Dariusz Makowski

Katedra Mikroelektroniki i Technik Informatycznych

tel. 631 2720

dmakow@dmcs.pl

http://neo.dmcs.p.lodz.pl/sw

Sprawy formalne

- Informacje wstępne
- Egzamin
- Laboratorium
- Materiały do wykładu

Zakres przedmiotu

- Laboratorium
- Systemy mikroprocesorowe, systemy wbudowane
- Rodzina procesorów ARM
- Urządzenia peryferyjne
- Pamięci i dekodery adresowe
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych
- Interfejsy w systemach wbudowanych
- Systemy czasu rzeczywistego

Zakres przedmiotu

- Laboratorium
- Systemy mikroprocesorowe, systemy wbudowane
- Rodzina procesorów ARM
- Urządzenia peryferyjne
- Pamięci i dekodery adresowe
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych
- Interfejsy w systemach wbudowanych
- Systemy czasu rzeczywistego

Laboratorium

Adres:

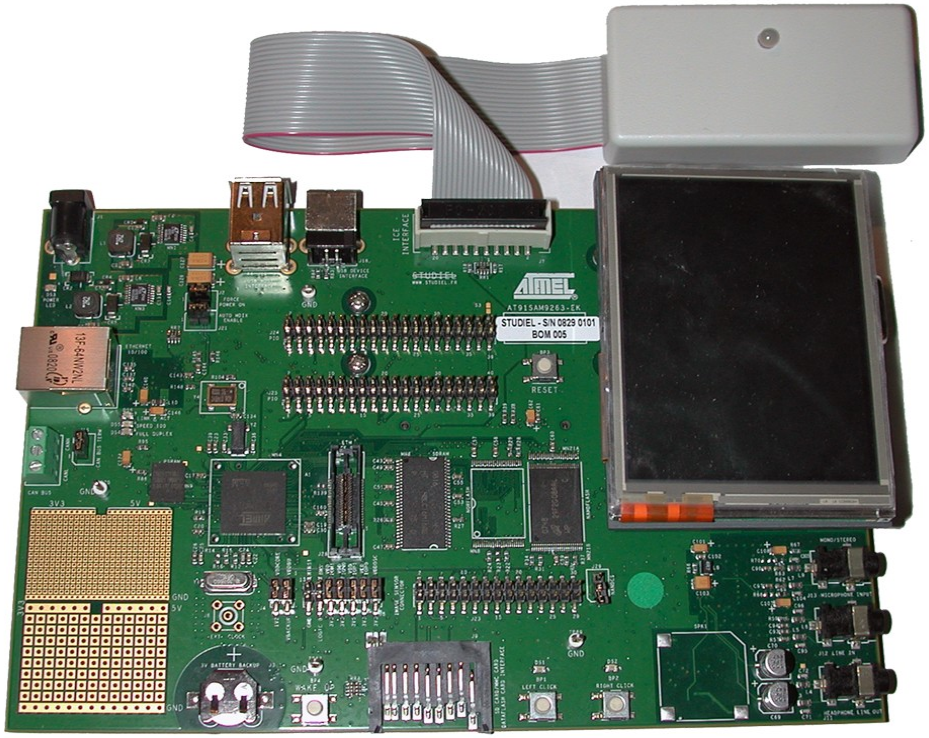
- Budynek B18, laboratorium M

Praktyczne ćwiczenia z użyciem procesora ARM:

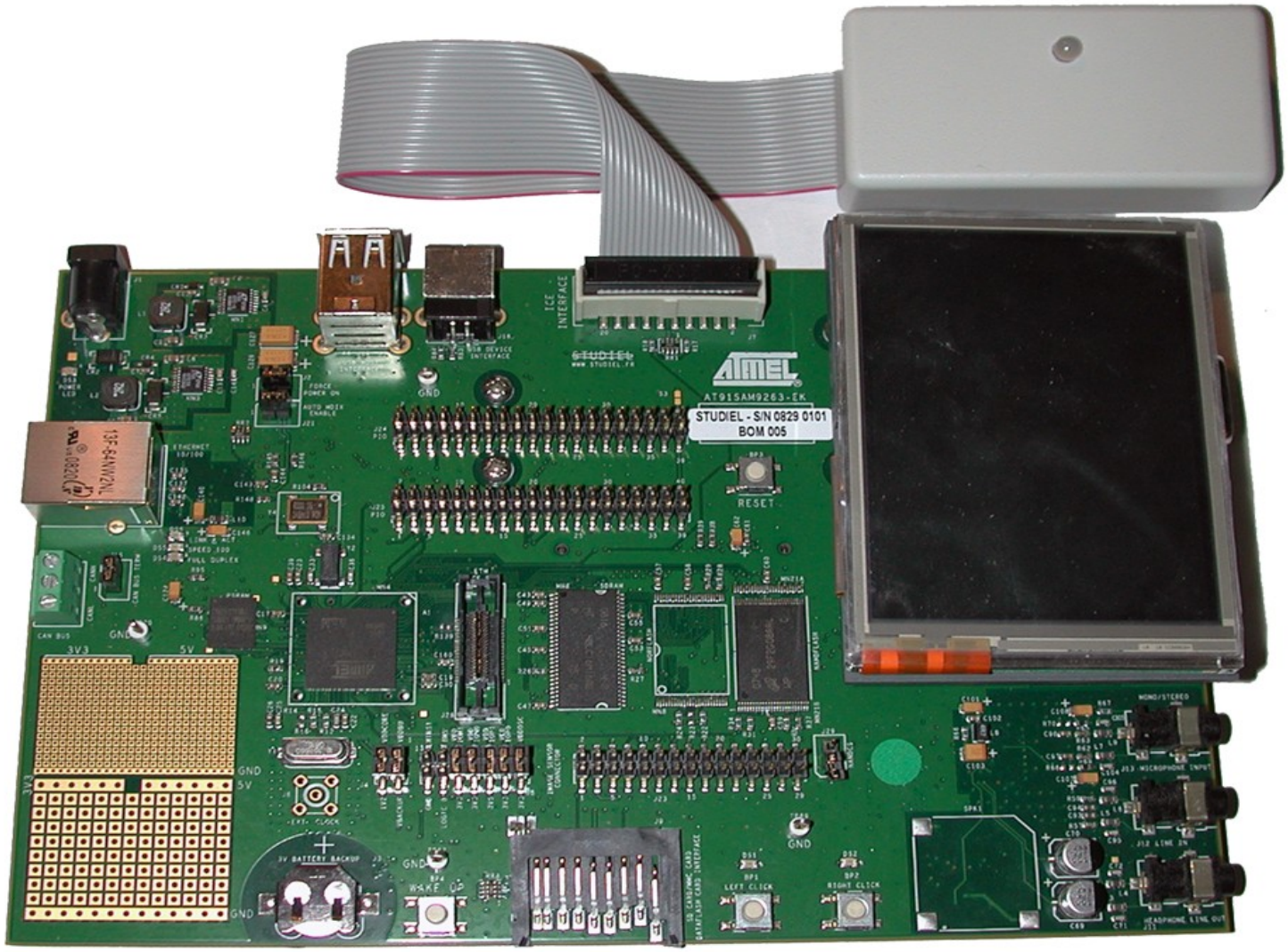
- GNU tools,
- AT91SAM9263-STARTUP-PAKET,
- Płyta nakładkowa z wyświetlaczem LED, enkoderem i termometrem,
- System czasu rzeczywistego RTEMS.

Zestaw ewaluacyjny firmy MSC (1)

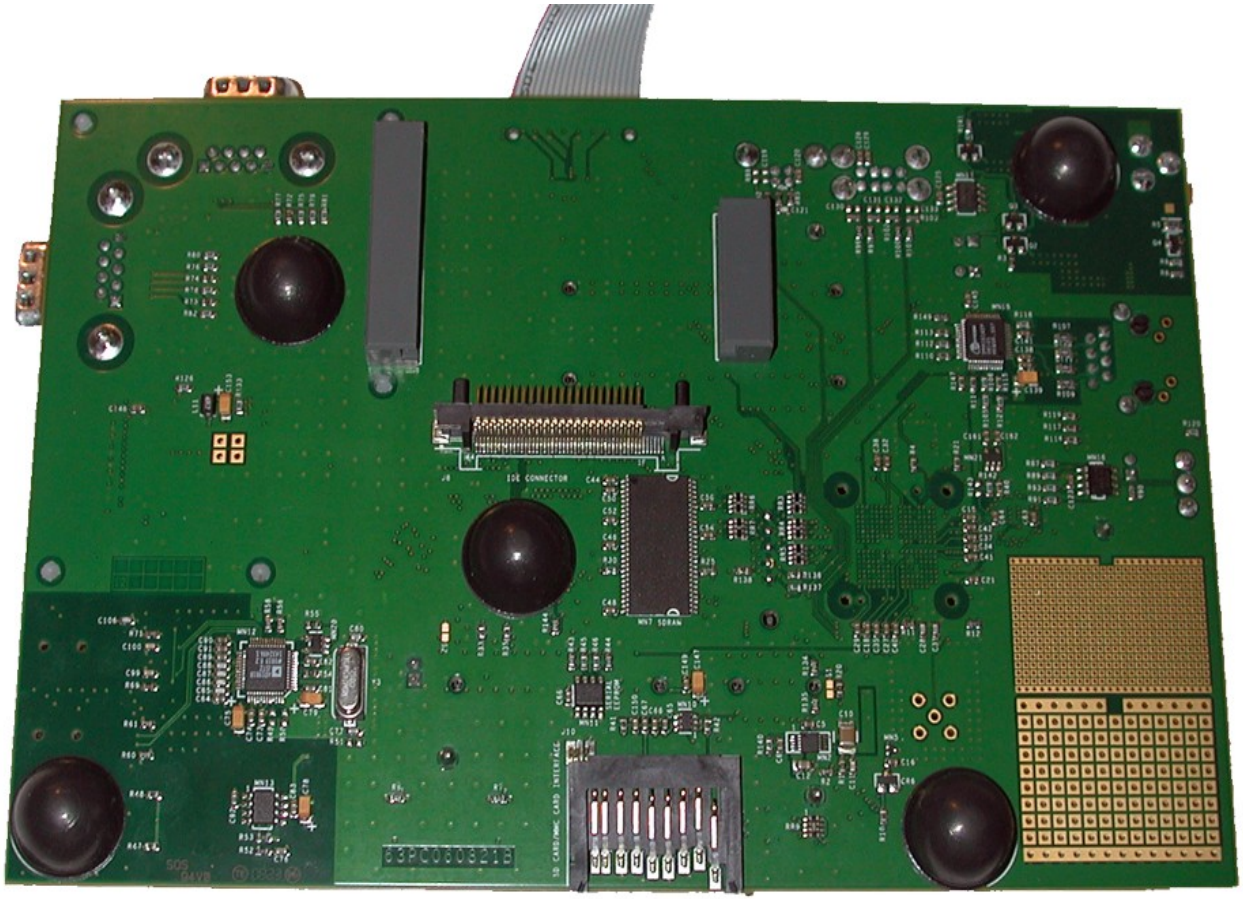
- **Procesor z rdzeniem ARM9TDMI firmy ATMEL:** AT91SAM9263
- **Dostępne pamięci:** 64 MB SDRAM, 256 MB NAND FLASH, 4 MB DataFlash, FlashCard slots
- **Dostępne interfejsy:** Ethernet 100-base TX, USB FS device, 2 x USB FS Host, CAN 2.0B, EIA RS232,
- **Wyświetlacz:** 3.5" 1/4 VGA TFT LCD z ekranem dotykowym
- **Kodek audio:** AC97 Audio DAC
- **Debug intefece:** JTAG
- **Interfejs do programowania:** JTAG, Free Atmel SAM-BA tools
- **Złącze:** SD/SDIO/MMC card slot
- **Oznaczenie producenta:** AT91SAM9263-STARTUP-PAKET



Zestaw ewaluacyjny firmy MSC (2)



Zestaw ewaluacyjny firmy MSC (3)



Mikrokontroler AT91SAM9263 (1)

Cechy mikrokontrolera AT91SAM9263:

- architektura typu System-On-Chip,
- Jądro ARM926EJ-S (220 MIPS dla 200 MHz),
- Jednostka zarządzająca pamięcią MMU (Memory Management Unit),
- Sterownik bezpośredniego dostępu do pamięci DMA (27 kanałów DMA),
- Wsparcie dla systemu debugowania EmbeddedICE,
- Dostępne instrukcje DSP (Digital Signal Processing) oraz wsparcie dla j. Java,
- Bogate urządzenia peryferyjne:
  - Sterownik wyświetlacza LCD TFT/STN (2D graphics co-processor, 2048x2048),
  - Sterownik camery cyfrowej,
- Układ dostarczany w obudowie BGA 324 (wyprowadzenia),

Chip Identification

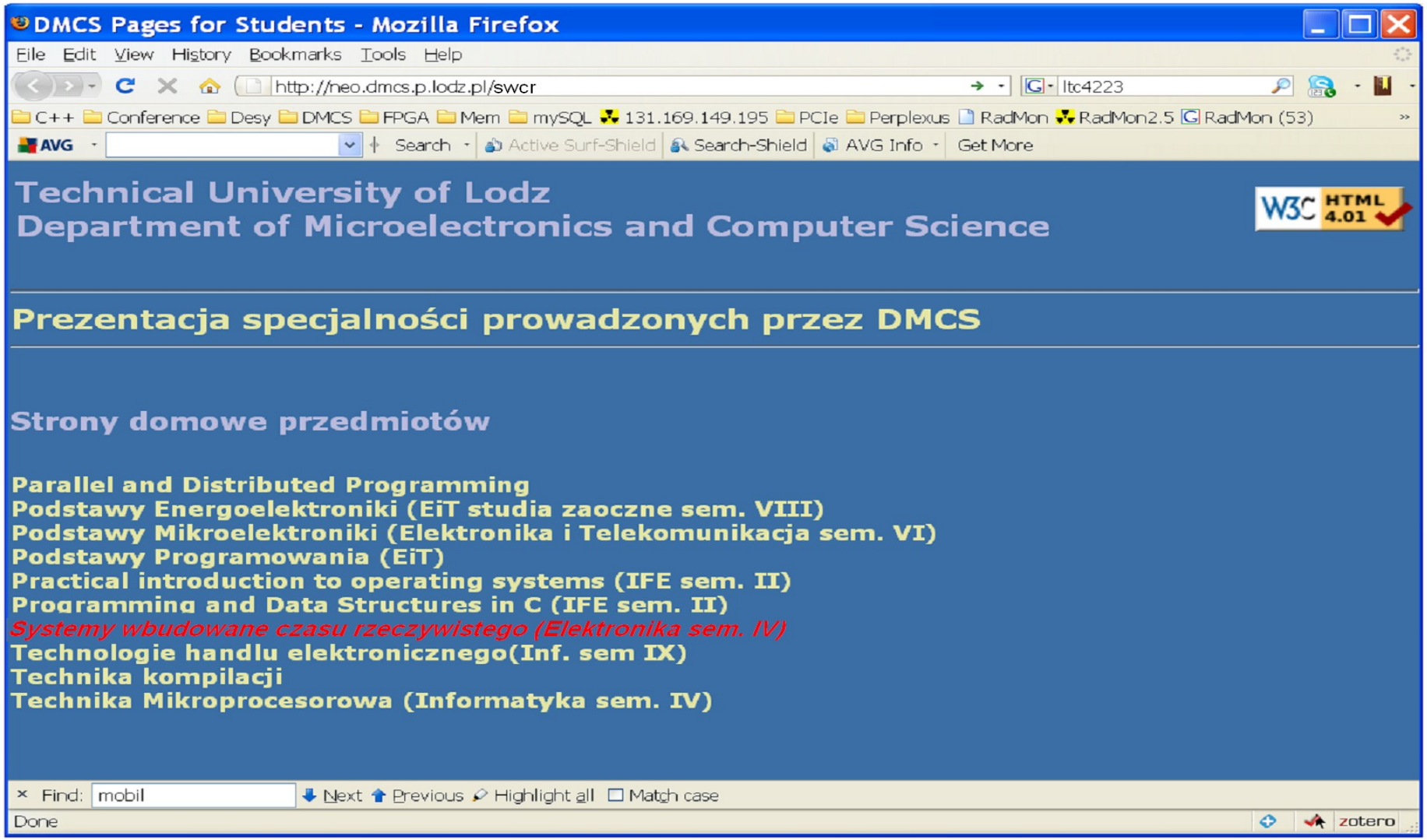
- Chip ID: 0x019607A0
- JTAG ID: 0x05B0C03F
- ARM926 TAP ID: 0x0792603F

Mikrokontroler AT91SAM9263 (2)

Interfejsy komunikacyjne AT91SAM9263:

- 9-warstwowa magistrala EBI (External Bus Interface, 41,6 Gbps),
- Sterownik interfejsu I2S,
- Sterownik USB (host + device, 12 Mbps),
- Sterownik sieci komputerowej Ethernet 10/100 Mbit/s,
- Sterownik magistrali CAN (Controlled Area Network, 1 Mbps),
- Sterownik USART (Universal Serial Asynchronous Receiver-Transmitter, 4 kanały),
- Sterownik interfejsu SPI (Serial Peripheral Interfaces, 50 Mbps),
- Sterownik pamięci CompactFlash oraz MMC/SD ,SDIO (MCI),
- Sterownik interfejsu TWI (two-wire interface, GPRS modem, Wi-Fi, ...).

