Embedded Systems

Dariusz Makowski, D.Sc., Associate Professor

Department of Microelectronics and Computer Science

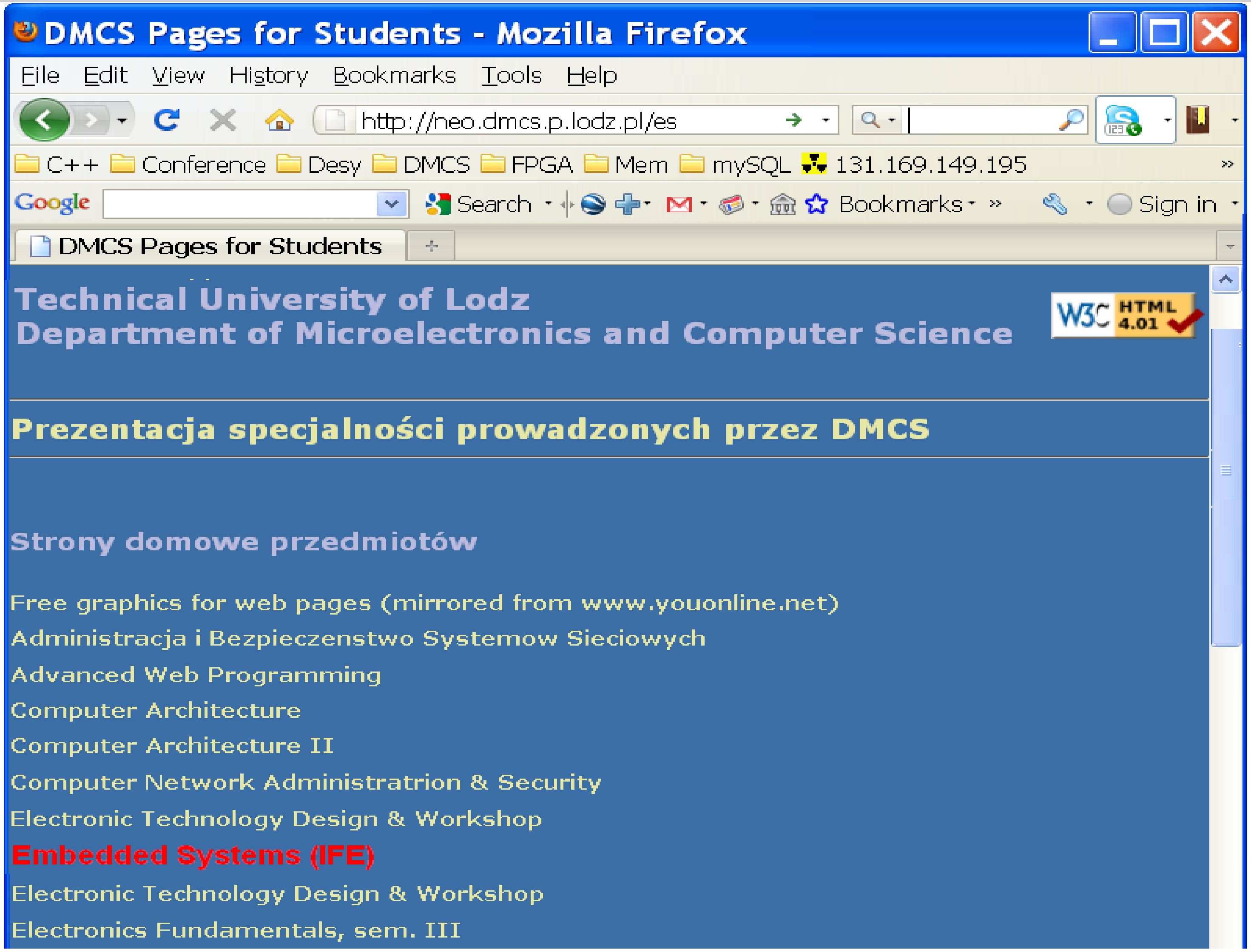
tel. 631 2720

dmakow@dmcs.pl

http://fiona.dmcs.pl/es

- Introduction
- Exam
- Reading materials
- Laboratory

Web page for Embedded Systems


DMCS Pages for Students - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://neo.dmcs.p.lodz.pl/es
C++ Conference Desy DMCS FPGA Mem mySQL 131.169.149.195
Google Search Bookmarks Sign in
DMCS Pages for Students

Technical University of Lodz
Department of Microelectronics and Computer Science

Prezentacja specjalności prowadzonych przez DMCS

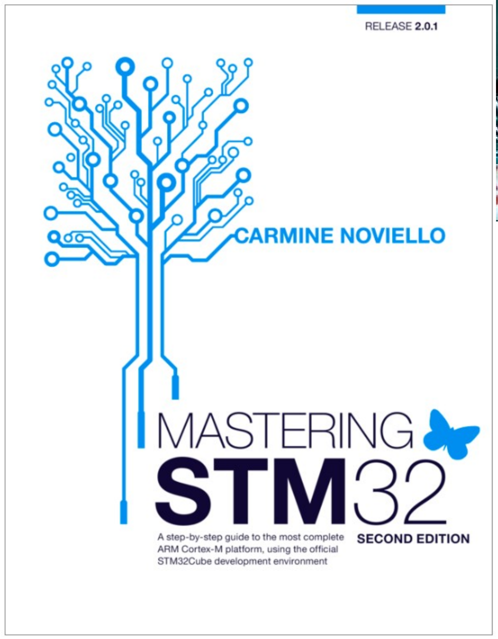
Strony domowe przedmiotów

Free graphics for web pages (mirrored from www.youonline.net)
Administracja i Bezpieczenstwo Systemow Sieciowych
Advanced Web Programming
Computer Architecture
Computer Architecture II
Computer Network Administratrion & Security
Electronic Technology Design & Workshop
Embedded Systems (IFE)
Electronic Technology Design & Workshop
Electronics Fundamentals, sem. III

Literature

Reading Materials:

- **Lecture and laboratory materials**
- A. Dean, “Embedded Systems Fundamentals with ARM Cortex-M based Microcontrollers: A Practical Approach”, ARM Education Media, 2017, ISBN-13 9781911531012, ISBN-10 1911531018
- C. Noviello, “Mastering STM32 - Second Edition”, Leanpub 2022
- STM32 MCU Developer Resources
  - https://www.st.com/content/st_com/en/stm32-mcu-developer-zone/developer-resources.html
  - https://www.st.com/resource/en/user_manual/dm00173145-description-of-stm32l4l4-hal-and-lowlayer-drivers-stmicroelectronics.pdf


RELEASE 2.0.1
CARMINE NOVIELLO
MASTERING STM32
A step-by-step guide to the most complete ARM Cortex-M platform, using the official STM32Cube development environment
SECOND EDITION

Laboratory

Address:

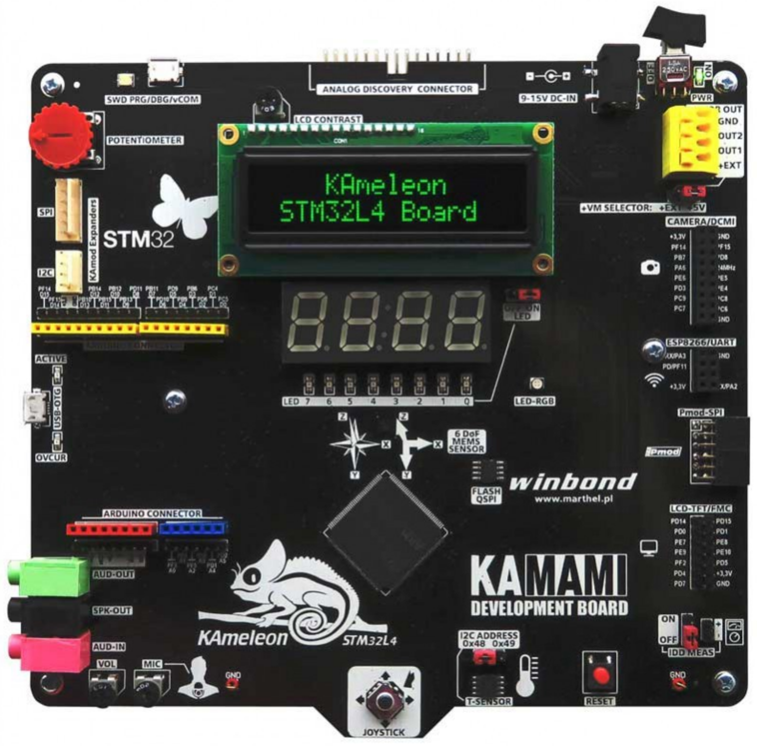
- Building B18, 1st floor – laboratory M

Practical exercises with application of ARM processor:

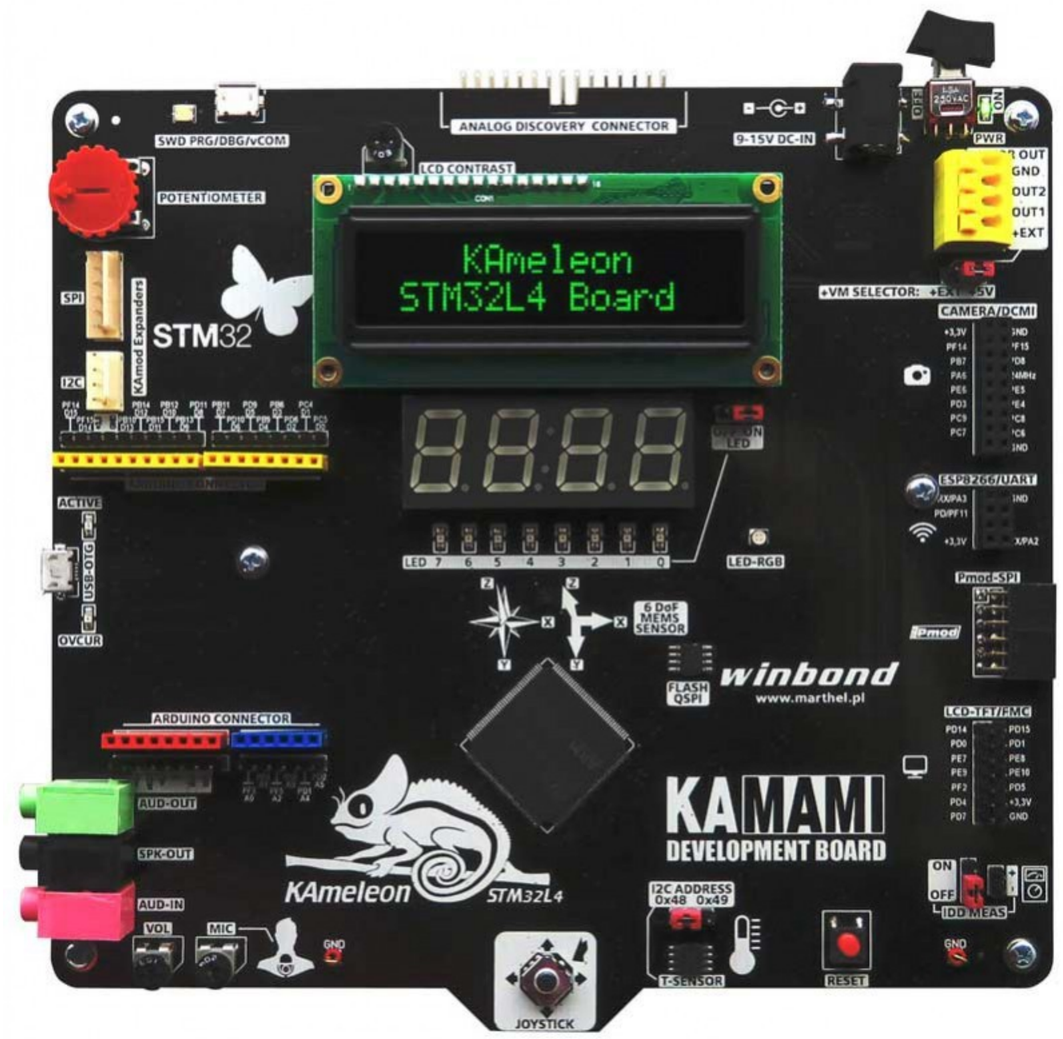
- Windows operating system
- GNU tools
- KAmeleon-STM32L4
- Extension board with peripheral devices
- FreeRTOS real-time system for embedded devices

Hardware – STM32 Evaluation Module (1)

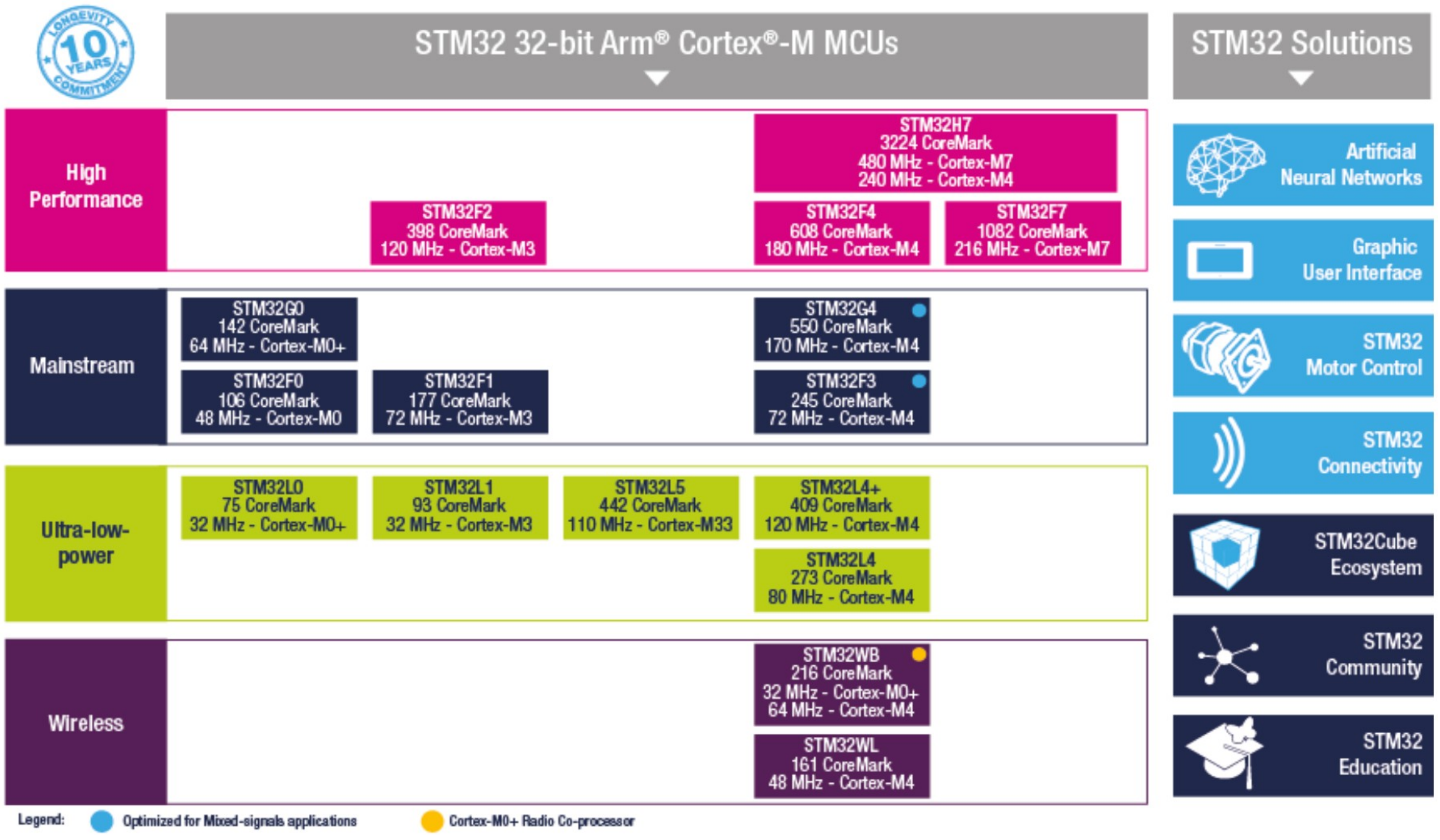
- **Microcontroller with ARM Cortex M4:** STM32L496ZGT6
- **Memory:** 320 kB SRAM,1 MB FLASH, 1 MB SPI-PROM
- **Interfaces:** SPI, I2C, USB-OTG, EIA RS232
- **Display:** 4 digit LED and alpha-numeric displays, 8x LED, RGB LED diode
- **Audio:** microphone, audio amplifier
- **Peripheral devices:** DC motor controller, thermometer, accelerometer, magnetometer
- **Interfaces (debugger, programmer):** Serial Wire Debug
- **Connectors:** DCMI (camera), WiFi, PMOD, etc.
- **Programmer:** ST-Link
- **Manufacture mark:** KAmeleon-STM32L4



Hardware – STM32 Evaluation Module (2)



Family of 32-bits STM32 Microcontrollers



STM32 32-bit Arm® Cortex®-M MCUs

High Performance		STM32F2 398 CoreMark 120 MHz - Cortex-M3		STM32H7 3224 CoreMark 480 MHz - Cortex-M7 240 MHz - Cortex-M4 STM32F4 608 CoreMark 180 MHz - Cortex-M4	STM32F7 1082 CoreMark 216 MHz - Cortex-M7
Mainstream	STM32G0 142 CoreMark 64 MHz - Cortex-M0+ STM32F0 106 CoreMark 48 MHz - Cortex-M0	STM32F1 177 CoreMark 72 MHz - Cortex-M3		STM32G4 ● 550 CoreMark 170 MHz - Cortex-M4 STM32F3 ● 245 CoreMark 72 MHz - Cortex-M4	
Ultra-low-power	STM32L0 75 CoreMark 32 MHz - Cortex-M0+	STM32L1 93 CoreMark 32 MHz - Cortex-M3	STM32L5 442 CoreMark 110 MHz - Cortex-M33	STM32L4+ 409 CoreMark 120 MHz - Cortex-M4 STM32L4 273 CoreMark 80 MHz - Cortex-M4	
Wireless				STM32WB ● 216 CoreMark 32 MHz - Cortex-M0+ 64 MHz - Cortex-M4 STM32WL 161 CoreMark 48 MHz - Cortex-M4	

Legend: ● Optimized for Mixed-signals applications ● Cortex-M0+ Radio Co-processor

STM32 Solutions

- Artificial Neural Networks
- Graphic User Interface
- STM32 Motor Control
- STM32 Connectivity
- STM32Cube Ecosystem
- STM32 Community
- STM32 Education

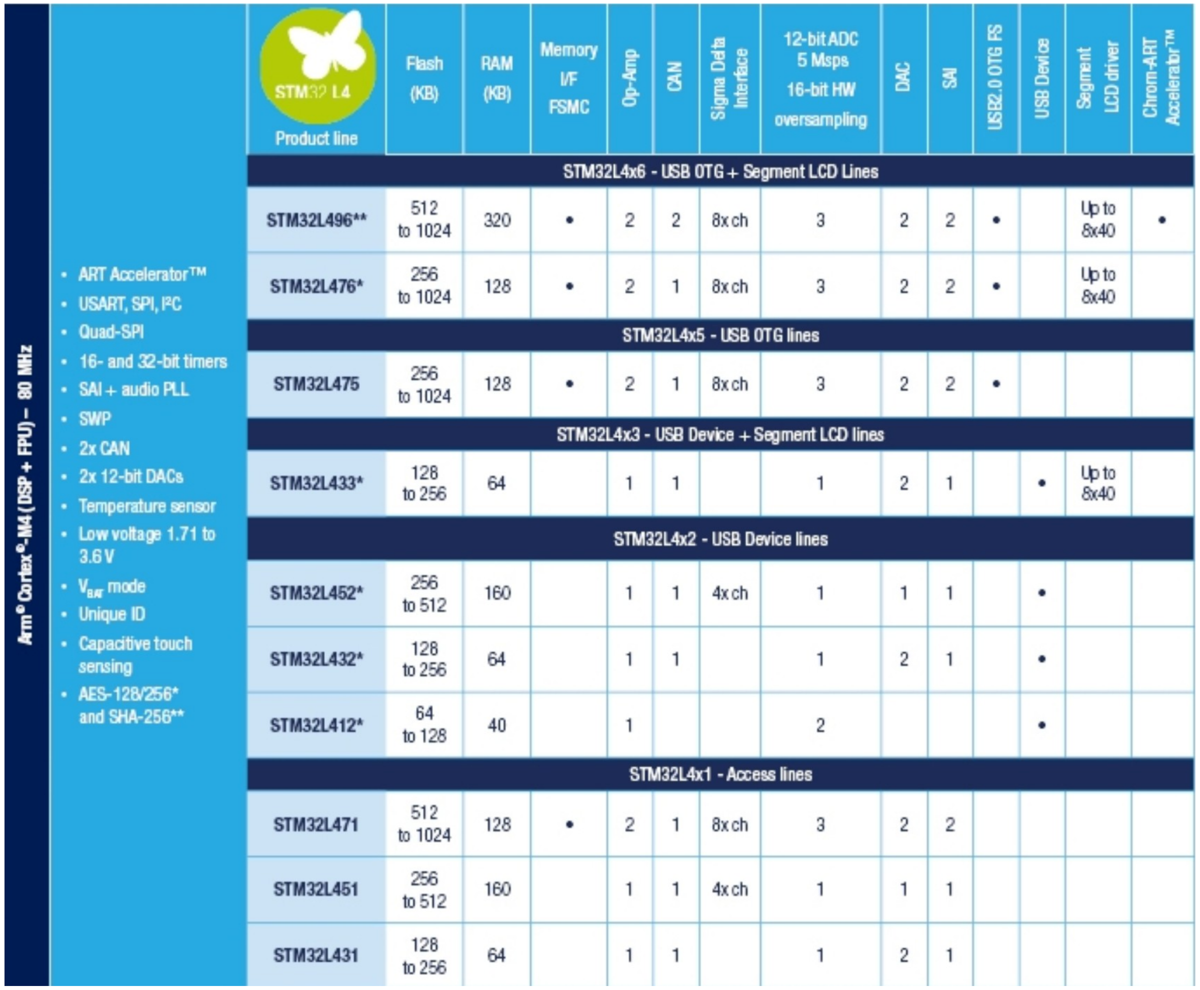
https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

STM Family and ARM Core

STM32 Family	Cortex-M	Thumb	Thumb-2	Multiply in Hardware	Divide in Hardware	Saturated math	DSP	FPU	ARM Architecture
F0	M0	Most	Some	32-bit result	No	No	No	No	ARMv6-M
L0	M0+	Most	Some	32-bit result	No	No	No	No	ARMv6-M
F1, F2, L1	M3	Entire	Entire	32/64-bit result	Yes	Yes	No	No	ARMv7-M
F3, F4, L4	M4	Entire	Entire	32/64-bit result	Yes	Yes	Yes	Yes SP	ARMv7E-M
F7	M7	Entire	Entire	32/64-bit result	Yes	Yes	Yes	Yes SP & DP	ARMv7E-M

STM32 Family	Cortex-M	SysTick Timer	Bit-Banding	Memory Protection Unit (MPU)	CPU Cache	OS Support	Memory Architecture
F0	M0	Yes	Yes	No	No	Yes	Von Neumann
L0	M0+	Yes	Yes	Yes	No	Yes	Von Neumann
F1, F2, L1	M3	Yes	Yes	Yes	No	Yes	Harvard
F3, F4, L4	M4	Yes	Yes	Yes	No	Yes	Harvard
F7	M7	Yes	No	Yes	Yes	Yes	Harvard

STM32L4 Ultra-low-power Series Microcontrollers



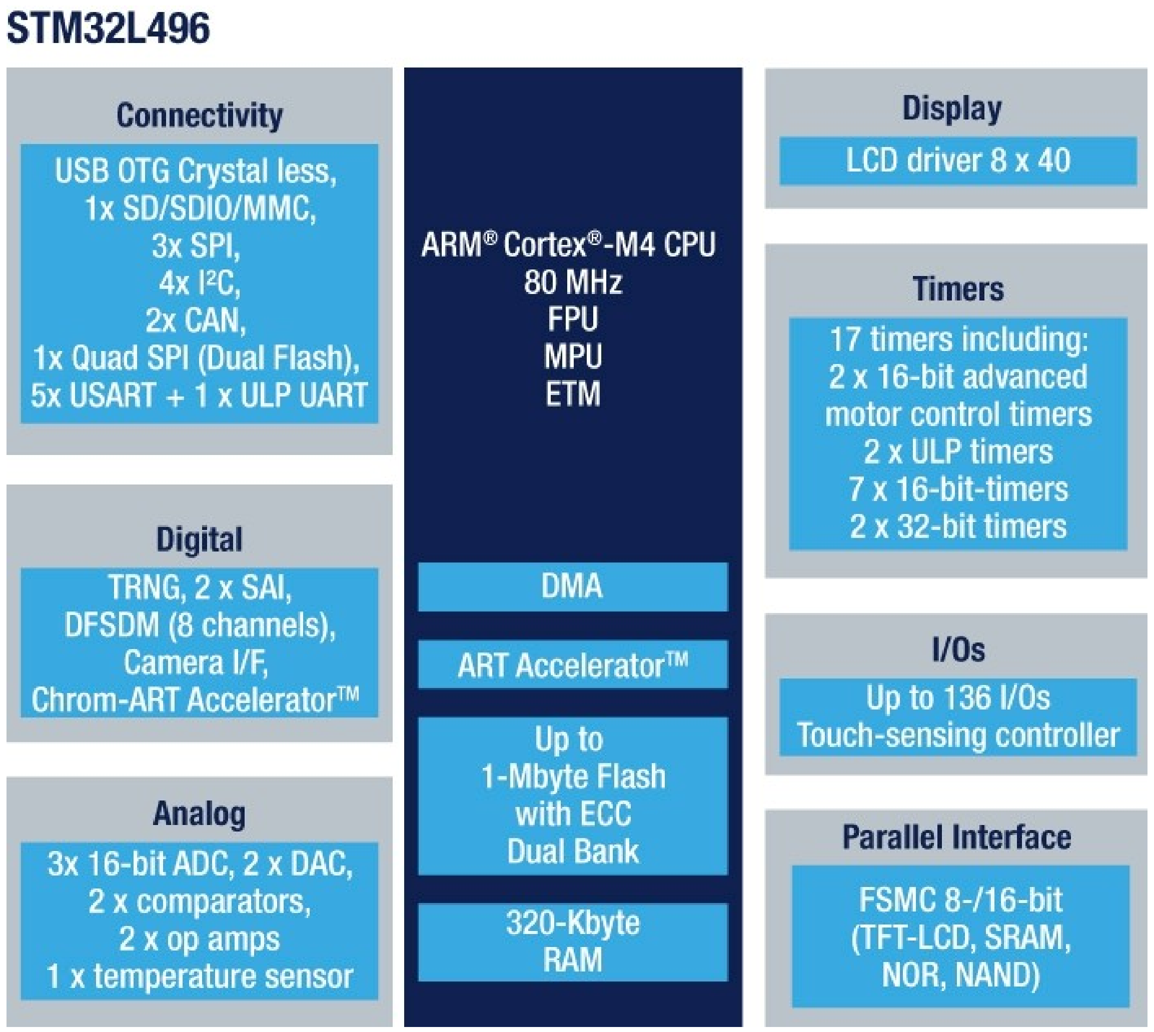
Arm® Cortex®-M4 (DSP + FPU) – 80 MHz

- ART Accelerator™
- USART, SPI, I²C
- Quad-SPI
- 16- and 32-bit timers
- SAI + audio PLL
- SWP
- 2x CAN
- 2x 12-bit DACs
- Temperature sensor
- Low voltage 1.71 to 3.6 V
- V_{BAT} mode
- Unique ID
- Capacitive touch sensing
- AES-128/256* and SHA-256**

STM32 L4 Product line	Flash (KB)	RAM (KB)	Memory I/F FSMC	Op-Amp	CAN	Sigma Delta Interface	12-bit ADC 5 Msps 16-bit HW oversampling	DAC	SAI	USB2.0 OTG FS	USB Device	Segment LCD driver	Chrom-ART Accelerator™
STM32L4x6 - USB OTG + Segment LCD Lines													
STM32L496**	512 to 1024	320	•	2	2	8x ch	3	2	2	•		Up to 8x40	•
STM32L476*	256 to 1024	128	•	2	1	8x ch	3	2	2	•		Up to 8x40	
STM32L4x5 - USB OTG lines													
STM32L475	256 to 1024	128	•	2	1	8x ch	3	2	2	•			
STM32L4x3 - USB Device + Segment LCD lines													
STM32L433*	128 to 256	64		1	1		1	2	1		•	Up to 8x40	
STM32L4x2 - USB Device lines													
STM32L452*	256 to 512	160		1	1	4x ch	1	1	1		•		
STM32L432*	128 to 256	64		1	1		1	2	1		•		
STM32L412*	64 to 128	40		1			2				•		
STM32L4x1 - Access lines													
STM32L471	512 to 1024	128	•	2	1	8x ch	3	2	2				
STM32L451	256 to 512	160		1	1	4x ch	1	1	1				
STM32L431	128 to 256	64		1	1		1	2	1				

https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus/stm32l4-series.html

Microcontroller STM32L496 ...



