UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

System architecture

utbm

Nicolas Lacaille

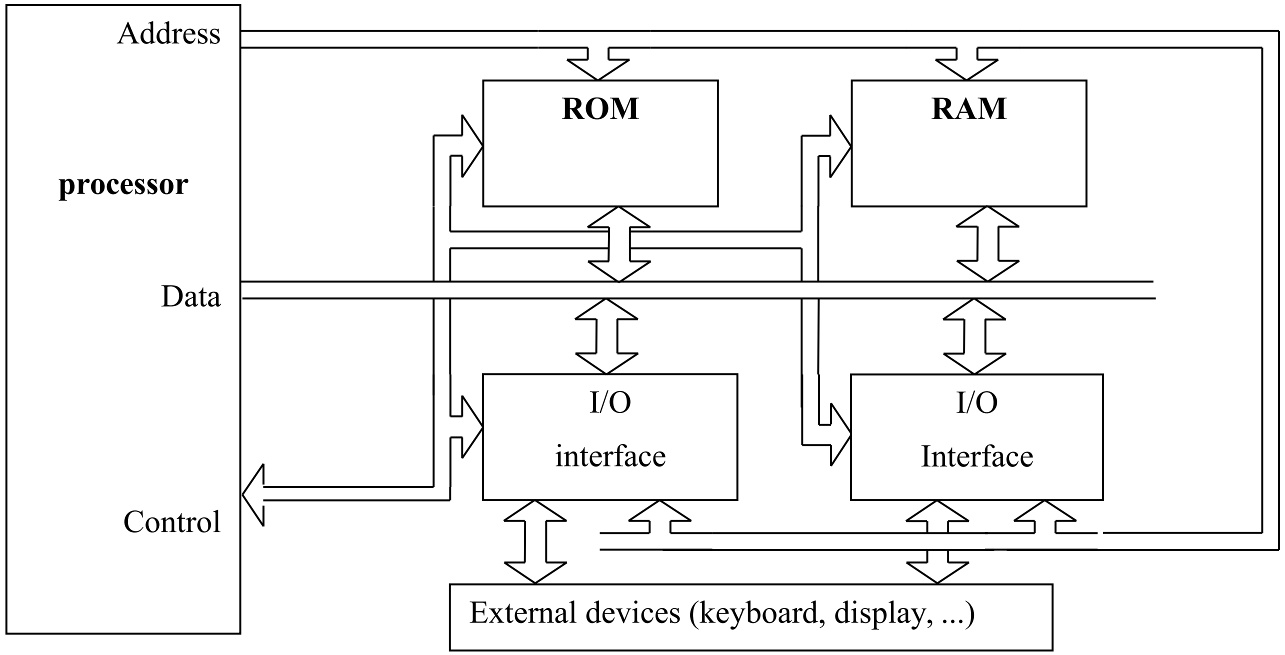
Presentation

- Computer system is made up of
  - Microprocessor
  - Clock
  - Memory
- For each cycle processor
  - Fetch an instruction from memory (program)
  - Execute instruction
- Instruction can do
  - Data processing
  - Move data from/to memory
  - Branch to an other address in memory

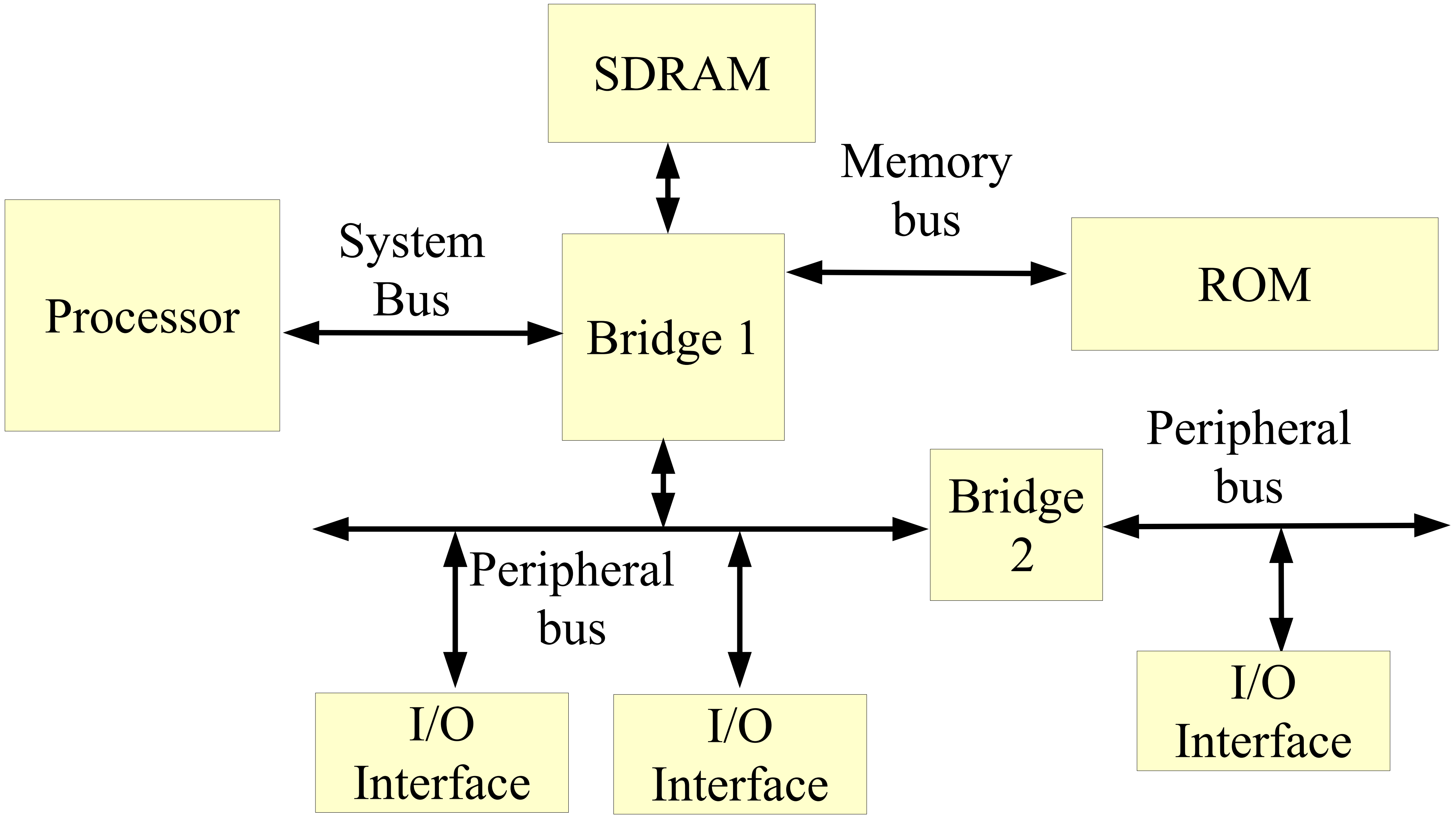
External communication

- Since a microprocessor can only move data to/from memory external communication can only be done with special memory device : interfaces
- To exchange data with external peripherals, processor need interfaces
  - Processor side interface are memory device called I/O port or memory mapped registers
  - User side interface are specialized device for specific peripheral
- Applications control peripheral through the I/O port of interfaces (exchange data, control the device, knowing the state of the device, ...)

Simple system



More complex system



Example

- An embedded system control an industrial process
  - On one side you have captor connected to input ports
  - On the other side you have motor unit connected to output ports
- The application do cycle made up of
  - Reading data from the input port (at a known memory address)
  - Computing the data
  - Writing new data to the output port

Memory mapped register

- Input / output port are device register that can be acceded in the physical memory map
  - Memory mapped register
- Memory mapped register, most of time, don't work like standard memory use for variable
  - Read only (RO) or Write only (WO) registers
  - Variable size (8, 16, 32 bits)
  - Values can change outside the running application
- The correct type must be used in C language
- The 'volatile' term must be used (signals compiler that the variable can be changed outside the program)

Samples

- Using a simple pointer to access the I/O port

```
volatile unsigned *port = (unsigned int *) 0x40000000;
/*for an output port */     *port = value ;
/* for an input port */     variable = *port;
```

- C macro

```
#define port *(volatile unsigned int *) 0x40000000
port = value ;     /* or value = port */
```

- Using a macro provided by the kernel
  - *HAL_WRITE_UINT32(address, value)* or *inl(int)* or …
- Using structure

```
struct port {
        volatile unsigned config;
        volatile unsigned data;
        } *portA;

portA = (struct port *) 0x40000000;
portA->config = value1;
value2 = portA->data;
```

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

ARM7TDMI microprocessor

utbm

Presentation

- 32 bits general purpose architecture
- 3 stages pipeline RISC architecture
- 32 bits instructions (ARM mode) or 16 bits instructions for code compression (THUMB mode)
- Register to register and load/store architecture
- Single bus for instruction and data
- Low consumption (for embedded system)

Execution modes

- 7 execution modes
  - User (usr)
  - Supervisor (svc) privilieged for OS
  - FIQ (fiq) : Fast Interrupt
  - IRQ (irq) : Normal Interrupt
  - System (sys) privileged for OS
  - Abort (abt) addressing's fault
  - Undefined : not defined instruction
- Changed are done by software or on special event (exceptions)
- Modes out of usr are privileged modes

Registers

- 31 general purpose registers
- Only 16 registers can be used in each mode
  - r0 → r15
- In all mode
  - r15 is program counter (pc)
  - r14 is the link register (lr)
  - r13 is the stack pointer (sp)
- By convention (AAPCS/EABI)
  - r4 to r11 are variable registers (v1 to v8)
  - r0 to r3 are scratch/argument registers (a1 to a4)

Modes

	Privileged modes					
		Exception modes				
User	**System**	**Supervisor**	**Abort**	**Undefined**	**Interrupt**	**Fast interrupt**
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	◺ R8_fiq
R9	R9	R9	R9	R9	R9	◺ R9_fiq
R10	R10	R10	R10	R10	R10	◺ R10_fiq
R11	R11	R11	R11	R11	R11	◺ R11_fiq
R12	R12	R12	R12	R12	R12	◺ R12_fiq
R13	R13	◺ R13_svc	◺ R13_abt	◺ R13_und	◺ R13_irq	◺ R13_fiq
R14	R14	◺ R14_svc	◺ R14_abt	◺ R14_und	◺ R14_irq	◺ R14_fiq
PC	PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		◺ SPSR_svc	◺ SPSR_abt	◺ SPSR_und	◺ SPSR_irq	◺ SPSR_fiq

◺ *indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode*

PSR

- PSR : program state register
  - cpsr : current program state register
  - spsr : saved program state register (only present in privileged mode)
- CPSR contains
  - ALU flags (C,V, Z, N)
  - I and F flags for allowing interrupts
  - Processor mode

PSR

31	30	29	28	27	26 25	24	23 20	19 16	15 10	9	8	7	6	5	4 0
N	Z	C	V	Q	Res	J	RESERVED	GE[3:0]	RESERVED	E	A	I	F	T	M[4:0]

Mode	***M[4:0]***
User	10000
FIQ	10001
IRQ	10010
Supervisor	10011
Abort	10111
Undefined	11011
System	11111

