Dariusz Makowski

Katedra Mikroelektroniki i Technik Informatycznych

tel. 631 2720

dmakow@dmcs.pl

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

Sprawy formalne

- Informacje wstępne
- Laboratorium
- Materiały do wykładu

Zakres przedmiotu

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

Laboratorium

Adres:

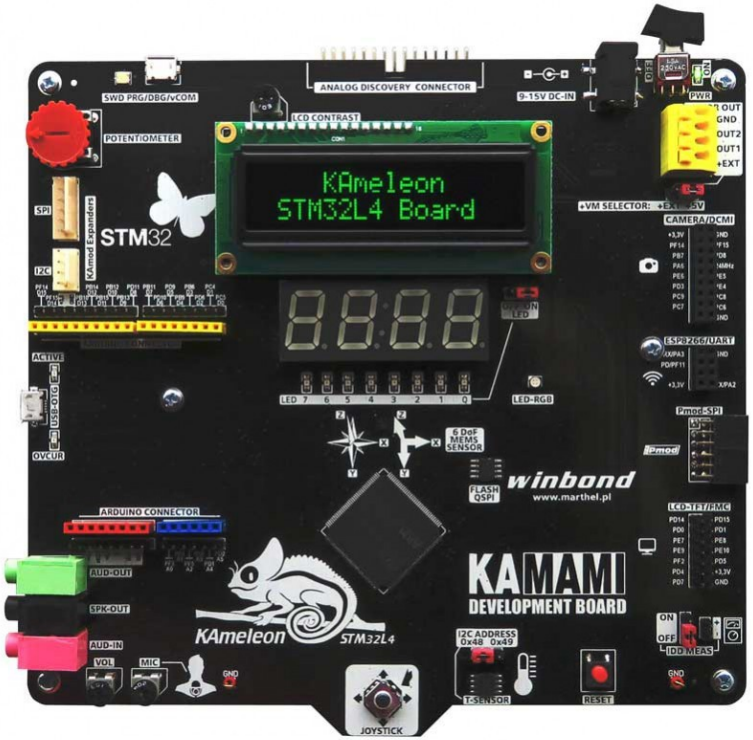
- Budynek B18, laboratorium M

Praktyczne ćwiczenia z użyciem procesora ARM:

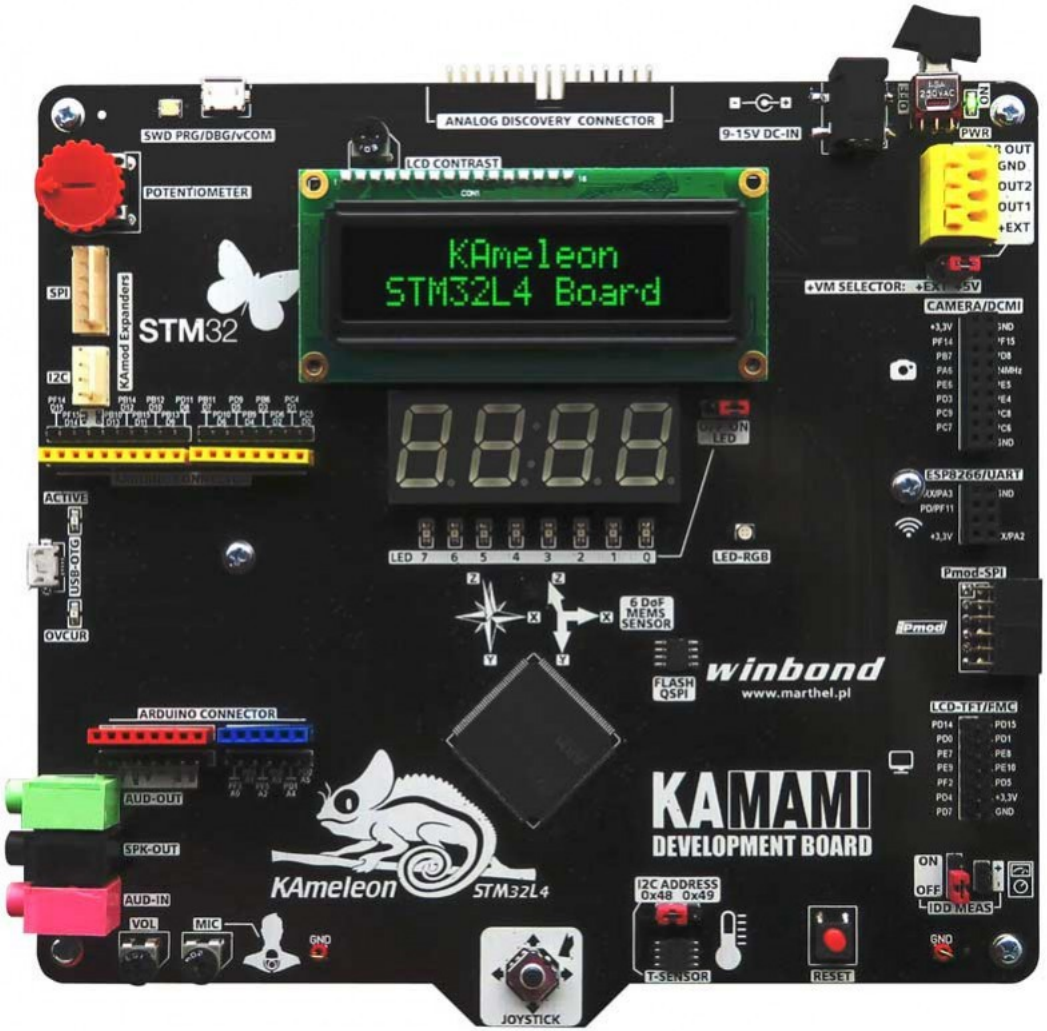
- GNU tools,
- Mikrokontroler STM32L496,
- Płyta ewaluacyjna Kameleon STM32L4
- Analizator uniwersalny Analog Discovery 2

Zestaw ewaluacyjny firmy STM (1)

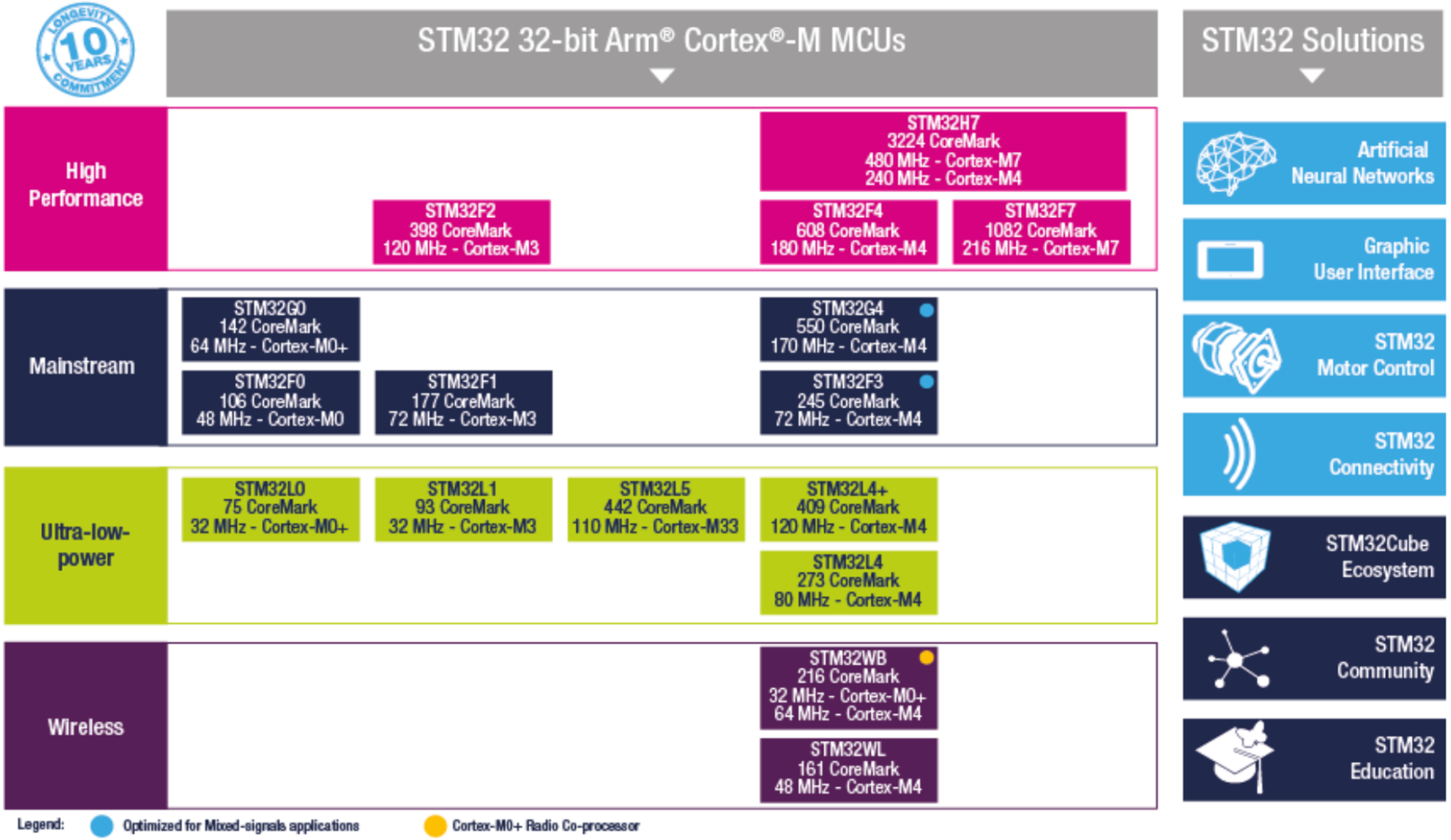
- **Procesor z rdzeniem ARM Cortex M4 firmy STM:** STM32L496ZGT6
- **Dostępne pamięci:** 320 kB SRAM,1 MB FLASH, 1 MB SPI-PROM
- **Dostępne interfejsy:** SPI, I2C, USB-OTG, EIA RS232
- **Wyświetlacz:** LED 4 cyfry, matrycowy, linijka 8x LED, dioda RGB
- **Audio:** mikrofon, wzmacniacz mocy
- **Urządzenia per.:** ster. silnika DC, termometr, akcelerometr, magnetometr
- **Interfejs debugera, programatora:** Serial Wire Debug
- **Złącze:** kamery DCMI, WiFi, PMOD, etc.
- **Programator:** ST-Link
- **Oznaczenie producenta:** KAmeleon-STM32L4



Zestaw ewaluacyjny firmy MSC (2)



Rodzina 32-bitowych mikrokontrolerów STM32



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

Rodzina STM a rdzeń mikrokontrolera ARM

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

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

Mikrokontrolery serii STM32L4 (ultra-low-power)

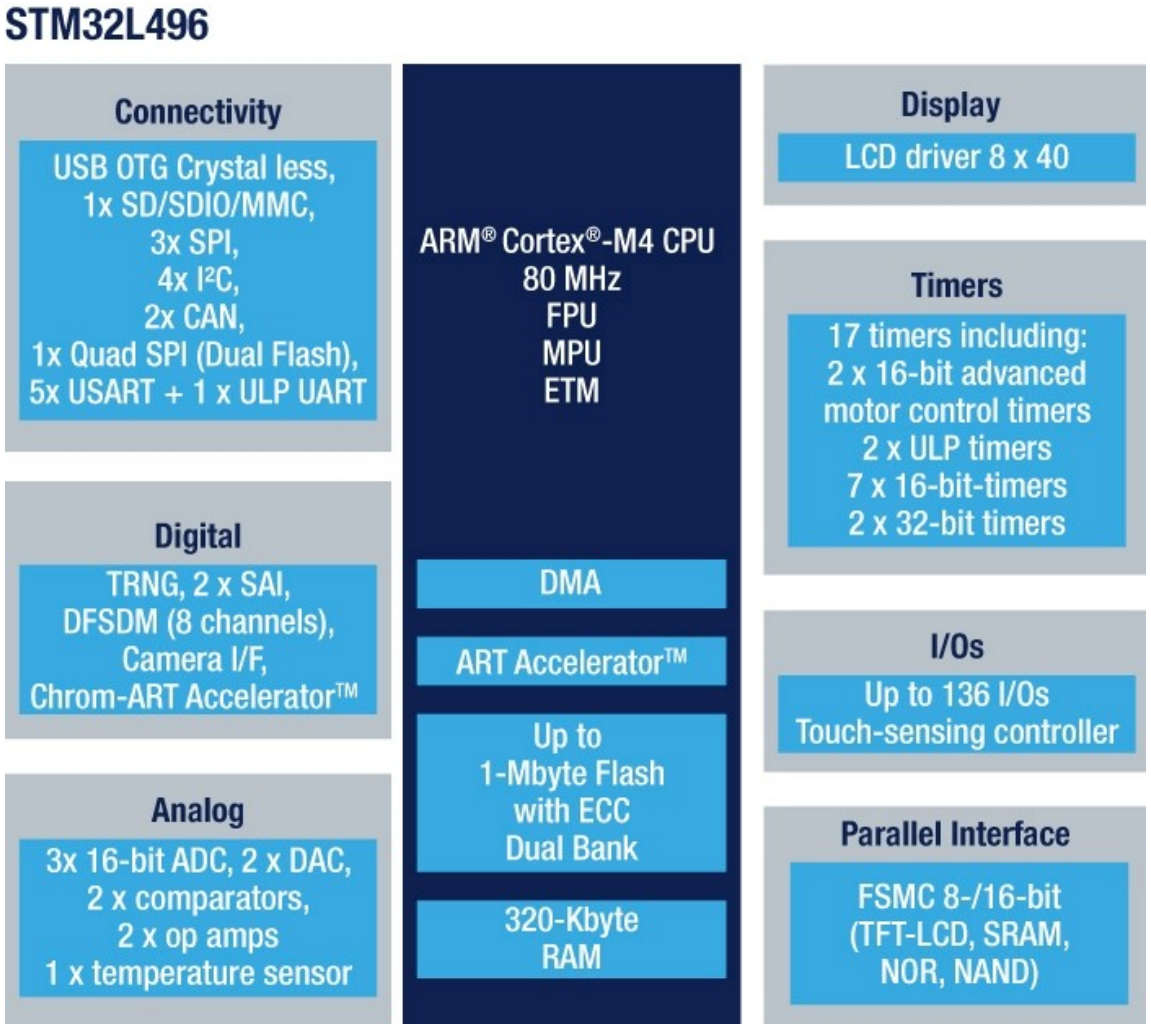
Arm® Cortex®-M4 (DSP + FPU) – 80 MHz

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

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

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

Mikrokontroler STM32L496 ...



Strona przedmiotu – materiały do wykładu

Pages for students

Start typing.../Zacznij pisać...

- Administracja i Bezpieczenstwo Systemow Sieciowych
- Advanced Mobile Applications
- Advanced Web Programming
- Algorytmy i struktury danych (OiZ)
- Projektowanie Systemów wbudowanych

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

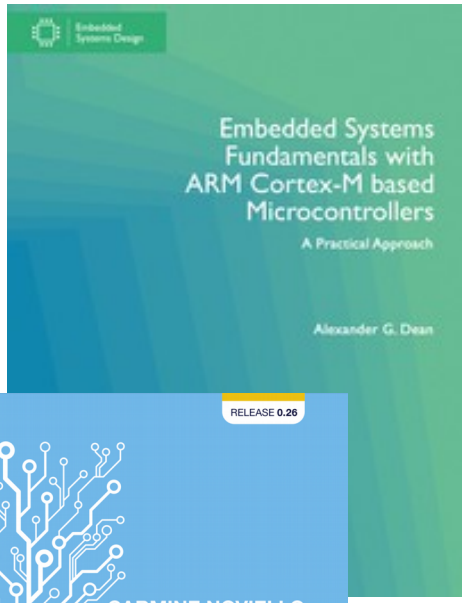
Literatura

Literatura obowiązkowa:

- Materiały wykładowe i laboratoryjne
- Strony producenta STM
- Inne materiały w Internecie

Literatura uzupełniająca:

- A. Dean, “Embedded Systems Fundamentals with ARM Cortex-M based Microcontrollers: A Practical Approach”, Arm Education Media, 2017, ISBN-13 9781911531012, ISBN-10 1911531018
- C. Noviello, “Mastering STM32” Leanpub 2018



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

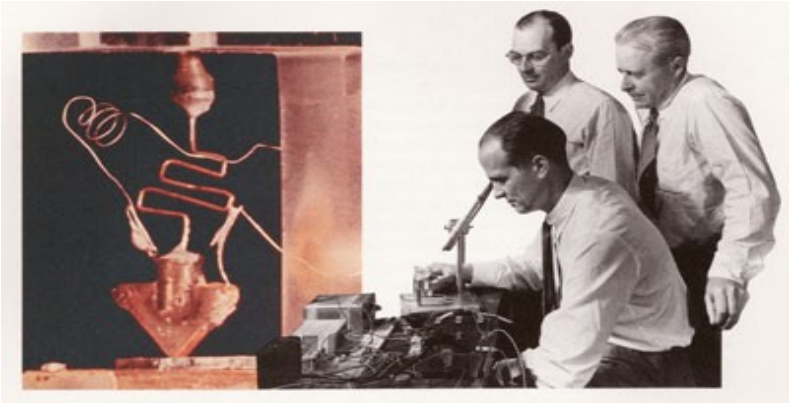
- **Mikrokontroler (ang. Microcontroller)**

  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

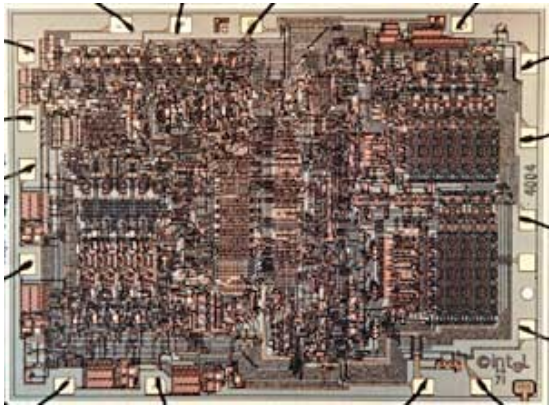
1967 – Laboratorium Fairchild oferuje pierwszą pamięć nieulotną ROM (64 bity)

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.

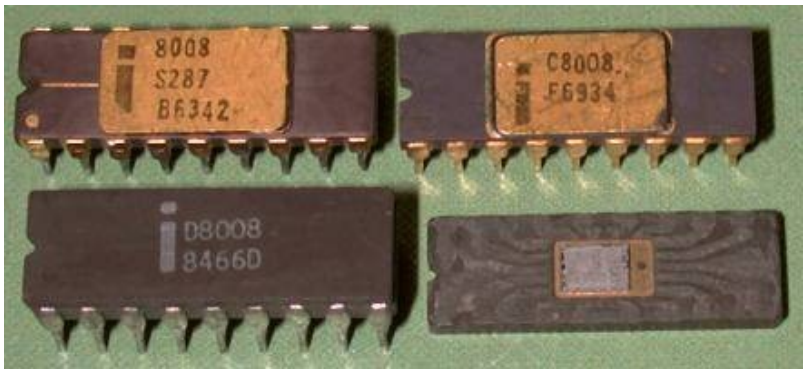
Historia mikroprocesorów (2)

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).

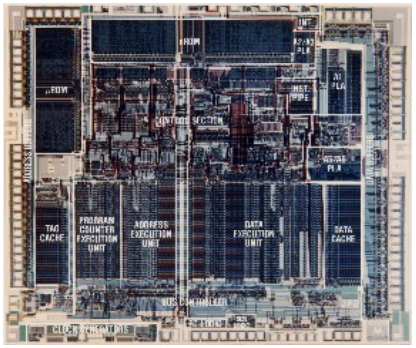
1975 – 8-bitowy procesor INTEL 6502 (technologia MOS) – najtańszy proc. na rynku.