Lecture Agenda

- Microprocessor systems, embedded systems
- ARM processors family
- Peripheral devices
- Memories and address decoders
- ARM processor as platform for embedded programs
- Methodology of designing embedded systems
- Interfaces in embedded systems
- Real-time microprocessor systems

Lecture Agenda

- Microprocessor Systems, Embedded Systems
- ARM Processors Family
- Peripheral Devices
- Memories and Address Decoders
- ARM Processor as Platform for Embedded Programs
- Methodology of designing embedded systems
- Interfaces in Embedded Systems
- Real-Time Microprocessor Systems

Basic Definitions

- **Processor (Central Processing Unit)**

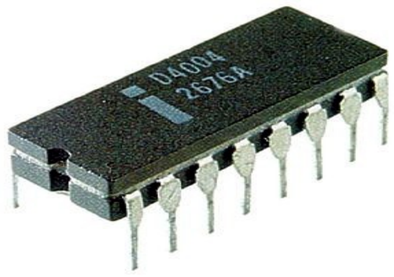
  Digital, sequential device able to read data from memory, interpret and process it as a commands

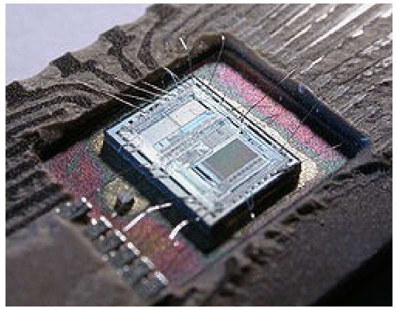
- **Microprocessor**

  Digital circuit fabricated as a single integrated device (Very High Scale Iterated Circuit) able to process digital operations according to provided digital information, e.g.: x86, Z80, 68k

- **Microcontroller**

  Computer fabricated as a single chip used to control electronic devices. Microcontroller is usually composed of CPU, integrated memories (RAM and ROM) and peripheral devices, e.g.: Intel 80C51, Atmel Atmega128, Freescale MCF5282, ARM926EJ-S

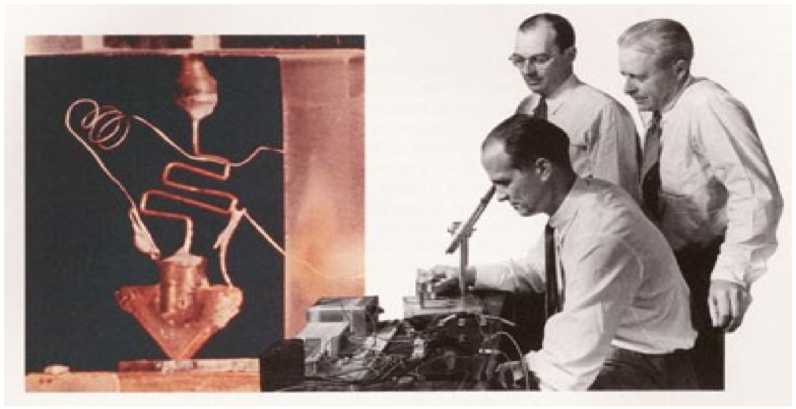




History of Microprocessors (1)

1940 – Russell Ohl – demonstration of simple semiconductor junction, diode (germanium diode, solar battery)

1947 – Shockley, Bardeen, Brattain present the first transistor


The first transistor, Bell Laboratories

The first integrated circuit , TI

1958 – Jack Kilby invented integrated circuit

1967 – Fairchild Laboratory provides first non-volatile memory ROM (64 bits)

1969 – Noyce and Moore left Fairchild, set up small silicon business INTEL. INTEL fabricates mainly volatile memories SRAM (64 bits). Japanese company, Busicom, orders twelfth different circuits for calculators.

History of Microprocessors (2)

1970 - **F14 CADC** (Central Air Data Computer) microprocessor designed by Steve Geller and Ray Holt for American army (F-14 Tomcat supersonic tighter)

1971 - **Intel 4004** 4-bits processor used for programmable calculator (the chip designed by Intel is recognised as the first processor on the world), the chip contains 3200 transistors. INTEL continue work on processors, Faggin (from Fairchild) works for INTEL and he helps to solve some problems.

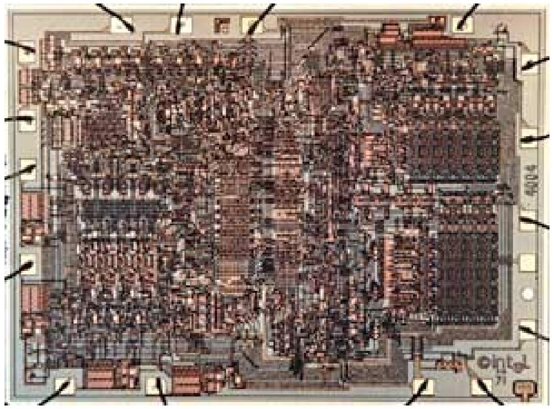
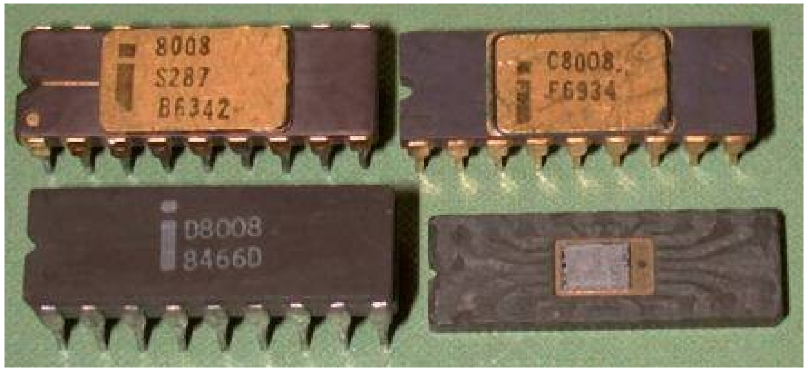


Photo of 4 bits INTEL 4004 processor



8 bits INTEL processors

1972 – Faggin starts work on the first 8-bits processor INTEL 8008. Industry is more and more interested in programmable devices - processors.

History of Microprocessors (3)

1974 – INTEL introduce improved version of 8008 processor, Intel 8080. Faggin left Intel and run out his own company called Zilog. Motorola offers another version of 8-bits processor Motorola 6800 (NMOS, 5 V).

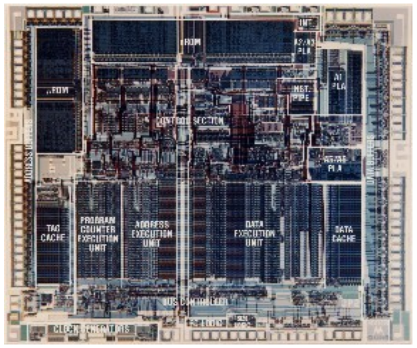
1975 – new 8-bits processor from INTEL 6502 (MOS technology) – the cheapest microprocessor on the world that time.

1978 – the first 16-bits processor 8086 (based on 8080).

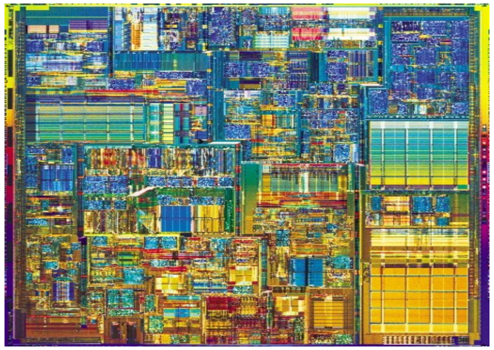
1979 – Motorola also offers 16-bits processor, 68000 family.

1980 – Motorola introduce new 32-bits processor 68020, 200,000 transistors.

....



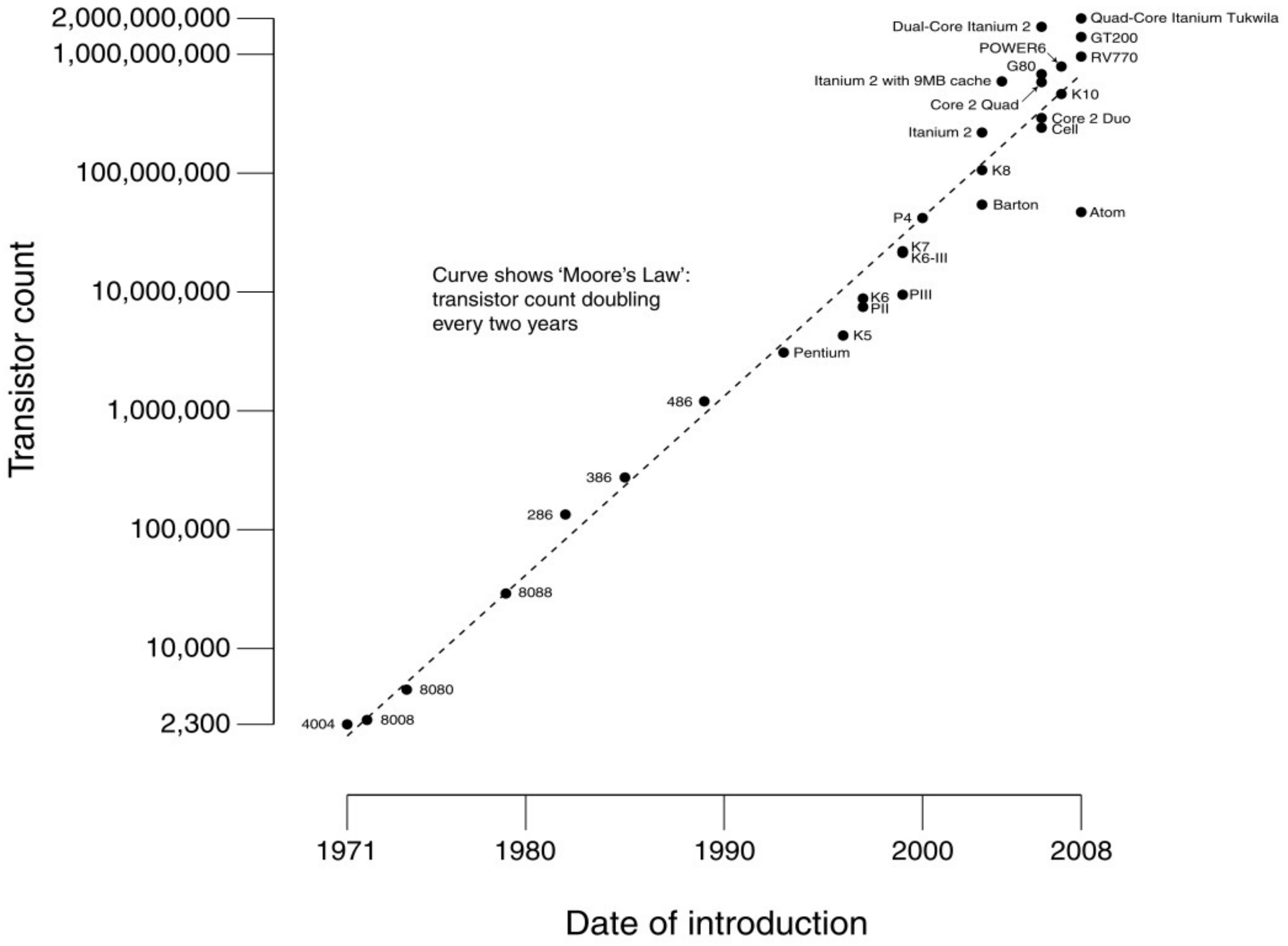
Motorola 68020


Intel, Pentium 4 Northwood

Intel 386, 486, Pentium I, II, III, IV, Centrino, Pentium D, Duo/Quad core, ...

Motorola 68030, 68040, 68060, PowerPC, ColdFire, ARM 7, ARM 9, StrongARM, ...

CPU Transistors Counts 1971-2008 and Moore law



Basic Definitions (2)

- **Computer**

Electronic device, digital machine able to read and process digital data according to provided commands, program or firmware.

- **Embedded Computer (EC)**

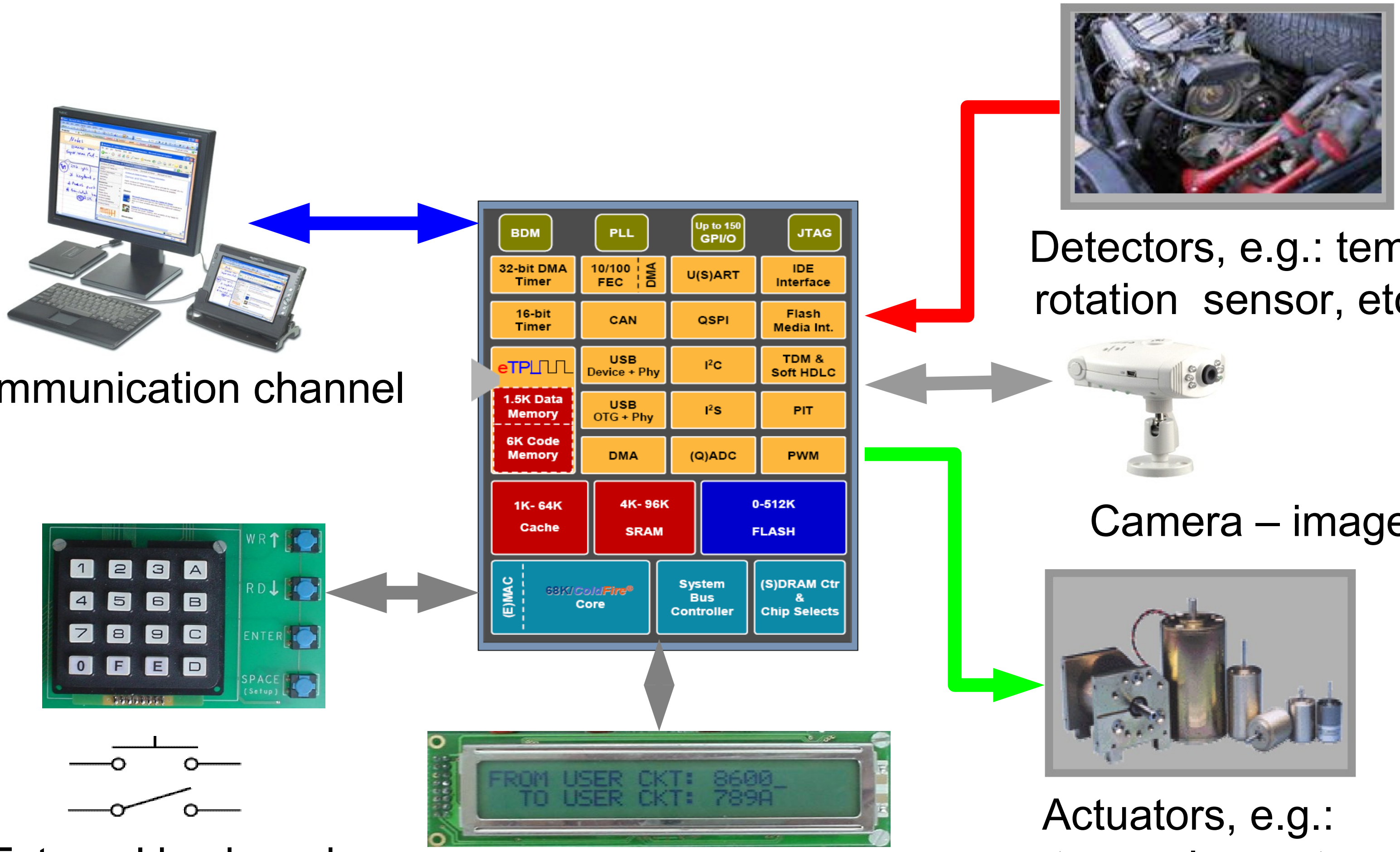
Dedicated computer designed to perform one or a few dedicated functions, usually ***build in*** the device. Embedded computers are used to control at least mechanical, electrical or electronic, devices.

- **Personal Computer (PC)**

Computers and computer systems dedicated for personal usage at home, office or work. The general-purpose computers are equipped with operating system responsible for processing user applications.

- **Computer Architecture**
  - Method of organisation and cooperation of basic computer components: processor-memory-peripheral devices.
  - Description of computer from programmer point of view (low level language, assembler). Processor design, processing pipeline and programming model.

Embedded Computer



Communication channel

Detectors, e.g.: temp., rotation sensor, etc...

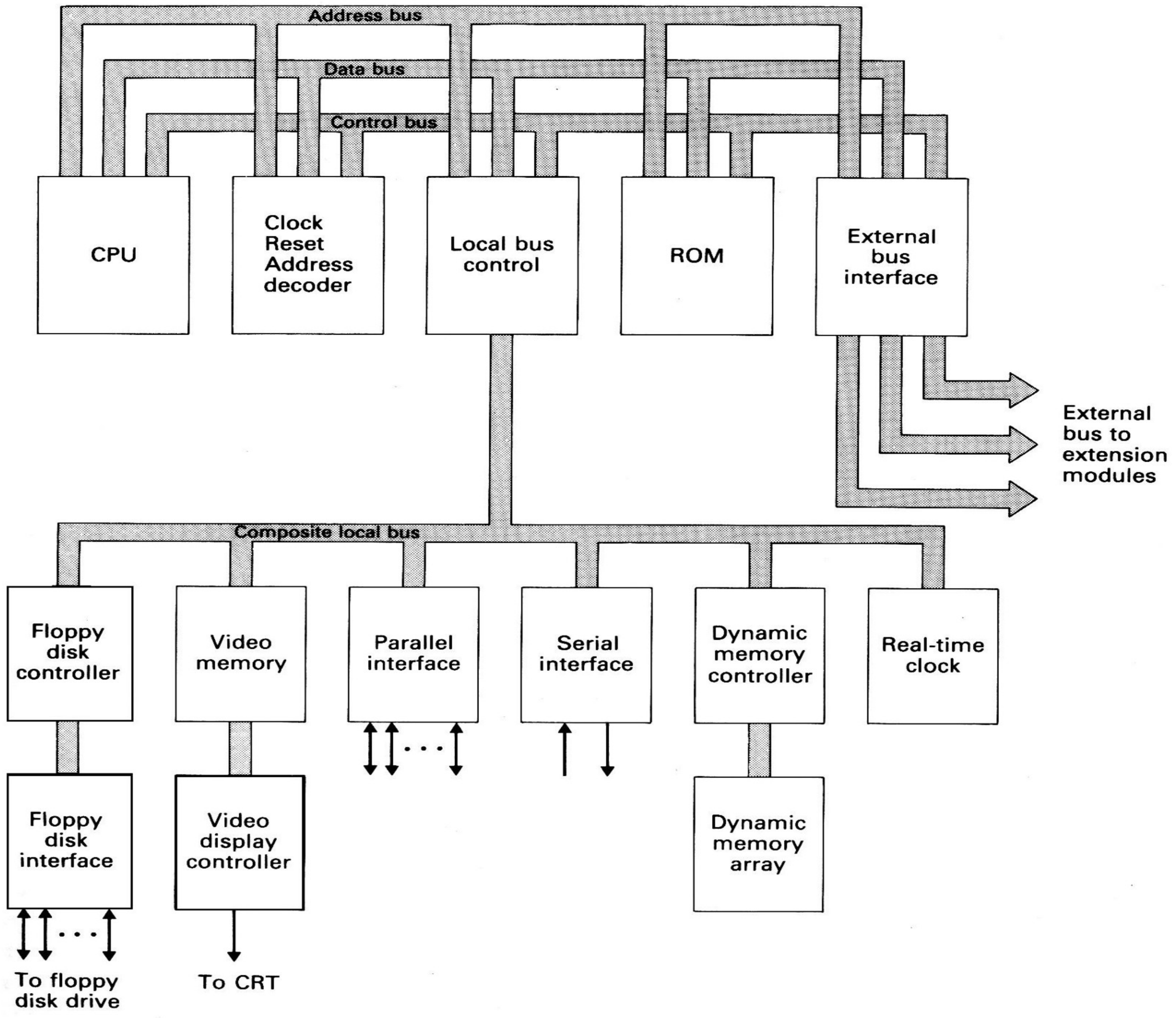
Camera – image recogn.

External keyboard

Display panel

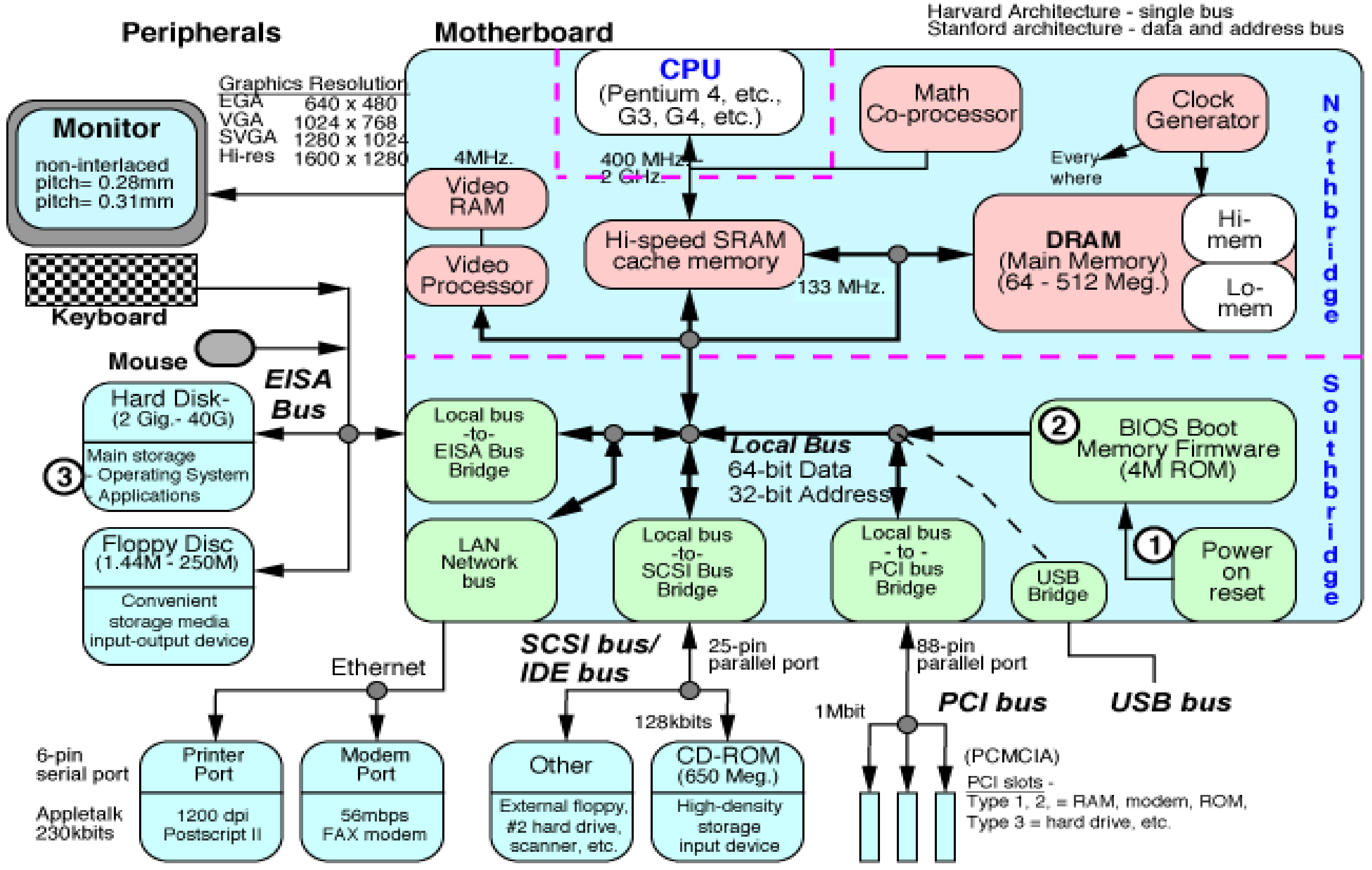
Actuators, e.g.: motors, relays, etc...