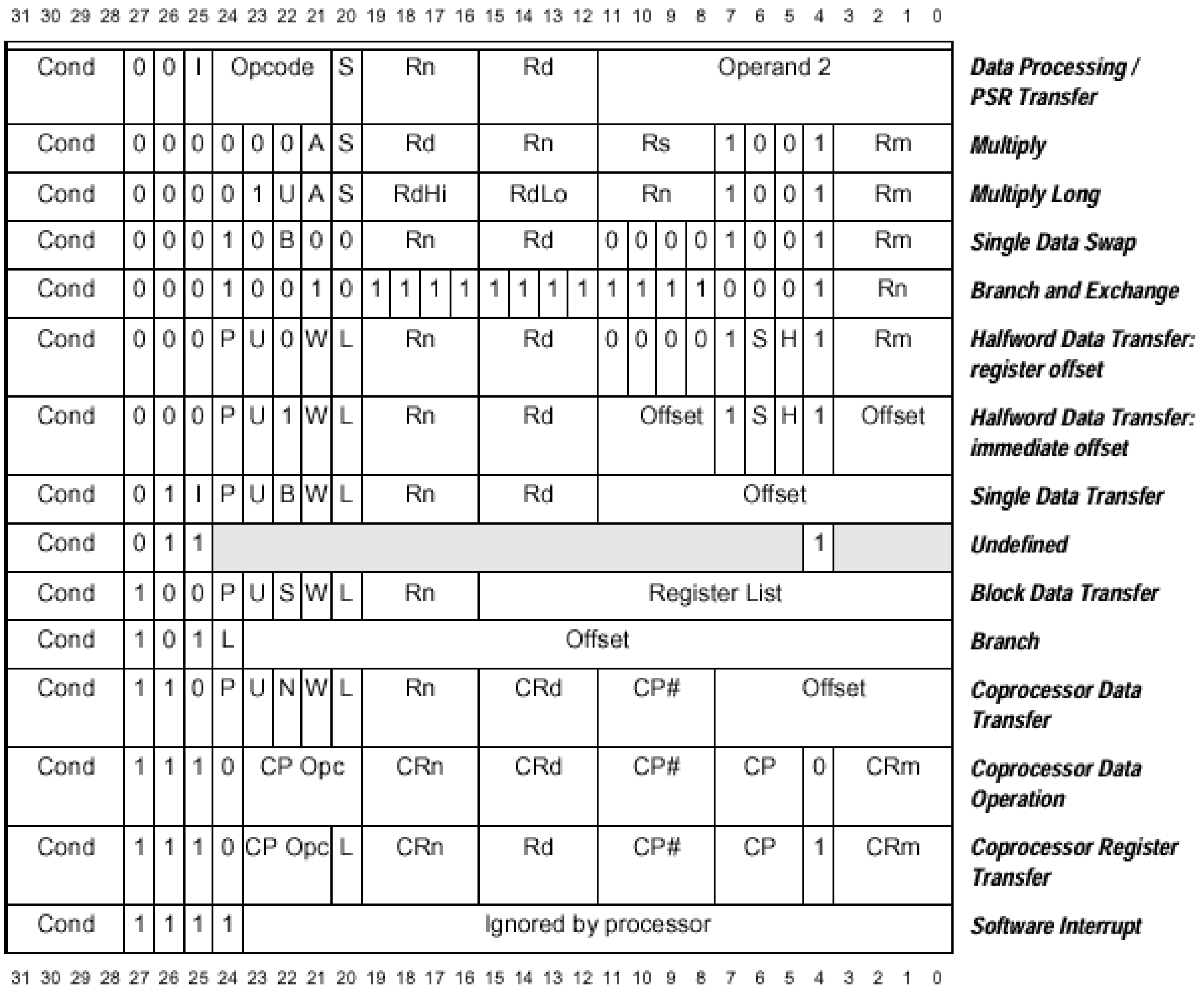
3 stages pipeline

- 3 operations per cycles (instructions parallelism)
  - Fetch : instructions fetch
  - Decode : operands fetch
  - Execute : integer operation and store
- PC points 2 instructions forward the executing one (fetch)
- No branch prediction

ARM instruction set

- 32 bits RISC instructions :
  - Register to register or register to immediate operand operations
  - CPSR flags are not changed except if explicitly asked
  - Load/store instruction for moving data from register to/from memory (register based addressing)
- Most instructions can be conditionally executed

ARM instructions



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	
Cond				0	0	I	Opcode				S	Rn				Rd				Operand 2												*Data Processing / PSR Transfer*
Cond				0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				*Multiply*
Cond				0	0	0	0	1	U	A	S	RdHi				RdLo				Rn				1	0	0	1	Rm				*Multiply Long*
Cond				0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm				*Single Data Swap*
Cond				0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				*Branch and Exchange*
Cond				0	0	0	P	U	0	W	L	Rn				Rd				0	0	0	0	1	S	H	1	Rm				*Halfword Data Transfer: register offset*
Cond				0	0	0	P	U	1	W	L	Rn				Rd				Offset				1	S	H	1	Offset				*Halfword Data Transfer: immediate offset*
Cond				0	1	I	P	U	B	W	L	Rn				Rd				Offset												*Single Data Transfer*
Cond				0	1	1	▨																				1	▨				*Undefined*
Cond				1	0	0	P	U	S	W	L	Rn				Register List																*Block Data Transfer*
Cond				1	0	1	L	Offset																								*Branch*
Cond				1	1	0	P	U	N	W	L	Rn				CRd				CP#				Offset								*Coprocessor Data Transfer*
Cond				1	1	1	0	CP Opc				CRn				CRd				CP#				CP			0	CRm				*Coprocessor Data Operation*
Cond				1	1	1	0	CP Opc			L	CRn				Rd				CP#				CP			1	CRm				*Coprocessor Register Transfer*
Cond				1	1	1	1	Ignored by processor																								*Software Interrupt*
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	

THUMB instruction set

- 16 bits : instructions are more constrained
  - Only 8 registers are code reachable
  - Shortest immediate operands
  - …
- Flags are always updated (no more explicitly)
- Only branch instruction can be conditional
- …

THUMB instruction

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	Op		Offset5					Rs			Rd			Move shifted register
2	0	0	0	1	1	I	Op	Rn/offset3			Rs			Rd			Add/subtract
3	0	0	1	Op		Rd			Offset8								Move/compare/add /subtract immediate
4	0	1	0	0	0	0	Op				Rs			Rd			ALU operations
5	0	1	0	0	0	1	Op		H1	H2	Rs/Hs			Rd/Hd			Hi register operations /branch exchange
6	0	1	0	0	1	Rd			Word8								PC-relative load
7	0	1	0	1	L	B	0	Ro			Rb			Rd			Load/store with register offset
8	0	1	0	1	H	S	1	Ro			Rb			Rd			Load/store sign-extended byte/halfword
9	0	1	1	B	L	Offset5					Rb			Rd			Load/store with immediate offset
10	1	0	0	0	L	Offset5					Rb			Rd			Load/store halfword
11	1	0	0	1	L	Rd			Word8								SP-relative load/store
12	1	0	1	0	SP	Rd			Word8								Load address
13	1	0	1	1	0	0	0	0	S	SWord7							Add offset to stack pointer
14	1	0	1	1	L	1	0	R	Rlist								Push/pop registers
15	1	1	0	0	L	Rb			Rlist								Multiple load/store
16	1	1	0	1	Cond				Soffset8								Conditional branch
17	1	1	0	1	1	1	1	1	Value8								Software Interrupt
18	1	1	1	0	0	Offset11											Unconditional branch
19	1	1	1	1	H	Offset											Long branch with link
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

ARM7 exceptions

Exception	Description
Reset	Reset pin activated
Undefined Instruction	Special instruction code
Software Interrupt (SWI)	Instruction code used to generate exception, system call
Prefetch Abort	Memory access violation during fetch
Data Abort	Data memory access violation
IRQ	IRQ pin activated
FIQ	FIQ pin activated

Principe

- When an exception occurs
  - PC-4 is saved in lr_mode
  - CPSR is saved in SPSR_mode
  - CPSR is changed
    - Mode becomes : svc, irq, fiq, data or prefetch abort depending the exception
    - I bit is set (IRQ not allowed) for all exceptions
    - F bit is set if the exception is a FIQ or reset
  - PC is loaded with exception vector
    - Address between 0x0 (reset) to 0x1C (FIQ)

Exception vectors

address	Exception	Processor's mode	Priority
0x00	Reset	Supervisor (svc)	1
0x04	Undefined Instruction	Undef	6
0x08	Software Interrupt (SWI)	Supervisor (svc)	6
0x0C	Prefetch Abort	Abort	5
0x10	Data Abort	Abort	2
0x14	Reserved		
0x18	IRQ (Interrupt)	irq	4
0x1C	FIQ (Fast Interrpt)	fiq	3

- For arm processor each mode has their own stack pointer
  - Allow the exception handler to save data in its own memory area without corrupting the application data
- During the execution of the exception handler no interrupt are allowed
  - No peripheral or system services can be serviced without re-enable interrupt
- Exception handler are architecture specific and differs from standard function
  - Exception routine need special entry and exit code that can be written in asm or provided by a library