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

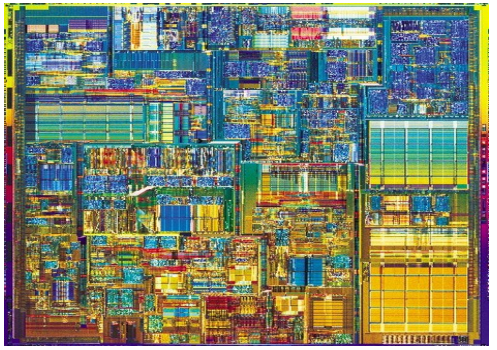
1979 – Motorola oferuje 16-bitowy procesor 68000.

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)**

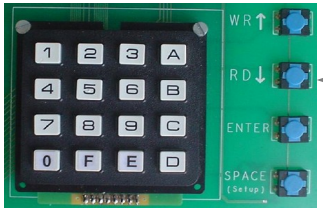
  - Sposób organizacji oraz współpracy podstawowych elementów systemu komputerowego, tj. procesora, pamięci oraz urządzeń peryferyjnych
  - Opis komputera z punktu widzenia programisty w języku niskiego poziomu (asembler). Budowa procesora, potoku wykonawczego oraz model programowy procesora

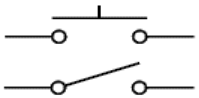
Komputer wbudowany (embedded computer)

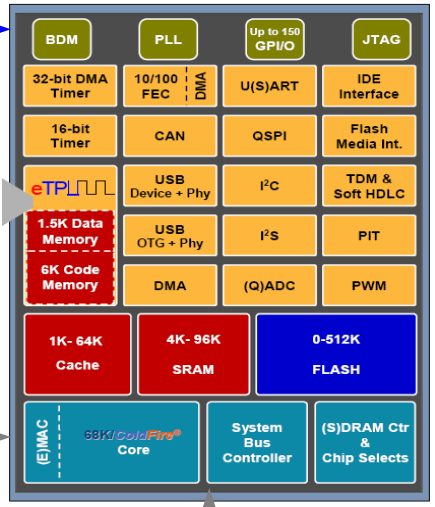
Komunikacja z komputerem zewnętrznym

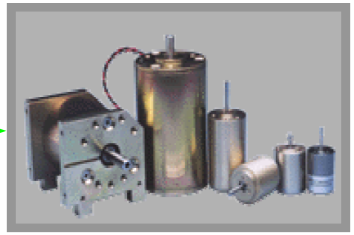
Czujniki, np. czujnik temperatury, obrotów, itd

Kamera



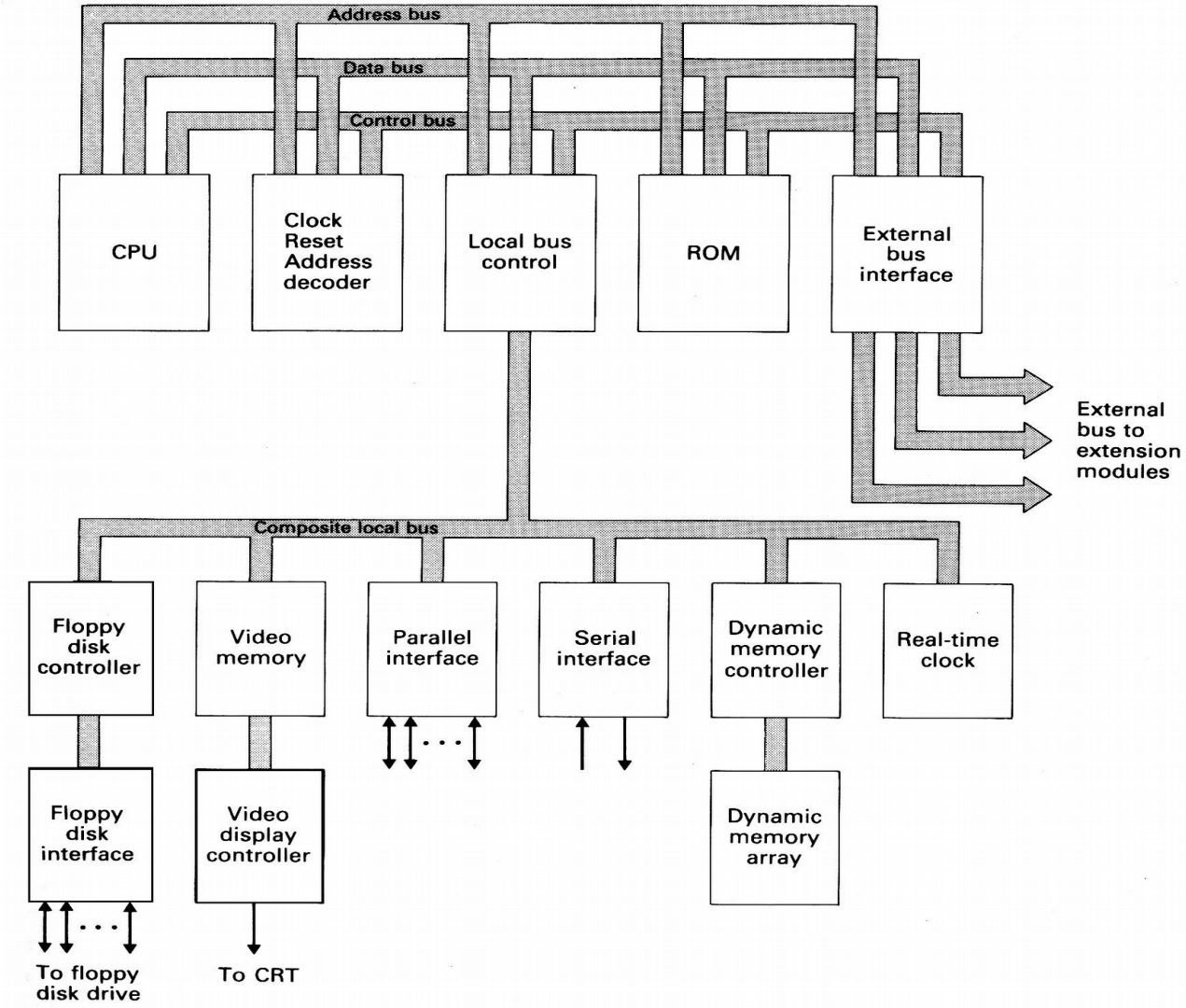




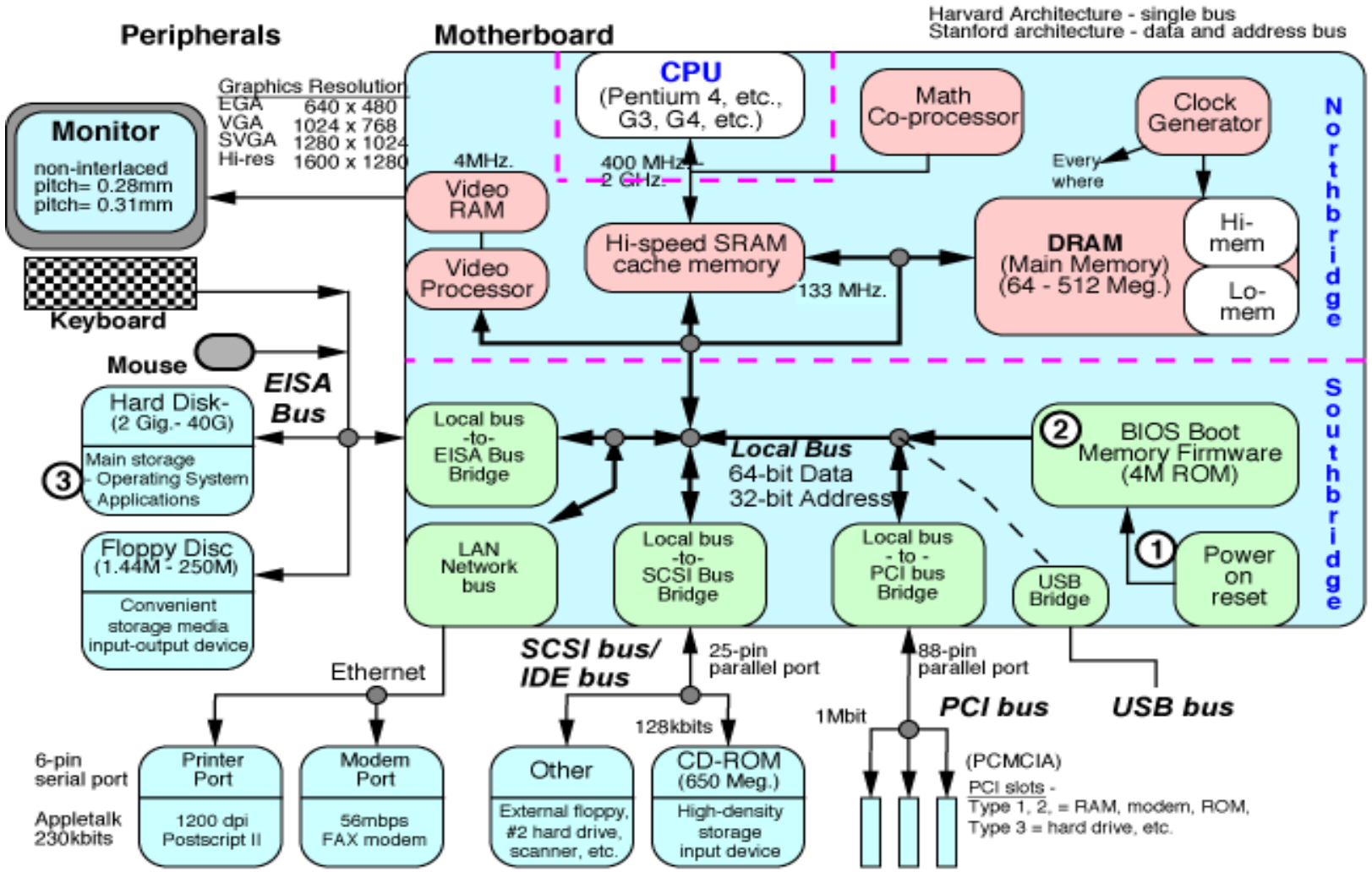


Elementy wykonawcze, np. silniki, przekaźniki

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.

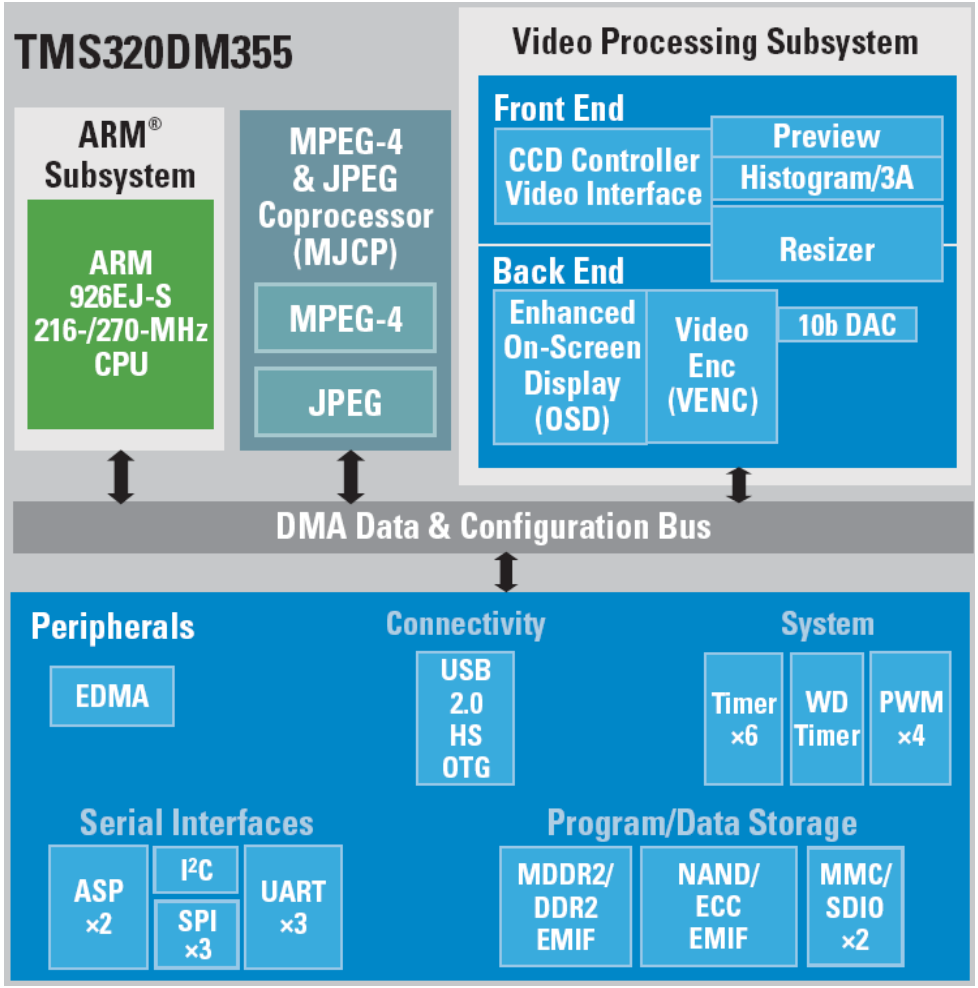
  W przypadku, gdy nie jest możliwa integracja wszystkich układów na jednym podłożu półprzewodnikowym, poszczególne moduły wykonuje się na osobnych kryształach, a całość zamyka się w jednej obudowie, SiP (ang. System-in-a-package).

  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

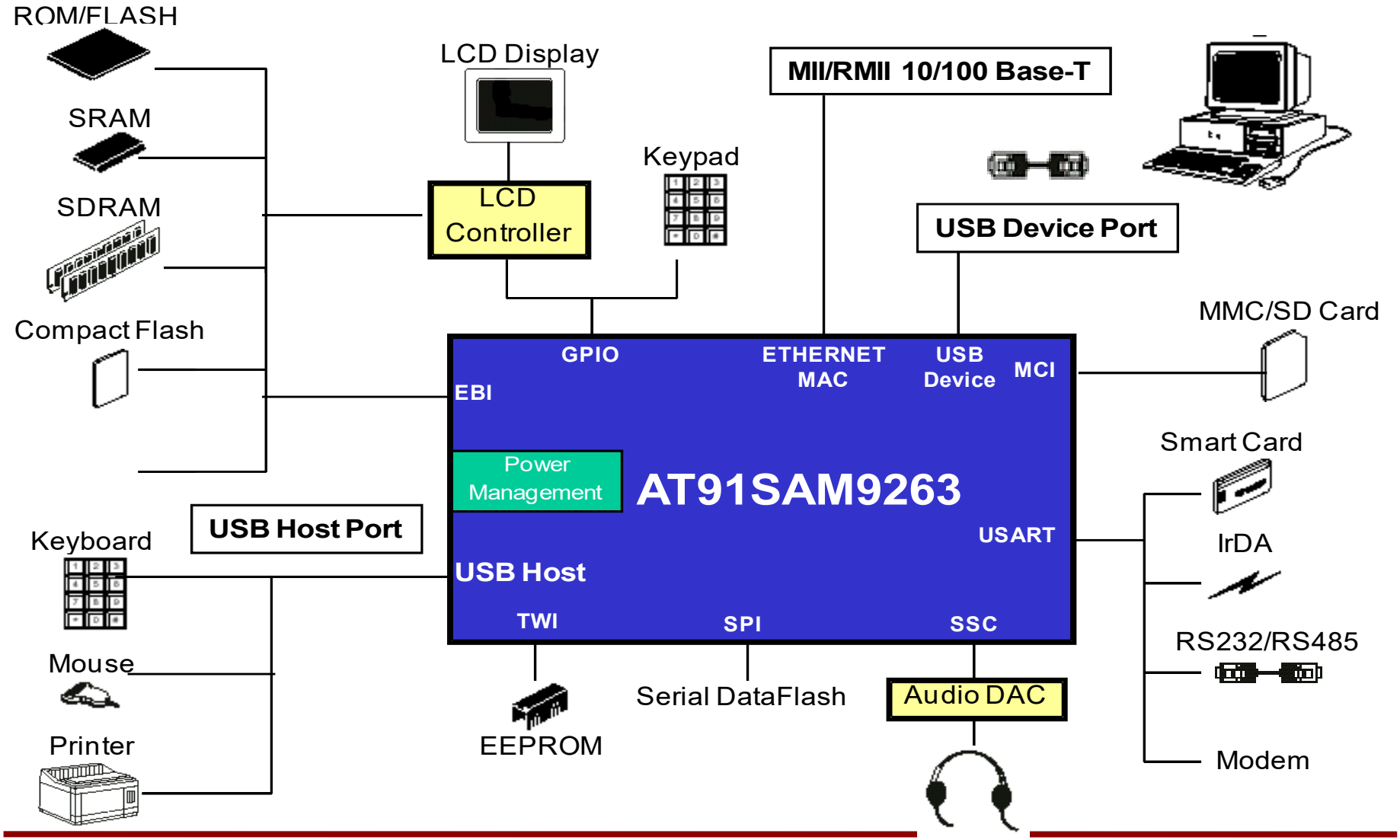
DaVinci DM355

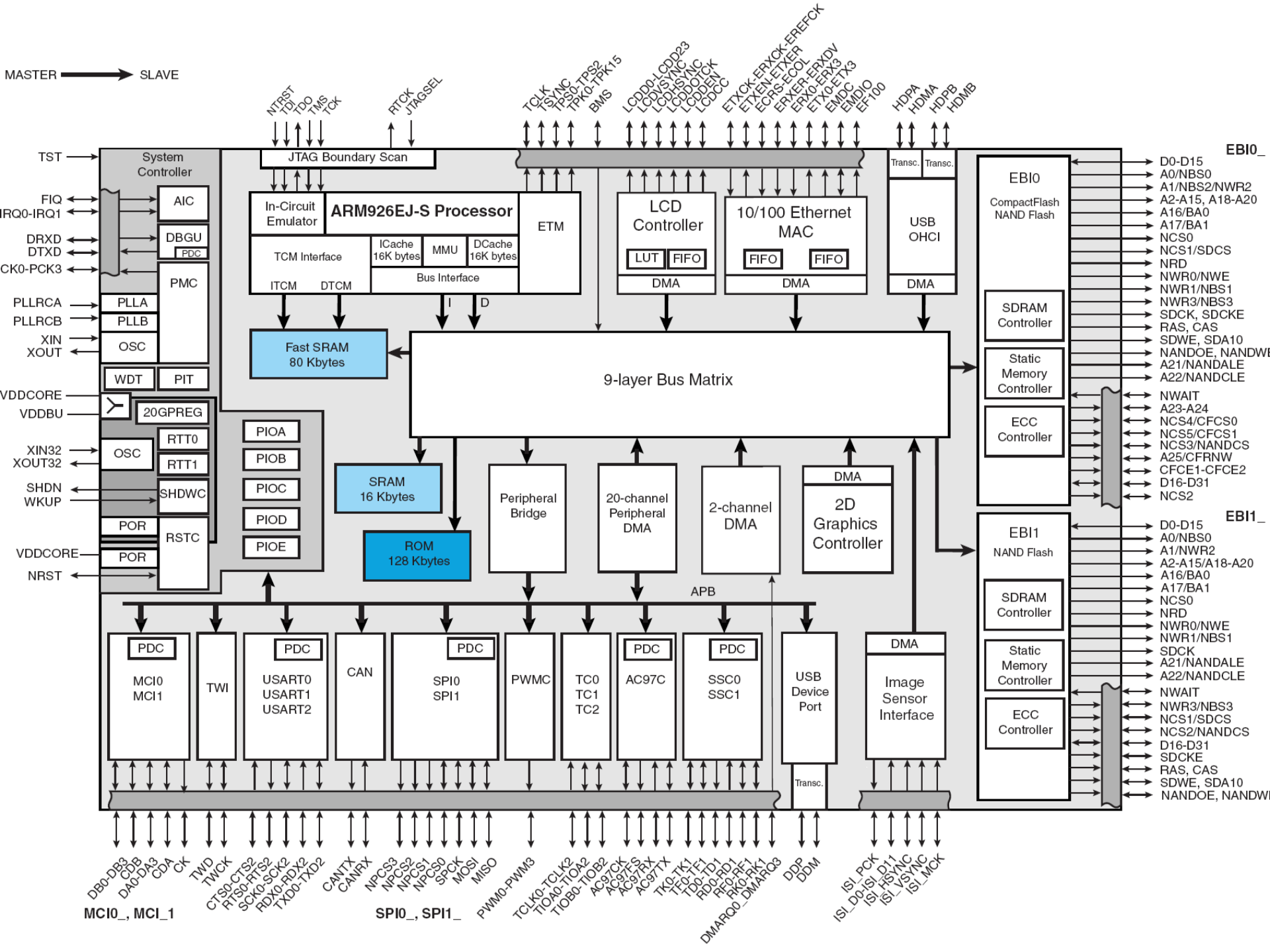
- SoC opracowany przez firmę Texas Instruments
- Dedykowany co-procesor do przetwarzania dźwięku i obrazu w czasie rzeczywistym
- Niski pobór energii 400 mW podczas dekodowania HD MPEG4, 1 mW w stanie czuwania (systemy przenośne)
- Bogate interfejsy i układy peryferyjne (sterownik HDD, SD/MMC, USB, Ethernet,...)



