Dariusz Makowski

Department of Microelectronics and Computer Science

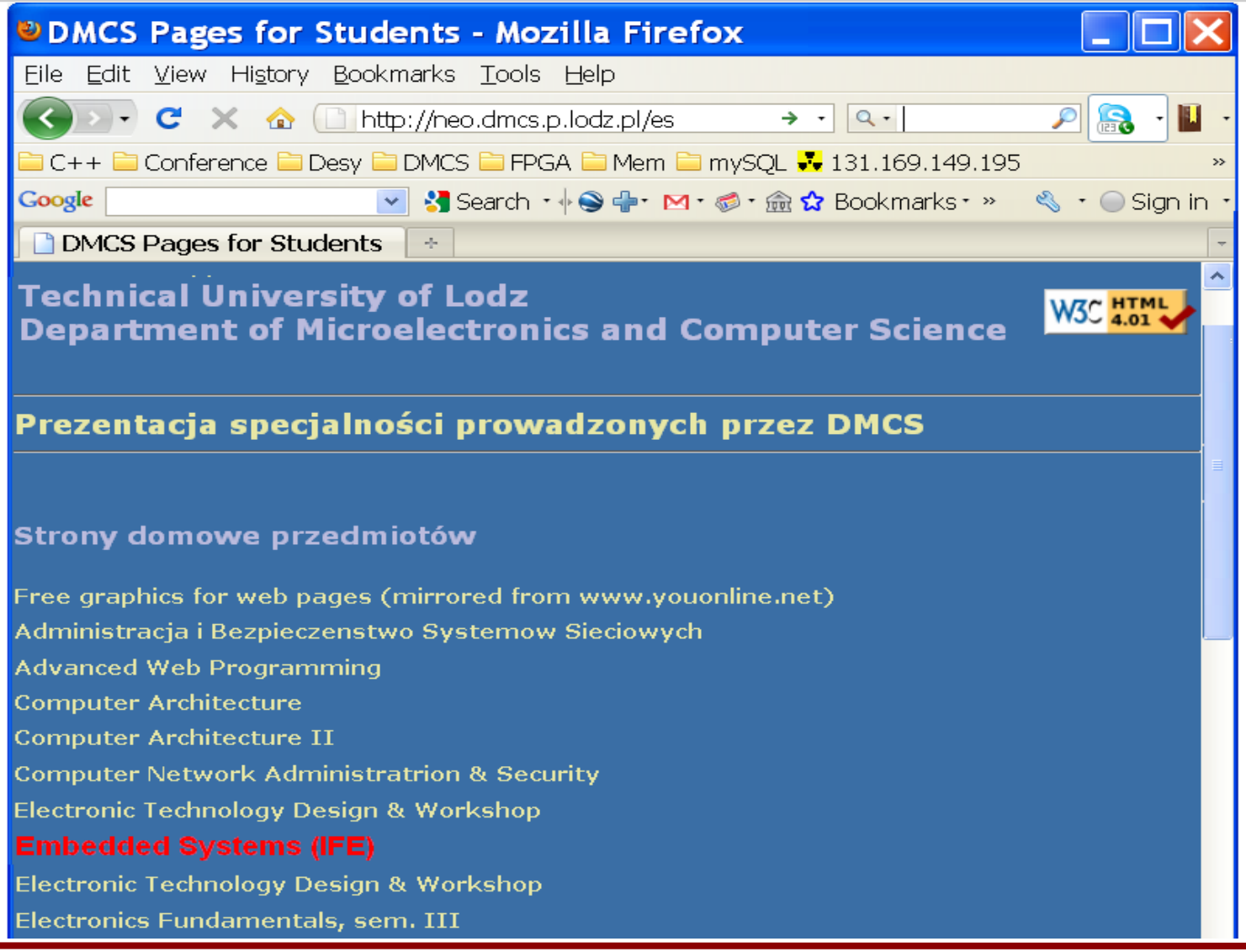
tel. 631 2720

dmakow@dmcs.pl

http://fiona.dmcs.pl/es

- Introduction
- Exam
- Reading materials
- Laboratory

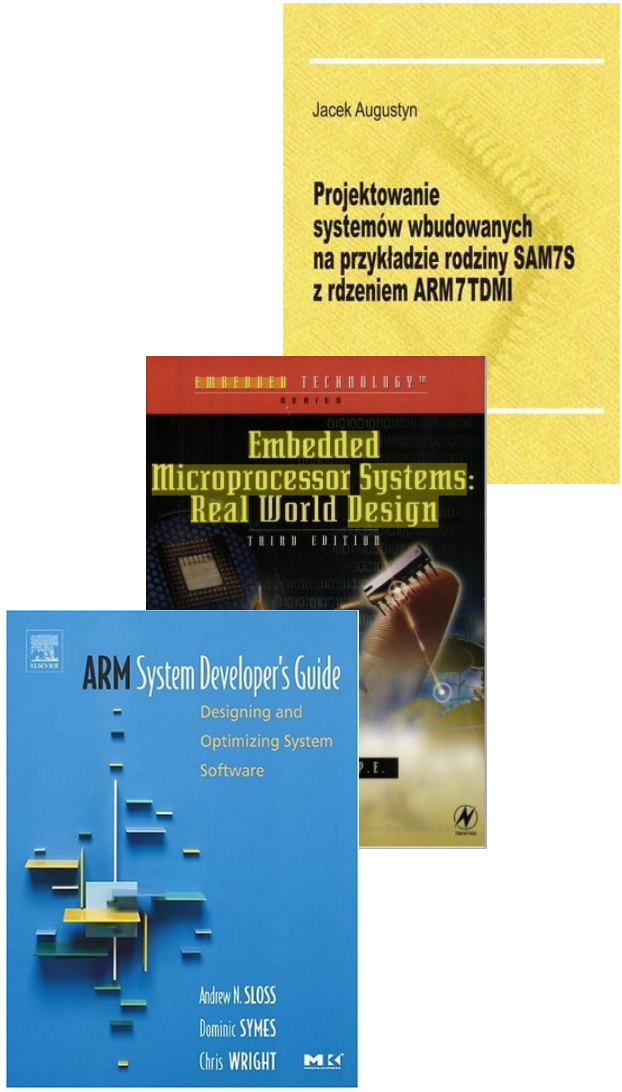
Web page for Embedded Systems



Literature

Reading Materials:

- Lecture and laboratory materials
- A. Sloss, D. Symes, C. Wright, „ARM System Developer's Guide: Designing and Optimizing System Software”, Elsevier, 2004
- S. R. Ball, “Embedded Microprocessor Systems: Real World Design”, Elsevier Science, 2002
- J. Augustyn, “Projektowanie systemów wbudowanych na przykładzie rodziny SAM7S z rdzeniem ARM7TDMI”, IGSMiE PAN, 2007, ISBN: 978-83-60195-55-0



Laboratory

Address:

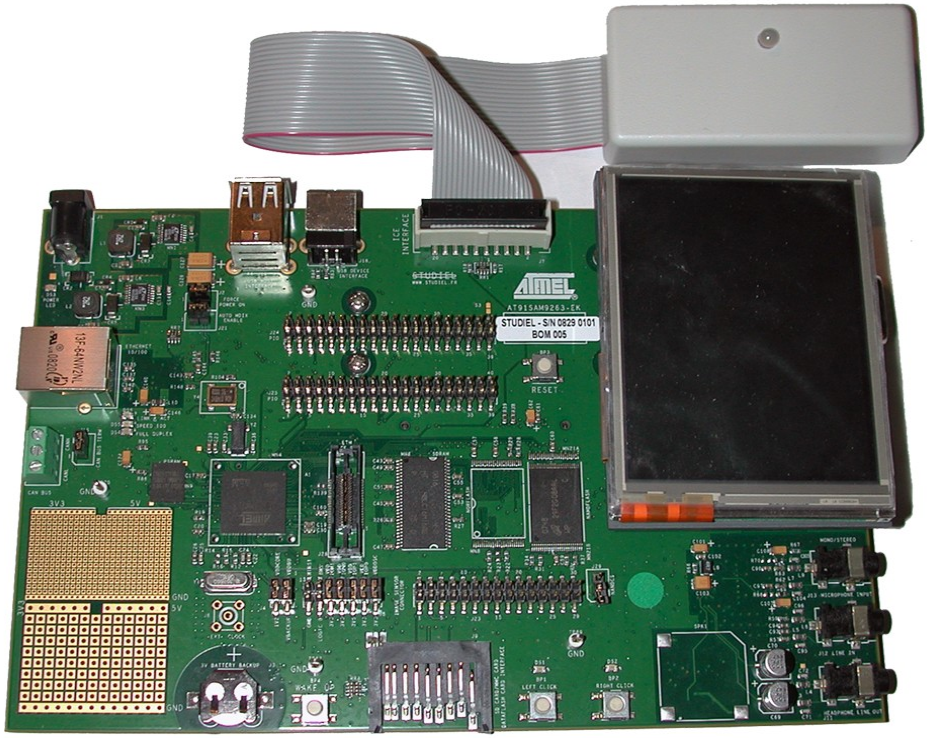
- Building B18, 1st floor – laboratory M

Practical exercises with application of ARM processor:

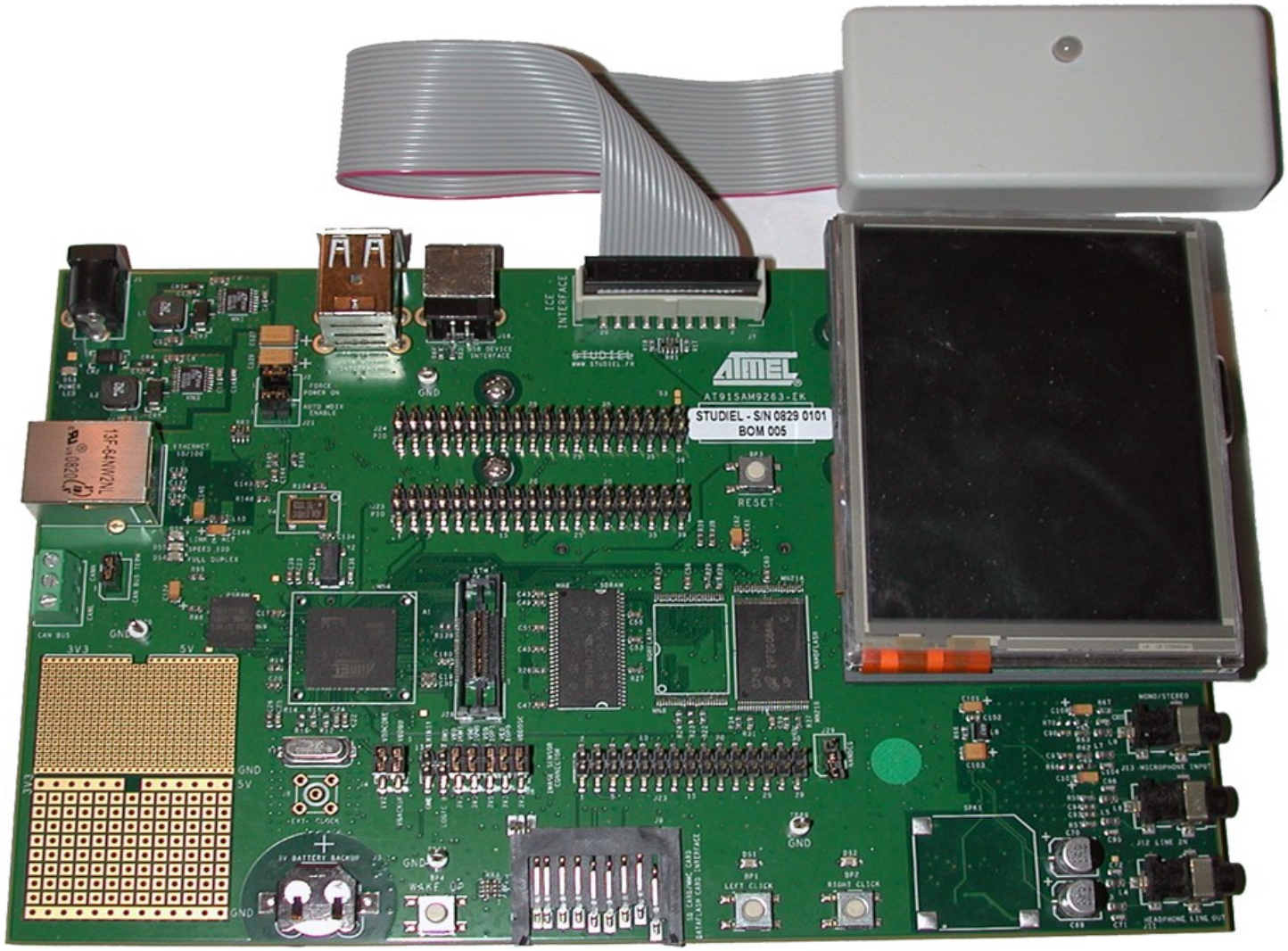
- Linux operating system
- GNU tools
- AT91SAM9263-STARTUP-PAKET
- Extension board with peripheral devices
- RTEMS real-time system for embedded devices

MSC starter-kit (1)