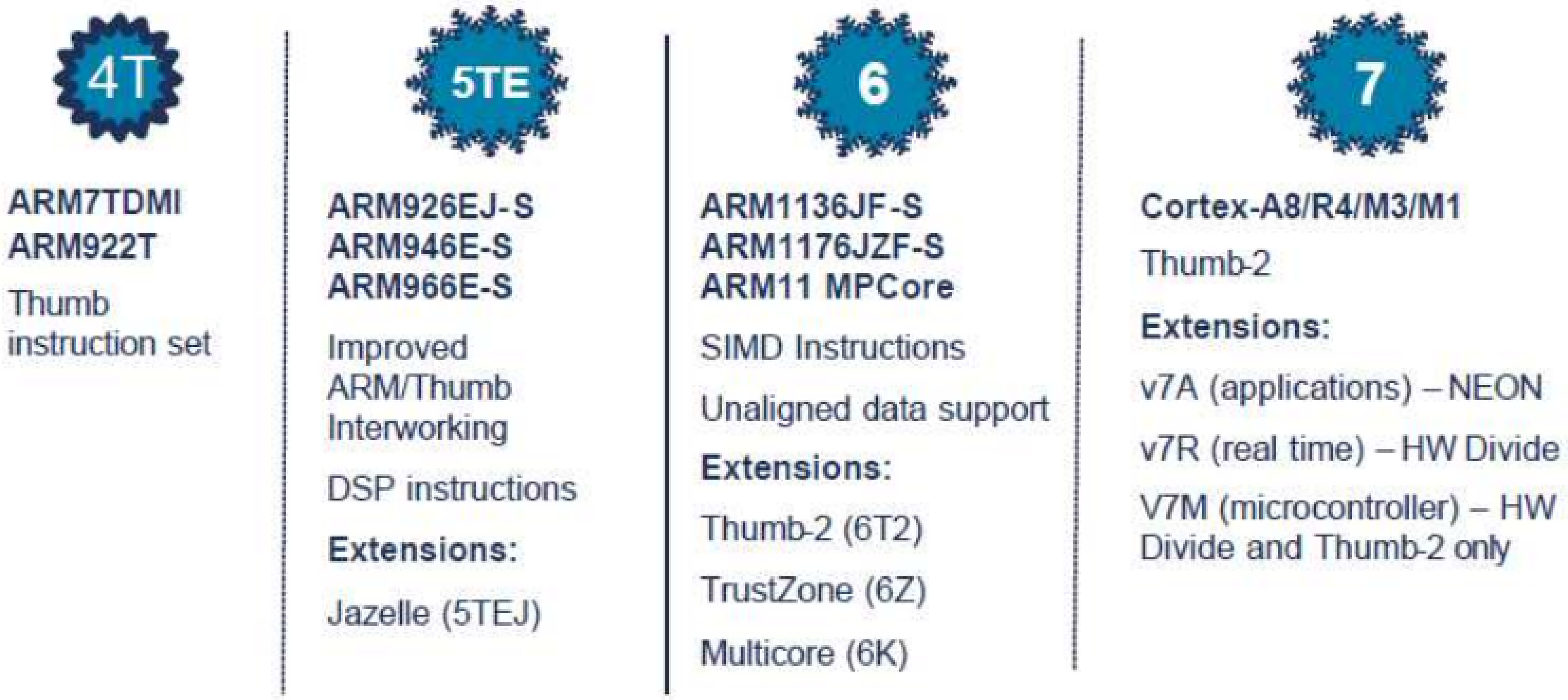
Code example

```
; Exception Vectors
;  Mapped to Address 0.
;  Absolute addressing mode must be used.
;  Dummy Handlers are implemented as infinite loops which can be modified.

Vectors     LDR     PC, Reset_Addr
            LDR     PC, Undef_Addr
            LDR     PC, SWI_Addr
            LDR     PC, PAbt_Addr
            LDR     PC, DAbt_Addr
            NOP                         ; Reserved Vector
            LDR     PC, IRQ_Addr
            LDR     PC, FIQ_Addr

Reset_Addr          DCD     Reset_Handler
Undef_Addr          DCD     Undef_Handler
SWI_Addr            DCD     SWI_Handler
PAbt_Addr           DCD     PAbt_Handler
DAbt_Addr           DCD     DAbt_Handler
                    DCD     0               ; Reserved Address
IRQ_Addr            DCD     IRQ_Handler
FIQ_Addr            DCD     FIQ_Handler
```

ARM architecture evolution



4T

ARM7TDMI
ARM922T

Thumb instruction set

5TE

ARM926EJ-S
ARM946E-S
ARM966E-S

Improved ARM/Thumb Interworking

DSP instructions

Extensions:

Jazelle (5TEJ)

6

ARM1136JF-S
ARM1176JZF-S
ARM11 MPCore

SIMD Instructions

Unaligned data support

Extensions:

Thumb-2 (6T2)

TrustZone (6Z)

Multicore (6K)

7

Cortex-A8/R4/M3/M1

Thumb-2

Extensions:

v7A (applications) – NEON

v7R (real time) – HW Divide

V7M (microcontroller) – HW Divide and Thumb-2 only

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

NXP - LPC2478

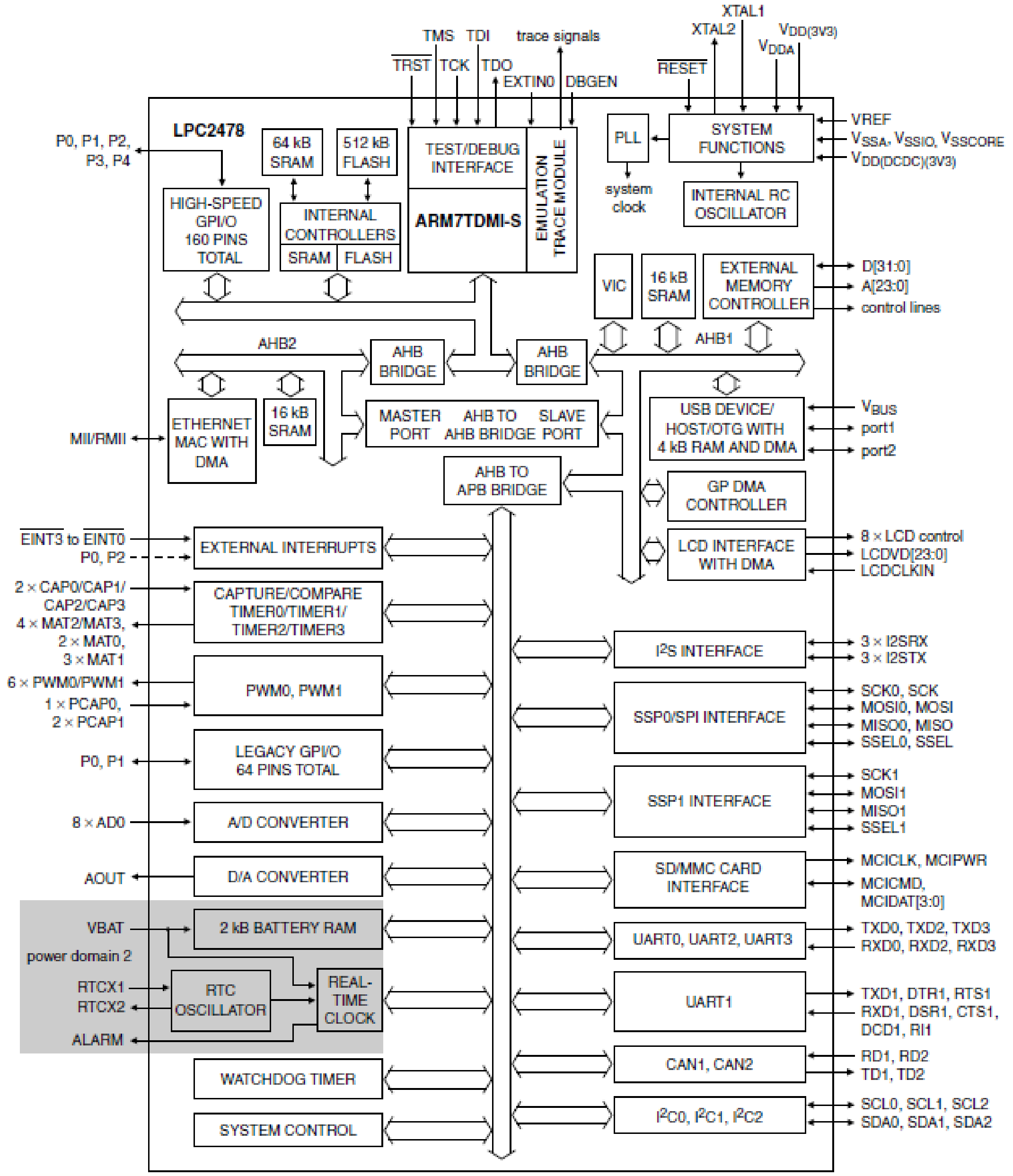
utbm

Nicolas Lacaille

Présentation

- Microcontroler from nxp with ARM7TDMI-S core
- Running up to 80MHz
- 64 kbyte of SRAM
- 518 kbyte of flash program memory
- External memory interface
  - An external memory controller is present to connect static or dynamic RAM or FLASH
- Peripherals
  - AHB peripherals (VIC, ethernet, usb, memory, FastGPIO)
  - APB peripherals (sérial, Timer, PWM, ADC, RT clock, ...)