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/***):

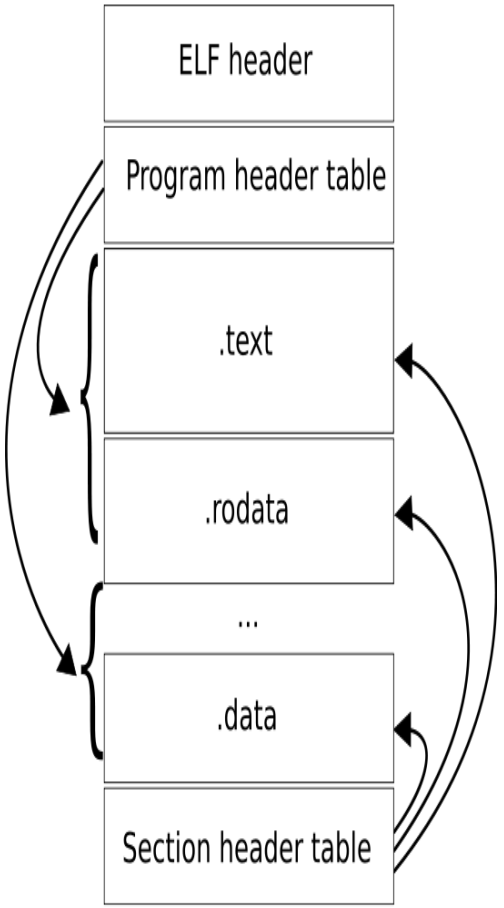
- GCC-4.3 toolchain (Linux):
  - Compiler: **gcc-4.3.2**
  - Useful tools: **binutils-2.19**
  - Library C/C++: **newlib-1.16**
  - Debugger compatible with GDB: **insight-6.8**
  - Tools are installed in **/opt/arm_tools/...**
  - Header files and scripts are in **/opt/arm_user/...**
- Cygwin (Windows):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

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

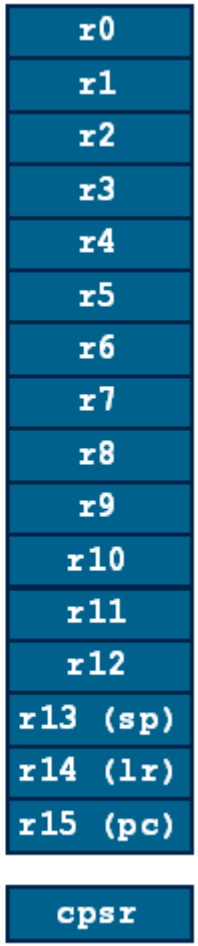
/t - wyświetla w postaci binarnej

/d - wyświetla w postaci dziesiętnej

Rejestry procesora a GDB

(gdb) info r

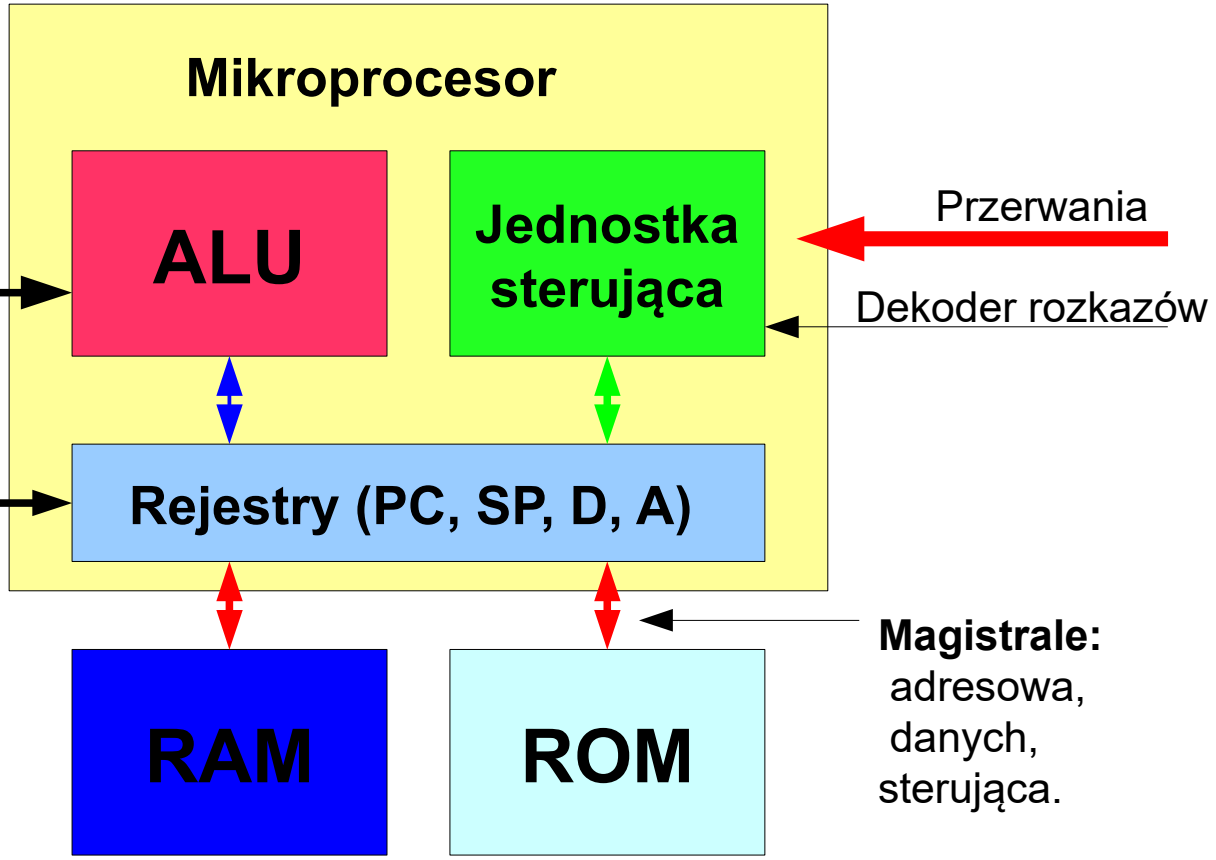
r0	0x2	0x2
r1	0x20000ba4	0x20000ba4
r2	0x57b	0x57b
r3	0x270f	0x270f
r4	0x300069	0x300069
r5	0x3122dc	0x3122dc
r6	0x1000	0x1000
r7	0x800bc004	0x800bc004
r8	0x3122c4	0x3122c4
r9	0x407c81a4	0x407c81a4
r10	0x441029ab	0x441029ab
r11	0x313f2c	0x313f2c
r12	0x313f30	0x313f30
sp	0x313f18	0x313f18
lr	0x20000a7c	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.

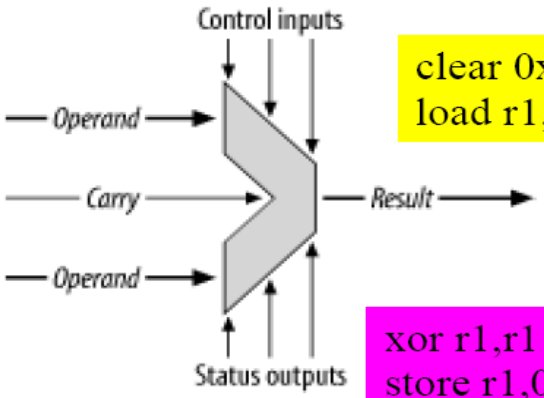
Jednostka arytmetyczno-logiczna (ang. Arithmetic Logic Unit), realizuje podstawowe operacje matematyczne 8, 16, 32, 64-bit



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

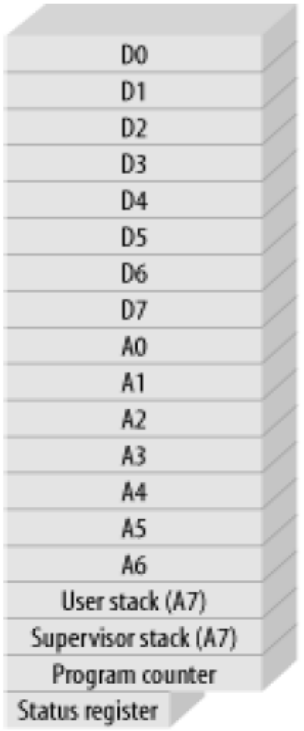
Architektura procesora (2)

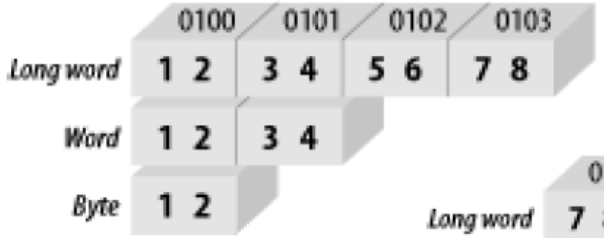
- ALU
  - lista operacji
- zestaw rejestrów
  - A, I, PC, SR, SP,...
- lista rozkazów
  - CISC, RISC
- tryby adresowania
  - R, A, I,...
- przerwania
  - hardware, software
- big/little endian

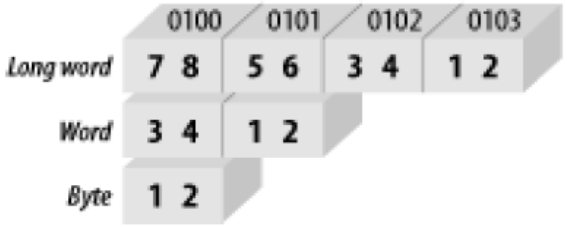



```
clear 0x1000
load r1,#5
```

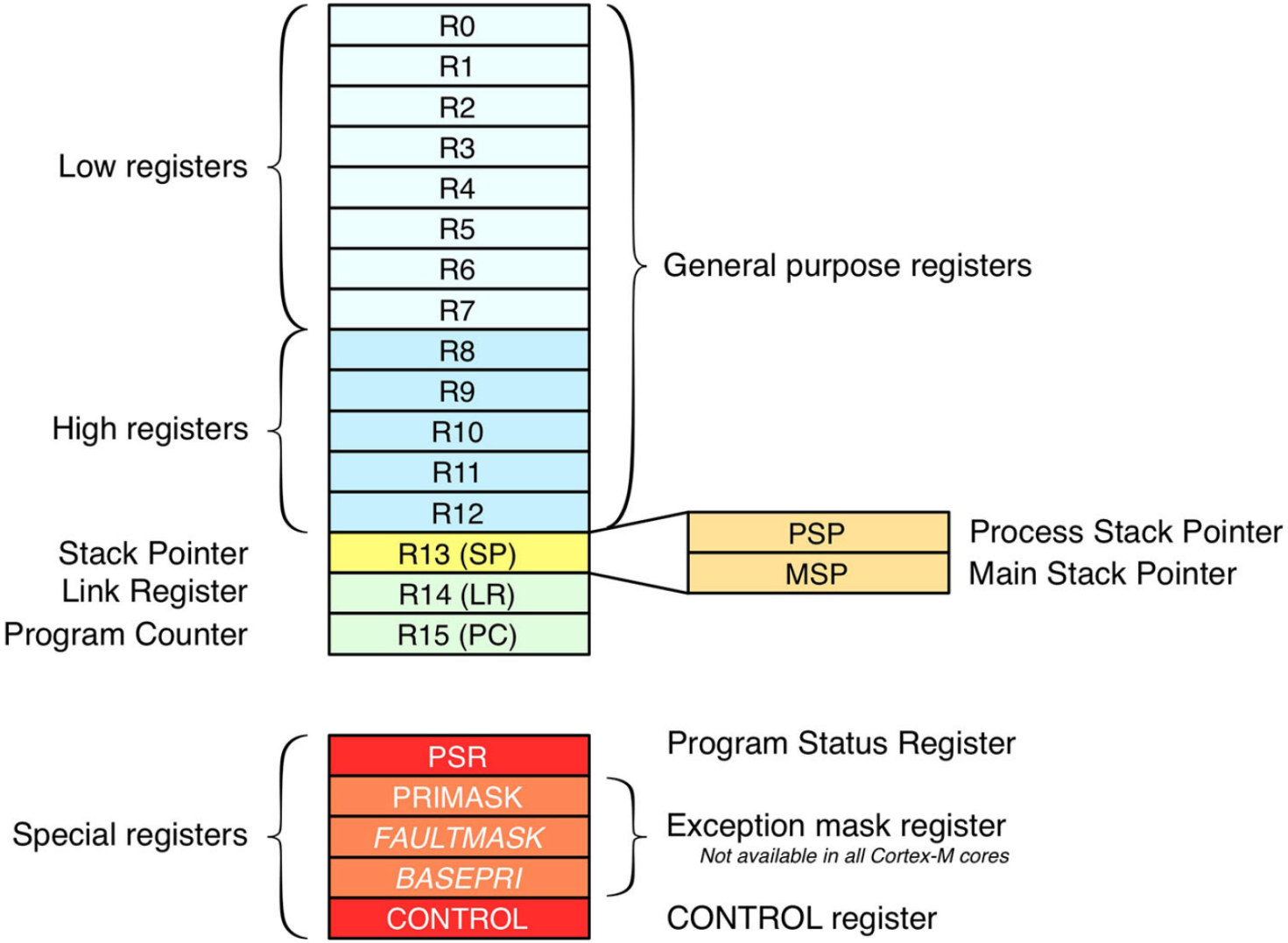
```
xor r1,r1
store r1,0x1000
add r1,#5
```







Rejestry procesorów z rodziny Cortex-M



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:

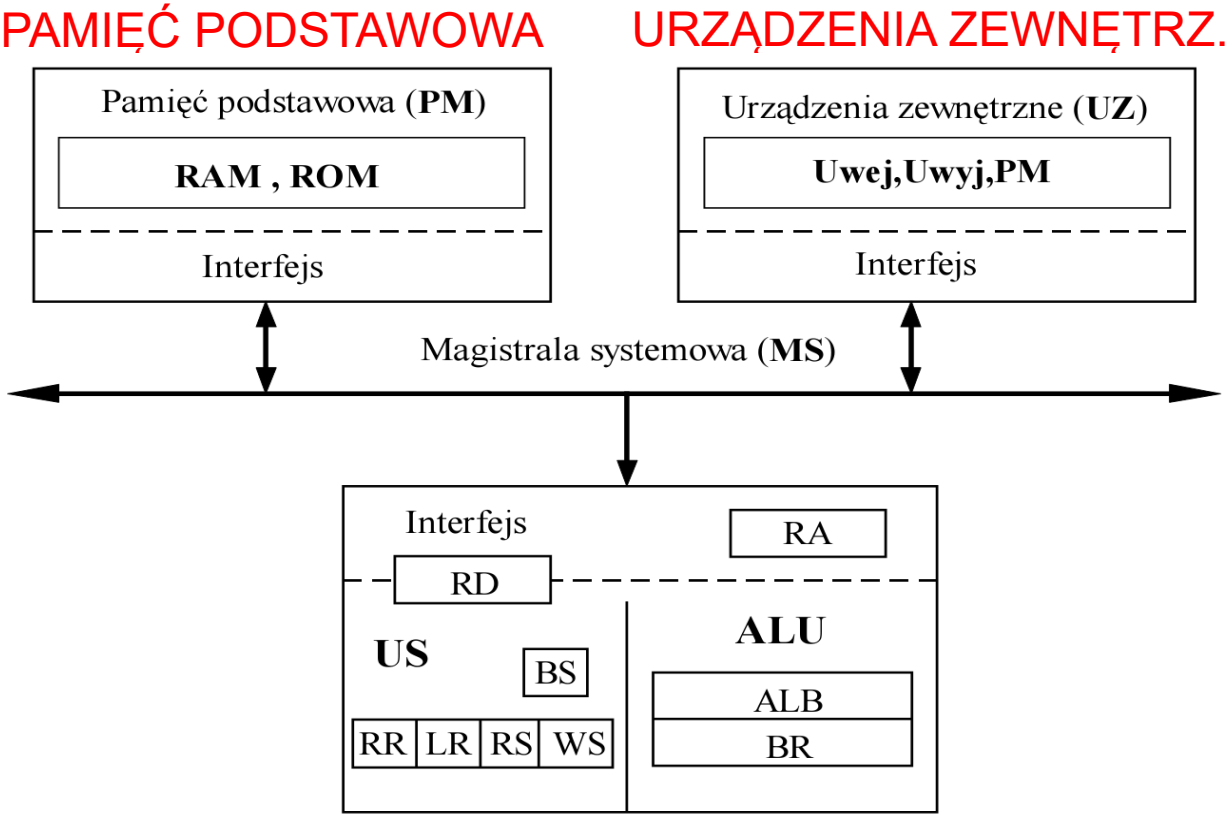
- IBM 801
- PowerPC
- MIPS
- Alpha
- **ARM**
- Motorola 88000
- ColdFire
- SPARC
- PA-RISC
- Atmel_AVR

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.

