Dariusz Makowski

Department of Microelectronics and Computer Science

tel. 631 2720

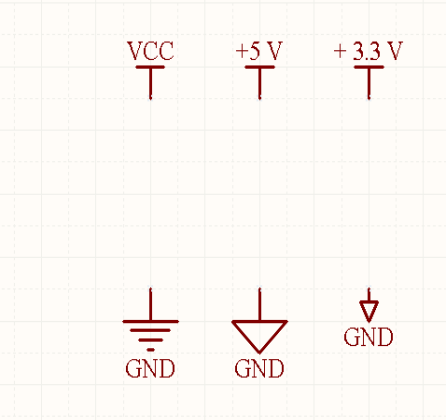
dmakow@dmcs.pl

http://neo.dmcs.pl/es

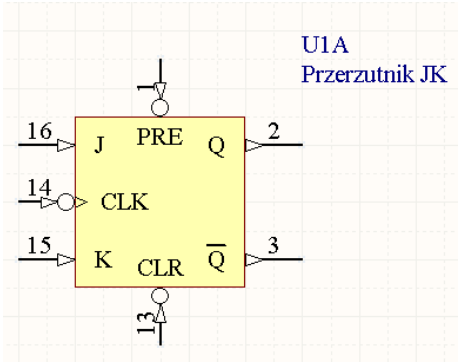
Schematic Diagrams

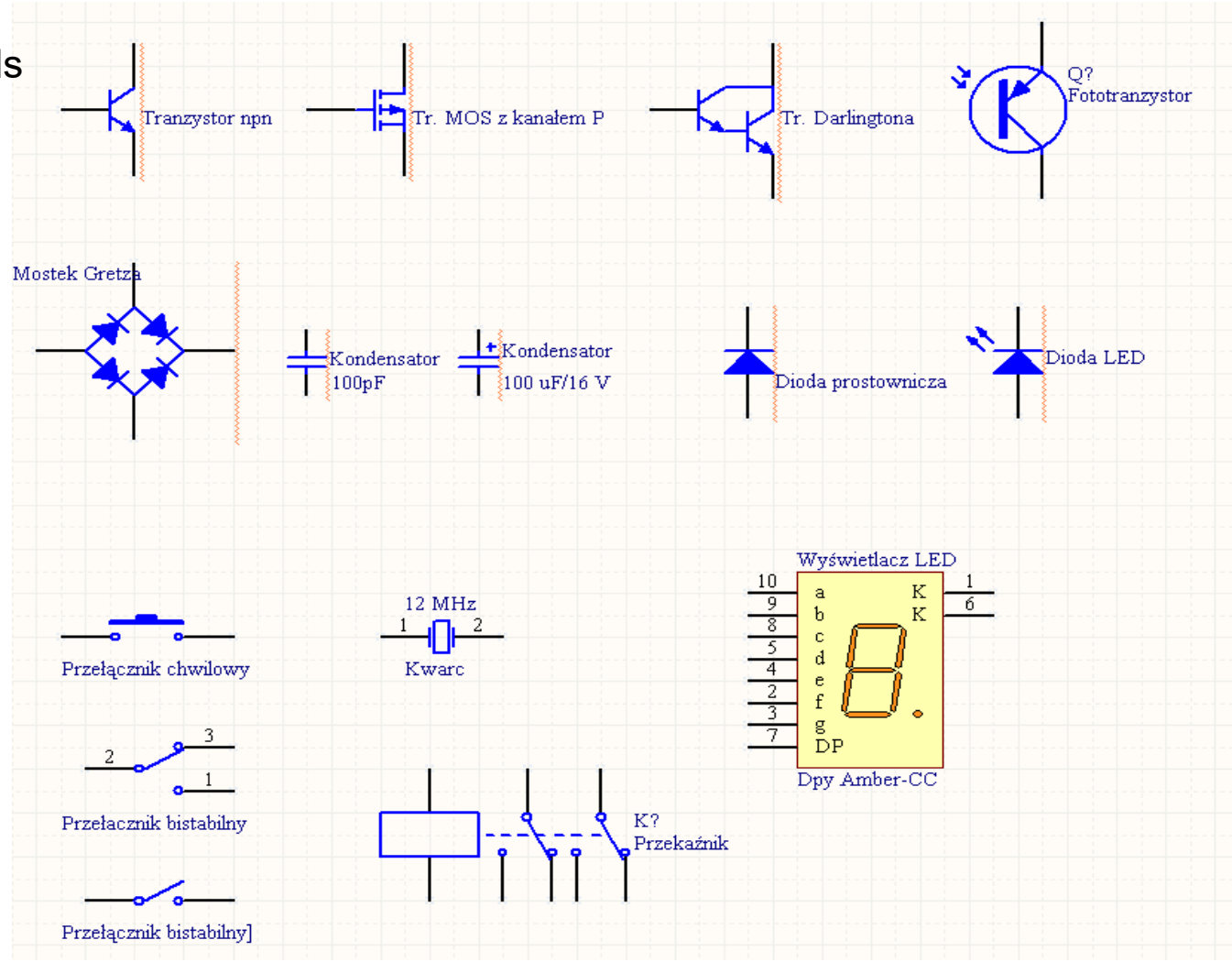
Schematic Diagrams (1)

Power Supply Bus Symbols



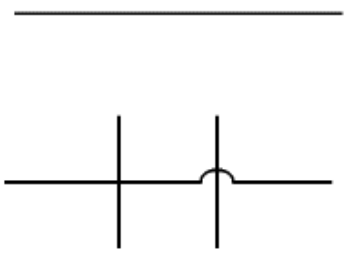
Ground Symbols

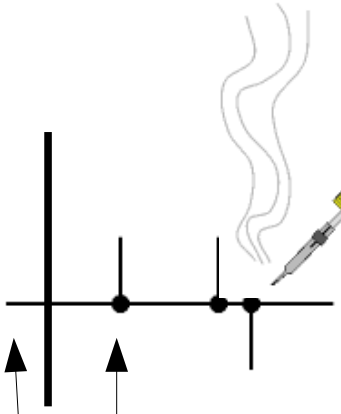




Schematic Diagrams (2)

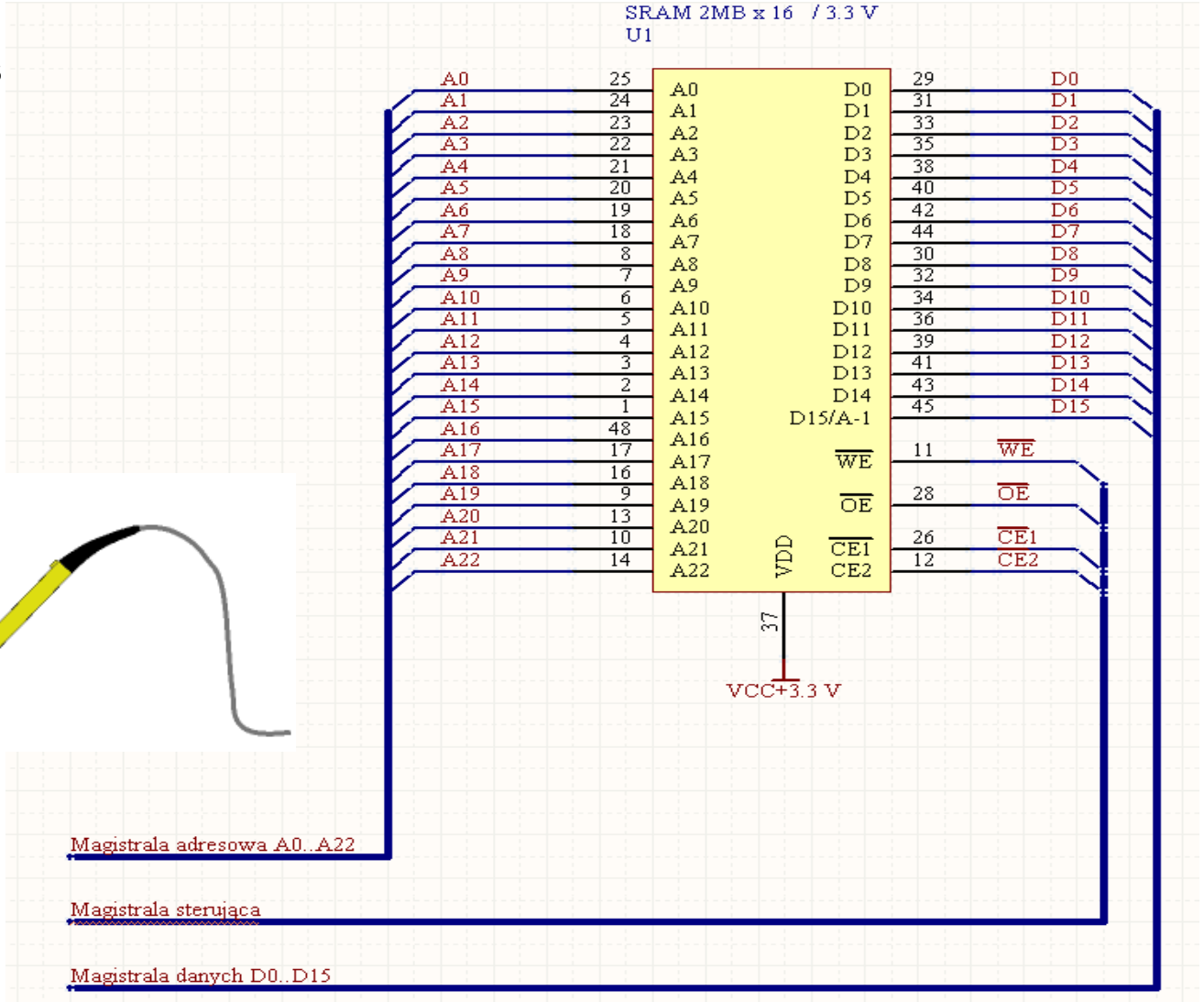
Electrical connections



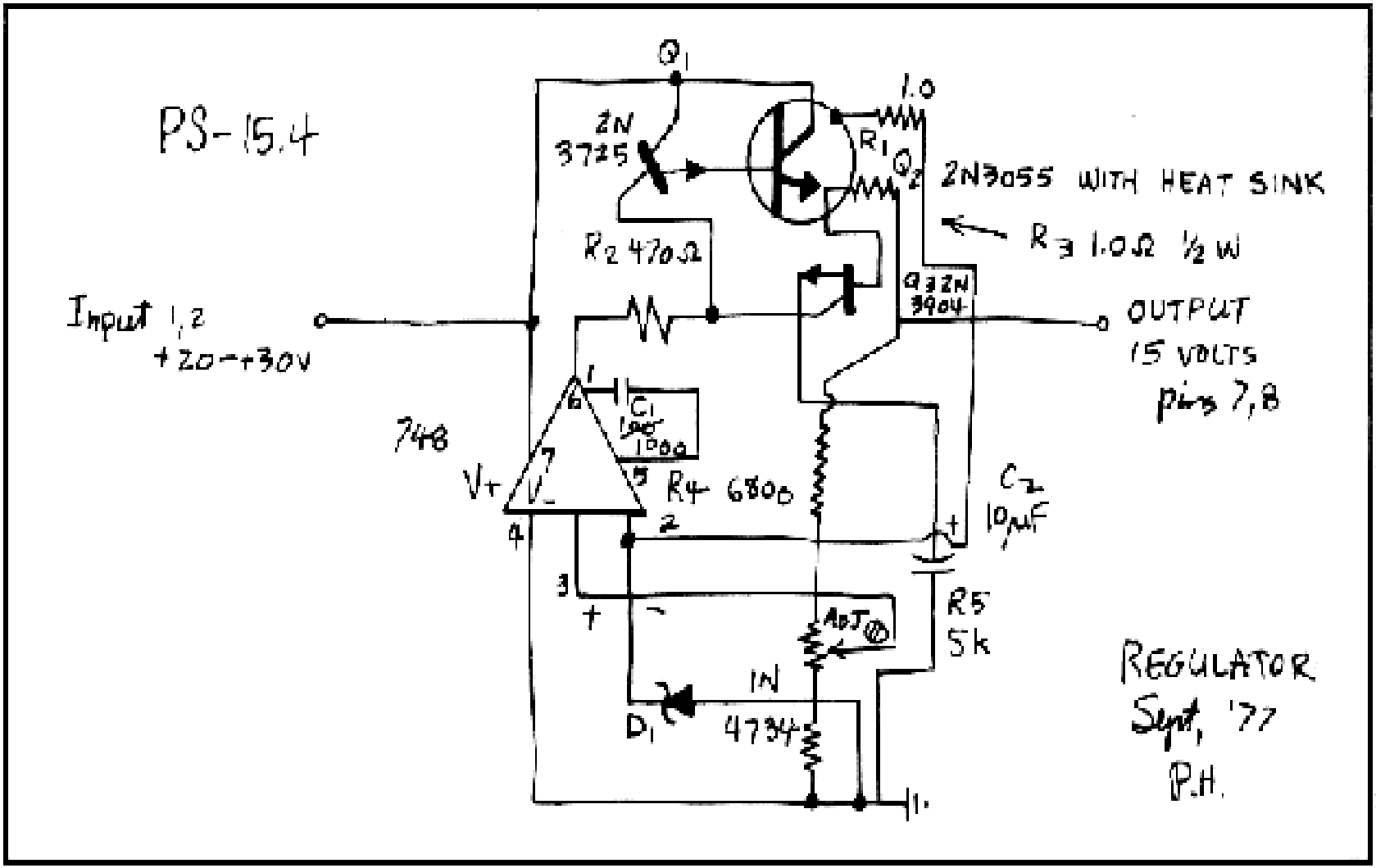


Connection

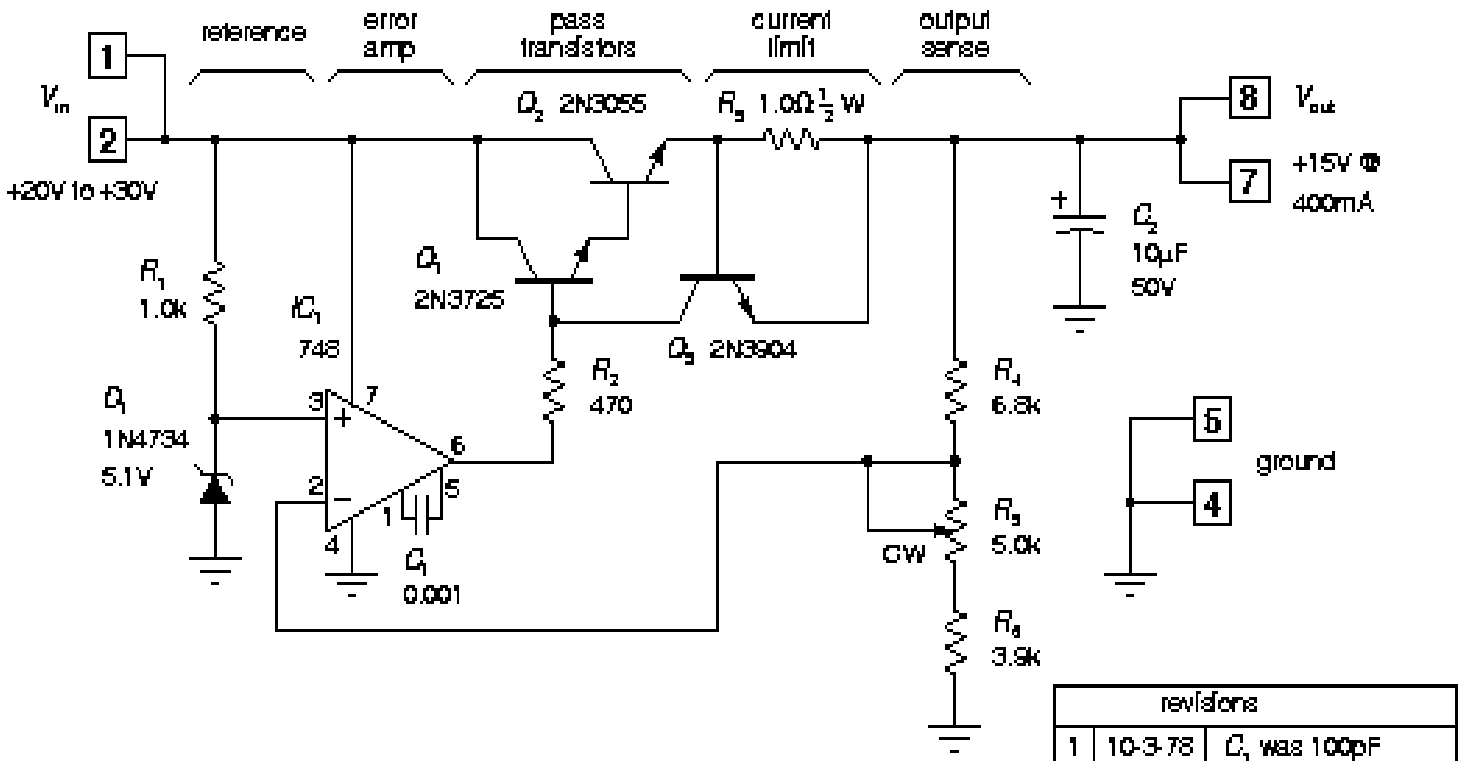
No connection



Schematic Diagram – How to Draw ?



Schematic Diagrams – Better Way



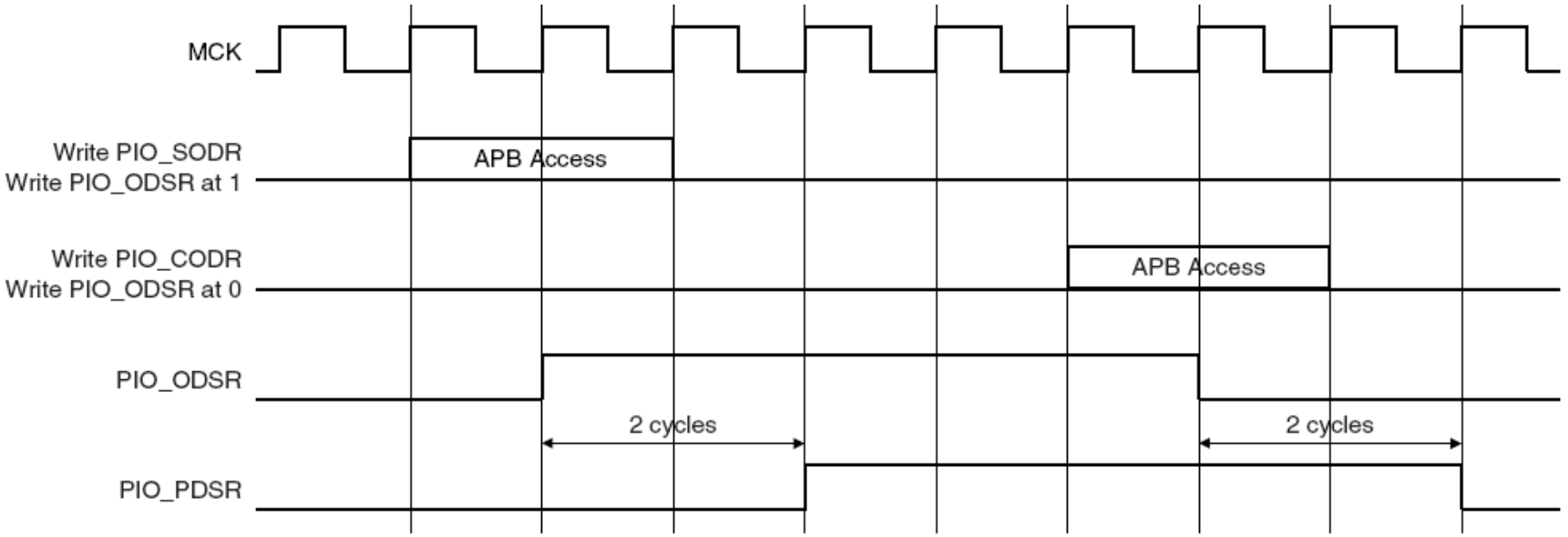
NOTES:

1. Q_2 on Wakefield 421AK 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

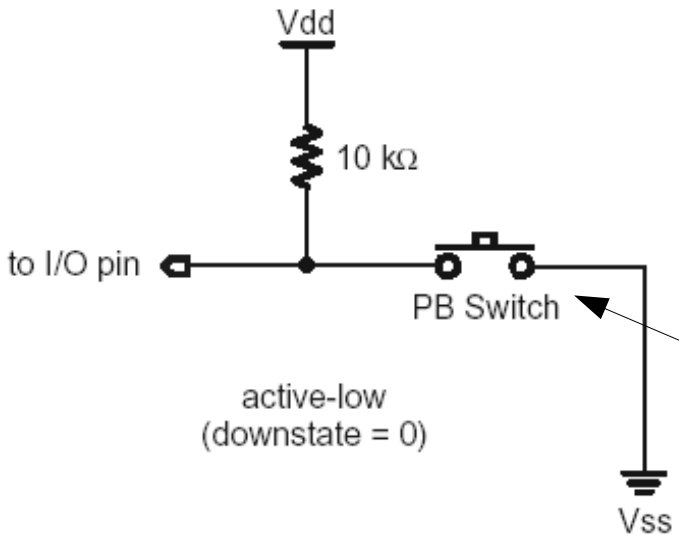
Timing charts during I/O operations



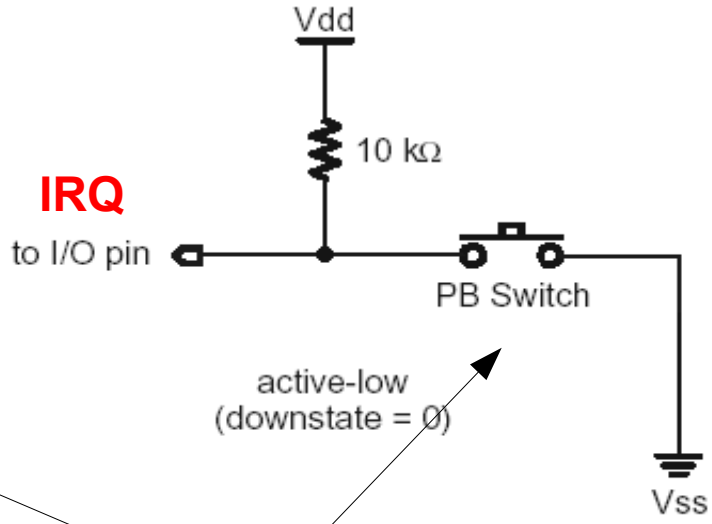
- 1 clock delay, when output driven from registers SODR/CODR,
- 2 clocks delay during access to the whole port (32 bits, set bits of PIO_OWSR register).