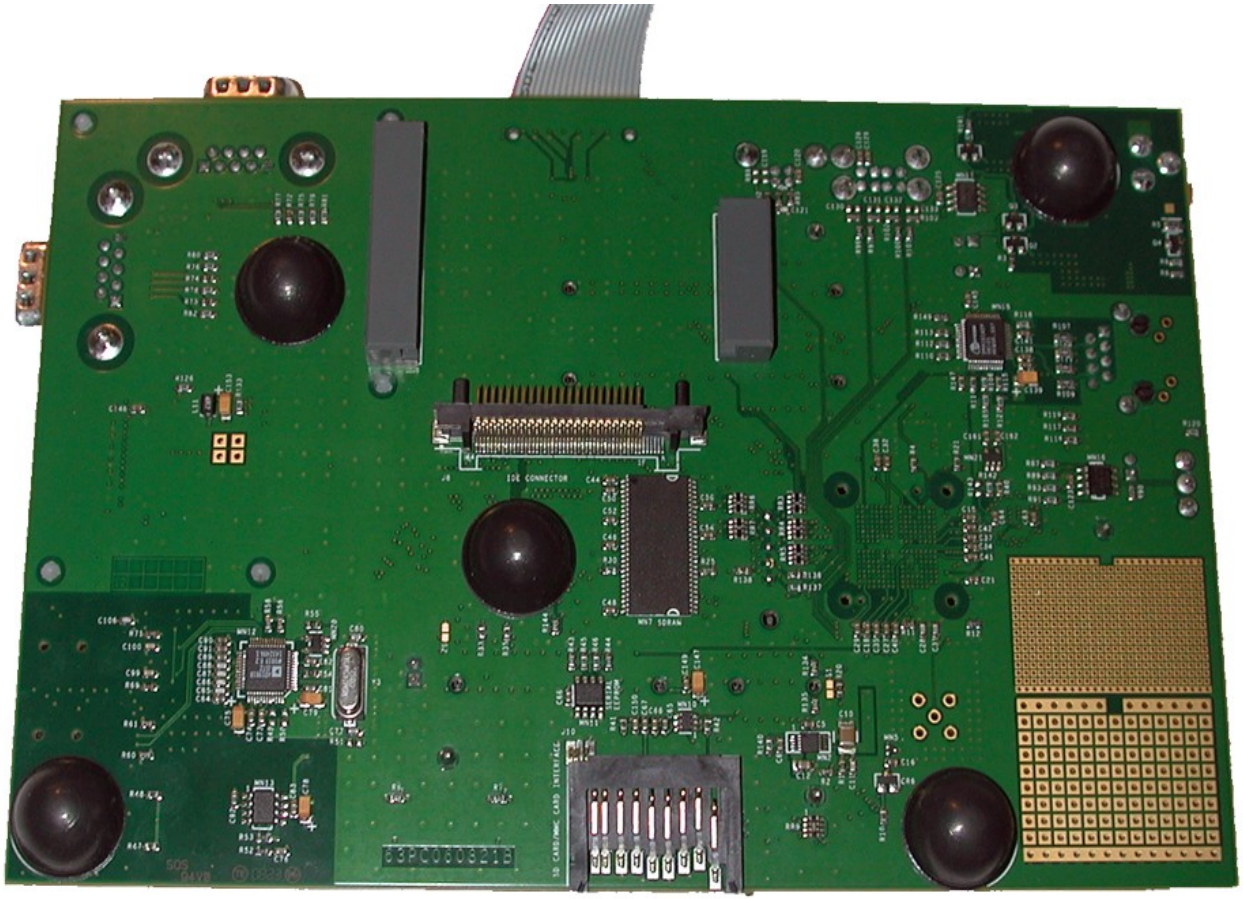
- **ARM processor with ARM9TDMI core made by ATMEL:** AT91SAM9263
- **Memory:** 64 MB SDRAM, 256 MB NAND FLASH, 4 MB DataFlash, FlashCard slots
- **Interfaces:** Ethernet 100-base TX, USB FS device, 2 x USB FS Host, CAN 2.0B, EIA RS232
- **Display:** 3.5" 1/4 VGA TFT LCD with touch screen
- **Audio codec:** AC97 Audio DAC
- **Debug interface:** JTAG
- **Programming interface:** JTAG, Free Atmel SAM-BA tools
- **Extensions:** SD/SDIO/MMC card slot
- **Starter-kit code:** AT91SAM9263-STARTUP-PAKET



MSC starter-kit (2)



MSC starter-kit (3)



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

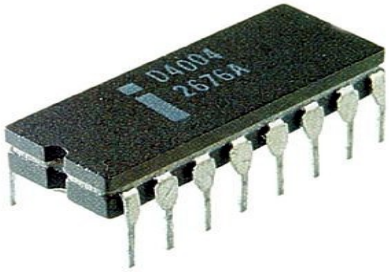
Lecture Agenda

- 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

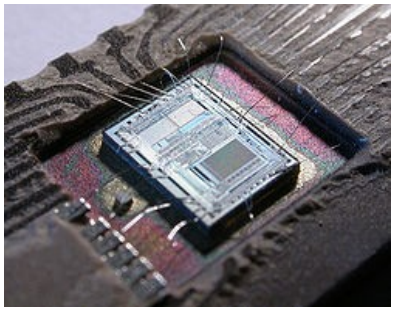
- **Processor (Central Processing Unit)**

  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



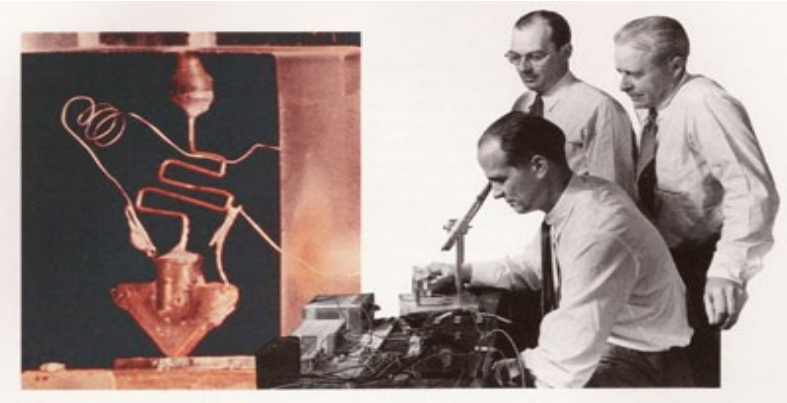
- **Microcontroller**

  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.

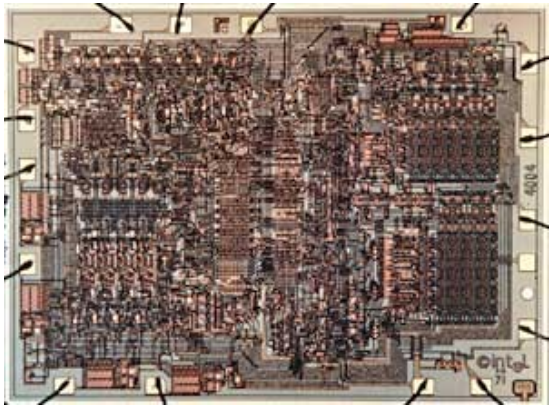
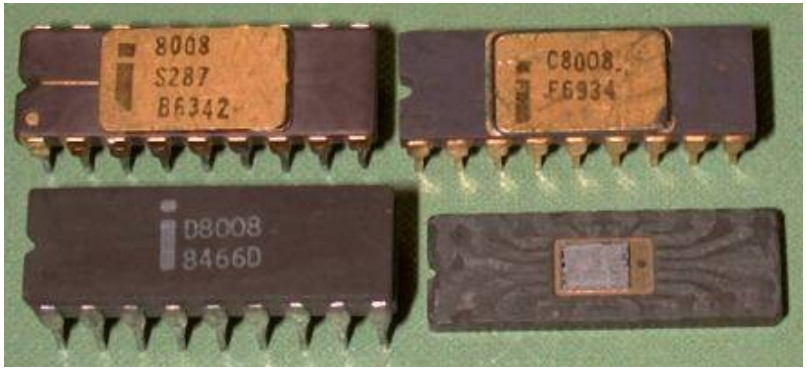


Photo of 4 bits INTEL 4004 processor



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).

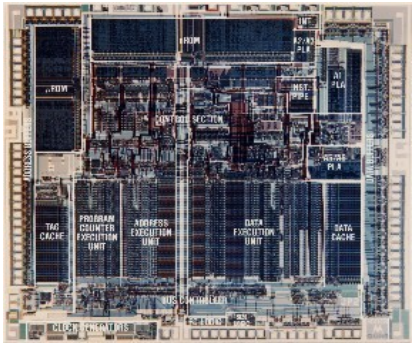
1975 – new 8-bits processor from INTEL 6502 (MOS technology) – the cheapest microprocessor on the world that time.

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

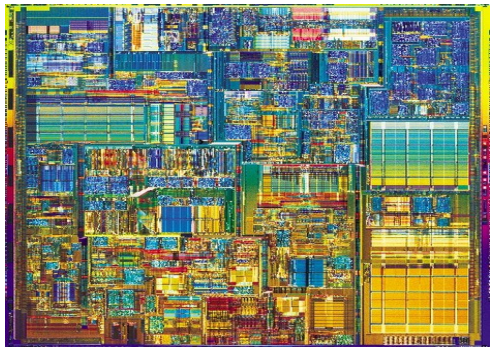
1979 – Motorola also offers 16-bits processor, 68000 family.

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

.....



Motorola 68020

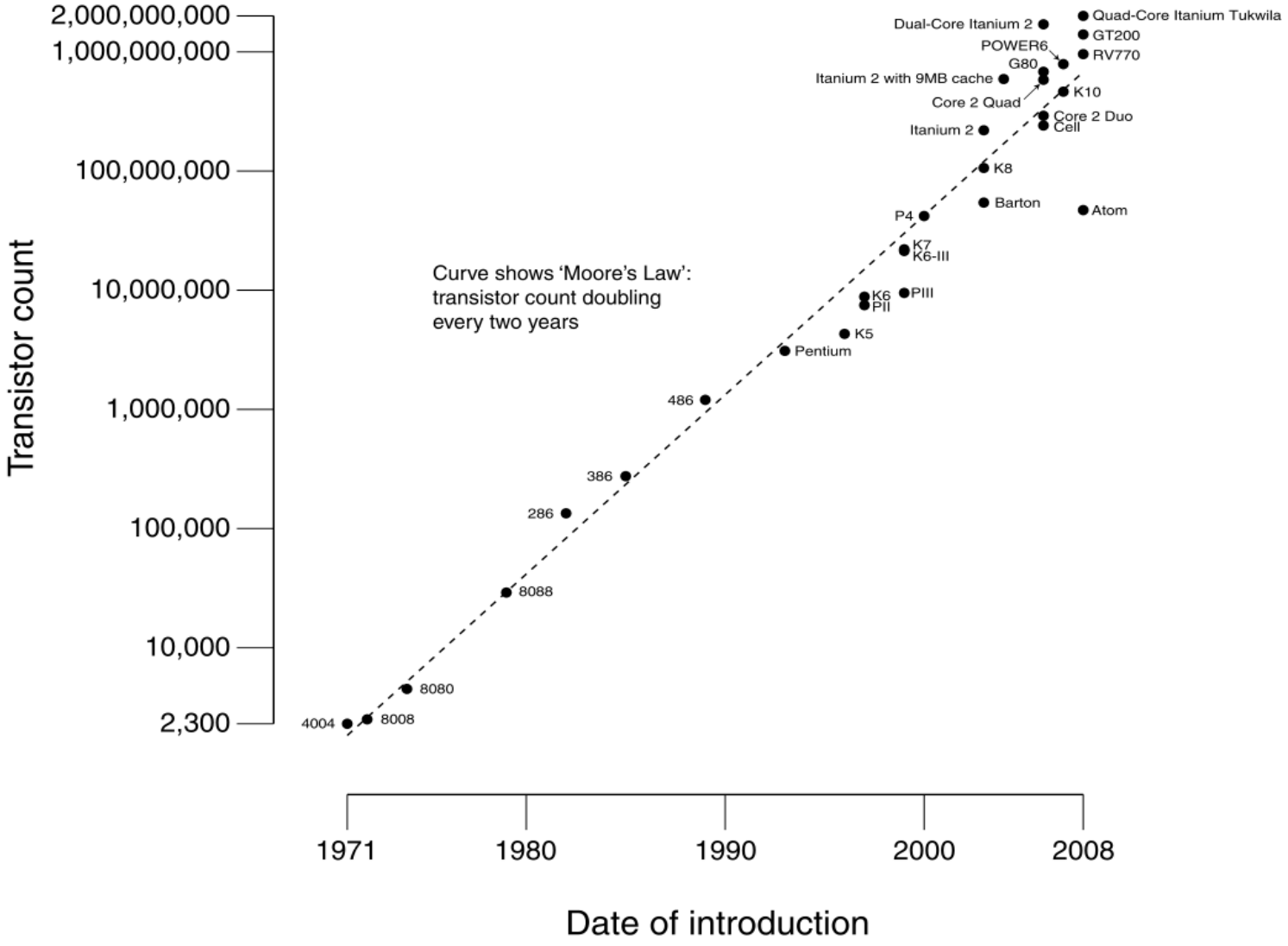


Intel, Pentium 4 Northwood

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)**

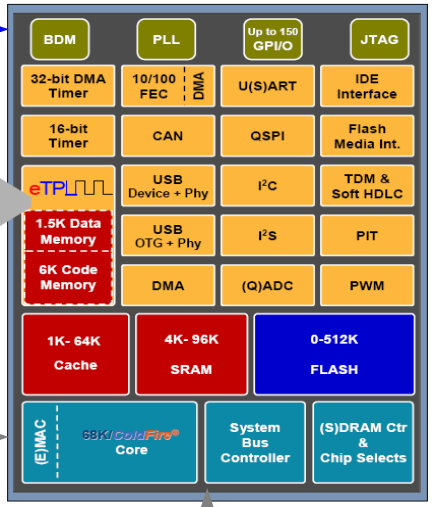
Computers and computer systems dedicated for personal usage at home, office or work. The general-purpose computers are equipped with operating system responsible for processing user applications.

