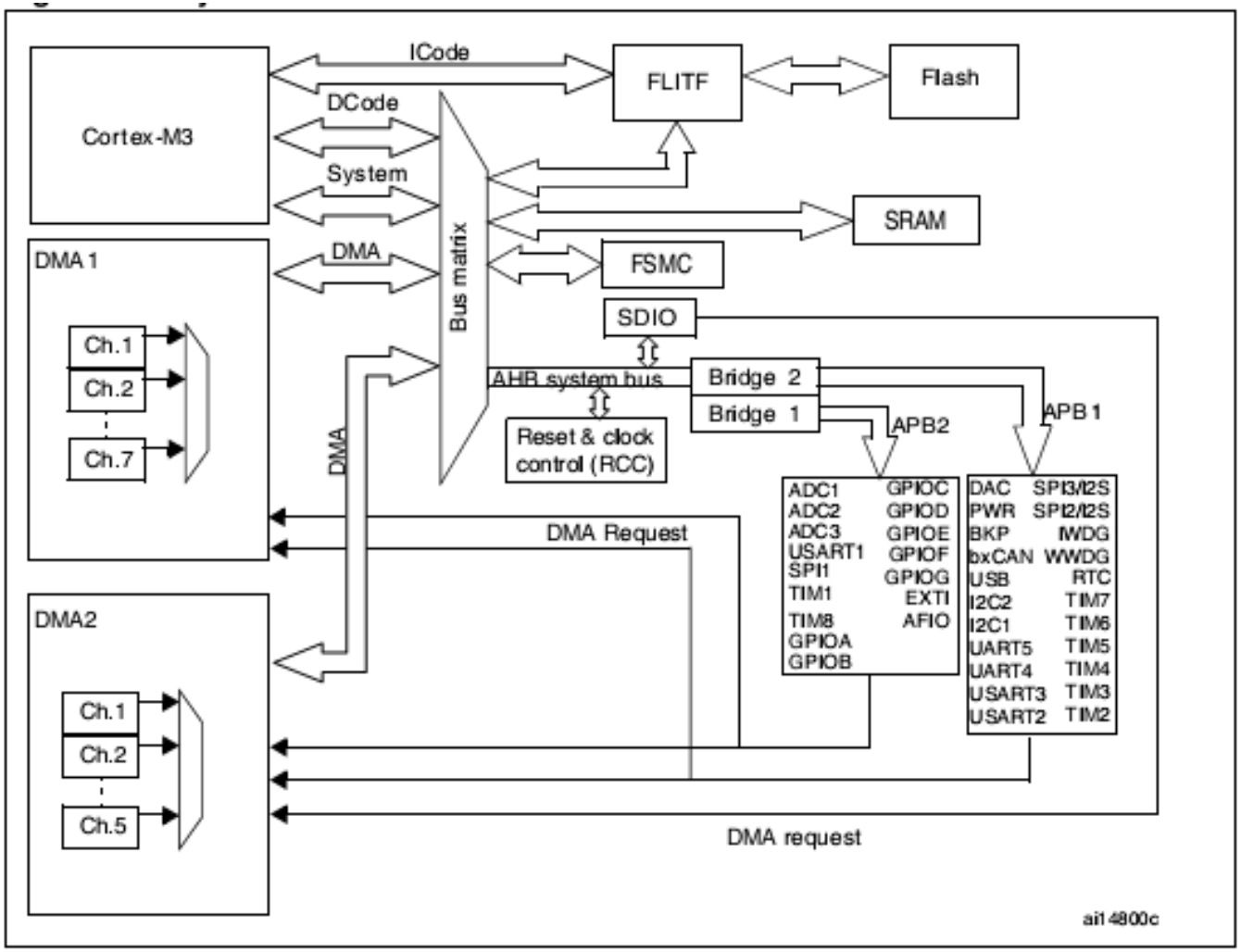
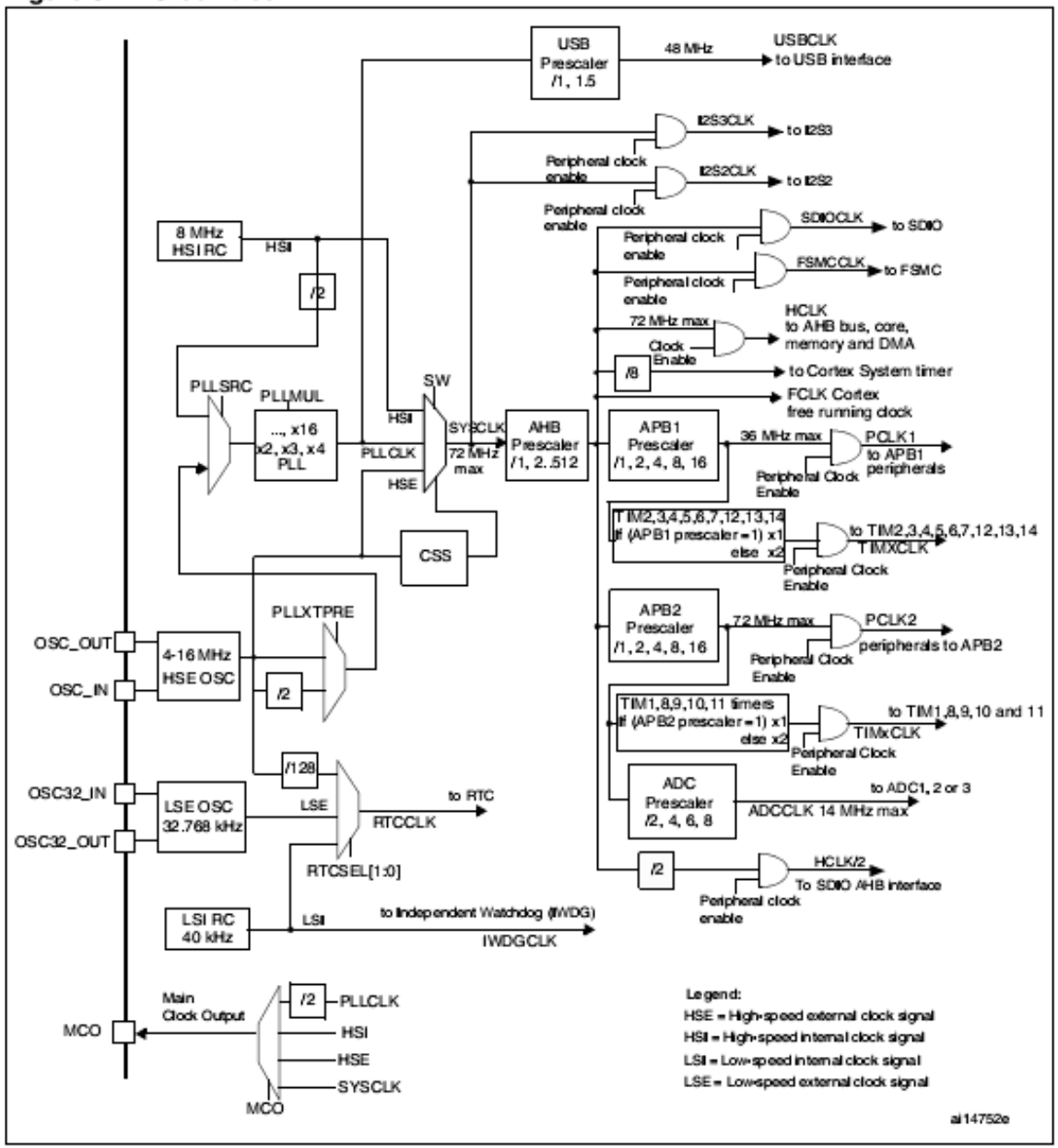
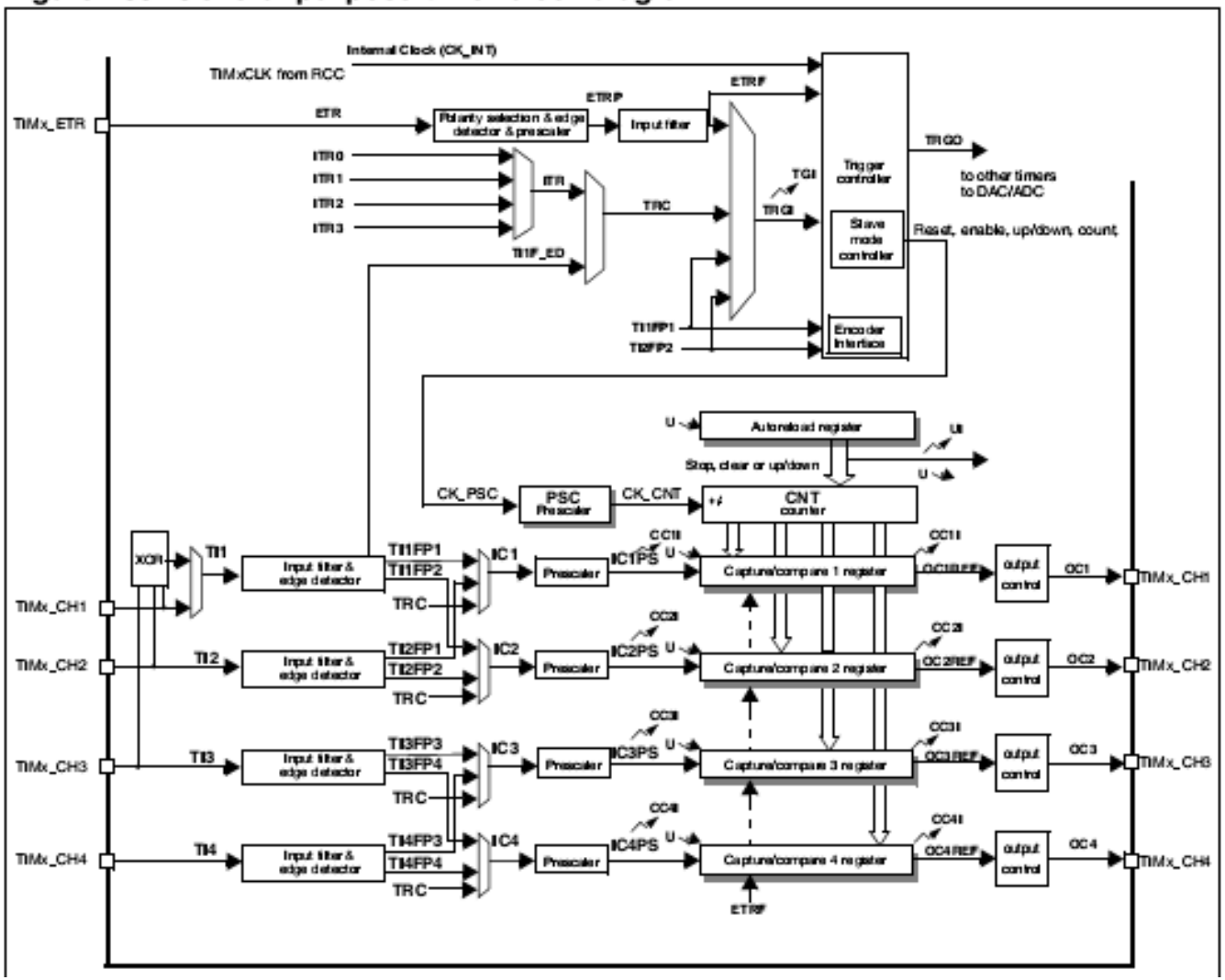


Figure 8. Clock tree



Datasheet example: Timers

Figure 100. General-purpose timer block diagram



14.4.1 TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

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

Bits 15:10 Reserved, always read as 0

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

00: $t_{DTS} = t_{CK\_INT}$
01: $t_{DTS} = 2 \times t_{CK\_INT}$
10: $t_{DTS} = 4 \times t_{CK\_INT}$
11: Reserved

Bit 7 **ARPE**: Auto-reload preload enable

0: TIMx_ARR register is not buffered
1: TIMx_ARR register is buffered

Bits 6:5 **CMS**: Center-aligned mode selection

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.
10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.
11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

Bit 4 **DIR**: Direction

0: Counter used as upcounter
1: Counter used as downcounter

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM**: One-pulse mode

0: Counter is not stopped at update event
1: Counter stops counting at the next update event (clearing the bit CEN)

Questions?