Universal Computer, Personal Computer



Personal Computer

Harvard Architecture - single bus
Stanford architecture - data and address bus



Basic Definitions (3)

- **Computer Memory**

Electronic or mechanic device used for storing digital data or computer programs (operating system and applications).

- **Peripheral Device**

Electronic device connected to processor via system bus or computer interface. External devices are used to realise dedicated functionality of the computer system. Internal devices are mainly used by processor and operating system.

- **Computer Bus**

Electrical connection or subsystem that transfers data between computer components: processors, memories and peripheral devices. System bus is composed of dozens of multiple connections (Parallel Bus) or a few single serial channels (Serial Bus).

- **Interface**

Electronic or optical device that allows to connect two or more devices. Interface can be parallel or serial.

Basic Definitions (4)

- **System-on-Chip**

Integrated circuits fabricated in VLSI technology, creating uniform device, containing all electronic components including processor, memories, peripheral devices, analogue, digital and RF (Radio Frequency) subsystems.

The components of the system are usually fabricated by different manufactures because of its complexity, e.g. 1st manufacturer Processor Core, 2nd man. peripheral devices, 3rd man. Interfaces, etc...

Typical application area of SoCs are embedded systems and the most suitable example of SoC are computer systems based of ARM processors.

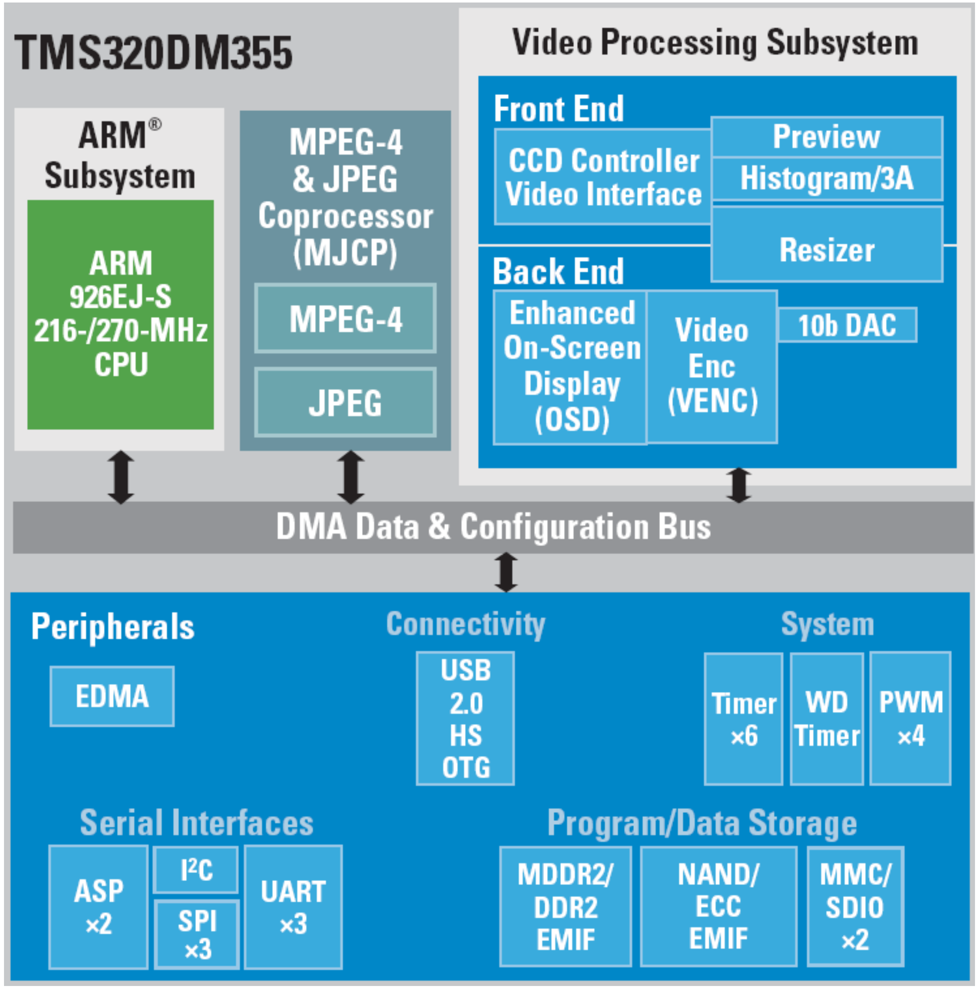
In the case when all subcomponents cannot be integrated on common silicon substrate, the following components are fabricated on different crystals and packed in single package, SiP (System-in-a-package).

SoC are different from microcontrollers that also include different peripheral devices because they include more powerful CPU (can run operating systems, Linux, Windows, RTEMS) and they are equipped with specialised peripheral devices (also programmable, e.g. FPGA).

SoC - DaVinci, digital media processor

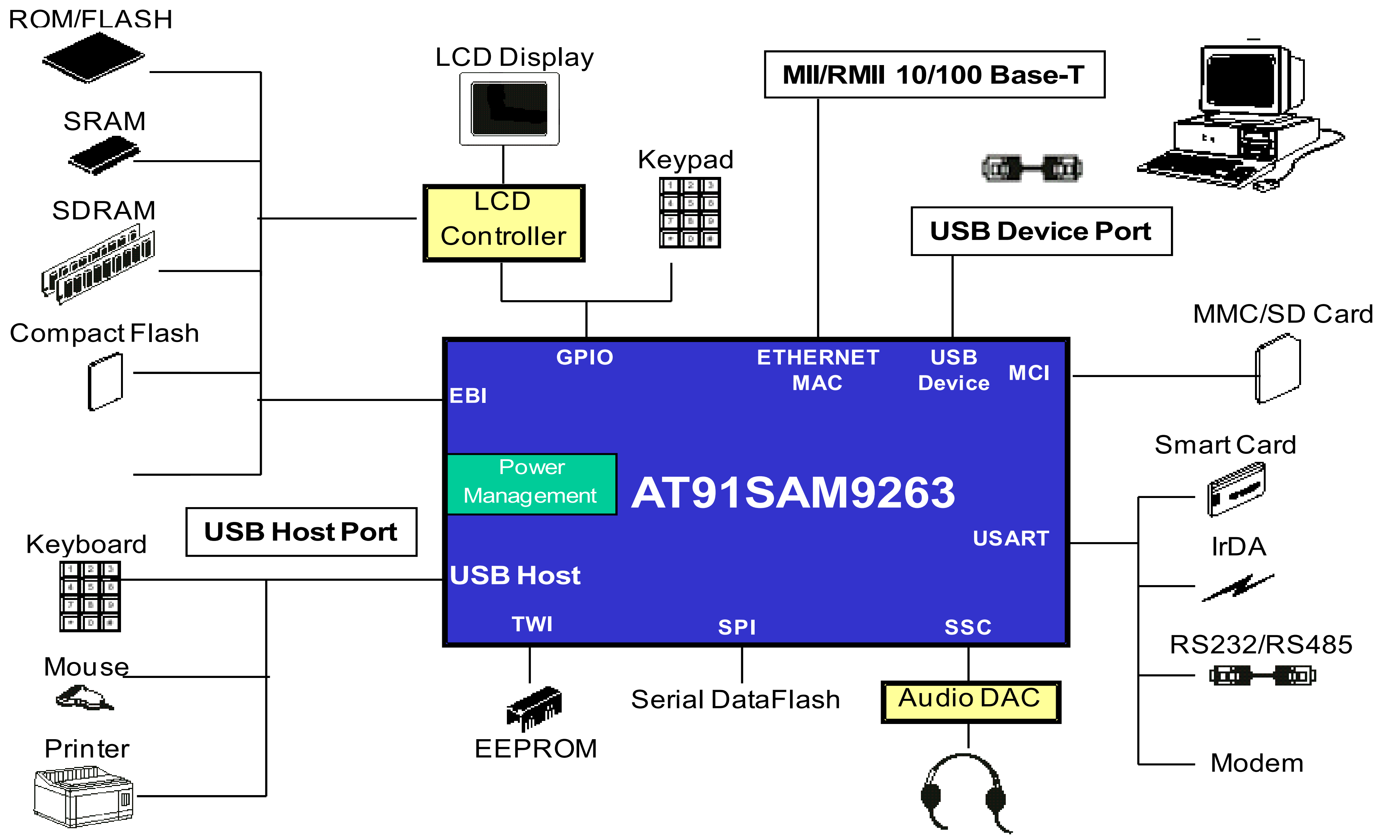
DaVinci DM355

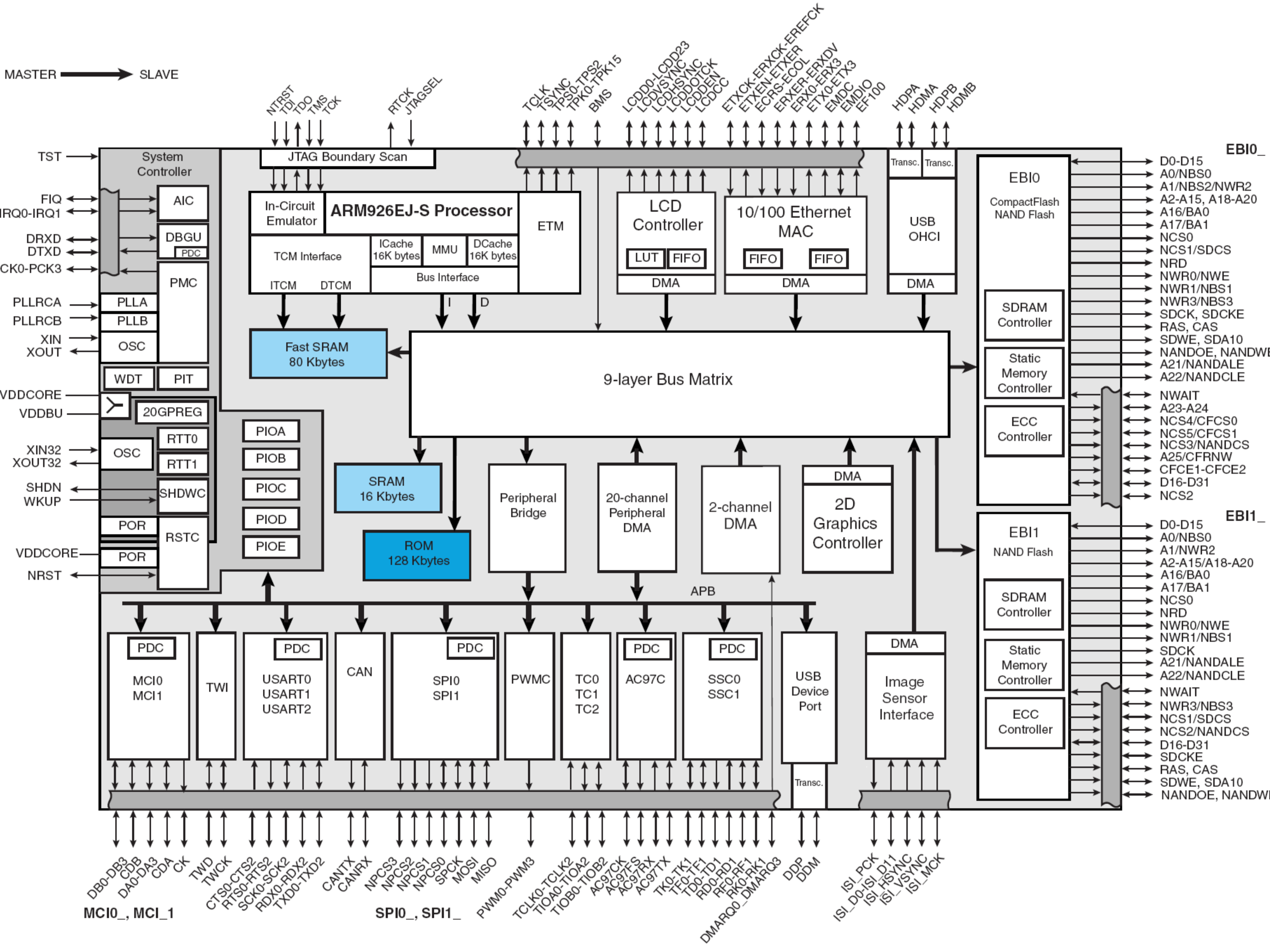
- SoC developed by Texas Instruments company
- Dedicated co-processor for image and sound processing in real-time
- Low power consumption 400 mW during decoding HD MPEG4, 1 mW in standby mode (mobile systems)
- Rich interfaces and peripheral devices (HDD, SD/MMC controllers, USB, Ethernet,...)



Źródło: www.ti.com

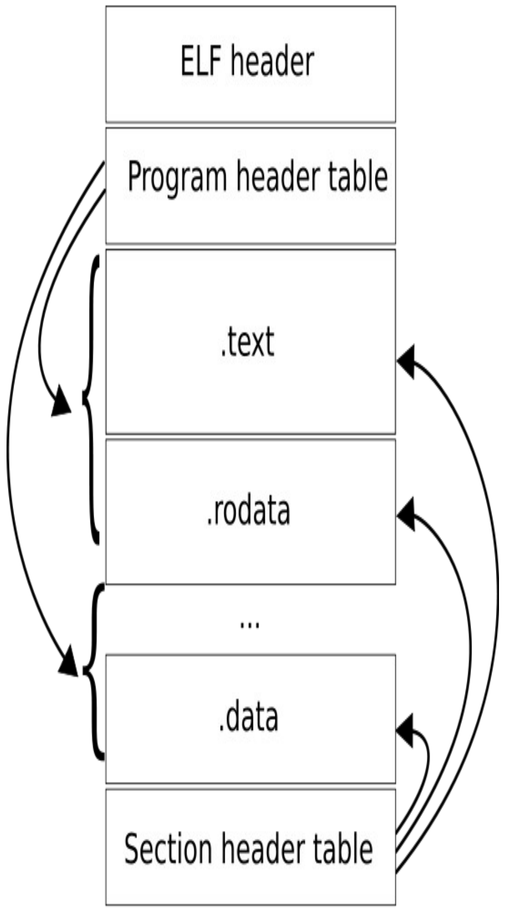
Microcontroller AT91SAM9263 (3)




MASTER
SLAVE
TST
FIQ
IRQ0-IRQ1
DRXD
DTXD
PCK0-PCK3
PLLRCA
PLLRCB
XIN
XOUT
VDDCORE
VDDBU
XIN32
XOUT32
SHDN
WKUP
VDDCORE
NRST
System Controller
AIC
DBGU
PDC
PMC
PLLA
PLLB
OSC
WDT
PIT
20GPREG
RTT0
RTT1
OSC
SHDWC
POR
RSTC
POR
PIOA
PIOB
PIOC
PIOD
PIOE
NTRST
TDI
TDO
TMS
TCK
RTCK
JTAGSEL
JTAG Boundary Scan
In-Circuit Emulator
ARM926EJ-S Processor
TCM Interface
ITCM
DTCM
ICache 16K bytes
MMU
DCache 16K bytes
Bus Interface
I
D
ETM
TCLK
TSYNC
TPS0-TPS2
TPK0-TPK15
BMS
LCDD0-LCDD23
LCDVSYNC
LCDHSYNC
LCDDOTCK
LCDDEN
LCDCC
LCD Controller
LUT
FIFO
DMA
ETXCK-ERXCK-EREFCK
ETXEN-ETXER
ECRS-ECOL
ERXER-ERXDV
ERX0-ERX3
ETX0-ETX3
EMDC
EMDIO
EF100
10/100 Ethernet MAC
FIFO
FIFO
DMA
HDPA
HDMA
HDPB
HDMB
Transc.
Transc.
USB OHCI
DMA
Fast SRAM 80 Kbytes
9-layer Bus Matrix
SRAM 16 Kbytes
ROM 128 Kbytes
Peripheral Bridge
20-channel Peripheral DMA
2-channel DMA
DMA
2D Graphics Controller
APB
EBI0_
EBI0
CompactFlash NAND Flash
SDRAM Controller
Static Memory Controller
ECC Controller
D0-D15
A0/NBS0
A1/NBS2/NWR2
A2-A15, A18-A20
A16/BA0
A17/BA1
NCS0
NCS1/SDCS
NRD
NWR0/NWE
NWR1/NBS1
NWR3/NBS3
SDCK, SDCKE
RAS, CAS
SDWE, SDA10
NANDOE, NANDWE
A21/NANDALE
A22/NANDCLE
NWAIT
A23-A24
NCS4/CFCS0
NCS5/CFCS1
NCS3/NANDCS
A25/CFRNW
CFCE1-CFCE2
D16-D31
NCS2
EBI1_
EBI1
NAND Flash
SDRAM Controller
Static Memory Controller
ECC Controller
D0-D15
A0/NBS0
A1/NWR2
A2-A15/A18-A20
A16/BA0
A17/BA1
NCS0
NRD
NWR0/NWE
NWR1/NBS1
SDCK
A21/NANDALE
A22/NANDCLE
NWAIT
NWR3/NBS3
NCS1/SDCS
NCS2/NANDCS
D16-D31
SDCKE
RAS, CAS
SDWE, SDA10
NANDOE, NANDWE
PDC
MCI0
MCI1
TWI
PDC
USART0
USART1
USART2
CAN
PDC
SPI0
SPI1
PWMC
TC0
TC1
TC2
PDC
AC97C
PDC
SSC0
SSC1
USB Device Port
Transc.
DMA
Image Sensor Interface
DB0-DB3
CDB
DA0-DA3
CDA
CK
TWD
TWCK
CTS0-CTS2
RTS0-RTS2
SCK0-SCK2
RDX0-RDX2
TXD0-TXD2
CANTX
CANRX
NPCS3
NPCS2
NPCS1
NPCS0
SPCK
MOSI
MISO
PWM0-PWM3
TCLK0-TCLK2
TIOA0-TIOA2
TIOB0-TIOB2
AC97CK
AC97FS
AC97RX
AC97TX
TK0-TK1
TF0-TF1
TD0-TD1
RD0-RD1
RF0-RF1
RK0-RK1
DMARQ0_DMARQ3
DDP
DDM
ISI_PCK
ISI_D0-ISI_D11
ISI_HSYNC
ISI_VSYNC
ISI_MCK
MCI0_, MCI_1
SPI0_, SPI1_

COFF vs ELF

- **COFF (Common Object File Format) –** standard of executable, relocable files or dynamic libraries used in Linux systems. COFF was created to substitute the old **a.out** format. COFF is used in different systems, also Windows. Currently COFF standard is supplanted by files compatible with ELF format.
- **ELF (Executable and Linkable Format) –** standard of executable, relocable files, dynamic libraries or memory dumps used in different computers and operating systems, e.g.: x86, PowerPC, OpenVMS, BeOS, PlayStation, Portable, PlayStation 2, PlayStation 3, Wii, Nintendo DS, GP2X, AmigaOS 4 and Symbian OS v9.
- **Useful tools:**
  - readelf
  - elfdump
  - objdump



GDB debugger

arm-elf-gdb <filename.elf>

run – run program (load and run), load – load program

c (continue) – continue execution of program

b (breakpoint) – set breakpoint, e.g. b 54, b main, b sleep

n (next) – execute next function go to next function

s (step) – execute next function, stop in function

d (display) – display variable/register, disp Counter, disp $r0

p (print) – print (only once) variable/register

x – display memory region, e.g. **x/10x 0xFFFF.F000**

i (info) – display data describing breaks in program registers

Modifications:

/x – display data in hexadecimal format

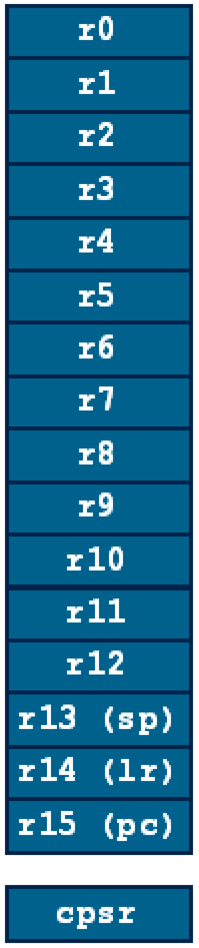
/t – display data in binary format

/d – display data in decimal format

Processor registers and GDB

(gdb) info r

```
r0      0x2          0x2
r1      0x20000ba4   0x20000ba4
r2      0x57b        0x57b
r3      0x270f       0x270f
r4      0x300069     0x300069
r5      0x3122dc     0x3122dc
r6      0x1000       0x1000
r7      0x800bc004   0x800bc004
r8      0x3122c4     0x3122c4
r9      0x407c81a4   0x407c81a4
r10     0x441029ab   0x441029ab
r11     0x313f2c     0x313f2c
r12     0x313f30     0x313f30
sp      0x313f18     0x313f18
lr      0x20000a7c   0x20000a7c
pc      0x20000474   0x20000474 <delay+60>
fps     0x0          0x0
cpsr    0x80000053   0x80000053
```



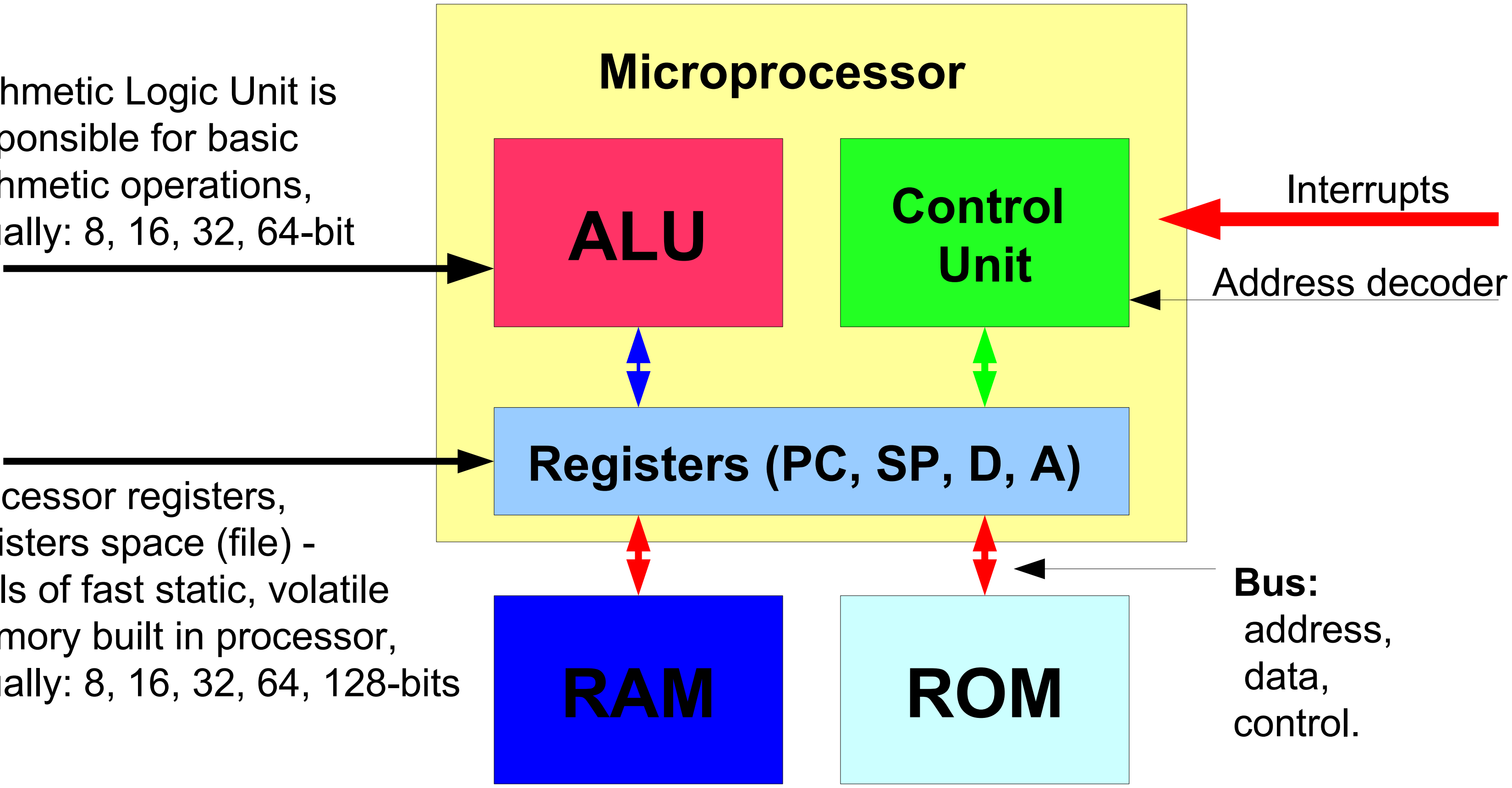
Lecture Agenda

- Microprocessor systems, embedded systems
- Peripheral devices
- Memories and address decoders
- ARM processor as platform for embedded programs
- Methodology of designing embedded systems
- Interfaces in embedded systems
- Real-time microprocessor systems

Microprocessor

Microprocessor - digital circuit fabricated as a single integrated device able to process digital operations according to provided binary program

Arithmetic Logic Unit is responsible for basic arithmetic operations, usually: 8, 16, 32, 64-bit



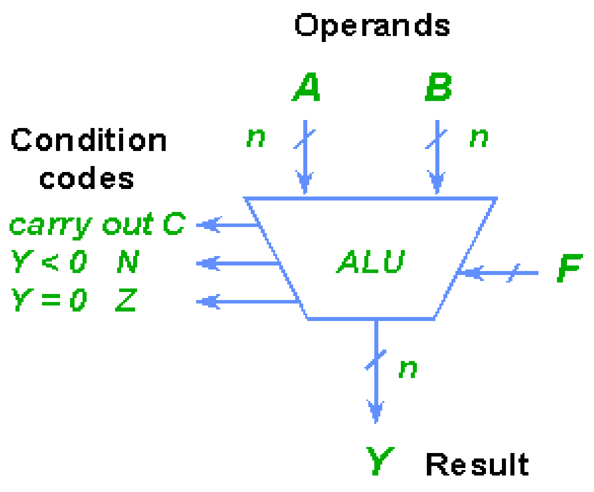
Processor registers, registers space (file) - Cells of fast static, volatile memory built in processor, usually: 8, 16, 32, 64, 128-bits

Bus:
address,
data,
control.