Reading state of button

Polling loop

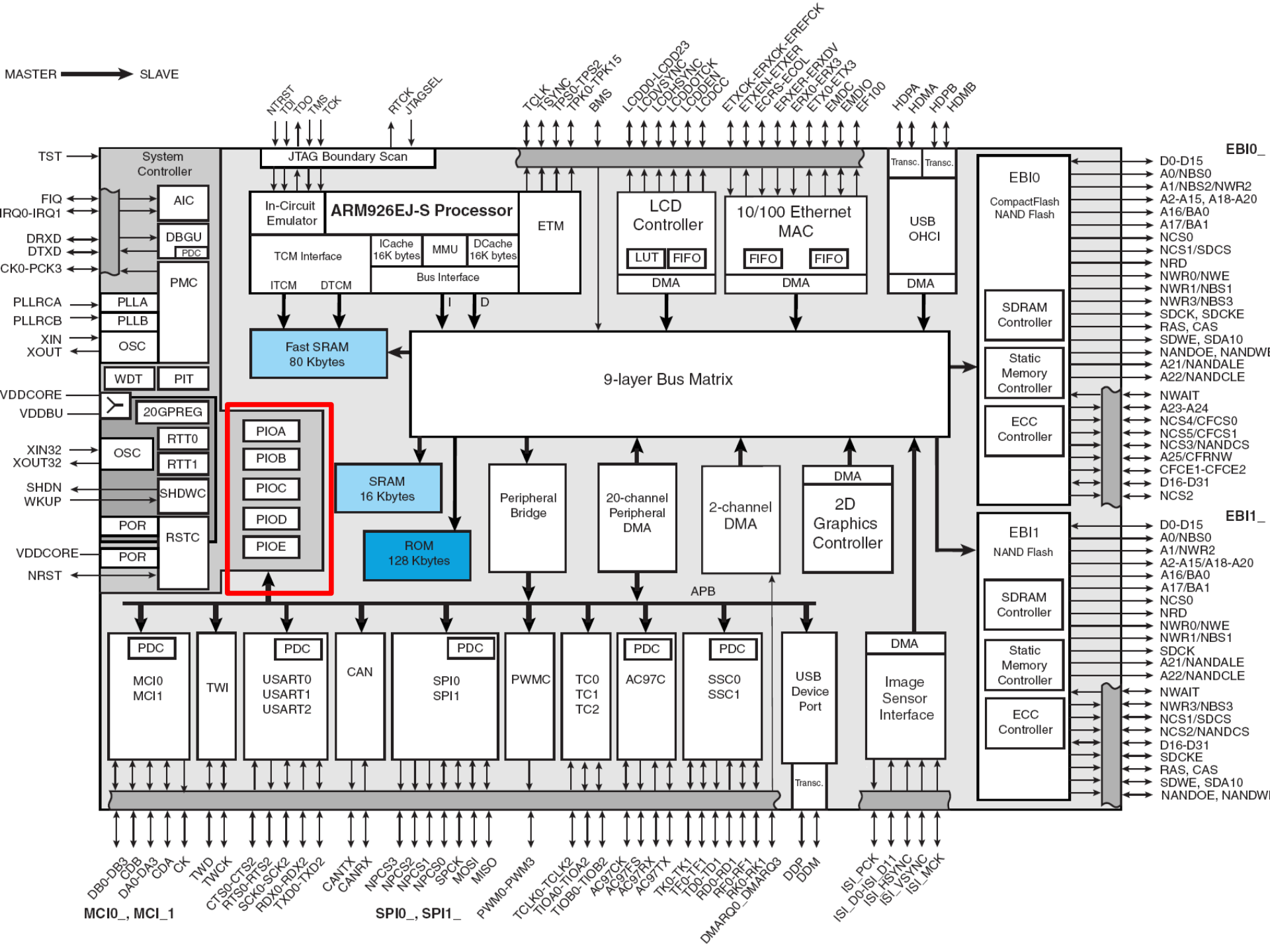


Interrupt



Asynchronous signal

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
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
TCM Interface
ICache 16K bytes
MMU
DCache 16K bytes
Bus Interface
ITCM
DTCM
ETM
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_

Documentation for AT91SAM9263 Microcontroller

Features

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

ATMEL®

AT91 ARM Thumb Microcontrollers

AT91SAM9263

Preliminary

Documentation for AT91SAM9263 – I/O Ports

AT91SAM9263 Preliminary

31. Parallel Input/Output Controller (PIO)

31.1 Overview

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

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

Each I/O line of the PIO Controller features:

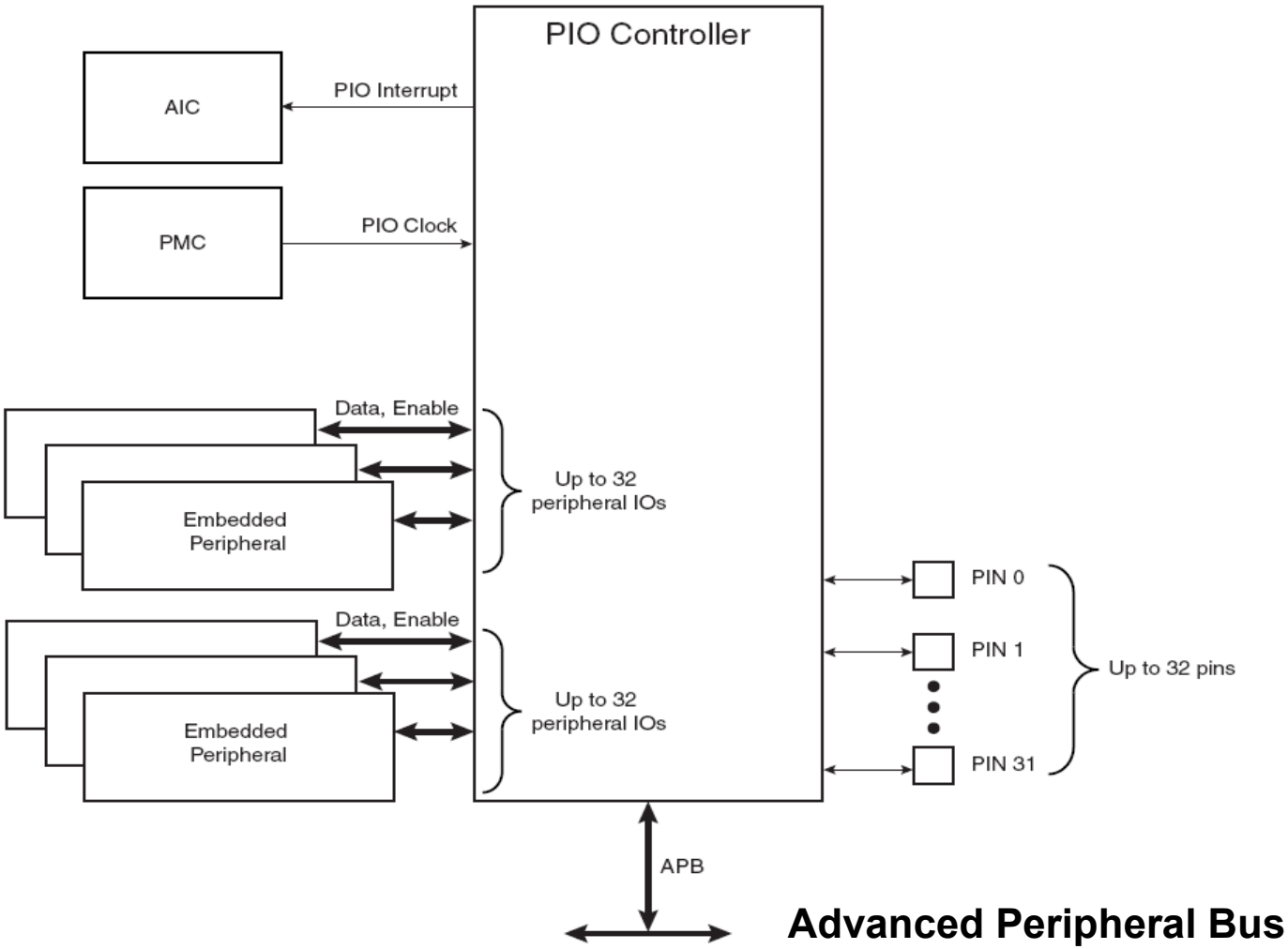
- An input change interrupt enabling level change detection on any I/O line.
- A glitch filter providing rejection of pulses lower than one-half of clock cycle.
- Multi-drive capability similar to an open drain I/O line.
- Control of the the pull-up of the I/O line.
- Input visibility and output control.

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

Source: ATMEL, doc6249.pdf, page 425

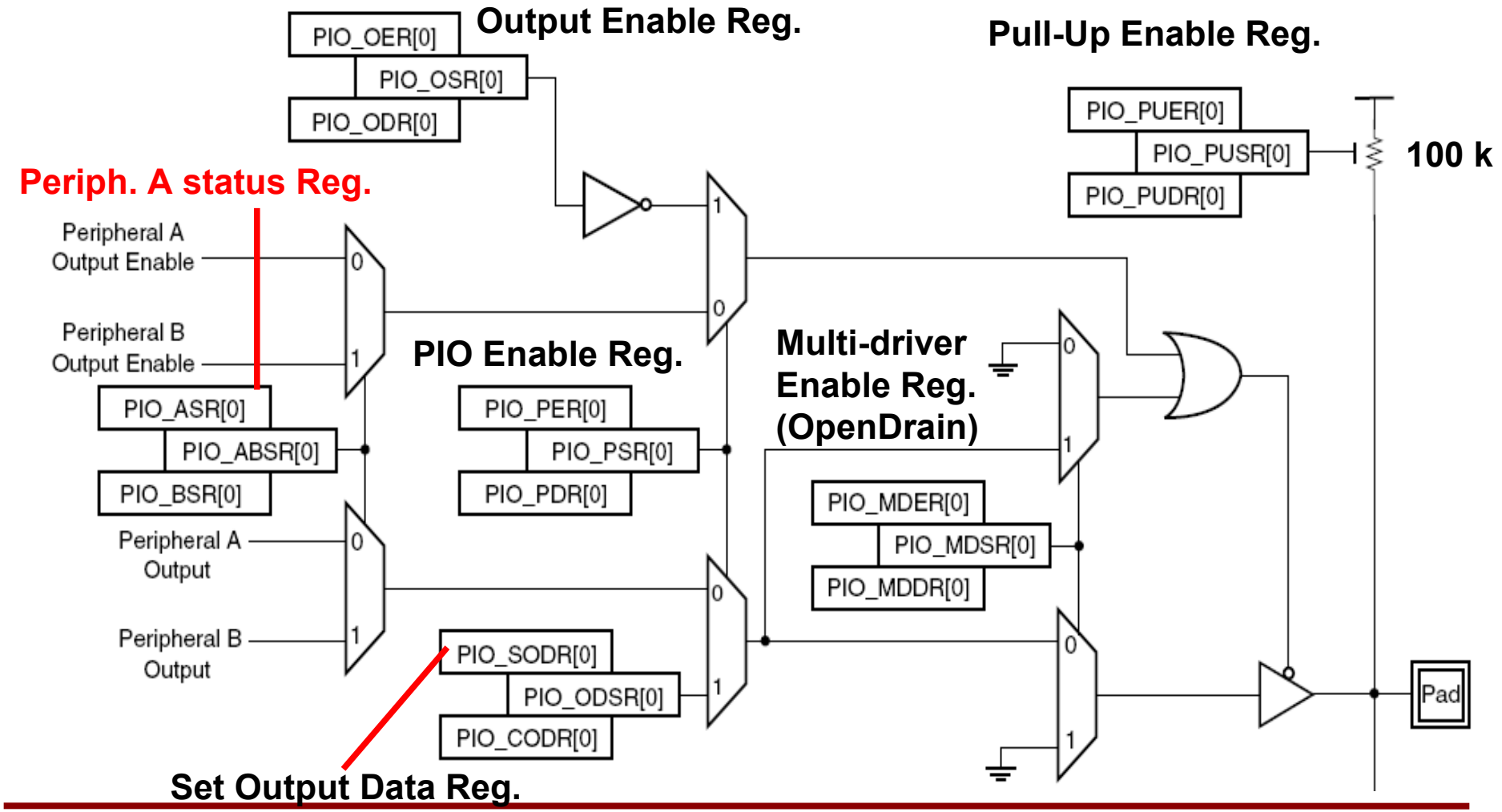
Block Diagram of 32-bits I/O Port

Figure 31-1. Block Diagram

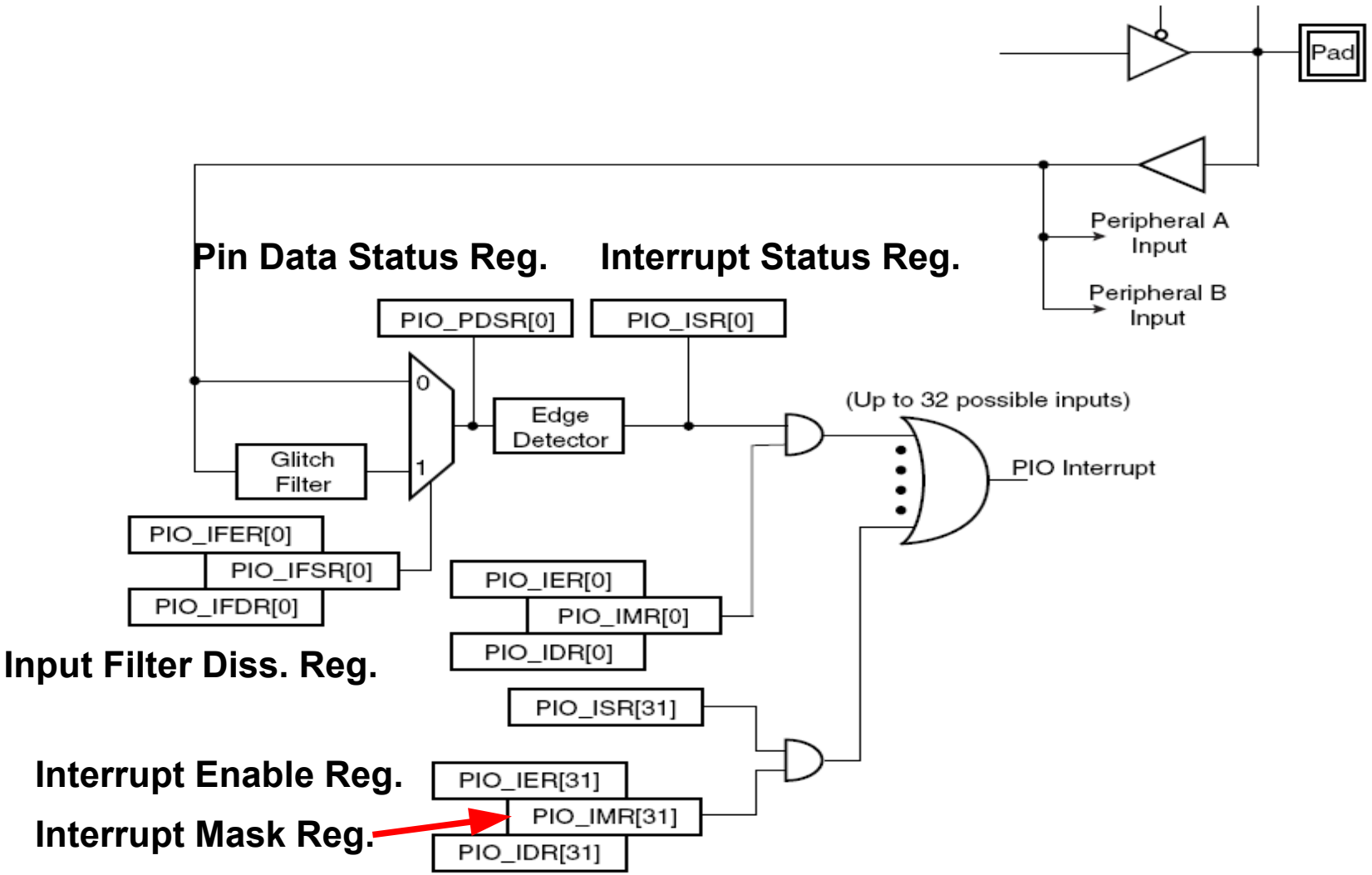


I/O Port – How to Control Output ?

Figure 31-3. I/O Line Control Logic



I/O – How to Read Input ?

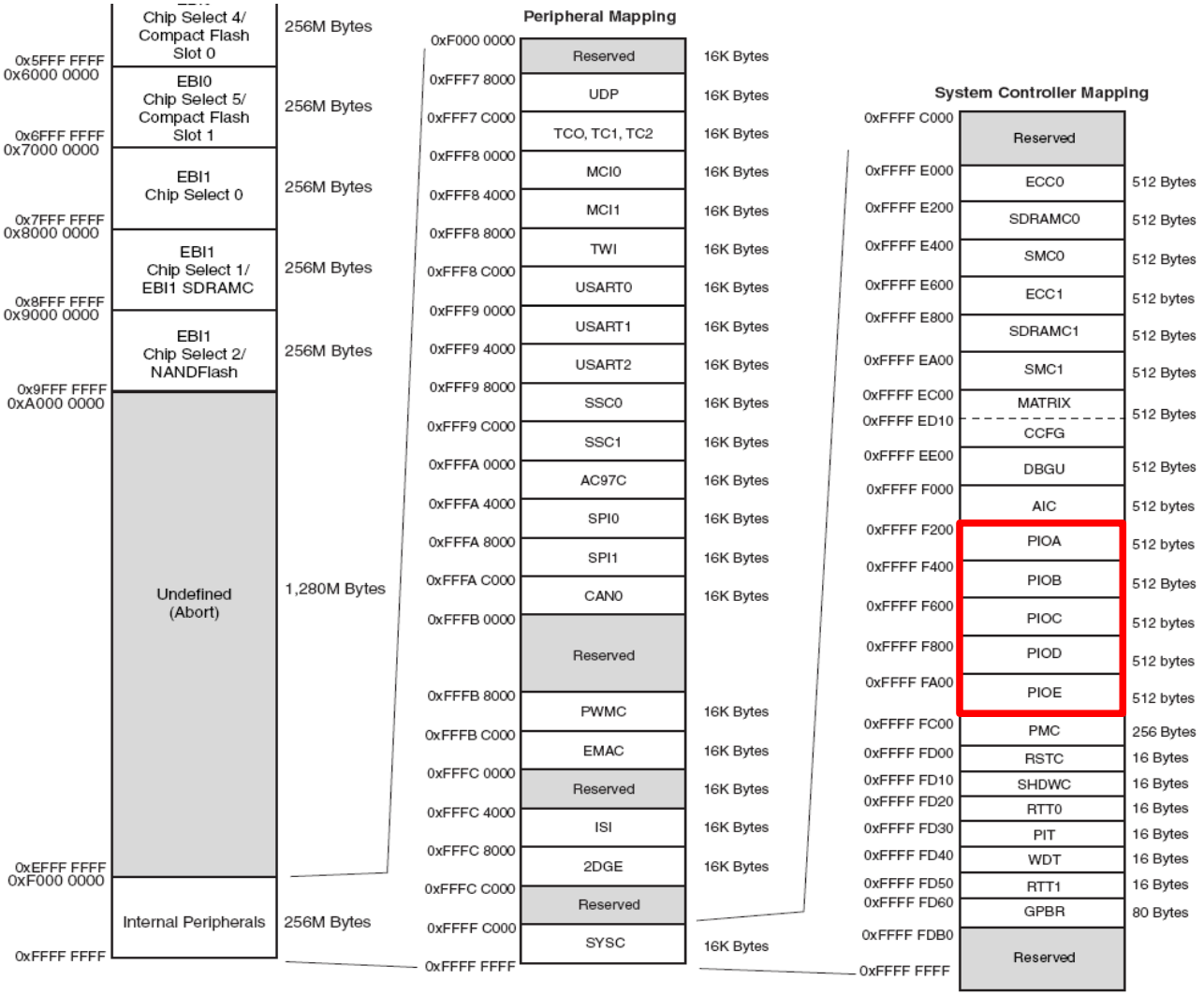


Control Registers for I/O ports

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

I/O Registers for I/O Ports

```
typedef volatile unsigned int *AT91_REG;        // Hardware register definition
        AT91_REG  PIO_PER = 0xFFFFF200;     // PIO Enable Register, 32-bit register
        AT91_REG  PIO_PDR = 0xFFFFF204;     // PIO Disable Register
        AT91_REG  PIO_PSR = 0xFFFFF208;     // PIO Status Register
        AT91_REG  Reserved0[1]= 0xFFFFF20C; // Filler
        AT91_REG  PIO_OER;              // Output Enable Register
        AT91_REG  PIO_ODR;              // Output Disable Registerr
        AT91_REG  PIO_OSR;              // Output Status Register
        AT91_REG  Reserved1[1];             //
        AT91_REG  PIO_IFER;             // Input Filter Enable Register
        AT91_REG  PIO_IFDR;             // Input Filter Disable Register
        AT91_REG  PIO_IFSR;             // Input Filter Status Register
        AT91_REG  Reserved2[1];             //
        AT91_REG  PIO_SODR;             // Set Output Data Register
        AT91_REG  PIO_CODR;             // Clear Output Data Register
        AT91_REG  PIO_ODSR;             // Output Data Status Register
```

I/O Registers Mapped into Structure (1)

```
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];                 //
        AT91_REG  PIO_OER;              // Output Enable Register
        AT91_REG  PIO_ODR;              // Output Disable Registerr
        AT91_REG  PIO_OSR;              // Output Status Register
        AT91_REG  Reserved1[1];                 //
        AT91_REG  PIO_IFER;             // Input Filter Enable Register
        AT91_REG  PIO_IFDR;             // Input Filter Disable Register
        AT91_REG  PIO_IFSR;             // Input Filter Status Register
        AT91_REG  Reserved2[1];                 //
        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;
```

I/O Registers Mapped into Structure (2)

Declartion of a new structure type creates a template for registers mapped on the memory of the processor. A Symbolic name is assigned to each register. The created structure is called according to used processor and functionality defined by registers, e.g. **AT91S_PIO** and ***AT91PS_PIO.**

Lack of information describing access to registers, e.g. access mode R/W, value after reset, offset.