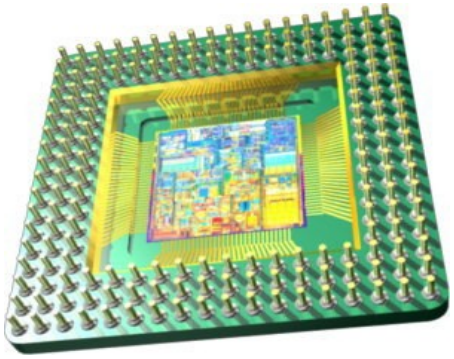
Architektura systemu komputerowego

Architektura polega na ścisłym podziale komputera na trzy podstawowe części:

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



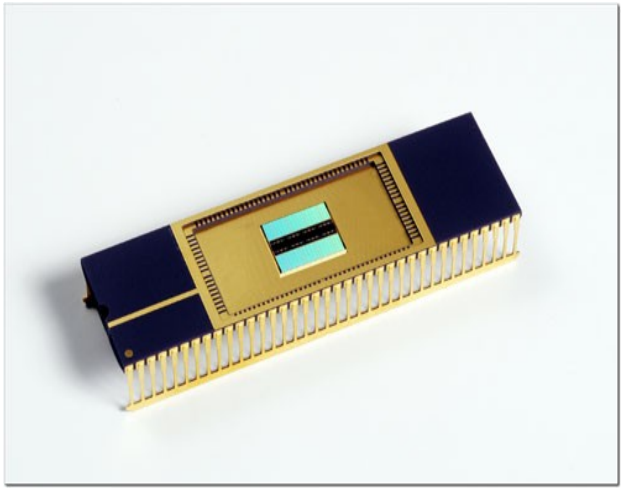
Magistrale komputera



Magistrala adresowa

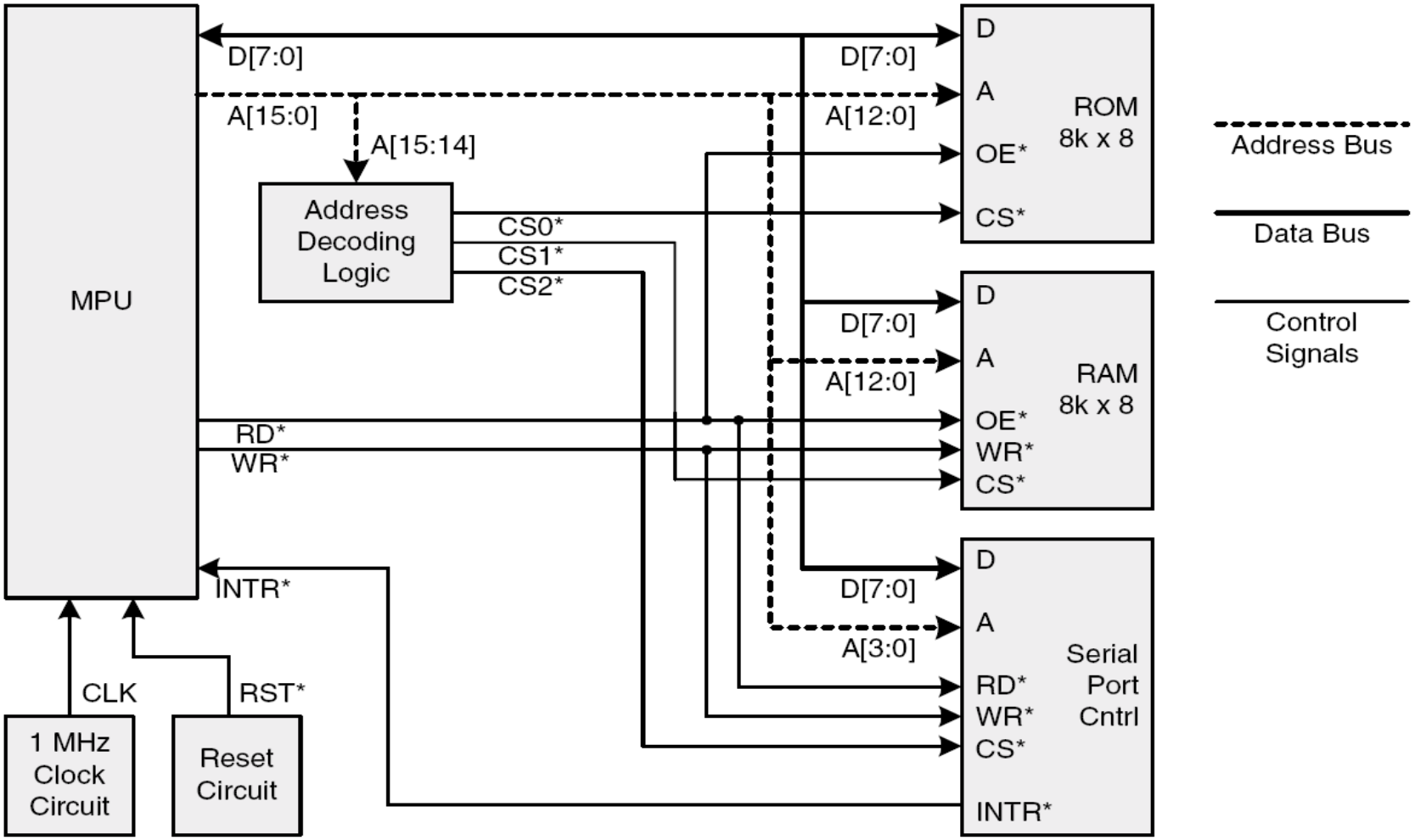
Magistrala danych

Magistrala sterująca



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

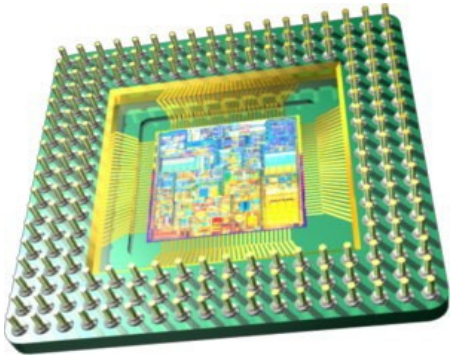
Przykładowy komputer 8-bitowy



Architektura von Neumanna

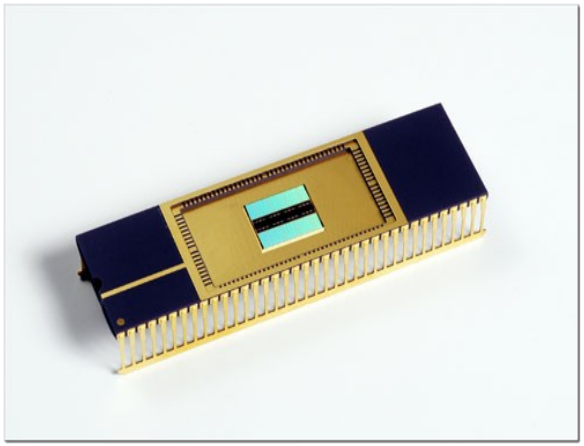
Cechy architektury von Neumanna:

- 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

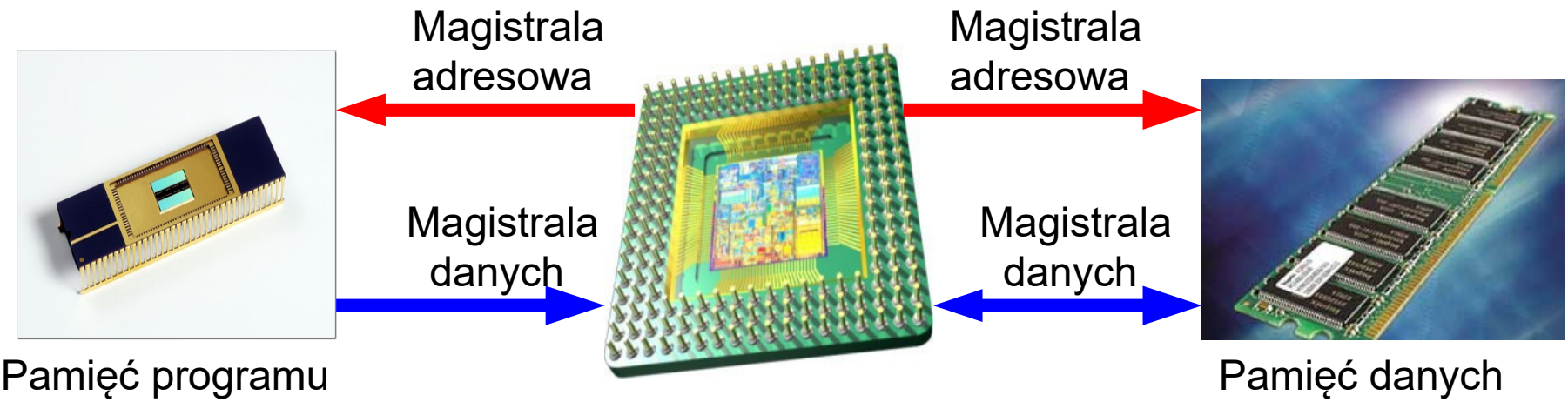


Architektura Harwardzka

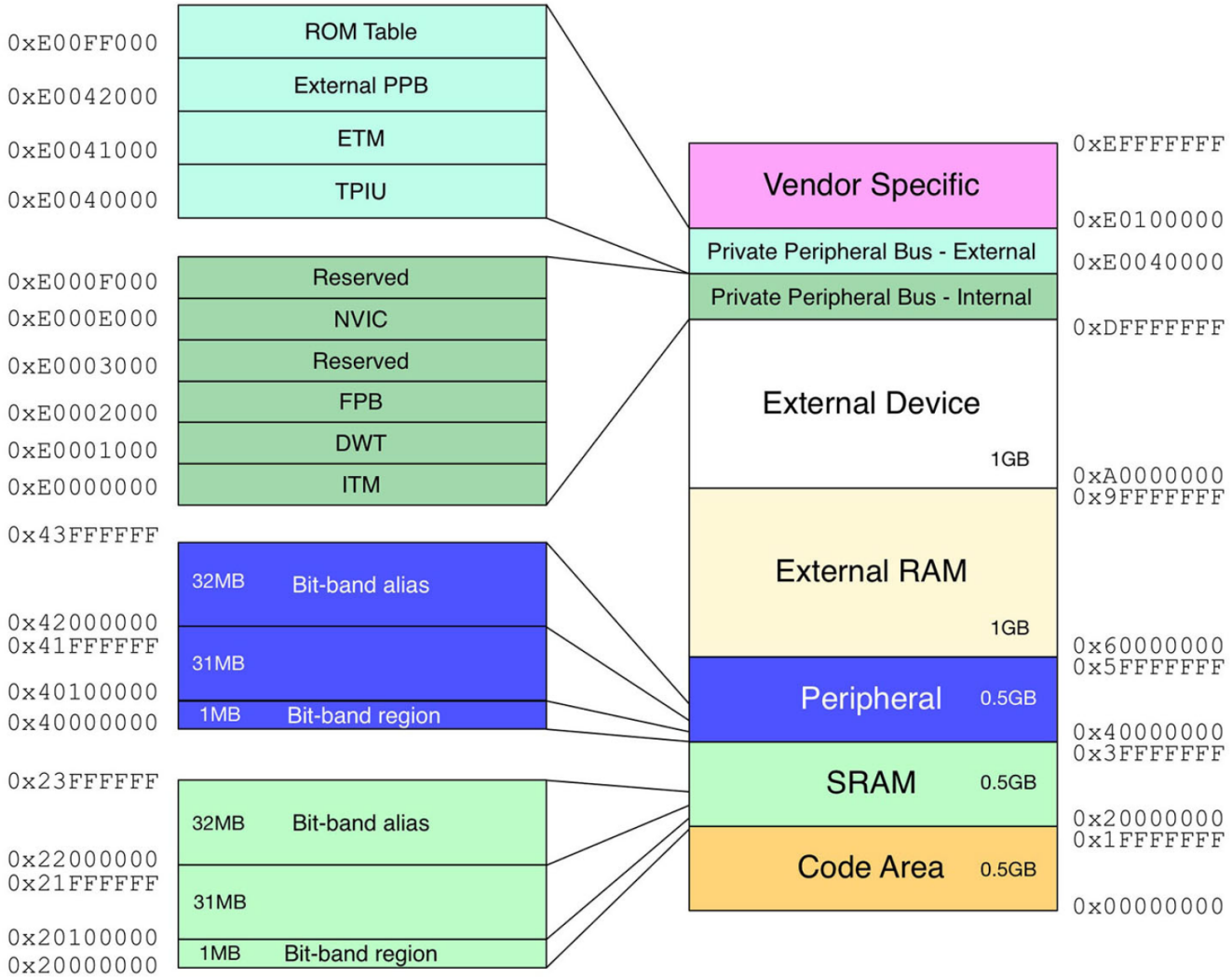
Prostsza (w stosunku do architektury Von Neumanna) budowa przekłada się na większą szybkość działania - dlatego ten typ architektury jest często wykorzystywany w procesorach sygnałowych oraz przy dostępie procesora do pamięci cache.

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.



Mapa pamięci procesorów z rodziny Cortex-M



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)

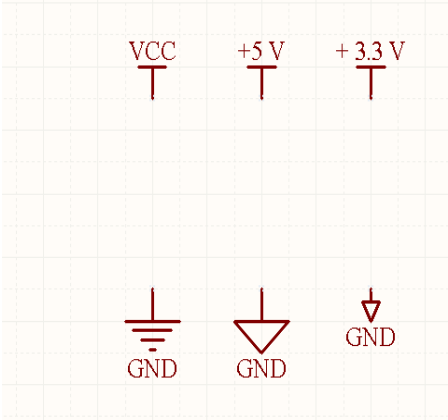
Jak rozpoznać architekturę procesora oraz rozkład bajtów w pamięci?

```
#define LITTLE_ENDIAN   0
#define BIG_ENDIAN      1

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

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

   else
      return BIG_ENDIAN;
}
```

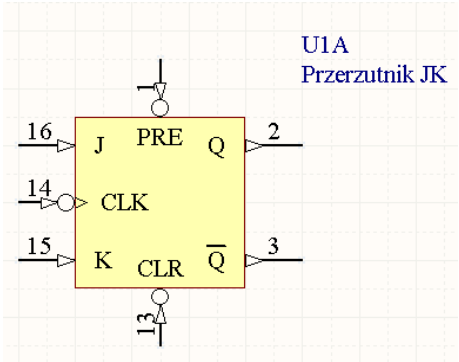
Schematy elektryczne

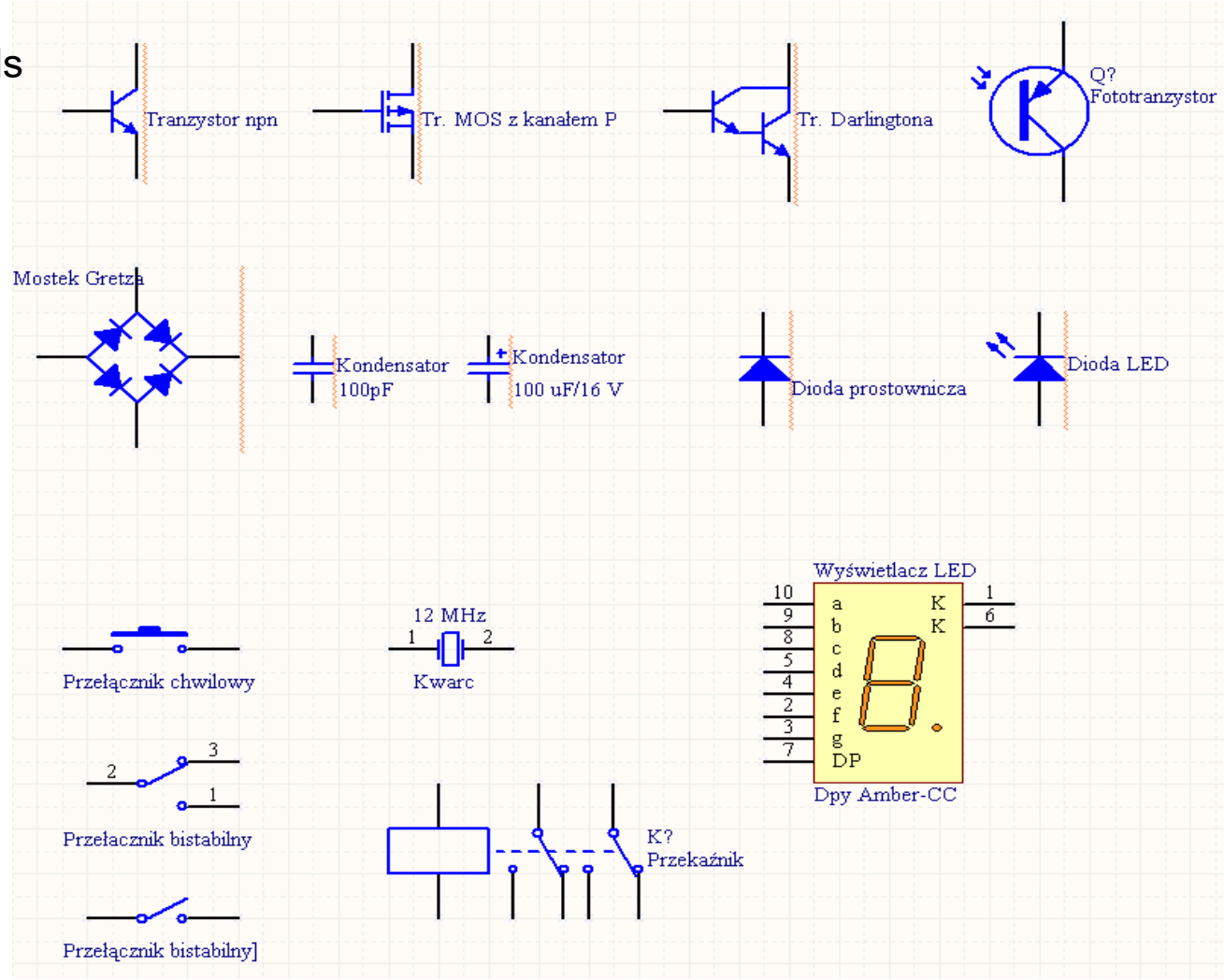
Schematic Diagrams (1)

Power Supply Bus Symbols



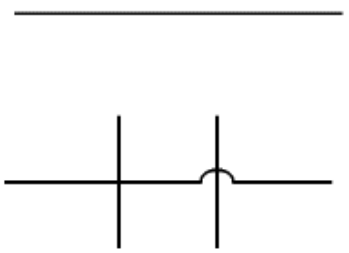
Ground Symbols

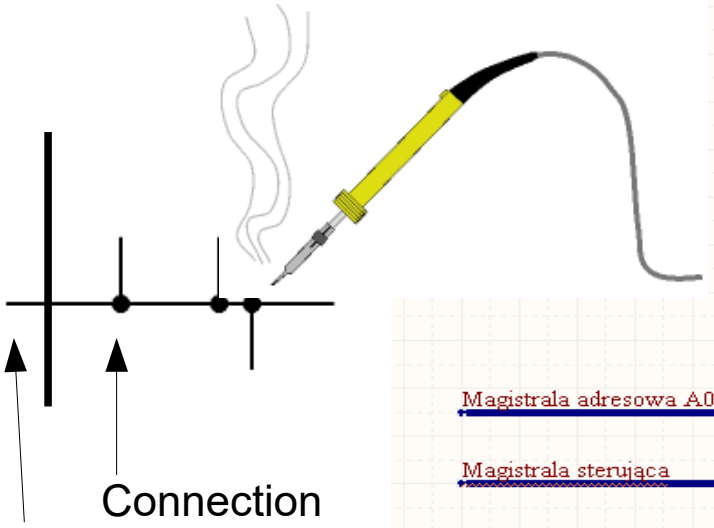




Schematic Diagrams (2)