Arithmetic Logic Unit

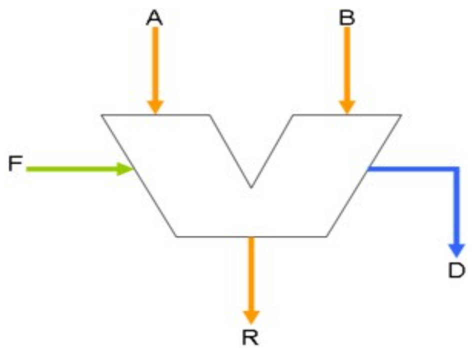
ALU is used to realise the following operations:

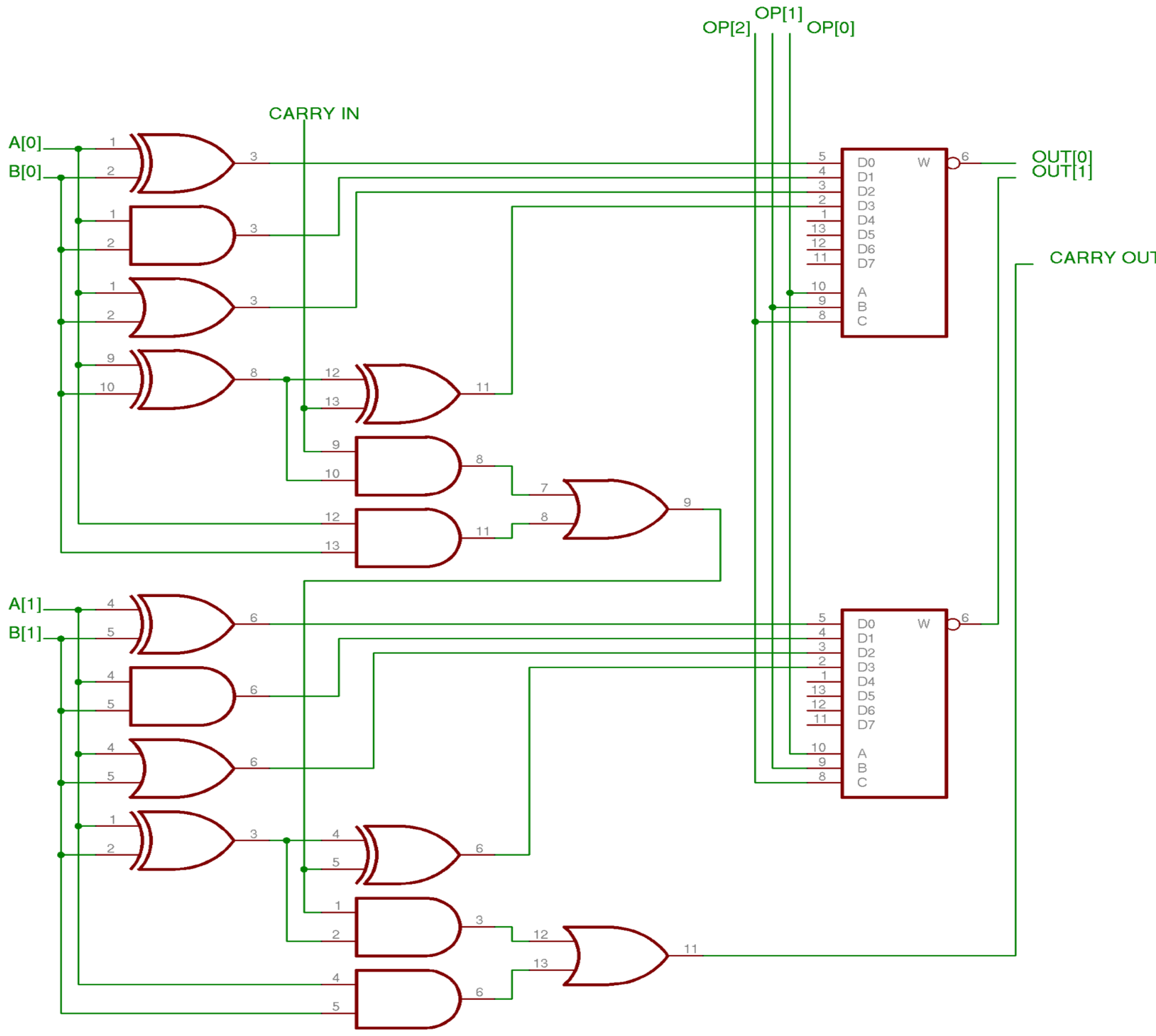
- logical operations AND, OR, NOT, XOR,
- addition,
- subtraction (negate number, add with carry),
- increment/decrement by 1,
- bit-shifting by constant number of bits,
- Multiply and/or division (division modulo).



F	Y
000	A + B
001	A - B
010	A - 1
011	A and B
100	A or B
101	A * B

Two-bit ALU





ALU operations:

- OP = 000 → XOR
- OP = 001 → AND
- OP = 010 → OR
- OP = 011 → ADD

Others operations:

- subtraction,
- multiplication,
- division,
- NOT A,
- NOT B

Computer Architecture (1)

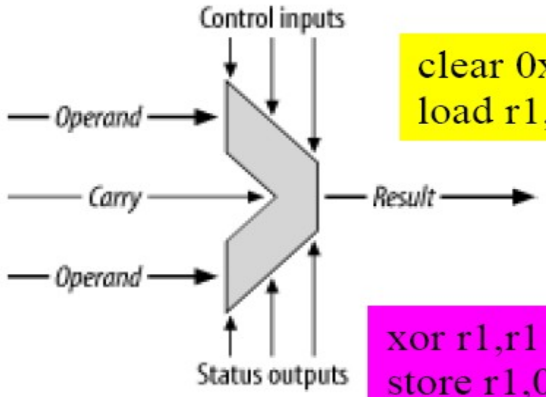
Computer architecture is the conceptual design and fundamental operational structure of a computer system

Computer Architecture defines:

- Programming model of processor - Instruction Set Architecture and others features important from programmer point of view. It is not important how it is realised in the processor, this is barrier between hardware and program layers.
- Microarchitecture of processor, hardware implementation of the given programming model of processor, defines how basic operations/command are executed by processor with special consideration of internal processor design.

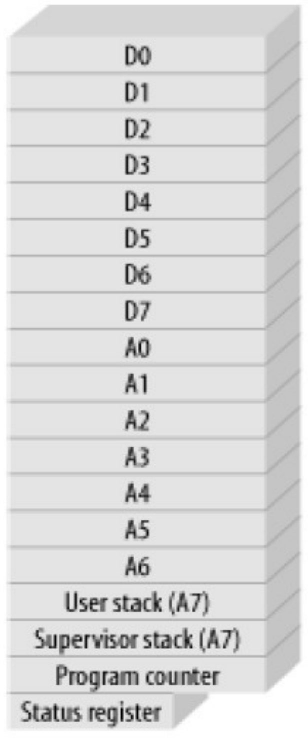
Computer Architecture (2)

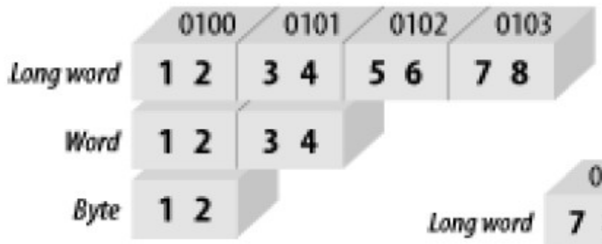
- ALU:
  - supported operations
- Basic registers:
  - PC, Data/Address, Status register
- List of command:
  - RISC/CISC
- Addressing modes:
  - Direct/Indirect/etc...
- Interrupts
  - Hardware/Software
- Endianess architecture
  - Big/Little endian

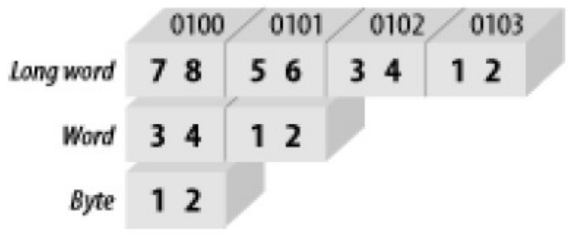


clear 0x1000
load r1,#5

xor r1,r1
store r1,0x1000
add r1,#5







CISC Architecture

Features of CISC architecture (Complex Instruction Set Computers):

- Significant number of commands (instructions),
- Complex and specialised commands,
- Complex commands requires a few machine cycles,
- Significant number of addressing modes,
- Complex addressing modes MOVE.L D1, (PC+A0.D0)++,
- Possible direct access to memory MOVE #4, (0x1000),
- Lower than for RISC clock frequency (lower MIPS),
- Slow complex instruction decoder because of large number of instructions and complex addressing modes

CISC family examples:

- x86
- **M68000**
- PDP-11
- AMD

RISC Architecture

Features of RISC architecture (Reduced Instruction Set Computers):

- Reduced and optimised number of instructions. Simple command decoder,
- Reduced addressing modes, faster instructions,
- Limited communication with ALU, dedicated commands for data transfer between registers and memory, e.g. MOVE #5, D0, STORE D0, (1000).
- Increased number of registers (e.g. 32, 192, 256),
- Pipeline processing – most of commands needs only single machine cycle

RISC family examples:

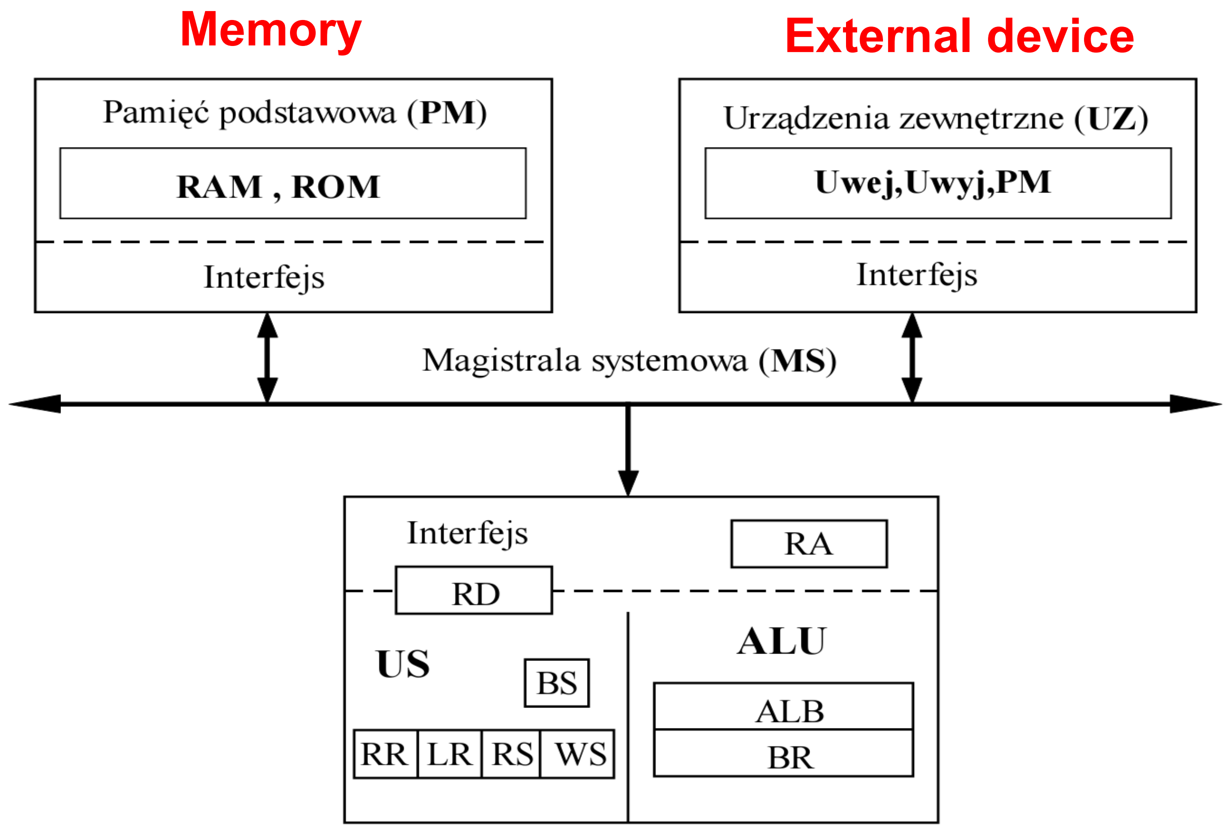
- IBM 801
- PowerPC
- MIPS
- Alpha
- **ARM**
- Motorola 88000
- ColdFire
- SPARC
- PA-RISC
- Atmel_AVR

Currently, most of INTEL processors are seen as CISC processors, CISC commands are divided into microoperations and executed by fast RISC core. Compatibility with older processors.

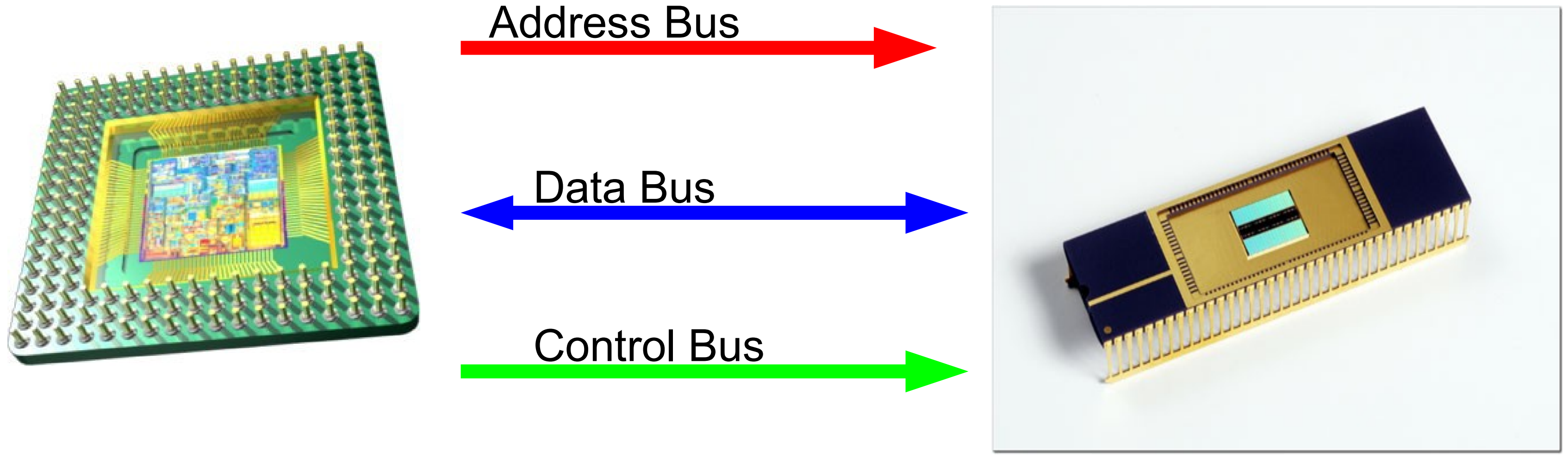
Computer System Architecture

Computer System is composed of three basic subsystems:

- processor,
- memory (data and program),
- Input-output devices (I/O).

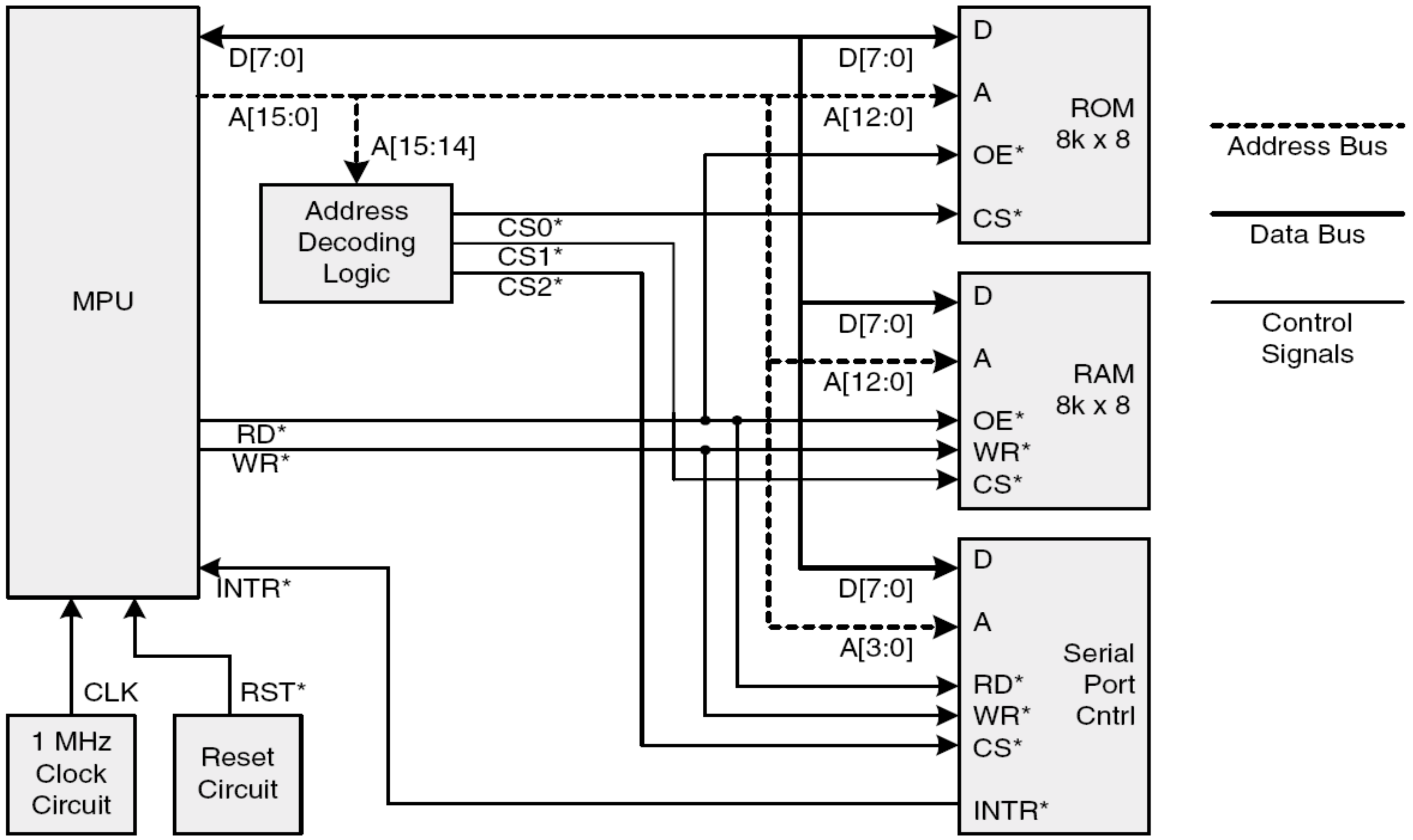


Computer buses



1. Type of Bus ?
2. How wide is the bus ?
3. Maximum data transfer frequency – throughput ?

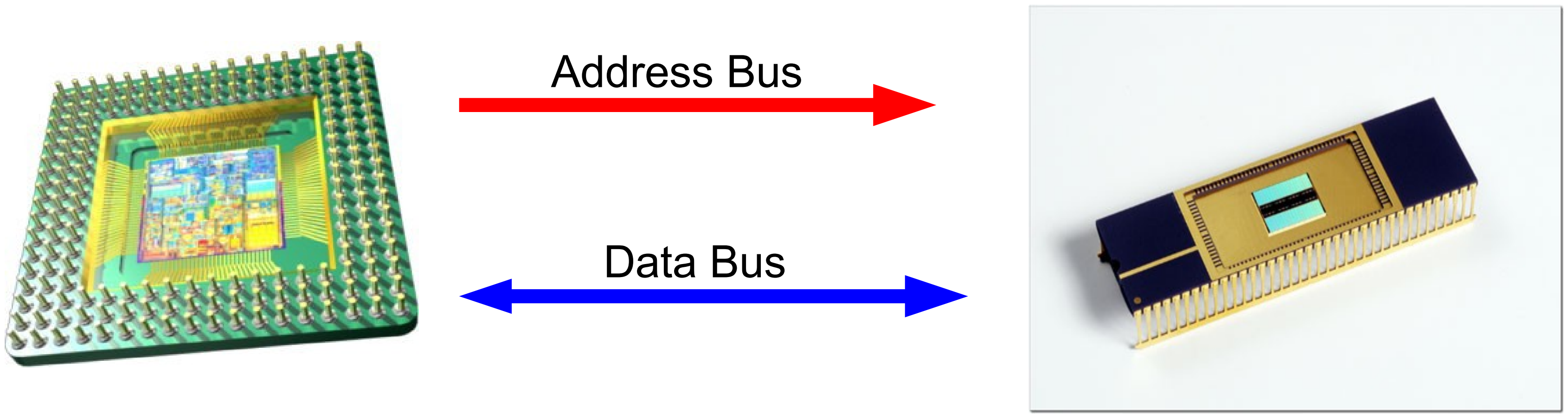
Example of 8-bit computer system



Von Neumann architecture

Von Neumann architecture:

- Data and command stored in the same memory,
- Commands and Data cannot be distinguished,
- Data has no meaning,
- Memory is seen as linear table identified with data address provided by processor
- Processor has access to memory and address decoders (and MMU) maps real memories to memory space.

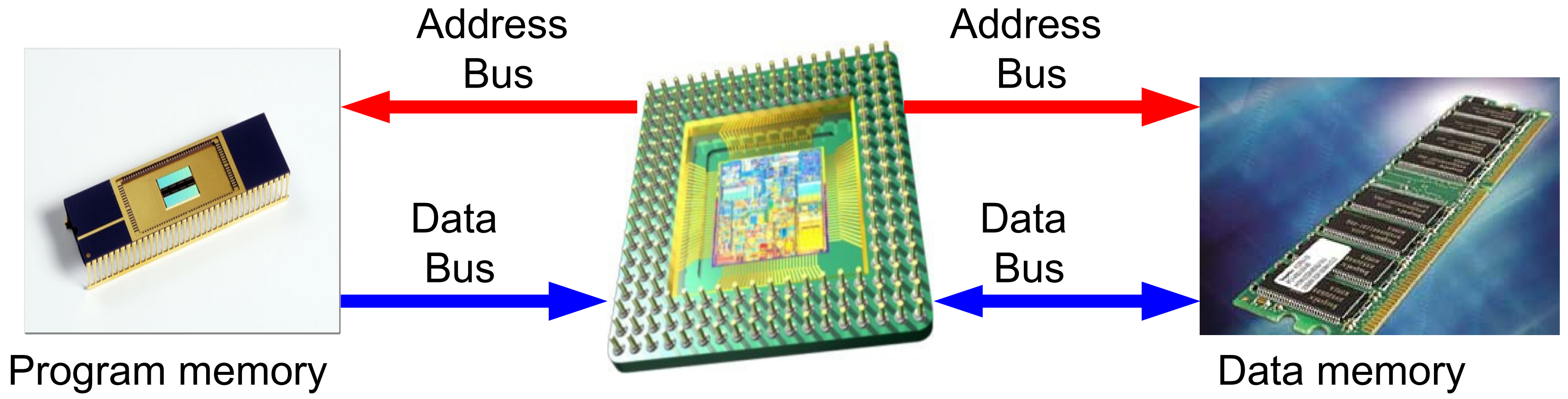


Harvard Architecture

Simple, in comparison to Von Neumann architecture, provides faster microprocessors. Used in DSP (Digital Signal Processors) and cache memories.