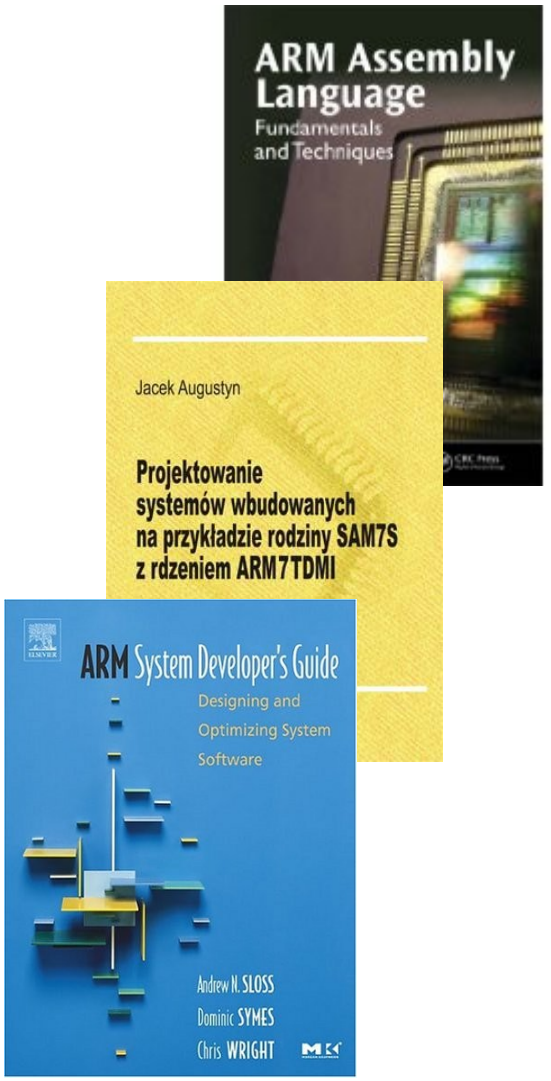
Strona przedmiotu – materiały do wykładu



Literatura

Literatura obowiązkowa:

- Materiały wykładowe i laboratoryjne
- W. Hohl, “ARM Assembly Language: Fundamentals and Techniques” CRC Press 2009, ISBN-10: 1439806101
- J. Augustyn, “Projektowanie systemów wbudowanych na przykładzie rodziny SAM7S z rdzeniem ARM7TDMI”, IGSMiE PAN, 2007, ISBN: 978-83-60195-55-0
- A. Sloss, D. Symes, C. Wright, „ARM System Developer's Guide: Designing and Optimizing System Software”, Elsevier, 2004



Literatura uzupełniająca:

- S. R. Ball, “Embedded Microprocessor Systems: Real World Design”, Elsevier Science, 2002

Zakres przedmiotu

- Laboratorium
- Systemy mikroprocesorowe, systemy wbudowane
- Rodzina procesorów ARM
- Urządzenia peryferyjne
- Pamięci i dekodery adresowe
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych
- Interfejsy w systemach wbudowanych
- Systemy czasu rzeczywistego

Definicje podstawowe (1)

- **Procesor (ang. Processor, Central Processing Unit)**

  Urządzenie cyfrowe, sekwencyjne, potrafiące pobierać dane z pamięci, interpretować je i wykonywać jako rozkazy

- **Mikroprocesor (ang. Microprocessor)**

  Układ cyfrowy wykonany jako pojedynczy układ scalony o wielkim stopniu integracji (VLSI) zdolny do wykonywania operacji cyfrowych według dostarczonych mu informacji, np.: x86, Z80, 68k

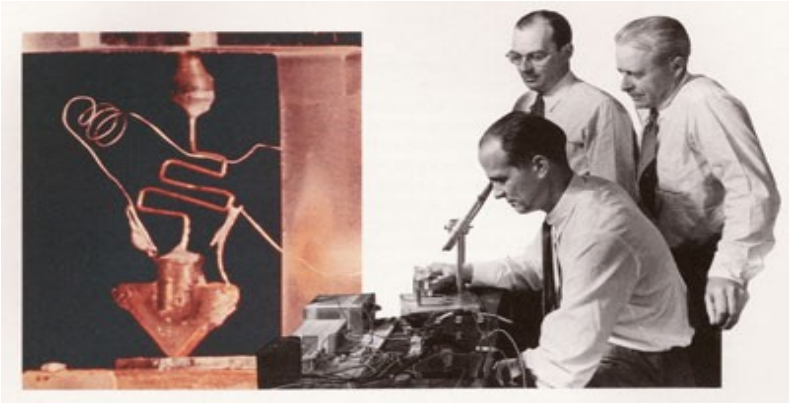
- **Mikrokontroler (ang. Microcontroller)**

  Komputer wykonany w jednym układzie scalonym, używany do sterowania urządzeniami elektronicznymi. Oprócz jednostki centralnej CPU posiada zintegrowane pamięci oraz urządzenia peryferyjne, np.: Intel 80C51, Atmel Atmega128, Freescale MCF5282, ARM926EJ-S

Historia mikroprocesorów (1)

1940 – Russell Ohl – demonstracja złącza półprzewodnikowego (dioda germanowa, bateria słoneczna)

1947 – Shockley, Bardeen, Brattain prezentują pierwszy tranzystor



Pierwszy tranzystor, Bell Laboratories

Pierwszy układ scalony, TI

1958 – Jack Kilby wynalazł pierwszy układ scalony

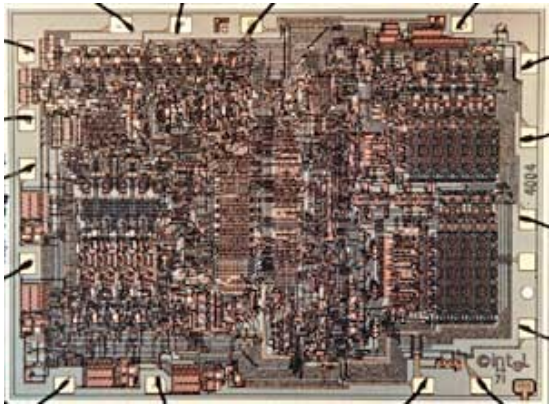
1967 – Laboratorium Fairchild oferuje pierwszą pamięć nieulotną ROM (64 bity)

1969 – Noyce i Moore opuszczają laboratorium Fairchild, powstaje niewielka firma INTEL. INTEL produkuje pamięci SRAM (64 bity). Japońska firma Busicom zamawia 12 różnych układów realizujących funkcje kalkulatorów.

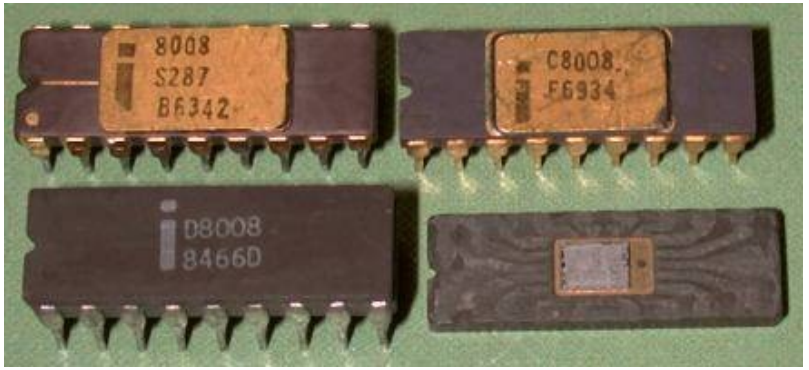
Historia mikroprocesorów (2)

1970 - **F14 CADC** (Central Air Data Computer) mikroprocesor zaprojektowany przez Steve'a Gellera i Raya Holta na potrzeby armii amerykańskiej (myśliwiec F-14 Tomcat)

1971 - **Intel 4004** 4-bitowy procesor realizujące funkcje programowalnego kalkulatora **(**powszechnie uznaje się za pierwszy na świecie mikroprocesor), 3200 tranzystorów. INTEL wznawia pracę nad procesorami, Faggin z Fairchild pomaga rozwiązać problemy.



Zdjęcie 4-bitowego procesora INTEL 4004



8-bitowe procesory INTEL-a

1972 – Faggin rozpoczyna prace nad 8-bitowym procesorem INTEL 8008. Rynek zaczyna się interesować układami “programowalnymi” - procesorami.

Historia mikroprocesorów (3)

1974 – INTEL wprowadza na rynek ulepszona wersję 8008, procesor Intel 8080. Faggin opuszcza firmę Intel i zakłada firmę o nazwie Zilog. Motorola oferuje 8-bitowy procesor 6800 (NMOS, 5 V).

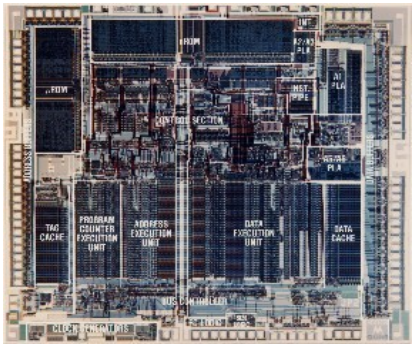
1975 – 8-bitowy procesor INTEL 6502 (technologia MOS) – najtańszy proc. na rynku.

1978 – Pierwszy 16-bitowy procesor 8086 (ulepszony 8080).

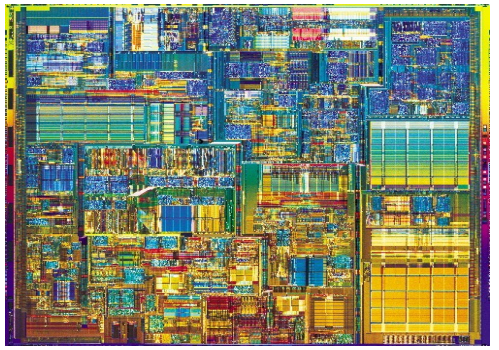
1979 – Motorola oferuje 16-bitowy procesor 68000.

1980 – Motorola wprowadza nowy 32-bitowy procesor 68020, 200,000 tranzystorów.

.....



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, ...

Definicje podstawowe (2)

- **Komputer (ang. Computer)**

  Urządzenie elektroniczne, maszyna cyfrowa zdolna do przetwarzania danych cyfrowych zgodnie z dostarczonym programem

- **Komputer (system) wbudowany (ang. Embedded Computer)**

  Dedykowany system komputerowy, niewielkich rozmiarów sterownik *wbudowany w urządzenie*, przeznaczony do sterowania urządzeniem mechanicznym, elektrycznym lub elektronicznym

- **Komputer osobisty (ang. Personal Computer)**

  System komputery przeznaczony do użytku osobistego, domowego lub biurowego. Komputer wyposażony w system operacyjny przeznaczony do wykonywania *aplikacji wykorzystywanych przez użytkownika*

- **Architektura komputera (ang. Computer Architecture)**

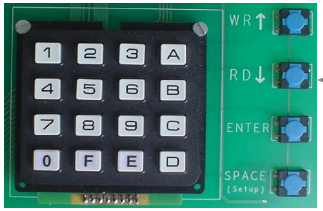
  - Sposób organizacji oraz współpracy podstawowych elementów systemu komputerowego, tj. procesora, pamięci oraz urządzeń peryferyjnych
  - Opis komputera z punktu widzenia programisty w języku niskiego poziomu (asembler). Budowa procesora, potoku wykonawczego oraz model programowy procesora

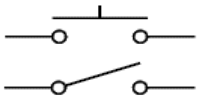
Komputer wbudowany (embedded computer)

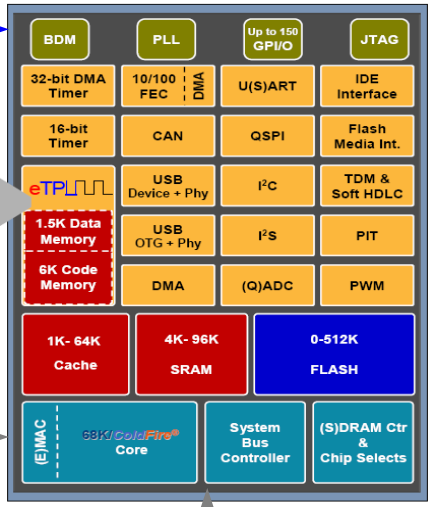
Komunikacja z komputerem zewnętrznym

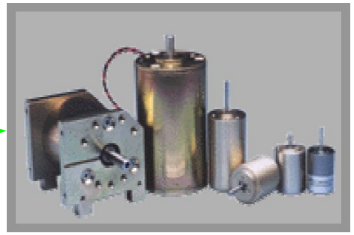
Czujniki, np. czujnik temperatury, obrotów, itd