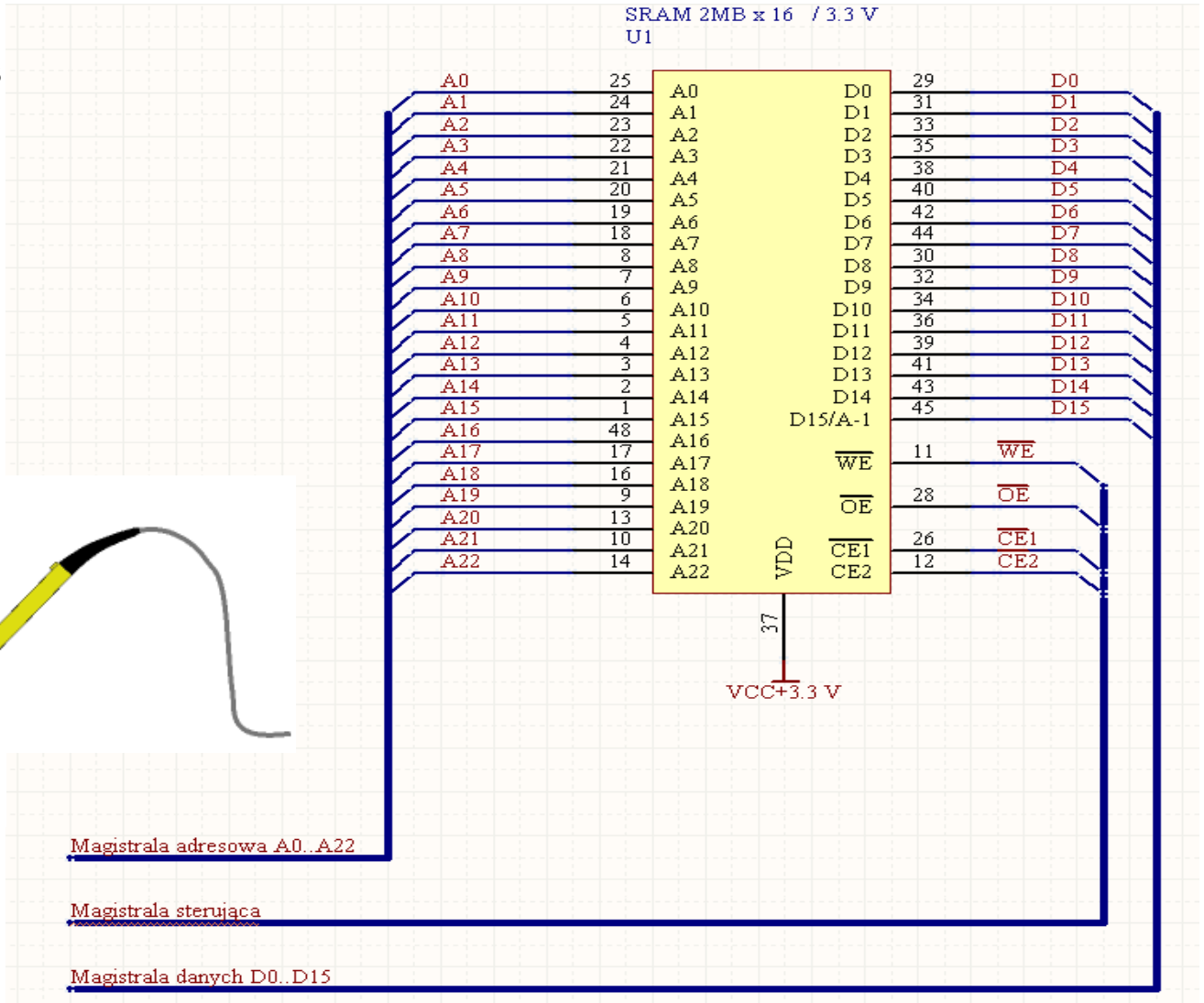
Electrical connections



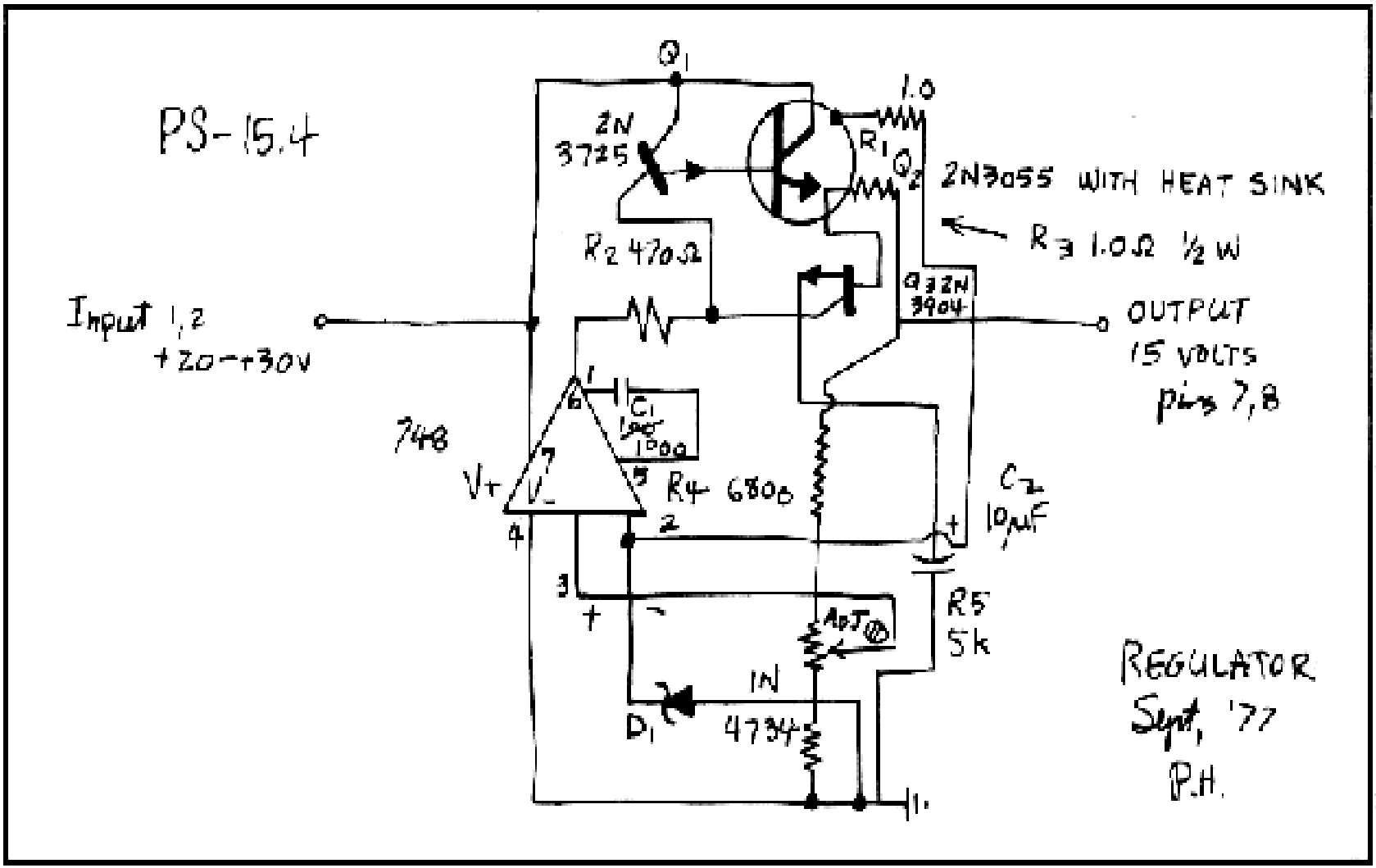


Connection

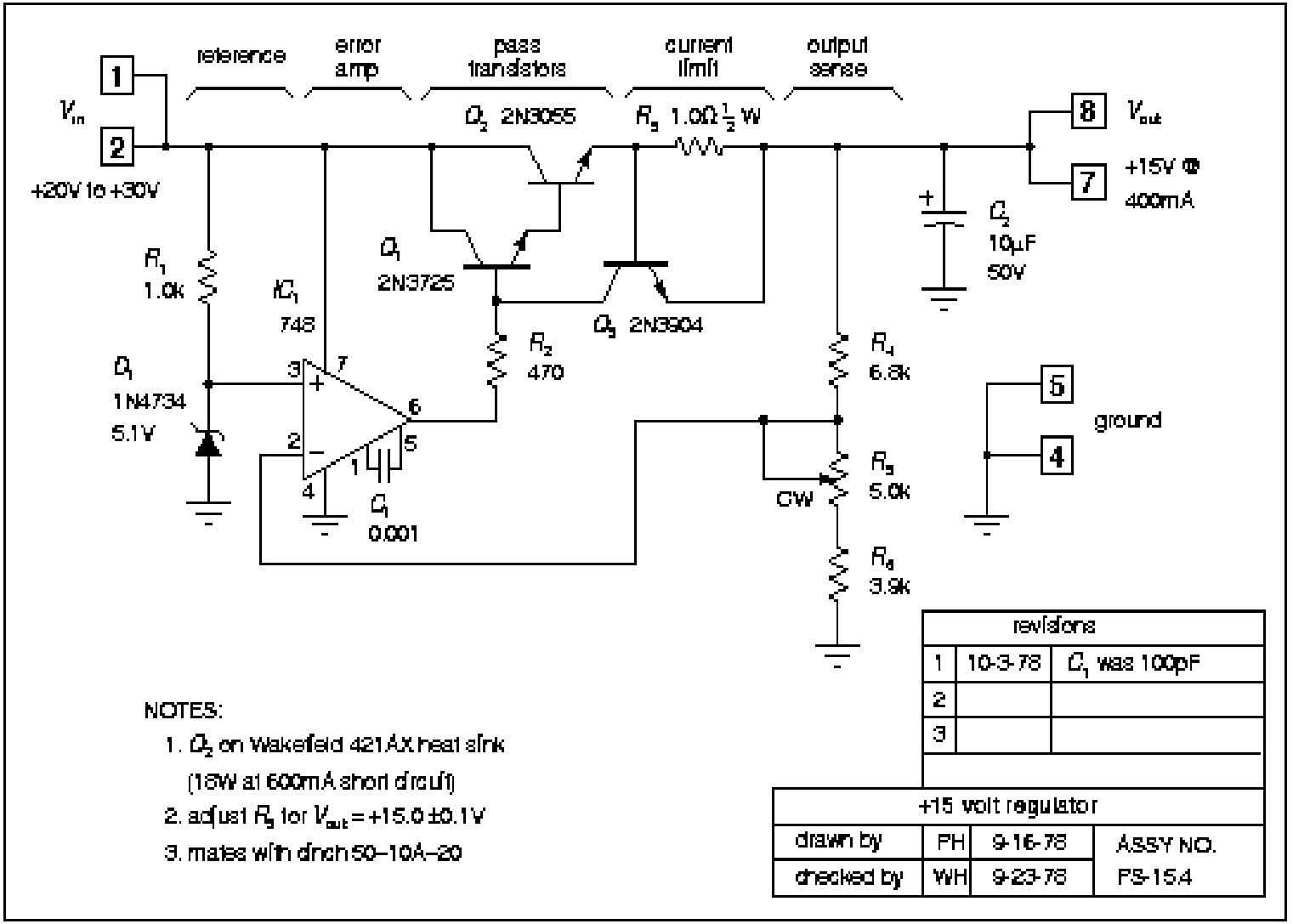
No connection



Schematic Diagram – How to Draw ?



Schematic Diagrams – Better Way



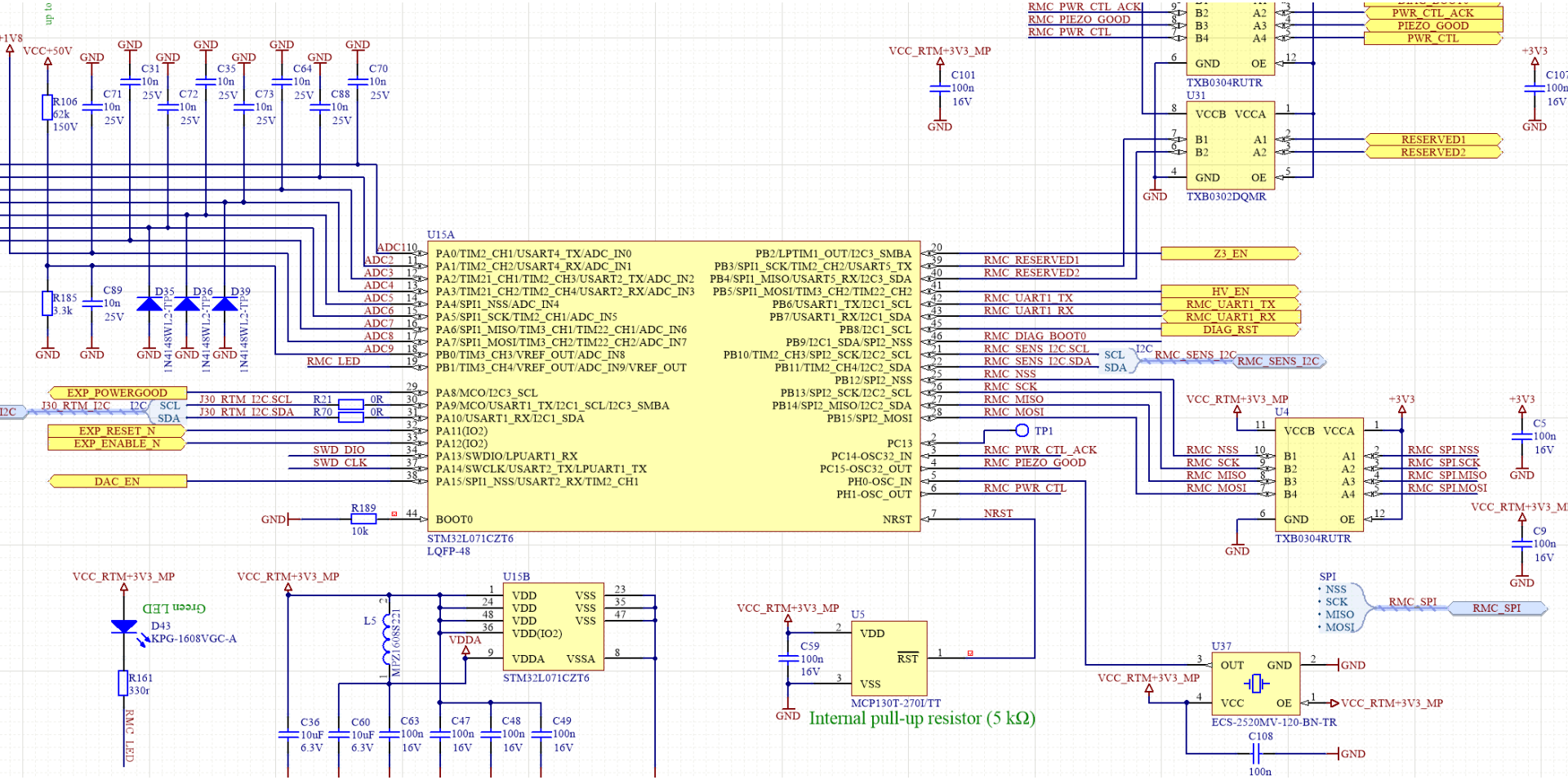
NOTES:

1. Q_2 on Wakefield 421AX heat sink (16W at 600mA short circuit)
2. adjust R_5 for V_{out} = +15.0 ±0.1V
3. mates with dinch 50–10A–20

revisions		
1	10-3-78	C_1 was 100pF
2		
3		

+15 volt regulator			
drawn by	PH	9-16-78	ASSY NO.
checked by	WH	9-23-78	PS-154

Dyskusja przykładowego schematu z mikrokontrolerem



Port wejścia-wyjścia

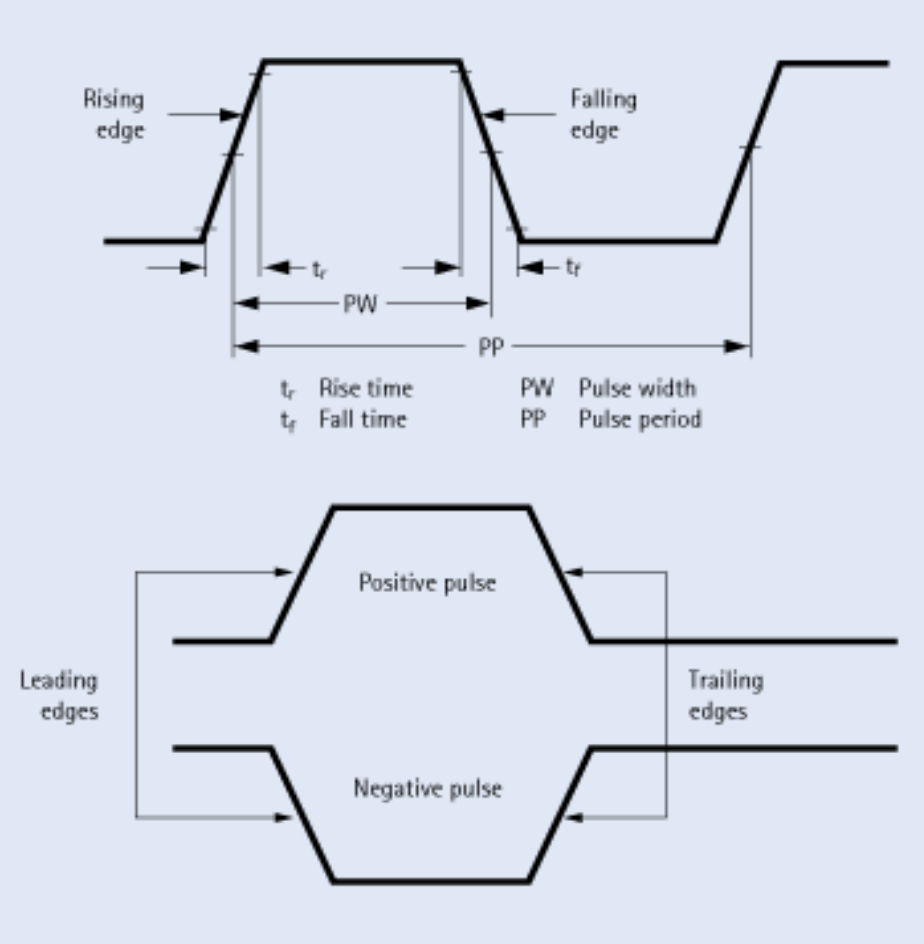
Digital Signal

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').

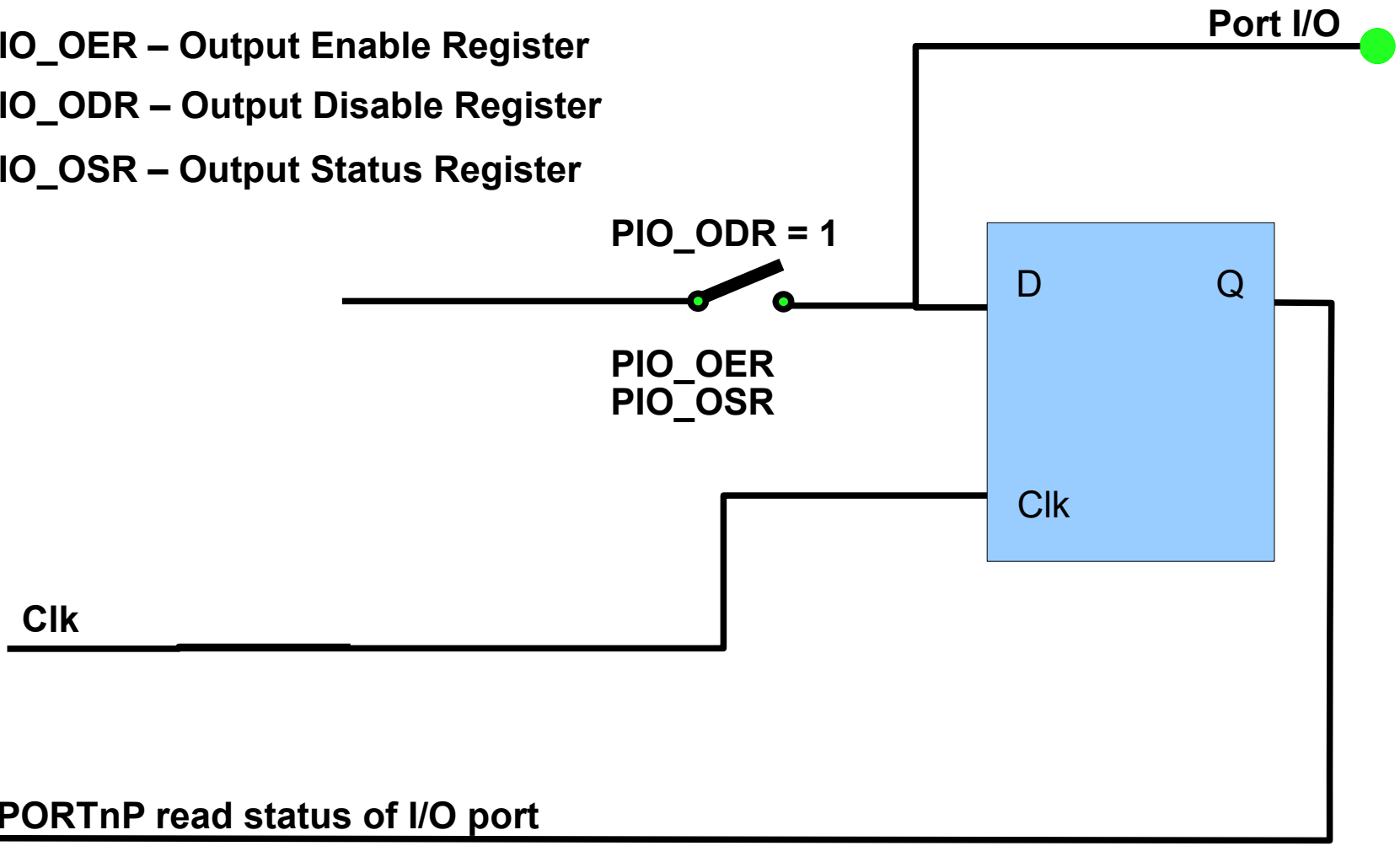


Moduł portów I/O (2)