Block diagram



Memory map

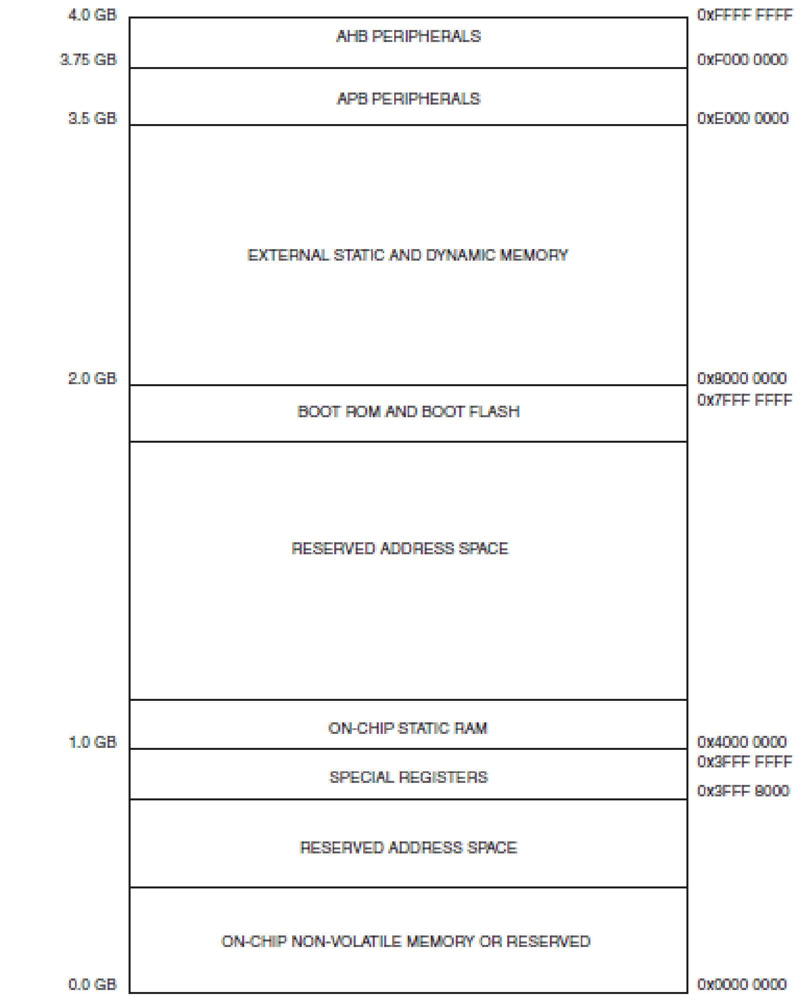


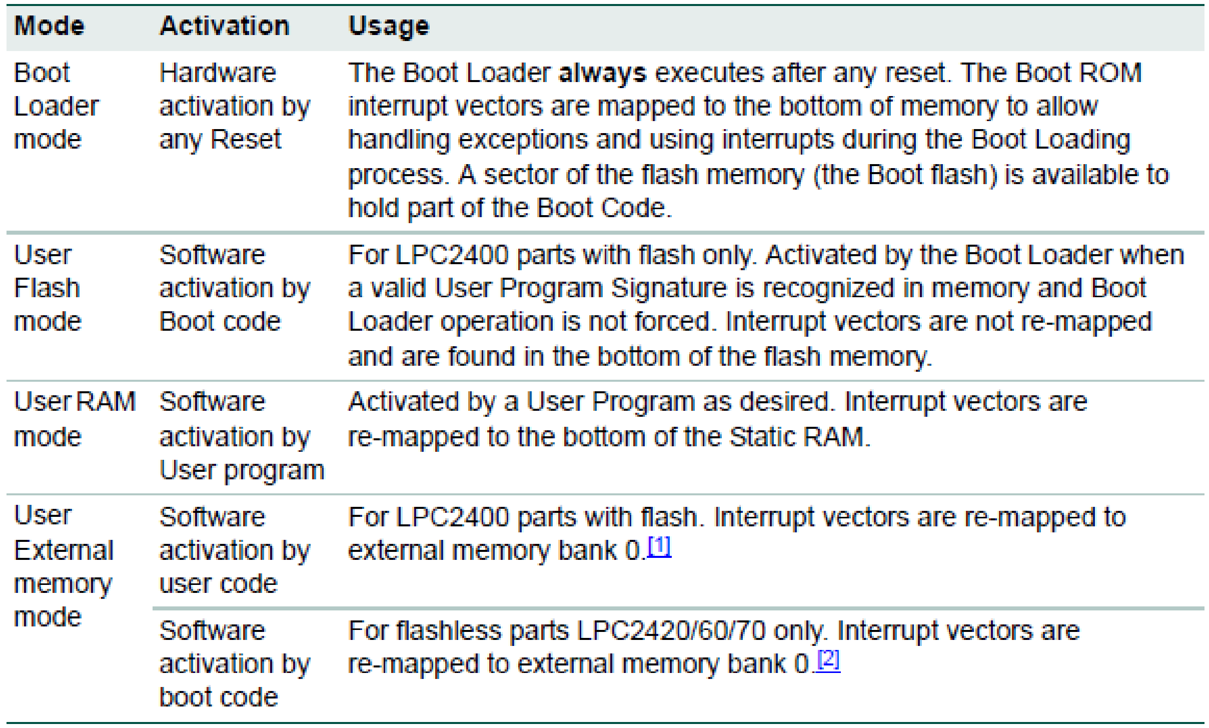
Fig 6. LPC2400 system memory map

Memory map

Address range	General use	Address range details and description	
0x0000 0000 to 0x3FFF FFFF	On-chip non-volatile memory and Fast I/O	0x0000 0000 - 0x0007 FFFF	Flash Memory (512 kB)
		0x3FFF C000 - 0x3FFF FFFF	Fast GPIO registers
0x4000 0000 to 0x7FFF FFFF	On-chip RAM	0x4000 0000 - 0x4000 FFFF	RAM (64 kB)
		0x7FE0 0000 - 0x7FE0 3FFF	Ethernet RAM (16 kB)
		0x7FD0 0000 - 0x7FD0 3FFF	USB RAM (16 kB)
0x8000 0000 to 0xDFFF FFFF	Off-Chip Memory	Four static memory banks, 16 MB each	
		0x8000 0000 - 0x80FF FFFF	Static memory bank 0
		0x8100 0000 - 0x81FF FFFF	Static memory bank 1
		0x8200 0000 - 0x82FF FFFF	Static memory bank 2
		0x8300 0000 - 0x83FF FFFF	Static memory bank 3
		Four dynamic memory banks, 256 MB each	
		0xA000 0000 - 0xAFFF FFFF	Dynamic memory bank 0
		0xB000 0000 - 0xBFFF FFFF	Dynamic memory bank 1
		0xC000 0000 - 0xCFFF FFFF	Dynamic memory bank 2
		0xD000 0000 - 0xDFFF FFFF	Dynamic memory bank 3
0xE000 0000 to 0xEFFF FFFF	APB Peripherals	36 peripheral blocks, 16 kB each	
0xF000 0000 to 0xFFFF FFFF	AHB peripherals		

Remapping

- ARM exception vectors are at address 0x0 → 0x1C
- Remapping on LPC2478 consists in changing some memory address to map vector address (64 byte from 0x0 to 0x3F)
- Modes :



Mode	Activation	Usage
Boot Loader mode	Hardware activation by any Reset	The Boot Loader **always** executes after any reset. The Boot ROM interrupt vectors are mapped to the bottom of memory to allow handling exceptions and using interrupts during the Boot Loading process. A sector of the flash memory (the Boot flash) is available to hold part of the Boot Code.
User Flash mode	Software activation by Boot code	For LPC2400 parts with flash only. Activated by the Boot Loader when a valid User Program Signature is recognized in memory and Boot Loader operation is not forced. Interrupt vectors are not re-mapped and are found in the bottom of the flash memory.
User RAM mode	Software activation by User program	Activated by a User Program as desired. Interrupt vectors are re-mapped to the bottom of the Static RAM.
User External memory mode	Software activation by user code	For LPC2400 parts with flash. Interrupt vectors are re-mapped to external memory bank 0.[1]
	Software activation by boot code	For flashless parts LPC2420/60/70 only. Interrupt vectors are re-mapped to external memory bank 0.[2]

MEMMAP Register

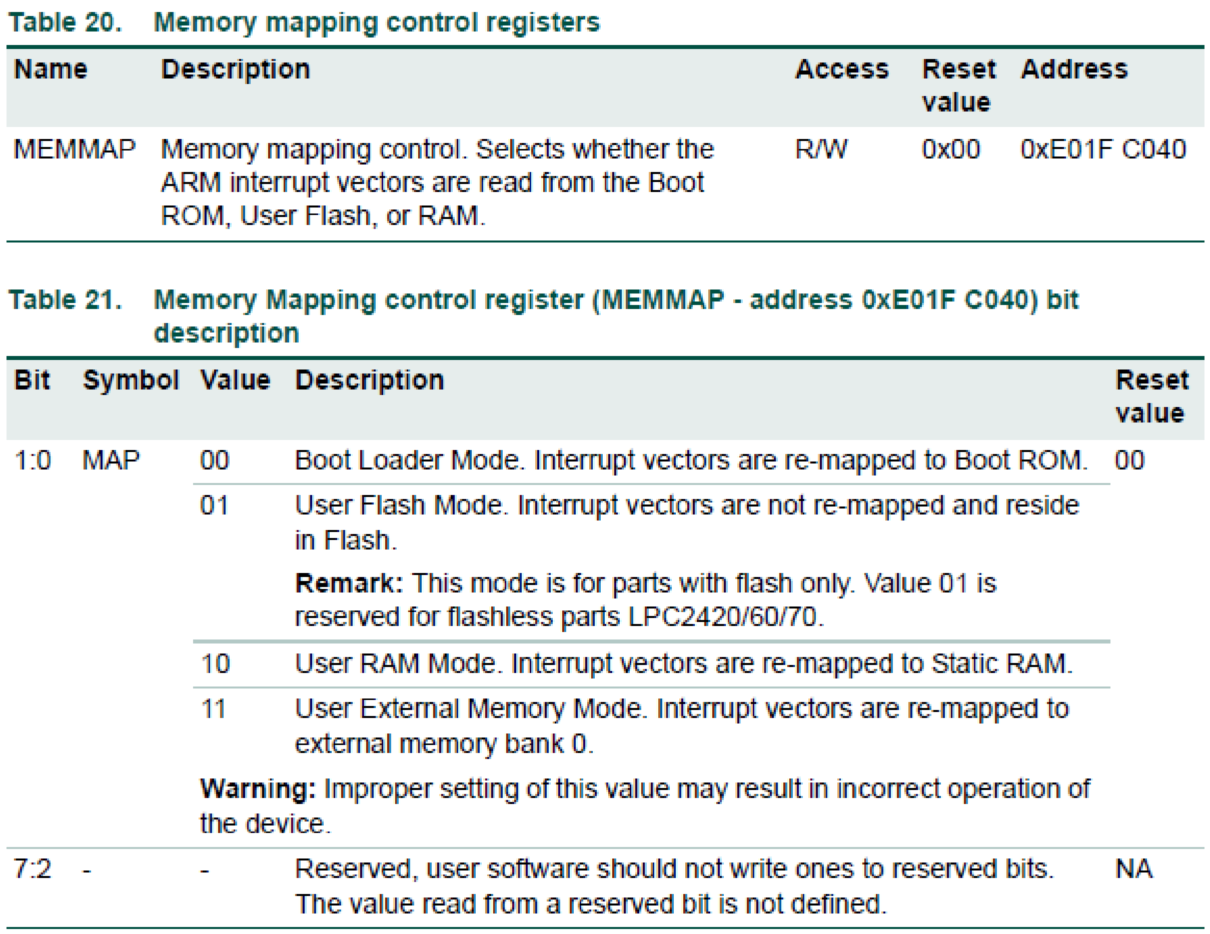


Table 20. Memory mapping control registers

Name	Description	Access	Reset value	Address
MEMMAP	Memory mapping control. Selects whether the ARM interrupt vectors are read from the Boot ROM, User Flash, or RAM.	R/W	0x00	0xE01F C040

Table 21. Memory Mapping control register (MEMMAP - address 0xE01F C040) bit description

Bit	Symbol	Value	Description	Reset value
1:0	MAP	00	Boot Loader Mode. Interrupt vectors are re-mapped to Boot ROM.	00
		01	User Flash Mode. Interrupt vectors are not re-mapped and reside in Flash. **Remark:** This mode is for parts with flash only. Value 01 is reserved for flashless parts LPC2420/60/70.	
		10	User RAM Mode. Interrupt vectors are re-mapped to Static RAM.	
		11	User External Memory Mode. Interrupt vectors are re-mapped to external memory bank 0.	
		Warning: Improper setting of this value may result in incorrect operation of the device.		
7:2	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Flash bootloader

- Provide initial operation after reset and means to programs user flash memory
- At reset, with certain conditions an ISP handler is invoked (In System Programming)
  - P2.10 sampled low
  - Watchdog flag not set
- If P2.10 is sampled High, the boot loader search for a valid user program in flash
  - A checksum of exception vector is done (signature in 0x14 added with the sum of other exception vectors must be 0)
  - If checksum is valid the user program is launch otherwise no

Clock

- 3 oscillators
  - Main oscillator : 1 to 24 MHz (12MHz)
  - Internal RC oscillator (4MHz)
  - RTC oscillator
- All oscillator can drive a PLL and subsequently the CPU
- PLL allow to choose the CPU clock frequency from the clock source

$$f_{pll} = \frac{2 * M}{N} * f_{sected}$$

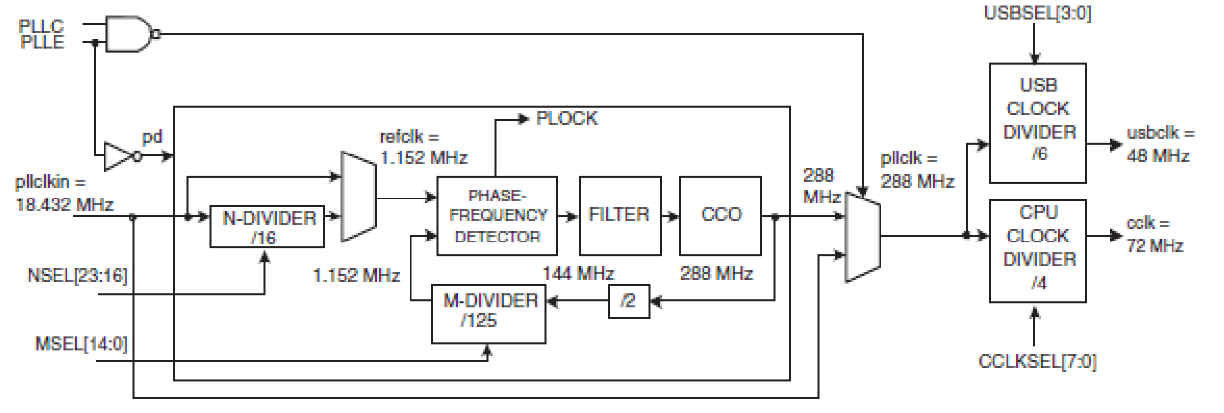


Fig 14. PLL block diagram (N = 16, M = 125, USBSEL = 6, CCLKSEL = 4)