- **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.

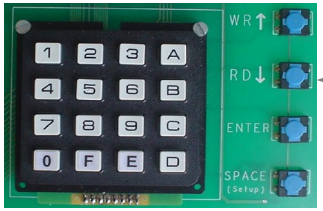
Embedded Computer

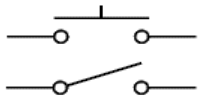
Communication channel

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



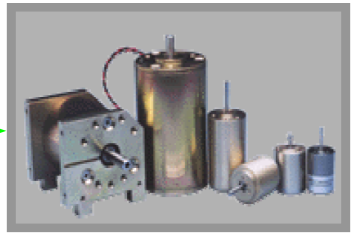
Camera – image recogn.





External keyboard

Display panel



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

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.

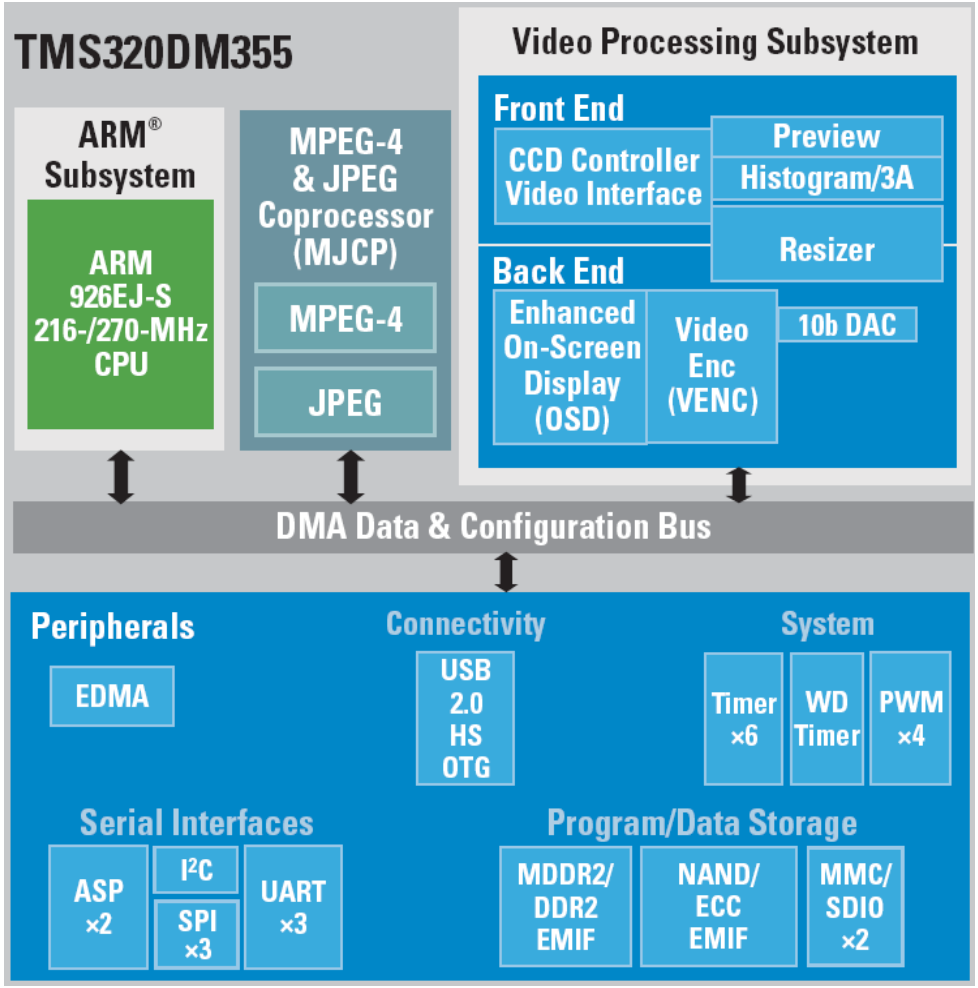
In the case when all subcomponents cannot be integrated on common silicon substrate, the following components are fabricated on different crystals and packed in single package, SiP (System-in-a-package).

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 (1)

Features of AT91SAM9263 Microcontroller:

- Architecture type: System-On-Chip,
- ARM core: ARM926EJ-S (220 MIPS at 200 MHz),
- MMU (Memory Management Unit),
- Direct Memory Access controller (27 channels DMA),
- Support for EmbeddedICE debugger,
- Available DSP (Digital Signal Processing) instructions and support for Java,
- Rich peripheral devices:
  - Driver for LCD display: TFT/STN (2D graphics co-processor, 2048x2048),
  - Driver for digital camera,
- Chip package: BGA 324,

Chip Identification

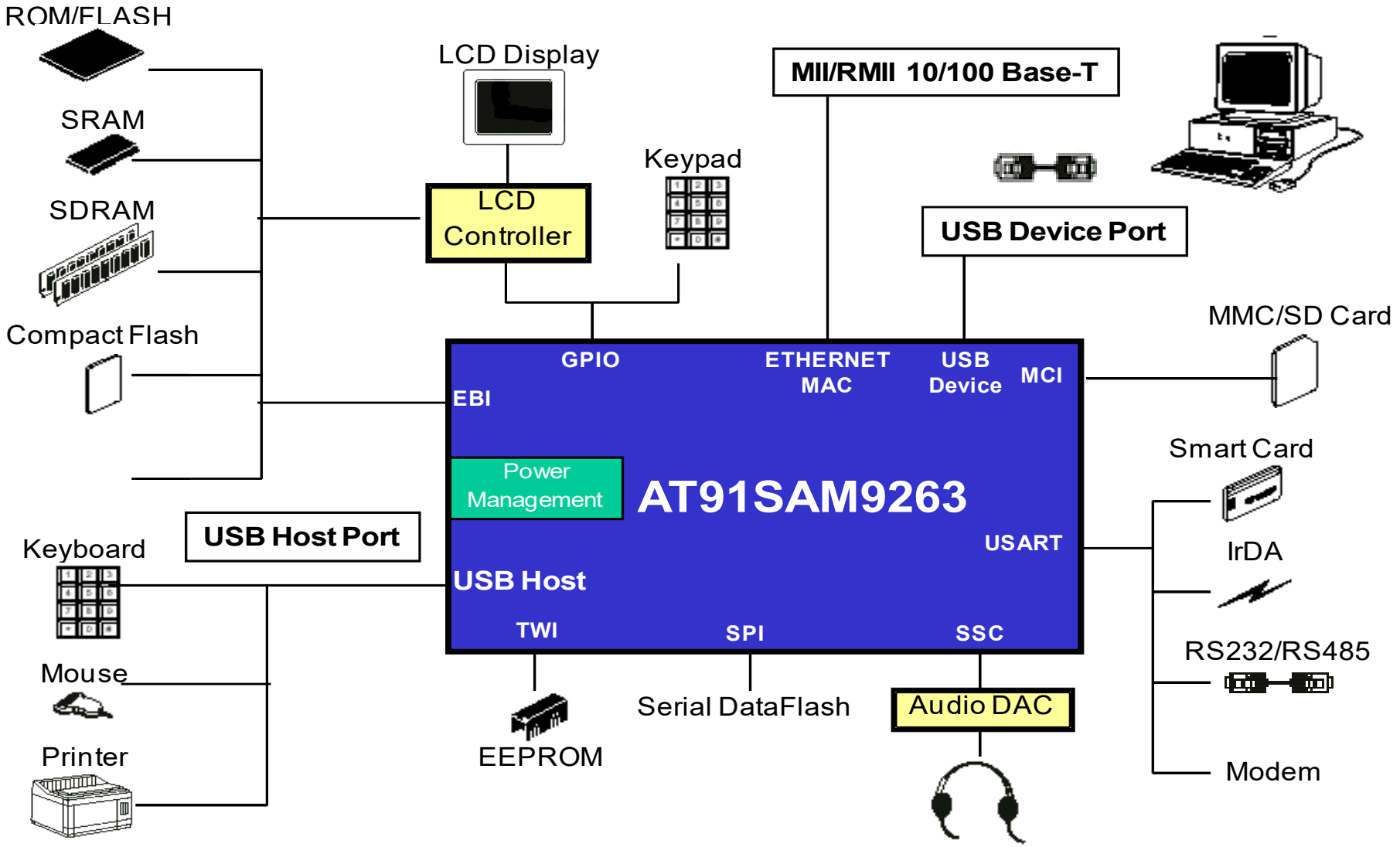
- Chip ID: 0x019607A0
- JTAG ID: 0x05B0C03F
- ARM926 TAP ID: 0x0792603F

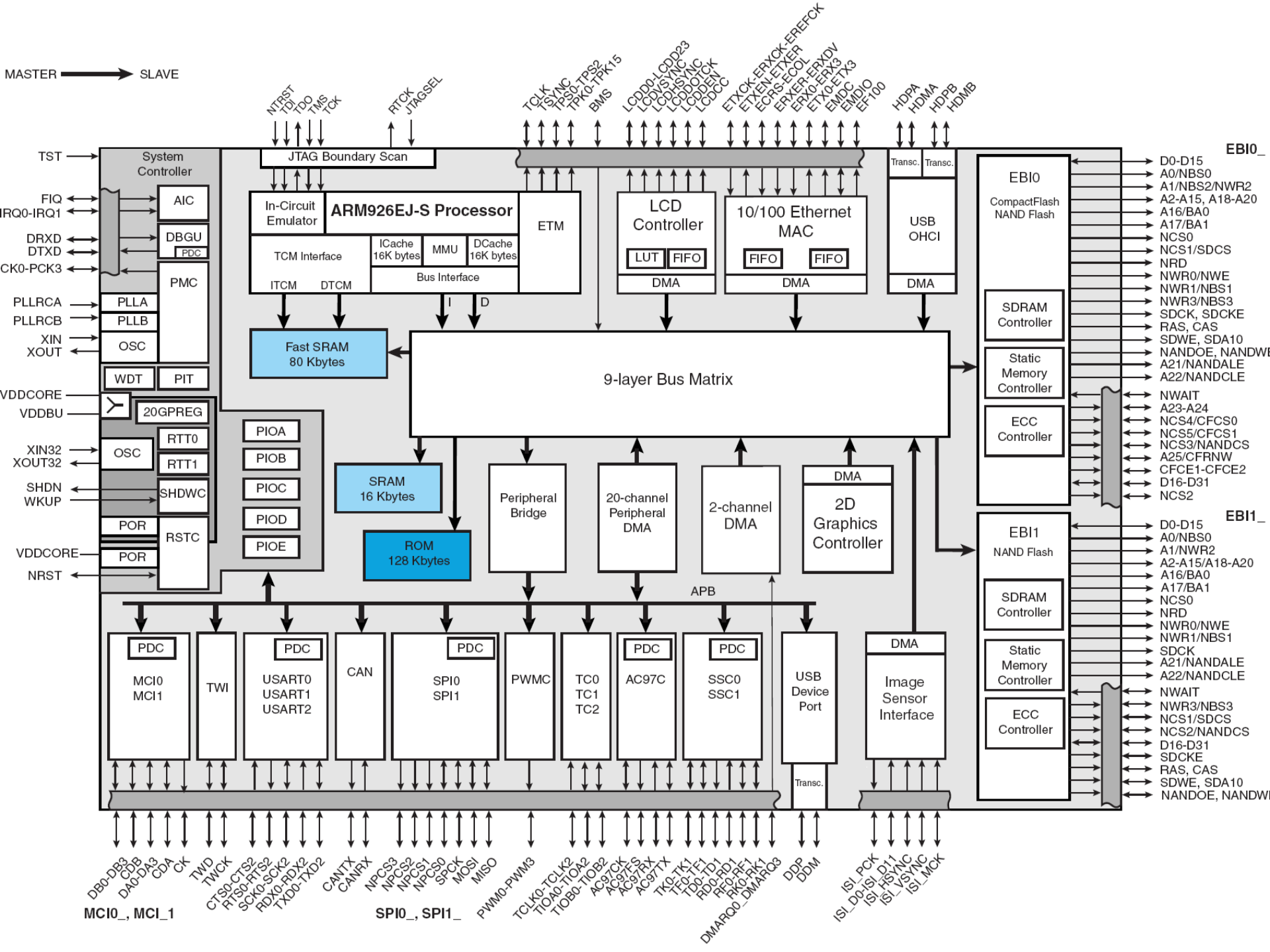
Microcontroller AT91SAM9263 (2)

Communication interfaces of AT91SAM9263:

- 9-layers EBI bus (External Bus Interface, 41.6 Gbps),
- I2S controller,
- USB 2.0 controller (host + device, 12 Mbps),
- Ethernet 10/100 Mbit/s controller,
- CAN controller (Controlled Area Network, 1 Mbps),
- USART controller (Universal Serial Asynchronous Receiver-Transmitter, 4 channels),
- SPI controller (Serial Peripheral Interfaces, 50 Mbps),
- CompactFlash and MMC/SD, SDIO (MCI) cards controller ,
- TWI controller (two-wire interface, GPRS modem, Wi-Fi, ...).