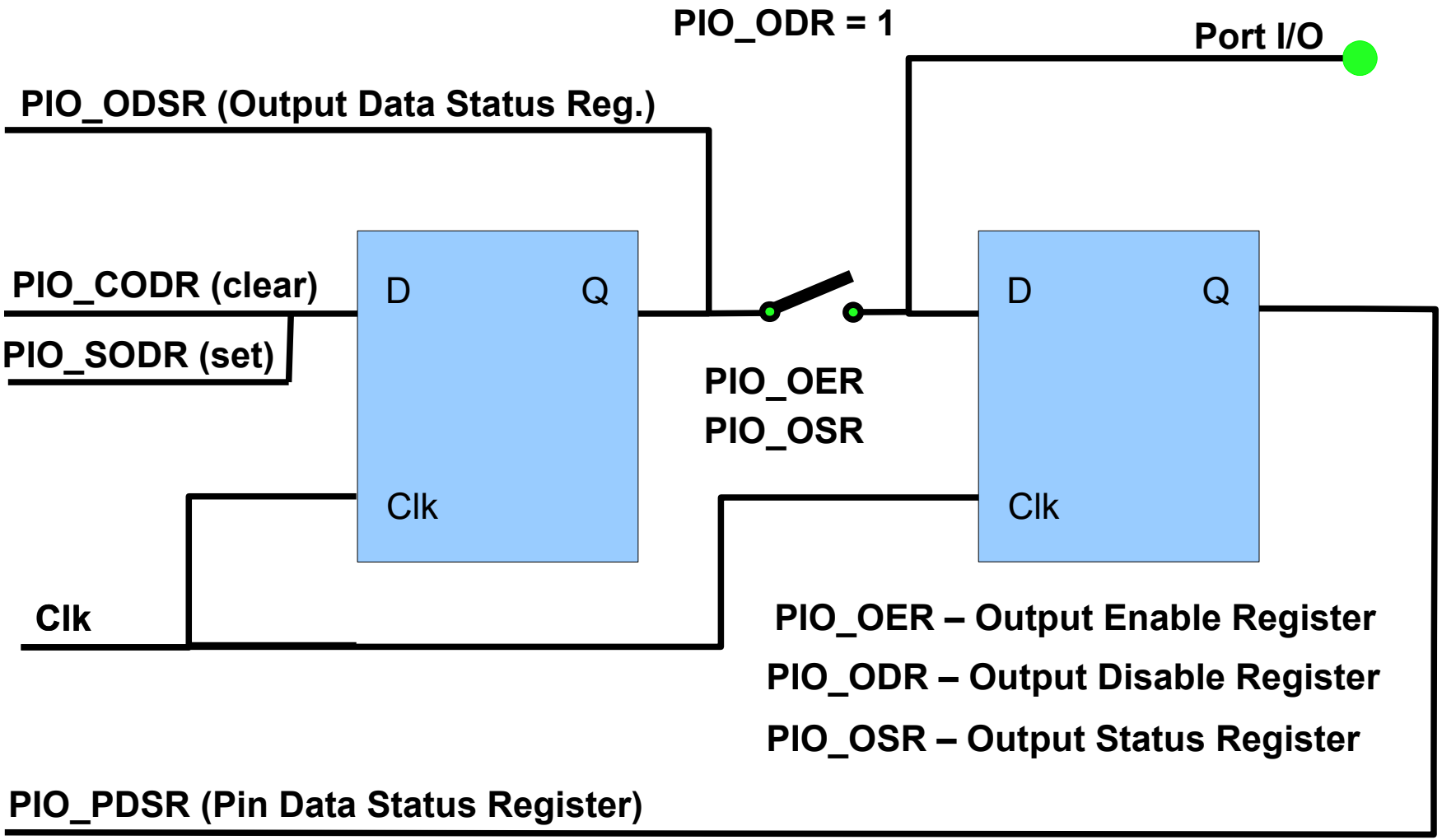
PIO_OER – Output Enable Register

PIO_ODR – Output Disable Register

PIO_OSR – Output Status Register

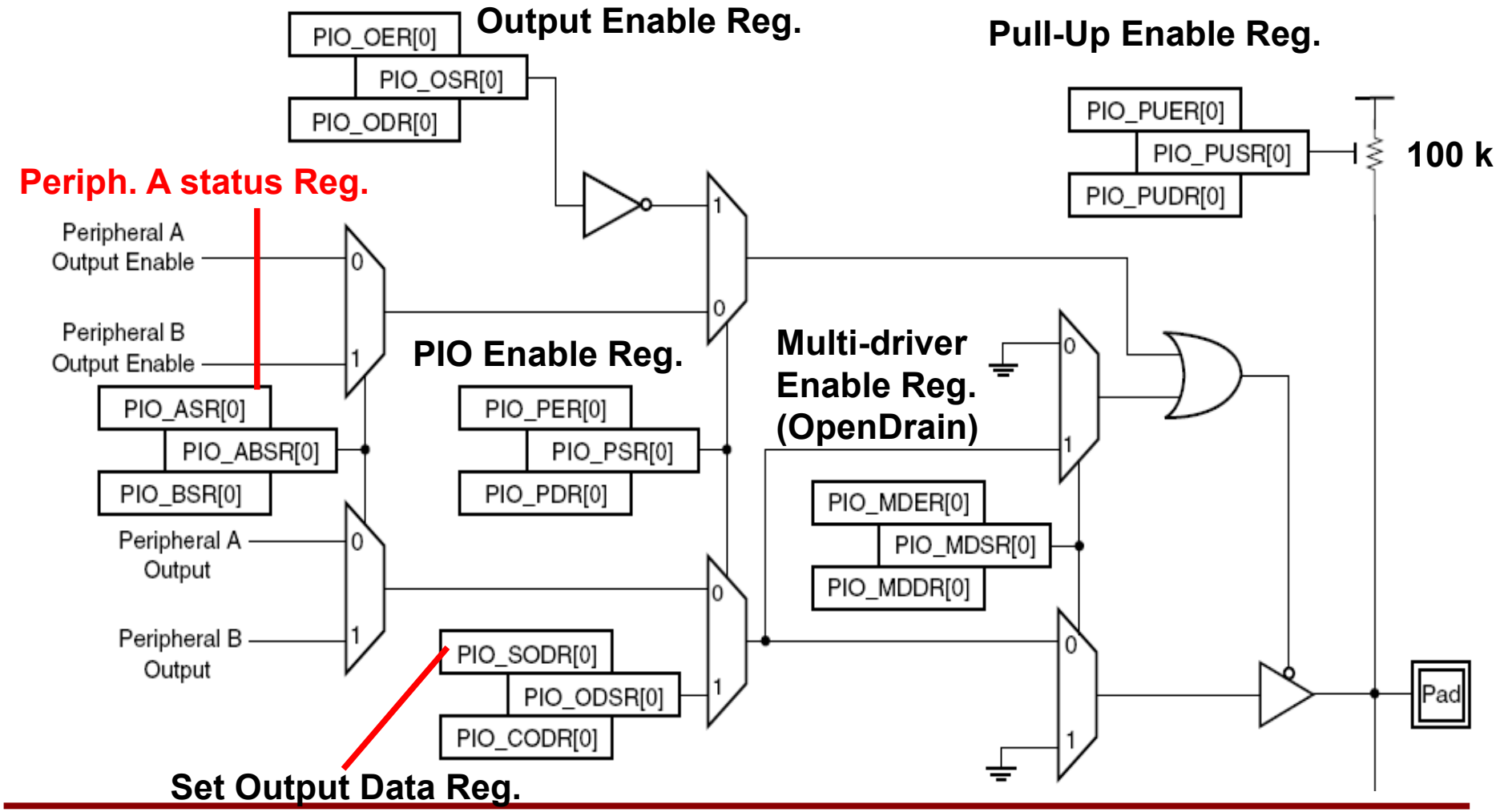


Simplified Block Diagram of I/O Port

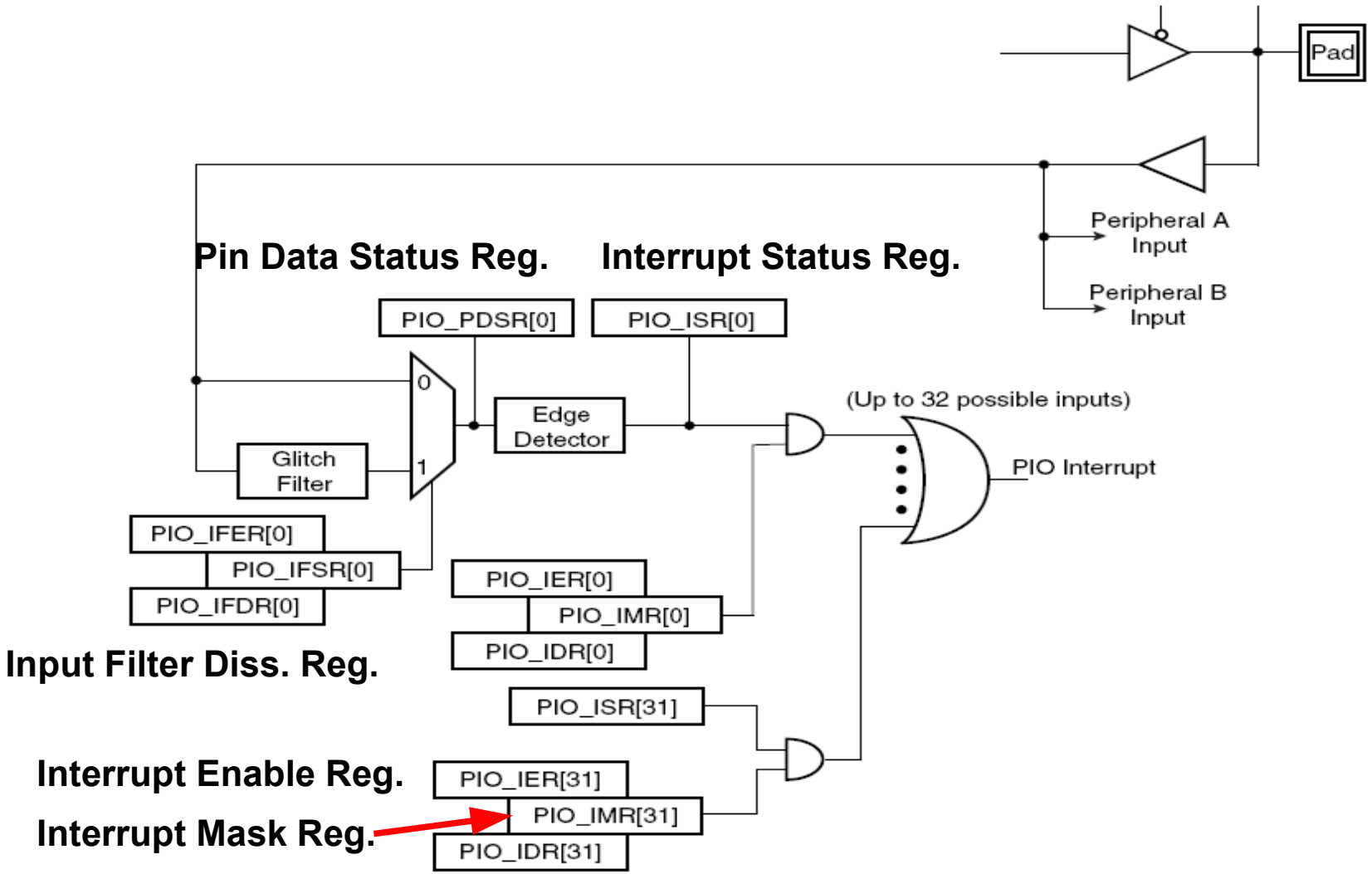


I/O Port – How to Control Output ?

Figure 31-3. I/O Line Control Logic

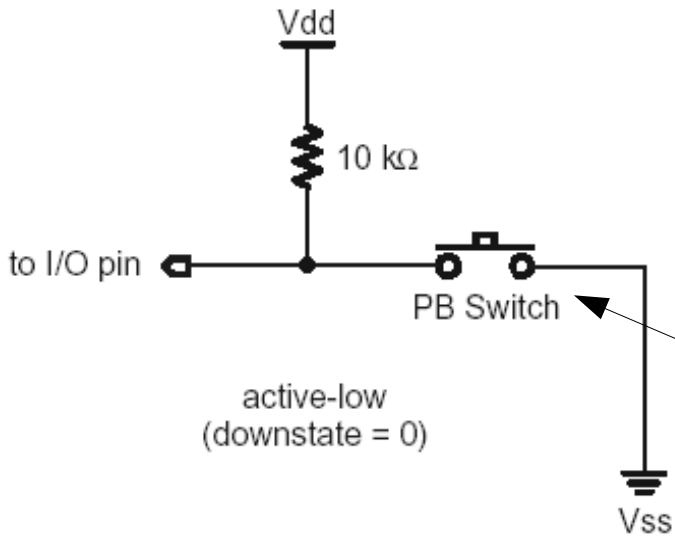


I/O – How to Read Input ?

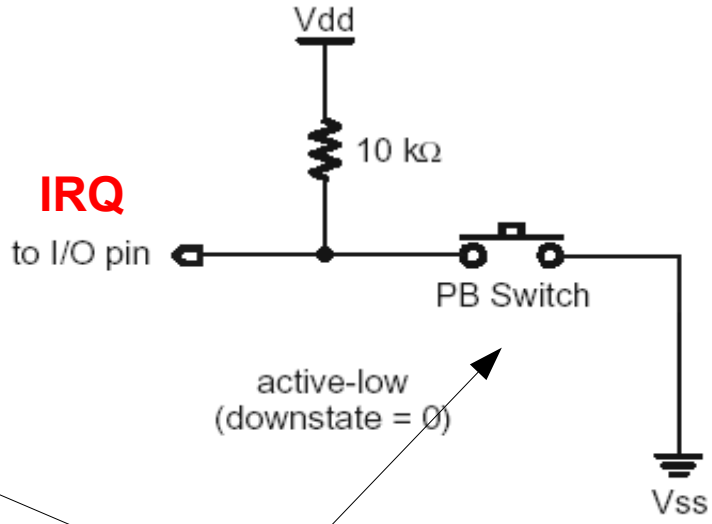


Odczyt stanu przełącznika

Polling loop



Interrupt



Sygnał asynchroniczny

Zarządzanie sygnałem zegarowym urządzeń peryferyjnych

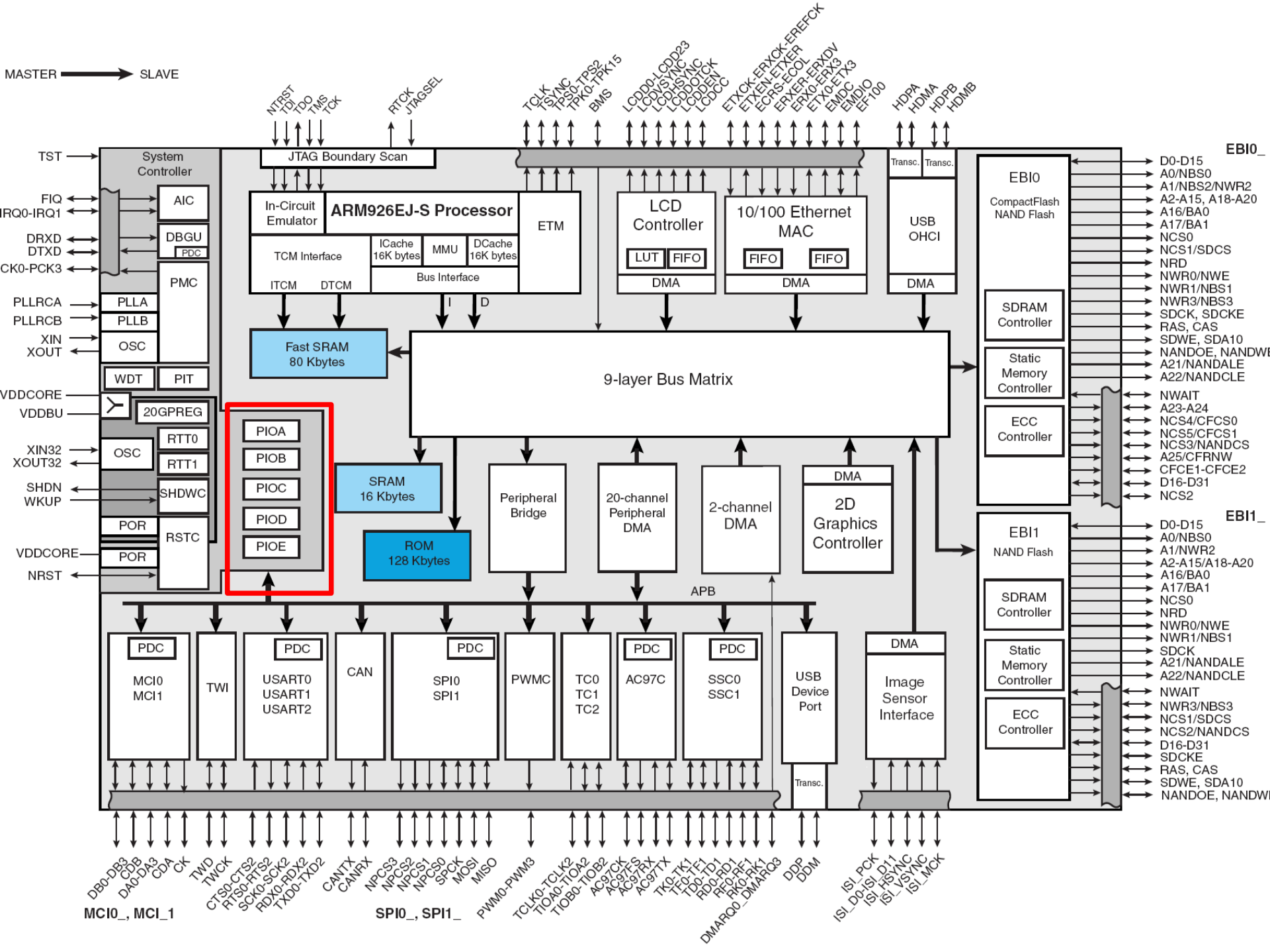
PMC Peripheral Clock Enable Register
Register Name:PMC_PCER
Address: 0xFFFFFC10