Clock

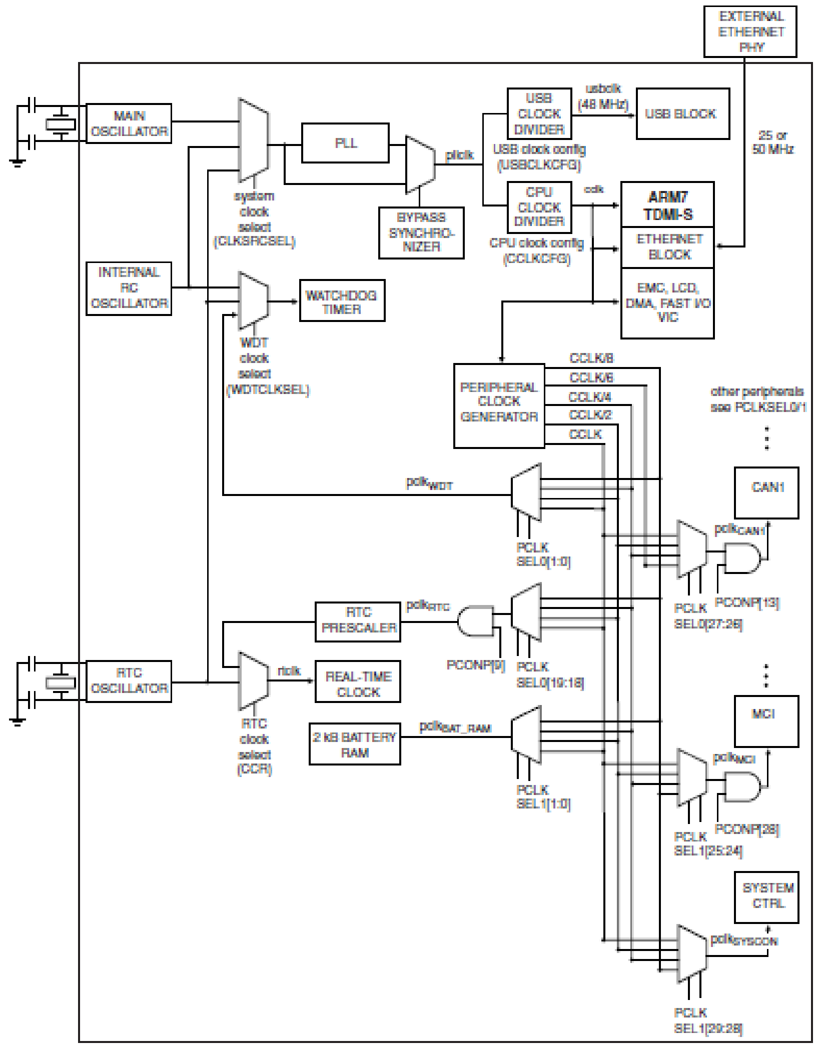


Fig 12. Clock generation for the LPC2400

Selecting clock

- At startup, the internal RC oscillator is used and PLL is bypassed
- User boot can activate the main oscillator (SCS : system Control ans Status Register)
- When main oscillator is stabilized user program can use it as clock source for the PLL (CLKSRCSEL register) and activate it with specific value (PLLCFG to choose M and N)
- CPU (CCLKCFG) and USB (USBCLKCFG) divider are set
- Peripheral clocks are set (PCLKSEL0 and 1)

Example : ConfigurePLL() (framework.c)

```
void ConfigurePLL ( void ){
      unsigned int MValue, NValue;

      if ( PLLSTAT & (1 << 25) ){
            PLLCON = 1;                         /* Enable PLL, disconnected */
            PLLFEED = 0xaa;
            PLLFEED = 0x55;
   }
   PLLCON = 0;                                  /* Disable PLL, disconnected */
   PLLFEED = 0xaa;
   PLLFEED = 0x55;

   SCS |= 0x20;                                 /* Enable main OSC */
   while( !(SCS & 0x40) );                      /* Wait until main OSC is usable */

   CLKSRCSEL = 0x1;                             /* select main OSC, 12MHz, as the PLL clock source */

   PLLCFG = PLL_MValue | (PLL_NValue << 16);
   PLLFEED = 0xaa;
   PLLFEED = 0x55;

   PLLCON = 1;                                  /* Enable PLL, disconnected */
   PLLFEED = 0xaa;
   PLLFEED = 0x55;

   CCLKCFG = CCLKDivValue;                      /* Set clock divider */
#if USE_USB
   USBCLKCFG = USBCLKDivValue;       /* usbclk = 288 MHz/6 = 48 MHz */
#endif

   while ( ((PLLSTAT & (1 << 26)) == 0) );     /* Check lock bit status */

   MValue = PLLSTAT & 0x00007FFF;
   NValue = (PLLSTAT & 0x00FF0000) >> 16;
   while ((MValue != PLL_MValue) && ( NValue != PLL_NValue) );

   PLLCON = 3;                                  /* enable and connect */
   PLLFEED = 0xaa;
   PLLFEED = 0x55;
  while ( ((PLLSTAT & (1 << 25)) == 0) );      /* Check connect bit status */
}
```

Peripheral clocks

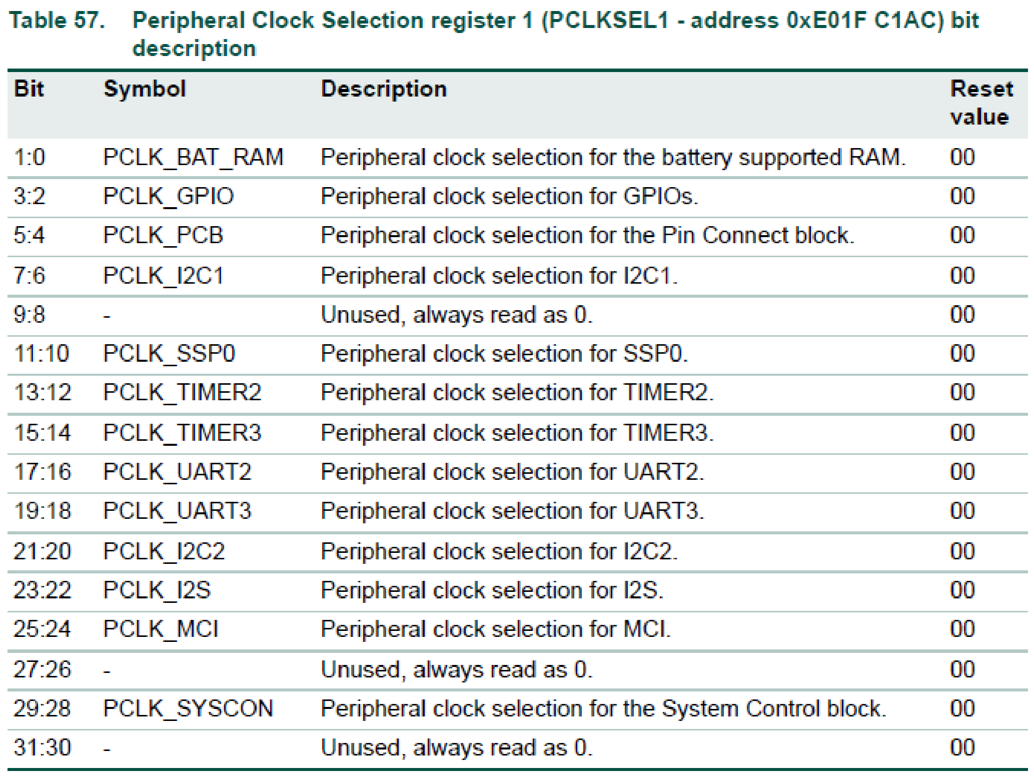


Table 57. Peripheral Clock Selection register 1 (PCLKSEL1 - address 0xE01F C1AC) bit description

Bit	Symbol	Description	Reset value
1:0	PCLK_BAT_RAM	Peripheral clock selection for the battery supported RAM.	00
3:2	PCLK_GPIO	Peripheral clock selection for GPIOs.	00
5:4	PCLK_PCB	Peripheral clock selection for the Pin Connect block.	00
7:6	PCLK_I2C1	Peripheral clock selection for I2C1.	00
9:8	-	Unused, always read as 0.	00
11:10	PCLK_SSP0	Peripheral clock selection for SSP0.	00
13:12	PCLK_TIMER2	Peripheral clock selection for TIMER2.	00
15:14	PCLK_TIMER3	Peripheral clock selection for TIMER3.	00
17:16	PCLK_UART2	Peripheral clock selection for UART2.	00
19:18	PCLK_UART3	Peripheral clock selection for UART3.	00
21:20	PCLK_I2C2	Peripheral clock selection for I2C2.	00
23:22	PCLK_I2S	Peripheral clock selection for I2S.	00
25:24	PCLK_MCI	Peripheral clock selection for MCI.	00
27:26	-	Unused, always read as 0.	00
29:28	PCLK_SYSCON	Peripheral clock selection for the System Control block.	00
31:30	-	Unused, always read as 0.	00

PCLKSEL0
PCLKSEL1

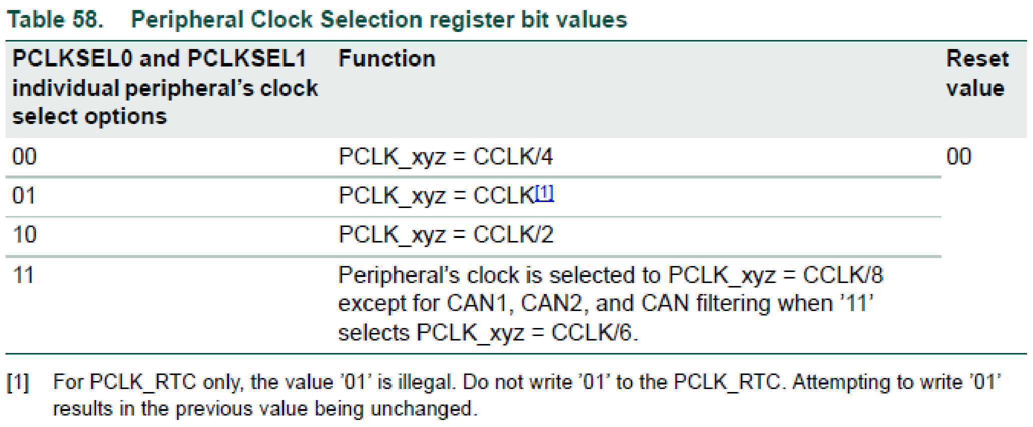


Table 58. Peripheral Clock Selection register bit values

PCLKSEL0 and PCLKSEL1 individual peripheral's clock select options	Function	Reset value
00	PCLK_xyz = CCLK/4	00
01	PCLK_xyz = CCLK[1]	
10	PCLK_xyz = CCLK/2	
11	Peripheral's clock is selected to PCLK_xyz = CCLK/8 except for CAN1, CAN2, and CAN filtering when '11' selects PCLK_xyz = CCLK/6.	