Microcontroller AT91SAM9263 (3)




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
TST
System Controller
JTAG Boundary Scan
FIQ
IRQ0-IRQ1
AIC
DRXD
DTXD
DBGU
PDC
CK0-PCK3
PMC
PLLRCA
PLLRCB
PLLA
PLLB
XIN
XOUT
OSC
WDT
PIT
VDDCORE
VDDBU
20GPREG
XIN32
XOUT32
OSC
RTT0
RTT1
SHDN
WKUP
SHDWC
POR
RSTC
VDDCORE
POR
NRST
In-Circuit Emulator
ARM926EJ-S Processor
ETM
TCM Interface
ITCM
DTCM
ICache 16K bytes
MMU
DCache 16K bytes
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
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
EBI0_
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
NAND Flash
SDRAM Controller
Static Memory Controller
ECC Controller
EBI1_
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_

GNU Tools for ARM processors

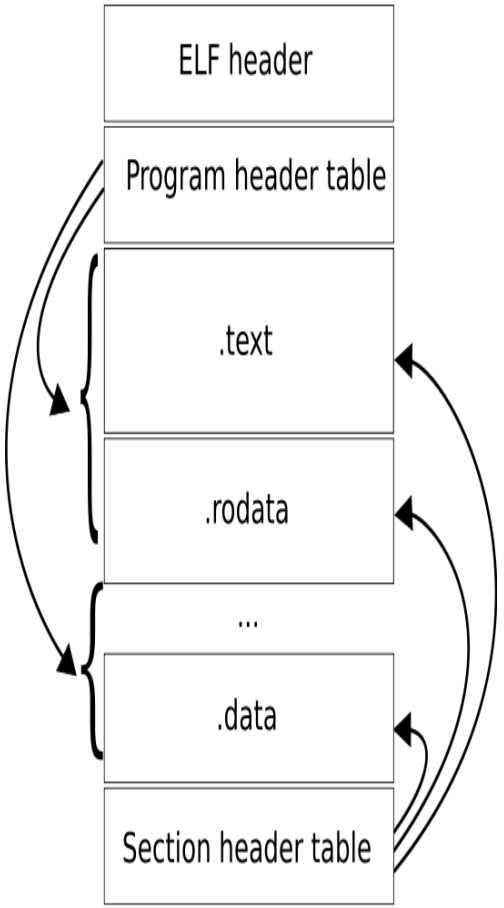
GNU ARM toolchain – tools for ARM processors available on GNU GPL (General Public License) licence.

Currently available tools (***http://www.gnuarm.org/***):

- GCC-4.3 toolchain (Linux):
  - Compiler: **gcc-4.3.2**
  - Useful tools: **binutils-2.19**
  - Library C/C++: **newlib-1.16**
  - Debugger compatible with GDB: **insight-6.8**
  - Tools are installed in **/opt/arm_tools/...**
  - Header files and scripts are in **/opt/arm_user/...**
- Cygwin (Windows):
  - binutils-2.19, gcc-4.3.2-c-c++, newlib-1.16.0, insight-6.8, **setup.exe**
- Debuger JTAG with USB interface
  (more information available in DMCS)

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

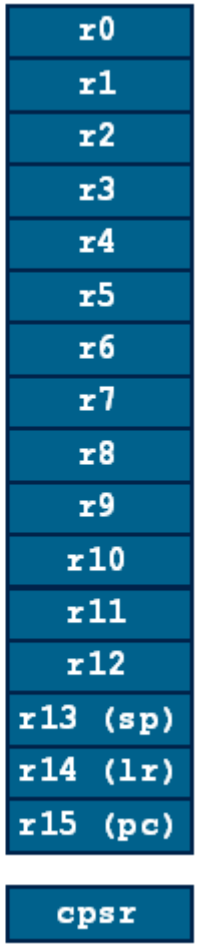
/t – display data in binary format

/d – display data in decimal format

Processor registers and GDB

(gdb) info r

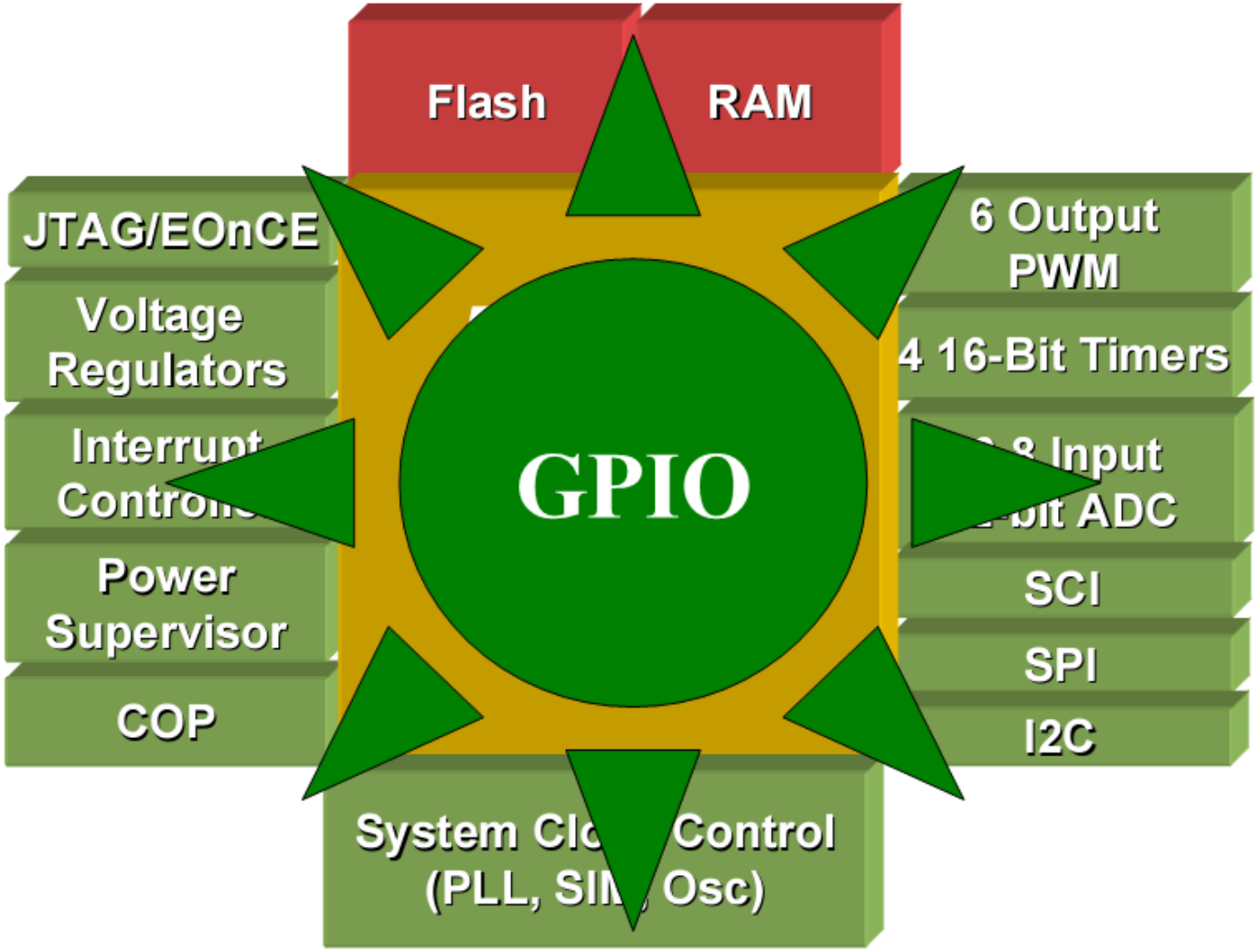
```
r0             0x2          0x2
r1             0x20000ba4   0x20000ba4
r2             0x57b        0x57b
r3             0x270f       0x270f
r4             0x300069     0x300069
r5             0x3122dc     0x3122dc
r6             0x1000       0x1000
r7             0x800bc004   0x800bc004
r8             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
```

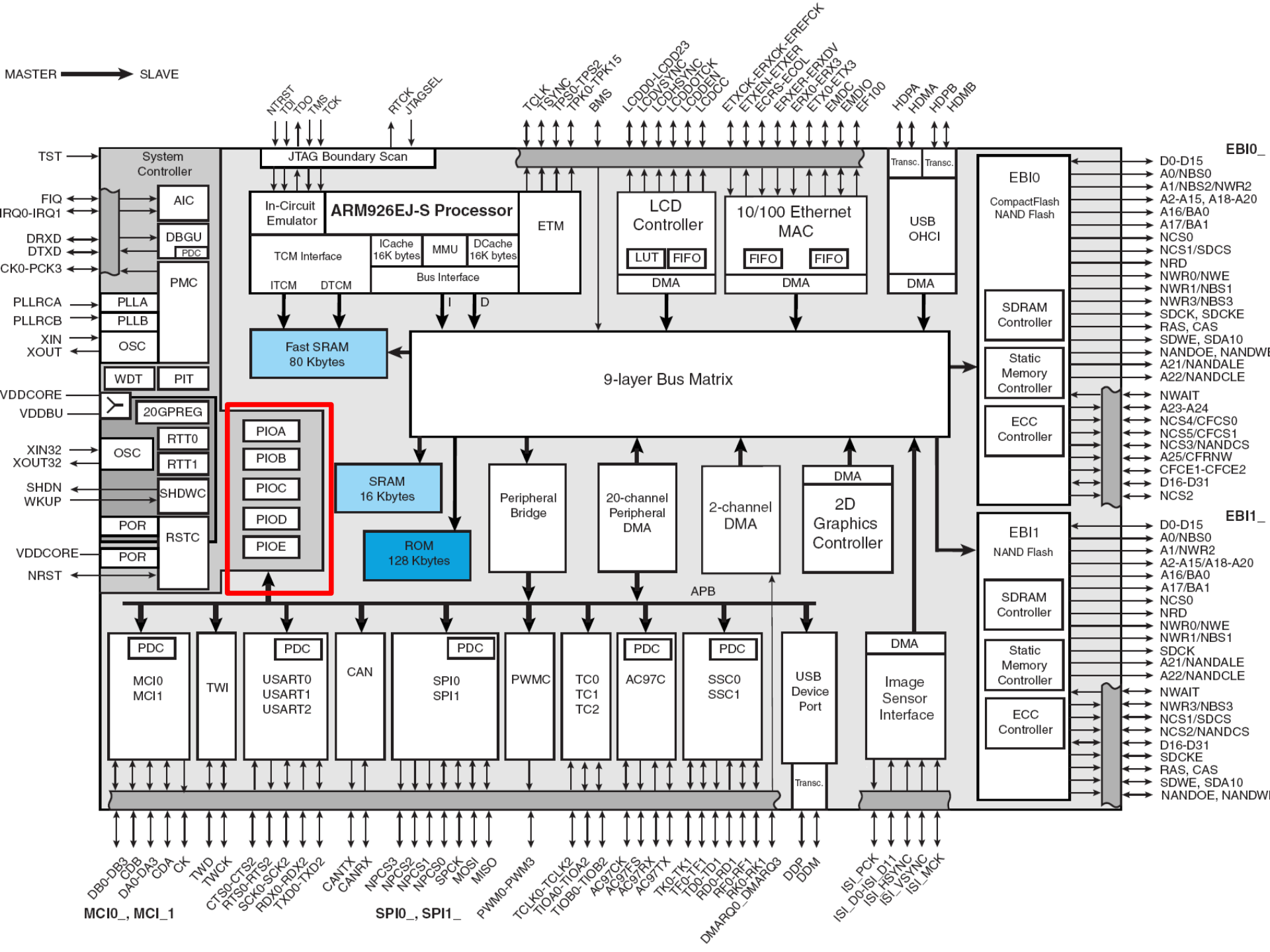


ARM Microcontrollers

Input-Output ports of AMR processor based on ATMEL ARM AT91SAM9263

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
TST
System Controller
JTAG Boundary Scan
FIQ
IRQ0-IRQ1
AIC
DRXD
DTXD
DBGU
PDC
CK0-PCK3
PMC
PLLRCA
PLLRCB
PLLA
PLLB
XIN
XOUT
OSC
WDT
PIT
VDDCORE
VDDBU
20GPREG
XIN32
XOUT32
RTT0
RTT1
SHDN
WKUP
SHDWC
POR
RSTC
NRST
In-Circuit Emulator
ARM926EJ-S Processor
ETM
TCM Interface
ICache 16K bytes
MMU
DCache 16K bytes
Bus Interface
ITCM
DTCM
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
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
EBI1
NAND Flash
SDRAM Controller
Static Memory Controller
ECC Controller
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
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
EBI0_
EBI1_
DB0-DB3
CDB
DA0-DA3
CDA
CK
TWD
TWCK
CTS0-CTS2
RTS0-RTS2
SCK0-SCK2
RDX0-RDX2
TXD0-TXD2
MCI0_, MCI_1
CANTX
CANRX
NPCS3
NPCS2
NPCS1
NPCS0
SPCK
MOSI
MISO
SPI0_, SPI1_
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

Documentation for AT91SAM9263 Microcontroller

Features

- Incorporates the ARM926EJ-S™ ARM® Thumb® Processor
  - DSP Instruction Extensions, Jazelle® Technology for Java® Acceleration
  - 16 Kbyte Data Cache, 16 Kbyte Instruction Cache, Write Buffer
  - 220 MIPS at 200 MHz
  - Memory Management Unit
  - EmbeddedICE™, Debug Communication Channel Support
  - Mid-level Implementation Embedded Trace Macrocell™
- Bus Matrix
  - Nine 32-bit-layer Matrix, Allowing a Total of 28.8 Gbps of On-chip Bus Bandwidth
  - Boot Mode Select Option, Remap Command
- Embedded Memories
  - One 128 Kbyte Internal ROM, Single-cycle Access at Maximum Bus Matrix Speed
  - One 80 Kbyte Internal SRAM, Single-cycle Access at Maximum Processor or Bus Matrix Speed
  - One 16 Kbyte Internal SRAM, Single-cycle Access at Maximum Bus Matrix Speed
- Dual External Bus Interface (EBI0 and EBI1)
  - EBI0 Supports SDRAM, Static Memory, ECC-enabled NAND Flash and CompactFlash®
  - EBI1 Supports SDRAM, Static Memory and ECC-enabled NAND Flash
- DMA Controller (DMAC)
  - Acts as one Bus Matrix Master
  - Embeds 2 Unidirectional Channels with Programmable Priority, Address Generation, Channel Buffering and Control
- Twenty Peripheral DMA Controller Channels (PDC)
- LCD Controller
  - Supports Passive or Active Displays
  - Up to 24 bits per Pixel in TFT Mode, Up to 16 bits per Pixel in STN Color Mode
  - Up to 16M Colors in TFT Mode, Resolution Up to 2048x2048, Supports Virtual Screen Buffers

ATMEL®

AT91 ARM Thumb Microcontrollers

AT91SAM9263

Preliminary

Documentation for AT91SAM9263 – I/O Ports

AT91SAM9263 Preliminary

31. Parallel Input/Output Controller (PIO)

31.1 Overview

The Parallel Input/Output Controller (PIO) manages up to 32 fully programmable input/output lines. Each I/O line may be dedicated as a general-purpose I/O or be assigned to a function of an embedded peripheral. This assures effective optimization of the pins of a product.