The information can be supplied as a comments in header file.

```
typedef struct _AT91S_PIO {          /*     Register name              R/W     Reset value          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       -                    0x08
    AT91_REG  Reserved0[1];                  // memory filler
    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
    } AT91S_PIO, *AT91PS_PIO
    /* structure describing registers file (block of registers) for I/O ports PIOA...PIOE */
#define AT91C_BASE_PIOA      (AT91PS_PIO)    0xFFFFF200     // (PIOA) Base Address
    /* definition of bit mask for zero bit in port PA */
#define AT91C_PIO_PA0        (1 <<  0)       // Pin Controlled by PA0
```

How can we set 0 and 19 bits of OER register ?

Pointers to Base Registers for GPIO Ports

```
/* 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
```

Manipulation on Registers Bits

Save value to register:

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

Read value from register:

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

Bit operations:

```
AT91C_BASE_PIOA->ENABLE_REGISTER  = (AT91C_PIO_PA0 | AT91C_PIO_PA19);
AT91C_BASE_PIOA->DISABLE_REGISTER = (AT91C_PIO_PA0 | AT91C_PIO_PA19);
```

How to negate bit ?

Configuration of I/O ports

```
#define AT91C_PIO_PB8       (1U <<  8)                          // Pin Controlled by PB8
#define AT91C_BASE_PIOB   (AT91PS_PIO)        0xFFFF.F400U          // (PIOB) Base Address
```

Input mode:

```
/* Enable the peripheral 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 in input */
AT91C_BASE_PIOD->PIO_ODR = 0x0000.000FU;                // 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
```

Output mode:

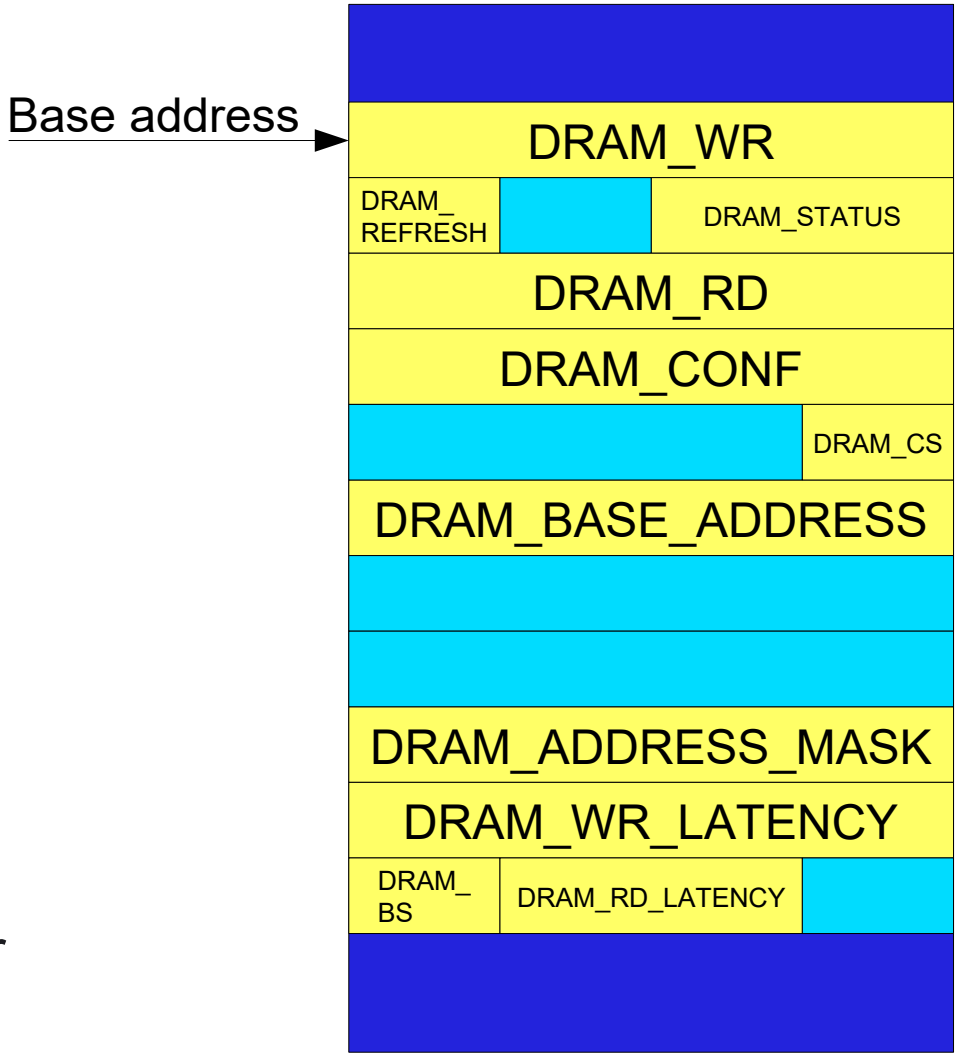
```
/* Configure the pin in output */
AT91C_BASE_PIOB->PIO_OER = AT91C_PIO_PB8 ;
/* Set the PIO controller in PIO mode instead of peripheral mode */
AT91C_BASE_PIOD->PIO_PER = 0xFFFF.FFFFU;                // 1 – Enable PIO to control the pin
AT91C_BASE_PIOE->PIO_PER = AT91C_PIO_PB31;
/* Disable pull-up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

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)



Time in processor systems

How can We Measure Time ?

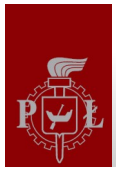
- Generate defined delay ?
- Generate date and time ?
- Measure length of pulses ?
- Delay in Real-Time systems ?

Crystal Clock...

- Quartz from chemical point of view is a compound called silicon dioxide. Properly cut and mounted crystal of quartz can be made to vibrate, or oscillate, using an alternating electric current.
- The frequency at which the crystal oscillates is dependent on its shape and size, and the positions at which electrodes are placed on it.
- If the crystal is accurately shaped and positioned, it will oscillate at a desired frequency; in clocks and watches, the frequency is usually 32,768 Hz, as a crystal for this frequency is conveniently small. Such a crystals are usually used in digital systems.

Timers

- **Timer –** peripheral device of processor dedicated for **time measurement** (counting single processor cycles).
- Flag is marked or interrupt is triggered when timer counter reaches threshold level.
- Timers are used as a system time source.
- They can be used to generate delays, switch threads, generate events, etc...



Timers

Example of different timers in STM32L4x6 MCU:

Basic Timers (TIM6, TIM7)

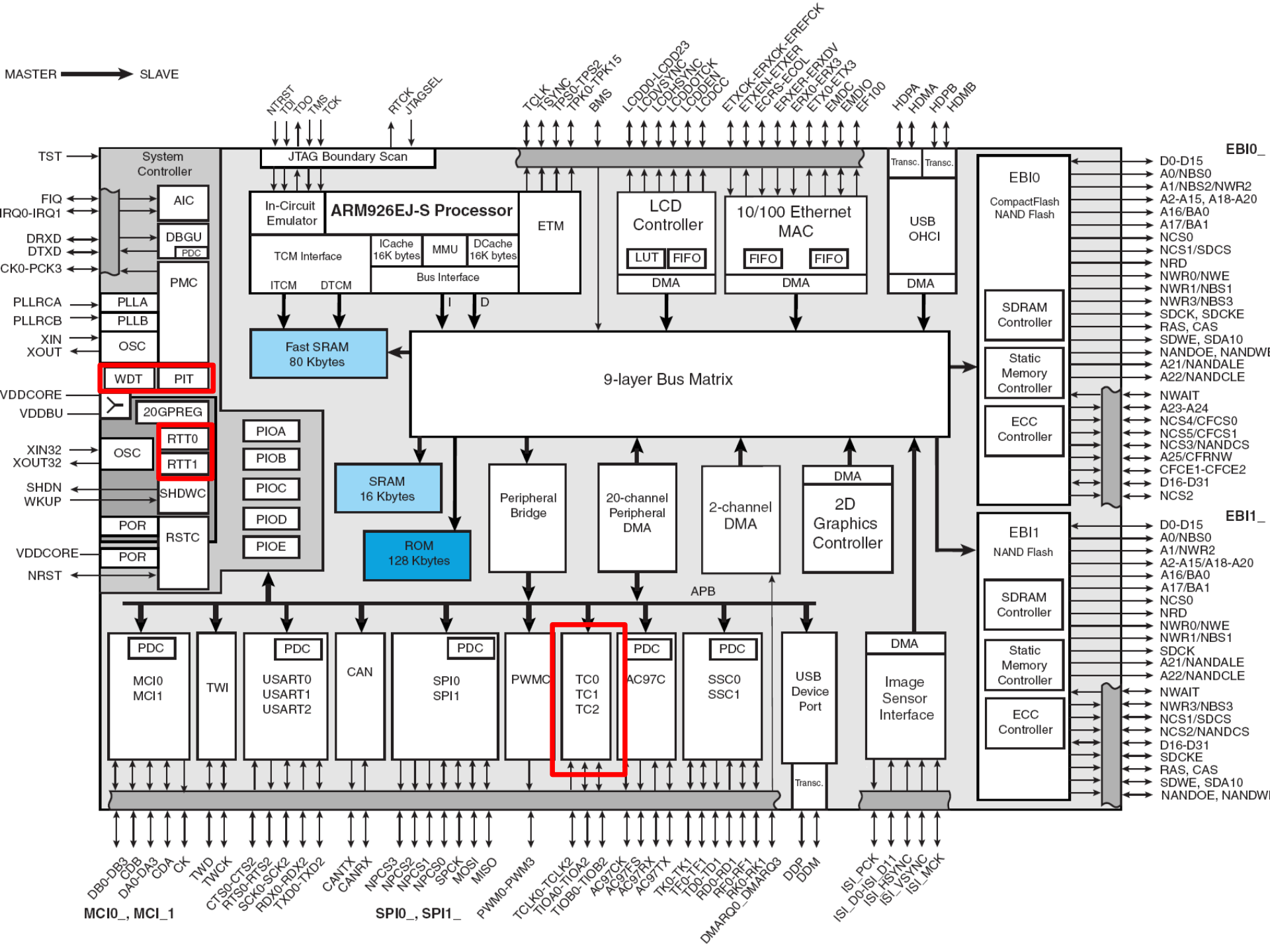
General-purpose Timers / PWM Timer (TIM2-TIM5, TIM15-TIM17)

Advanced-control Timers / PWM Timer (TIM1, TIM8)

Low-power Timer (LPTIM)

Watchdog Timers (IWDG, WWDG)

Real-time Timer (RTC)


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_

Basic Timers
(TIM6, TIM7, LPTIM)
Chapter 33

There will be no lecture on 5th May (14.15-16.00)

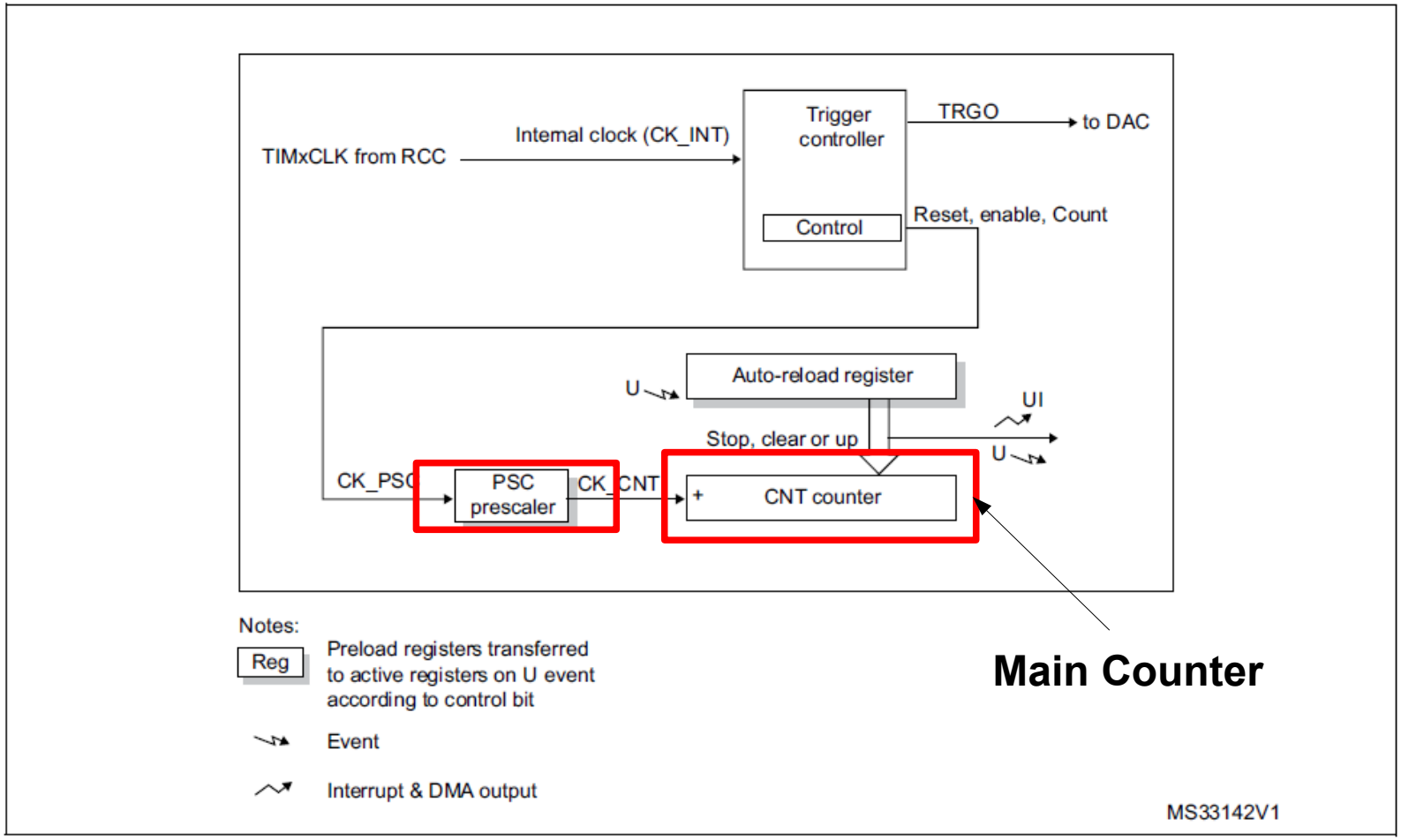
Basic timers (TIM6/TIM7)

Basic timer (TIM6/TIM7) features include:

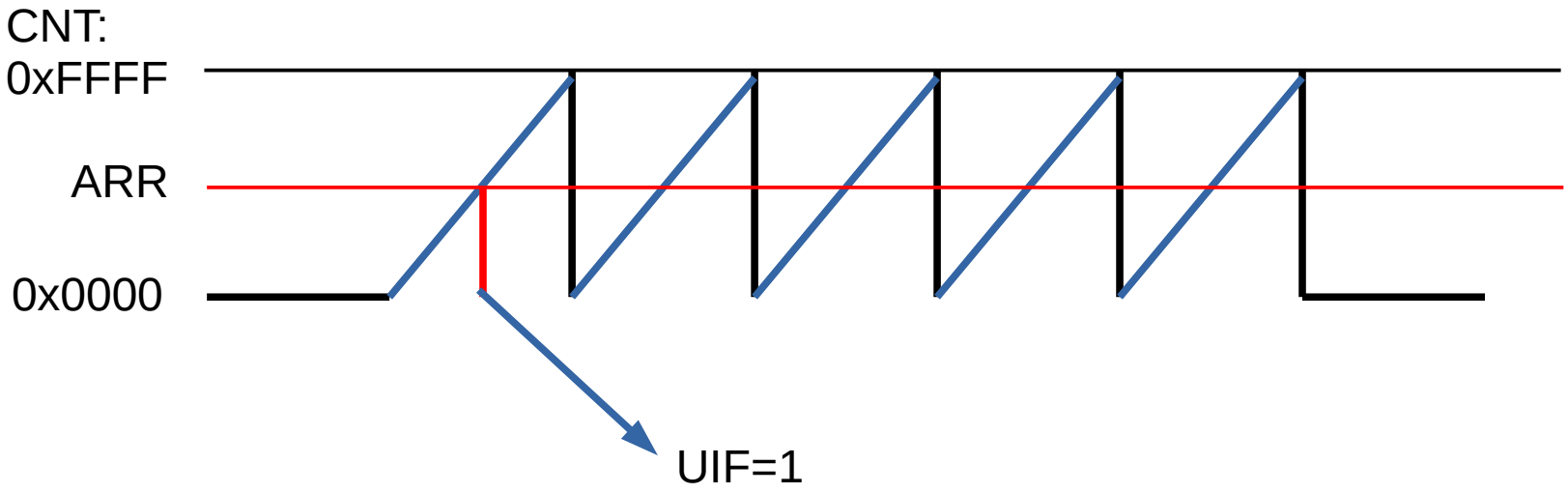
- 16-bit auto-reload up-counter
- Additional 16-bit programmable prescaler used to divide the counter clock frequency (1 to 65535)
- Synchronization circuit to trigger other peripheral devices (DAC)
- Interrupt/DMA generation on the update event: counter overflow

Block Diagram of Basic Timer

Figure 368. Basic timer block diagram



Automatic Reload of Timer



Period of generated events:

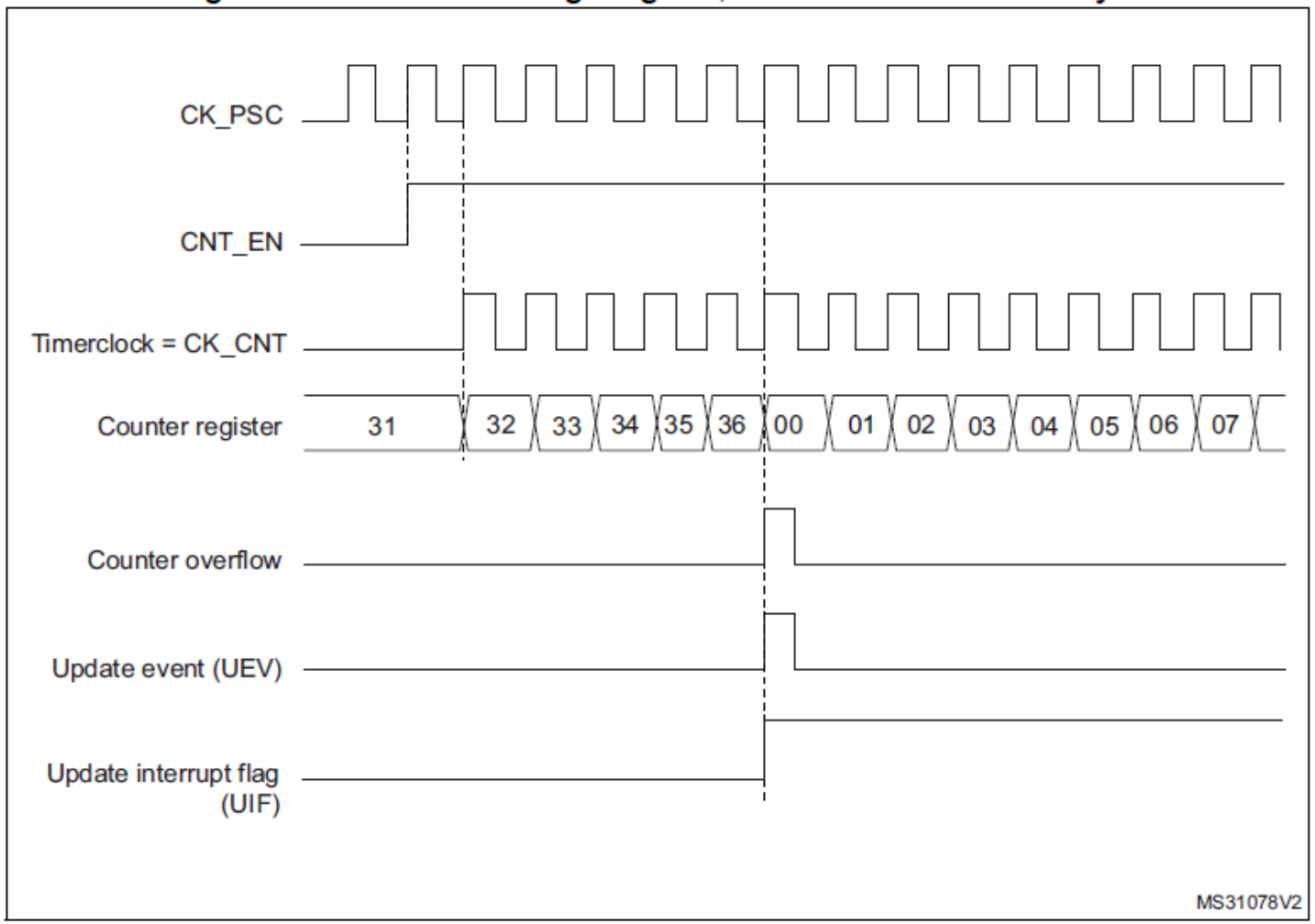
t_{TIM} = (TIM6_ARR+1) * (TIM6_PSC+1) / Clk => **ARR = ….**

Clk = 4 MHz, TIM6_ARR = 1999, TIM6_PSC = 3999 => t_{TIM} = 2000 ms

(**please provide detailed calculations during lab**)

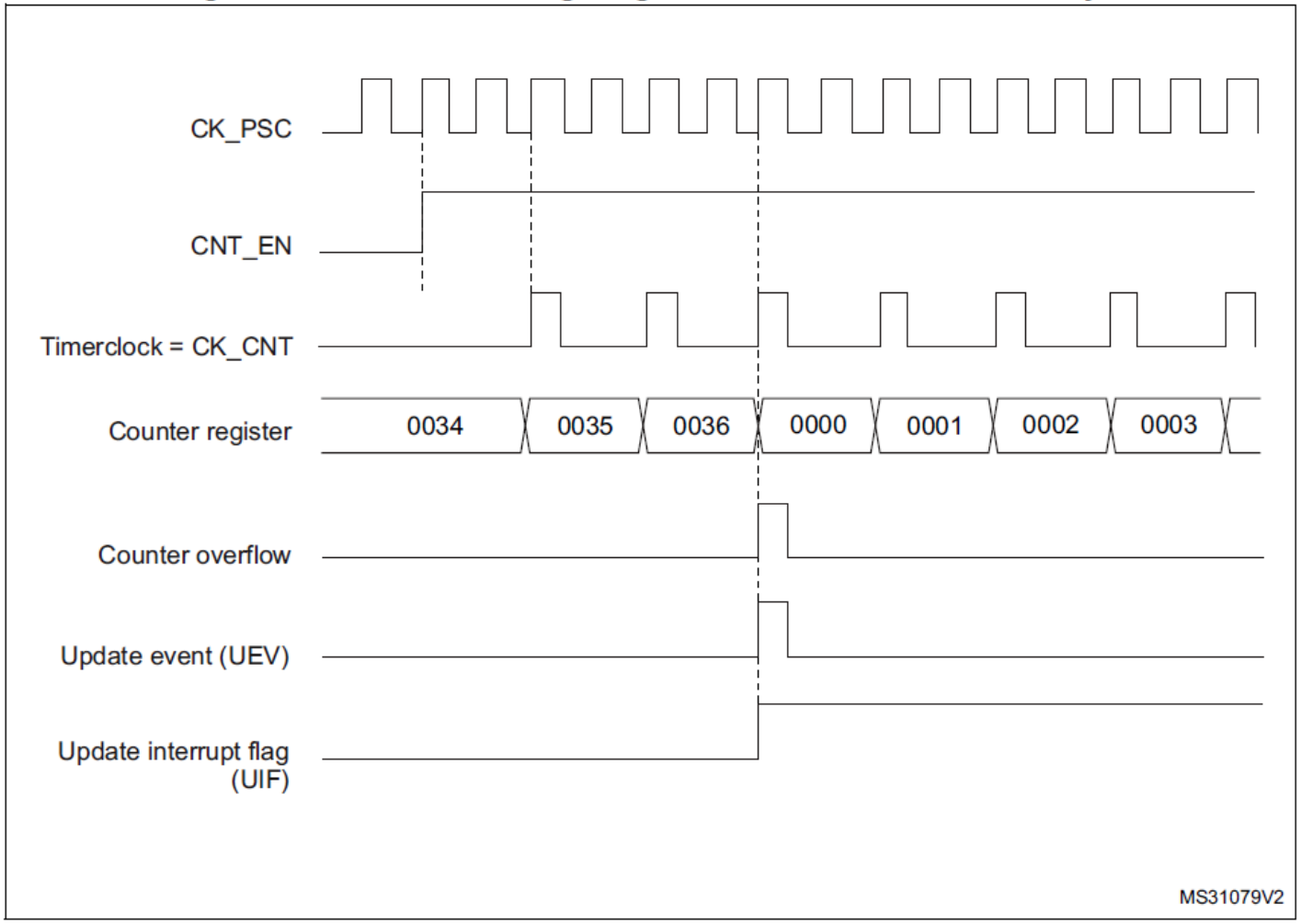
Counting Mode (Divider set to 1)

Figure 371. Counter timing diagram, internal clock divided by 1



Counting Mode (Divider set to 2)

Figure 372. Counter timing diagram, internal clock divided by 2



Basic Timer Registers

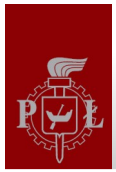
Offset	Register	Name	Access	Reset Value
0x00	Control register 1	TIMx_CR1	R/W	0x0000
0x04	Control register 2	TIMx_CR2	R/W	0x0000
0x0C	DMA/Interrupt enable register	TIMx_DIER	R/W	0x0000
0x10	Status register	TIMx_SR	R/W	0x0000
0x14	Event generation register	TIMx_EGR	W	0x0000
0x24	Counter	TIMx_CNT	RW	0x0000 0000
0x28	Prescaler	TIMx_PSC	RW	0x0000
0x2C	Auto-reload register	TIMx_ARR	RW	0xFFFF

Timer 6 – Base Address

APB1	0x4000 2C00 - 0x4000 2FFF	1 KB	WWDG	*Section 37.4.4: WWDG register map*
	0x4000 2800 - 0x4000 2BFF	1 KB	RTC	*Section 38.6.21: RTC register map*
	0x4000 2400 - 0x4000 27FF	1 KB	LCD	*Section 25.6.6: LCD register map*
	0x4000 1800 - 0x4000 2400	3 KB	Reserved	-
	0x4000 1400 - 0x4000 17FF	1 KB	TIM7	*Section 33.4.9: TIM6/TIM7 register map*
	0x4000 1000 - 0x4000 13FF	1 KB	TIM6	*Section 33.4.9: TIM6/TIM7 register map*

Programming Timer 6 with HAL - Initialization

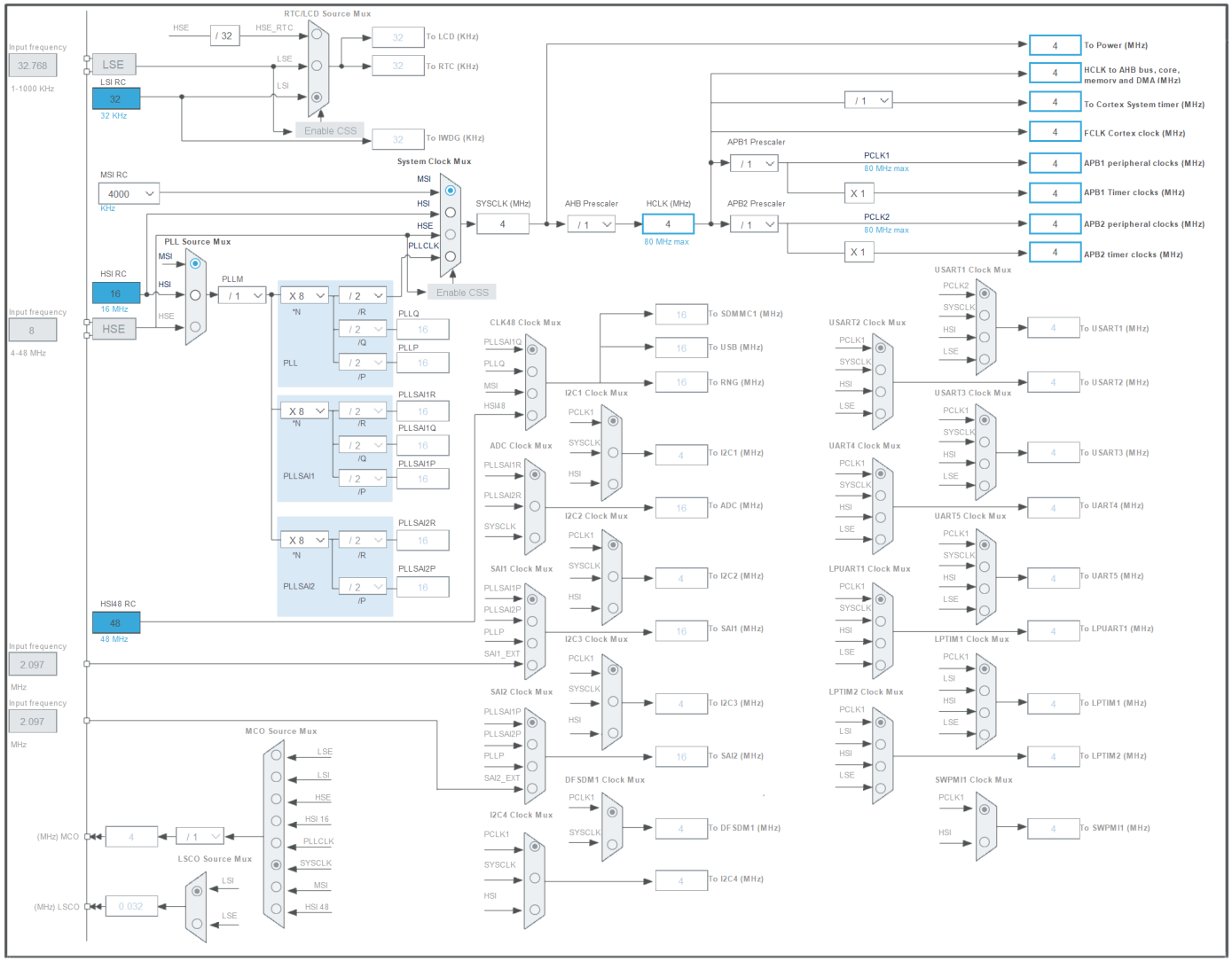
```
static TIM_HandleTypeDef s_TimerInstance = {
    .Instance = TIM6
  };
  __HAL_RCC_TIM6_CLK_ENABLE();