Kamera



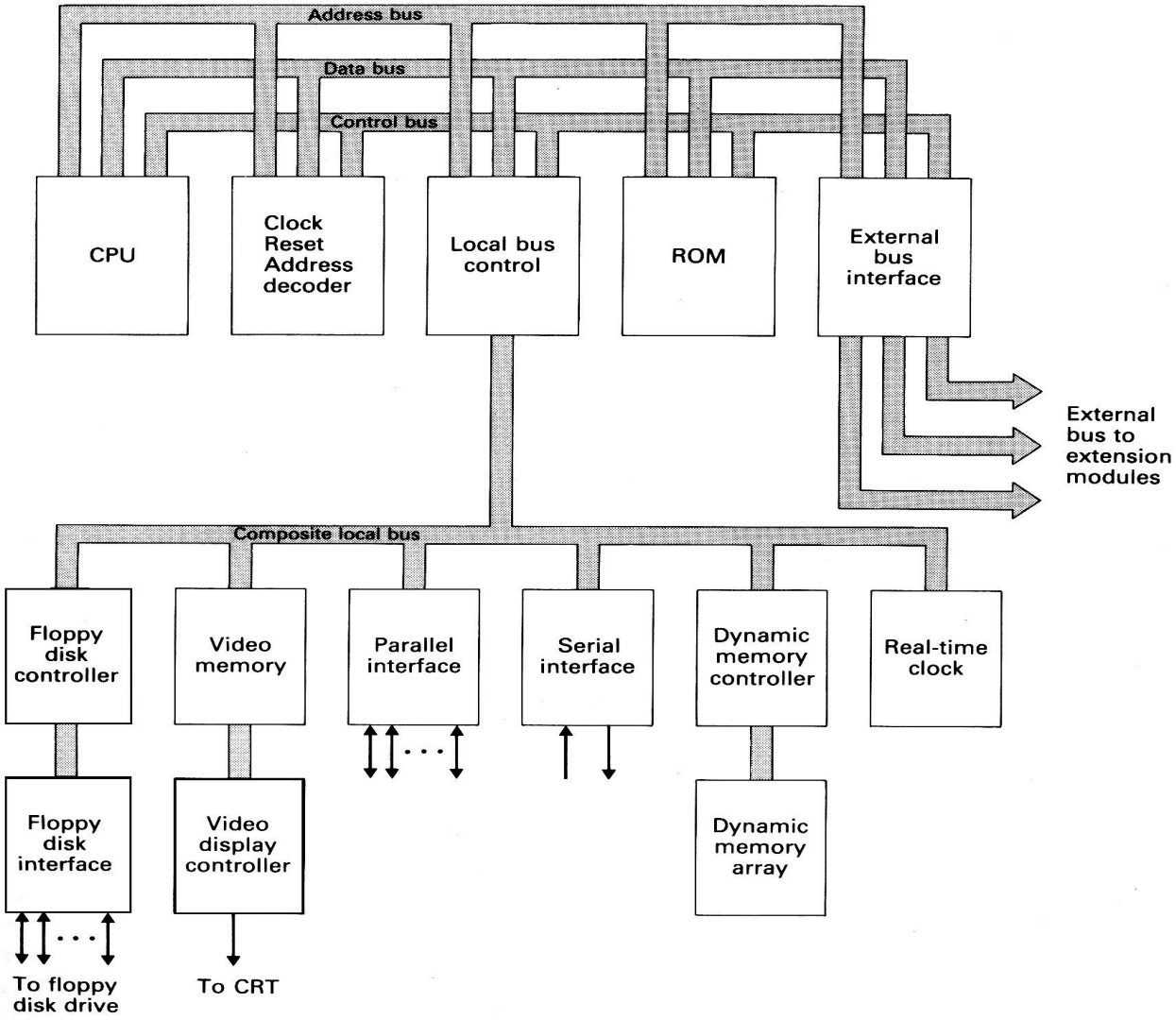




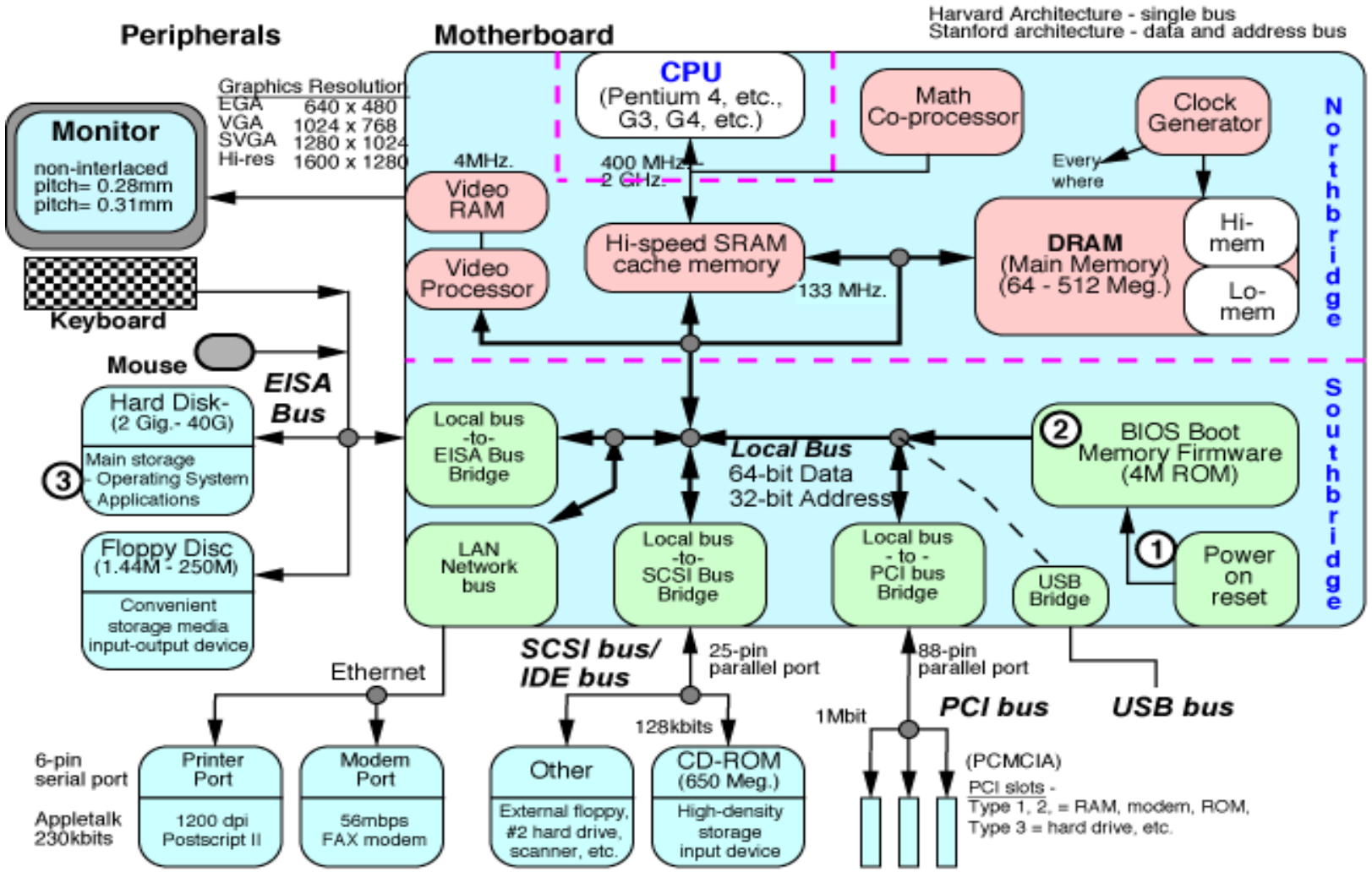


Elementy wykonawcze, np. silniki, przekaźniki

Komputer uniwersalny



Schemat blokowy komputera osobistego



Definicje podstawowe (3)

- **Pamięć komputerowa (ang. Computer Memory)**
  Pamięć komputerowa to urządzenie elektroniczne lub mechaniczne służące do przechowywania danych i programów (systemu operacyjnego oraz aplikacji).
- **Urządzenia zewnętrzne, peryferyjne (ang. Peripheral Device)**
  Urządzenia elektroniczne dołączone do procesora przez magistrale systemową lub interfejs. Urządzenia zewnętrzne wykorzystywane są do realizowania specjalizowanej funkcjonalności systemu.
- **Magistrala (ang. bus)**
  Połączenie elektryczne umożliwiające przesyłanie danym pomiędzy procesorem, pamięcią i urządzeniami peryferyjnymi. Magistra systemowa zbudowane jest zwykłe z kilkudziesięciu połączeń elektrycznych (ang. Parallel Bus) lub szeregowego połączenia (ang. Serial Bus).
- **Interface (ang. Interface)**
  Urządzenie elektroniczne lub optyczne pozwalające na komunikację między dwoma innymi urządzeniami, których bezpośrednio nie da się ze sobą połączyć.

Definicje podstawowe (4)

- **Komputer SoC (ang. System-on-Chip)**

  Układ scalony wielkiej integracji zawierający ***kompletny system elektroniczny zintegrowany z układami analogowymi, cyfrowo-analogowymi oraz radiowymi.*** Poszczególne moduły tego systemu, ze względu na ich złożoność, pochodzą zwykle od różnych dostawców, np. rdzeń procesora od jednego producenta, układy peryferyjne od innego, interfejsy od jeszcze innego, itd...

  Typowym obszarem zastosowań SoC są systemy wbudowane, a najbardziej rozpowszechnionym przedstawicielem tego rozwiązania są systemy oparte na procesorach ARM.

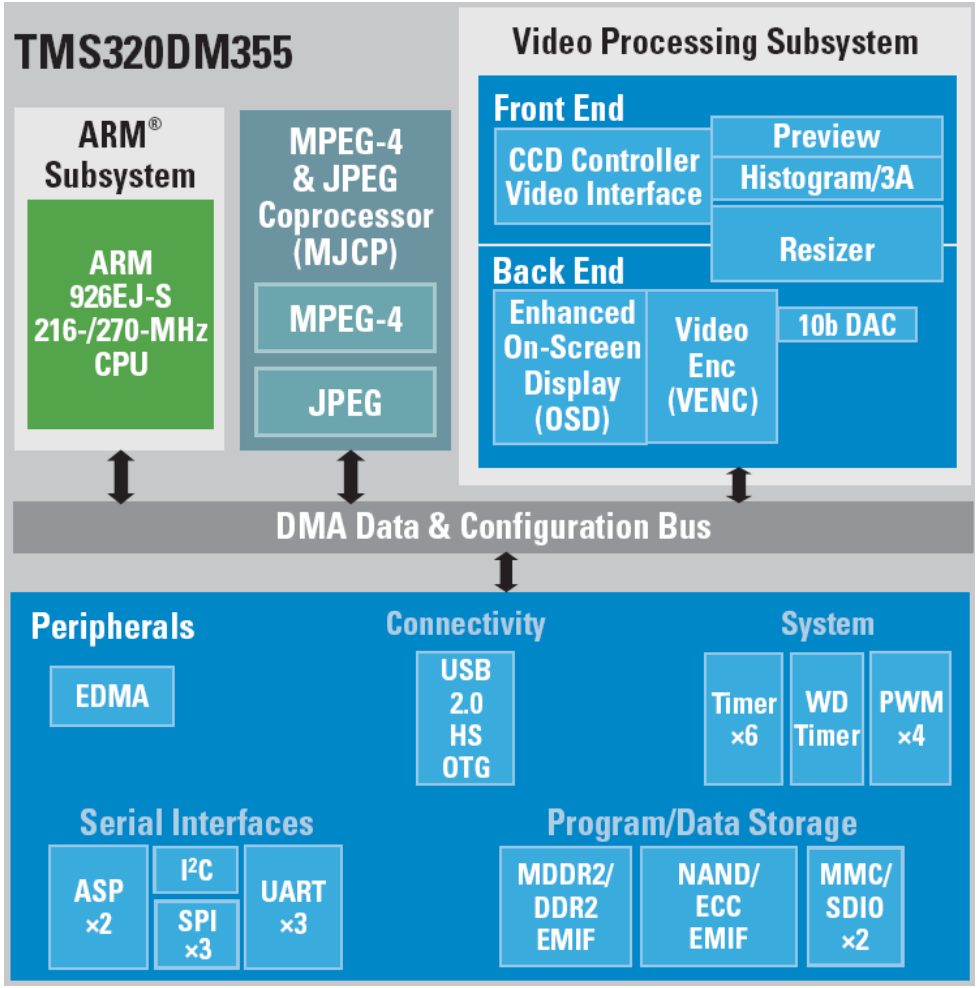
  W przypadku, gdy nie jest możliwa integracja wszystkich układów na jednym podłożu półprzewodnikowym, poszczególne moduły wykonuje się na osobnych kryształach, a całość zamyka się w jednej obudowie, SiP (ang. System-in-a-package).

  SoC różnią się od mikrokontrolerów ***znacznie wydajniejszą jednostka obliczeniową CPU*** (pozwalającej uruchamiać systemy operacyjne, np. Linux, Windows) oraz są zwykłe ***wyposażone w specjalizowane układy peryferyjne.***

SoC - DaVinci, digital media processor

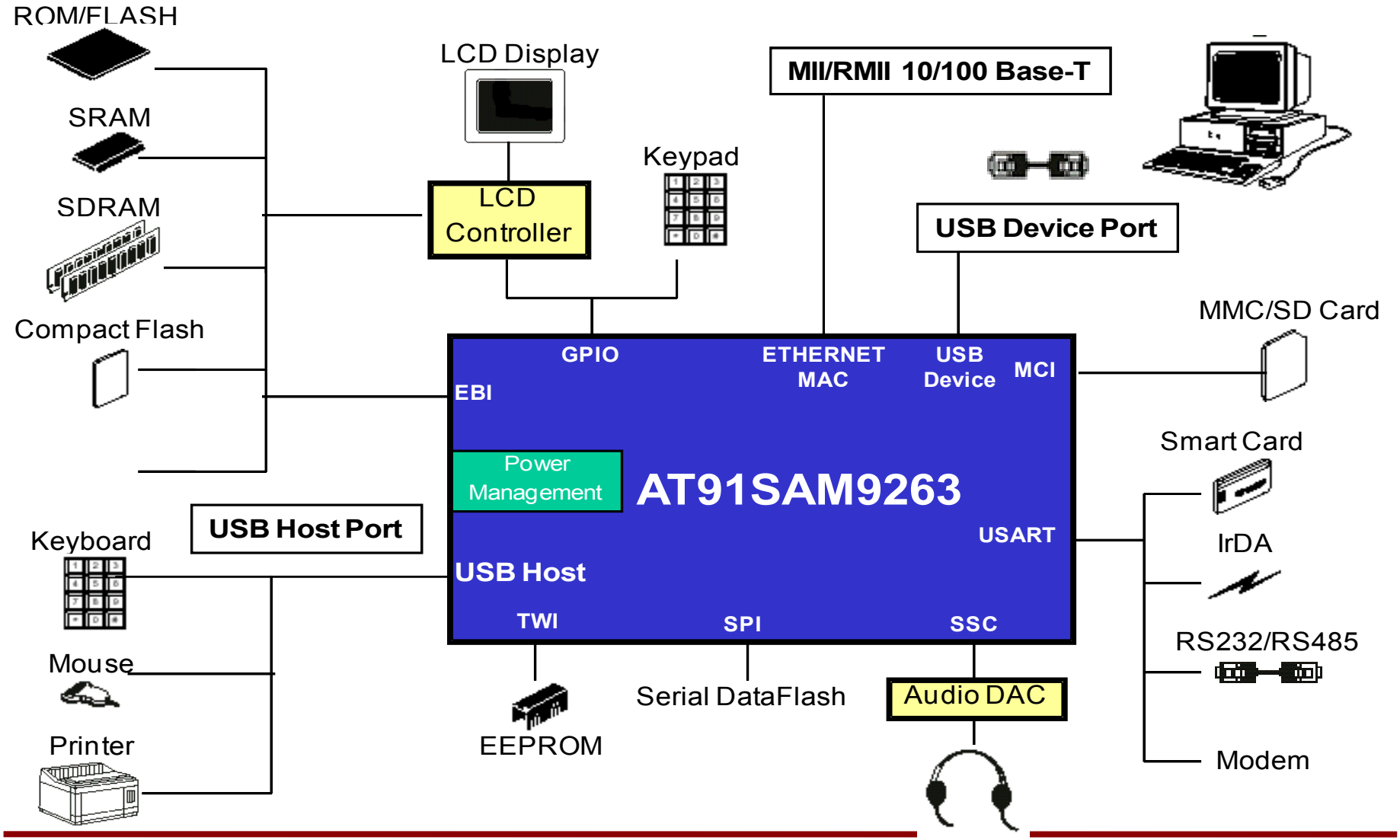
DaVinci DM355

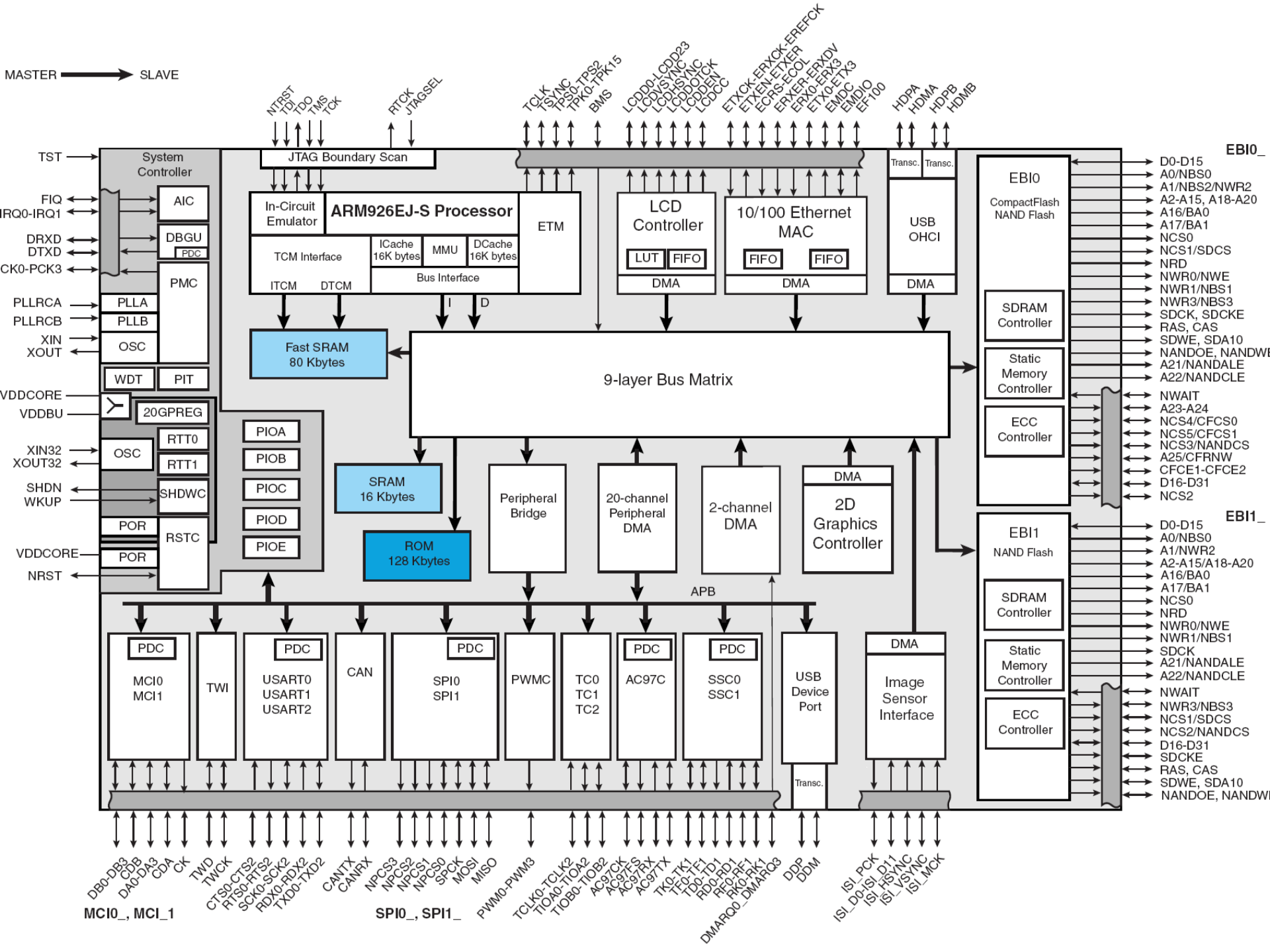
- SoC opracowany przez firmę Texas Instruments
- Dedykowany co-procesor do przetwarzania dźwięku i obrazu w czasie rzeczywistym
- Niski pobór energii 400 mW podczas dekodowania HD MPEG4, 1 mW w stanie czuwania (systemy przenośne)
- Bogate interfejsy i układy peryferyjne (sterownik HDD, SD/MMC, USB, Ethernet,...)



Źródło: www.ti.com

Mikrokontroler 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
DRXD
DTXD
CK0-PCK3
AIC
DBGU
PDC
PMC
PLLRCA
PLLRCB
XIN
XOUT
PLLA
PLLB
OSC
WDT
PIT
VDDCORE
VDDBU
20GPREG
RTT0
RTT1
XIN32
XOUT32
OSC
SHDN
WKUP
SHDWC
POR
RSTC
VDDCORE
POR
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
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 – zestaw narzędzi programistycznych ddla procesorów ARM dostępnych na zasadach licencji GNU GPL (General Public License).