Each I/O line is associated with a bit number in all of the 32-bit registers of the 32-bit wide User Interface.

Each I/O line of the PIO Controller features:

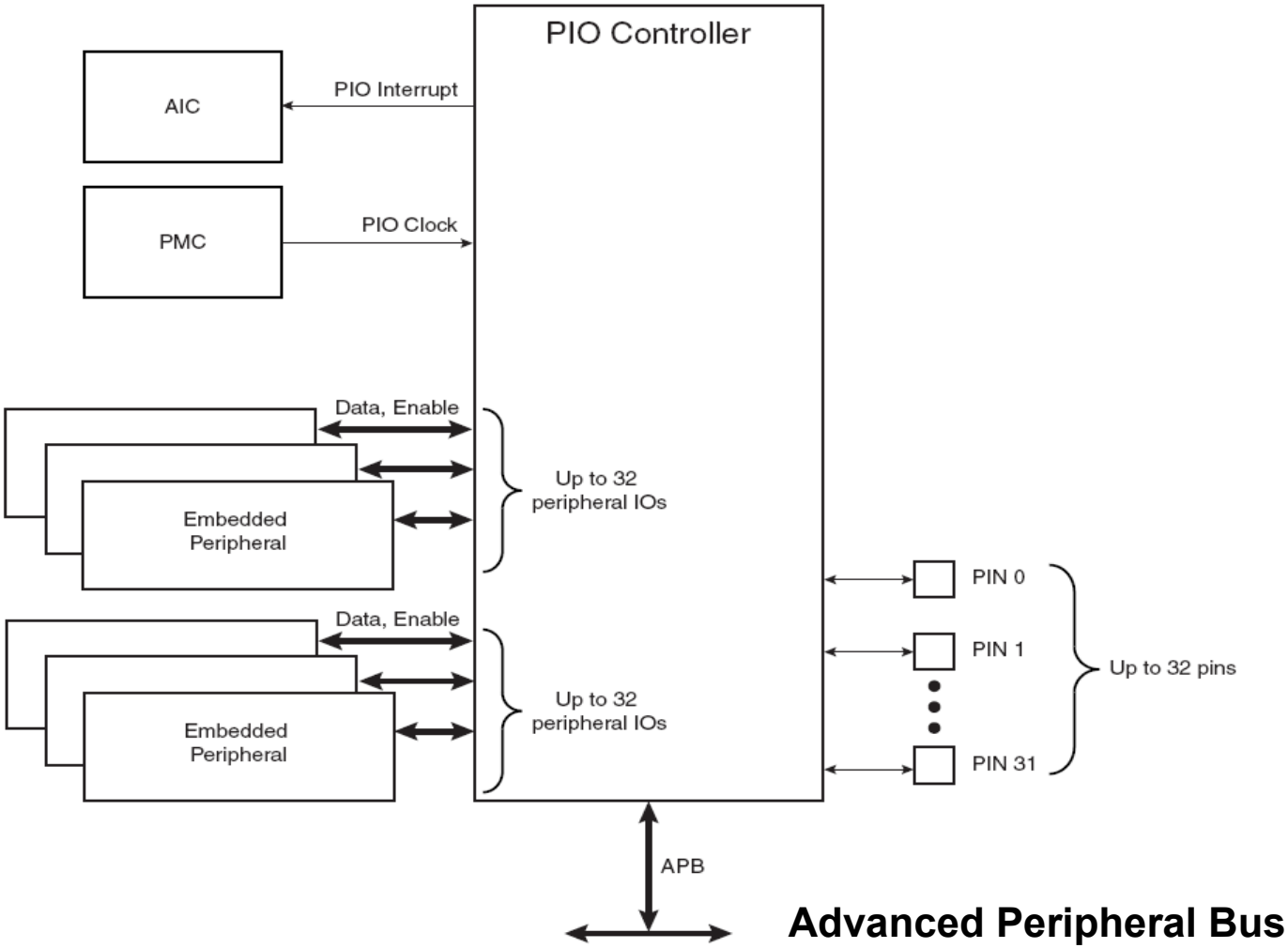
- An input change interrupt enabling level change detection on any I/O line.
- A glitch filter providing rejection of pulses lower than one-half of clock cycle.
- Multi-drive capability similar to an open drain I/O line.
- Control of the the pull-up of the I/O line.
- Input visibility and output control.

The PIO Controller also features a synchronous output providing up to 32 bits of data output in a single write operation.

Źródło: ATMEL, doc6249.pdf, strona 425

Block Diagram of 32-bits I/O Port

Figure 31-1. Block Diagram



Power Consumption vs Clock Signal

31.3.3 Power Management

The Power Management Controller controls the PIO Controller clock in order to save power. Writing any of the registers of the user interface does not require the PIO Controller clock to be enabled. This means that the configuration of the I/O lines does not require the PIO Controller clock to be enabled.

However, when the clock is disabled, not all of the features of the PIO Controller are available. Note that the Input Change Interrupt and the read of the pin level require the clock to be validated.

After a hardware reset, the PIO clock is disabled by default.

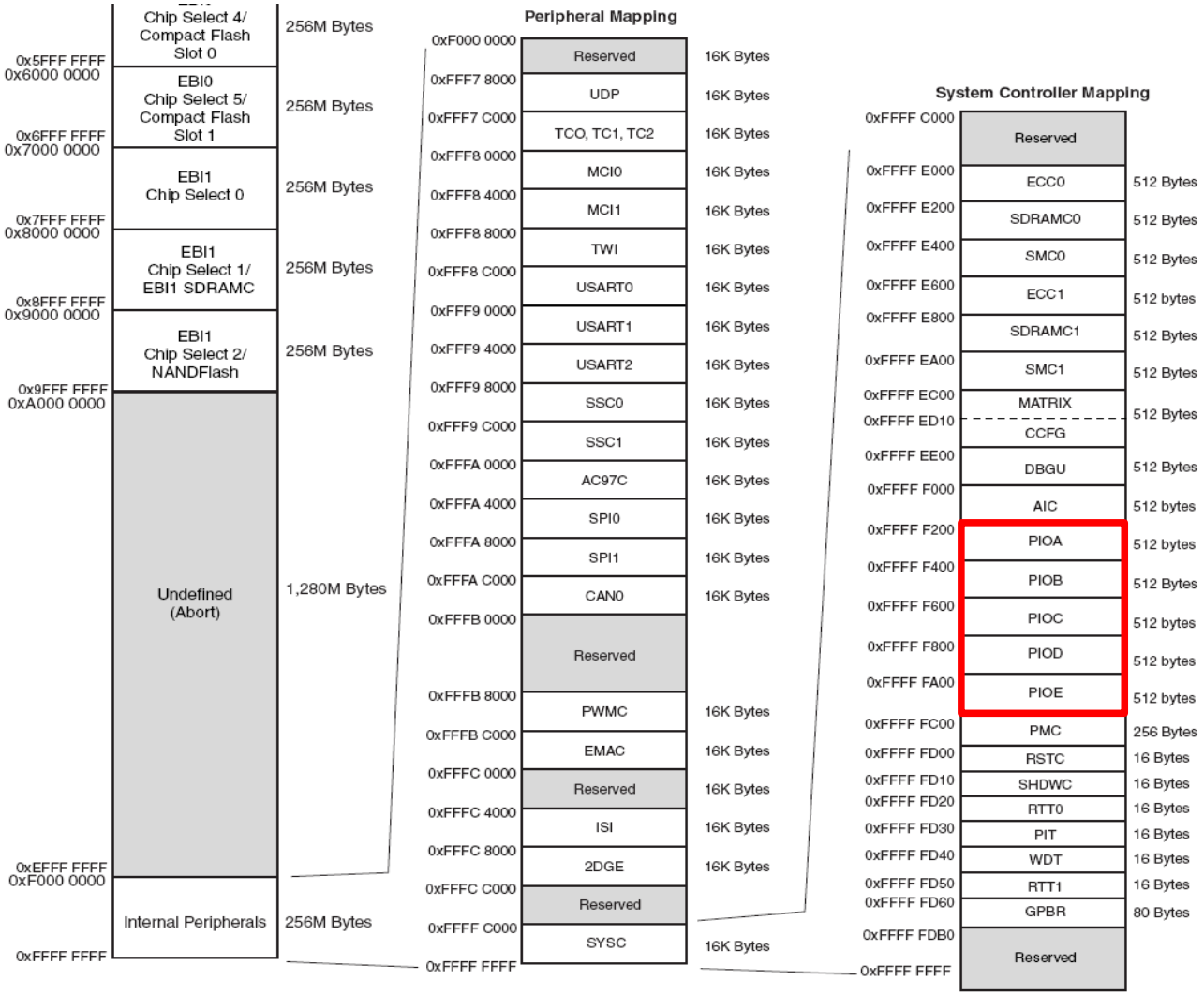
The user must configure the Power Management Controller before any access to the input line information.

Control Registers for I/O ports

Table 31-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved			
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or(2) Read-write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register(4)	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved			
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	0x00000000
0x006C	Reserved			

Memory Map


Chip Select 4/ Compact Flash Slot 0
256M Bytes
0x5FFF FFFF
0x6000 0000
EBI0 Chip Select 5/ Compact Flash Slot 1
256M Bytes
0x6FFF FFFF
0x7000 0000
EBI1 Chip Select 0
256M Bytes
0x7FFF FFFF
0x8000 0000
EBI1 Chip Select 1/ EBI1 SDRAMC
256M Bytes
0x8FFF FFFF
0x9000 0000
EBI1 Chip Select 2/ NANDFlash
256M Bytes
0x9FFF FFFF
0xA000 0000
Undefined (Abort)
1,280M Bytes
0xEFFF FFFF
0xF000 0000
Internal Peripherals
256M Bytes
0xFFFF FFFF
Peripheral Mapping
0xF000 0000
Reserved
16K Bytes
0xFFF7 8000
UDP
16K Bytes
0xFFF7 C000
TCO, TC1, TC2
16K Bytes
0xFFF8 0000
MCI0
16K Bytes
0xFFF8 4000
MCI1
16K Bytes
0xFFF8 8000
TWI
16K Bytes
0xFFF8 C000
USART0
16K Bytes
0xFFF9 0000
USART1
16K Bytes
0xFFF9 4000
USART2
16K Bytes
0xFFF9 8000
SSC0
16K Bytes
0xFFF9 C000
SSC1
16K Bytes
0xFFFA 0000
AC97C
16K Bytes
0xFFFA 4000
SPI0
16K Bytes
0xFFFA 8000
SPI1
16K Bytes
0xFFFA C000
CAN0
16K Bytes
0xFFFB 0000
Reserved
0xFFFB 8000
PWMC
16K Bytes
0xFFFB C000
EMAC
16K Bytes
0xFFFC 0000
Reserved
16K Bytes
0xFFFC 4000
ISI
16K Bytes
0xFFFC 8000
2DGE
16K Bytes
0xFFFC C000
Reserved
0xFFFF C000
SYSC
16K Bytes
0xFFFF FFFF
System Controller Mapping
0xFFFF C000
Reserved
0xFFFF E000
ECC0
512 Bytes
0xFFFF E200
SDRAMC0
512 Bytes
0xFFFF E400
SMC0
512 Bytes
0xFFFF E600
ECC1
512 bytes
0xFFFF E800
SDRAMC1
512 Bytes
0xFFFF EA00
SMC1
512 Bytes
0xFFFF EC00
MATRIX
512 Bytes
0xFFFF ED10
CCFG
0xFFFF EE00
DBGU
512 Bytes
0xFFFF F000
AIC
512 bytes
0xFFFF F200
PIOA
512 bytes
0xFFFF F400
PIOB
512 Bytes
0xFFFF F600
PIOC
512 bytes
0xFFFF F800
PIOD
512 bytes
0xFFFF FA00
PIOE
512 bytes
0xFFFF FC00
PMC
256 Bytes
0xFFFF FD00
RSTC
16 Bytes
0xFFFF FD10
SHDWC
16 Bytes
0xFFFF FD20
RTT0
16 Bytes
0xFFFF FD30
PIT
16 Bytes
0xFFFF FD40
WDT
16 Bytes
0xFFFF FD50
RTT1
16 Bytes
0xFFFF FD60
GPBR
80 Bytes
0xFFFF FDB0
Reserved
0xFFFF FFFF

Documentation as Source of Registers' Information

31.6.12 PIO Controller Output Data Status Register

Name: PIO_ODSR

Addresses: 0xFFFFF238 (PIOA), 0xFFFFF438 (PIOB), 0xFFFFF638 (PIOC), 0xFFFFF838 (PIOD), 0xFFFFFA38 (PIOE)