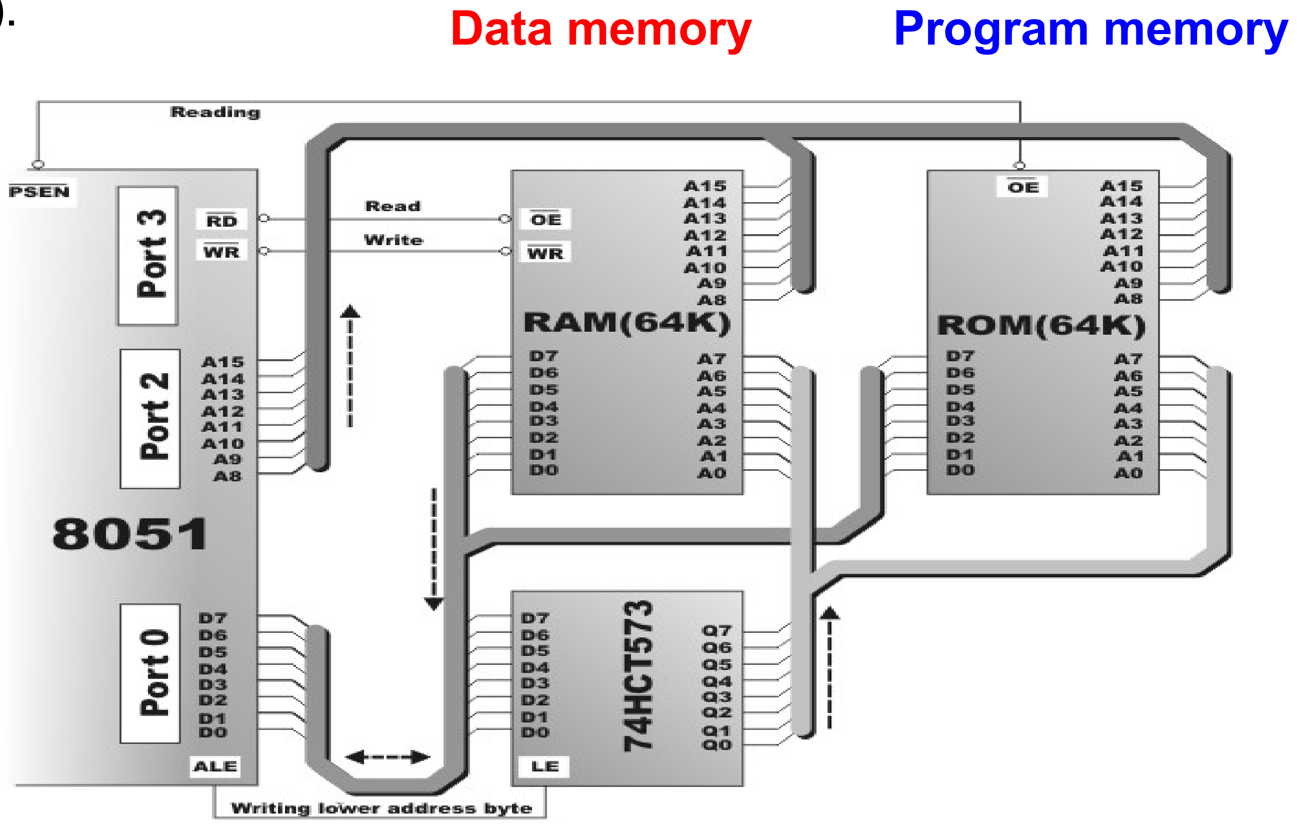
Harvard Architecture**:**

- Command and Data are stored in different memories,
- Allowed different organisation of memories (different data and command words lengths),
- Possible parallel access,
- Used in single-chip Microcontrollers



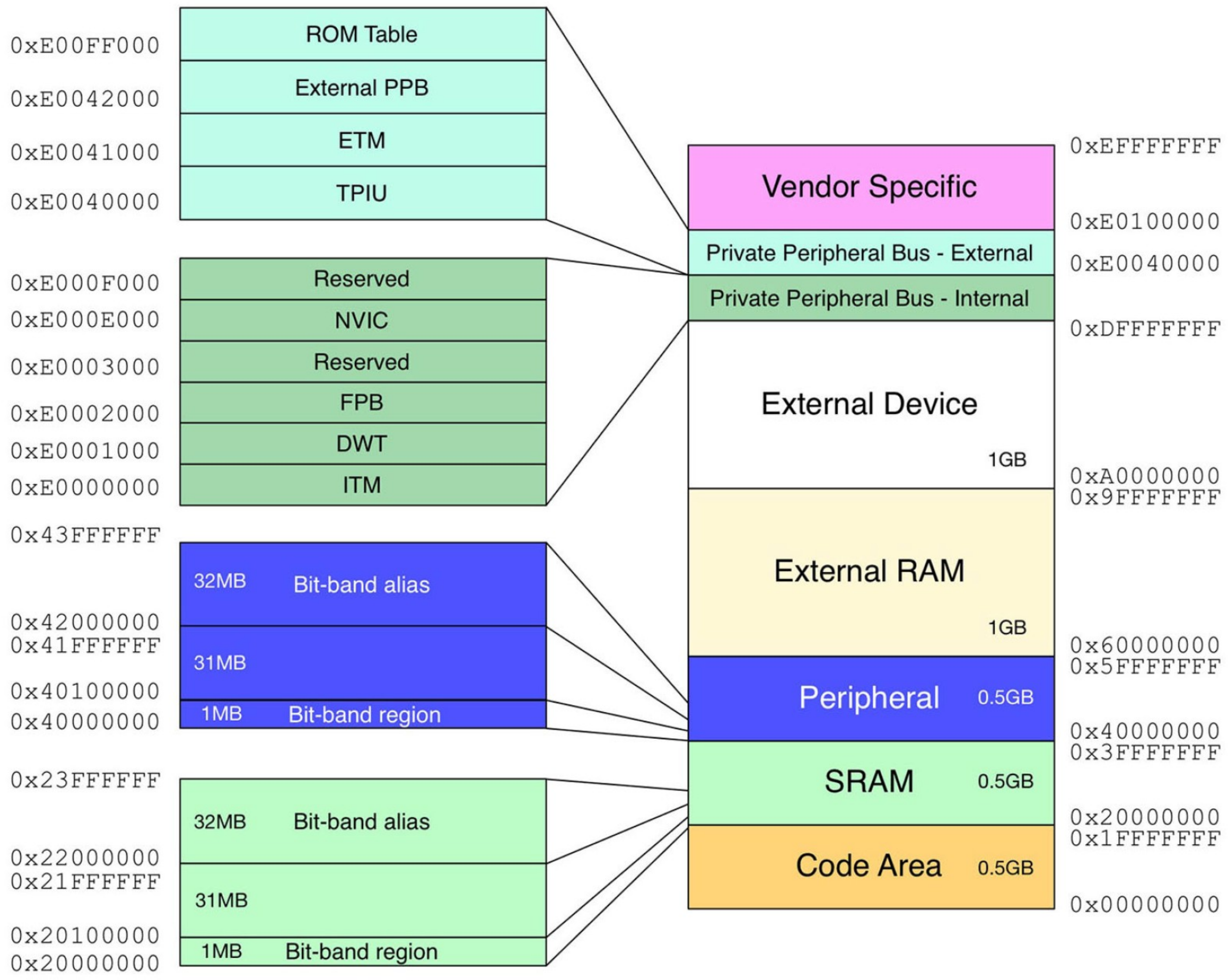
Modified Harvard Architecture

Modified Harvard architecture includes features of both computer architectures, program and data memories are separated however connected via the same buses (data and address).



Example of mixed Harward architecture – 8051 processor with Data and Program memories

Memory Map for Cortex-M Microcontrollers



Registers

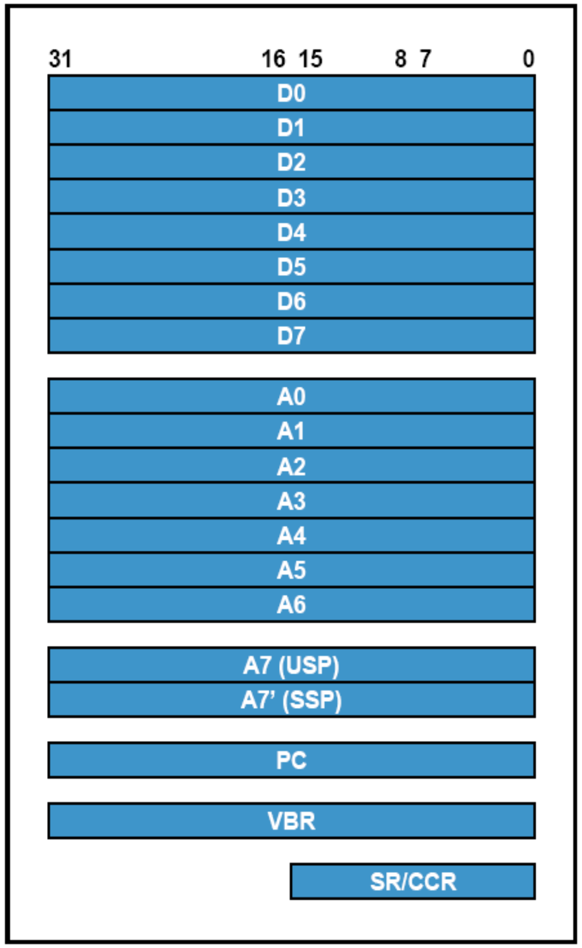
Processor registers are realised as cells of internal processor memory. Registers have usually small size (8/16/32/64/128 bits). Registers are used for storing temporary results, addresses in computer memory, configuration of peripheral devices, etc...

Feature of processor registers:

- The highest level in memory hierarchy (memory with the highest access),
- Implemented as bistable triggers,
- Number of registers depends on the processor type (RISC/CISC).

Registers can be divided into the following groups:

- Data registers – used for storing data and results, e.g. mnemonic arguments, results of calculations,
- Address registers – used for operations on addresses (stack pointer, program counter, segment register, etc...),
- General purpose registers – store both data and addresses,
- Floating point registers – used for operations on floating points registers (FPU coprocessor).



Registers of Freescale/NXP microcontroller

Endianess (1)

Byte – the smallest addressable unit of computer memory

Endianess

Big-endian	middle-endian	Little-endian
...the lowest address contains the most significant byte...		...the lowest address contains the least significant byte...
Polish, English language	Floating, double precision numbers	Arabian, German language
Motorola, SPARC, ARM	VAX and ARM	Intel x86, 6502 VAX

Bi-Endian

ARM, PowerPC (except PPC970/G5), DEC Alpha, MIPS, PA-RISC oraz IA64

Endianess (2)

8-bit data, **Byte 1, Byte 2, Byte 3, ...**

	7 0
0x0000.0000	Byte 1
0x0000.0001	Byte 2
0x0000.0002	Byte 3
0x0000.0003	Byte 4
0x0000.0004	Byte 5

	7 0
0x0000.0000	0x01
0x0000.0001	0x02
0x0000.0002	0x03
0x0000.0003	0x04
0x0000.0004	0x05

Endianess (3)

32-bit data, Double Word (DW): **Byte 4 … Byte 1**

Big-endian

	7 0
0x0000.0000	Byte 4
0x0000.0001	Byte 3
0x0000.0002	Byte 2
0x0000.0003	Byte 1
0x0000.0004	Byte 8

Little-endian

	7 0
0x0000.0000	Byte 1
0x0000.0001	Byte 2
0x0000.0002	Byte 3
0x0000.0003	Byte 4
0x0000.0004	Byte 5

Endianess (4)

32-bit data, Double Word (DW): **0x0403.0201**

Big-endian

	7 0
0x0000.0000	0x04
0x0000.0001	0x03
0x0000.0002	0x02
0x0000.0003	0x01
0x0000.0004	0x08

Little-endian

	7 0
0x0000.0000	0x01
0x0000.0001	0x02
0x0000.0002	0x03
0x0000.0003	0x04
0x0000.0004	0x05

Endianess (5)

How to recognize endianess of computer memory?

```
#define LITTLE_ENDIAN   0
#define BIG_ENDIAN      1

int machineEndianness()
{
  long int i = 1;                         /* 32 bit = 0x0000.0001 */
  const char *p = (const char *) &i;      /* Pointer to ......?  */

  if (p[0] == 1)                          /* Lowest address contains the least significant byte */
    return LITTLE_ENDIAN;

  else
    return BIG_ENDIAN;
}
```

Bitwise Operations

Bits and C language

```
volatile unsigned int * DataInMemory = 0x4800.0000;
volatile uint32_t * DataInMemory = 0x1000;
*DataInMemory = 0;
*DataInMemory = 0x12345678;
*DataInMemory = 0x12345678U;
*DataInMemory = 0xFFFF.FFFFU;
```

How to clear single bit ?

How to set single bit ?

Operations on Register Bits (1)

```
volatile unsigned int* GPIOA_PUPDR = 0x4800.000C;
volatile uint32_t * GPIOB_PUPDR = 0x4800.040C;
#define GPIOC_PUPDR (volatile uint32_t *)0x4800.080C
    *GPIOA_PUPDR    = 0x1U;
    *GPIOA_PUPDR    = 7U;
    *GPIOA_PUPDR    = 010U;
    *GPIOA_PUPDR    = *GPIOA_PUPDR  |  0x2U;
    *GPIOA_PUPDR   |= 0x1U | 0x2U | 0x8U ;
    *GPIOA_PUPDR  &=  ~(0x2U | 0x4U);
    *GPIOA_PUPDR   ^=    (0x1U | 0x2U);
    *GPIOA_PUPDR   ^=     0x3U;
    If (*GPIOA_PUPDR & (0x1U | 0x4U)) == 0  {...}
    while (*GPIOA_PUPDR != 0x6U)   {...}
    do {...} while (*GPIOA_PUPDR & 0x4U)
```

Operations on Register Bits (2)

```
#define PB0   0x1U
#define PB1   0x2U
#define PB2   1U<<2
#define PB3   1U<<3

volatile unsigned char* PORTA=0x4010.000A;

*PORTA   |= PB1 | PB2;
*PORTA &=  ~(PB1 | PB2);
*PORTA  ^=  (PB1 | PB2);
If (*PORTA & (PB1 | PB2)) == 0

enum {PB0=1U<<0, PB1=1U<<2, PB2=1U<<3, PB3=1U<<3};
```

Operations on Register Bits (3)

```
volatile unsigned char* PORTA=0x4010.000A;

/* macro for bit-mask */
#define BIT(x)      (1U  << (x))
*PORTA  |=  BIT(0);
*PORTA &=~BIT(1);
*PORTA  ^= BIT(2);

/* macro for setting and clearing bits */
#define SETBIT(P, B)        (P)   |=   BIT(B)
#define CLRBIT(P, B)        (P)  &= ~BIT(B)

SETBIT(*PORTA, 7);
CLRBIT(*PORTA, 2);
```

Register Concatenation

```
int main(void) {
        unsigned char reg1=0x15U, reg2=0x55U;
        unsigned char reg3=0x11, reg4=0x22;
        unsigned int tmp;
        /* concatenation operation */
        tmp = reg1;
        tmp = tmp<<8 | reg2;

        /* operation of  */
        reg3 = tmp>>8;              /* be carefull on numbers with sign*/
        reg4 = tmp & 0xFF;

}
```

Registers Mapped as Structure

```
typedef volatile unsigned int AT91_REG;        // Hardware register definition
typedef struct _AT91S_PIO {
        AT91_REG        PIO_PER;                // PIO Enable Register, 32-bit register
        AT91_REG        PIO_PDR;                // PIO Disable Register
        AT91_REG        PIO_PSR;                // PIO Status Register
        AT91_REG        Reserved1[1];       //
        AT91_REG        PIO_IFER;               // Input Filter Enable Register
        AT91_REG        PIO_IFDR;               // Input Filter Disable Register
        AT91_REG        PIO_IFSR;               // Input Filter Status Register
        AT91_REG        Reserved2[1];       //
} AT91S_PIO, *AT91PS_PIO;
        /* registers for paraller port of ARM processor I/O PIOA...PIOE */
#define AT91C_BASE_PIOA     (AT91PS_PIO)    0xFFFFF200U     // (PIOA) Base Address
        /* mask for zero bit of port PA */
#define AT91C_PIO_PA0       (1 <<  0)       // Pin Controlled by PA0
```

How to set 0 and 19th bith of register PIO_PER ?

```
AT91C_BASE_PIOA->PIO_PER |= AT91C_PIO_PA0 | AT91C_PIO_PA19;
```

Bit-fields – Register Mapped as Structure

```
Struct Port_4bit {
unsigned Bit_0          :    1;
unsigned Bit_1          :    1;
unsigned Bit_2          :    1;
unsigned Bit_3          :    1;
unsigned Bit_Filler     :    4;
};
#define PORTC (*(Port_4bit*)0x4010.0002U)
int i = PORTC.Bit_0;      /* read data */
PORTC.Bit_2 = 1;          /* write data */

Port_4bit* PortTC = (Port_4bit*) 0x4010.000FU;
int i = PortTC->Bit_0;
PortTC->Bit_0 = 1;
```

- Bit-fields allows to 'pack' data – usage of single bits, e.g. bit flags
- Increase of code complexity required for operations on registers
- Bit-fields can be mapped in different ways in memory according different compilers and processors architectures
- Cannot use ***offsetof*** macro to calculate data offset in structure
- Cannot use ***sizeof*** macro to calculate size of data
- Tables cannot use bit-fields

Union – Registers With Different Functionality

```
extern volatile union {
        struct {
                unsigned EID16          :1;
                unsigned EID17          :1;
                unsigned                :1;
                unsigned EXIDE          :1;
                unsigned                :1;
                unsigned SID0           :1;
                unsigned SID1           :1;
                unsigned SID2           :1;
        };
        struct {
                unsigned                :3;
                unsigned EXIDEN         :1;
        };
} RXF3SIDLbits_;
```

Structures have the same address:

```
#define RXF3SIDLbits
    (*(Port_RXF3SIDLbits_*)0x4010.0000)
```

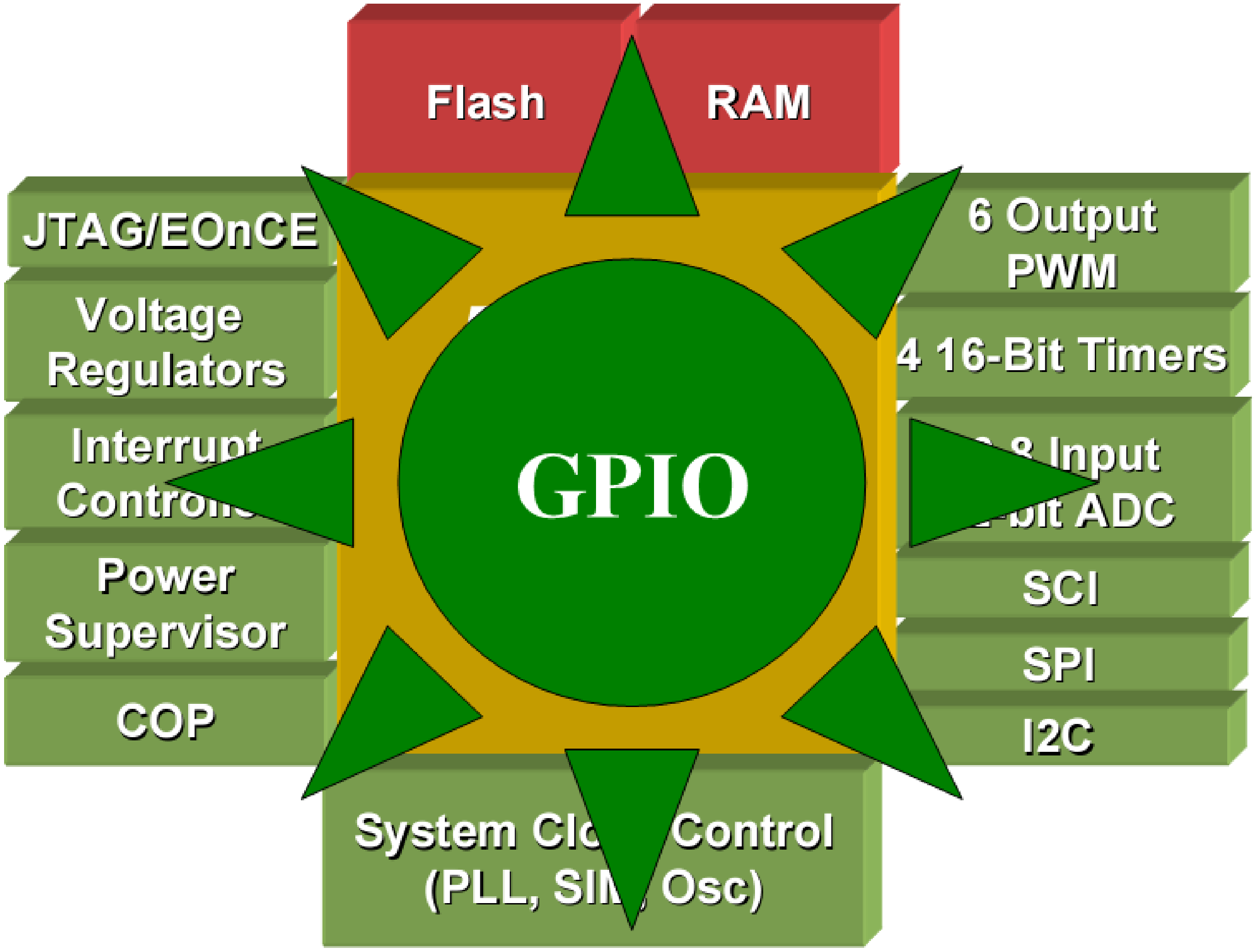
Access to data mapped into structure:

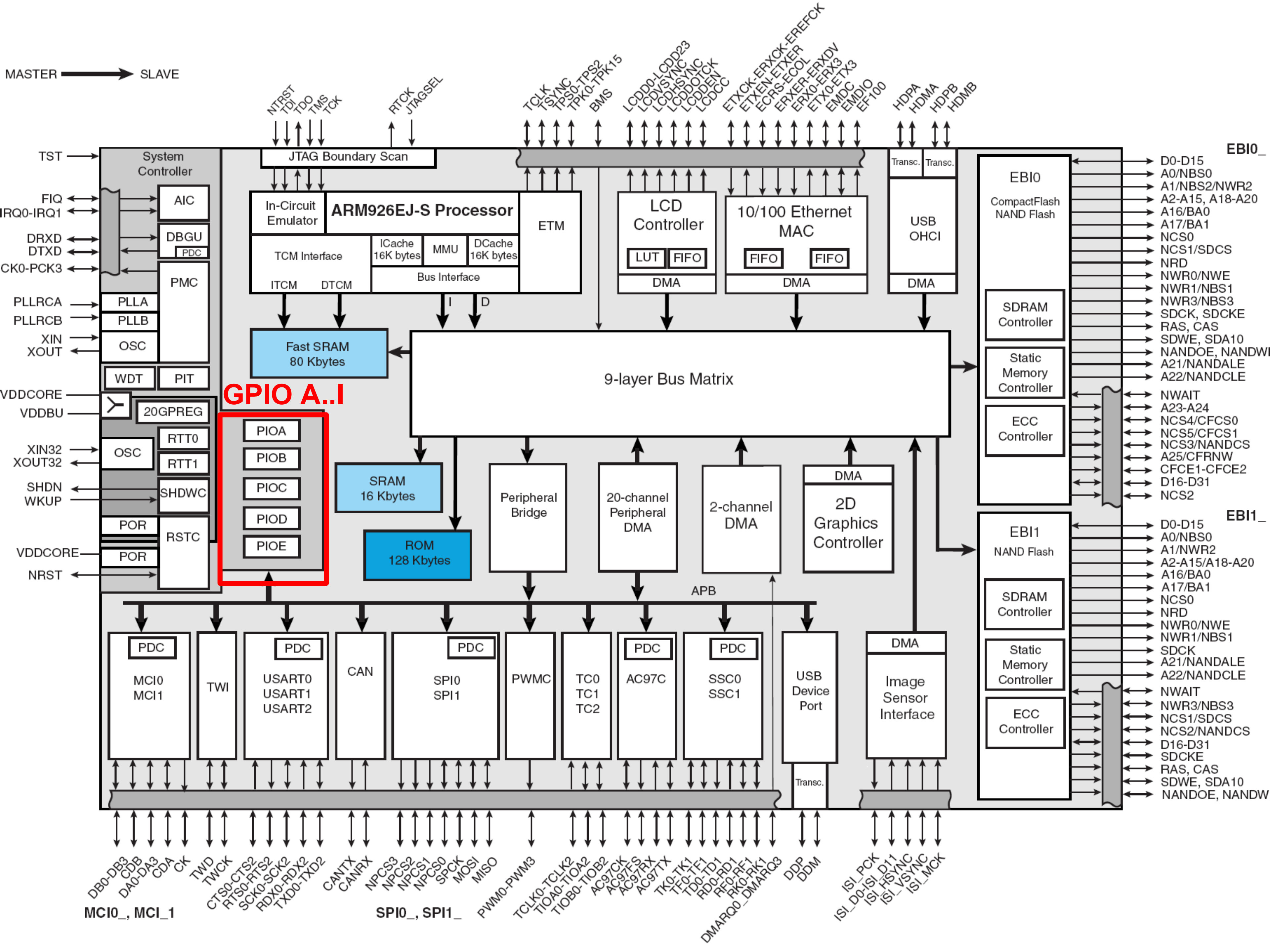
```
/* data in first structure */
        RXF3SIDLbits.EID16 = 1;

/* data in second structure */
        RXF3SIDLbits.EXIDEN = 0;
```