s_TimerInstance.Init.AutoReloadPreload=TIM_AUTORELOAD_PRELOAD_ENABLE;
s_TimerInstance.Init.Period = 999;              // 2000 counts = 2000 ms
s_TimerInstance.Init.Prescaler = 3999;          // 4 MHz / (4000)  = 1 kHz

HAL_TIM_Base_Init(&s_TimerInstance);
HAL_TIM_Base_Start(&s_TimerInstance);
```



Programming Timer 6 with HAL - Usage

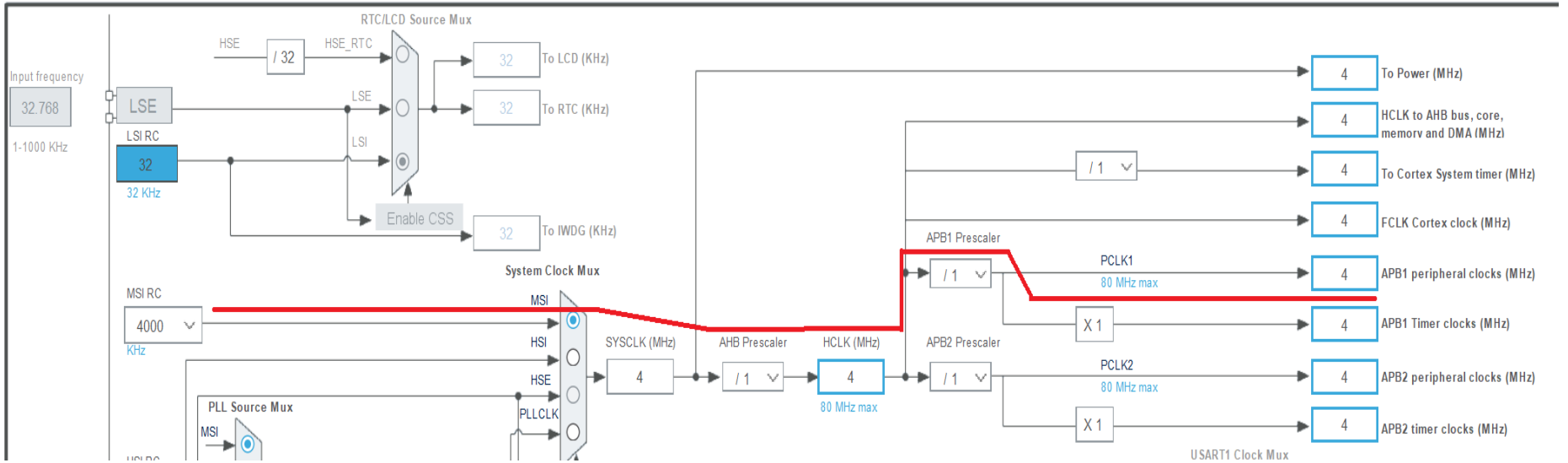
```
while (1)
 {
 uint8_t flag = __HAL_TIM_GET_FLAG(&s_TimerInstance, TIM_FLAG_UPDATE);
       if (flag)
       {
            /* USER CODE END WHILE */
            __HAL_TIM_CLEAR_FLAG(&s_TimerInstance, TIM_FLAG_UPDATE);
       }
  /* USER CODE BEGIN 3 */
 }
```

Configuration of Reference Clock



Configuration of Reference Clock (2)

- Timer 6 uses the APB1 clock
- Generated by internal MSI RC oscillator, f = 4 MHz
- APB1 clock derived directly from MSI clock (also 4 MHz)
- We do not recommend changing these settings



Timer Usage...

- Enable timer clock: RCC_APB1ENR1, bit TIM6EN
- Configure timer (2000 ms, periodic)

  1. Configure timer to auto-reload, periodic mode: TIM6_CR1, bits: ARPE, OPM, URS
  2. Enable update event: TIM6_CR1, bit: UDIS
  3. Configure prescaler: TIM6_PSC
  4. Configure period: TIM6_ARR
  5. Configure reload timer after event: TIM6_EGR, bit UG
- Enable timer: TIM6_CR1, bit: CEN
- Use polling for checking Status Flag: TIM6_SR, bit UIF
- Clear UIF (if required) after counted period → Count next period, etc.

Optional:

- Check TIM6_CNT to see if timer is counting

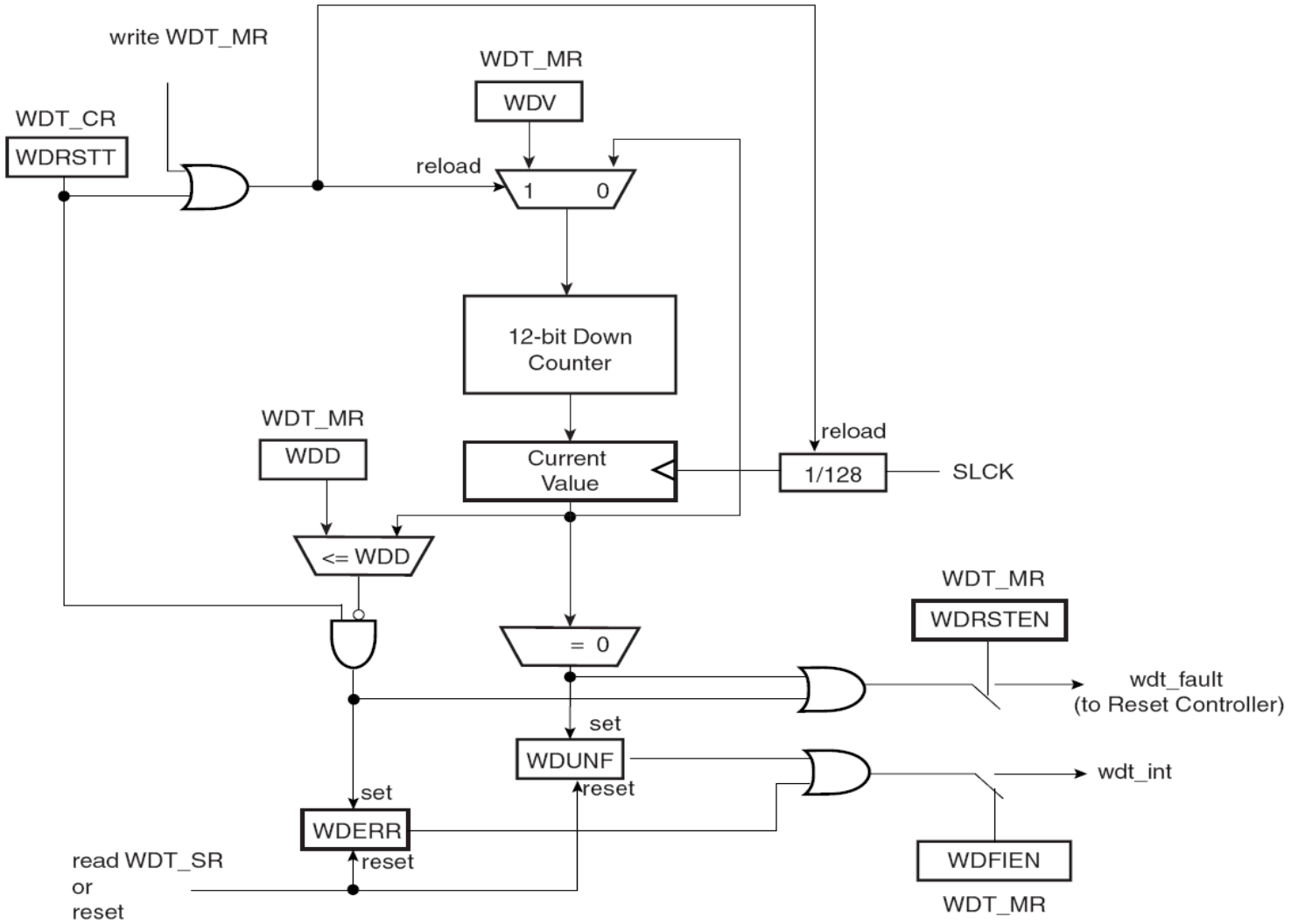
Watchdog Timer

Watchdog Timer (WDT) is used to prevent microprocessor system lock-up if the software becomes trapped in a deadlock.

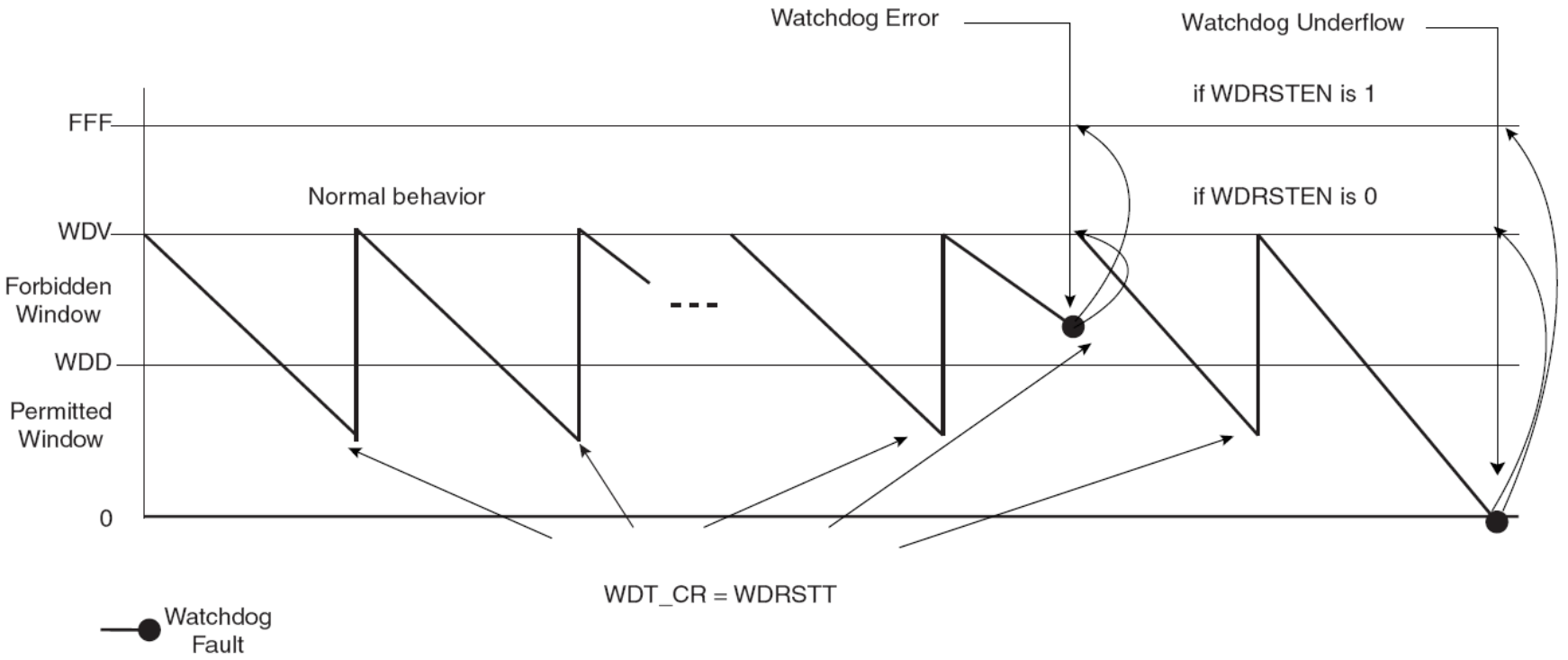
Features of WDT:

- 12-bit down counter,
- Triggered with slow clock (32.768 kHz),
- Maximum watchdog period of up to 16 seconds,
- Can generate a general reset or a processor reset only,
- WDT can be stopped while the processor is in debug mode or idle mode,
- Write protected WDT_CR (control register).

Watchdog Timer – block diagram



WDT – timing charts



WDT – registers (1)

17.4.1 Watchdog Timer Control Register

Register Name: WDT_CR

Access Type: Write-only

31	30	29	28	27	26	25	24
KEY							
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WDRSTT

- **WDRSTT: Watchdog Restart**

0: No effect.

1: Restarts the Watchdog.

- **KEY: Password**

Should be written at value 0xA5. Writing any other value in this field aborts the write operation.

WDT – registers (2)

17.4.2 Watchdog Timer Mode Register

Register Name: WDT_MR

Access Type: Read/Write Once

31	30	29	28	27	26	25	24
		WDIDLEHLT	WDDBGHLT	WDD			
23	22	21	20	19	18	17	16
WDD							
15	14	13	12	11	10	9	8
WDDIS	WDRPROC	WDRSTEN	WDFIEN	WDV			
7	6	5	4	3	2	1	0
WDV							

Interfaces in Embedded Systems

Fundamental Definitions

- **Computer Memory**

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

- **Peripheral Device**

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

- **Computer Bus**

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

- **Interface**

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

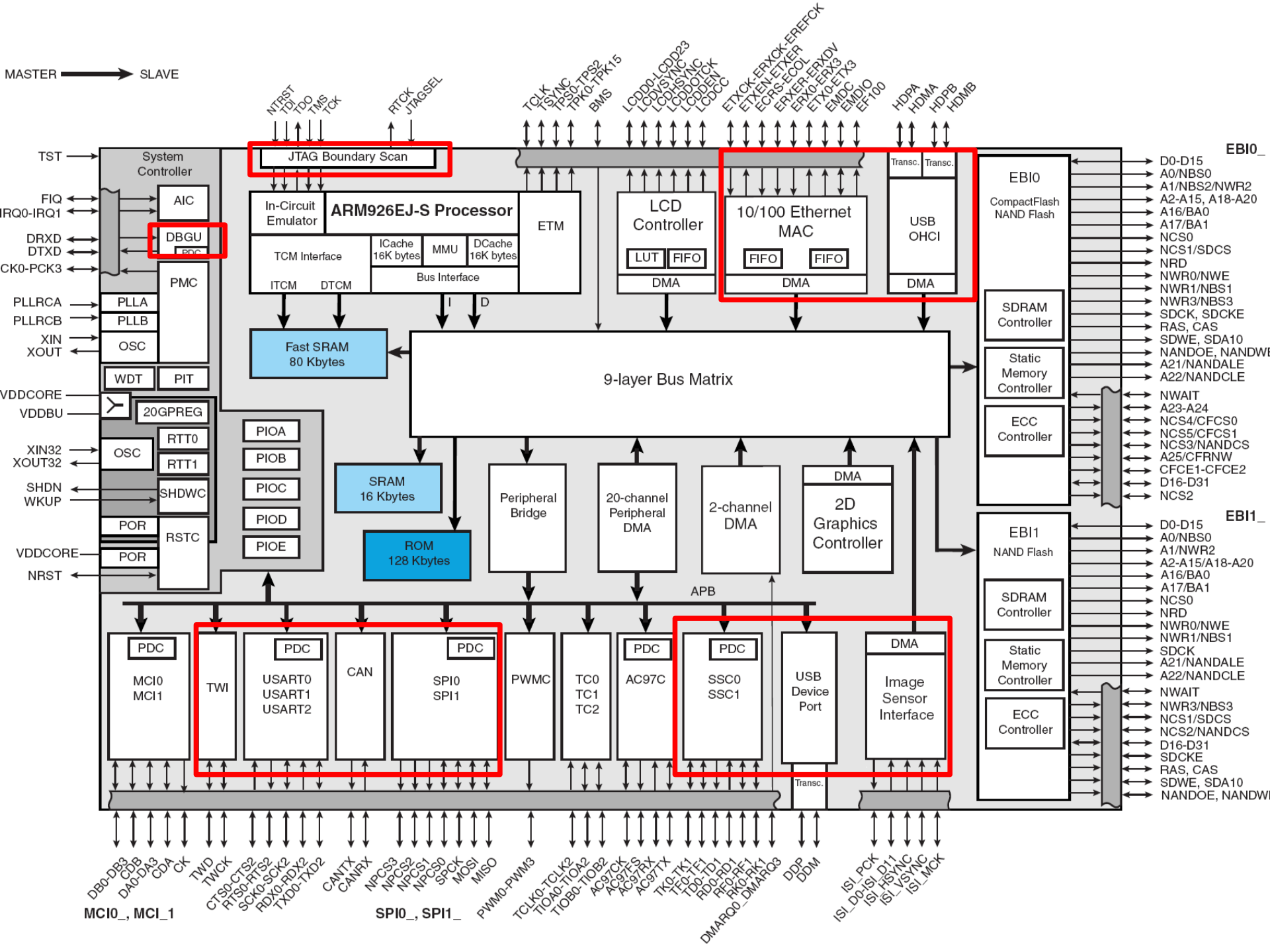
Connectivity of Processor and Peripheral Devices

Interfaces used in Embedded Systems:

- Parallel Interface PIO (usually 8, 16 or 32 bits),
- Serial interfaces:
  - Universal (Serial) Asynchronous Receiver-Transmitter (UART/USART),
  - Serial Peripheral Interface (SPI),
  - Synchronous Serial Controller (SSC)
  - I2C, Two-wire Interface (TWI),
  - Controlled Area Network (CAN),
  - Universal Serial Bus (USB),
  - Ethernet 10/100 Mbits (1 Gbit),
  - Debug/programming interface (EIA RS232, JTAG, SPI, DBGU),
  - Single Wire Interface (SWI),