Access Type: Read-only or Read-write

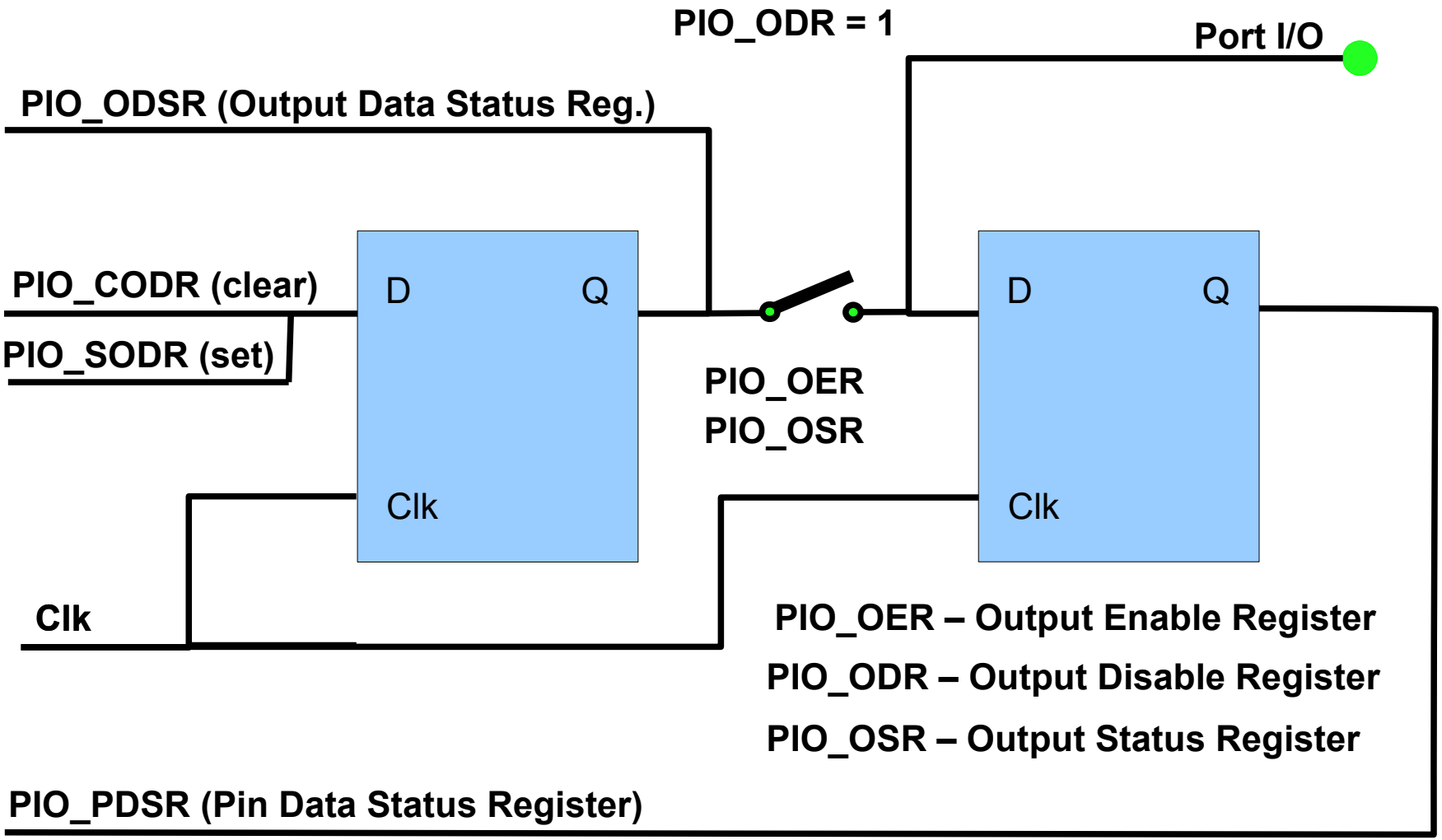
31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Data Status**

0 = The data to be driven on the I/O line is 0.

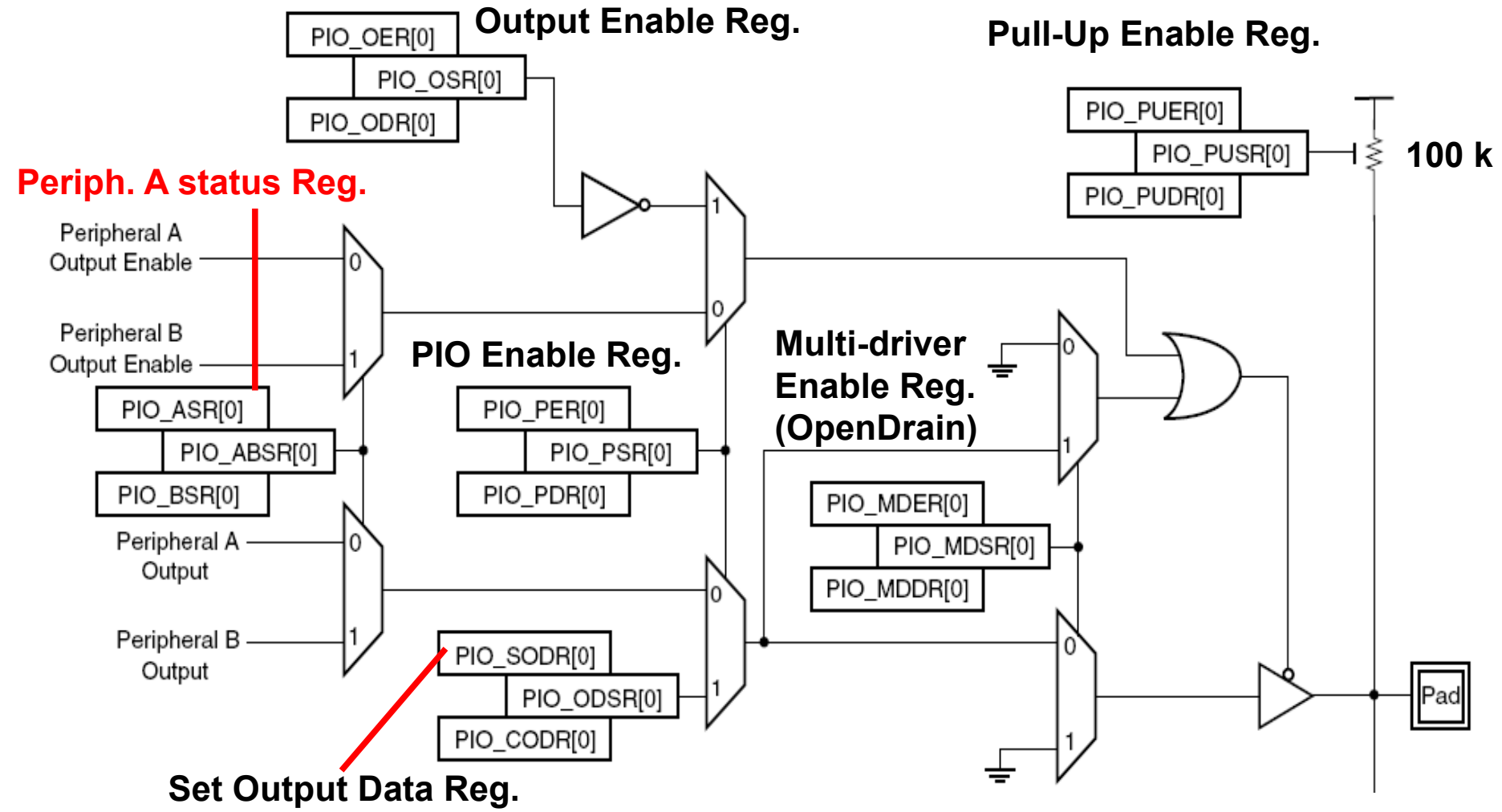
1 = The data to be driven on the I/O line is 1.

Simplified Block Diagram of I/O Port

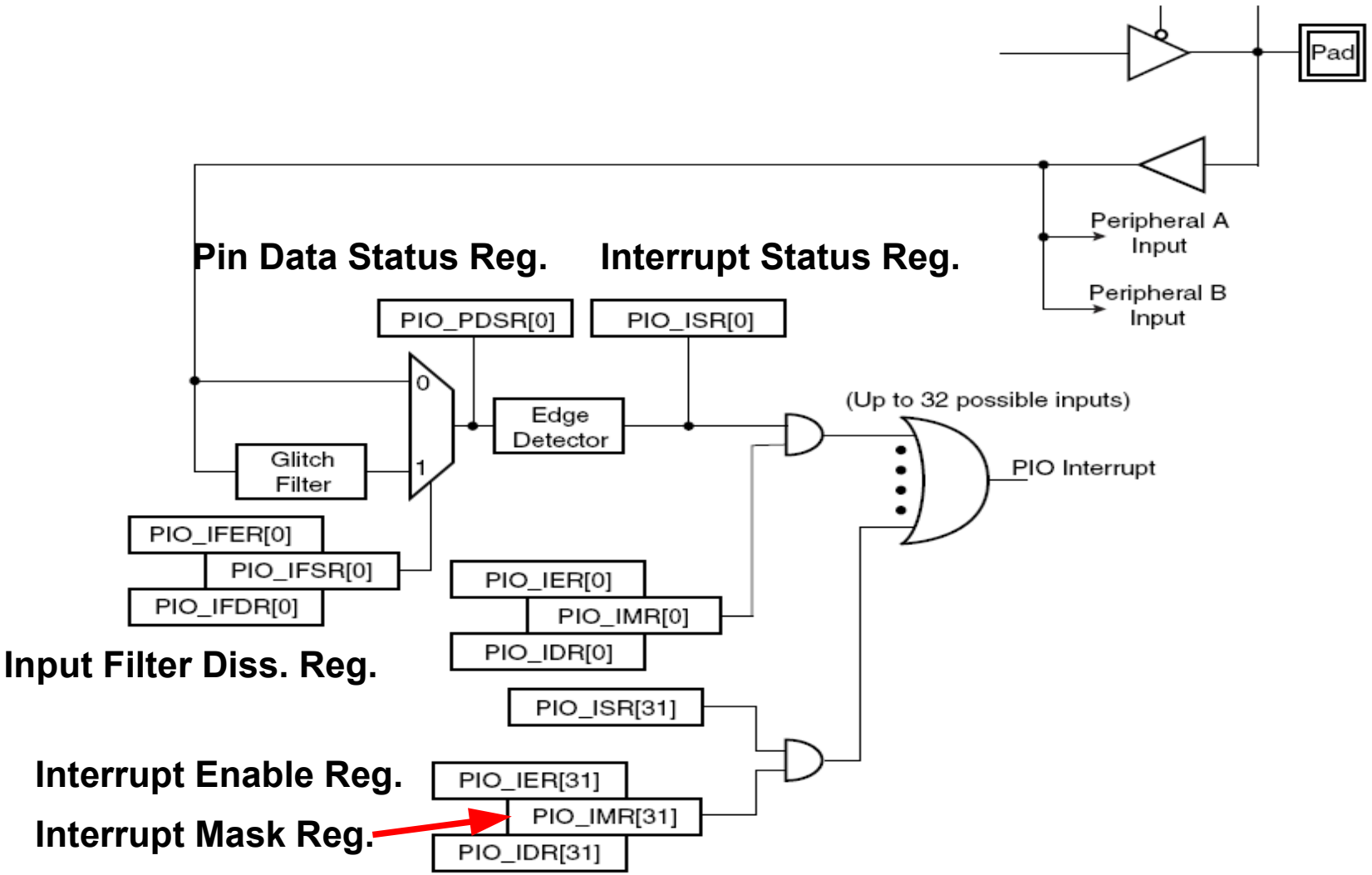


I/O Port – How to Control Output ?

Figure 31-3. I/O Line Control Logic



I/O – How to Read Input ?