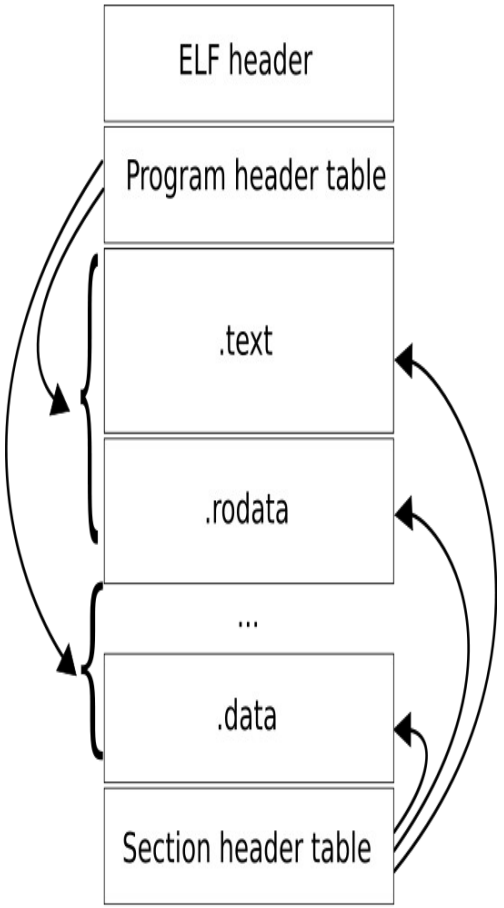
Dostępne narzędzia (***http://www.gnuarm.org/***):

- 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 plików wykonywalnych, relokowalnych i bibliotek dynamicznych opracowany na potrzeby systemów operacyjnych bazujących na systemie Unix. COFF miał zastąpić standard plików **a.out**. Wykorzystywany na różnych systemach, również Windows. Obecnie standard COFF wypierany jest przez pliki ELF.

- **ELF (Executable and Linkable Format) –** standard plików wykonywalnych, relokowalnych, bibliotek dynamicznych i zrzutów pamięci wykorzystywany na różnych komputerach i systemach operacyjnych, np.: rodzina x86, PowerPC, OpenVMS, BeOS, konsole PlayStation Portable, PlayStation 2, PlayStation 3, Wii, Nintendo DS, GP2X, AmigaOS 4 oraz Symbian OS v9.
- **Przydatne narzędzia:**
  - readelf
  - elfdump
  - objdump



Źródło: wikipedia

Debuger GDB

arm-elf-gdb <nazwa pliku.elf>

run – uruchomienie programu (załadowanie i uruchomienie), load – ładuje program

c (continue) – kontynuacja wykonywania programu

b (breakpoint) – ustawienie pułapki, np. b 54, b main, b sleep

n (next) – wykonanie funkcji (bez zagłębiania się do jej wnętrza)

s (step) – wykonanie funkcji, zatrzymuje się wewnątrz funkcji

d (display) – wyświetlenie zmiennej/rejestru, disp Counter

p (print) – wyświetlenie (jednorazowe) zmiennej/rejestru

x – wyświetlenie obszaru pamięci **x/10x 0xFFFF.F000**

i (info) – wyświetla informacje o pułapkach, rejestrach, etc...

Modyfikatory:

/x – wyświetla w postaci szesnastkowej

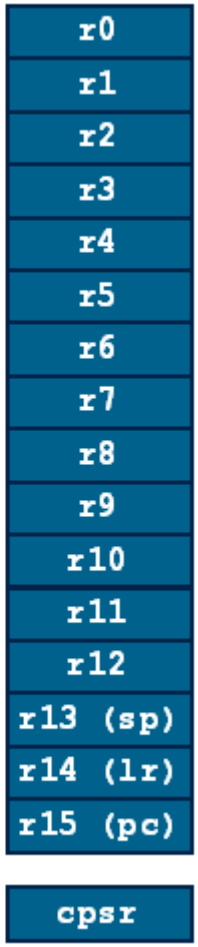
/t - wyświetla w postaci binarnej

/d - wyświetla w postaci dziesiętnej

Rejestry procesora a GDB

(gdb) info r

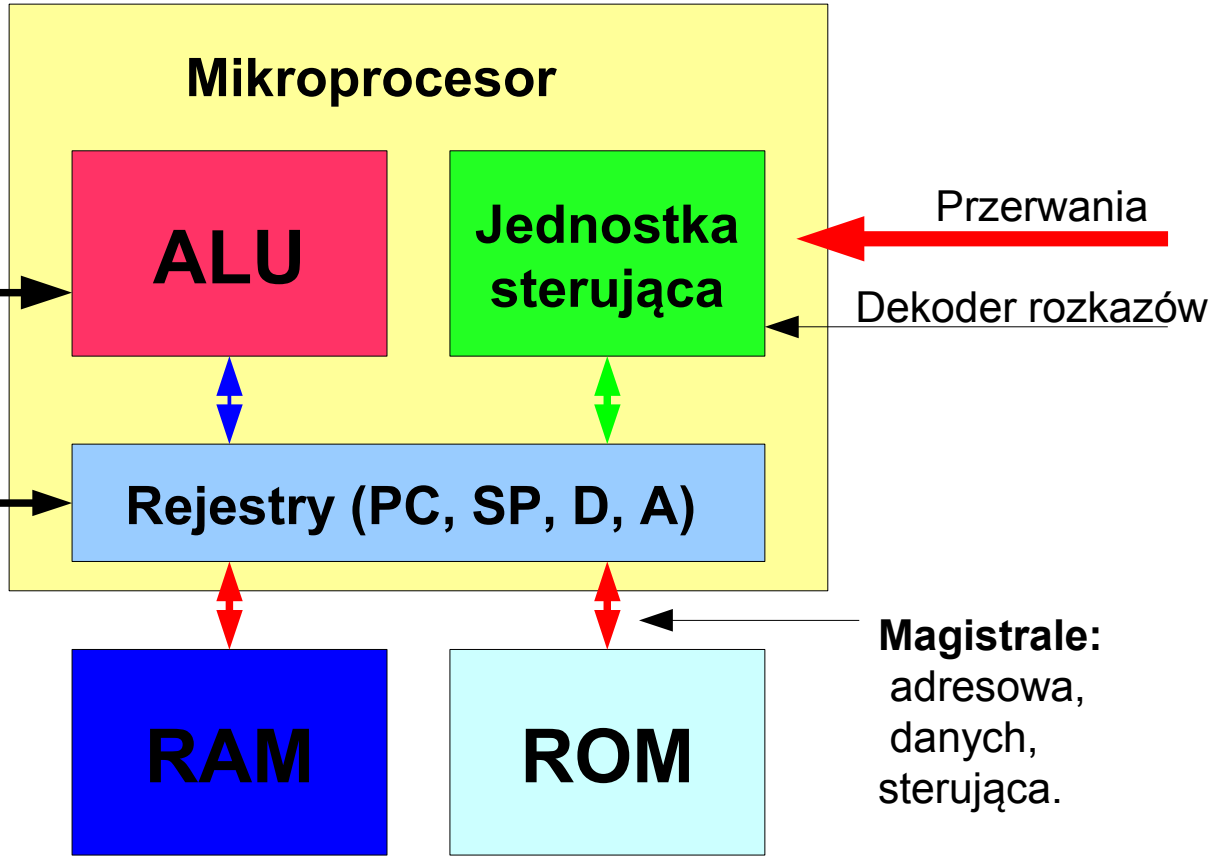
r0	0x2	0x2
r1	0x20000ba4	0x20000ba4
r2	0x57b	0x57b
r3	0x270f	0x270f
r4	0x300069	0x300069
r5	0x3122dc	0x3122dc
r6	0x1000	0x1000
r7	0x800bc004	0x800bc004
r8	0x3122c4	0x3122c4
r9	0x407c81a4	0x407c81a4
r10	0x441029ab	0x441029ab
r11	0x313f2c	0x313f2c
r12	0x313f30	0x313f30
sp	0x313f18	0x313f18
lr	0x20000a7c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



Mikroprocesor

Mikroprocesor to układ cyfrowy wykonany jako pojedynczy układ scalony o wielkim stopniu integracji zdolny do wykonywania operacji cyfrowych według dostarczonych mu instrukcji.

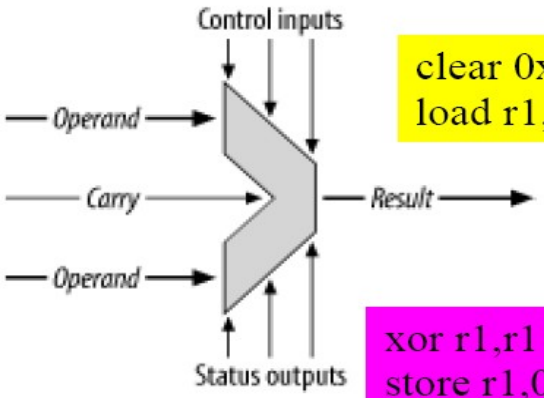
Jednostka arytmetyczno-logiczna (ang. Arithmetic Logic Unit), realizuje podstawowe operacje matematyczne 8, 16, 32, 64-bit



Rejestry procesora - obszar (plik) rejestrów (ang. registers file) - komórki szybkiej pamięci statycznej, umieszczonej, wewnątrz procesora, 8, 16, 32, 64, 128-bit,

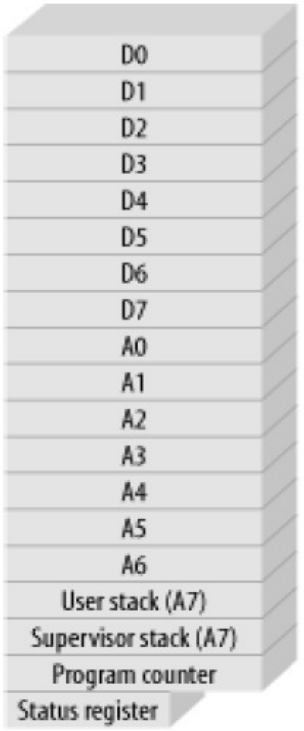
Architektura procesora (2)

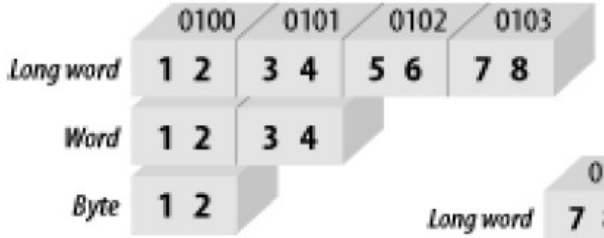
- ALU
  - lista operacji
- zestaw rejestrów
  - A, I, PC, SR, SP,...
- lista rozkazów
  - CISC, RISC
- tryby adresowania
  - R, A, I,...
- przerwania
  - hardware, software
- big/little endian

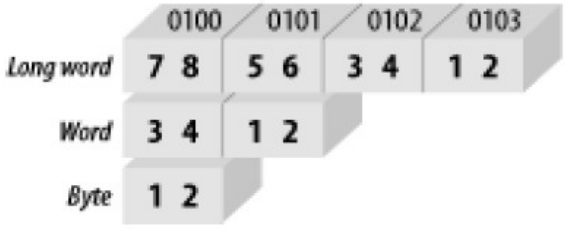



```
clear 0x1000
load r1,#5
```

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







Architektura procesora CISC

Cechy architektury CISC (Complex Instruction Set Computers):

- Duża liczba rozkazów (instrukcji),
- Niektóre rozkazy potrzebują dużej liczby cykli procesora do wykonania,
- Występowanie złożonych, specjalistycznych rozkazów,
- Duża liczba trybów adresowania,
- Do pamięci może się odwoływać bezpośrednio duża liczba rozkazów,
- Mniejsza od układów RISC częstotliwość taktowania procesora,
- Powolne działanie dekodera rozkazów, ze względu na dużą ich liczbę i skomplikowane adresowanie

Przykłady rodzin procesorów o architekturze CISC to:

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

Architektura procesora RISC

Cechy architektury RISC (Reduced Instruction Set Computer):

- Zredukowana liczba rozkazów. Upraszcza to znacznie dekoder rozkazów.
- Redukcja trybów adresowania, dzięki czemu kody rozkazów są prostsze,
- Ograniczenie komunikacji pomiędzy pamięcią, a procesorem. Do przesyłania danych pomiędzy pamięcią, a rejestrami służą dedykowane instrukcje (load, store) .
- Zwiększenie liczby rejestrów (np. 32, 192, 256),
- Dzięki przetwarzaniu potokowemu (ang. pipelining) wszystkie rozkazy wykonują się w jednym cyklu maszynowym.

Przykłady rodzin mikroprocesorów o architekturze RISC:

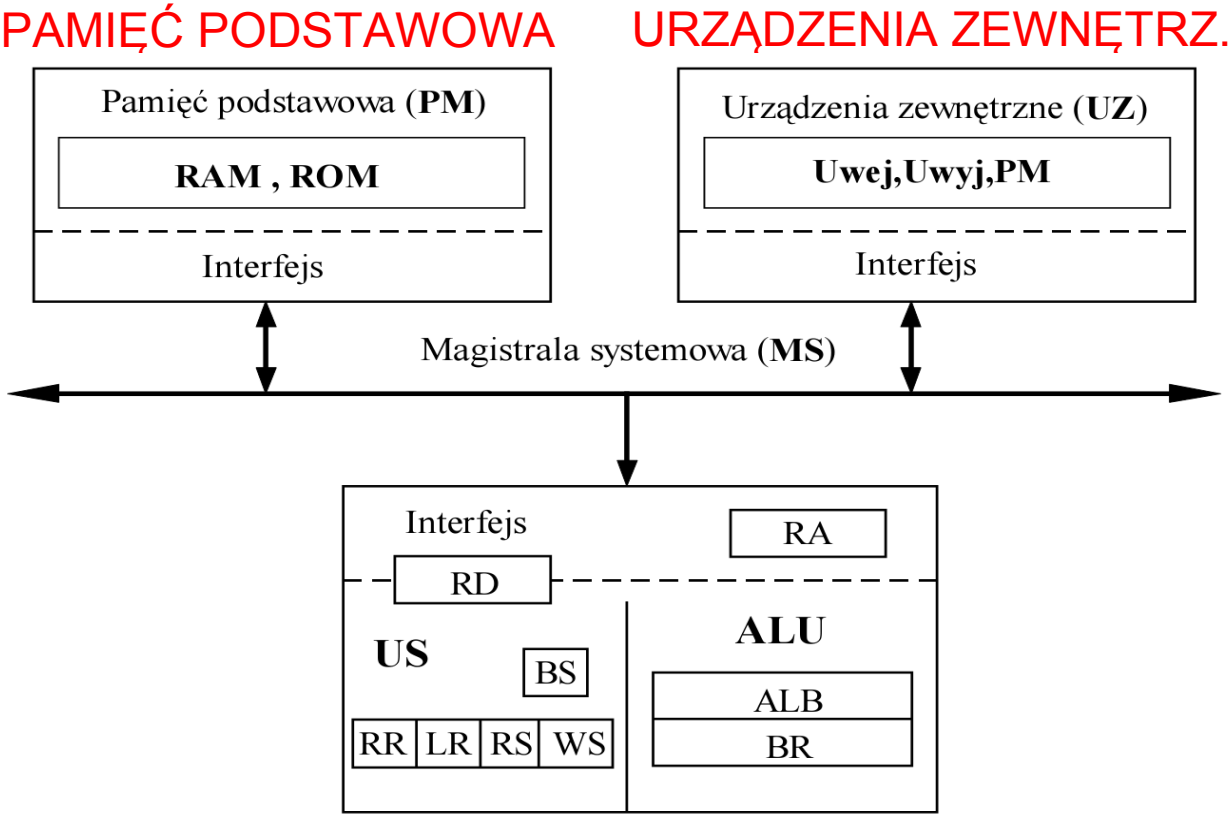
- IBM 801
- PowerPC
- MIPS
- Alpha
- **ARM**
- Motorola 88000
- ColdFire
- SPARC
- PA-RISC
- Atmel_AVR

Obecnie produkowane procesory Intela z punktu widzenia programisty są widziane jako CISC, ale ich rdzeń jest zgodny z RISC. Rozkazy CISC są rozbijane na mikrorozkazy (ang. microops), które są następnie wykonywane przez szybki blok wykonawczy zgodny z architekturą RISC.

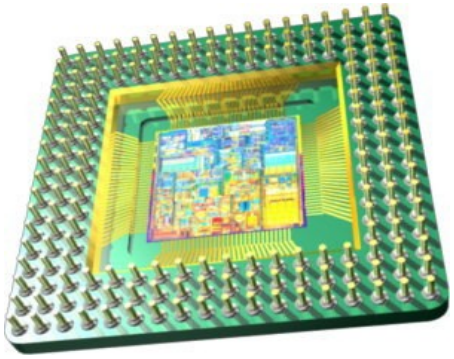
Architektura systemu komputerowego

Architektura polega na ścisłym podziale komputera na trzy podstawowe części:

- procesor,
- pamięć (zawierająca dane oraz program),
- urządzenia wejścia/wyjścia (I/O).