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.
```

Porty wejścia-wyjścia procesora ARM


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

Sterownik portów wejścia-wyjścia (1)

8 General-purpose I/Os (GPIO)

8.1 Introduction

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

8.2 GPIO main features

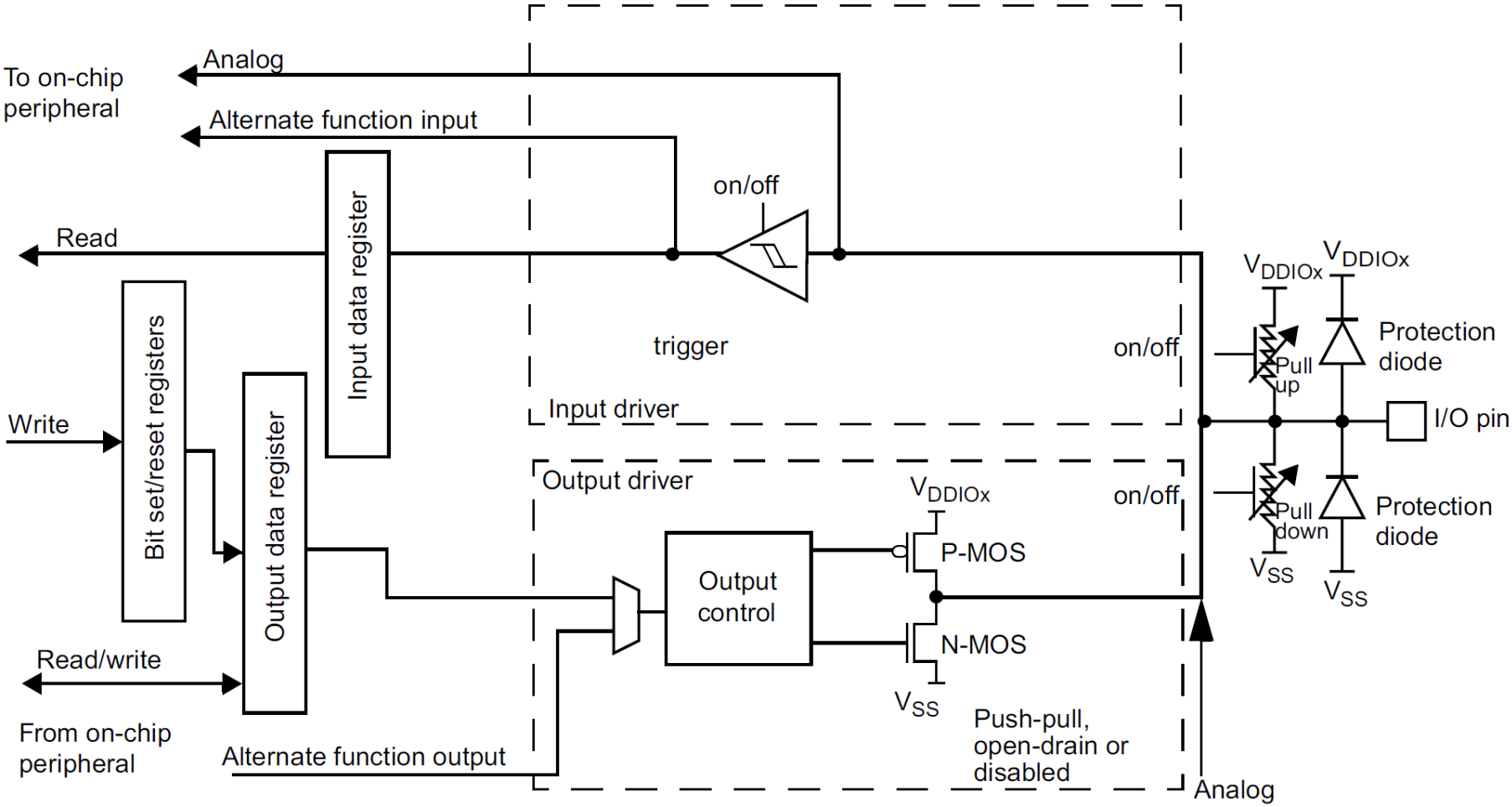
- Output states: push-pull or open drain + pull-up/down
- Output data from output data register (GPIOx_ODR) or peripheral (alternate function output)
- Speed selection for each I/O
- Input states: floating, pull-up/down, analog
- Input data to input data register (GPIOx_IDR) or peripheral (alternate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

Sterownik portów wejścia-wyjścia (2)

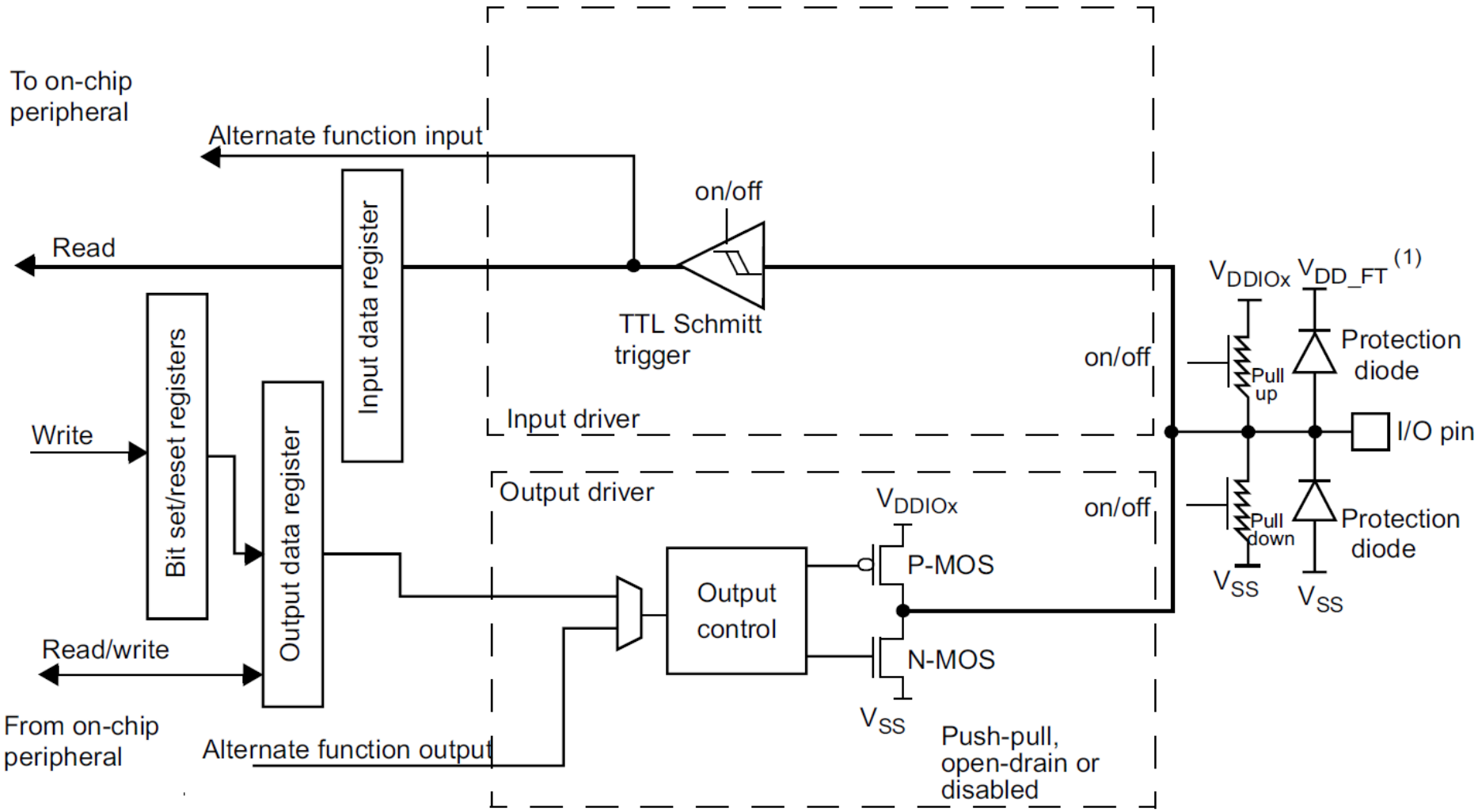
Każdy z portów IO może pracować w jednym następujących trybów:

- Wejście pływające (floating input)
- Wejście podciągane do VCC (pull-up)
- Wejście ściągane do GND (pull-down)
- Wejście analogowe
- Wyjście z otwartym drenem (z opcjonalną funkcją pull-up lub pull-down)
- Wyjście push-pull (z opcjonalną funkcją pull-up lub pull-down)
- Alternatywna funkcja z otwartym drenem (z opcjonalną funkcją pull-up lub pull-down)
- Alternatywna funkcja w trybie push-pull (z opcjonalną funkcją pull-up lub pull-down)

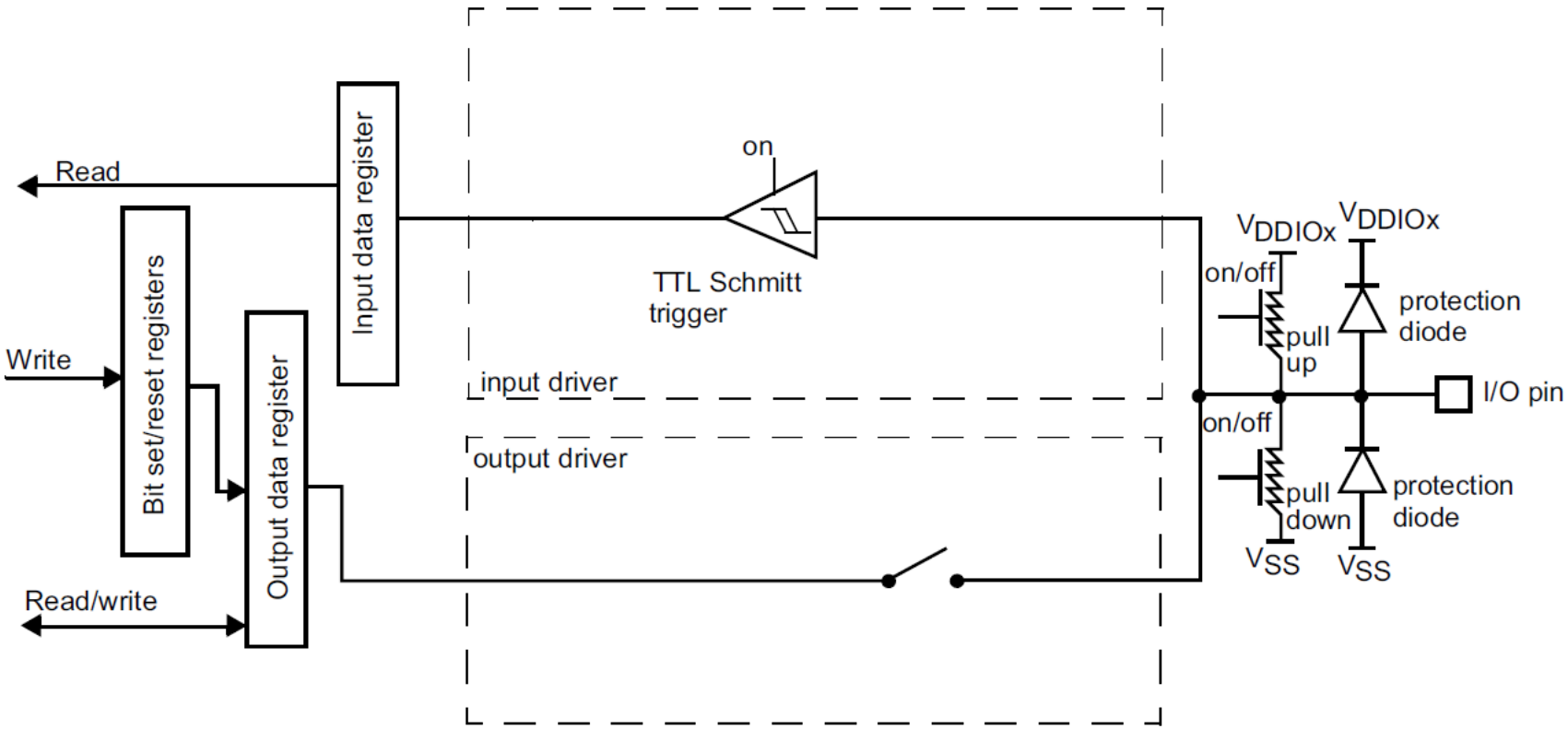
Obwody wejściowe portów IO



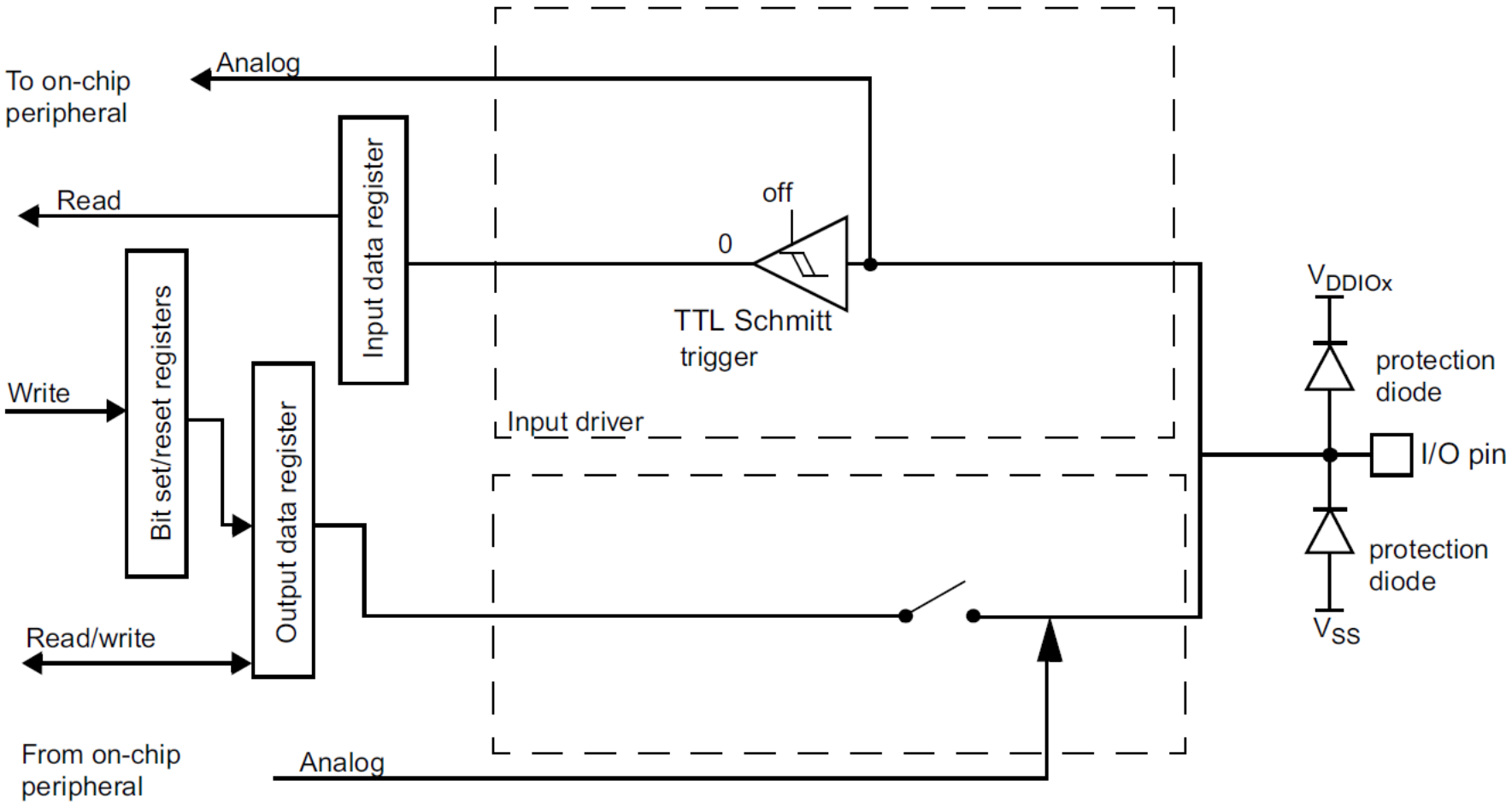
Porty akceptujące sygnały 5 V (5 V tolerant)



Porty wejściowe

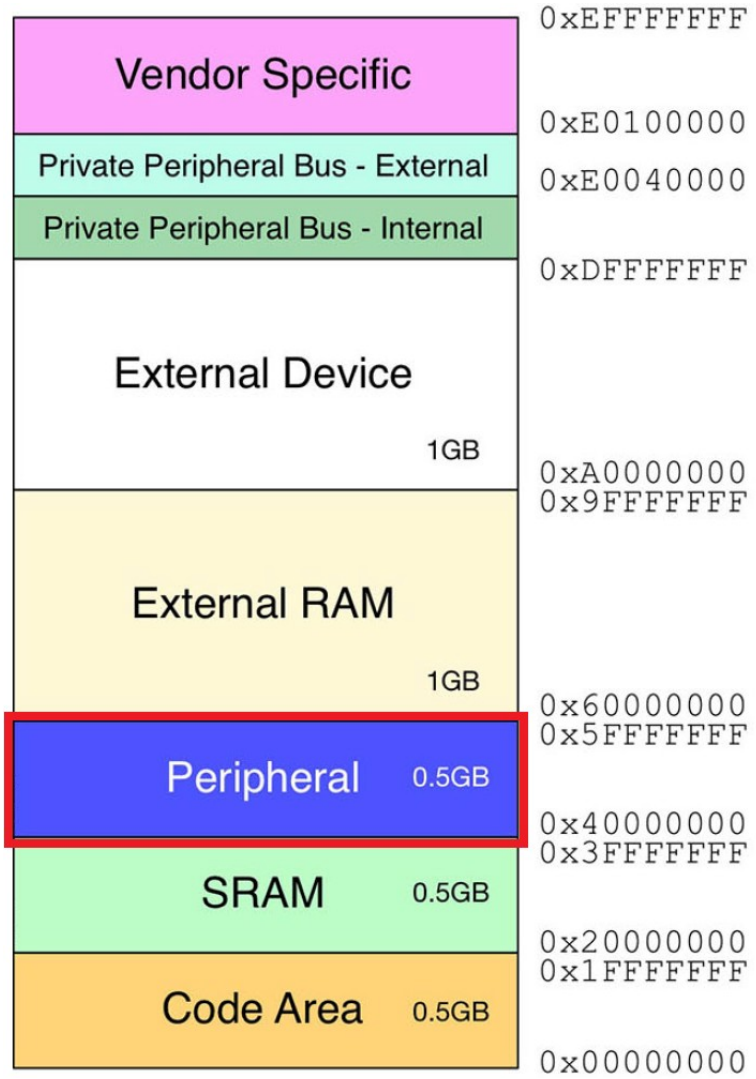


Porty analogowe



Rejestry sterujące portami IO (1)

- Procesor posiada 8 (porty A-H) 16-bitowych portów wejścia-wyjścia
- Obsługa do 128 sygnałów IO
- Każdy z portów sterowany jest przy pomocy 12 32-bitowych rejestrów umieszczonych w obszarze 1 kB pamięci
- Rejestry dostępna są głównie w trybie RW, poza wyjątkami:
  - IDR – tylko odczyt
  - BSRR/BRR – tylko zapis (bitwise control)



Rejestry sterujące portami IO (2)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOA_MODER	MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]		MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
	Reset value	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x04	GPIOx_OTYPER (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	GPIOA_OSPEEDR	OSPEED15[1:0]		OSPEED14[1:0]		OSPEED13[1:0]		OSPEED12[1:0]		OSPEED11[1:0]		OSPEED10[1:0]		OSPEED9[1:0]		OSPEED8[1:0]		OSPEED7[1:0]		OSPEED6[1:0]		OSPEED5[1:0]		OSPEED4[1:0]		OSPEED3[1:0]		OSPEED2[1:0]		OSPEED1[1:0]		OSPEED0[1:0]	
	Reset value	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOA_PUPDR	PUPD15[1:0]		PUPD14[1:0]		PUPD13[1:0]		PUPD12[1:0]		PUPD11[1:0]		PUPD10[1:0]		PUPD9[1:0]		PUPD8[1:0]		PUPD7[1:0]		PUPD6[1:0]		PUPD5[1:0]		PUPD4[1:0]		PUPD3[1:0]		PUPD2[1:0]		PUPD1[1:0]		PUPD0[1:0]	
	Reset value	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_IDR (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
	Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Rejestry sterujące portami IO (3)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x14	**GPIOx_ODR** (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**GPIOx_BSRR** (where x = A..I)	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	**GPIOx_LCKR** (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LCKK	LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	**GPIOx_AFRL** (where x = A..I)	AFSEL7[3:0]				AFSEL6[3:0]				AFSEL5[3:0]				AFSEL4[3:0]				AFSEL3[3:0]				AFSEL2[3:0]				AFSEL1[3:0]				AFSEL0[3:0]			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	**GPIOx_AFRH** (where x = A..I)	AFSEL15[3:0]				AFSEL14[3:0]				AFSEL13[3:0]				AFSEL12[3:0]				AFSEL11[3:0]				AFSEL10[3:0]				AFSEL9[3:0]				AFSEL8[3:0]			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	**GPIOx_BRR** (where x = A..I)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
	Reset value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	**GPIOx_ASCR** (where x = A..H)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ASC15	ASC14	ASC13	ASC12	ASC11	ASC10	ASC9	ASC8	ASC7	ASC6	ASC5	ASC4	ASC3	ASC2	ASC1	ASC0
	Reset value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Adresy bazowe portów A-H

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

Operacje na rejestrach

Operacje na pamięci w języku C