  - and more...

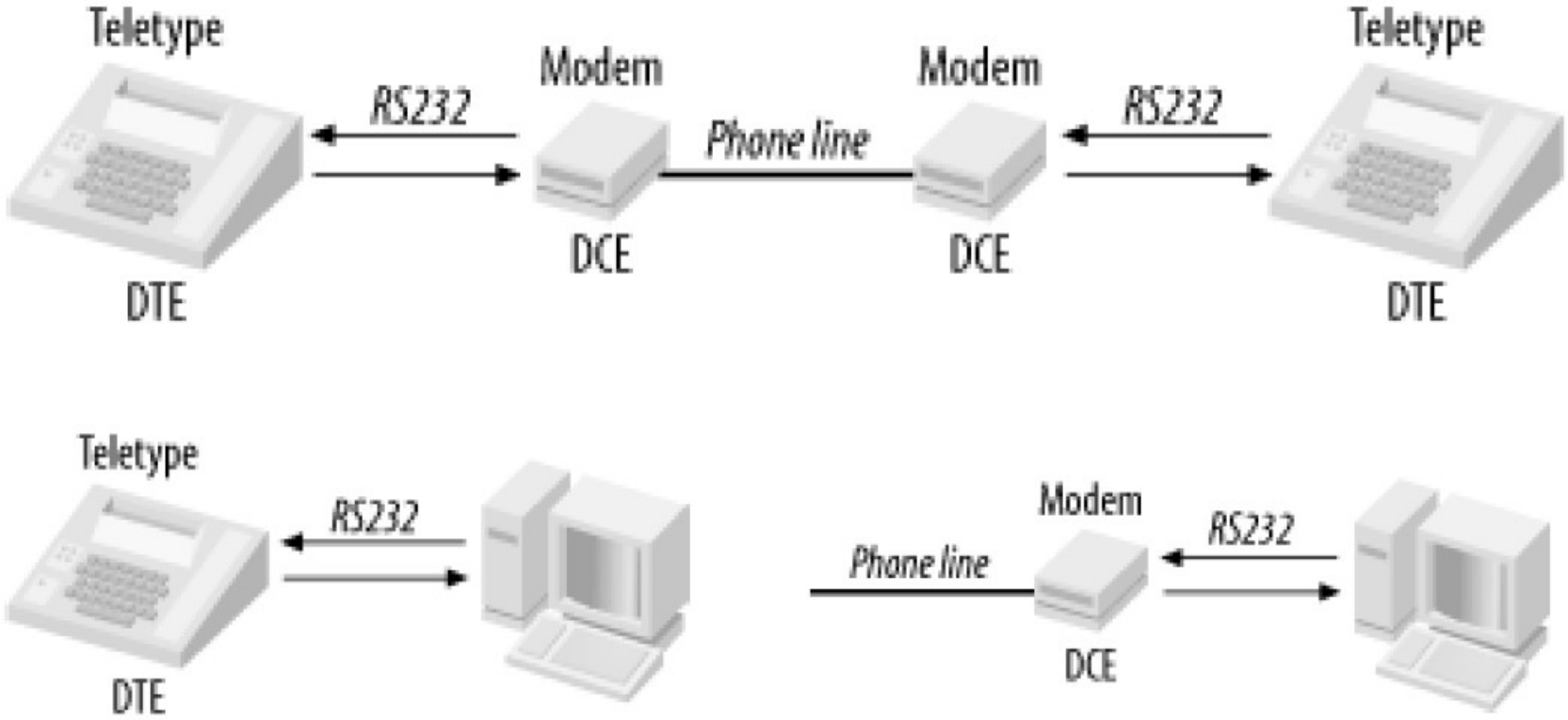
Interfaces available in AT91SAM9263

- Parallel Interface PIO (configurable 32 bits),
- Serial interfaces:
  - Debug interface (DBGU),
  - Universal (Serial) Asynchronous Receiver-Transmitter (UART/USART),
  - Serial Peripheral Interface (SPI),
  - Synchronous Serial Controller (SSC),
  - I2C, Two-wire Interface (TWI),
  - Controlled Area Network (CAN),
  - Universal Serial Bus (USB, host, endpoint),
  - Ethernet 10/100 Mbits,
  - Programming interface (JTAG).


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
CK0-PCK3
PMC
PLLRCA
PLLRCB
PLLA
PLLB
XIN
XOUT
OSC
WDT
PIT
VDDCORE
VDDBU
20GPREG
XIN32
XOUT32
OSC
RTT0
RTT1
SHDN
WKUP
SHDWC
POR
RSTC
VDDCORE
POR
NRST
In-Circuit Emulator
ARM926EJ-S Processor
ETM
TCM Interface
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
PIOA
PIOB
PIOC
PIOD
PIOE
APB
PDC
MCI0
MCI1
TWI
PDC
USART0
USART1
USART2
CAN
PDC
SPI0
SPI1
PWMC
TC0
TC1
TC2
PDC
AC97C
PDC
SSC0
SSC1
USB Device Port
DMA
Image Sensor Interface
Transc.
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
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_

Universal Asynchronous Receiver/Transmitter Module

EIA RS232 Serial Interface



UART Transceiver

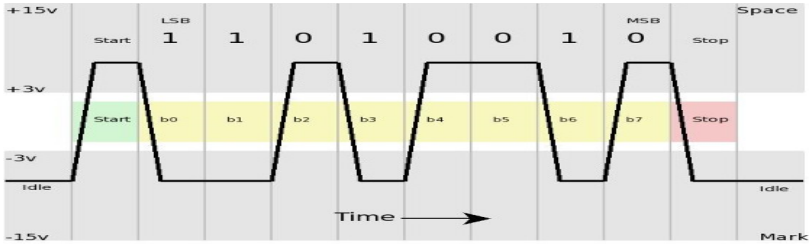
Shift register

D0-D7

transmitter

TxD

Clk



D0-D7

Receiver

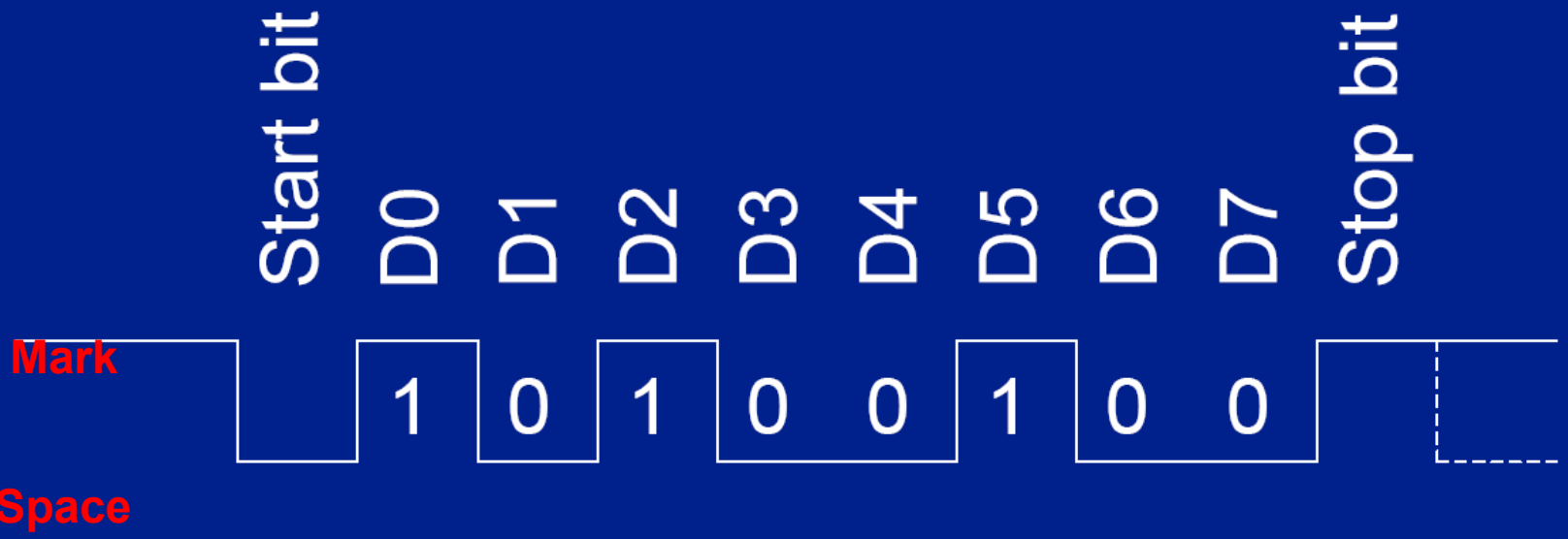
RxD

Clk

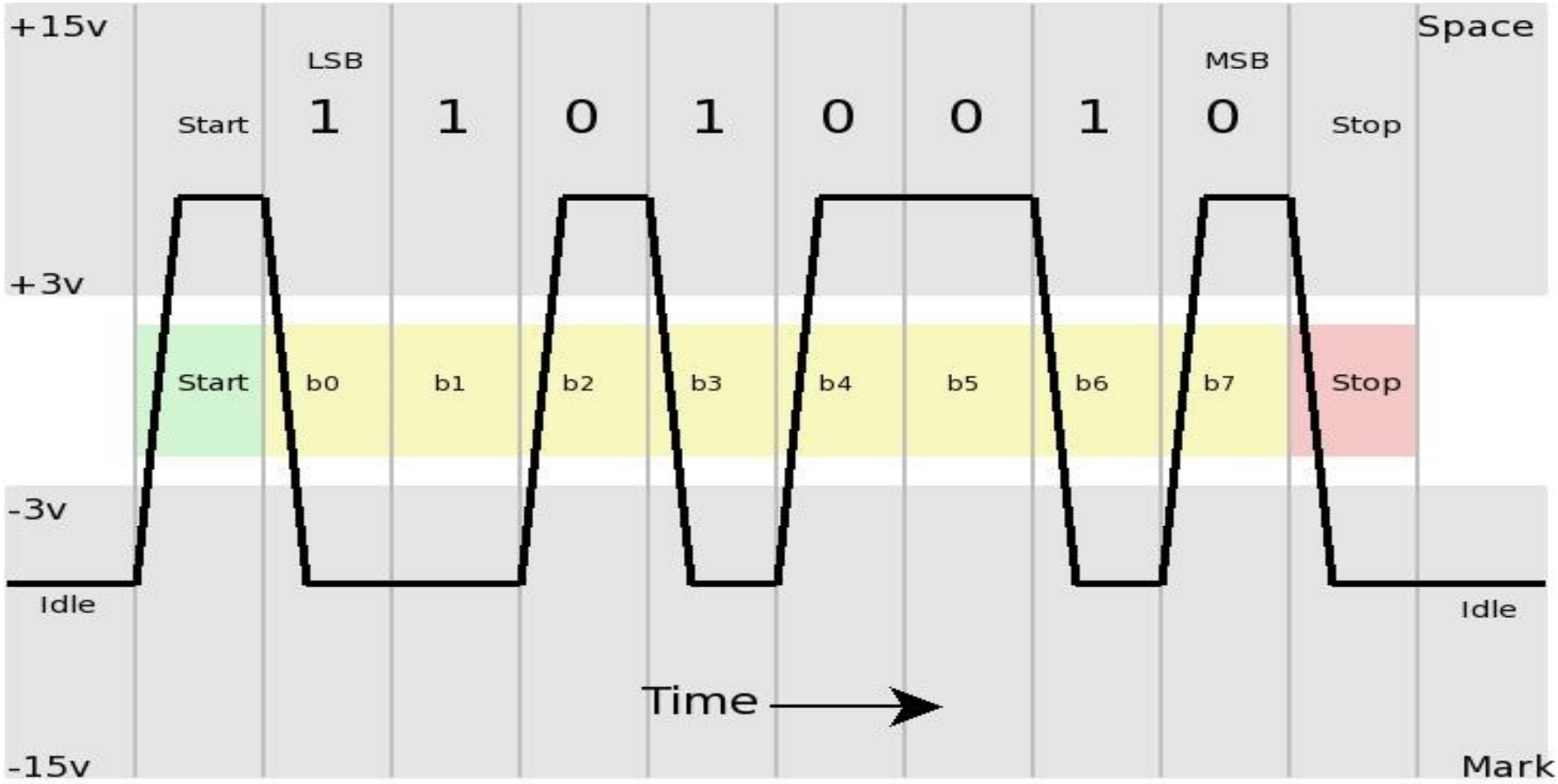
Data Frame of UART (1)

Asynchronous 8 bit waveform example

- Data is H'25' = B'00100101'

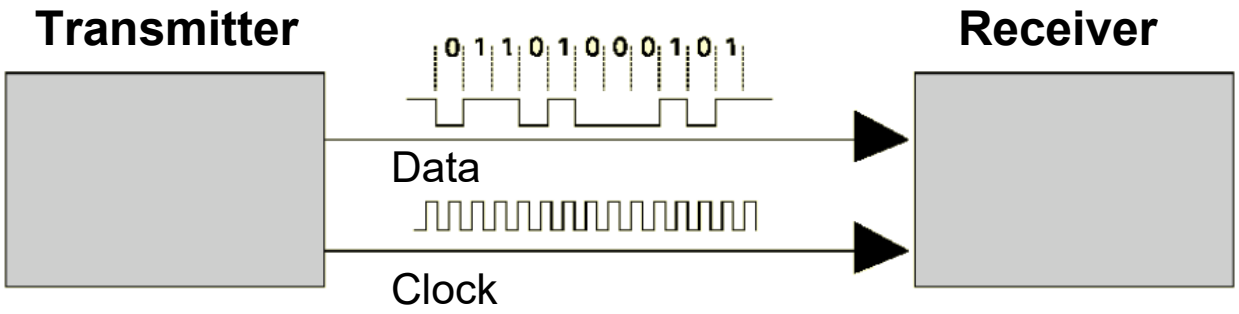


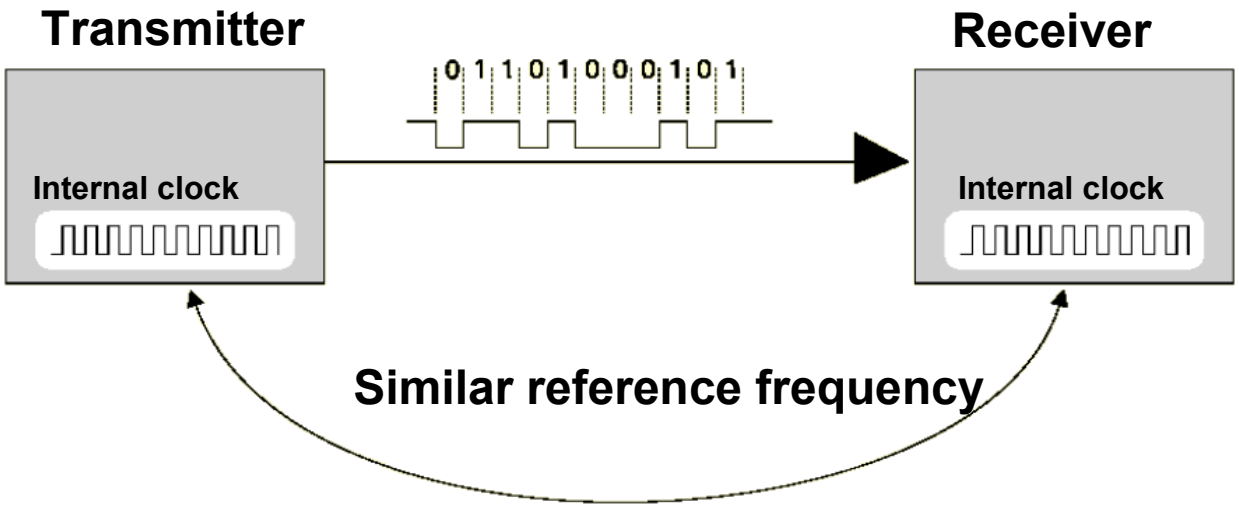
Data Frame of UART (2)



Send data: 0100.1011b = 0x4B

Synchronous vs asynchronous transmission

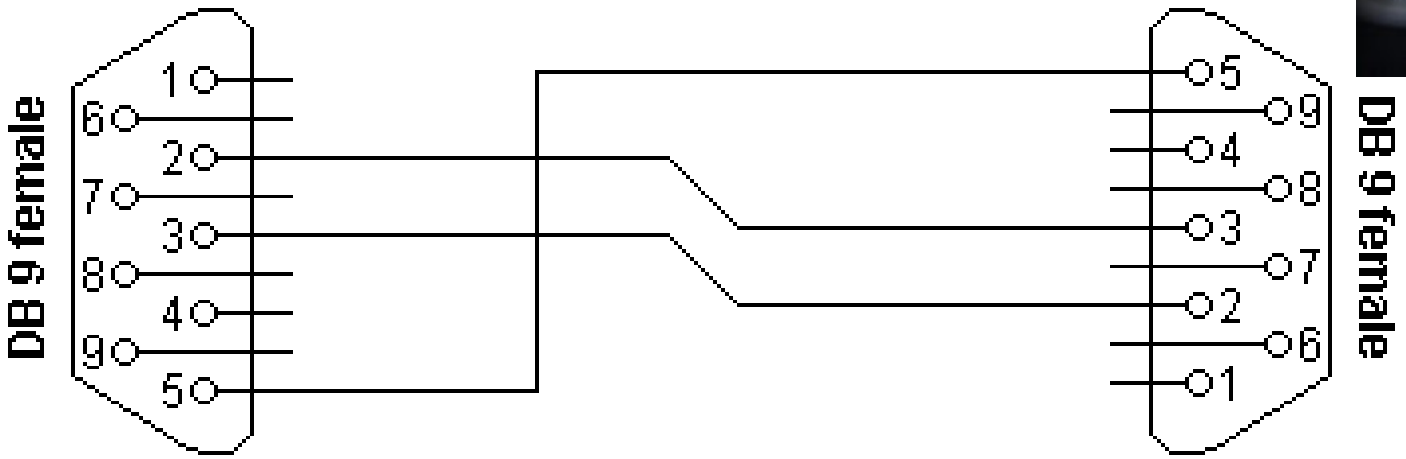




Electrical specification of EIA RS232c

SPECIFICATIONS		RS232
Mode of Operation		SINGLE -ENDED
Total Number of Drivers and Receivers on One Line		1 DRIVER 1 RECVR
Maximum Cable Length		50 FT.
Maximum Data Rate		20kb/s
Maximum Driver Output Voltage		+/-25V
Driver Output Signal Level (Loaded Min.)	Loaded	+/-5V to +/-15V
Driver Output Signal Level (Unloaded Max)	Unloaded	+/-25V
Driver Load Impedance (Ohms)		3k to 7k
Max. Driver Current in High Z State	Power On	N/A
Max. Driver Current in High Z State	Power Off	+/-6mA @ +/-2v
Slew Rate (Max.)		30V/uS
Receiver Input Voltage Range		+/-15V
Receiver Input Sensitivity		+/-3V
Receiver Input Resistance (Ohms)		3k to 7k

Null-Modem Cabel EIA 232



Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
5	5	Signal ground

Hardware Flow Control

Symbol	Circuit	Line state	Remarks
DTR	108/2		Computer ready
DSR	107		Modem ready
RTS	105		Request to send
CTS	106		Ready to send
TxD	103		Start transmission

DTE Data Terminal Equipment – terminal, PC

DCE - Data Circuit-terminating Equipment – Modem

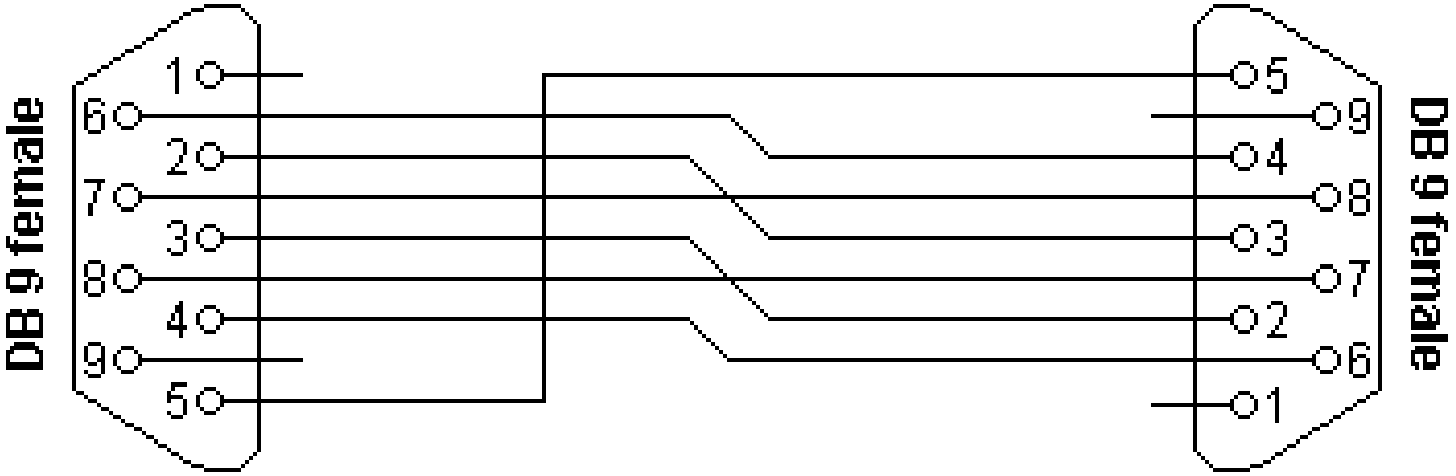
DSR - Data Set Ready - modem

DTR - Data Terminal Ready – terminal

RTS - Request to Send Data

CTS - Clear to Send - ready to send data

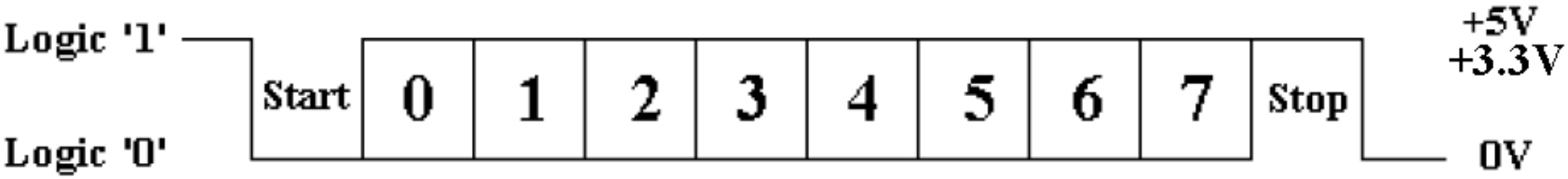
Null-Modem Cabel EIA 232 with Hardware flow Control



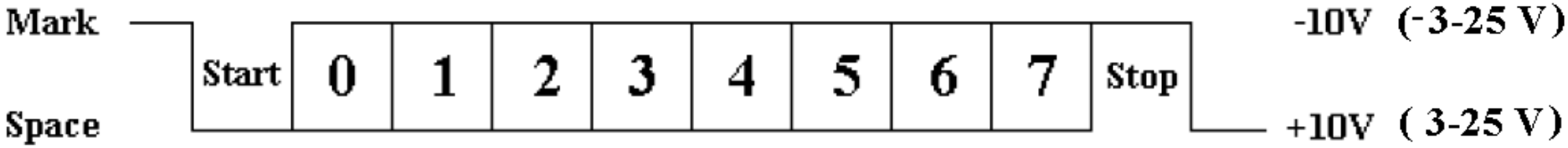
Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
4	6	DTR → DSR
5	5	Signal ground
6	4	DSR ← DTR
7	8	RTS → CTS
8	7	CTS ← RTS

Voltage Levels of EIA RS232

Processor output

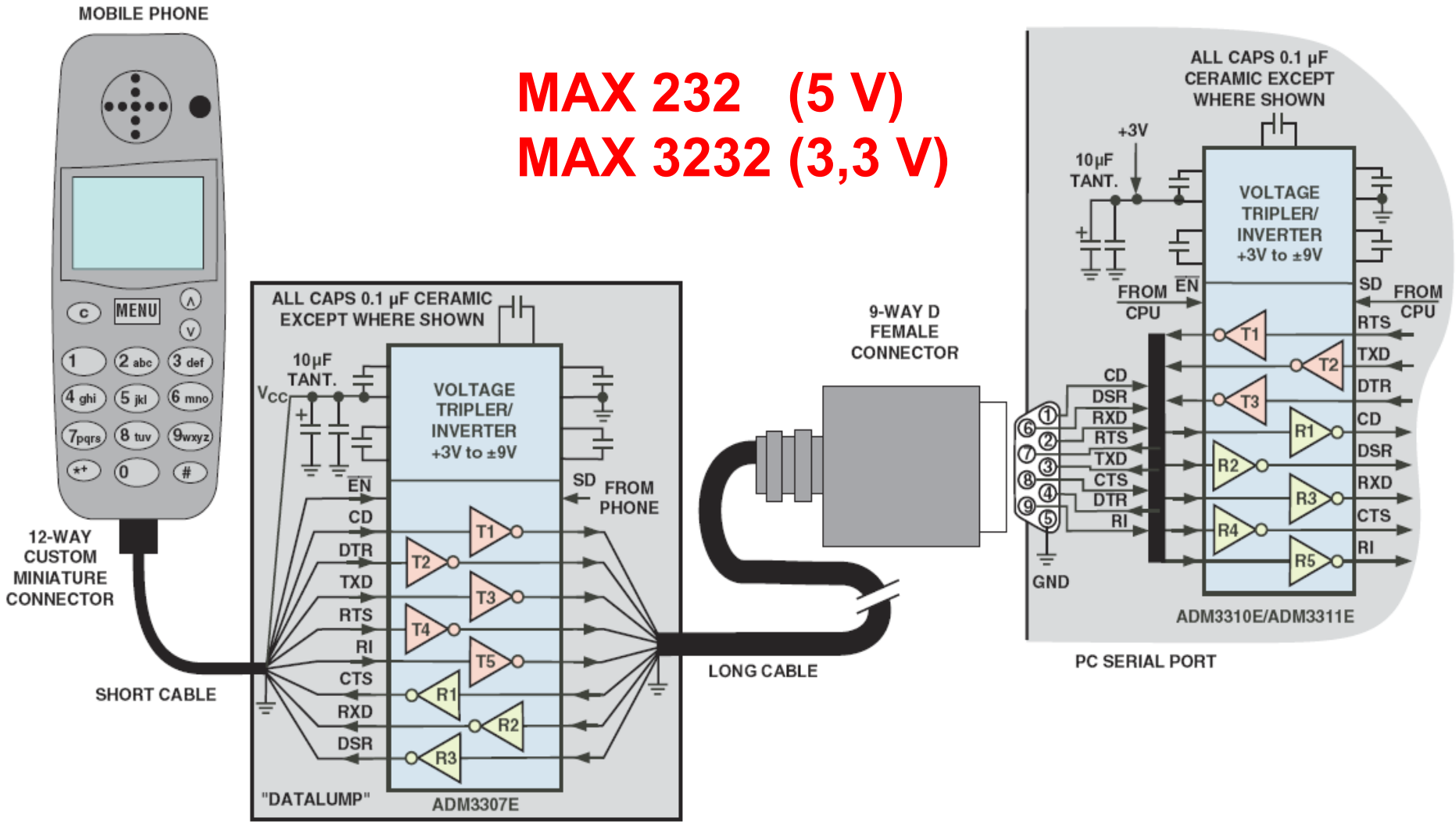


EIA RS 232



RS-232 Logic Waveform

Voltage Levels Translator



Software for EIA RS232 communication

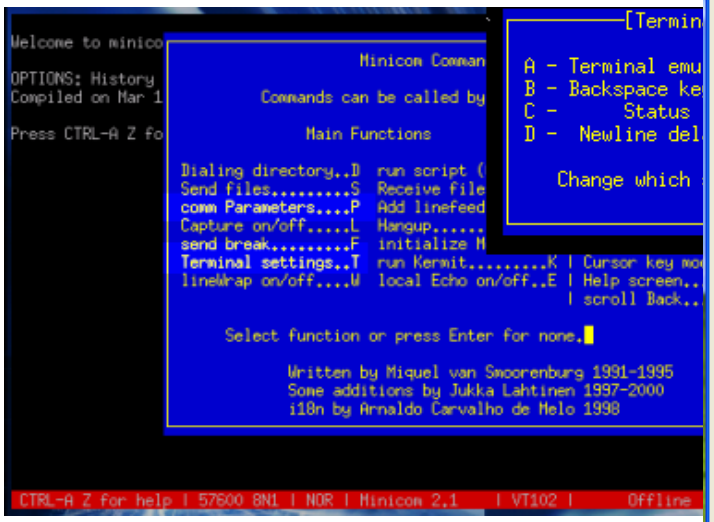
Hyper terminal

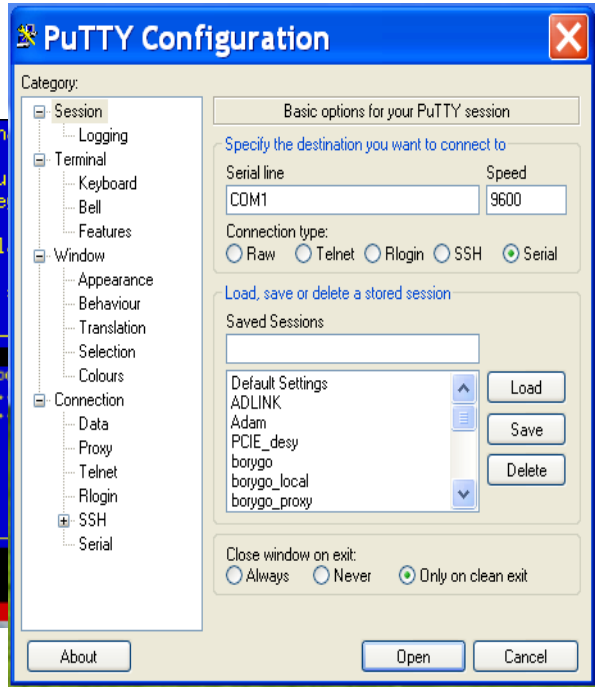
Minicom

ssh

Terminal

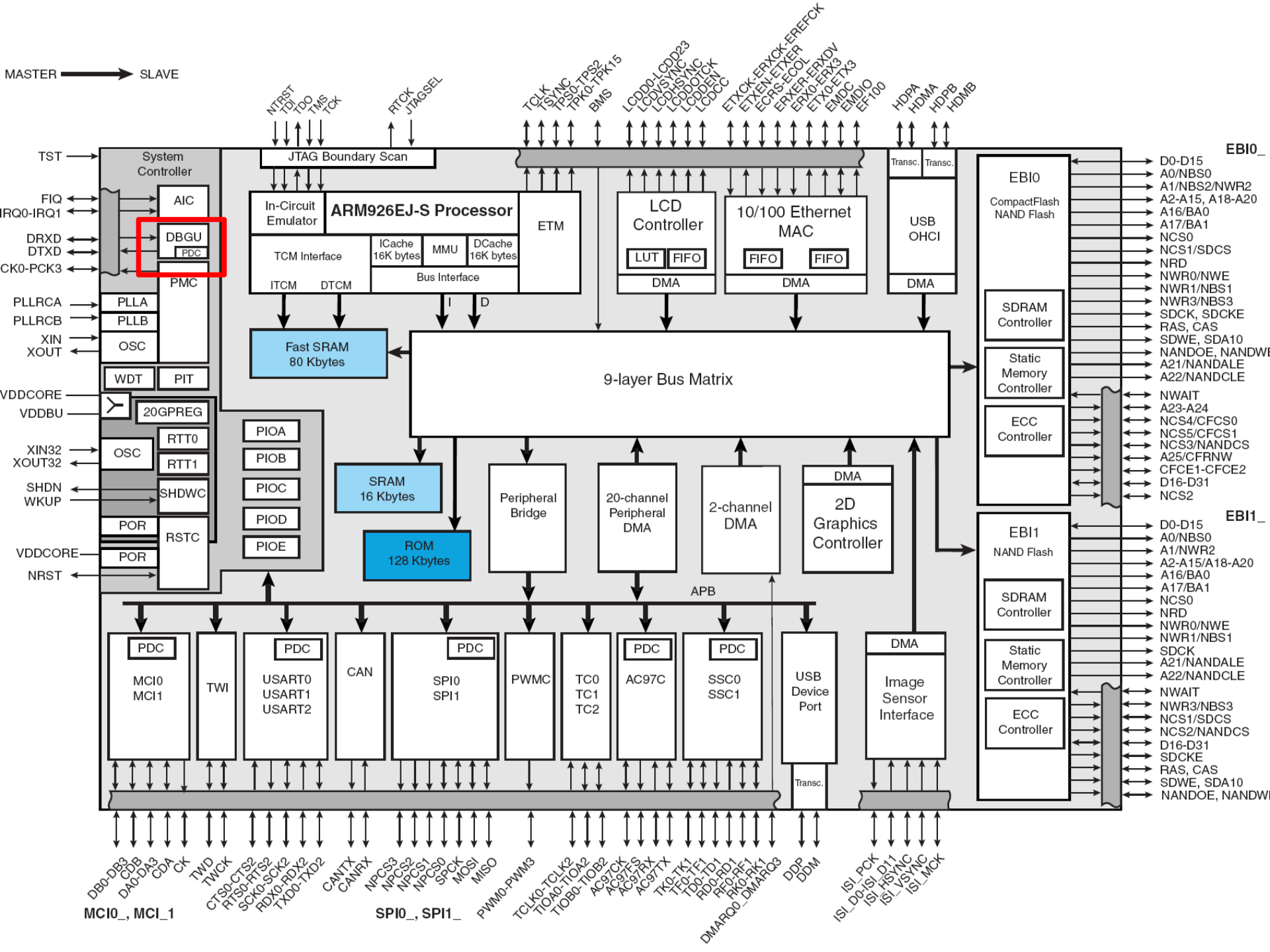
(http://www.elester-pkp.com.pl/index.php?id=92&lang=pl&zoom=0)