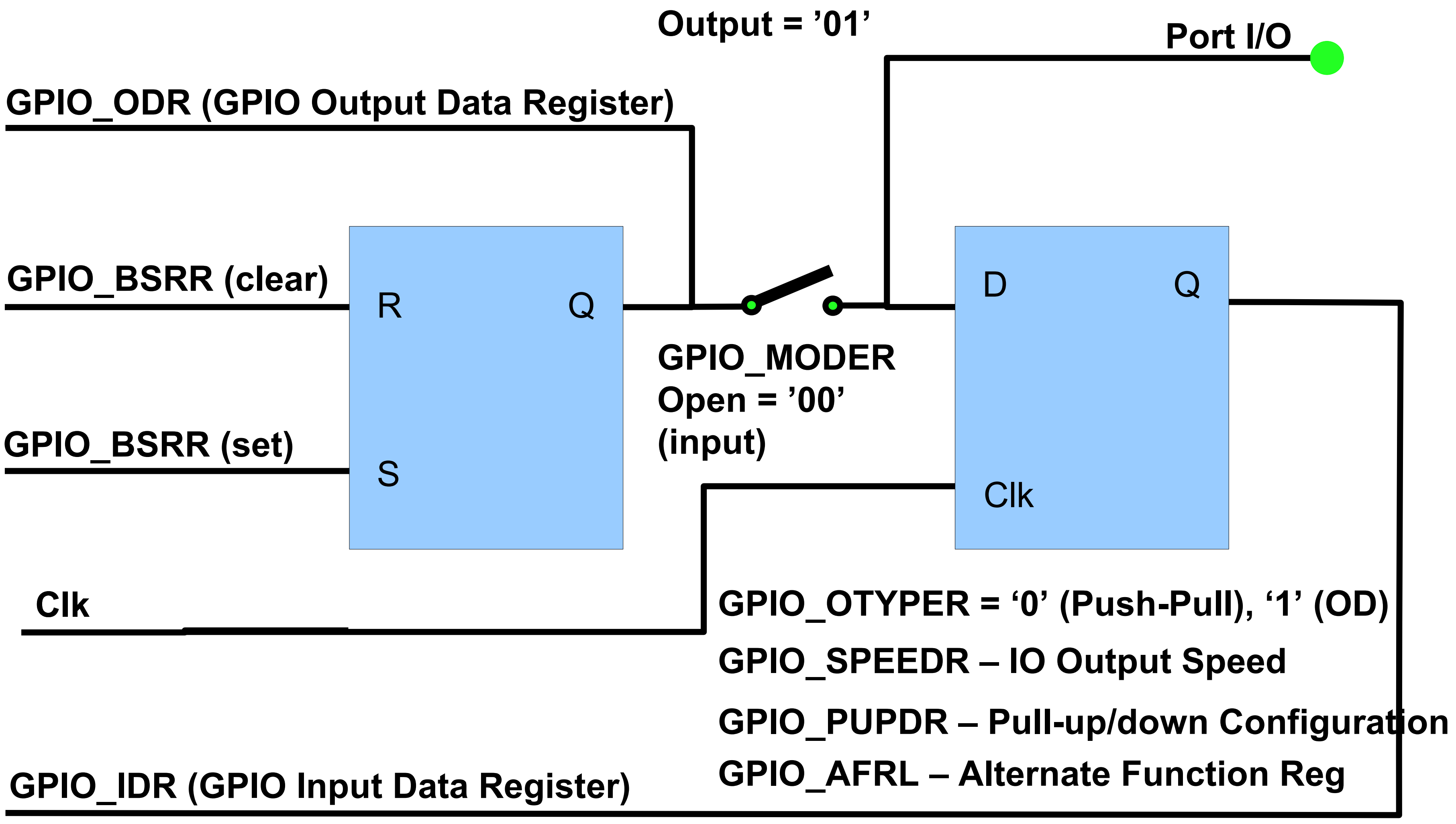
Input-Output ports of ARM processor

IO Port Module

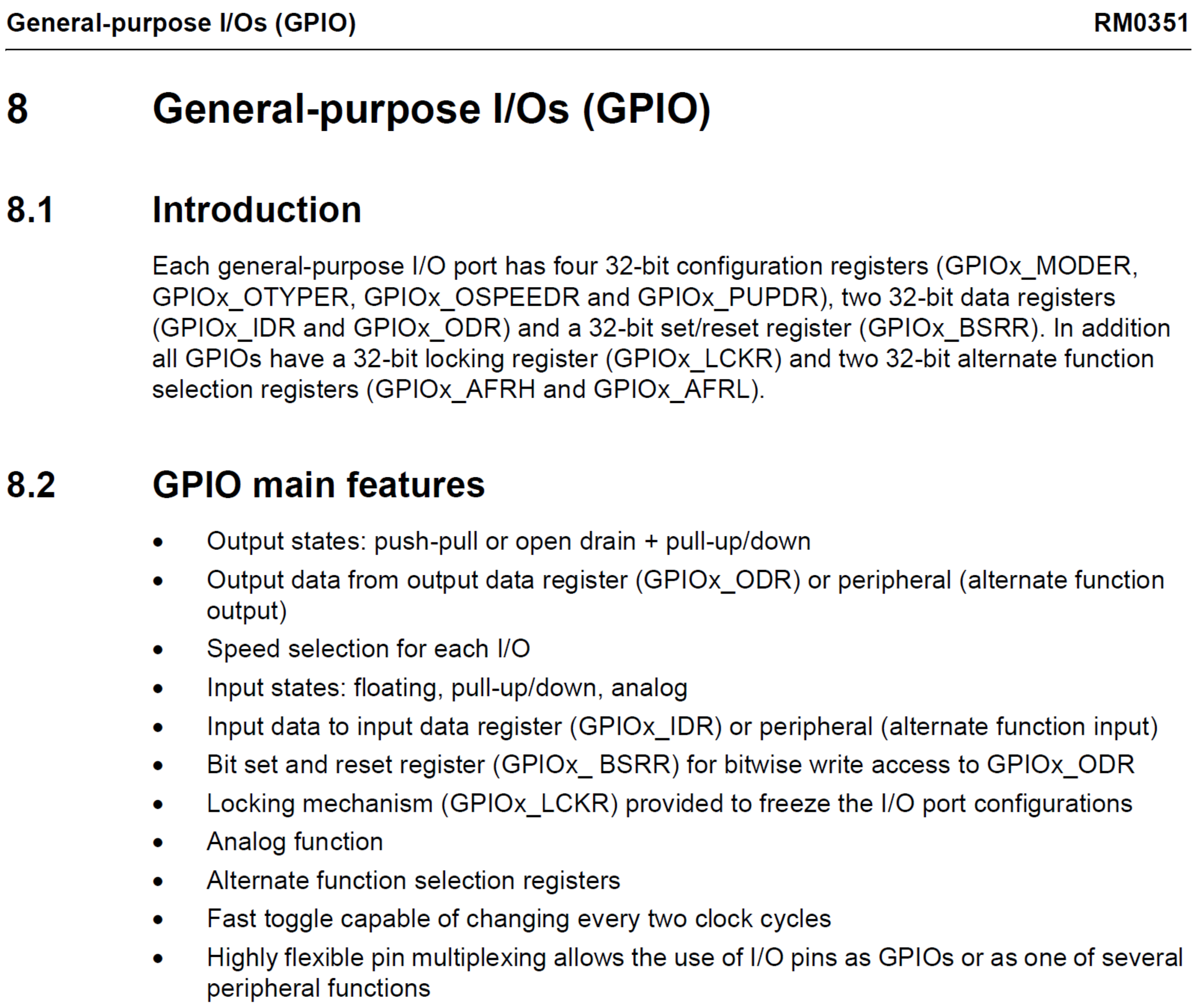



MASTER
SLAVE
NTRST
TDI
TDO
TMS
TCK
RTCK
JTAGSEL
TCLK
TSYNC
TPS0-TPS2
TPK0-TPK15
BMS
LCDD0-LCDD23
LCDVSYNC
LCDHSYNC
LCDDOTCK
LCDDEN
LCDCC
ETXCK-ERXCK-EREFCK
ETXEN-ETXER
ECRS-ECOL
ERXER-ERXDV
ERX0-ERX3
ETX0-ETX3
EMDC
EMDIO
EF100
HDPA
HDMA
HDPB
HDMB
EBI0_
TST
System Controller
FIQ
IRQ0-IRQ1
DRXD
DTXD
CK0-PCK3
PLLRCA
PLLRCB
XIN
XOUT
VDDCORE
VDDBU
XIN32
XOUT32
SHDN
WKUP
VDDCORE
NRST
AIC
DBGU
PDC
PMC
PLLA
PLLB
OSC
WDT
PIT
20GPREG
RTT0
OSC
RTT1
SHDWC
POR
RSTC
POR
JTAG Boundary Scan
In-Circuit Emulator
ARM926EJ-S Processor
ETM
TCM Interface
ICache 16K bytes
MMU
DCache 16K bytes
ITCM
DTCM
Bus Interface
I
D
LCD Controller
LUT
FIFO
DMA
10/100 Ethernet MAC
FIFO
FIFO
DMA
Transc.
Transc.
USB OHCI
DMA
Fast SRAM 80 Kbytes
9-layer Bus Matrix
GPIO A..I
PIOA
PIOB
PIOC
PIOD
PIOE
SRAM 16 Kbytes
ROM 128 Kbytes
Peripheral Bridge
20-channel Peripheral DMA
2-channel DMA
DMA
2D Graphics Controller
APB
EBI0
CompactFlash NAND Flash
SDRAM Controller
Static Memory Controller
ECC Controller
D0-D15
A0/NBS0
A1/NBS2/NWR2
A2-A15, A18-A20
A16/BA0
A17/BA1
NCS0
NCS1/SDCS
NRD
NWR0/NWE
NWR1/NBS1
NWR3/NBS3
SDCK, SDCKE
RAS, CAS
SDWE, SDA10
NANDOE, NANDWE
A21/NANDALE
A22/NANDCLE
NWAIT
A23-A24
NCS4/CFCS0
NCS5/CFCS1
NCS3/NANDCS
A25/CFRNW
CFCE1-CFCE2
D16-D31
NCS2
EBI1_
EBI1
NAND Flash
SDRAM Controller
Static Memory Controller
ECC Controller
D0-D15
A0/NBS0
A1/NWR2
A2-A15/A18-A20
A16/BA0
A17/BA1
NCS0
NRD
NWR0/NWE
NWR1/NBS1
SDCK
A21/NANDALE
A22/NANDCLE
NWAIT
NWR3/NBS3
NCS1/SDCS
NCS2/NANDCS
D16-D31
SDCKE
RAS, CAS
SDWE, SDA10
NANDOE, NANDW
PDC
MCI0
MCI1
TWI
PDC
USART0
USART1
USART2
CAN
PDC
SPI0
SPI1
PWMC
TC0
TC1
TC2
PDC
AC97C
PDC
SSC0
SSC1
USB Device Port
Transc.
DMA
Image Sensor Interface
DB0-DB3
CDB
DA0-DA3
CDA
CK
TWD
TWCK
CTS0-CTS2
RTS0-RTS2
SCK0-SCK2
RDX0-RDX2
TXD0-TXD2
CANTX
CANRX
NPCS3
NPCS2
NPCS1
NPCS0
SPCK
MOSI
MISO
PWM0-PWM3
TCLK0-TCLK2
TIOA0-TIOA2
TIOB0-TIOB2
AC97CK
AC97FS
AC97RX
AC97TX
TK0-TK1
TF0-TF1
TD0-TD1
RD0-RD1
RF0-RF1
RK0-RK1
DMARQ0_DMARQ3
DDP
DDM
ISI_PCK
ISI_D0-ISI_D11
ISI_HSYNC
ISI_VSYNC
ISI_MCK
MCI0_, MCI_1
SPI0_, SPI1_

Simplified Block Diagram of I/O Port


Output = '01'
Port I/O
GPIO_ODR (GPIO Output Data Register)
GPIO_BSRR (clear)
R
Q
GPIO_BSRR (set)
S
GPIO_MODER
Open = '00'
(input)
D
Q
Clk
Clk
GPIO_OTYPER = '0' (Push-Pull), '1' (OD)
GPIO_SPEEDR – IO Output Speed
GPIO_PUPDR – Pull-up/down Configuration
GPIO_AFRL – Alternate Function Reg
GPIO_IDR (GPIO Input Data Register)

IO Ports Controller (1)



General-purpose I/Os (GPIO) RM0351

8 General-purpose I/Os (GPIO)

8.1 Introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR and GPIOx_PUPDR), two 32-bit data registers (GPIOx_IDR and GPIOx_ODR) and a 32-bit set/reset register (GPIOx_BSRR). In addition all GPIOs have a 32-bit locking register (GPIOx_LCKR) and two 32-bit alternate function selection registers (GPIOx_AFRH and GPIOx_AFRL).

8.2 GPIO main features

- Output states: push-pull or open drain + pull-up/down
- Output data from output data register (GPIOx_ODR) or peripheral (alternate function output)
- Speed selection for each I/O
- Input states: floating, pull-up/down, analog
- Input data to input data register (GPIOx_IDR) or peripheral (alternate function input)
- Bit set and reset register (GPIOx_ BSRR) for bitwise write access to GPIOx_ODR
- Locking mechanism (GPIOx_LCKR) provided to freeze the I/O port configurations
- Analog function
- Alternate function selection registers
- Fast toggle capable of changing every two clock cycles
- Highly flexible pin multiplexing allows the use of I/O pins as GPIOs or as one of several peripheral functions

Sterownik portów wejścia-wyjścia (2)

Each of the IO ports can work in one of the following modes:

Input

- **Floating input**
- **Input** with VCC **pull-up**
- **Input** with GND **pull-down**
- **Analogue input**

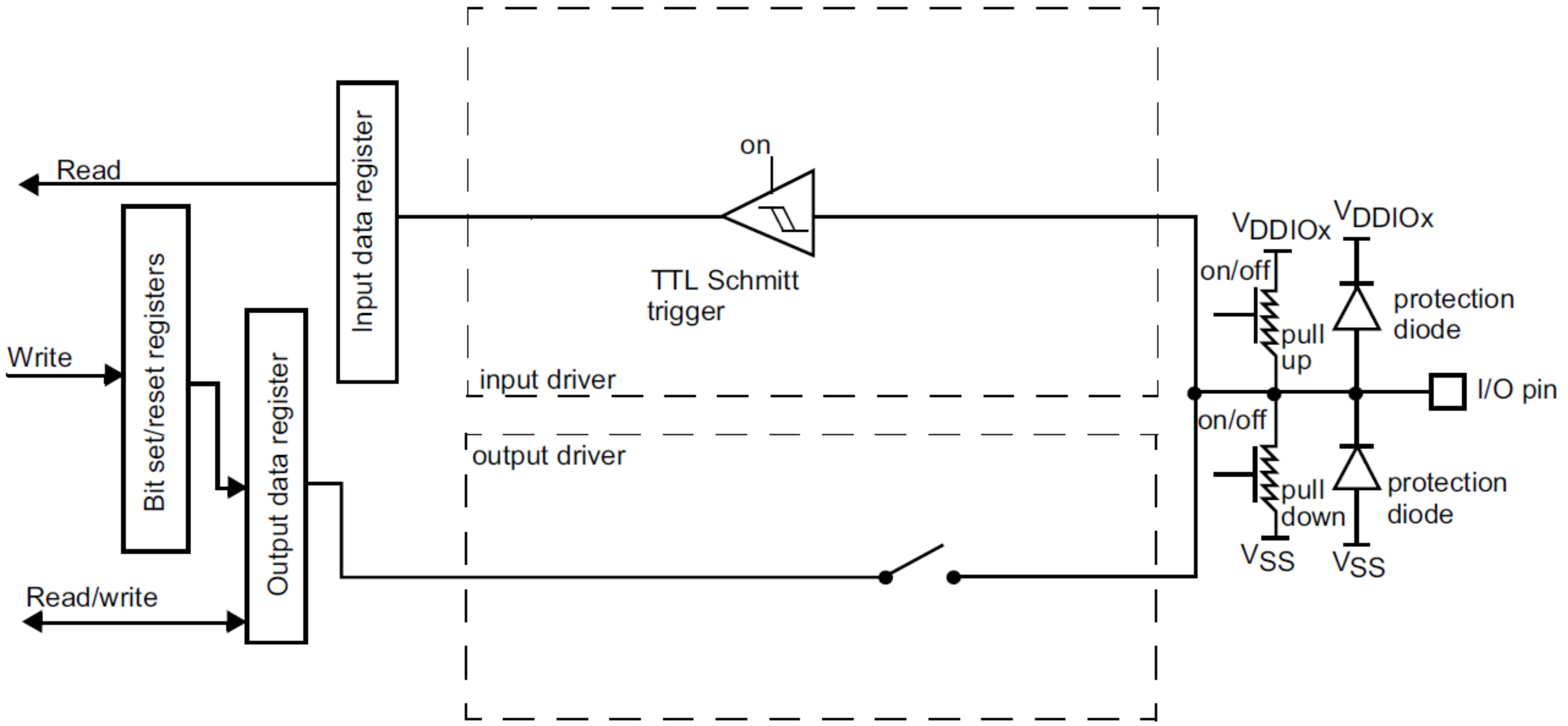
Output

- **Open drain output** (with optional pull-up or pull-down function)
- **Push-pull output** (with optional pull-up or pull-down function)

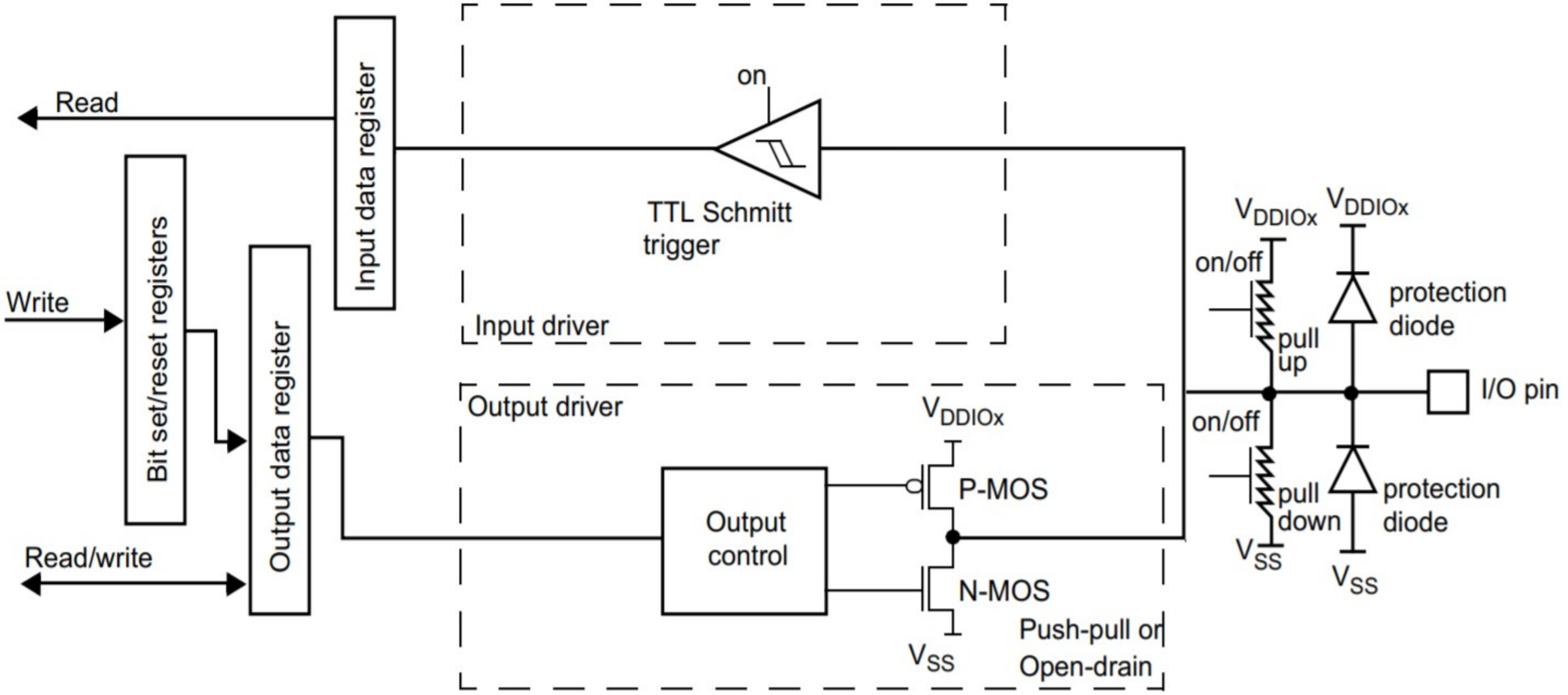
Other

- Alternative open drain function (with optional pull-up or pull-down function)
- Alternative function in push-pull mode (with optional pull-up or pull-down function)