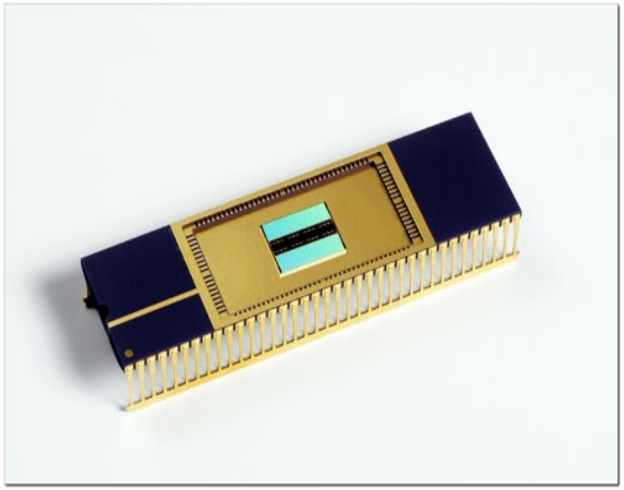
Magistrale komputera



Magistrala adresowa

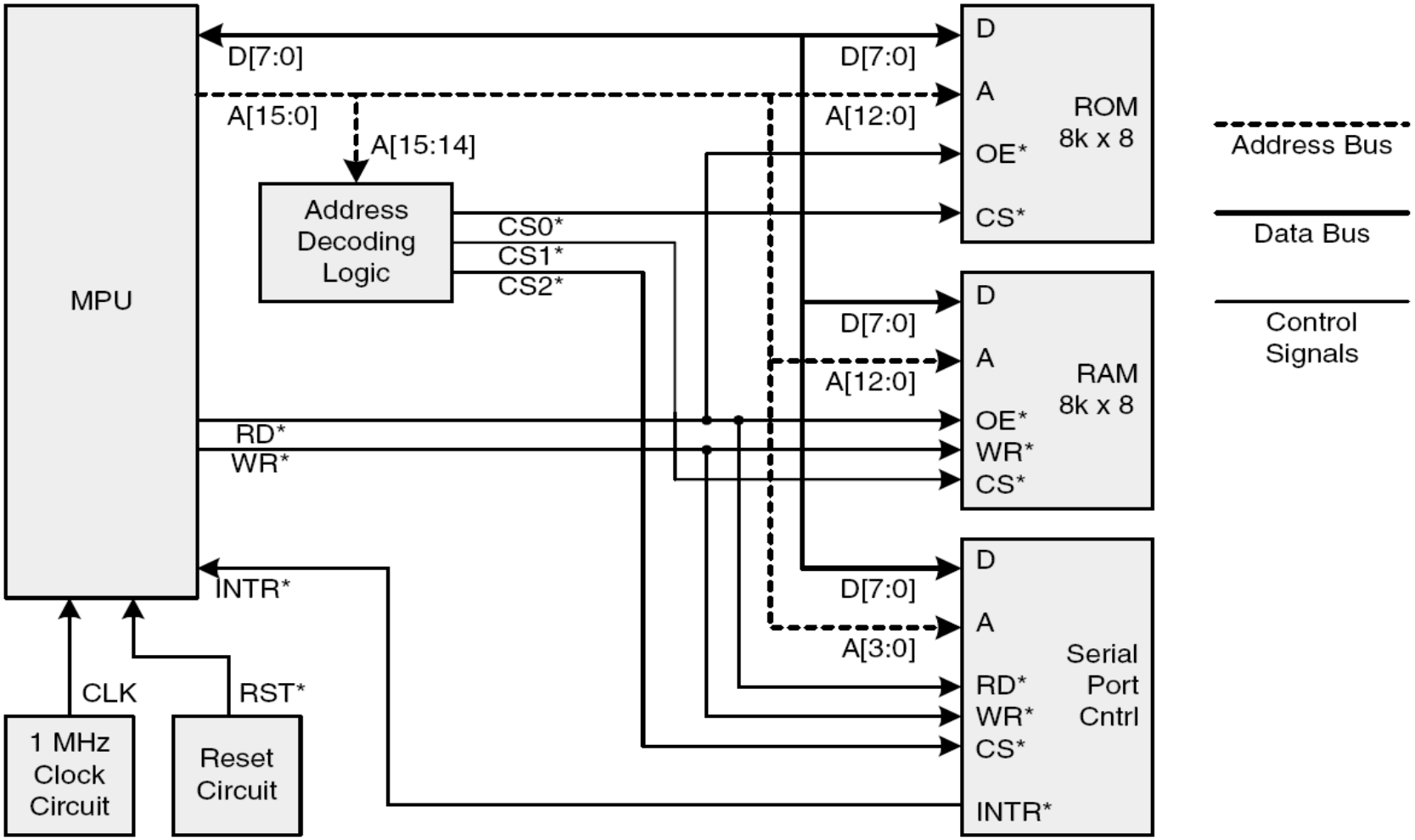
Magistrala danych

Magistrala sterująca



1. Rodzaj magistrali
2. Szerokość magistrali
3. Częstotliwość zegara – szybkość transmisji

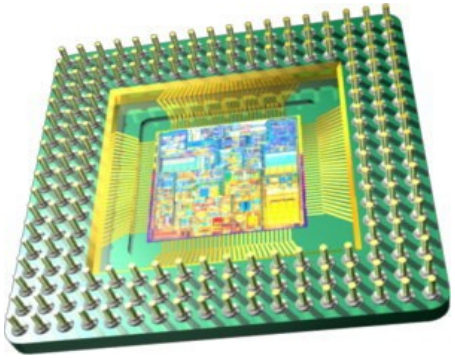
Przykładowy komputer 8-bitowy



Architektura von Neumanna

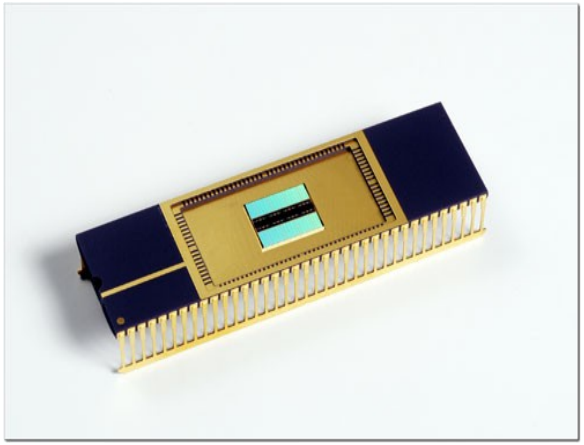
Cechy architektury von Neumanna:

- rozkazy i dane przechowywane są w tej samej pamięci,
- nie da się rozróżnić danych o rozkazów (instrukcji),
- dane nie maja przypisanego znaczenia,
- pamięć traktowana jest jako liniowa tablica komórek, które identyfikowane są przy pomocy dostarczanego przez procesor adresu,
- procesor ma dostęp do przestrzeni adresowej, dekodery adresowe zapewniają mapowanie pamięci na rzeczywiste układy.



Magistrala adresowa

Magistrala danych

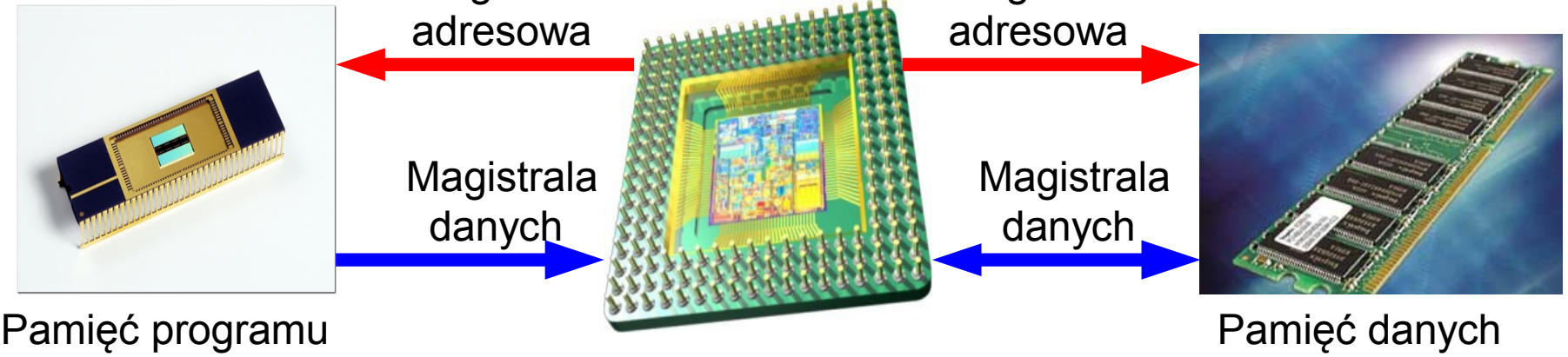


Architektura Harwardzka

Prostsza (w stosunku do architektury Von Neumanna) budowa przekłada się na większą szybkość działania - dlatego ten typ architektury jest często wykorzystywany w procesorach sygnałowych oraz przy dostępie procesora do pamięci cache.

Cechy architektury Harwardzkiej:

- rozkazy i dane przechowywane są w oddzielnych pamięciach,
- organizacja pamięci może być różna (inne długości słowa danych i rozkazów),
- możliwość pracy równoległej – jednoczesny odczyt danych z pamięci programu oraz danych,
- stosowana w mikrokontrolerach jednoukładowych.



Kolejność bajtów w pamięci (1)

Bajt – najmniejsza adresowalna jednostka pamięci komputerowej

Endianess

Big-endian

...pod najmłodszym adresem umieszczony jest najstarszy bajt

podobnie jak w języku polskim, angielskim

Motorola, SPARC, ARM

middle-endian

liczby zmiennoprzecinkowe podwójnej precyzji

VAX and ARM

Little-endian

...pod najstarszym adresem umieszczony jest najstarszy bajt

podobnie jak w językach arabskich, hebrajski

Intel x86, 6502 VAX

Bi-Endian

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

Kolejność bajtów w pamięci (2)

Architektura 8-bitowa

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

Podwójne słowo (DW):
0x1234.5678

7	0
0x0000.0000	0x12
0x0000.0001	0x34
0x0000.0002	0x56
0x0000.0003	0x78
0x0000.0004	0x90

Kolejność bajtów w pamięci (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				

Kolejność bajtów w pamięci (4)

Podwójne słowo (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				

Kolejność bajtów w pamięci (5)

Jak rozpoznać architekturę procesora oraz rozkład bajtów w pamięci?

```
#define LITTLE_ENDIAN   0
#define BIG_ENDIAN      1

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

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

   else
      return BIG_ENDIAN;
}
```

Operacje na rejestrach

Operacje na pamięci w języku C

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

Jak wyzerować pojedynczy bit ?

Jak ustawić pojedynczy bit ?

Operacje na rejestrach w języku C

```
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)
```

Operacje na rejestrach w języku C (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};
```

Operacje na rejestrach w języku C (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);
```

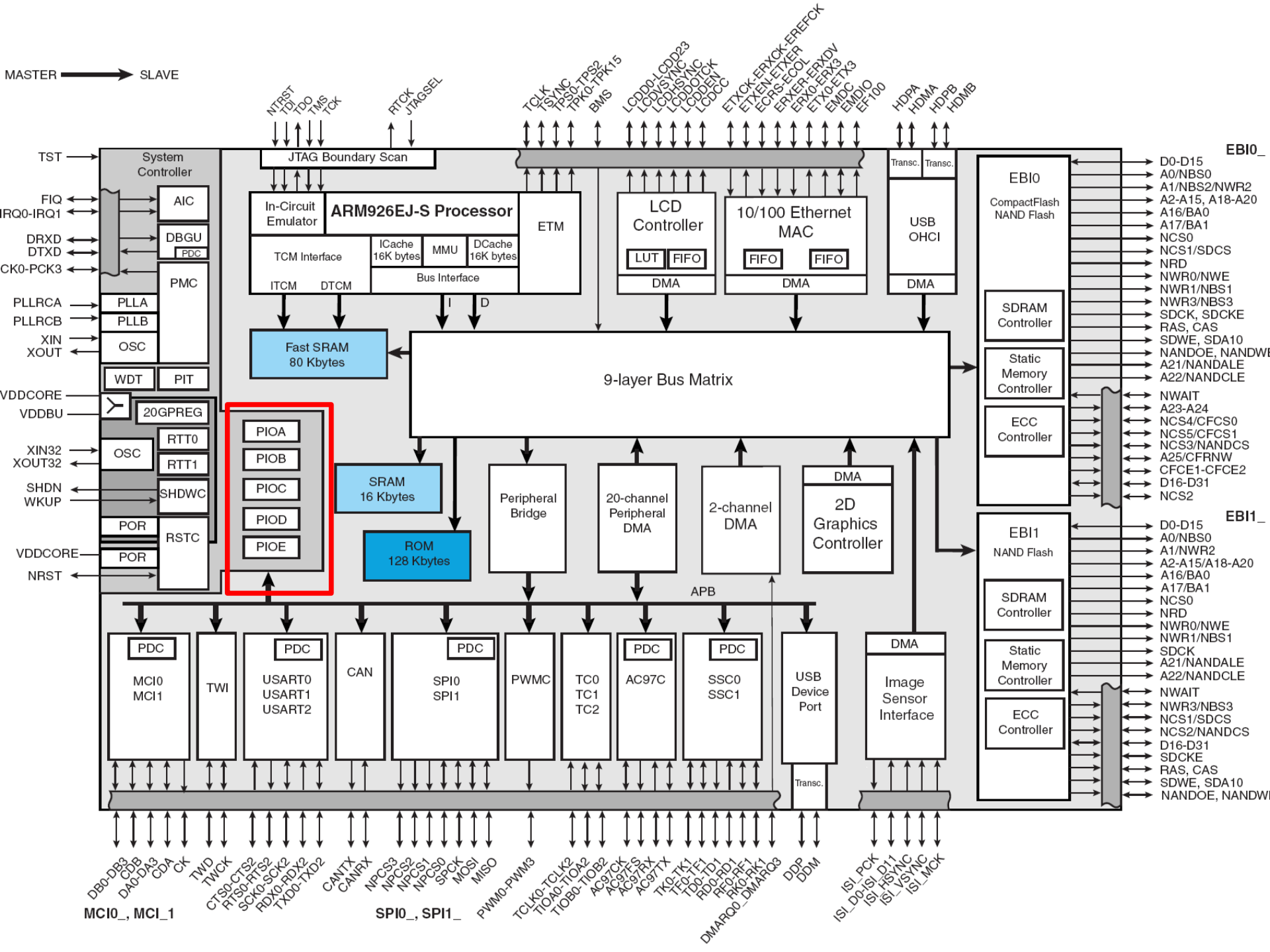
Register Concatenation

```
int main(void) {
        unsigned char reg1=0x15, reg2=0x55;
        unsigned char = reg3, reg4;
        unsigned int tmp;
        /* concatenation operation */
        tmp = reg1;
        tmp = tmp<<8 | reg2;

        /* deconcatenation operation  */
        reg3 = tmp>>8;              /* be careful with signed numbers */
        reg4 = tmp & 0xFF;

}
```

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


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
FIQ
IRQ0-IRQ1
DRXD
DTXD
CK0-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
OSC
RTT0
RTT1
SHDWC
POR
RSTC
POR
PIOA
PIOB
PIOC
PIOD
PIOE
JTAG Boundary Scan
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
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_

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

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:

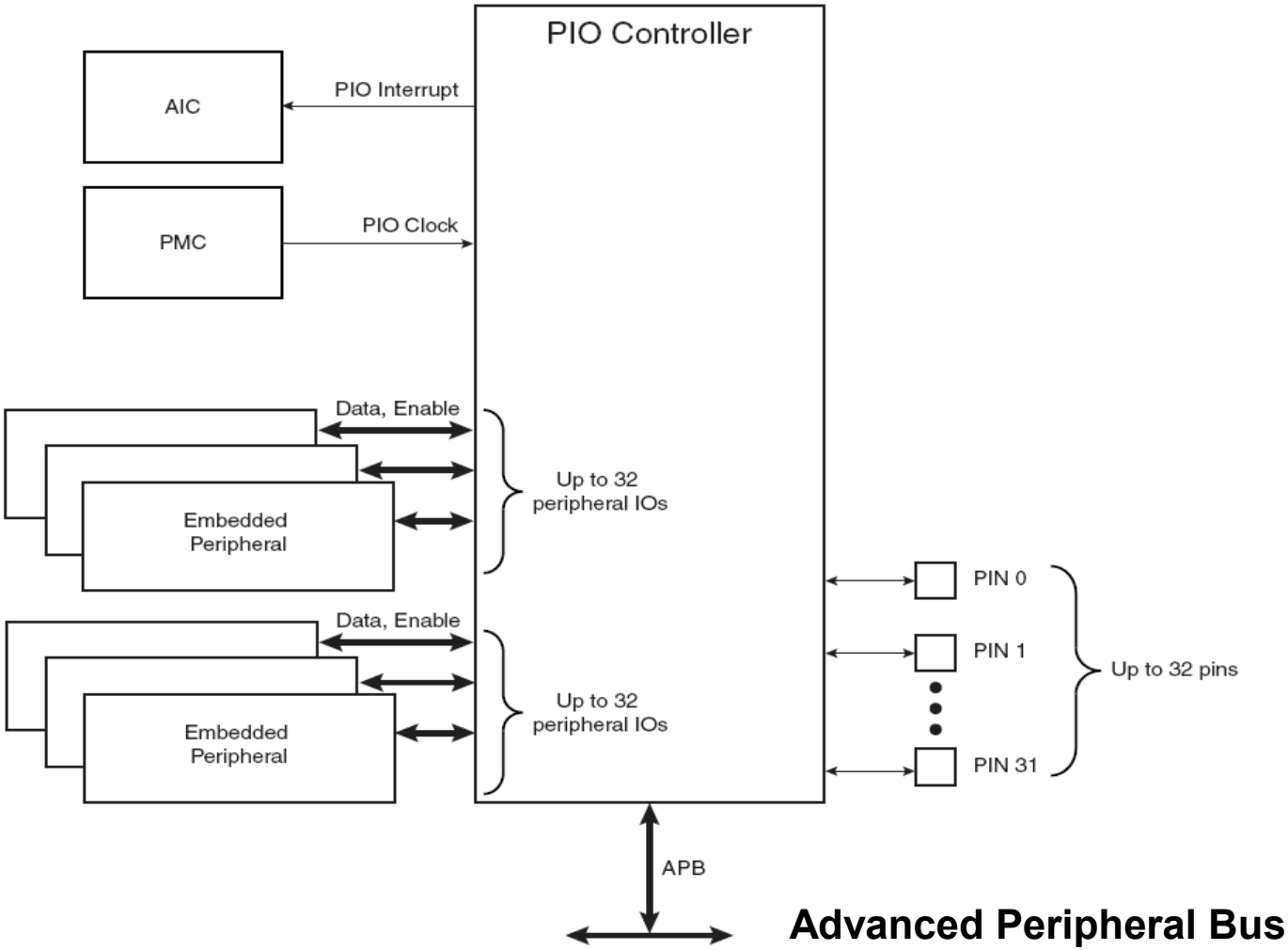
- An input change interrupt enabling level change detection on any I/O line.
- A glitch filter providing rejection of pulses lower than one-half of clock cycle.
- Multi-drive capability similar to an open drain I/O line.
- Control of the the pull-up of the I/O line.
- Input visibility and output control.