[1] For PCLK_RTC only, the value '01' is illegal. Do not write '01' to the PCLK_RTC. Attempting to write '01' results in the previous value being unchanged.

Power

- 4 special modes of power reduction :
  - Idle
    - Clocks core stopped
    - Resume on reset or interrupt
  - Sleep
    - Main oscillator powered down and all clock stopped
    - Wake up on reset or interrupt
    - PLL must be reconfigured
  - Power-down
    - All clock powered down
    - Flash is powered down (unlike sleep)
  - Deep power-down
    - Power regulator turned off (register values are not retained)

Peripheral power control

- Each peripheral can be turned off (clock disable)
- Control of power peripheral done through PCONP reg.

Bit	Symbol	Description	Reset value
0	-	Unused, always 0	0
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	PCPWM0	PWM0 power/clock control bit.	1
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I^{2}C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1
10	PCSSP1	The SSP1 interface power/clock control bit.	1
11	PCEMC	External Memory Controller	1
12	PCAD	A/D converter (ADC) power/clock control bit. **Note:** Clear the PDN bit in the AD0CR before clearing this bit, and set this bit before setting PDN.	0
13	PCCAN1	CAN Controller 1 power/clock control bit.	0
14	PCCAN2	CAN Controller 2 power/clock control bit.	0
18:15	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	
19	PCI2C1	The I^{2}C1 interface power/clock control bit.	1
20	PCLCD[1]	LCD controller power control bit.	0
21	PCSSP0	The SSP0 interface power/clock control bit.	1
22	PCTIM2	Timer 2 power/clock control bit.	0
23	PCTIM3	Timer 3 power/clock control bit.	0
24	PCUART2	UART 2 power/clock control bit.	0
25	PCUART3	UART 3 power/clock control bit.	0
26	PCI2C2	I^{2}S interface 2 power/clock control bit.	1
27	PCI2S	I^{2}S interface power/clock control bit.	0
28	PCSDC	SD card interface power/clock control bit.	0
29	PCGPDMA	GP DMA function power/clock control bit.	0
30	PCENET	Ethernet block power/clock control bit.	0
31	PCUSB	USB interface power/clock control bit.	0

1 : enable
0 : disable

If peripheral is disable, read or write register are not valid

Peripheral power control

- Each peripheral can be turned off (clock disable)
- Control of power peripheral done through PCONP reg.

Bit	Symbol	Description	Reset value
0	-	Unused, always 0	0
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	PCPWM0	PWM0 power/clock control bit.	1
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I^2C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1
10	PCSSP1	The SSP1 interface power/clock control bit.	1
11	PCEMC	External Memory Controller	1
12	PCAD	A/D converter (ADC) power/clock control bit. **Note:** Clear the PDN bit in the AD0CR before clearing this bit, and set this bit before setting PDN.	0
13	PCCAN1	CAN Controller 1 power/clock control bit.	0
14	PCCAN2	CAN Controller 2 power/clock control bit.	0
18:15	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	
19	PCI2C1	The I^2C1 interface power/clock control bit.	1
20	PCLCD[1]	LCD controller power control bit.	0
21	PCSSP0	The SSP0 interface power/clock control bit.	1
22	PCTIM2	Timer 2 power/clock control bit.	0
23	PCTIM3	Timer 3 power/clock control bit.	0
24	PCUART2	UART 2 power/clock control bit.	0
25	PCUART3	UART 3 power/clock control bit.	0
26	PCI2C2	I^2S interface 2 power/clock control bit.	1
27	PCI2S	I^2S interface power/clock control bit.	0
28	PCSDC	SD card interface power/clock control bit.	0
29	PCGPDMA	GP DMA function power/clock control bit.	0
30	PCENET	Ethernet block power/clock control bit.	0
31	PCUSB	USB interface power/clock control bit.	0

1 : enable
0 : disable

If peripheral is disable, read or write register are not valid

External Memory Controller

- Dynamic memory interface support including Single Data Rate SDRAM.
- Asynchronous static memory device support including RAM, ROM, and Flash, with or without asynchronous page mode.
- Low transaction latency.
- Read and write buffers to reduce latency and to improve performance.
- 8 bit, 16 bit, and 32 bit wide static memory support.
- 16 bit and 32 bit wide chip select SDRAM memory support.
- Static memory features include:
  - Asynchronous page mode read
  - Programmable wait states
  - Bus turnaround delay
  - Output enable and write enable delays
  - Extended wait
- Four chip selects for synchronous memory and four chip selects for static memory devices.
- Power-saving modes dynamically control CKE and CLKOUT to SDRAMs.
- Dynamic memory self-refresh mode controlled by software.
- Controller supports 2 kbit, 4 kbit, and 8 kbit row address synchronous memory parts. That is typical 512 MB, 256 MB, and 128 MB parts, with 4, 8, 16, or 32 data bits per device.
- Separate reset domains allow the for auto-refresh through a chip reset if desired.

Note: Synchronous static memory devices (synchronous burst mode) are not supported.

EMC

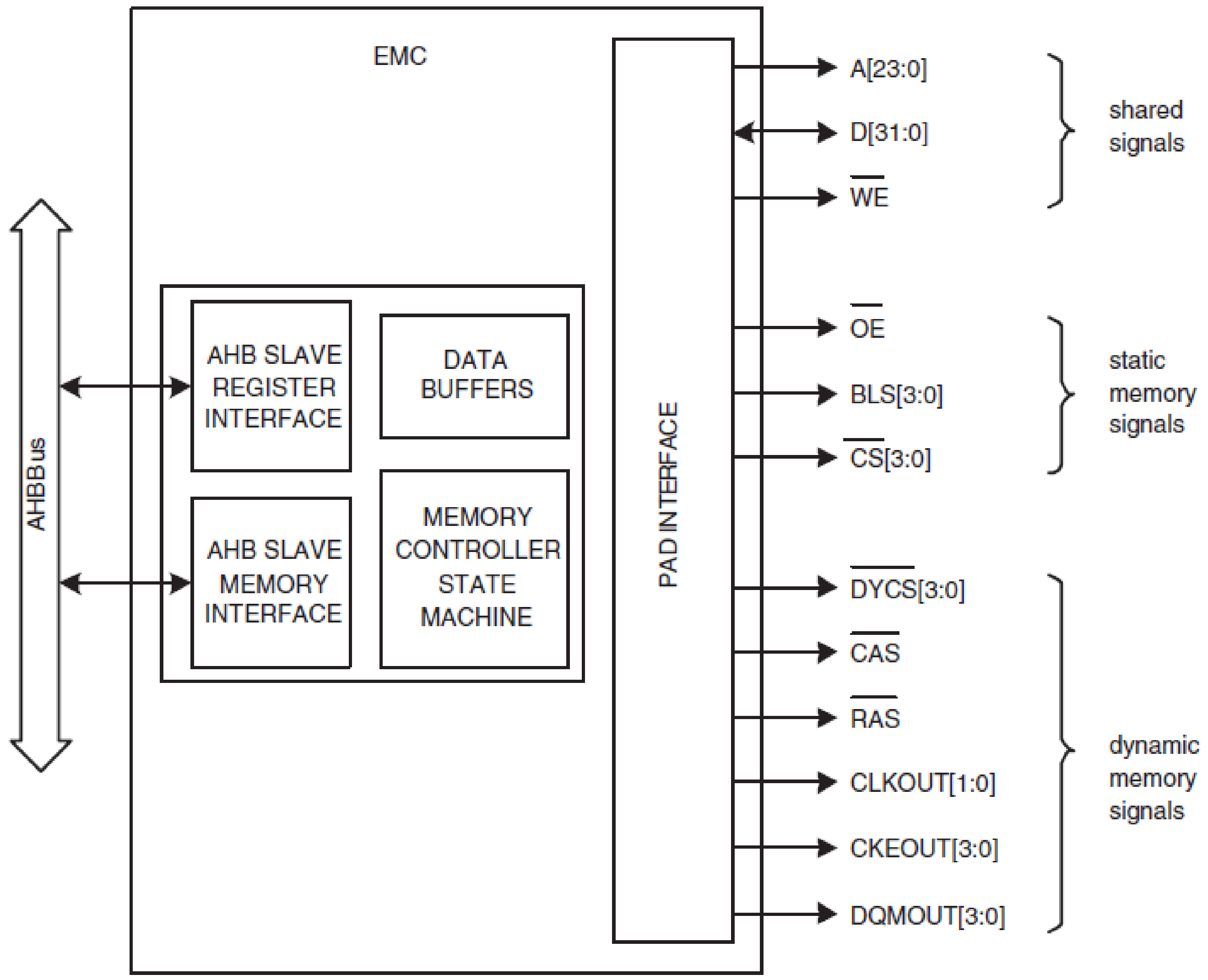


Fig 15. EMC block diagram

LPC2478 board : external memory

- External NOR FLASH (32 MBit = 4 MByte in size) addressed by CS0 (address range: 0x8000 0000 – 0x80FF FFFF). Accessed via 16-bit databus.
- External NAND FLASH (1 GBit = 128 MByte in size) addressed by CS1 (address range: 0x8100 0000 – 0x81FF FFFF). Accessed via 8-bit databus.
- External SDRAM (256 MBit = 32 MByte in size) addressed by DYCS0 (address range: 0xA000 0000 – 0xA1FF FFFF). Accessed via 32-bit databus

Memory Accelerator Module

- Small SRAM memory between flash and core
- Allow fast instruction access
  - Direct access to flash is limited to 20MHz (50ns access time)
- Load 4 arm instructions from flash
  - 2 buffers are alternatively used to maintain prefetch rate
- Include a branch trail buffer for loops

```
   /* Set memory accelerater module*/
   MAMCR = 0;
#if Fcclk < 20000000
   MAMTIM = 1;
#else
#if Fcclk < 40000000
   MAMTIM = 2;
#else
   MAMTIM = 3;
#endif
#endif
   MAMCR  = MAM_SETTING;     //0=disabled, 1=partly enabled (enabled for code prefetch, but not for data), 2=fully enabled
```

PIN

- To reduce number of pins on chip, pins are multiplexed
  - Different functions can use the pins
- Registers which are controlling pin function are
  - PINSEL (PINSEL0 to PINSEL11)
  - PINMODE (PINMODE0 to PINMODE9)
-

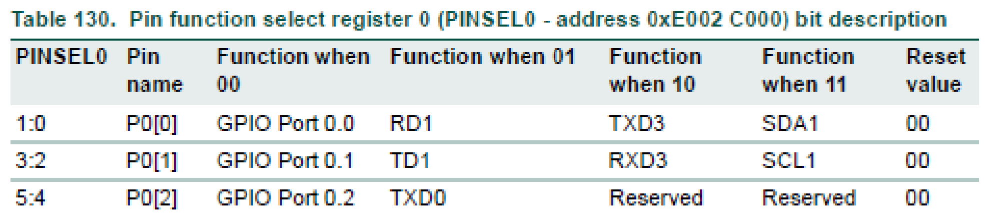


Table 130. Pin function select register 0 (PINSEL0 - address 0xE002 C000) bit description