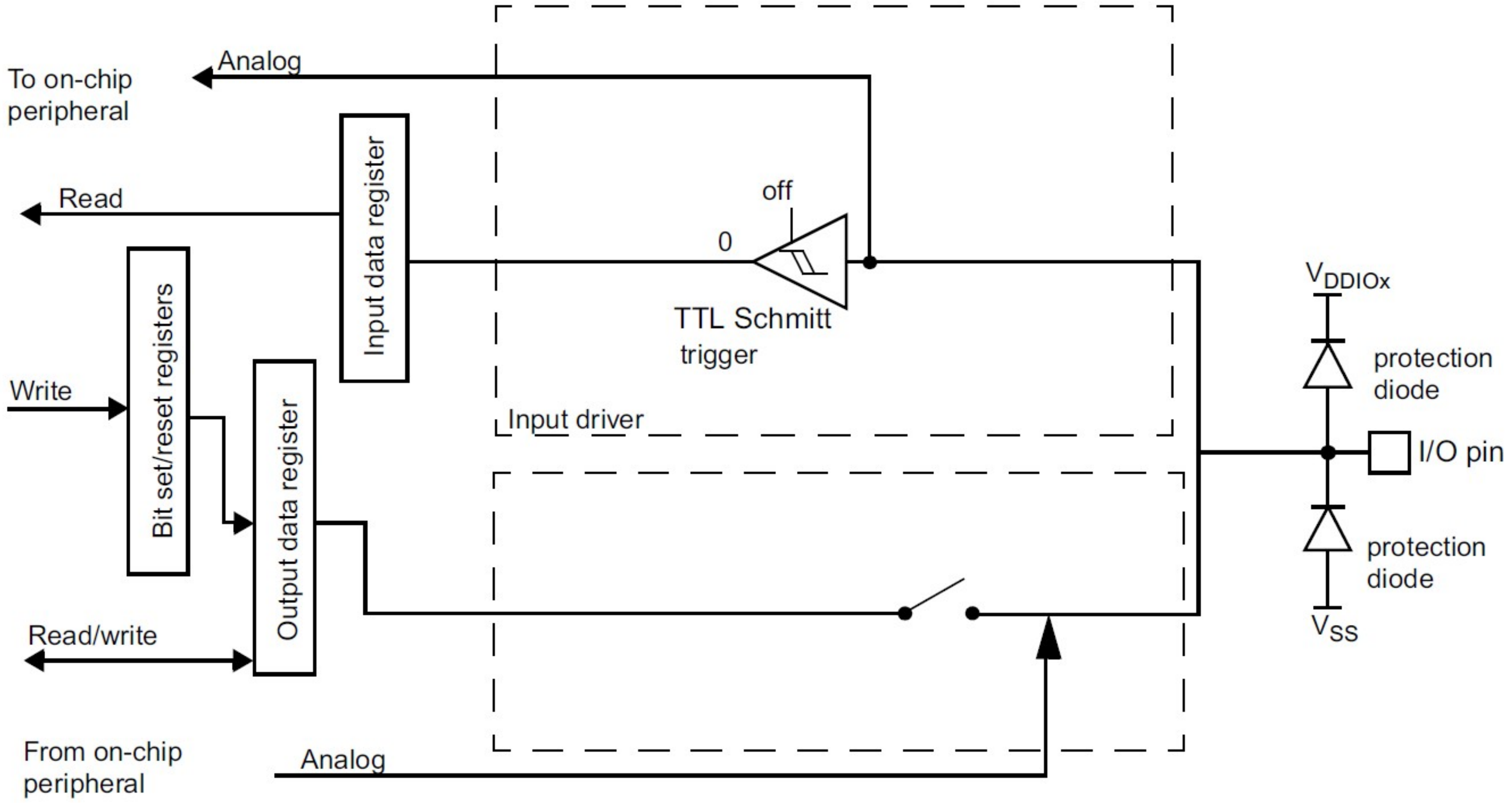
Input Port Configuration



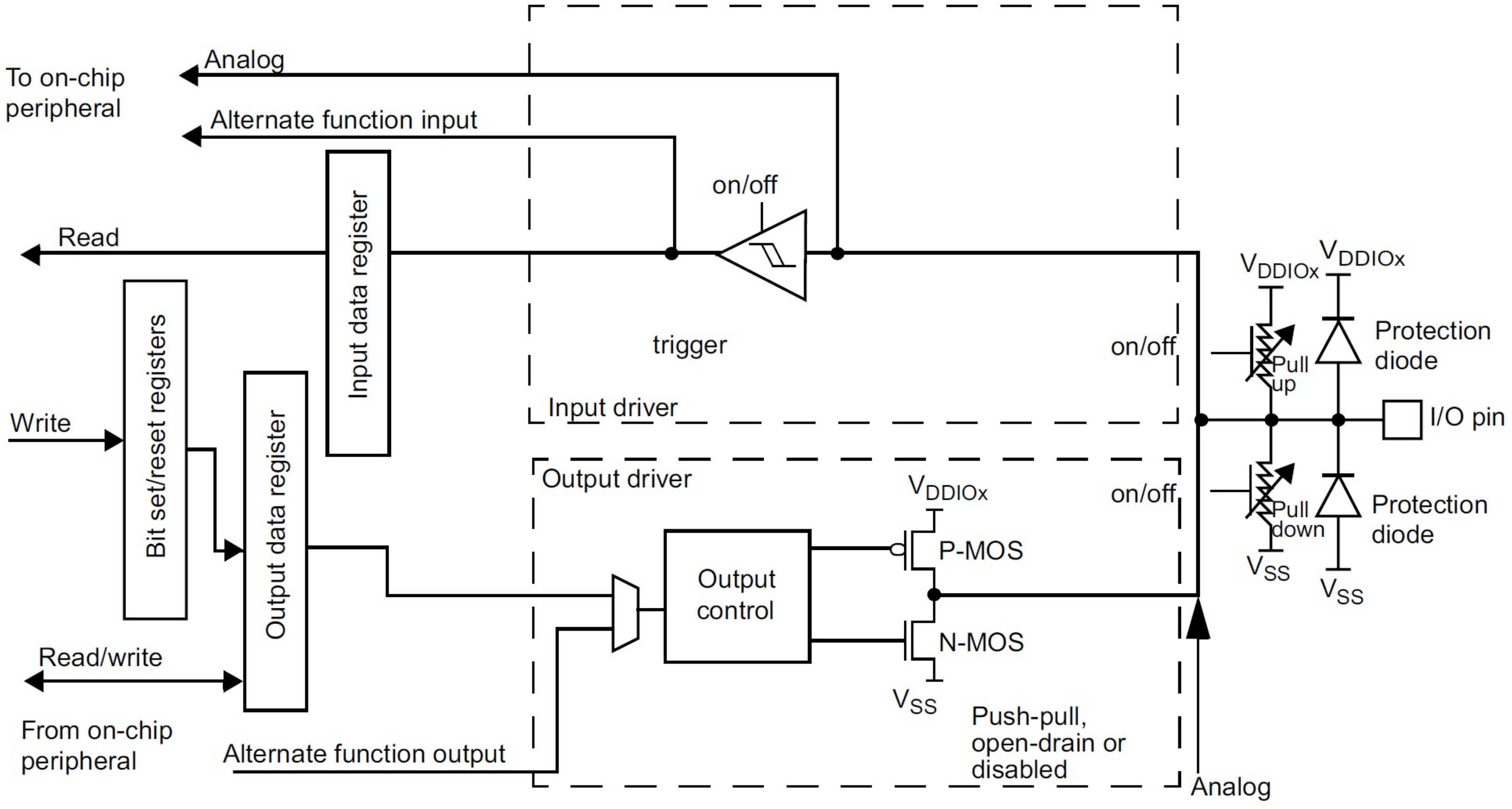
Output Port Configuration



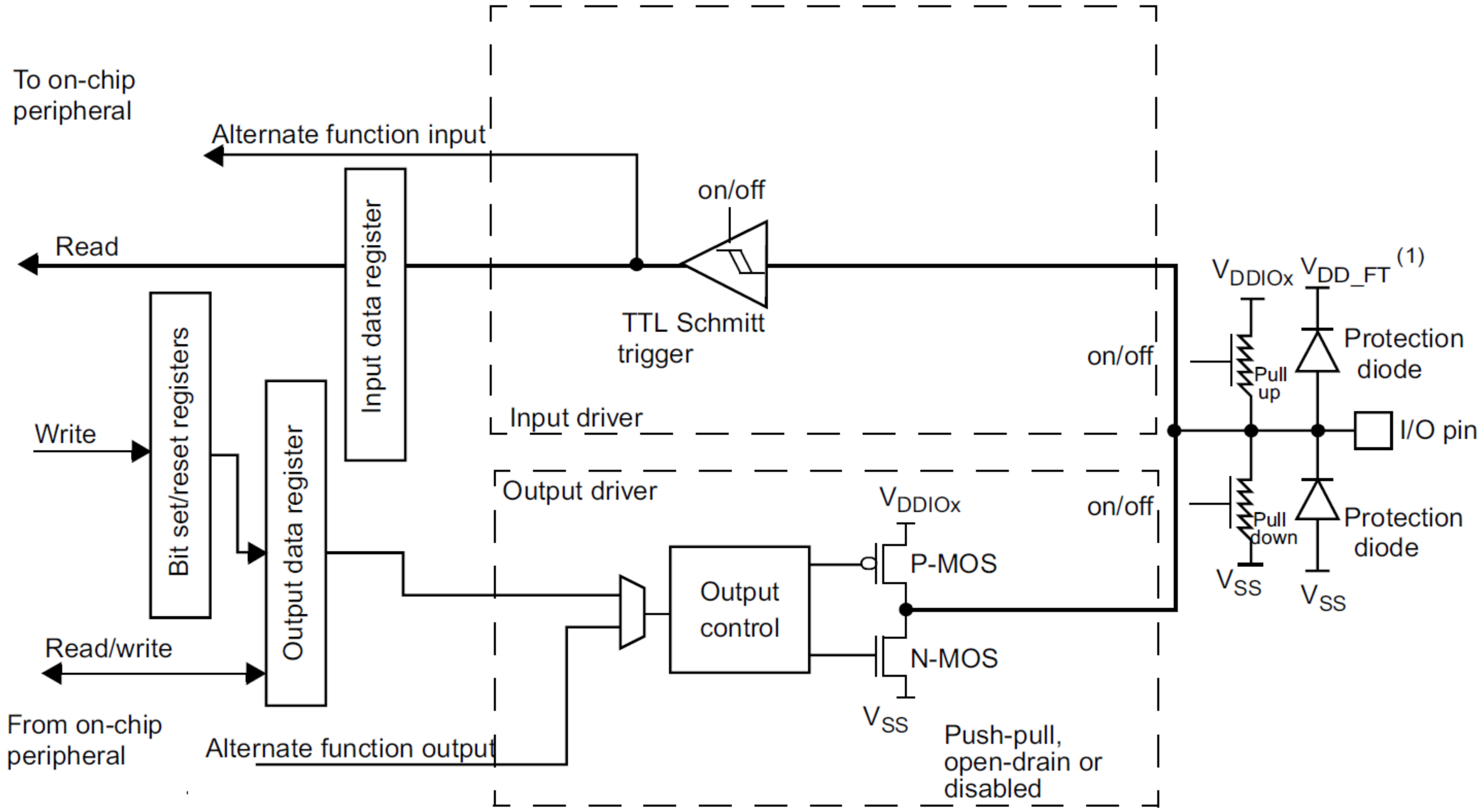
Analogue Ports



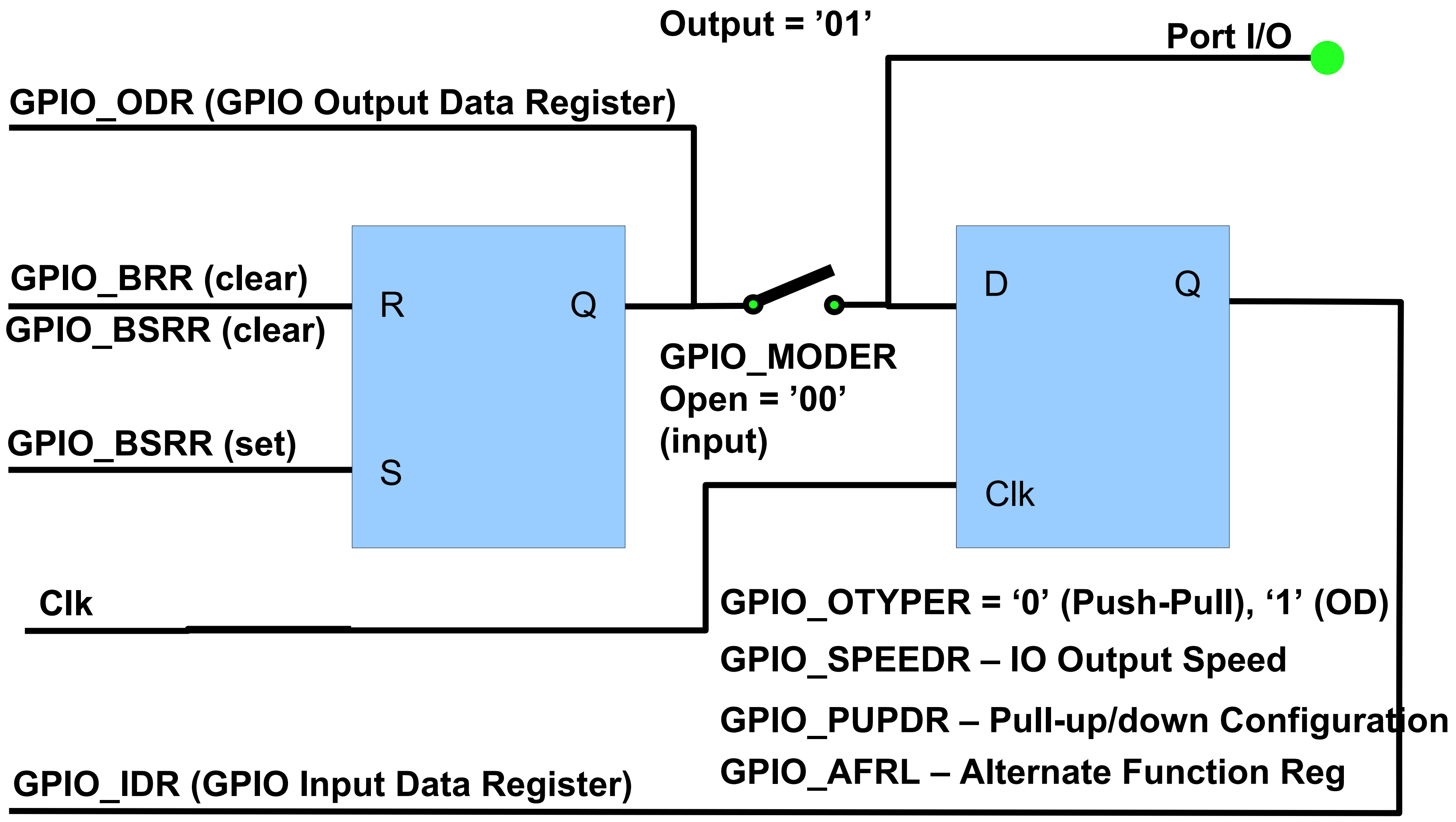
IO Circuit of ARM Microcontroller



5 V Tolerant IO Ports

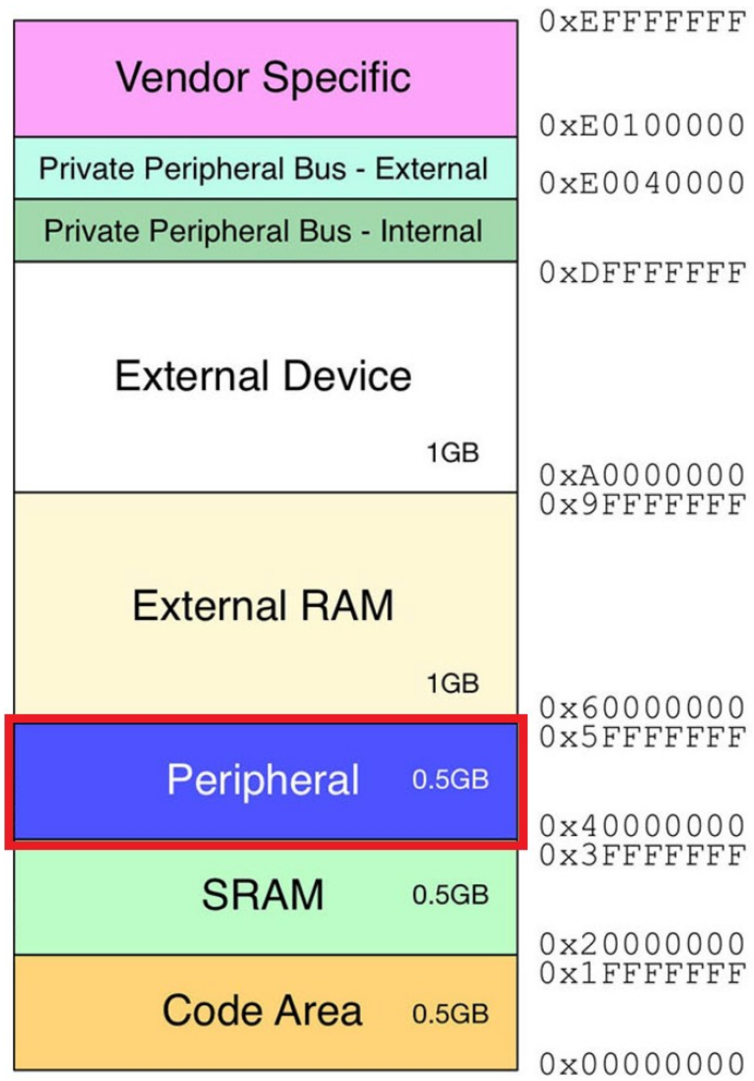


Simplified Block Diagram of I/O Port


Output = ’01’
Port I/O
GPIO_ODR (GPIO Output Data Register)
GPIO_BRR (clear)
GPIO_BSRR (clear)
R
Q
S
GPIO_BSRR (set)
GPIO_MODER
Open = ’00’
(input)
D
Q
Clk
Clk
GPIO_OTYPER = ‘0’ (Push-Pull), ‘1’ (OD)
GPIO_SPEEDR – IO Output Speed
GPIO_PUPDR – Pull-up/down Configuration
GPIO_AFRL – Alternate Function Reg
GPIO_IDR (GPIO Input Data Register)

Control Registers for IO Ports (1)

- Procesor is equipped with **8 (A-H ports) 16-bits IO ports**
- Supports up to **128 IO signals**
- Each port is controlled using **12 32-bits registers** placed in **1 kB memory**
- All registers are available in **Read/Write mode**, except:
  - **IDR – Read only**
  - **BSRR (BSR/BRR) – write only** (bitwise control)



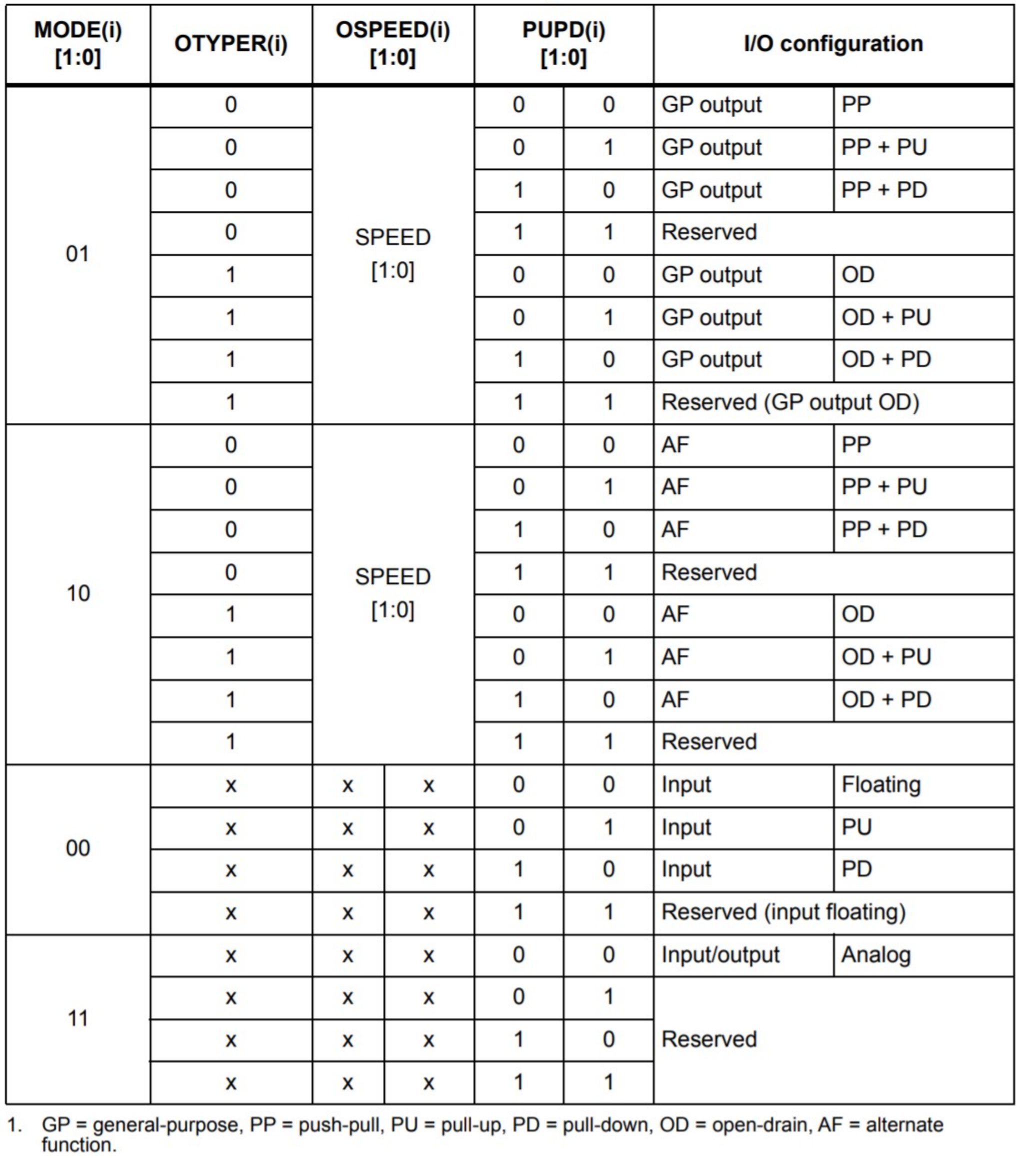
Control Registers for IO Ports (2)

Offset	Register name	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	**GPIOA_MODER**	MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]		MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
	Reset value	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x04	**GPIOx_OTYPER** (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	**GPIOA_OSPEEDR**	OSPEED15[1:0]		OSPEED14[1:0]		OSPEED13[1:0]		OSPEED12[1:0]		OSPEED11[1:0]		OSPEED10[1:0]		OSPEED9[1:0]		OSPEED8[1:0]		OSPEED7[1:0]		OSPEED6[1:0]		OSPEED5[1:0]		OSPEED4[1:0]		OSPEED3[1:0]		OSPEED2[1:0]		OSPEED1[1:0]		OSPEED0[1:0]	
	Reset value	0	0	0	0	1	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
0x0C	**GPIOA_PUPDR**	PUPD15[1:0]		PUPD14[1:0]		PUPD13[1:0]		PUPD12[1:0]		PUPD11[1:0]		PUPD10[1:0]		PUPD9[1:0]		PUPD8[1:0]		PUPD7[1:0]		PUPD6[1:0]		PUPD5[1:0]		PUPD4[1:0]		PUPD3[1:0]		PUPD2[1:0]		PUPD1[1:0]		PUPD0[1:0]	
	Reset value	0	1	1	0	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
0x10	**GPIOx_IDR** (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
	Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Control Registers for IO Ports (3)

Offset	Register name	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
0x14	**GPIOx_ODR** (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**GPIOx_BSRR** (where x = A..I)	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
0x1C	**GPIOx_LCKR** (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LCKK	LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	**GPIOx_AFRL** (where x = A..I)	AFSEL7[3:0]				AFSEL6[3:0]				AFSEL5[3:0]				AFSEL4[3:0]				AFSEL3[3:0]				AFSEL2[3:0]				AFSEL1[3:0]				AFSEL0[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
0x24	**GPIOx_AFRH** (where x = A..I)	AFSEL15[3:0]				AFSEL14[3:0]				AFSEL13[3:0]				AFSEL12[3:0]				AFSEL11[3:0]				AFSEL10[3:0]				AFSEL9[3:0]				AFSEL8[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
0x28	**GPIOx_BRR** (where x = A..I)	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
0x2C	**GPIOx_ASCR** (where x = A..H)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ASC15	ASC14	ASC13	ASC12	ASC11	ASC10	ASC9	ASC8	ASC7	ASC6	ASC5	ASC4	ASC3	ASC2	ASC1	ASC0
	Reset value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Control Registers for IO Ports - Summary



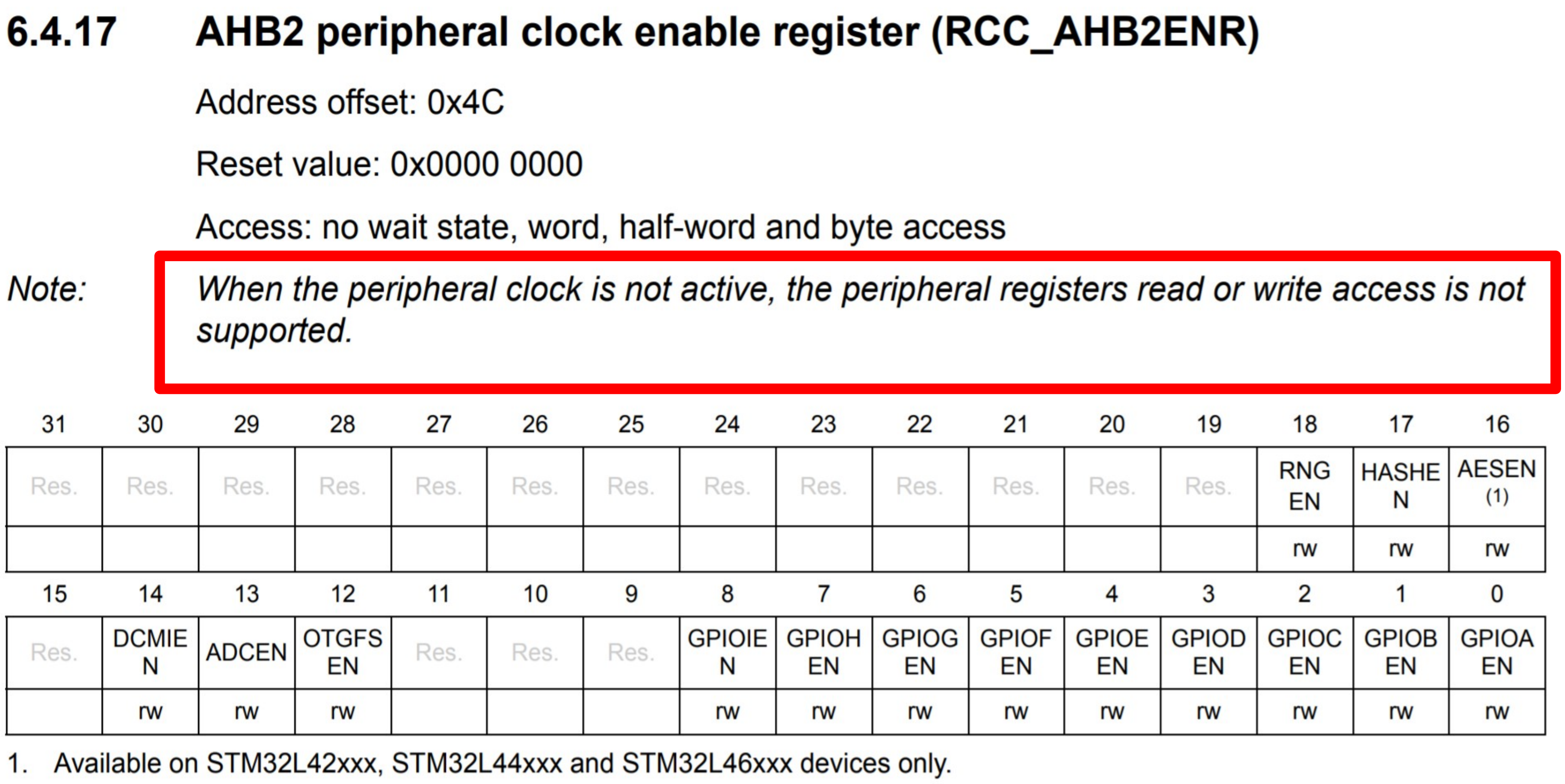
MODE(i) [1:0]	OTYPER(i)	OSPEED(i) [1:0]		PUPD(i) [1:0]		I/O configuration	
01	0	SPEED [1:0]		0	0	GP output	PP
	0			0	1	GP output	PP + PU
	0			1	0	GP output	PP + PD
	0			1	1	Reserved	
	1			0	0	GP output	OD
	1			0	1	GP output	OD + PU
	1			1	0	GP output	OD + PD
	1			1	1	Reserved (GP output OD)	
10	0	SPEED [1:0]		0	0	AF	PP
	0			0	1	AF	PP + PU
	0			1	0	AF	PP + PD
	0			1	1	Reserved	
	1			0	0	AF	OD
	1			0	1	AF	OD + PU
	1			1	0	AF	OD + PD
	1			1	1	Reserved	
00	x	x	x	0	0	Input	Floating
	x	x	x	0	1	Input	PU
	x	x	x	1	0	Input	PD
	x	x	x	1	1	Reserved (input floating)	
11	x	x	x	0	0	Input/output	Analog
	x	x	x	0	1	Reserved	
	x	x	x	1	0		
	x	x	x	1	1		

1. GP = general-purpose, PP = push-pull, PU = pull-up, PD = pull-down, OD = open-drain, AF = alternate function.

Base Address for A-H Ports

Bus	Boundary address	Size (bytes)	Peripheral	Peripheral register map
AHB2	0x4800 1C00 - 0x4800 1FFF	1 KB	GPIOH	*Section 8.4.13: GPIO register map*
	0x4800 1800 - 0x4800 1BFF	1 KB	GPIOG	*Section 8.4.13: GPIO register map*
	0x4800 1400 - 0x4800 17FF	1 KB	GPIOF	*Section 8.4.13: GPIO register map*
	0x4800 1000 - 0x4800 13FF	1 KB	GPIOE	*Section 8.4.13: GPIO register map*
	0x4800 0C00 - 0x4800 0FFF	1 KB	GPIOD	*Section 8.4.13: GPIO register map*
	0x4800 0800 - 0x4800 0BFF	1 KB	GPIOC	*Section 8.4.13: GPIO register map*
	0x4800 0400 - 0x4800 07FF	1 KB	GPIOB	*Section 8.4.13: GPIO register map*
	0x4800 0000 - 0x4800 03FF	1 KB	GPIOA	*Section 8.4.13: GPIO register map*
	0x4002 4400 - 0x47FF FFFF	~127 MB	Reserved	-