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.

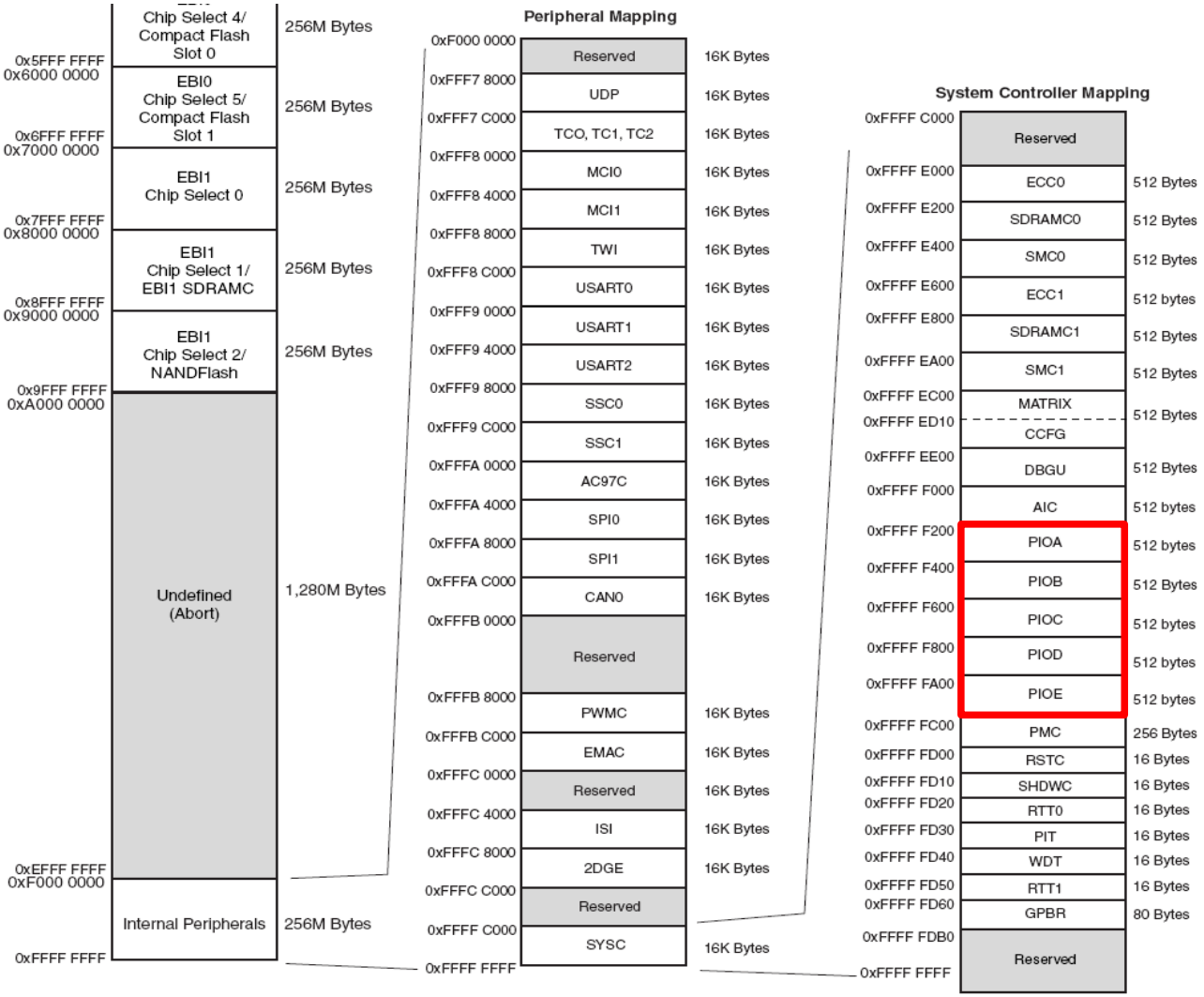
The user must configure the Power Management Controller before any access to the input line information.

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
0xFFFF ED10
CCFG
512 Bytes
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.

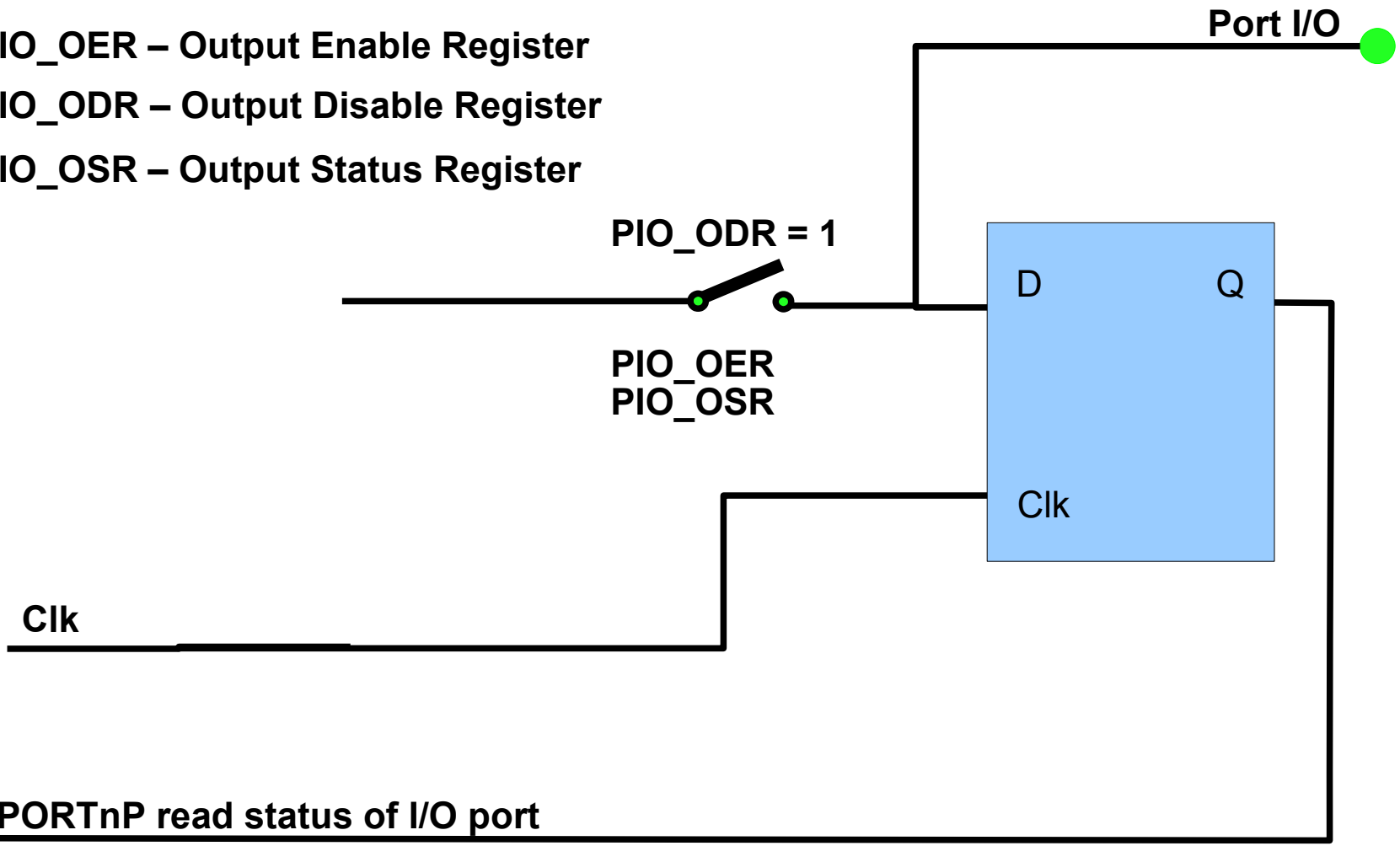
1 = The data to be driven on the I/O line is 1.

Moduł portów I/O (2)