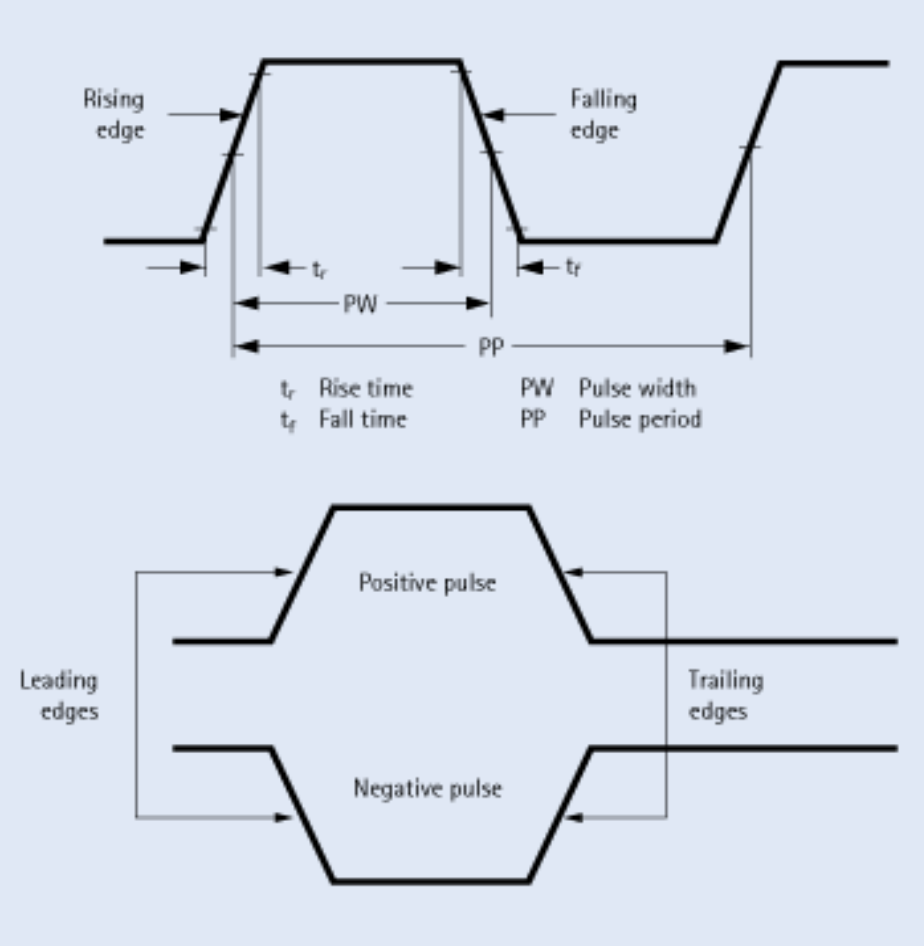
Digital Signal

Digital Signal can be characterised with:

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

Digital circuits can be triggered with:

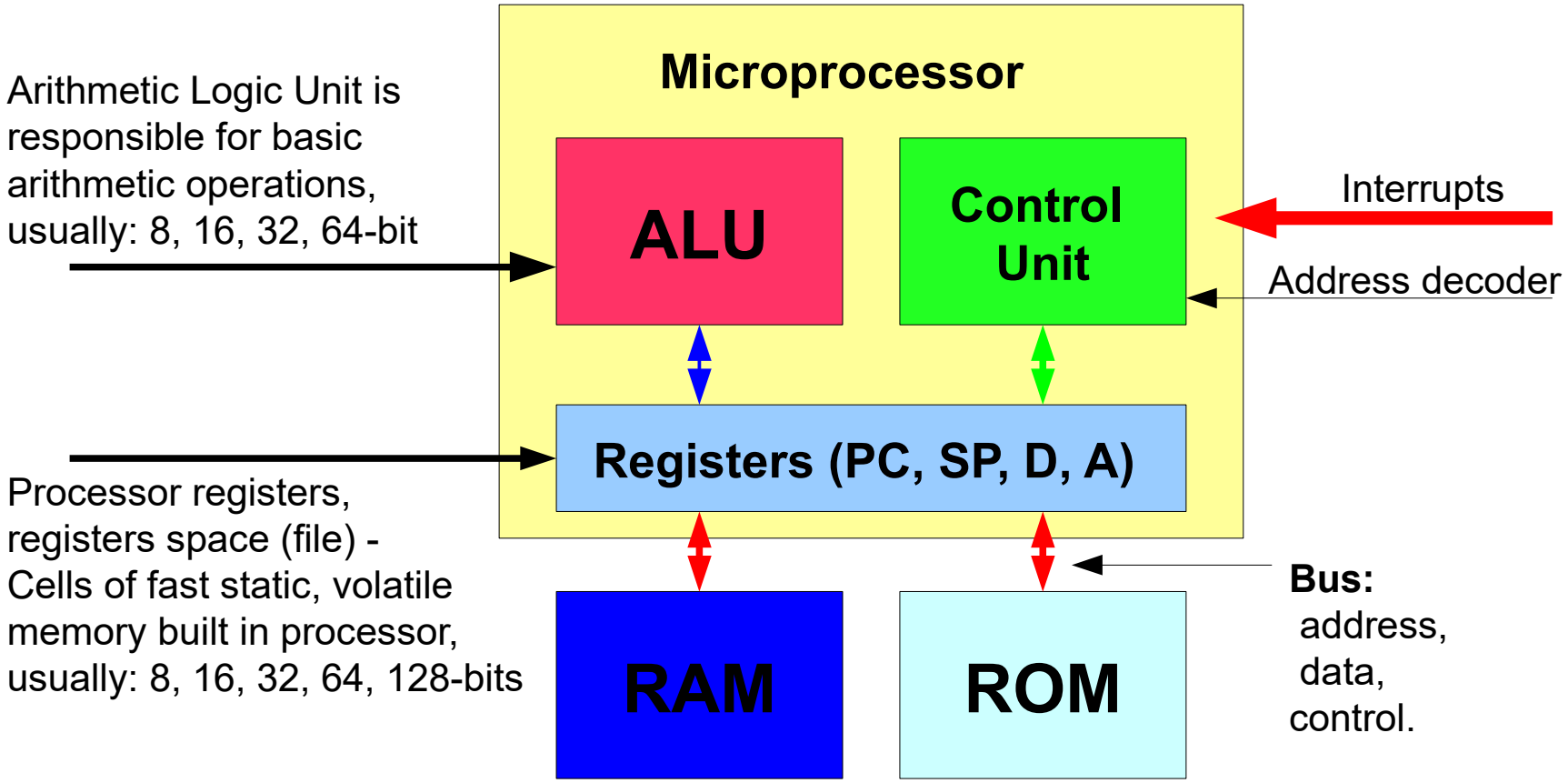
- Change of signal level (lower or higher than signal threshold level),
- Change of signal slope (transaction of digital signal from '0' to '1' or from '1' to '0').



Architectures of Embedded Devices

Microprocessor

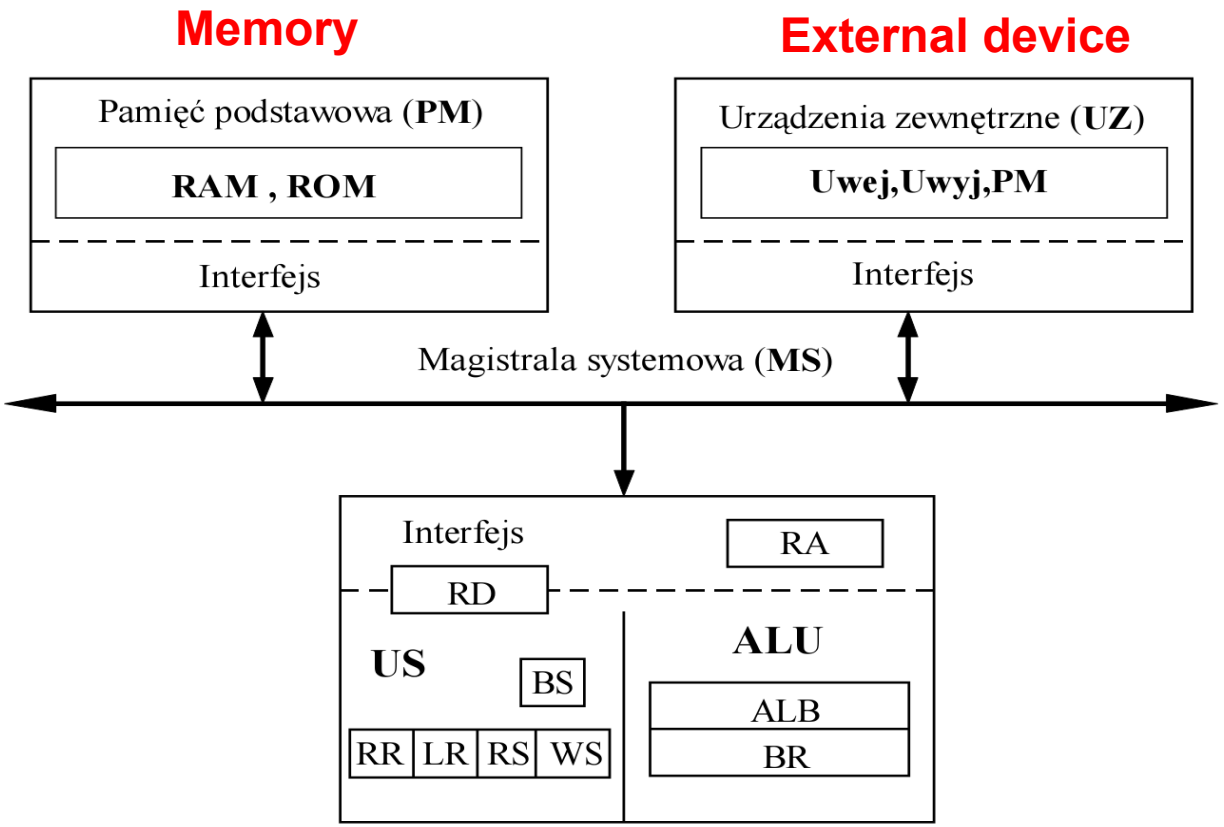
Microprocessor - digital circuit fabricated as a single integrated device able to process digital operations according to provided binary program



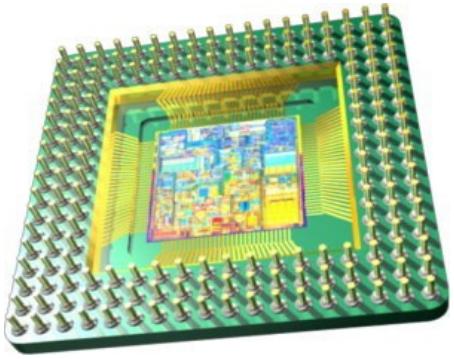
Computer System Architecture

Computer System is composed of three basic subsystems:

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



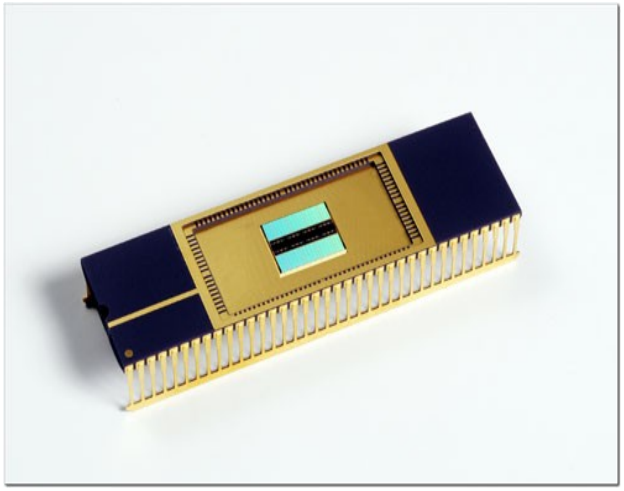
Computer buses



Address Bus

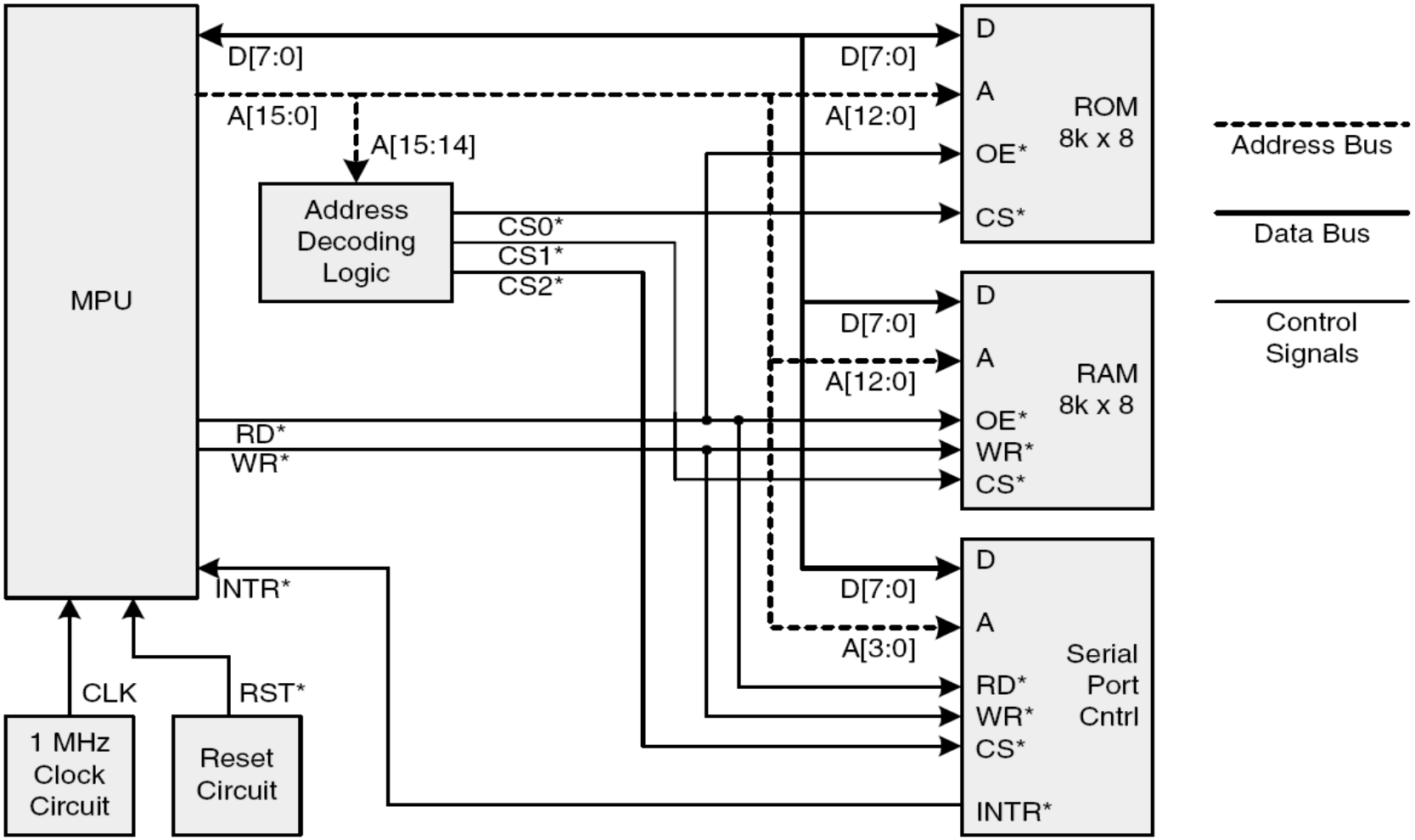
Data Bus

Control Bus



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

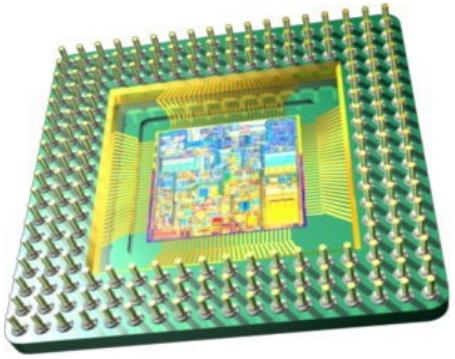
Example of 8-bit computer system

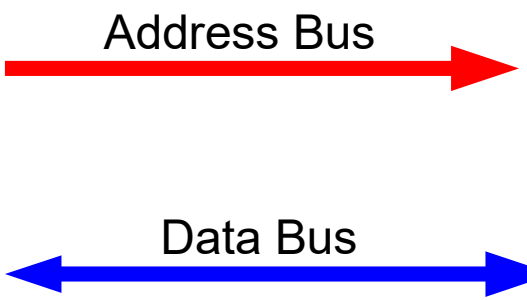


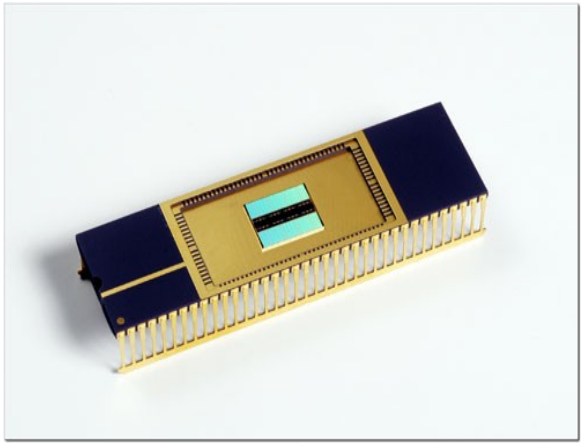
Von Neumann architecture

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.





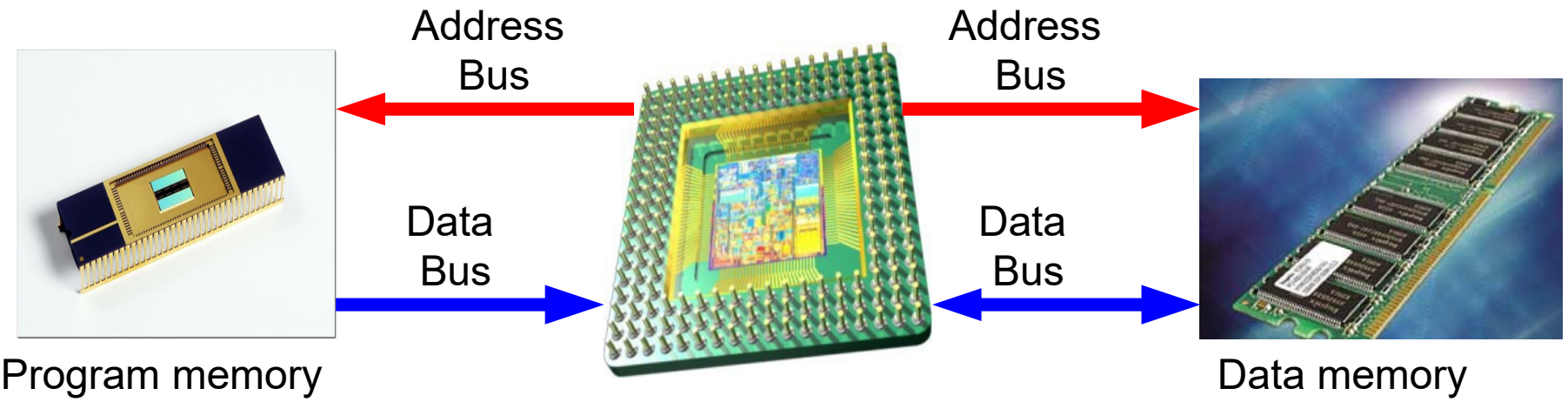


Harvard Architecture

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

Harvard Architecture**:**

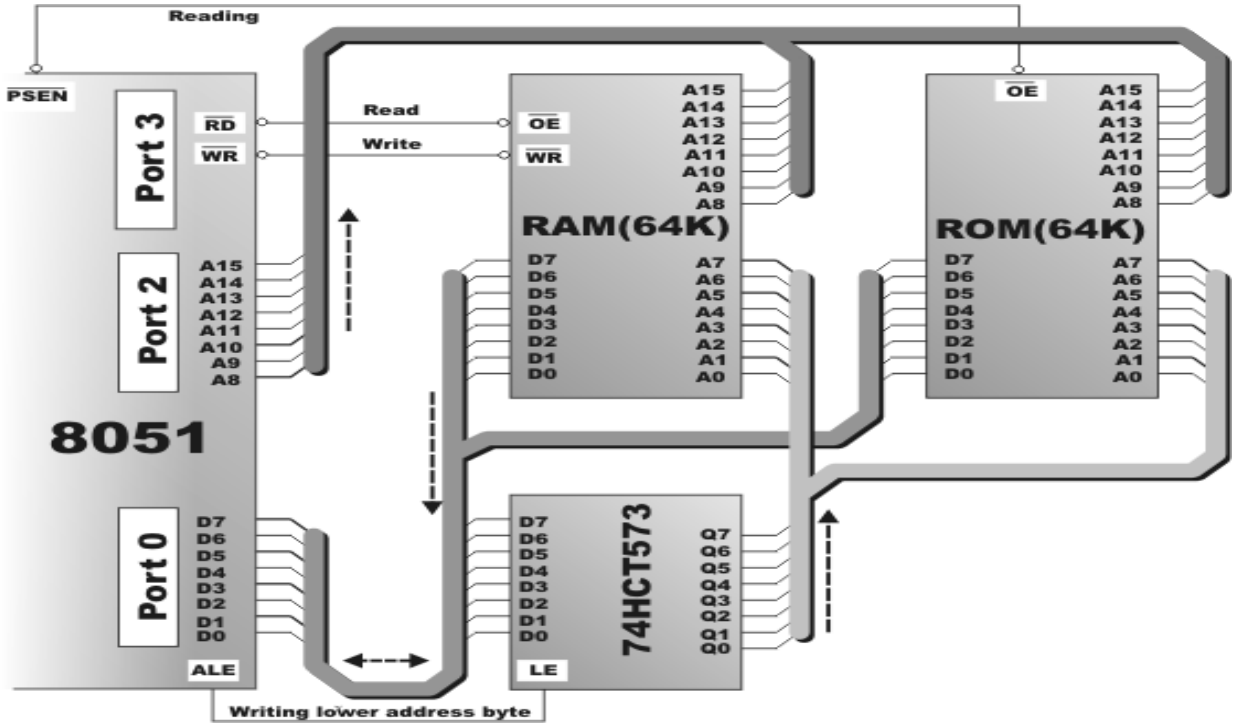
- Command and Data are stored in different memories,
- Allowed different organisation of memories (different data and command words lengths),
- Possible parallel access,
- Used in single-chip Microcontrollers



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).

Data memory **Program memory**



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

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

Registers

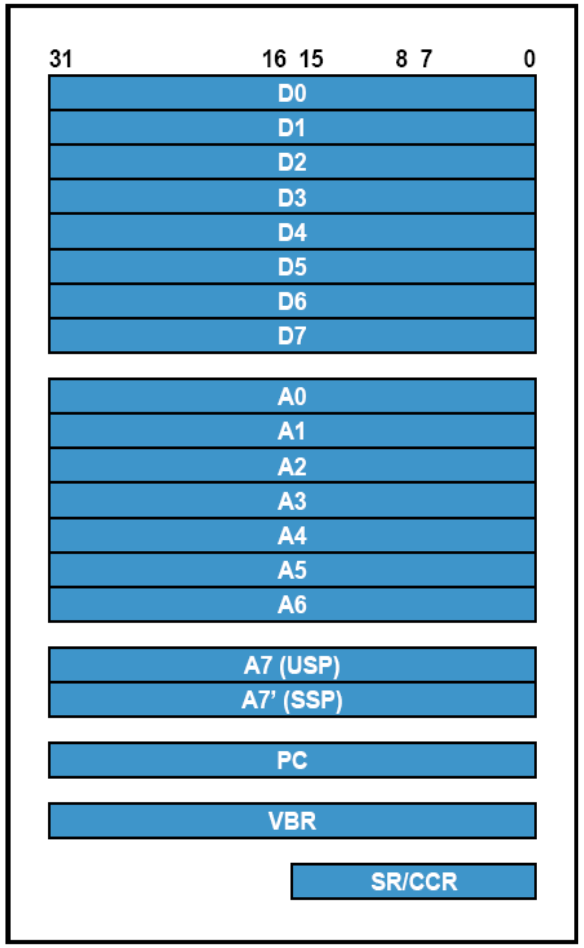
Processor registers are realised as cells of internal processor memory. Registers have usually small size (4/8/16/32/64/128 bits). Registers are used for storing temporary results, addresses in computer memory, configuration of peripheral devices, etc...

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:

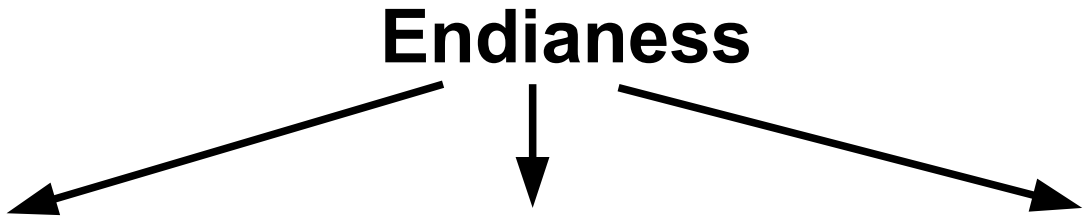
- Data registers – used for storing data and results, e.g. mnemonic arguments, results of calculations,
- Address registers – used for operations on addresses (stack pointer, program counter, segment register, etc...),
- General purpose registers – store both data and addresses,
- Floating point registers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



Registers of Freescale microcontroller

Endianess (1)

Byte – the smallest addressable unit of computer memory



Big-endian	middle-endian	Little-endian
...the lowest address contains the most significant bit...		...the highest address contains the most significant bit...
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 architecture

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	0x12
0x0000.0001	0x34
0x0000.0002	0x56
0x0000.0003	0x78
0x0000.0004	0x90

Endianess (3)

Byte 4 ... Byte 1
MSB LSB

Big-endian

0x0000.0000	Byte 4	Byte 3	Byte 2	Byte 1
0x0000.0004	Byte 8	Byte 7	Byte 6	Byte 5
0x0000.0008	Byte 12	...	...	...
0x0000.000C				
0x0000.0010				

Little-endian

0x0000.0000	Byte 1	Byte 2	Byte 3	Byte 4
0x0000.0004	Byte 5	Byte 6	Byte 7	Byte 8
0x0000.0008	Byte 9	...	...	...
0x0000.000C				
0x0000.0010				

Endianess (4)

Double Word (DW): **0x1234.5678**

Big-endian

	31 24	23 16	15 8	7 0
0x0000.0000	0x12	0x34	0x56	0x78
0x0000.0004	Byte 5	Byte 6	Byte 7	Byte 8
0x0000.0008	Byte 9	…	…	…
0x0000.000C				
0x0000.0010				

Little-endian

	31 24	23 16	15 8	7 0
0x0000.0000	0x78	0x56	0x34	0x12
0x0000.0004	Byte 8	Byte 7	Byte 6	Byte 5
0x0000.0008	Byte 12	…	…	…
0x0000.000C				
0x0000.0010				

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;
}
```

Bits and C language

```
volatile unsigned int * DataInMemory = 0x1000;
DataInMemory = 0;
DataInMemory = 0x12345678;
DataInMemory = 0xFFFF.FFFF;
```

How to clear single bit ?

How to set single bit ?

Operations on Register Bits (1)

```
volatile unsigned char* PORTA=0x4010.000A;
    *PORTA    = 0x1;
    *PORTA    = 7;
    *PORTA    = 010;
    *PORTA    = *PORTA  |  0x2;
    *PORTA   |= 0x1 | 0x2 | 0x8 ;
    *PORTA  &=  ~(0x2 | 0x4);
    *PORTA   ^=    (0x1 | 0x2);
    *PORTA   ^=     0x3;
    If (*PORTA & (0x1 | 0x4)) == 0  {...}
    while (*PORTA != 0x6)   {...}
    do {...} while (*PORTA & 0x4)
```

Operations on Register Bits (2)

```
#define PB0   0x1
#define PB1   0x2
#define PB2   1<<2
#define PB3   1<<3

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

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

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

Operations on Register Bits (3)

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

/* macro for bit-mask */
#define BIT(x)      (1 << (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=0x15, reg2=0x55;
        unsigned char = reg3, reg4;
        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)    0xFFFFF200     // (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*)0x4010.0002)
int i = PORTC.Bit_0;        /* read data */
PORTC.Bit_2 = 1;            /* write data */

Port_4bit* PortTC = (Port_4bit*) 0x4010.000F;
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;
```