Clocks for GPIO Peripheral Devices



6.4.17 AHB2 peripheral clock enable register (RCC_AHB2ENR)

Address offset: 0x4C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

Note: *When the peripheral clock is not active, the peripheral registers read or write access is not supported.*

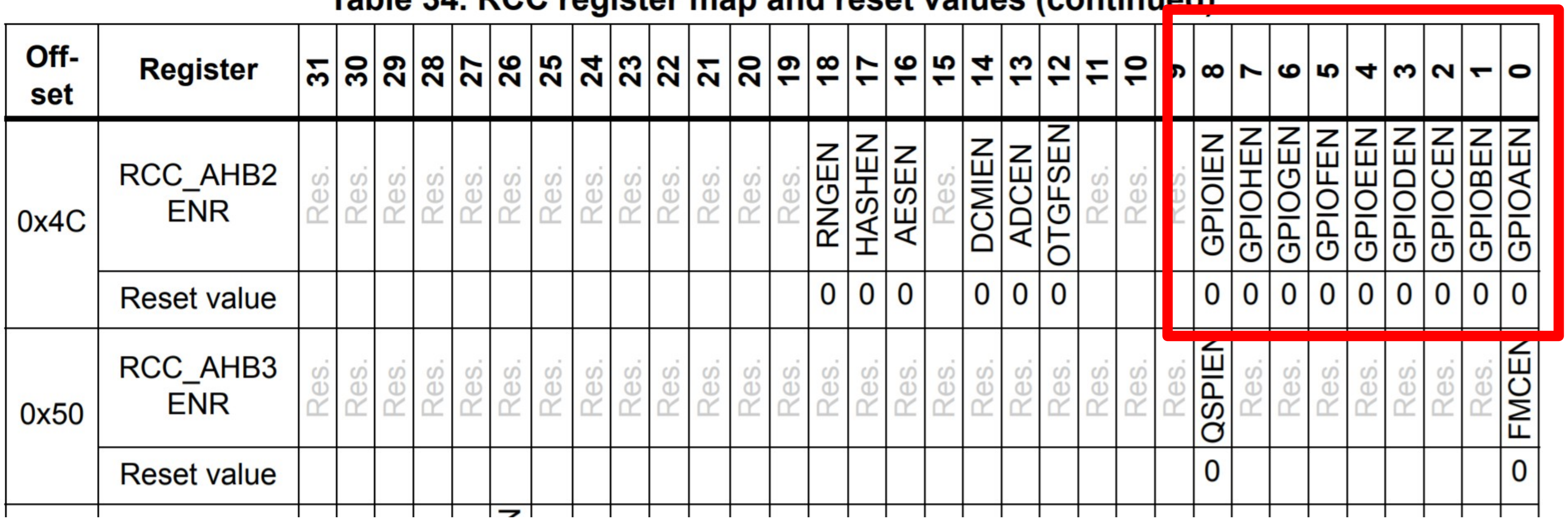
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNGEN	HASHEN	AESEN (1)
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DCMIEN	ADCEN	OTGFSEN	Res.	Res.	Res.	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
	rw	rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

1. Available on STM32L42xxx, STM32L44xxx and STM32L46xxx devices only.

Clocks for GPIO Peripheral Devices #2

Table 34. RCC register map and reset values (continued)



Off-set	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
0x4C	RCC_AHB2 ENR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNGEN	HASHEN	AESEN	Res.	DCMIEN	ADCEN	OTGFSEN	Res.	Res.	Res.	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
	Reset value														0	0	0		0	0	0				0	0	0	0	0	0	0	0	0
0x50	RCC_AHB3 ENR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	QSPIEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FMCEN
	Reset value																								0								0

Power Control (1)

5.4.2 Power control register 2 (PWR_CR2)

Address offset: 0x04

Reset value: 0x0000 0000. This register is reset when exiting the Standby mode.

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
Res.	Res.	Res.	Res.	Res.	USV	IOSV	Res.	PVME4	PVME3	PVME2	PVME1	PLS[2:0]			PVDE
					rw	rw		rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:11 Reserved, must be kept at reset value.

Bit 10 **USV**: V_{DDUSB} USB supply valid

This bit is used to validate the V_{DDUSB} supply for electrical and logical isolation purpose. Setting this bit is mandatory to use the USB OTG_FS peripheral. If V_{DDUSB} is not always present in the application, the PVM can be used to determine whether this supply is ready or not.

0: V_{DDUSB} is not present. Logical and electrical isolation is applied to ignore this supply.

1: V_{DDUSB} is valid.

Bit 9 **IOSV**: V_{DDIO2} Independent I/Os supply valid

This bit is used to validate the V_{DDIO2} supply for electrical and logical isolation purpose. Setting this bit is mandatory to use PG[15:2]. If V_{DDIO2} is not always present in the application, the PVM can be used to determine whether this supply is ready or not.

0: V_{DDIO2} is not present. Logical and electrical isolation is applied to ignore this supply.

1: V_{DDIO2} is valid.

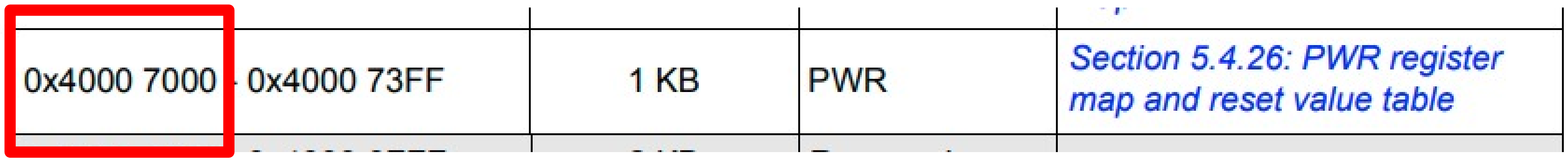
Power Control (2)

During first laboratory use the HAL function:

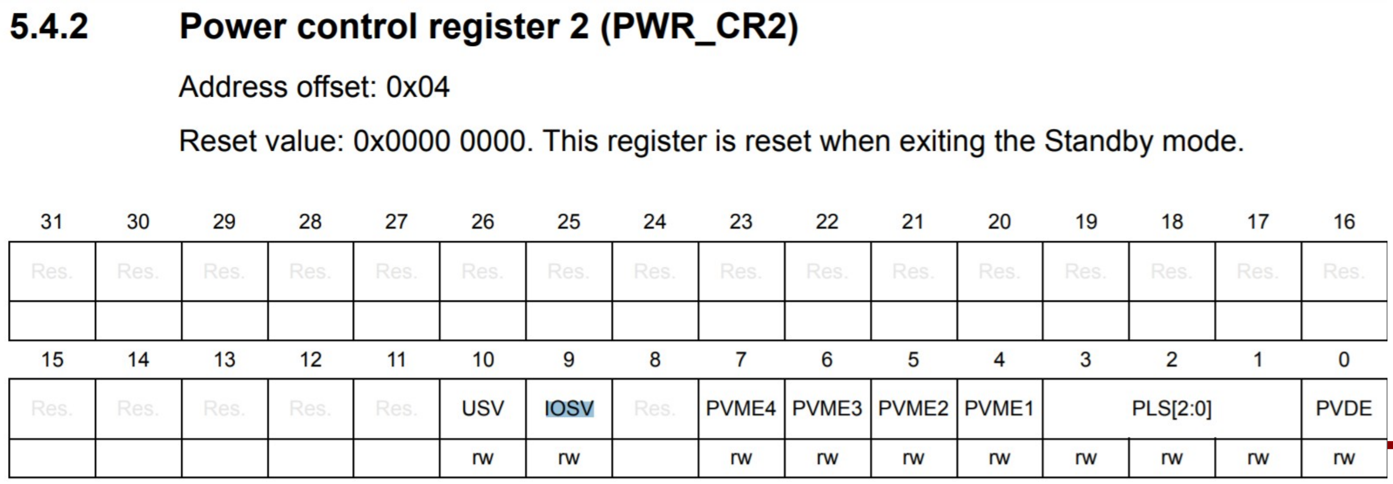
- __STATIC_INLINE void LL_PWR_EnableVddIO2 (void)

Later you will write your own function to do this

- You need: PWR – Base Address



0x4000 7000 - 0x4000 73FF	1 KB	PWR	*Section 5.4.26: PWR register map and reset value table*



5.4.2 Power control register 2 (PWR_CR2)

Address offset: 0x04

Reset value: 0x0000 0000. This register is reset when exiting the Standby mode.

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
Res.	Res.	Res.	Res.	Res.	USV	IOSV	Res.	PVME4	PVME3	PVME2	PVME1	PLS[2:0]			PVDE
					rw	rw		rw	rw	rw	rw	rw	rw	rw	rw

Main Power Control

6.4.19 APB1 peripheral clock enable register 1 (RCC_APB1ENR1)

Address: 0x58

Reset value: 0x0000 0400 (for STM32L496xx/4A6xx devices)
0x0000 0000 (for STM32L475xx/476xx/486xx devices)

Access: no wait state, word, half-word and byte access

Note: When the peripheral clock is not active, the peripheral registers read or write access is not supported.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LPTIM1 EN	OPAMP EN	DAC1 EN	PWR EN	Res.	CAN2 EN	CAN1 EN	CRSEN	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN[1]	USART3 EN	USART2 EN	Res.
rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

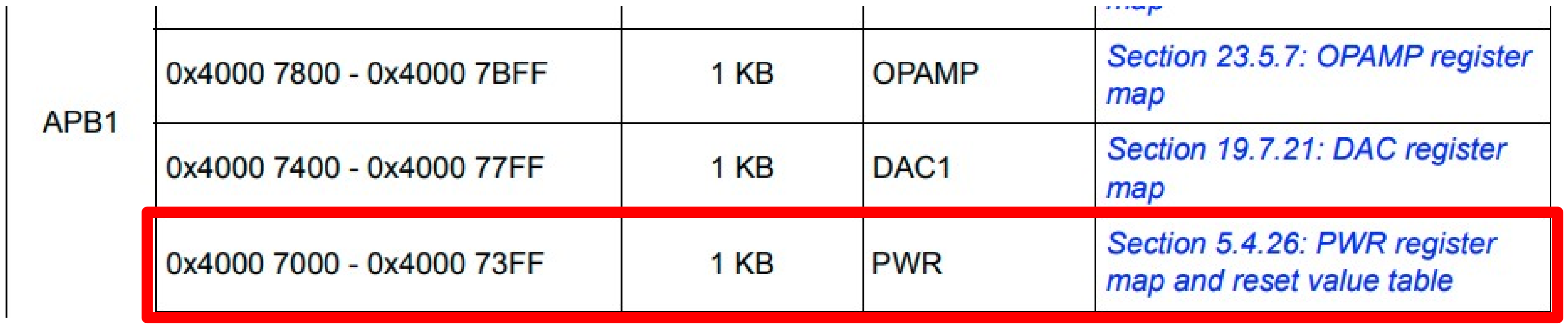
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Res.	Res.	WWD GEN	RTCA PBEN	LCD EN	Res.	Res.	Res.	TIM7 EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2 EN
rw	rw			rs	rw	rw				rw	rw	rw	rw	rw	rw

1. Available on STM32L45xxx and STM32L46xxx devices only.

Bit 28 **PWREN**: Power interface clock enable
Set and cleared by software.
0: Power interface clock disabled
1: Power interface clock enabled

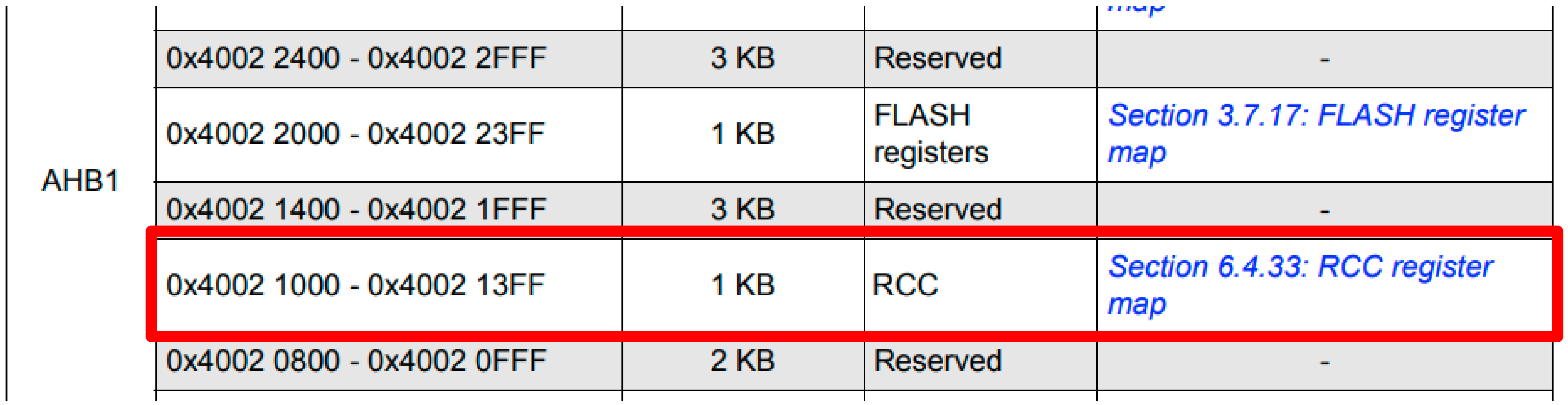
Base Addresses

- PWR – Base Address



APB1	0x4000 7800 - 0x4000 7BFF	1 KB	OPAMP	*Section 23.5.7: OPAMP register map*
	0x4000 7400 - 0x4000 77FF	1 KB	DAC1	*Section 19.7.21: DAC register map*
	0x4000 7000 - 0x4000 73FF	1 KB	PWR	*Section 5.4.26: PWR register map and reset value table*

- RCC – Base Address



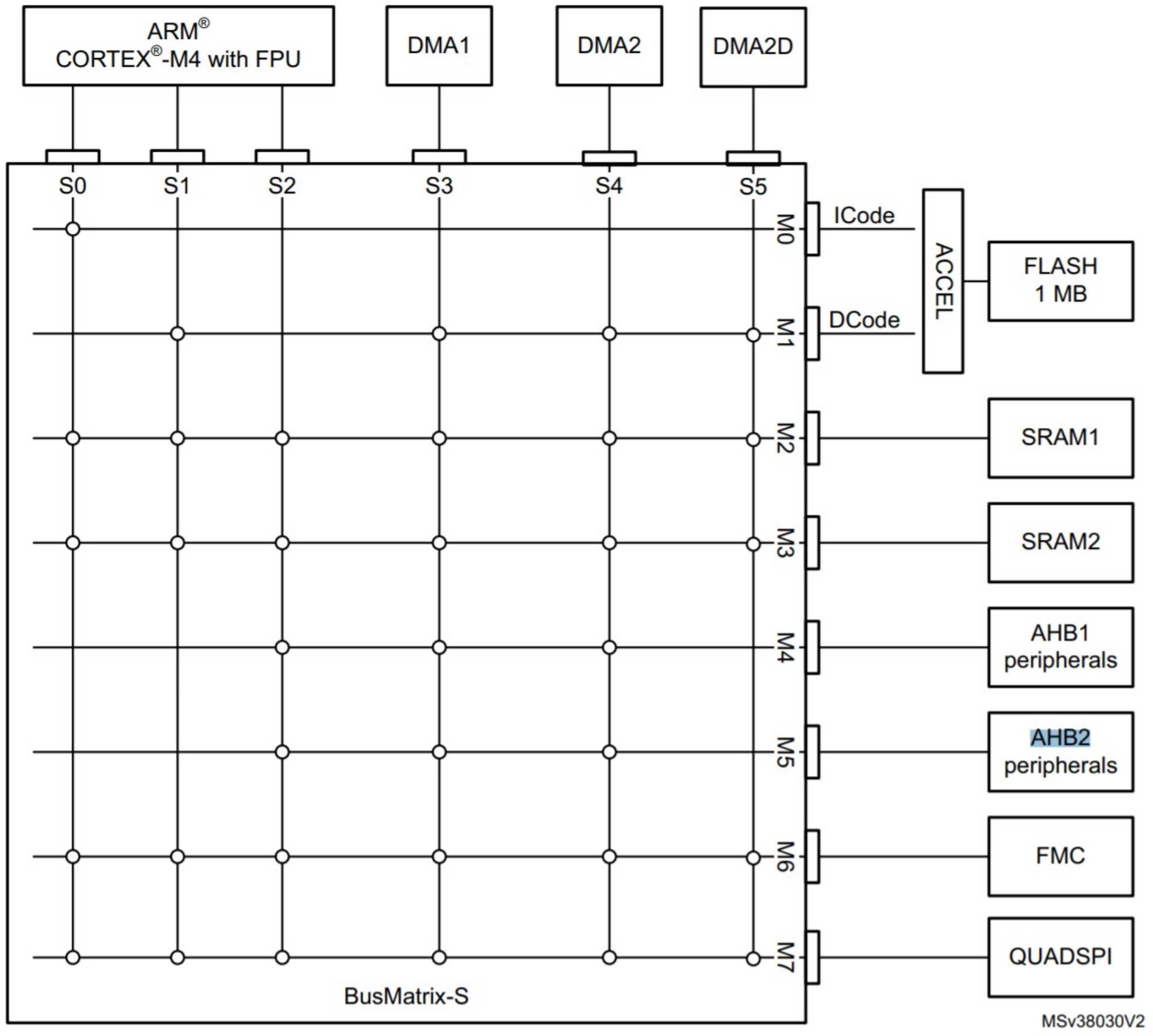
AHB1	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved	-
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH registers	*Section 3.7.17: FLASH register map*
	0x4002 1400 - 0x4002 1FFF	3 KB	Reserved	-
	0x4002 1000 - 0x4002 13FF	1 KB	RCC	*Section 6.4.33: RCC register map*
	0x4002 0800 - 0x4002 0FFF	2 KB	Reserved	-

Microcontroller Architecture #1

The main system consists of 32-bit multilayer AHB bus matrix that interconnects:

- **Up to six masters:**
  - – Cortex®-M4 with FPU core I-bus
  - – Cortex®-M4 with FPU core D-bus
  - – Cortex®-M4 with FPU core S-bus
  - – DMA1
  - – DMA2
  - – DMA2D (only for STM32L496xx/4A6xx devices)
- **Up to eight slaves:**
  - – Internal Flash memory on the ICode bus
  - – Internal Flash memory on DCode bus
  - – Internal SRAM1 (96 KB for STM32L475xx/476xx/486xx devices, 256 KB for
  - STM32L496xx/4A6xx devices)
  - – Internal SRAM2 (32 KB for STM32L475xx/476xx/486xx devices, 64 KB for
  - STM32L496xx/4A6xx devices)
  - – AHB1 peripherals including AHB to APB bridges and APB peripherals (connected
  - to APB1 and APB2)
  - – **AHB2 peripherals**
  - – Flexible Memory Controller (FMC)
  - – Quad SPI memory interface (QUADSPI)

Microcontroller Architecture #1



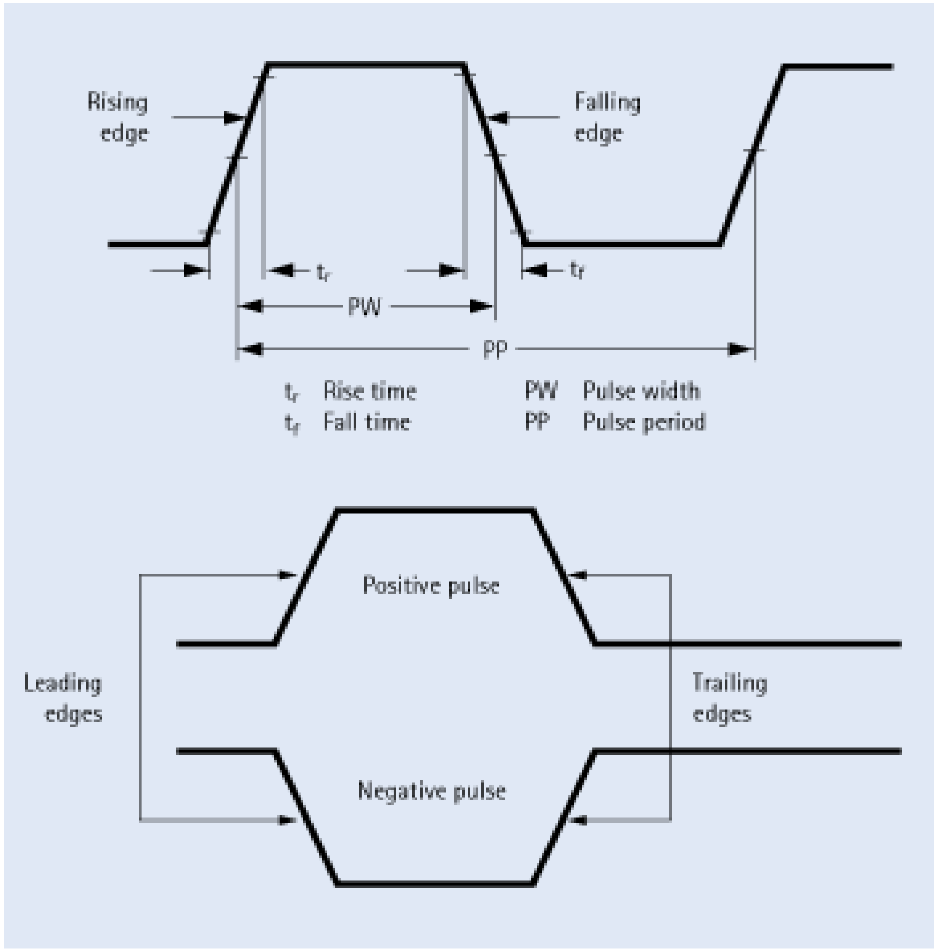
Digital Signal

Digital Signal can be characterised with:

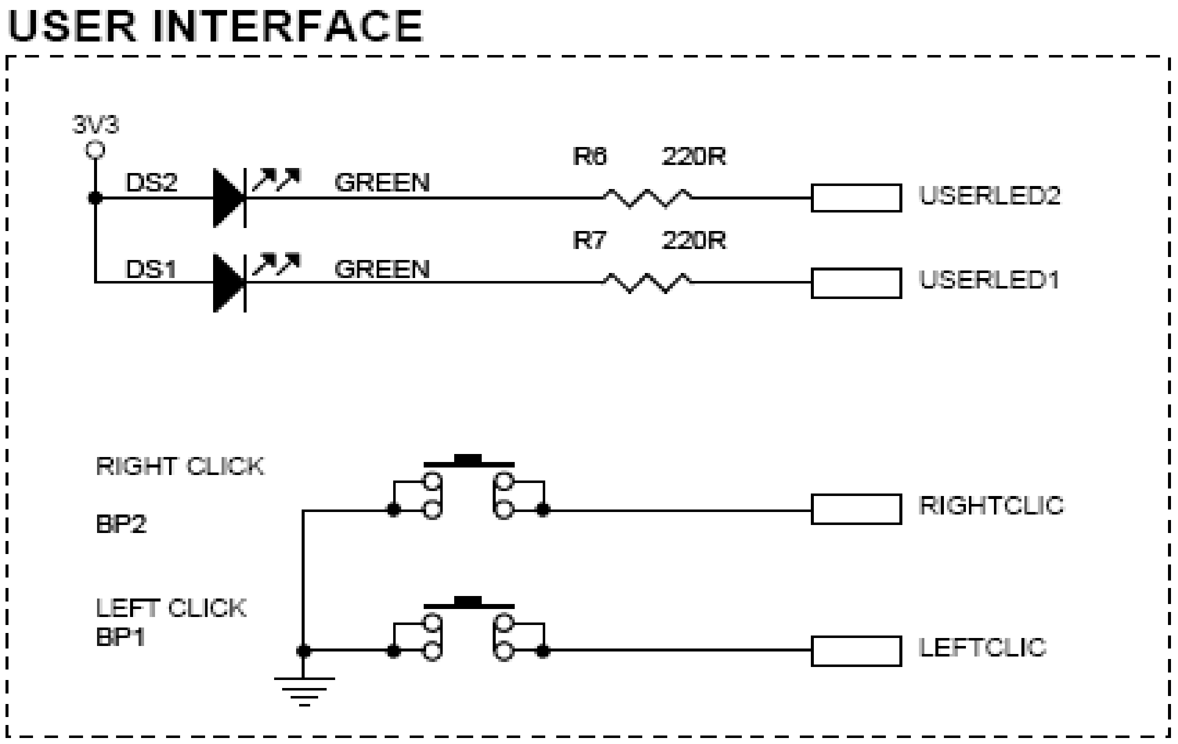
- f – frequency (period),
- A – amplitude.

Digital circuits can be triggered with:

- Change of signal level (lower or higher than signal threshold level),
- Change of signal slope (transaction of digital signal from '0' to '1' or from '1' to '0').



LEDs, Buttons



```
#define AT91B_LED1        AT91C_PIO_PB8   /* DS1 */
#define AT91B_LED2        AT91C_PIO_PC29  /* DS2 */
#define AT91B_BP1         AT91C_PIO_PC5  // Left click
#define AT91B_BP2         AT91C_PIO_PC4  // Right clic
```