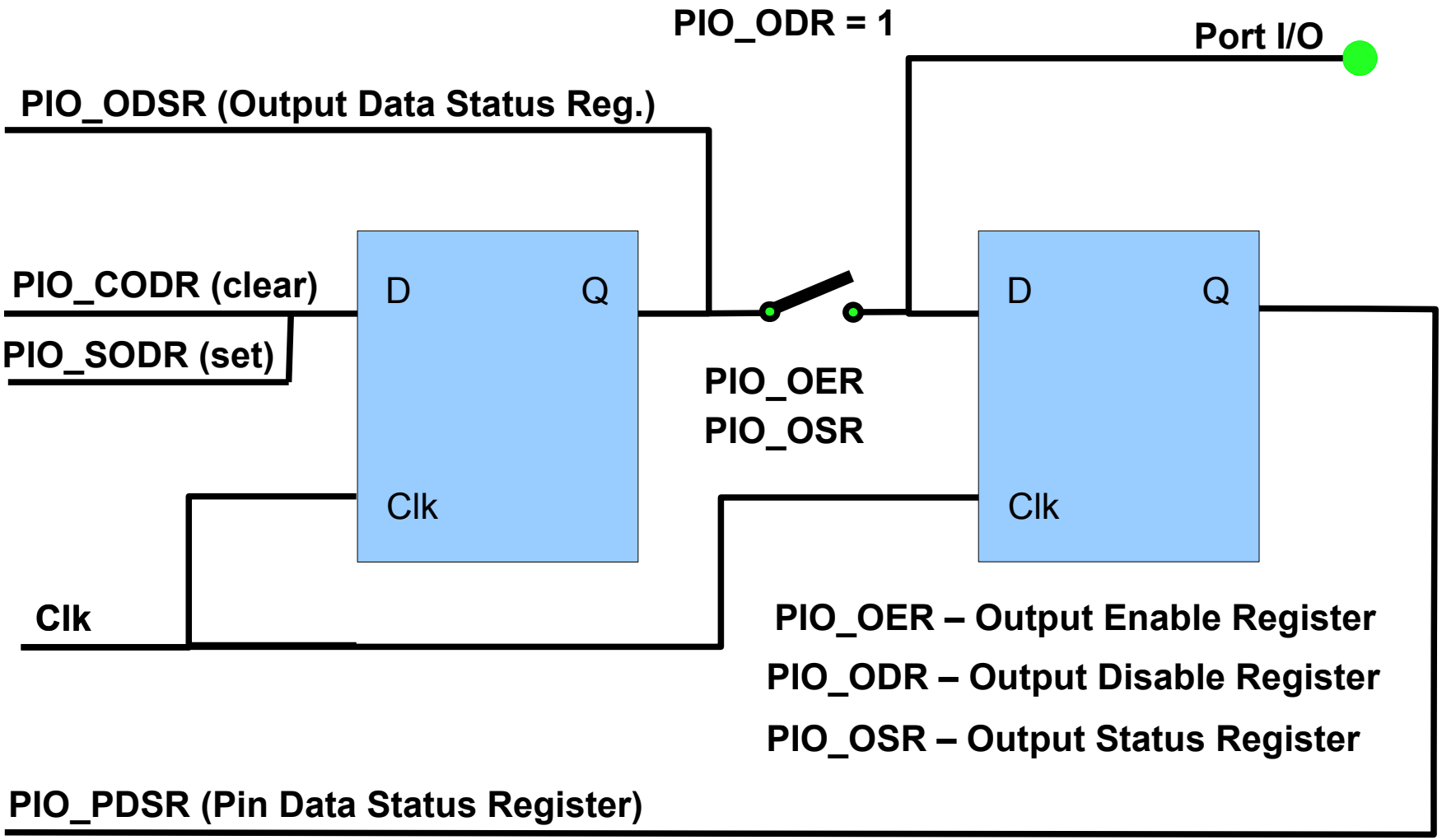
PIO_OER – Output Enable Register

PIO_ODR – Output Disable Register

PIO_OSR – Output Status Register

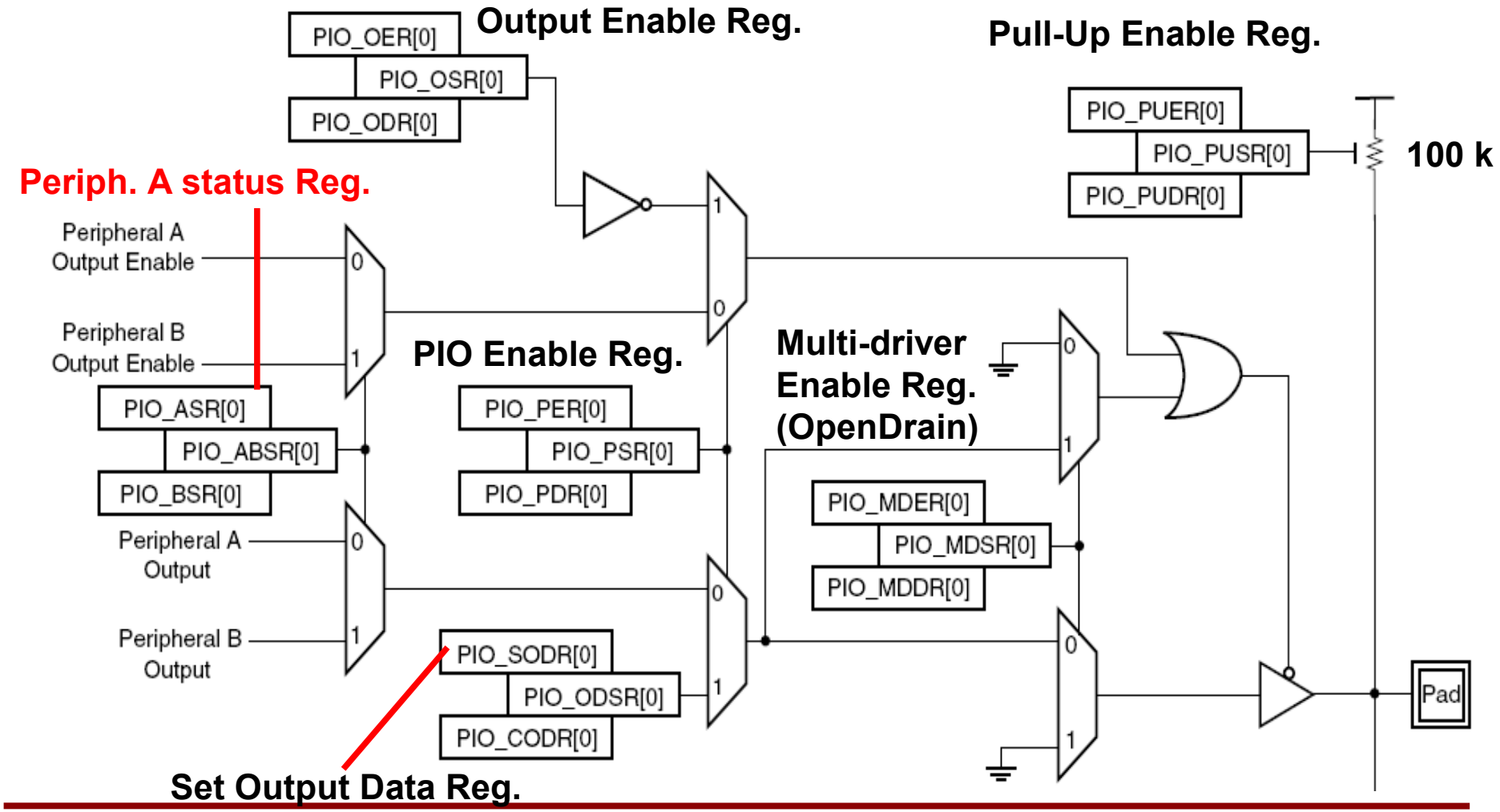


Simplified Block Diagram of I/O Port

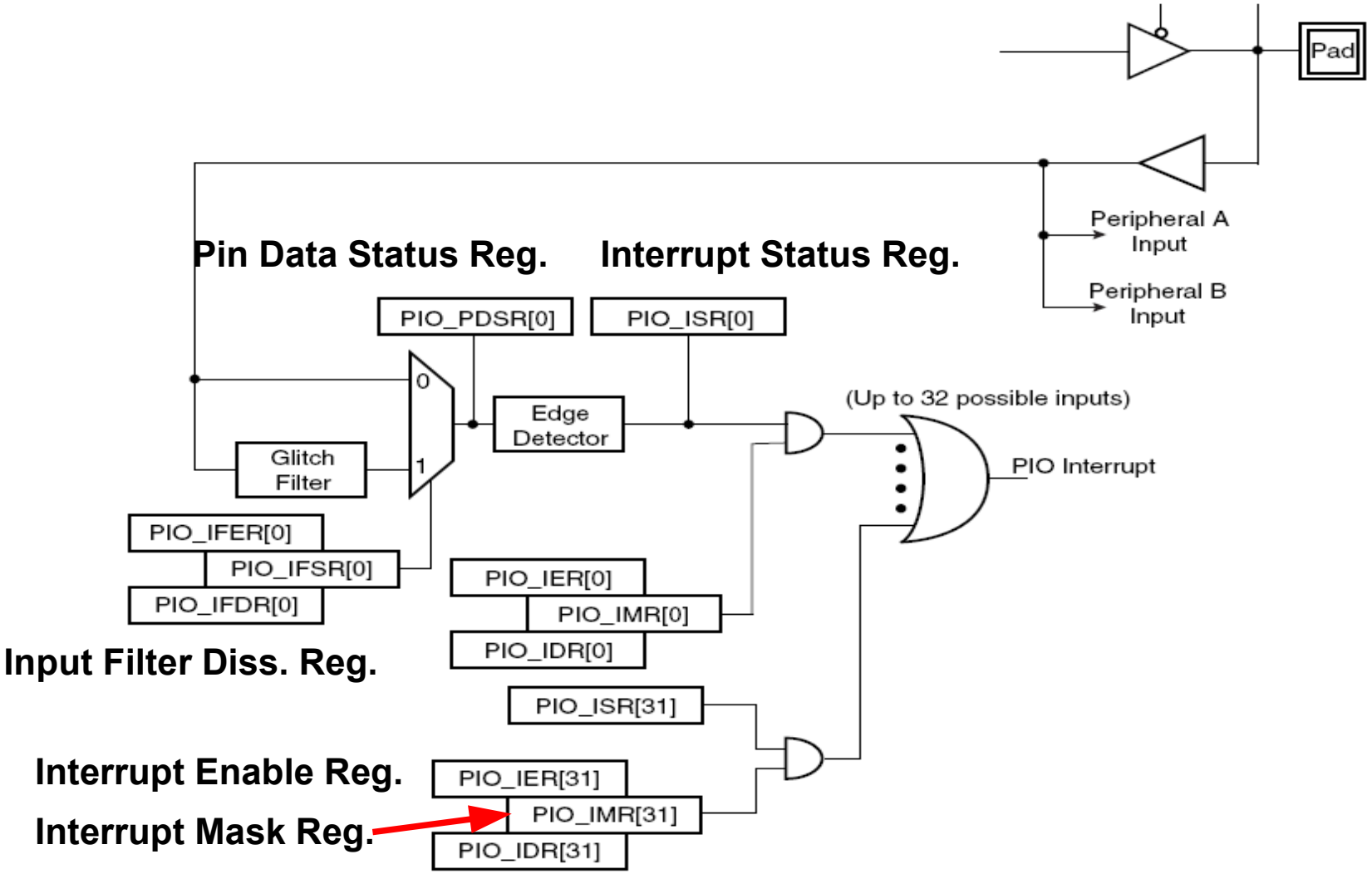


I/O Port – How to Control Output ?

Figure 31-3. I/O Line Control Logic



I/O – How to Read Input ?



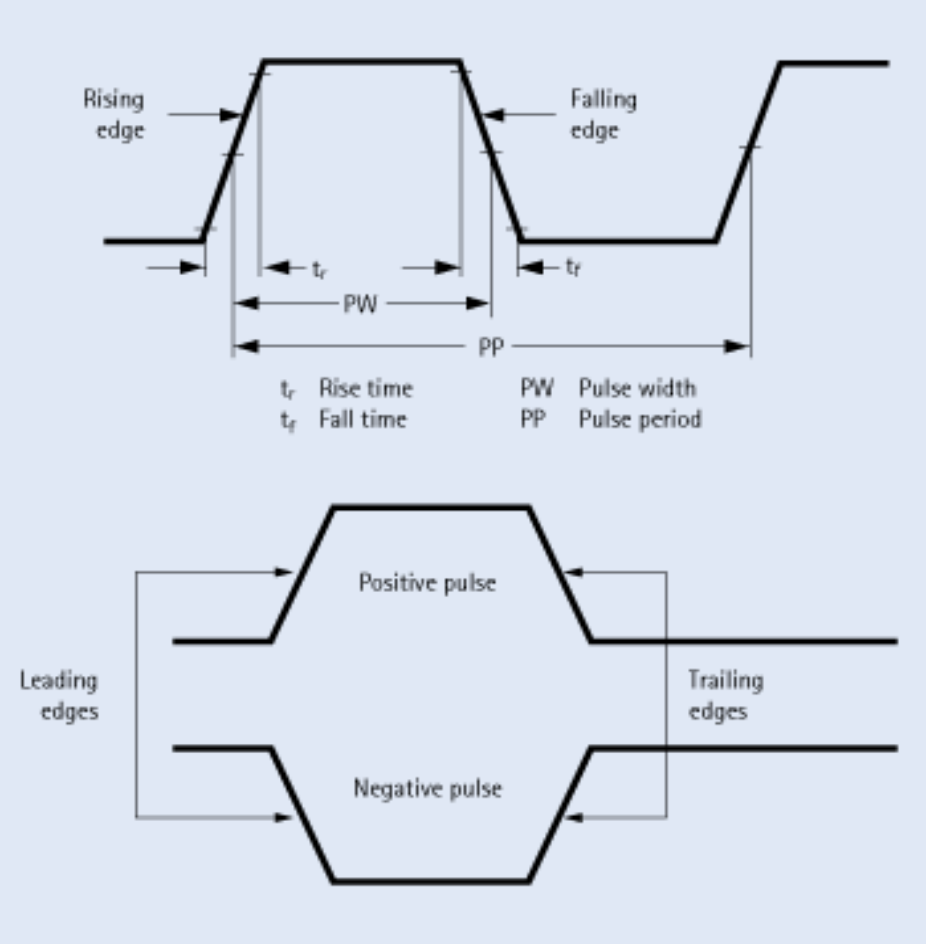
Digital Signal

Digital Signal can be characterised with:

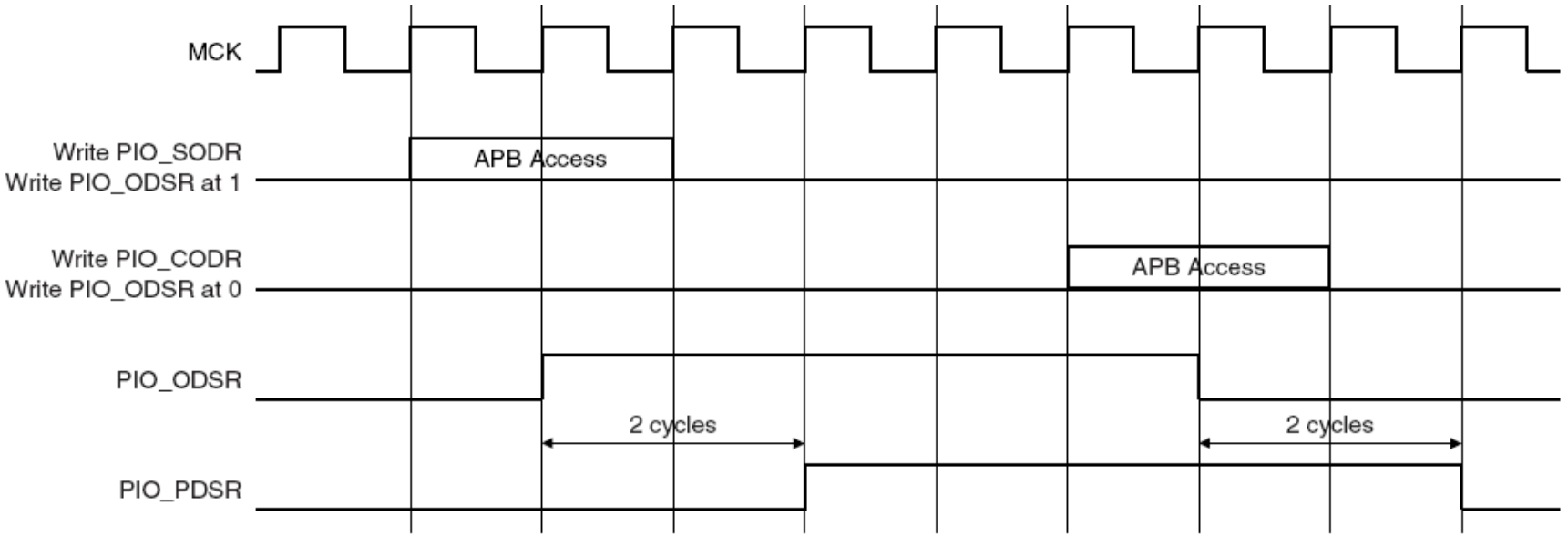
- f – frequency (period),
- A – amplitude.

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').



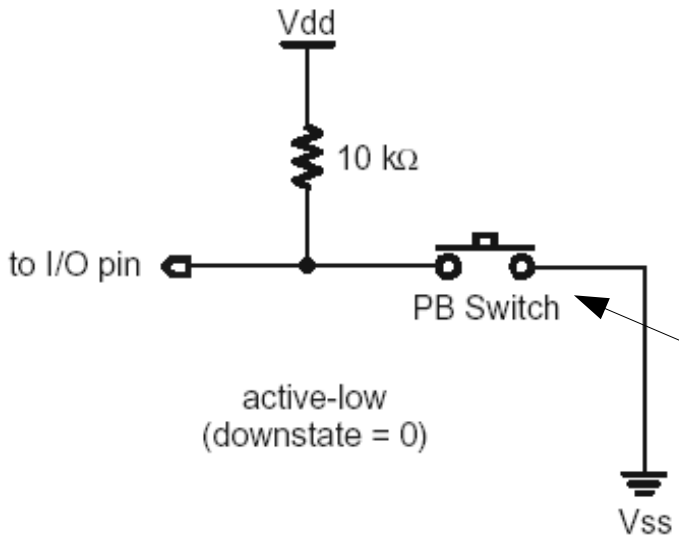
Przebiegi czasowe podczas sterowania portem I/O



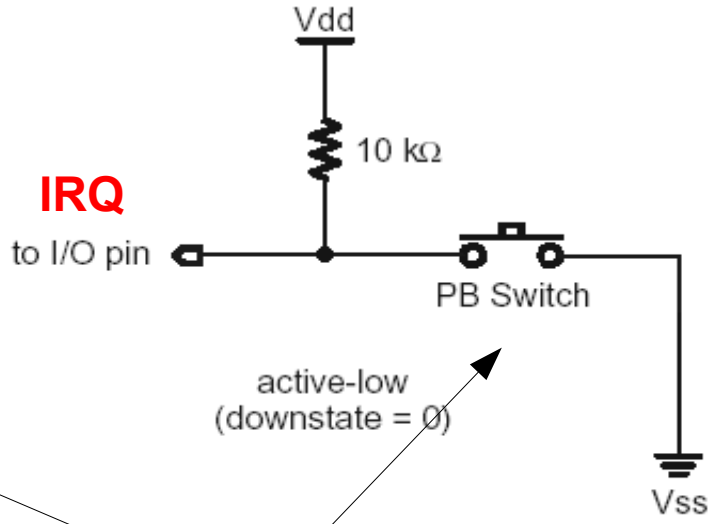
- 1 cykl zegarowy opóźnienia, gdy wyjście sterowane jest rejestrami SODR/CODR.
- 2 cykle opóźnienia podczas zapisu całego portu (32 bit, ustawione bity rejestru PIO_OWSR), dodatkowa synchronizacja słowa 32 bitowego

Odczyt stanu przełącznika

Polling loop



Interrupt



Sygnał asynchroniczny

Zarządzanie sygnałem zegarowym urządzeń peryferyjnych

PMC Peripheral Clock Enable Register
Register Name: PMC_PCER
Address: 0xFFFFFC10

Table 10-1. AT91SAM9263 Peripheral Identifiers

Peripheral ID	Peripheral Mnemonic	Peripheral Name	External Interrupt
0	AIC	Advanced Interrupt Controller	FIQ
1	SYSC	System Controller Interrupt	
2	PIOA	Parallel I/O Controller A	
3	PIOB	Parallel I/O Controller B	
4	PIOC to PIOE	Parallel I/O Controller C, D and E	
5	reserved		
6	reserved		
7	US0	USART 0	
8	US1	USART 1	
9	US2	USART 2	

```
write_register(PMC_PCER,0x00000110);     // Peripheral clocks 4 and 8 are enabled.
write_register(PMC_PCDR,0x00000010);     // Peripheral clock 4 is disabled.
```

Rejestry odwzorowane w strukturze (1)

Deklaracja nowego typu danych tworzy szablon układu rejestrów procesora w pamięci. Rejestrom zostają przypisane nazwy symboliczne. Utworzono nowy typ danych **AT91S_PIO** oraz wskaźnik ***AT91PS_PIO** na ten typ.

Brak informacji o dostępie do rejestrów (R/W) oraz wartości rejestrów po resecie.

W celu uzupełnienia brakujących informacji można umieścić dodatkowe komentarze.

```
typedef struct _AT91S_PIO {          /*     Register name              R/W     Reset val.     Offset
    AT91_REG  PIO_PER;               // PIO Enable Register             W       -              0x00
    AT91_REG  PIO_PDR;               // PIO Disable Register            W       -              0x04
    AT91_REG  PIO_PSR;               // PIO Status Register             R       0x0            0x08
    AT91_REG  Reserved0[1];                  //
    AT91_REG  PIO_OER;               // Output Enable Register          W       -              0x10
    AT91_REG  PIO_ODR;               // Output Disable Register         W       -              0x14
    AT91_REG  PIO_OSR;               // Output Status Register          W       -              0x18
    } AT91PS_PIO*;
    /* blok rejestrów portów I/O PIOA...PIOE */
#define AT91C_BASE_PIOA     (AT91PS_PIO)     0xFFFFF200     // (PIOA) Base Address
    /* maska zerowego bitu portu PA */
#define AT91C_PIO_PA0       (1 <<  0)        // Pin Controlled by PA0
```

Jak ustawić bity 0 i 19 rejestru PIO_PER ?

Rejestry odwzorowane w strukturze (2)

// Struktura opisująca rejestry portów IO procesora ARM

```
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  Reserved0[1];             // 32-bit filler
        AT91_REG  PIO_OER;          // Output Enable Register
        AT91_REG  PIO_ODR;          // Output Disable Register
        AT91_REG  PIO_OSR;          // Output Status Register
        AT91_REG  Reserved1[1];             // 32-bit filler
        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];             // 32-bit filler
        AT91_REG  PIO_SODR;         // Set Output Data Register
        AT91_REG  PIO_CODR;         // Clear Output Data Register
        AT91_REG  PIO_ODSR;         // Output Data Status Register
} AT91S_PIO, *AT91PS_PIO;
```

Odwołanie do rejestrów

Zapis do rejestru

```
AT91PS_PIO->PIO_OER = 0x5;
```

Odczyt danej z rejestru

```
volatile unsigned int ReadData;
ReadData = AT91PS_PIO->PIO_OSR;
```

Operacje bitowe

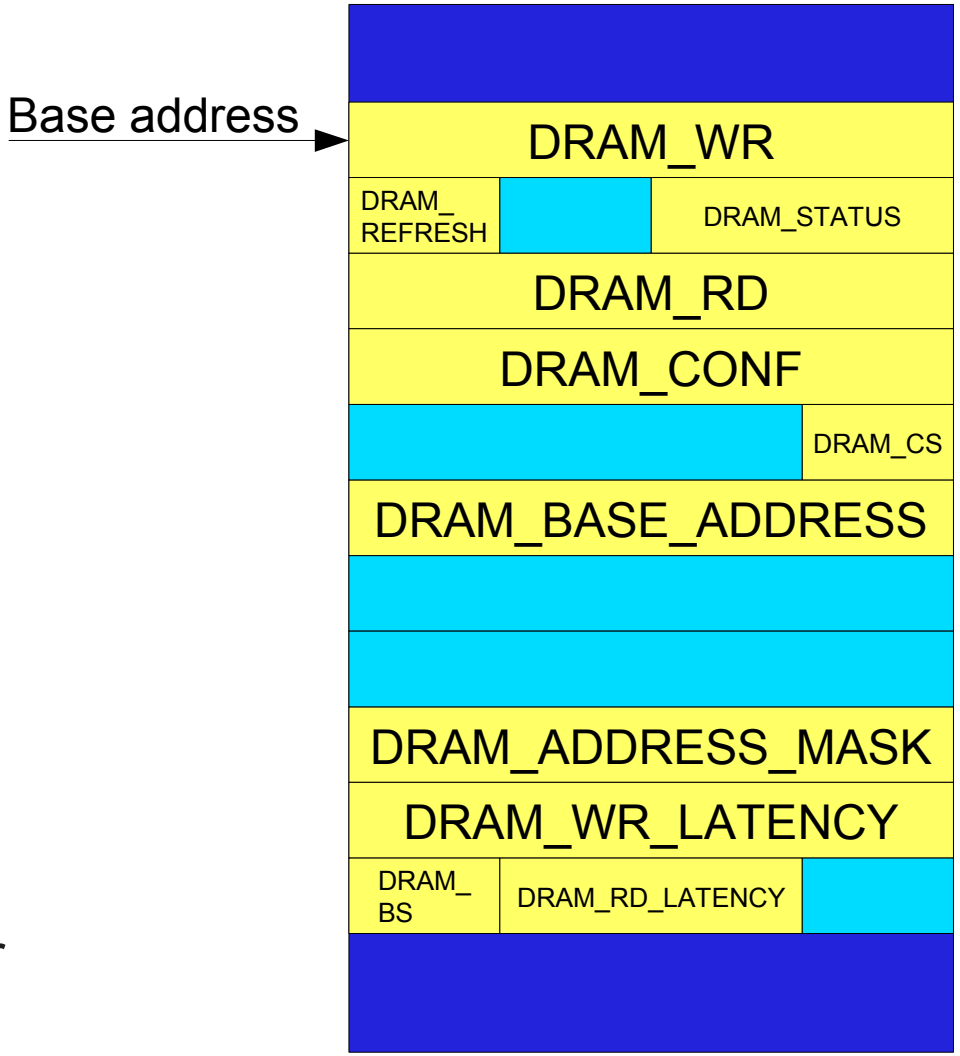
```
AT91C_BASE_PIOA->PIO_PER  = (AT91C_PIO_PA0 | AT91C_PIO_PA19);
AT91C_BASE_PIOA->PIO_PDR = (AT91C_PIO_PA0 | AT91C_PIO_PA19);
AT91C_BASE_PIOA->PIO_ODSR ^= (AT91C_PIO_PA0 | AT91C_PIO_PA19);
```