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0[0]	GPIO Port 0.0	RD1	TXD3	SDA1	00
3:2	P0[1]	GPIO Port 0.1	TD1	RXD3	SCL1	00
5:4	P0[2]	GPIO Port 0.2	TXD0	Reserved	Reserved	00

PINSEL controls pin multiplexer

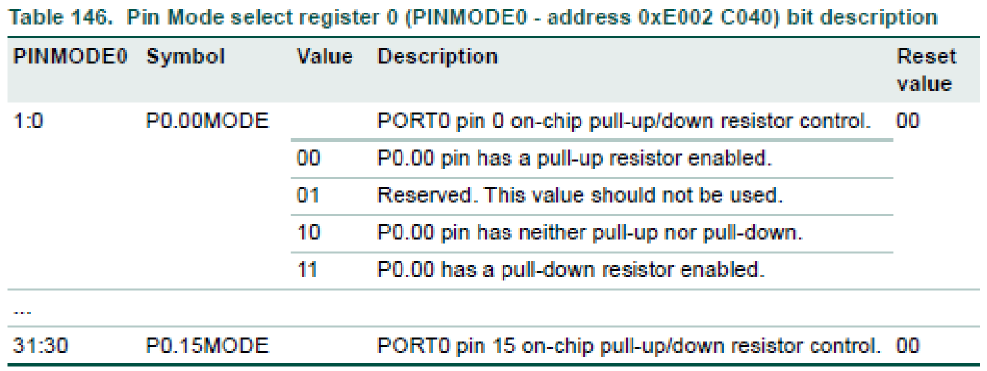


Table 146. Pin Mode select register 0 (PINMODE0 - address 0xE002 C040) bit description

PINMODE0	Symbol	Value	Description	Reset value
1:0	P0.00MODE		PORT0 pin 0 on-chip pull-up/down resistor control.	00
		00	P0.00 pin has a pull-up resistor enabled.	
		01	Reserved. This value should not be used.	
		10	P0.00 pin has neither pull-up nor pull-down.	
		11	P0.00 has a pull-down resistor enabled.	
...				
31:30	P0.15MODE		PORT0 pin 15 on-chip pull-up/down resistor control.	00

PINMODE define electrical pin connexion

Exemple

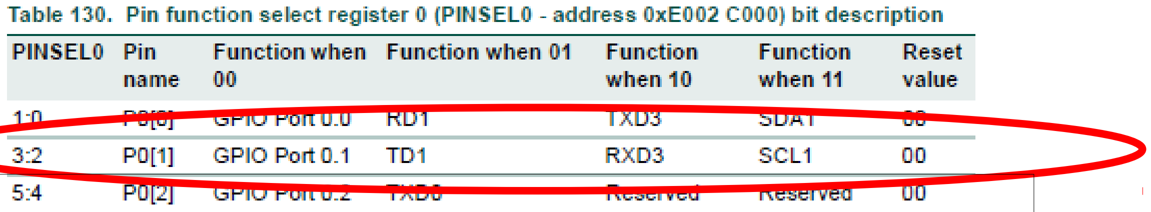
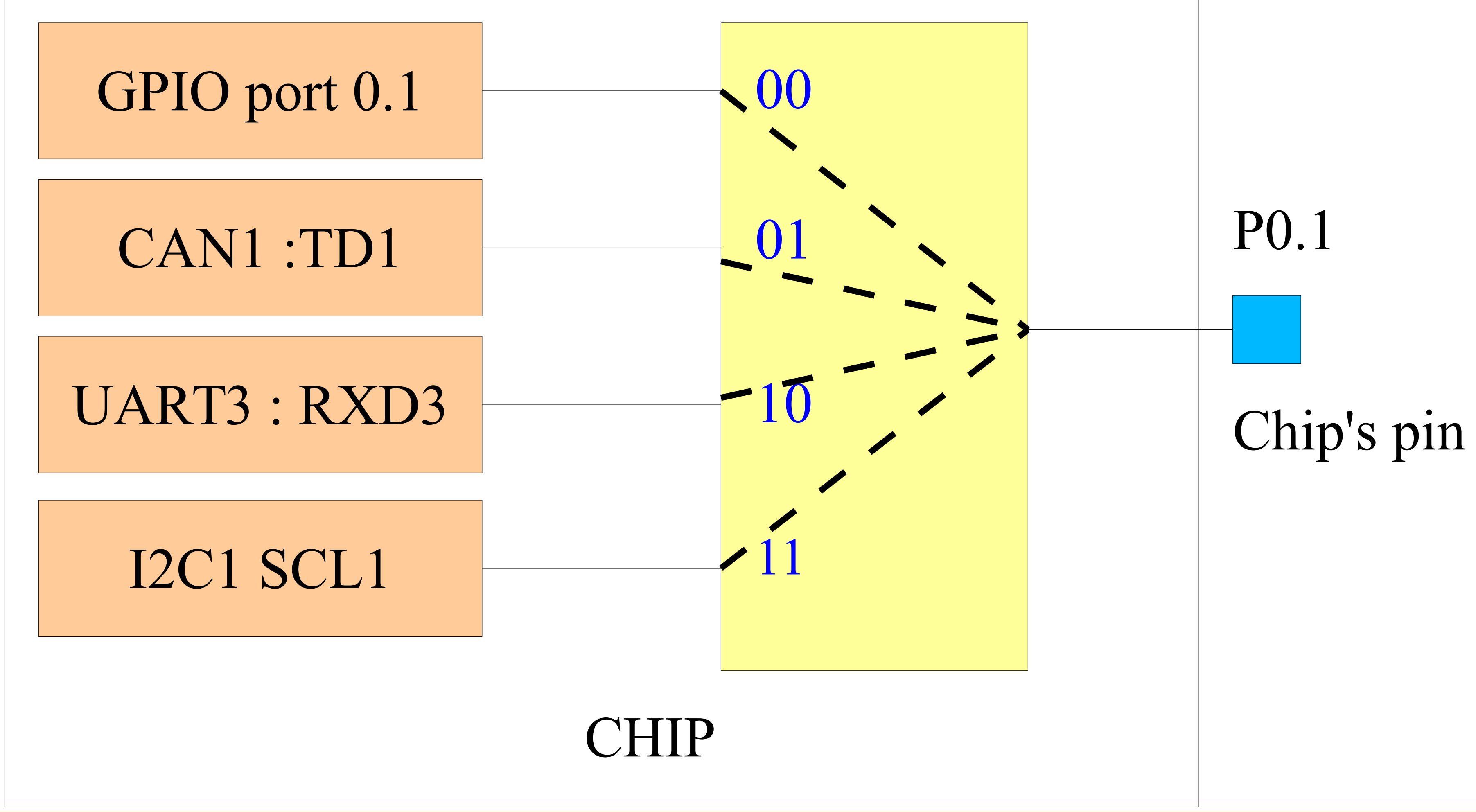


Table 130. Pin function select register 0 (PINSEL0 - address 0xE002 C000) bit description

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0[0]	GPIO Port 0.0	RD1	TXD3	SDA1	00
3:2	P0[1]	GPIO Port 0.1	TD1	RXD3	SCL1	00
5:4	P0[2]	GPIO Port 0.2	TXD0	Reserved	Reserved	00



GPIO : General Purpose I/O

- 5 general purpose 32 bits port
- GPIO controller are located on the local bus for fast controlling
- Port0 and Port1 can also be controlled by legacy control register on APB bus (slow)
- Port0 and Port2 can generate interrupts on individual change of individual pin
- Each individual pin can be configured as input or out put (FIOxDIR)
- Each individual pin can be masked (FIOxMASK) for reading and writing (read 0 and no effects on write)

Registers

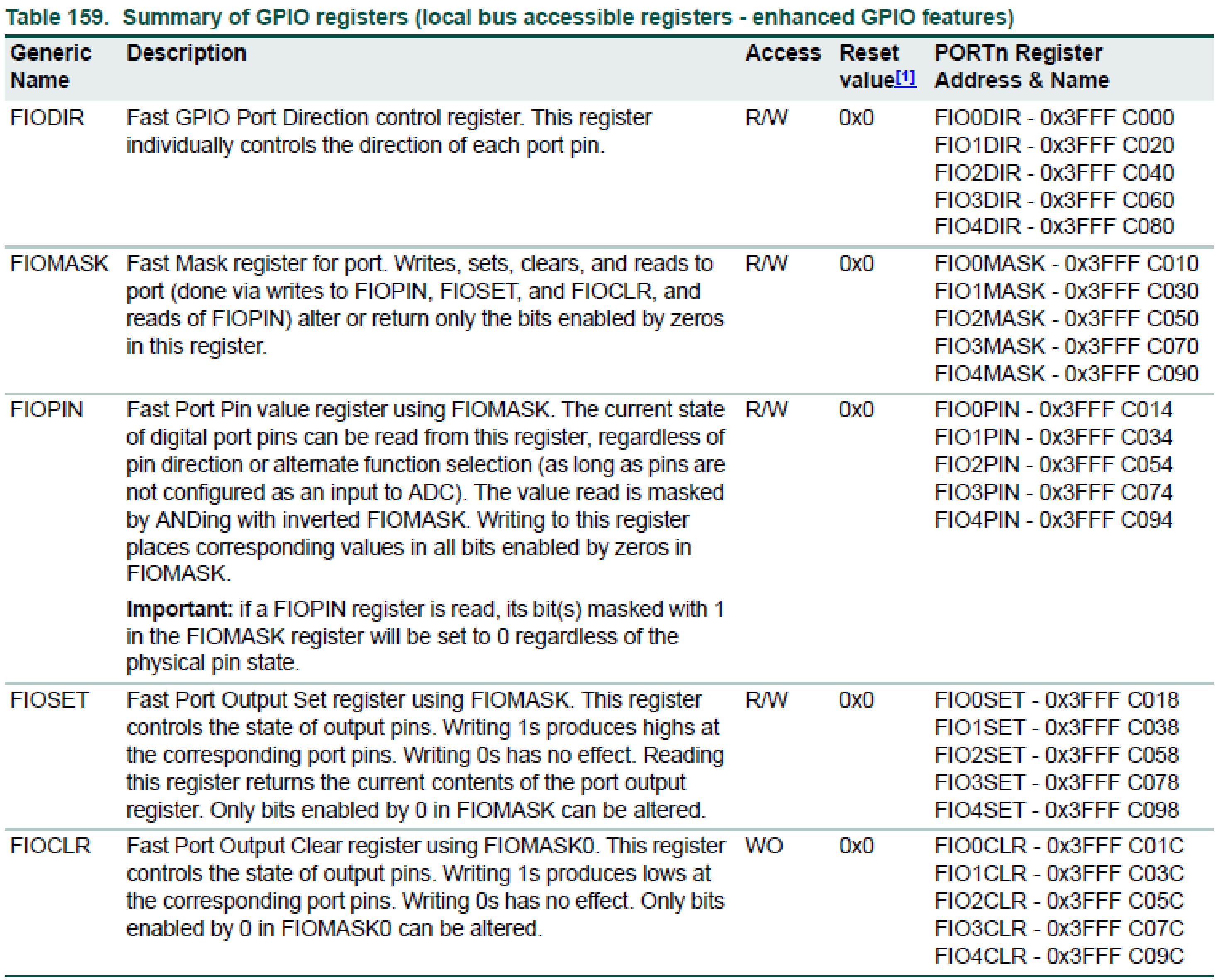


Table 159. Summary of GPIO registers (local bus accessible registers - enhanced GPIO features)