```
volatile unsigned int * DataInMemory = 0x1000;
    *DataInMemory = 0;
    *DataInMemory = 16789
    *DataInMemory = 0xFFFF.FFFF;
```

Jak wyzerować pojedynczy bit ?

Jak ustawić pojedynczy bit ?

Operacje na rejestrach w języku C

```
volatile unsigned char* PORTA = (volatile unsigned char*) 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   1U<<2
#define PB3   1U<<3

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

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

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

Operacje na rejestrach w języku C (3)

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

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

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

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

Register Concatenation

```
int main(void) {
        unsigned char reg1=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;

}
```

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
        }
        /* 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);
```

Struktura Rejestrów IO dla STM32L

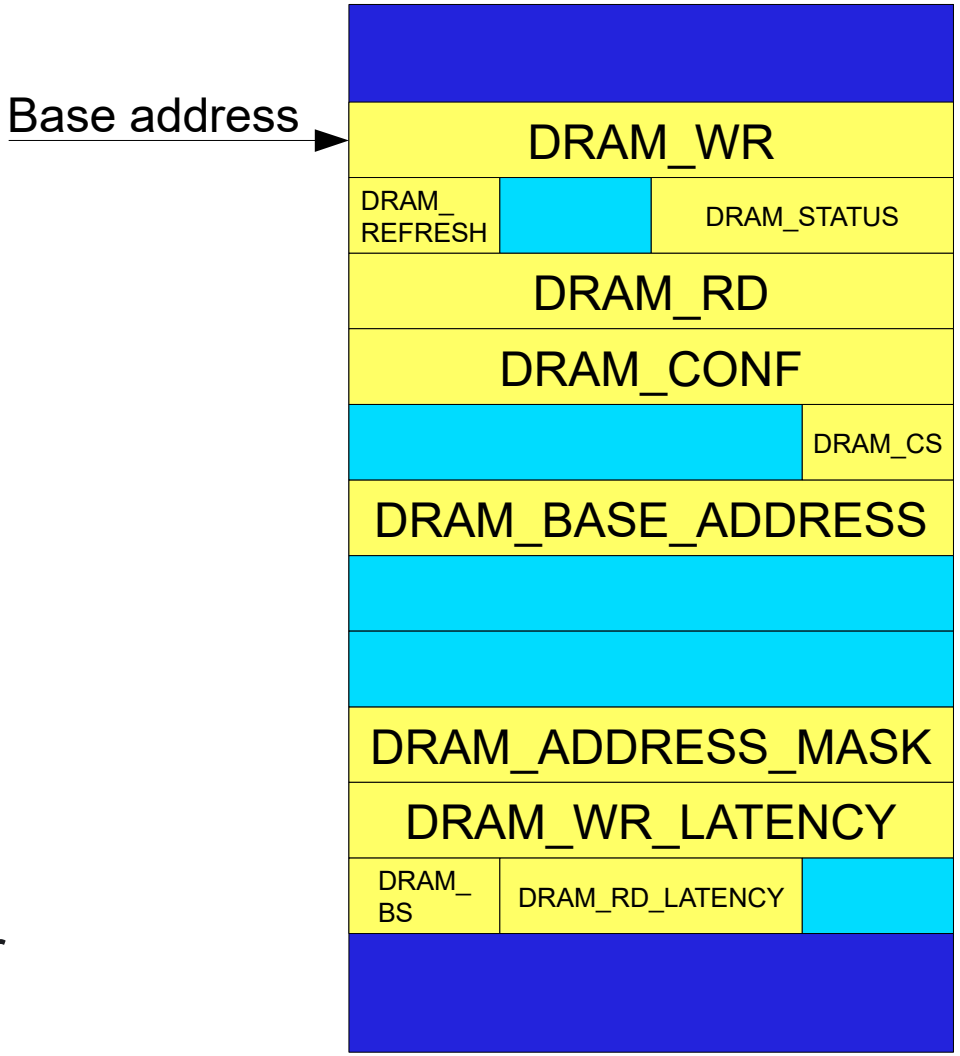
```
typedef struct
{
  __IO uint32_t MODER;  /*!< GPIO port mode register,               Address offset: 0x00      */
  __IO uint32_t OTYPER;      /*!< GPIO port output type register,        Address offset: 0x04      */
  __IO uint32_t OSPEEDR;     /*!< GPIO port output speed register,       Address offset: 0x08      */
  __IO uint32_t PUPDR;       /*!< GPIO port pull-up/pull-down register,  Address offset: 0x0C      */
  __IO uint32_t IDR;         /*!< GPIO port input data register,         Address offset: 0x10      */
  __IO uint32_t ODR;         /*!< GPIO port output data register,        Address offset: 0x14      */
  __IO uint32_t BSRR;        /*!< GPIO port bit set/reset  register,     Address offset: 0x18      */
  __IO uint32_t LCKR;        /*!< GPIO port configuration lock register, Address offset: 0x1C      */
  __IO uint32_t AFR[2];      /*!< GPIO alternate function registers,     Address offset: 0x20-0x24 */
  __IO uint32_t BRR;         /*!< GPIO Bit Reset register,               Address offset: 0x28      */

} GPIO_TypeDef;
```

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)



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;
```

Narzędzia
STM 32

STM32 IDE

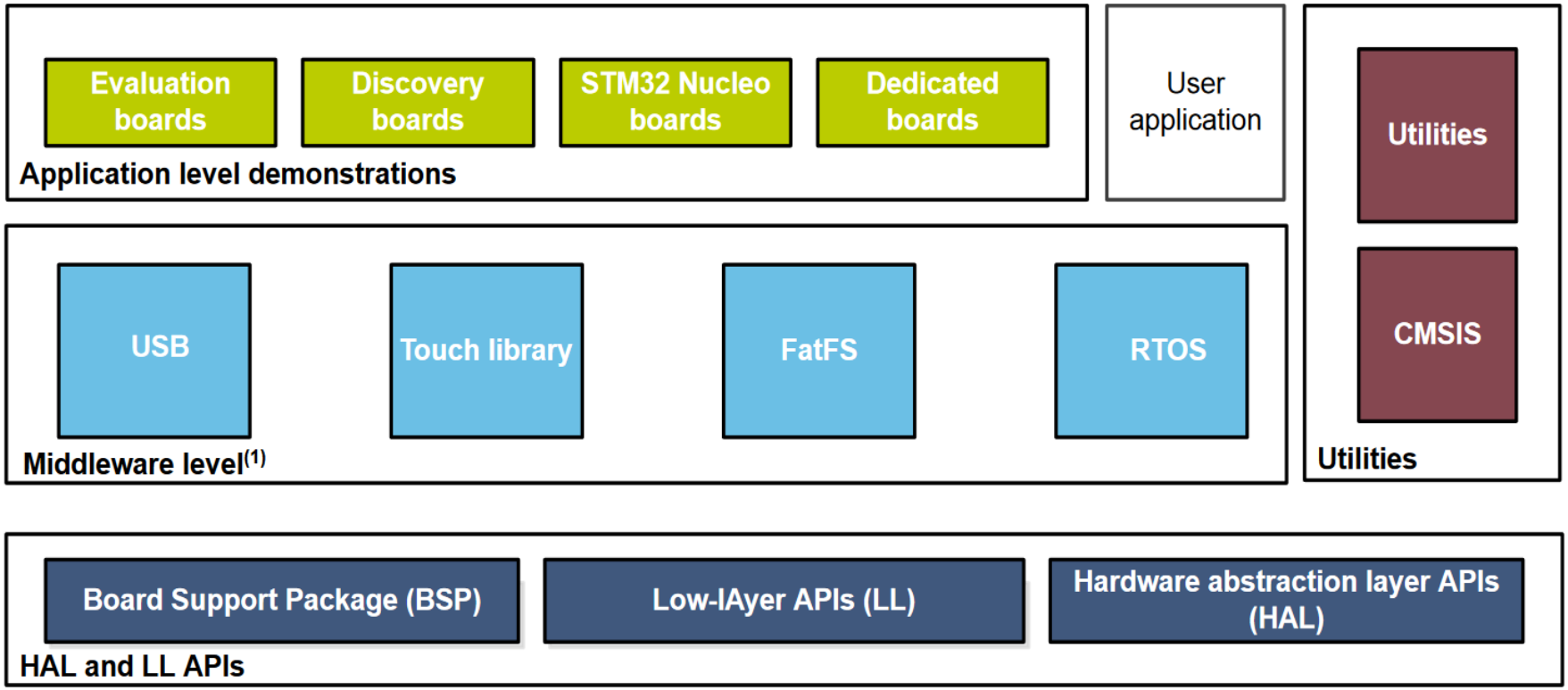
- **Procesor z rdzeniem ARM Cortex M4 firmy STM:** STM32L496ZGT6
- **Cube MX IDE**

  https://my.st.com/content/my_st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-ides/stm32cubeide.html#get-software
- https://www.st.com/content/ccc/resource/technical/document/user_manual/10/c5/1a/43/3a/70/43/7d/DM00104712.pdf/files/DM00104712.pdf/jcr:content/translations/en.DM00104712.pdf
- **HAL**

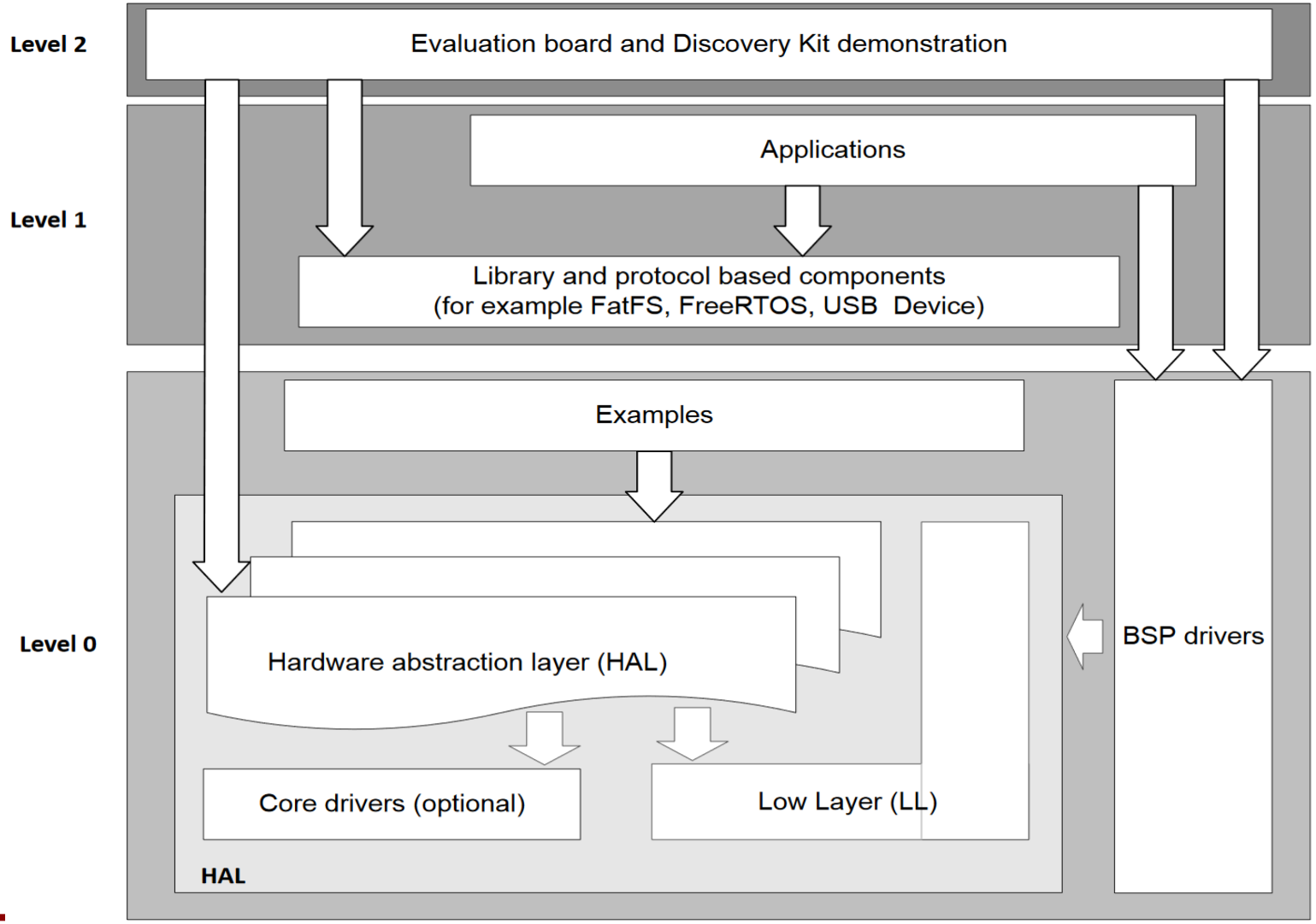
  https://www.st.com/content/ccc/resource/technical/document/user_manual/41/78/8e/63/f9/32/44/12/DM00113898.pdf/files/DM00113898.pdf/jcr:content/translations/en.DM00113898.pdf

  https://www.st.com/content/ccc/resource/technical/document/user_manual/e0/bc/bf/94/24/99/49/81/DM00114438.pdf/files/DM00114438.pdf/jcr:content/translations/en.DM00114438.pdf

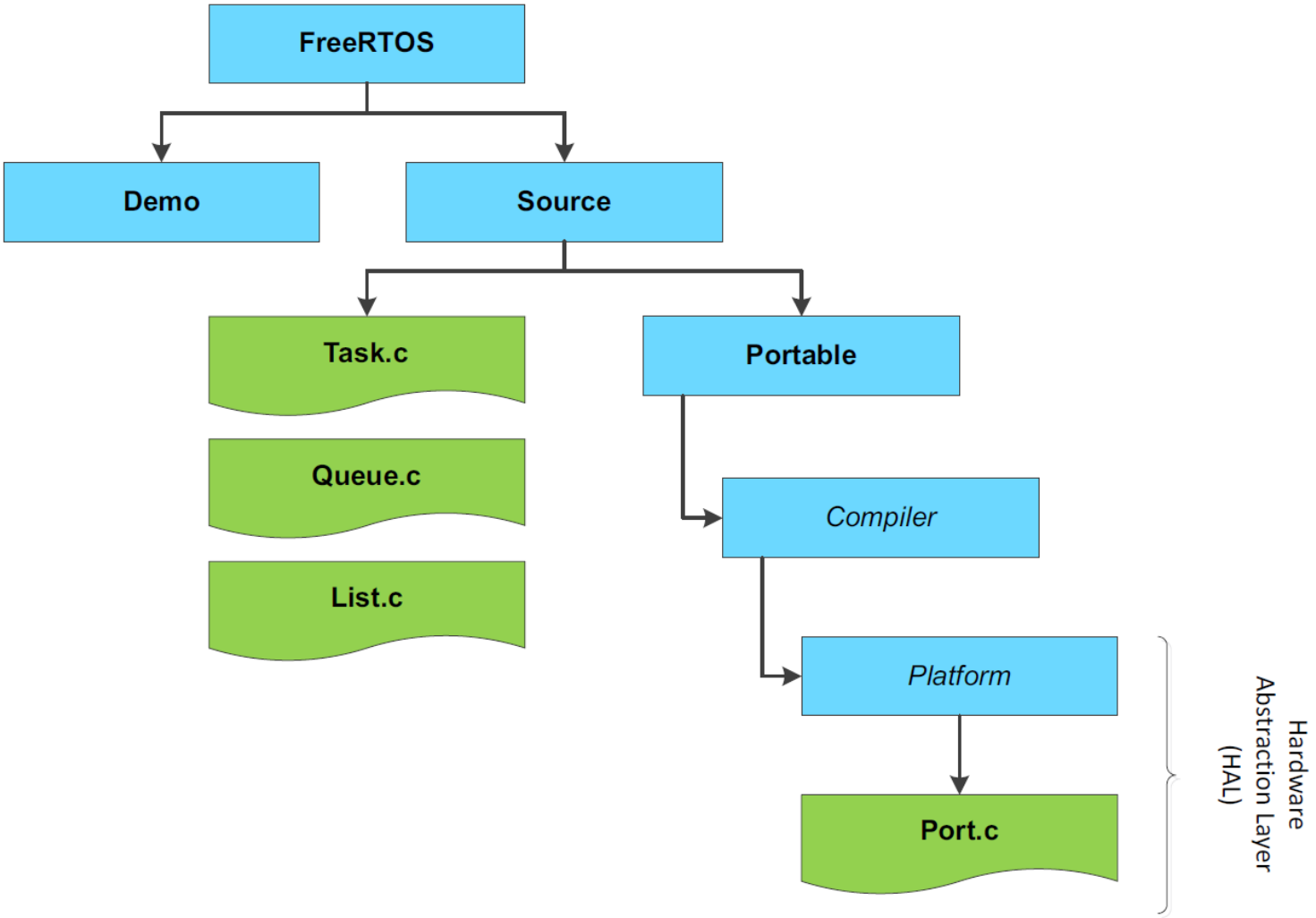
STM32 L0 Software



Architektura oprogramowania STM32CubeL0



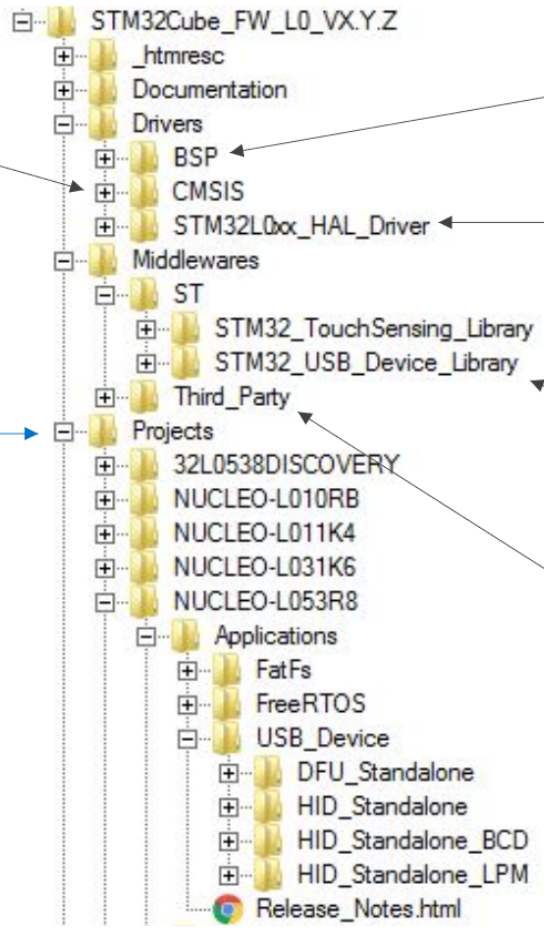
FreeRTOS on STM32



Struktura pakietu STM32CubeL0

Contains STM32L0xx CMSIS files that define peripheral's registers declarations, bits definition and the address mapping

Set of examples, applications and demonstrations organized by board and provided with preconfigured projects



BSP drivers for the supported boards:
- Evaluation board
- Discovery kit

STM32 HAL and LL drivers

Touch Sensing Library

USB Device Library offering the following classes: HID, MSC, CDC and DFU

Open source Middleware stacks