Low-Power Universal Asynchronous Receiver Transmitter (LPUART)

(chapter 41)


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
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
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
PLLRCA
PLLRCB
XIN
XOUT
PLLA
PLLB
OSC
WDT
PIT
Fast SRAM 80 Kbytes
9-layer Bus Matrix
VDDCORE
VDDBU
20GPREG
RTT0
RTT1
SHDWC
POR
RSTC
XIN32
XOUT32
OSC
SHDN
WKUP
VDDCORE
POR
NRST
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
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_

Low-Power Universal Asynchronous Receiver Transmitter

Features of LPUART:

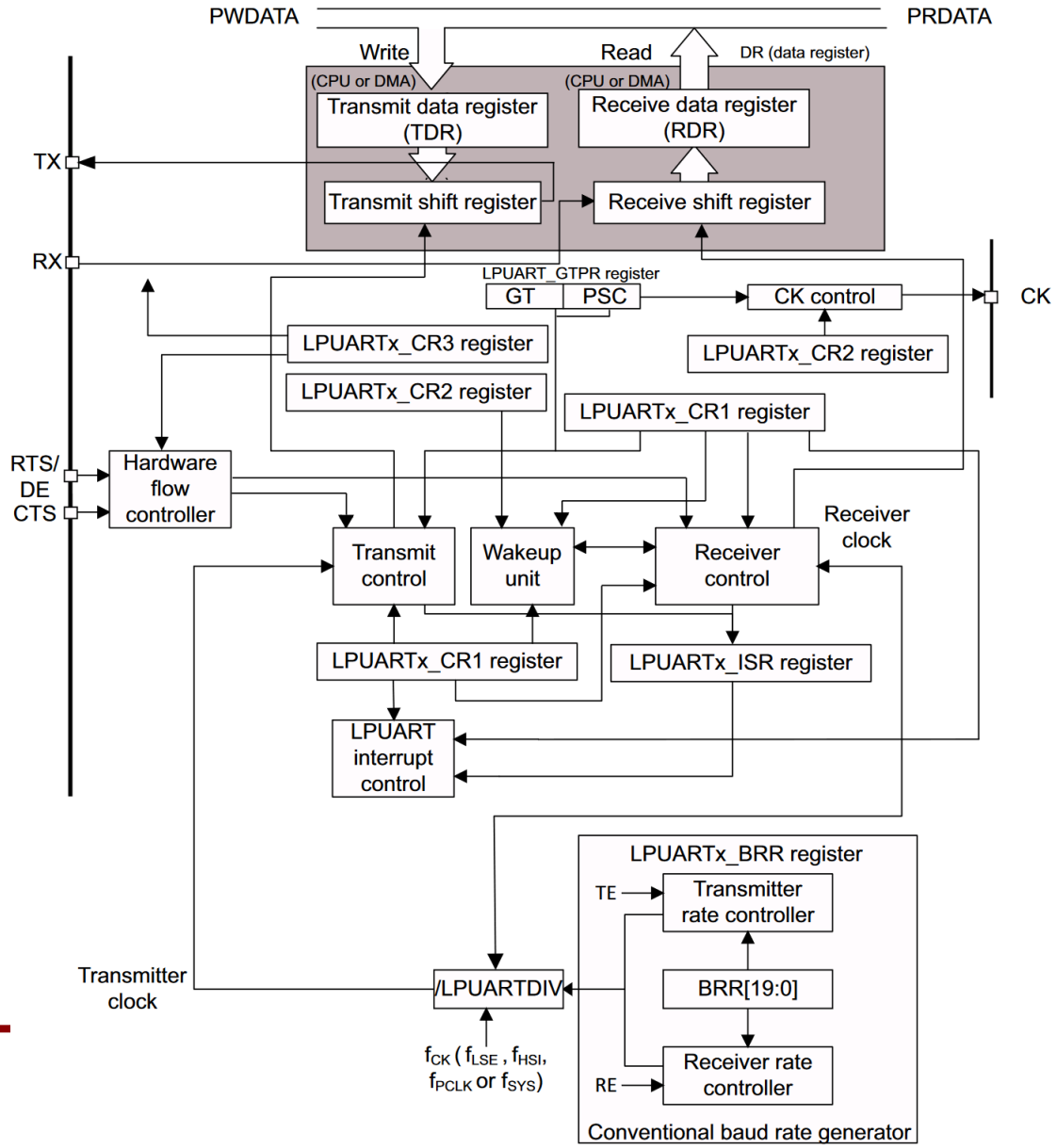
- Full-duplex asynchronous data transmission compatible with RS232 standard
- Programmable data packet size: 7, **8** or 9 bits
- Programmable data order (**LSB** or MSB first)
- Configurable stop bits (**1** or 2), configurable parity bit
- Single-wire half-dulex communication support
- Limited power consumption (available even in STOP mode)
- Hardware support for Direct Memory Access
- Swappable Tx/Rx pin configuration
- Hardware flow control for modem and RS-485 transceiver (Driver Enable)
- Support for hardware flow control: CTS, RTS

Low-Power Universal Asynchronous Receiver Transmitter

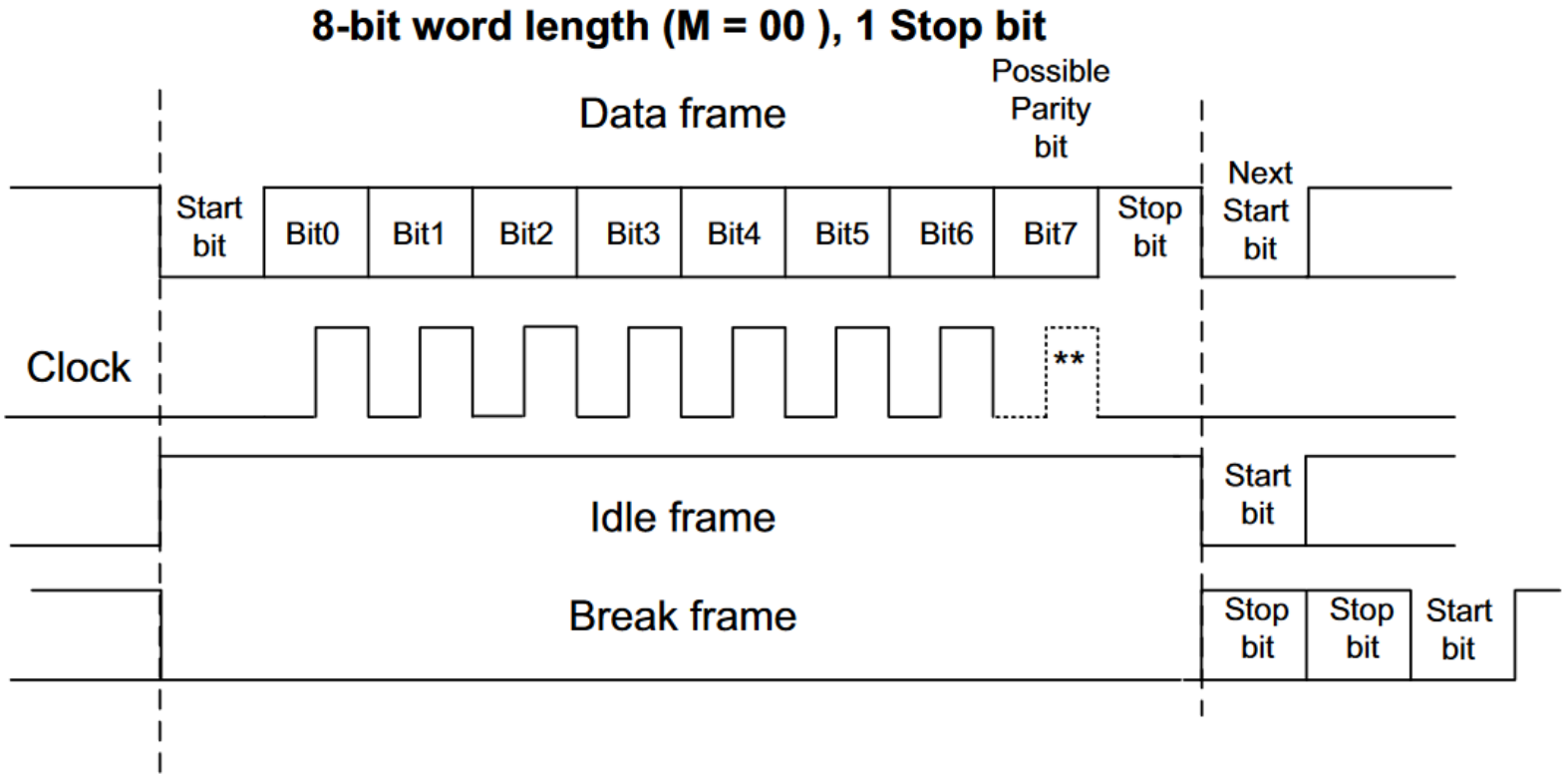
Features of LPUART:

- **Transfer detection flags:**
  - Receive buffer full
  - Transmit buffer empty
  - Busy and end of transmission flags
- **Parity control:**
  - Transmits parity bit
  - Checks parity of received data byte
- **Four error detection flags:**
  - Overrun error
  - Noise detection
  - Frame error
  - Parity error
- Fourteen interrupt sources with flags

Low-Power Universal Asynchronous Receiver Transmitter



LPAURT – 8N1 Typical Data Transmission



Character Transmission Procedure

Configure GPIO Port for Tx function

1. Configure GPIO_C Port to alternate/peripheral function (AF8).

Configuration (for 8N1)

1. Program the M bits in LPUART_CR1 to define the word length (8 bit).
2. Select the desired baud rate using the LPUART_BRR register (115.200 bps).
3. Program the number of stop bits in LPUART_CR2 (1 stop bit).
4. Enable the LPUART by writing the UE bit in LPUART_CR1 register to 1.
5. Disable DMA (DMAT) in LPUART_CR3

Transmission

6. Set the TE bit in LPUART_CR1 to send an idle frame as first transmission.
7. Write the data to send in the LPUART_TDR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the LPUART_TDR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the LPUART is disabled or enters the Halt mode to avoid corrupting the last transmission.

Character Reception Procedure

Configure GPIO Port for Rx function

1. Configure GPIO_C Port to alternate/peripheral function (AF8).

Configuration (for 8N1)

1. Program the M bits in LPUART_CR1 to define the word length (8 bits).
2. Select the desired baud rate using the baud rate register LPUART_BRR (115.200 kbps)
3. Program the number of stop bits in LPUART_CR2 (1 stop bit).
4. Enable the LPUART by writing the UE bit in LPUART_CR1 register to 1.
5. Disable DMA (DMAR) in LPUART_CR3.
6. Set the RE bit LPUART_CR1. This enables the receiver which begins searching for a start bit.

Reception

1. Wait for The RXNE bit to be set.
2. An interrupt is generated if the RXNEIE bit is set.
3. Check for possible error (frame, noise, overrun, parity error)
4. Read data from LPUART_RDR register.

Configuration of LPUART transceiver

static void Open_LPUART (void) {

1. Disable interrupts
2. Configure Receiver/Transmitter
3. Enable Receiver/Transmitter }

Baudrate configuration:

LPUARTDIV value in LPUART_BRR register

f_CK from SysTick_Config (here 4000000 / 1000)

$$\text{Tx/Rx baud} = \frac{256 \times f_{CK}}{\text{LPUARTDIV}}$$