Registers mapped into structure - exercise

- Registers of DRAM memory are mapped into memory space,
- Base address: 0xFFFE.2000,
- Registers type: 8, 16, 32 bit,

Task to do:

- Create new struct type for DRAM registers,
- Declare pointer,
- Read, write data from memory,
- Set and clear configuration registers (bit 5, bit 29),
- Check busy flag in status register (bit 9)



Wskaźniki do rejestrów bazowych, C oraz asembler

```
        /* wskaźniki do rejestrów portów I/O PIOA...PIOE */
#define AT91C_BASE_AIC      (AT91_CAST(AT91PS_AIC)      0xFFFFF000) // (AIC) Base Address
#define AT91C_BASE_PIOA     (AT91_CAST(AT91PS_PIO)      0xFFFFF200) // (PIOA) Base Address
#define AT91C_BASE_PIOB     (AT91_CAST(AT91PS_PIO)      0xFFFFF400) // (PIOB) Base Address
#define AT91C_BASE_PIOC     (AT91_CAST(AT91PS_PIO)      0xFFFFF600) // (PIOC) Base Address
#define AT91C_BASE_PIOD     (AT91_CAST(AT91PS_PIO)      0xFFFFF800) // (PIOD) Base Address
#define AT91C_BASE_PIOE     (AT91_CAST(AT91PS_PIO)      0xFFFFFA00) // (PIOE) Base Address
#define AT91C_BASE_CKGR     (AT91_CAST(AT91PS_CKGR)     0xFFFFFC20) // (CKGR) Base Address
#define AT91C_BASE_PMC      (AT91_CAST(AT91PS_PMC)      0xFFFFFC00) // (PMC) Base Address
```

Makro pozwalające na rzutowanie wskaźników na pożądany typ danych w języku C:

```
#ifndef __ASSEMBLY__
        typedef volatile unsigned int AT91_REG;          // Hardware register definition
#define AT91_CAST(a) (a)
#else
#define AT91_CAST(a)
#endif
```

Deklaracja rejestrów dla asemblera

```
// ********************************************************************************************
//            SOFTWARE API DEFINITION  FOR Parallel Input Output Controler
// ********************************************************************************************
```

Rejestry dostępne z poziomu języka C

```
#ifndef __ASSEMBLY__
typedef struct _AT91S_PIO {
        AT91_REG  PIO_PER;              // PIO Enable Register
        AT91_REG  PIO_PDR;              // PIO Disable Register
        AT91_REG  PIO_PSR;              // PIO Status Register
        AT91_REG  Reserved0[1];                 // 32-bit filler
        AT91_REG  PIO_OER;              // Output Enable Register
        AT91_REG  PIO_ODR;              // Output Disable Register
        ...
        AT91_REG  PIO_OWSR;             // Output Write Status Register
} AT91S_PIO, *AT91PS_PIO;
#else
```

Rejestry dostępne z poziomu asemblera

```
#define PIO_PER       (AT91_CAST(AT91_REG *)      0x00000000)     // (PIO_PER) PIO Enable Register
#define PIO_PDR       (AT91_CAST(AT91_REG *)      0x00000004)     // (PIO_PDR) PIO Disable Register
#define PIO_PSR       (AT91_CAST(AT91_REG *)      0x00000008)     // (PIO_PSR) PIO Status Register
#define PIO_OER       (AT91_CAST(AT91_REG *)      0x00000010)     // (PIO_OER) Output Enable Register
...
#define PIO_OWSR      (AT91_CAST(AT91_REG *)      0x000000A8)     // (PIO_OWSR) Output Write Stat. Register
#endif
```

Dostęp do rejestrów w asemblerze

W pliku nagłówkowym AT91SAM9263.h zdefiniowano adresy bazowe do rejestrów obsługujących urządzenia peryferyjne:

```
#define AT91C_BASE_PIOC       0xFFFFF600      // (PIOC) Base Address, brak rzutowania
```

W pliku nagłówkowym zdefiniowano przesunięcia pozwalające na dostęp do rejestrów względem adresu bazowego:

```
#define PIO_OER                 0x00000010      // (PIO_OER) Output Enable Register, brak rzutowania
```

Jak odwołać się do rejestru z poziomu asemblera, zapis, odczyt, negacja bitu ?

```
LDR     r0, =AT91C_BASE_PIOC        // adres bazowy
LDR     r1, =PIO_OER                // przesuniecie dla rejestru OER
MOV     r2, #100                    // wartość zapisywana
STR     r0, [r1,r2]                 // zapis OER
LDR     r0, [r1,r2]                 // odczyt OER
```

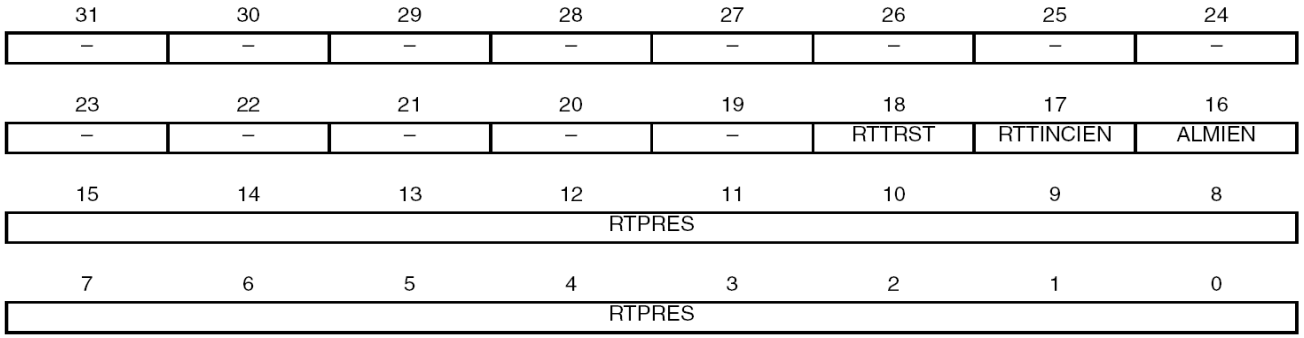
Przykład rejestru sterującego – timer czasu rzeczywistego

15.4.1 Real-time Timer Mode Register

Register Name: RTT_MR

Addresses: 0xFFFFFD20 (0), 0xFFFFFD50 (1)

Access Type: Read/Write



- **ALMIEN: Alarm Interrupt Enable**

0 = The bit ALMS in RTT_SR has no effect on interrupt.

1 = The bit ALMS in RTT_SR asserts interrupt.

- **RTTINCIEN: Real-time Timer Increment Interrupt Enable**

0 = The bit RTTINC in RTT_SR has no effect on interrupt.

1 = The bit RTTINC in RTT_SR asserts interrupt.

- **RTTRST: Real-time Timer Restart**

1 = Reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

// -------- RTTC_RTMR : (RTTC Offset: 0x0) Real-time Mode Register --------

```
#define AT91C_RTTC_RTPRES       (0xFFFF <<  0)   // (RTTC) Real-time Timer Prescaler Value
#define AT91C_RTTC_ALMIEN       (0x1 << 16)      // (RTTC) Alarm Interrupt Enable
#define AT91C_RTTC_RTTINCIEN    (0x1 << 17)      // (RTTC) Real Time Timer Increment Interrupt Enable
#define AT91C_RTTC_RTTRST       (0x1 << 18)      // (RTTC) Real Time Timer Restart
```

Definicja rejestrów – pliki nagłówkowe (1)

```
#ifndef _PROJECT_H
#define _PROJECT_H
/*
 * Include your AT91 Library files and specific
 * compiler definitions
 */
#include "AT91SAM9263-EK.h"
#include "AT91SAM9263.h"
#endif  // _PROJECT_H
```

```
/*-----------------------*/
/*  LEDs Definition  */
/*-----------------------*/
#define AT91B_LED1                AT91C_PIO_PB8   /* DS1 */
#define AT91B_LED2                AT91C_PIO_PC29  /* DS2 */
#define AT91B_NB_LEB        2
#define AT91D_BASE_PIO_LED1   (AT91C_BASE_PIOB)
#define AT91D_BASE_PIO_LED2   (AT91C_BASE_PIOC)
#define AT91D_ID_PIO_LED1     (AT91C_ID_PIOB)
#define AT91D_ID_PIO_LED2     (AT91C_ID_PIOC)
```

```
/*-------------------------------*/
/* Push Button Definition  */
/*-------------------------------*/
#define AT91B_BP1             AT91C_PIO_PC5  // Left click
#define AT91B_BP2             AT91C_PIO_PC4  // Right click
#define AT91D_BASE_PIO_BP     AT91C_BASE_PIOC
#define AT91D_ID_PIO_BP       AT91C_ID_PIOCDE
```

Definicja rejestrów – pliki nagłówkowe (2)

```
/*-----------------------*/
/*  LEDs Definition  */
/*-----------------------*/
#define AT91B_LED1                AT91C_PIO_PB8   /* DS1 */
#define AT91B_LED2                AT91C_PIO_PC29  /* DS2 */
#define AT91B_NB_LEB        2
#define AT91D_BASE_PIO_LED1   (AT91C_BASE_PIOB)
#define AT91D_BASE_PIO_LED2   (AT91C_BASE_PIOC)
#define AT91D_ID_PIO_LED1     (AT91C_ID_PIOB)
#define AT91D_ID_PIO_LED2     (AT91C_ID_PIOC)

/*-------------------------------*/
/* Push Button Definition  */
/*-------------------------------*/
#define AT91B_BP1             AT91C_PIO_PC5  // Left click
#define AT91B_BP2             AT91C_PIO_PC4  // Right click
#define AT91D_BASE_PIO_BP     AT91C_BASE_PIOC
#define AT91D_ID_PIO_BP       AT91C_ID_PIOCDE
```

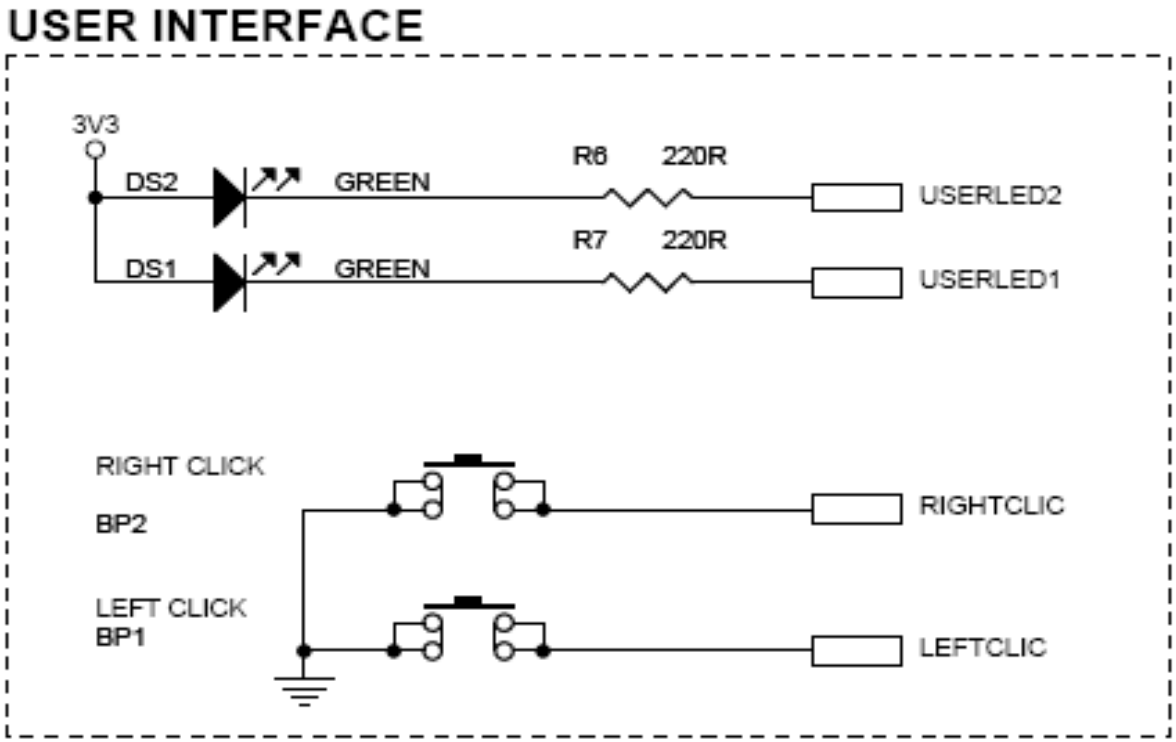
```
#define AT91C_PIO_PB8       (1 <<  8)    // Pin Controlled by PB8
#define AT91C_PIO_PC29      (1 << 29) // Pin Controlled by PC29
#define AT91C_BASE_PIOB (AT91_CAST(AT91PS_PIO)      0xFFFFF400) // (PIOB) Base Address
#define AT91C_BASE_PIOC (AT91_CAST(AT91PS_PIO)      0xFFFFF600) // (PIOC) Base Address

#define AT91C_ID_PIOB   ( 3) // Parallel IO Controller B

#define AT91C_PIO_PC4       (1 <<  4) // Pin Controlled by PC4
#define AT91C_PIO_PC5       (1 <<  5) // Pin Controlled by PC5

#define AT91C_ID_PIOCDE ( 4) // Parallel IO Controller C, Parallel IO Controller D, Parallel IO Controller E
```

Płyta uruchomieniowa MSC – diody LED, klawiatura




```
#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
```

Konfiguracja portów I/O procesora

```
#define AT91C_PIO_PB8        (1 <<  8)                    // Pin Controlled by PB8
#define AT91C_BASE_PIOB   (AT91PS_PIO)        0xFFFFF400    // (PIOB) Base Address
```

Konfiguracja Portu w tryb wejścia:

```
/* Enable the periph clock for the PIO controller, This is mandatory when PIO are configured as input */
AT91C_BASE_PMC->PMC_PCER = (1 << AT91C_ID_PIOCDE ); // peripheral clock enable register (port C, D, E)
/* Set the PIO line as input */
AT91C_BASE_PIOD->PIO_ODR = 0x0000.000F;              // 1 – Set direction of the pin to input
/* Set the PIO controller in PIO mode instead of peripheral mode */
AT91C_BASE_PIOD->PIO_PER = AT91C_PIO_PB8;            // 1 – Enable PIO to control the pin
```

Konfiguracja Portu w tryb wyjścia:

```
/* Configure the pin as output */
AT91C_BASE_PIOB->PIO_OER = AT91C_PIO_PB8 ;
/* Set the PIO controller in PIO mode instead of peripheral mode */
AT91C_BASE_PIOD->PIO_PER = 0xFFFF.FFFF;              // 1 – Enable PIO to control the pin
AT91C_BASE_PIOE->PIO_PER = AT91C_PIO_PB31;
/* Disable pull-up */
AT91C_BASE_PIOA->PIO_PPUDR = 0xFFFF.0000;            // 1 – Disable the PIO pull-up resistor
```

Powyższa konfiguracja dotyczy losowo wybranych bitów z portów A-E

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 Functionalities

```
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;
```

Prace inżynierskie (1)

1. Aplikacja diagnostyczna dla sterowników modułów zgodnych ze standardami ATCA i uTCA.

Diagnostic application for controllers of modules compliant with ATCA and uTCA standards.

2. Oprogramowanie dla układu inteligentnego zarządzania modułem AMC.
   Software for intelligent management controller of AMC module.

3. Aplikacja umożliwiająca zdalne monitorowanie i sterowanie systemem xTCA.
   Application for remote monitoring and control of xTCA system.

Prace inżynierskie (2)

1. Algorytmy określania trajektorii w przestrzeni trójwymiarowej z wykorzystaniem sygnałów pochodzących z czujników przyśpieszenia, żyroskopu oraz sensora ziemskiego pola magnetycznego

   (software).

2. Detekcja położenia w przestrzeni przy wykorzystaniu magnetometru i układów MEMS

   (hardware).

3. Bezprzewodowe urządzenie określające swoje położenie w trójwymiarowej przestrzeni z zastosowaniem magnetometru i układów MEMS

   (hardware + software).

4. Wizualizacja położenia w przestrzeni trójwymiarowej

   (software).