```
void LPUART_Init (void) {
...
RCC->APB1ENR2 |= RCC_APB1ENR2_LPUART1EN;
LPUART1->BRR = (256 * 4000000) / 115200;
LPUART1->CR1 |= USART_CR1_RE | USART_CR1_TE | USART_CR1_UE;
}
```

Read and write via LPUART port

Interrupts are disabled.

```
void SendData (const char Buffer)
{
   while ( data_are_in_buffer ) {
        while ( …TXE... ){};          /* wait until Tx buffer busy – not set TXE flag LPUART_ISR */
        LPUART->TDR = ...             /* write a single char to Transmitter Data Register  */
         }
}

void ReadData (char *Buffer, unsigned int Size){
        do {
        While ( ...RXNE... ){};       /* wait until data available, RXNE is set in ISR register */
        Buffer[...] = LPUART->RDR;    /* read data from Receiver Data Register */
        } while ( …read_enough_data... )
}
```

Clock for LPUART

- Enable clock for LPUART and GPIOC

6.4.20 APB1 peripheral clock enable register 2 (RCC_APB1ENR2)

Address offset: 0x5C

Reset value: 0x00000 0000

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LPTIM2 EN	Res.	Res.	SWP MI1 EN	I2C4EN	LP UART1 EN
													rw	rw	rw

Bits 31:6 Reserved, must be kept at reset value.

Bit 5 **LPTIM2EN** Low power timer 2 clock enable
Set and cleared by software.
0: LPTIM2 clock disable
1: LPTIM2 clock enable

Bits 4:3 Reserved, must be kept at reset value.

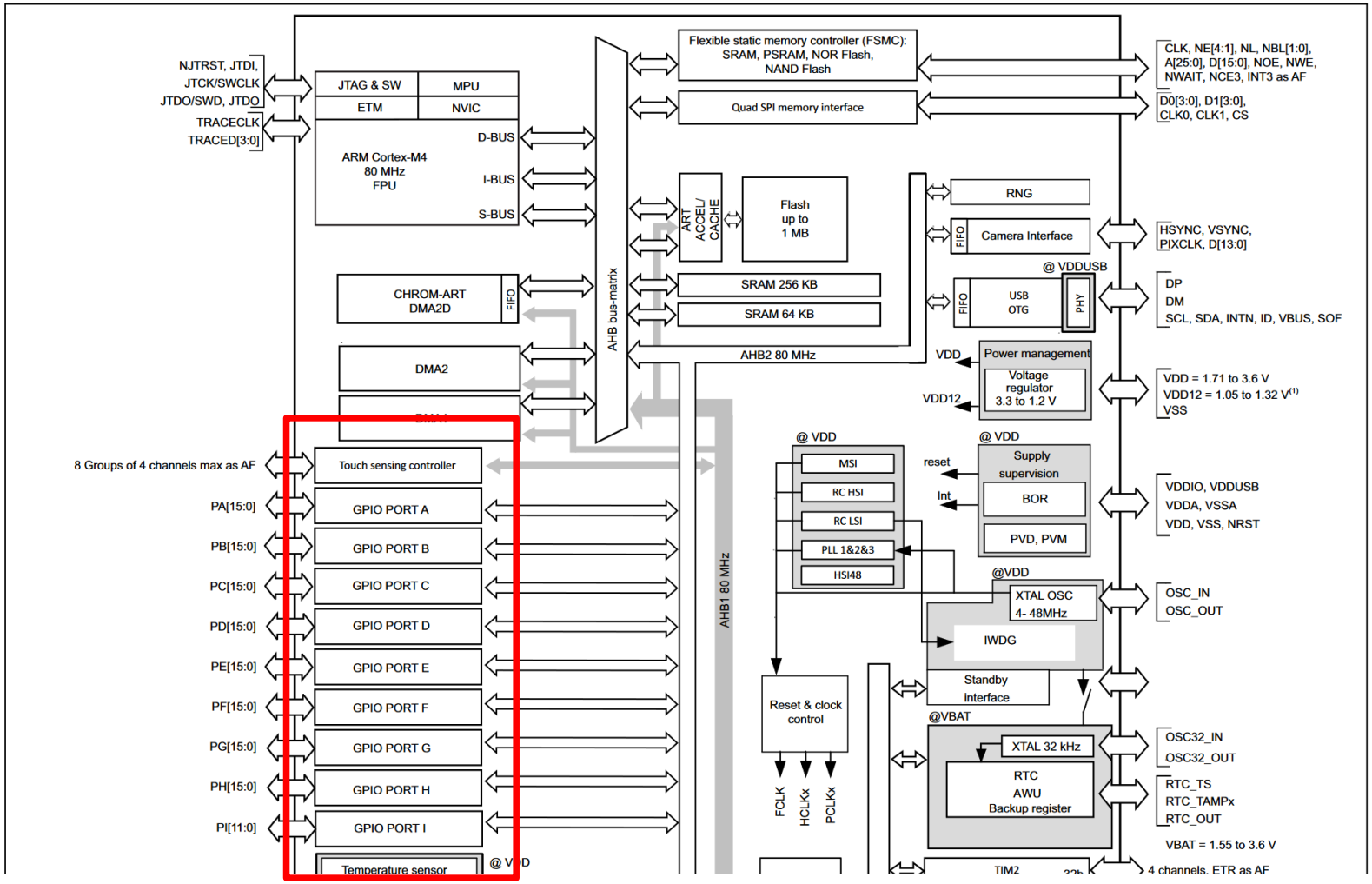
STM32L496xx Data sheet, electrical parameters

Table 17. Alternate function AF8 to AF15(1) (continued)

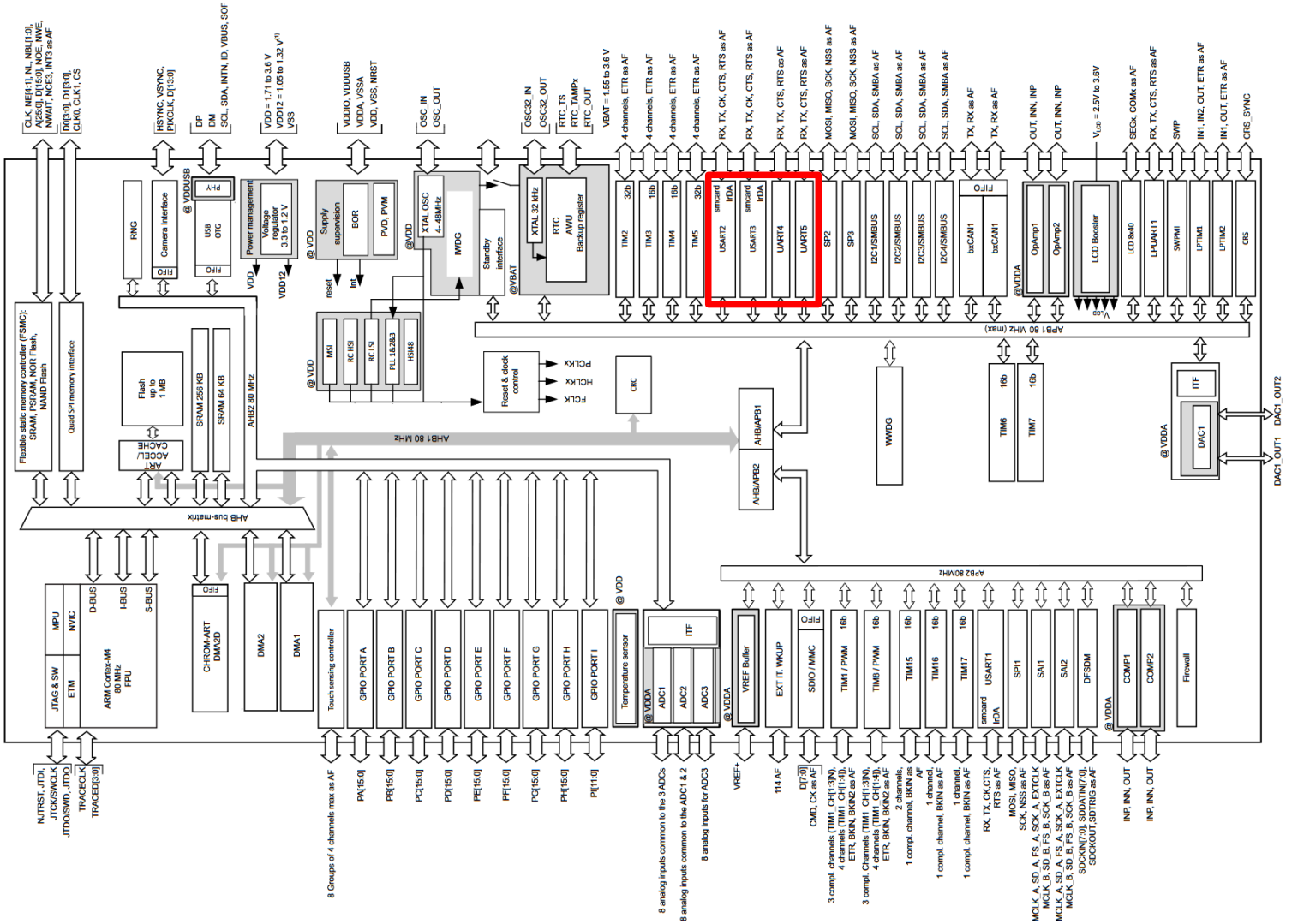
Port		AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		UART4/5/ LPUART1/ CAN2	CAN1/TSC	CAN2/ OTG_FS/DCMI/ QUADSPI	LCD	SDMMC/ COMP1/2/FM C/SWPMI1	SAI1/2	TIM2/15/16/17/ LPTIM2	EVENOUT
Port C	PC0	LPUART1_RX	-	-	LCD_SEG18	-	-	LPTIM2_IN1	EVENTOUT
	PC1	LPUART1_TX	-	QUADSPI_BK2_IO0	LCD_SEG19	-	SAI1_SD_A	-	EVENTOUT
	PC2		-	QUADSPI_BK2_IO1	LCD_SEG20	-	-	-	EVENTOUT
	PC3	-	-	QUADSPI_BK2_IO2	LCD_VLCD	-	SAI1_SD_A	LPTIM2_ETR	EVENTOUT
	PC4	-	-	QUADSPI_BK2_IO3	LCD_SEG22	-	-	-	EVENTOUT
	PC5	-	-	-	LCD_SEG23	-	-	-	EVENTOUT
	PC6	-	TSC_G4_IO1	DCMI_D0	LCD_SEG24	SDMMC1_D6	SAI2_MCLK_A	-	EVENTOUT
	PC7	-	TSC_G4_IO2	DCMI_D1	LCD_SEG25	SDMMC1_D7	SAI2_MCLK_B	-	EVENTOUT
	PC8	-	TSC_G4_IO3	DCMI_D2	LCD_SEG26	SDMMC1_D0	-	-	EVENTOUT
	PC9	-	TSC_G4_IO4	OTG_FS_NOE	LCD_SEG27	SDMMC1_D1	SAI2_EXTCLK	TIM8_BKIN2_C OMP1	EVENTOUT
	PC10	UART4_TX	TSC_G3_IO2	DCMI_D8	LCD_COM4/L CD_SEG28/L CD_SEG40	SDMMC1_D2	SAI2_SCK_B	-	EVENTOUT
	PC11	UART4_RX	TSC_G3_IO3	DCMI_D4	LCD_COM5/L CD_SEG29/L CD_SEG41	SDMMC1_D3	SAI2_MCLK_B	-	EVENTOUT
	PC12	UART5_TX	TSC_G3_IO4	DCMI_D9	LCD_COM6/L CD_SEG30/L CD_SEG42	SDMMC1_CK	SAI2_SD_B	-	EVENTOUT

STM32L496xx Data sheet, electrical parameters

Figure 1. STM32L496xx block diagram



STM32L496xx Data sheet, electrical parameters



Programming Timer 6 with HAL - Structure

```
typedef struct {
  TIM_TypeDef                 *Instance;       /* Reg. base address */
  TIM_Base_InitTypeDef        Init;           /* TIM Time Base par.*/
  HAL_TIM_ActiveChannel       Channel;        /* Active channel    */
  DMA_HandleTypeDef           *hdma[7];       /* DMA Handlers arr. */
  HAL_LockTypeDef             Lock;           /* Locking object    */
  __IO HAL_TIM_StateTypeDef   State;          /* TIM operat. state */
  __IO HAL_TIM_ChannelStateTypeDef  ChannelState[6];  /* TIM channel state */
  __IO HAL_TIM_ChannelStateTypeDef  ChannelNState[4]; /* TIM compl. ...    */
  __IO HAL_TIM_DMABurstStateTypeDef DMABurstState;    /* DMA burst state   */
  ...
  ...
} TIM_HandleTypeDef;
```

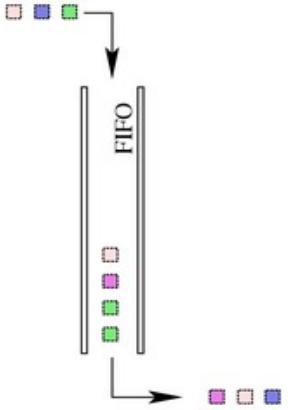
Data structures

Stack (1)

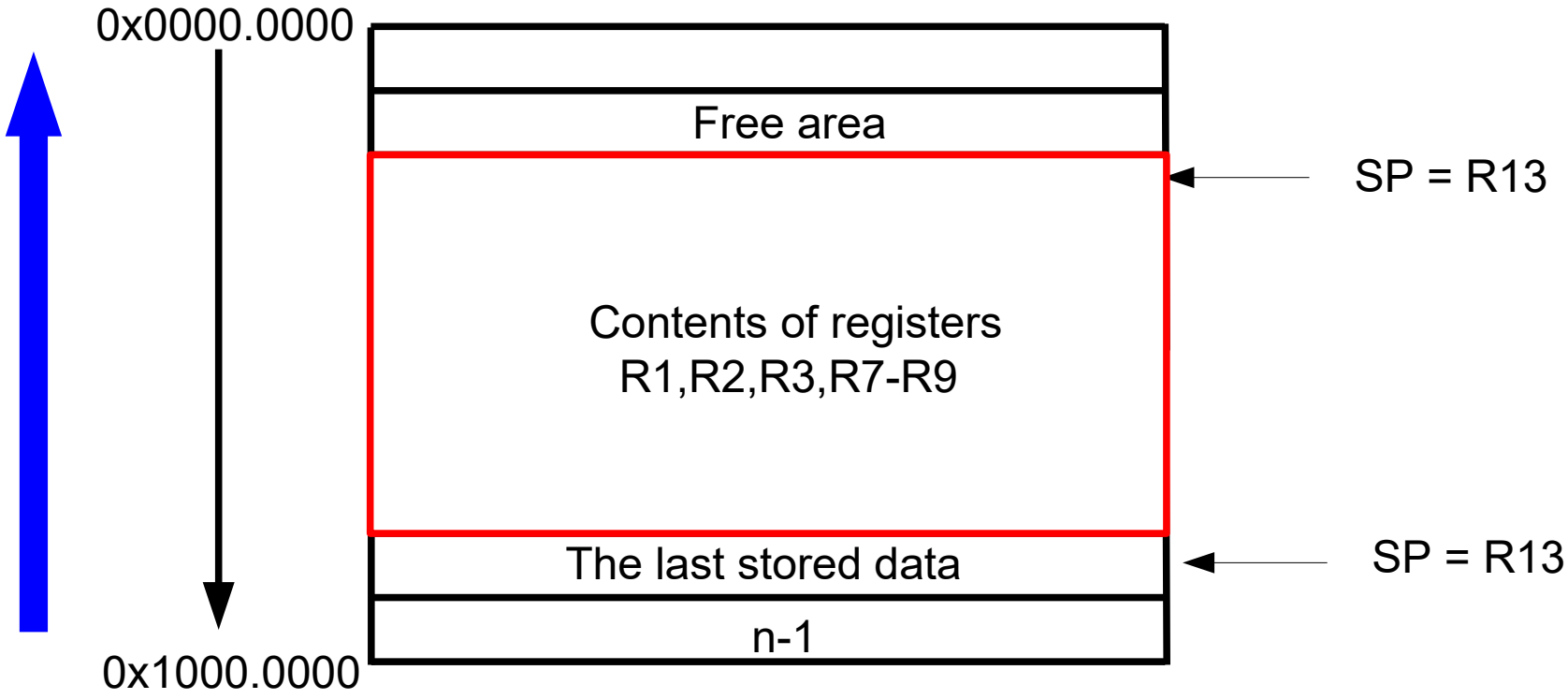
Stack or LIFO (Last-In, First-Out) – abstract data type and data structure. A stack can have any abstract data type as an element, but it is characterized by only two fundamental operations: push and pop. The push operation adds to the top of the list, hiding any items already on the stack, or initializing the stack if it is empty. The pop operation removes an item from the top of the list, and returns this value to the caller. A pop either reveals previously concealed items, or results in an empty list.

FIFO (First In, First Out) – a linear buffer, the opposite structure to stack. The first element placed into FIFO is immediately transferred to the end of the queue. Therefore the first element stored in FIFO is supposed to be processed first.

First–in First–out (FIFO)

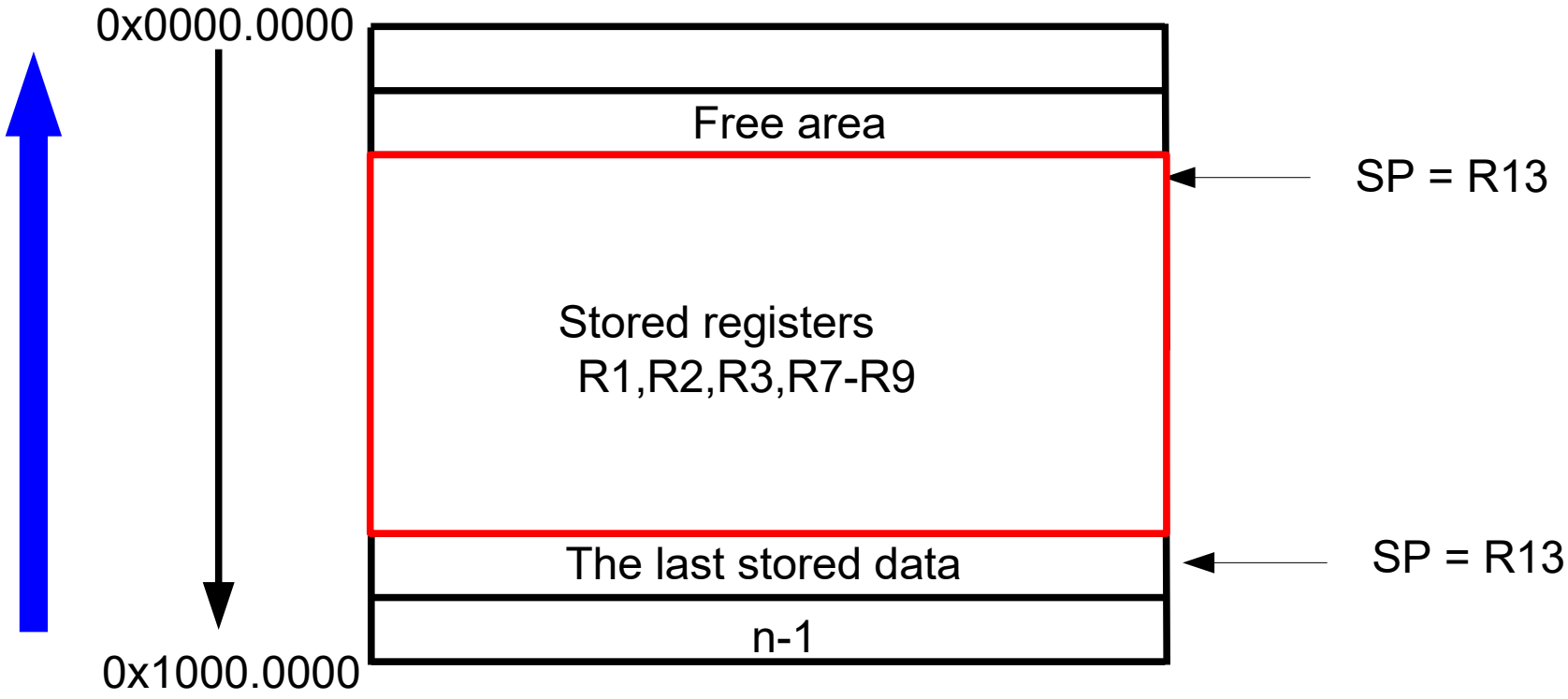


Stack – push data



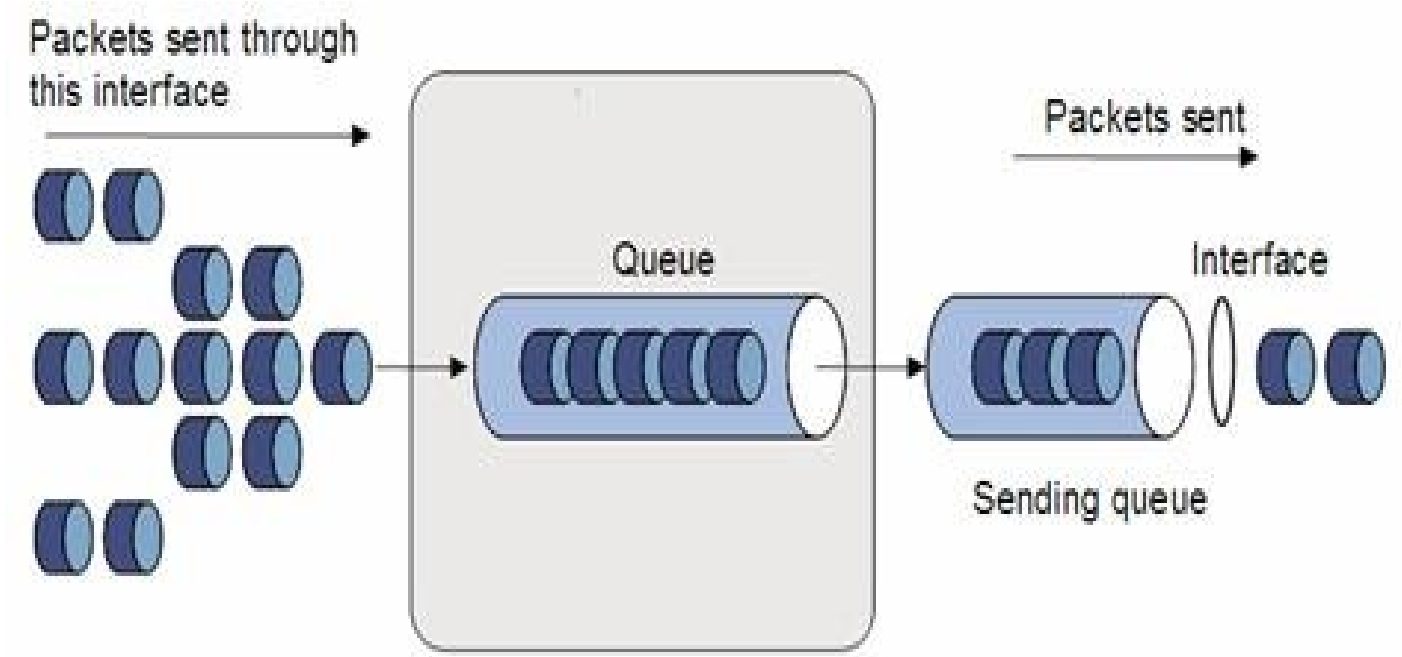

```
STMDB   SP!, {registers list}
STMDB   SP!, {R1,R2,R3,R7-R9}     | decrease SP by 24, stores 8 registers on stack
```

Stack - pop



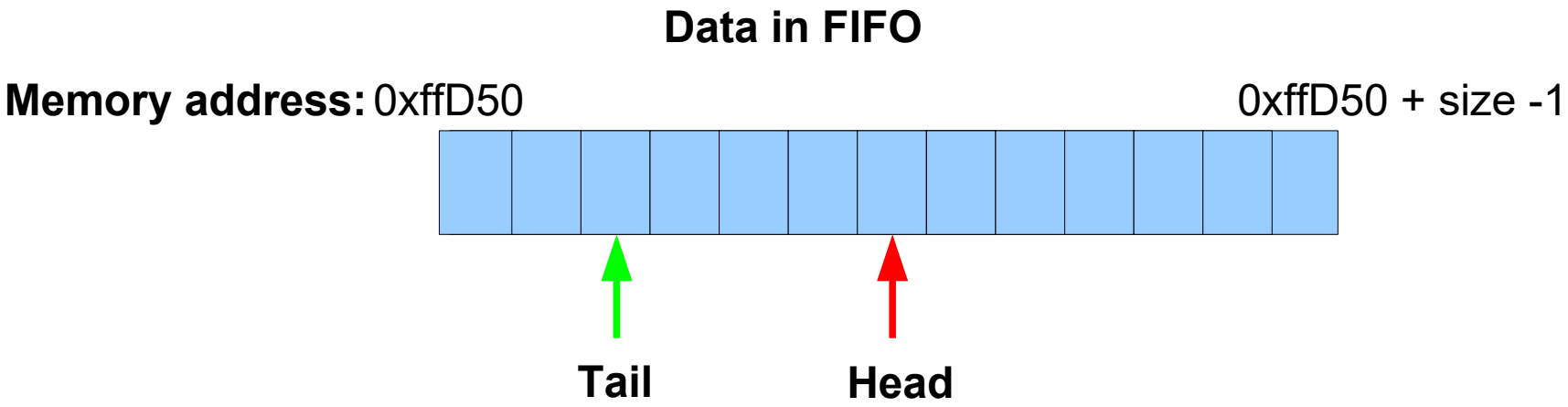

```
LDMIA   SP!, {list of registers}
LDMIA   SP!, {R1,R2,R3,R7-R9}    | increase SP by 24, recover 8 registers from stack
```

FIFO (1)



- A few different applications can try to write data into FIFO queue. In such a case a semaphore can be used to control access during writing data to queue.
- Data are read from queue in the same order as was written

FIFO (2)



Write data to FIFO:

- Increase Head by one, write data.

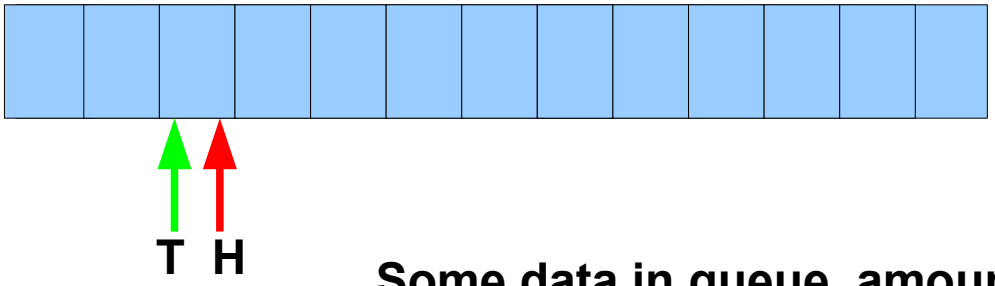
Read data from FIFO:

- Read data, increase Tail by one.

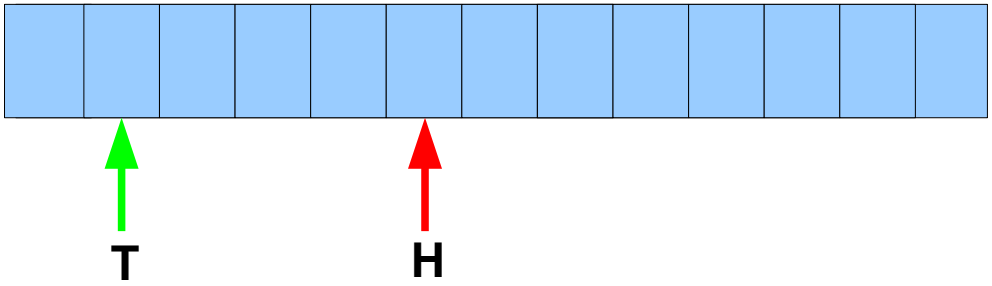
When the Tail or Head points the last element in queue the pointer is not increased (zero is written to the pointer) - circular buffer.

FIFO (3)

Empty FIFO T = H

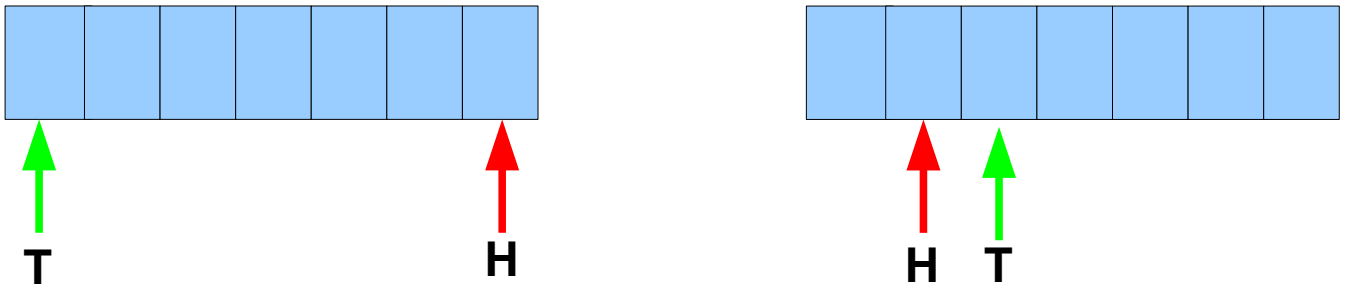


Some data in queue, amount of data = H – T



Full FIFO

(T = 0) & (H = Size) or T – H = 1



FIFO – implementation in C (1)

```
#define BUFFERSIZE   0xFF                    /* FIFO buffer size and mask */
typedef struct FIFO {
    char buffer [BUFFERSIZE+1];
    unsigned int head;
    unsigned int tail;
};
void FIFO_Init (struct FIFO *Fifo);
void FIFO_Empty (struct FIFO *Fifo);
int FIFO_Put (struct FIFO *Fifo, char Data);
int FIFO_Get (struct FIFO *Fifo, char *Data)


void FIFO_Init (struct FIFO *Fifo){
        Fifo->head=0;
        Fifo->tail=0;

        /* optional: initialize data in buffer with 0 */
}
```

FIFO – implementation in C (2)

```
void FIFO_Empty (struct FIFO *Fifo){
        Fifo->head = Fifo->tail;                                    /* now FIFO is empty*/
}
int FIFO_Put (struct FIFO *Fifo, char Data){
        if ((Fifo->tail-Fifo->head)==1 || (Fifo->tail-Fifo->head)==BUFFERSIZE)){
        return -1; };                                               /* FIFO overflow */
        Fifo->buffer[Fifo->head] = Data;
        Fifo->head = (Fifo->head + 1) & BUFFERSIZE;
        return 1;                                                   /* Put 1 byte successfully */
}
int FIFO_Get (struct FIFO *Fifo, char *Data){
        If ((TxFifo.head!=TxFifo.tail)){
        *Data = Fifo->buffer[Fifo->tail];
        Fifo->tail = (Fifo->tail + 1) & BUFFERSIZE;
        return 1;                                                   /* Get 1 byte successfully */
        } else return -1;                                           /* No data in FIFO */
}
```

FIFO – traps

```
void FIFO_Empty (struct FIFO *Fifo){
        Fifo->head = Fifo->tail;                                    /* now FIFO is empty*/
}
int FIFO_Put (struct FIFO *Fifo, char Data){
        if ((Fifo->tail-Fifo->head)==1 || (Fifo->tail-Fifo->head)==BUFFERSIZE)){
        return -1; };                                               /* FIFO overflow */
        Fifo->buffer[Fifo->head++] = Data;
        Fifo->head = Fifo->head & BUFFERSIZE;                       /* be carefull with interrupts */
        return 1;                                                   /* Put 1 byte successfully */
}
int FIFO_Get (struct FIFO *Fifo, char *Data){
        If ((TxFifo.head!=TxFifo.tail)){
        *Data = Fifo->buffer[Fifo->tail++];
        Fifo->tail &= BUFFERSIZE;                                   /* be carefull with interrupts */
        return 1;                                                   /* Get 1 byte successfully */
        } else return -1;                                           /* No data in FIFO */
}
```