Generic Name	Description	Access	Reset value[1]	PORTn Register Address & Name
FIODIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.	R/W	0x0	FIO0DIR - 0x3FFF C000 FIO1DIR - 0x3FFF C020 FIO2DIR - 0x3FFF C040 FIO3DIR - 0x3FFF C060 FIO4DIR - 0x3FFF C080
FIOMASK	Fast Mask register for port. Writes, sets, clears, and reads to port (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN) alter or return only the bits enabled by zeros in this register.	R/W	0x0	FIO0MASK - 0x3FFF C010 FIO1MASK - 0x3FFF C030 FIO2MASK - 0x3FFF C050 FIO3MASK - 0x3FFF C070 FIO4MASK - 0x3FFF C090
FIOPIN	Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC). The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK. **Important:** if a FIOPIN register is read, its bit(s) masked with 1 in the FIOMASK register will be set to 0 regardless of the physical pin state.	R/W	0x0	FIO0PIN - 0x3FFF C014 FIO1PIN - 0x3FFF C034 FIO2PIN - 0x3FFF C054 FIO3PIN - 0x3FFF C074 FIO4PIN - 0x3FFF C094
FIOSET	Fast Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.	R/W	0x0	FIO0SET - 0x3FFF C018 FIO1SET - 0x3FFF C038 FIO2SET - 0x3FFF C058 FIO3SET - 0x3FFF C078 FIO4SET - 0x3FFF C098
FIOCLR	Fast Port Output Clear register using FIOMASK0. This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIOMASK0 can be altered.	WO	0x0	FIO0CLR - 0x3FFF C01C FIO1CLR - 0x3FFF C03C FIO2CLR - 0x3FFF C05C FIO3CLR - 0x3FFF C07C FIO4CLR - 0x3FFF C09C

Writing on a pin

- To configure pin as output a 1 must be written on the corresponding pin in FIOxDIR (0 is for input)
- To set or clear a pin Two register can be used
  - FIOxSET : set the pin by writing a 1 on the corresponding bit
  - FIOxCLEAR : clear the bin by writing a 1 on the corresponding bit
- Writing a value in FIOxPIN can also be used
- Corresponding bit in FIOxMASK must be 0
- Ex

```
FIO0DIR = 0x2 ; // set direction for bit 1
FIO0SET = 0x2; // set P0.1
FIO0CLEAR = 0x2; // clear P0.1
```

```
FIO0DIR = 0x2 ;
FIO0PIN |= 0x2; // set P0.1
FIO0PIN &= ~0x2; // clear P0.1
```

Interrupt with GPIO

- Port0 and 2 can be configured to generate interrupt
- 2 pairs of enable/status registers are present : one for a rising edge and one for falling edge
  - InEnF/R : enable corresponding pin for interrupt
  - IntStatF/R (RO) : to verify which pin has generate interrupt
- Interrupt must be cleared through IntClr register

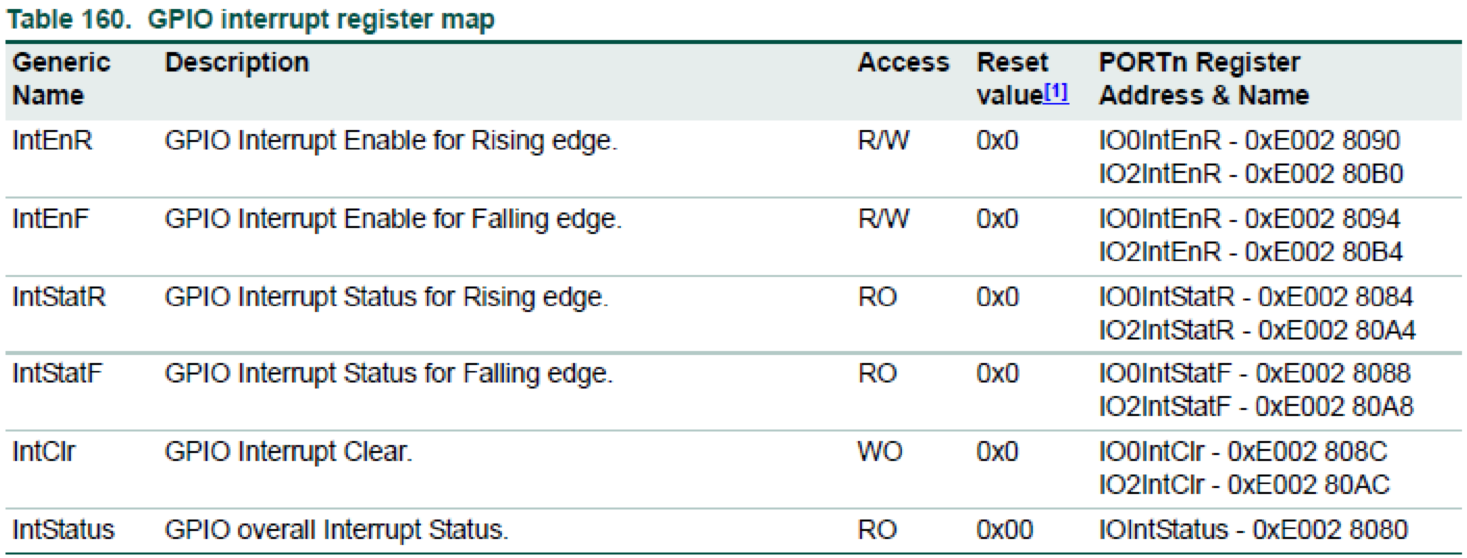


Table 160. GPIO interrupt register map

Generic Name	Description	Access	Reset value[1]	PORTn Register Address & Name
IntEnR	GPIO Interrupt Enable for Rising edge.	R/W	0x0	IO0IntEnR - 0xE002 8090 IO2IntEnR - 0xE002 80B0
IntEnF	GPIO Interrupt Enable for Falling edge.	R/W	0x0	IO0IntEnR - 0xE002 8094 IO2IntEnR - 0xE002 80B4
IntStatR	GPIO Interrupt Status for Rising edge.	RO	0x0	IO0IntStatR - 0xE002 8084 IO2IntStatR - 0xE002 80A4
IntStatF	GPIO Interrupt Status for Falling edge.	RO	0x0	IO0IntStatF - 0xE002 8088 IO2IntStatF - 0xE002 80A8
IntClr	GPIO Interrupt Clear.	WO	0x0	IO0IntClr - 0xE002 808C IO2IntClr - 0xE002 80AC
IntStatus	GPIO overall Interrupt Status.	RO	0x00	IOIntStatus - 0xE002 8080

Exemple

```
void led210_init(void){
  // Power control
  //GPIO cannot be turned off
  // CLOCK
  PCLKSEL1 &= ~(0x3 << 2); //3:2 = 0b00 (CCLK / 4)
  // PIN :
  // function select for P2.10 (GPIO) in PINSEL4 (PINSEL4[21..20] = 0b00) (RW)
  PINSEL4 &= ~(3 << 20) ;
  // connect mode selection for pin (00 = pull up resistor selected) (RW)
  PINMODE4 &= ~(3 << 20) ;
  //PIO
  // direction mode selection : output = 1 et input = 0 (out selected) (R/W)
  FIO2DIR |= (1 <<10);
  // to allowed read an write on the selected pin (0 = enable)
  FIO2MASK &= ~(1 <<10);
}

void led210_turn_on(void){
  FIO2CLR = 1<<10;
}

void led210_turn_off(void){
  FIO2SET = 1<< 10;
}
```

Clock for pio :chap4 p59 & 60

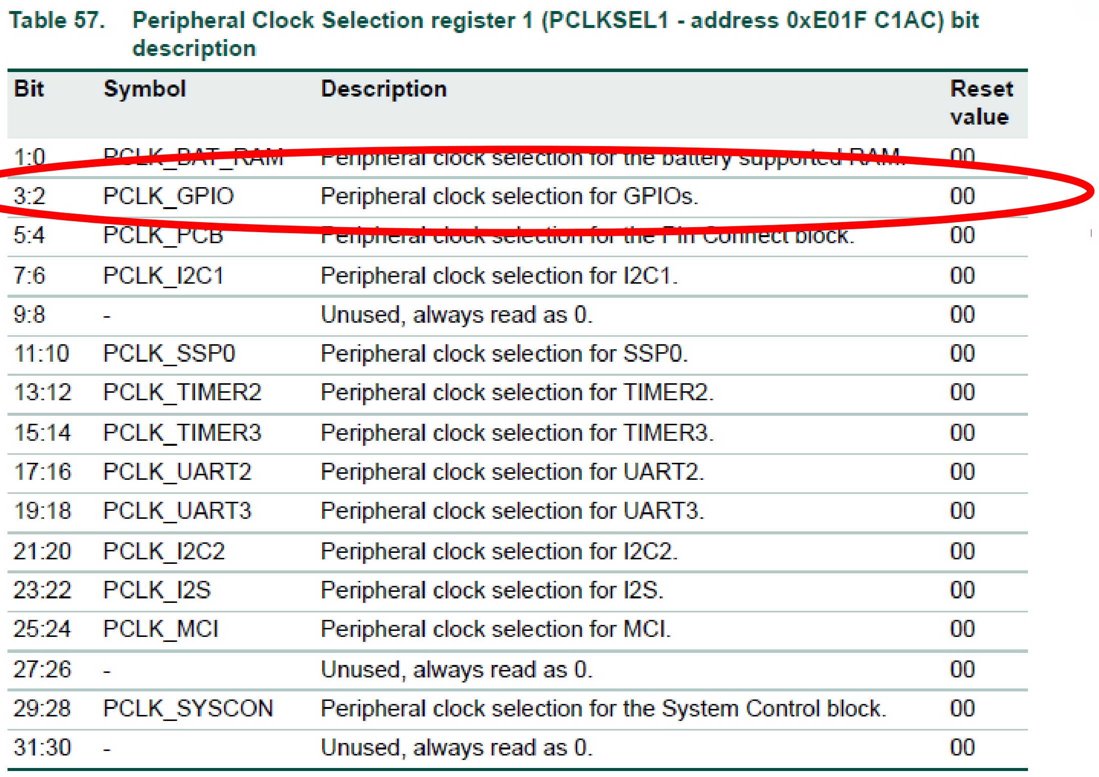


Table 57. Peripheral Clock Selection register 1 (PCLKSEL1 - address 0xE01F C1AC) bit description

Bit	Symbol	Description	Reset value
1:0	PCLK_BAT_RAM	Peripheral clock selection for the battery supported RAM.	00
3:2	PCLK_GPIO	Peripheral clock selection for GPIOs.	00
5:4	PCLK_PCB	Peripheral clock selection for the Pin Connect block.	00
7:6	PCLK_I2C1	Peripheral clock selection for I2C1.	00
9:8	-	Unused, always read as 0.	00
11:10	PCLK_SSP0	Peripheral clock selection for SSP0.	00
13:12	PCLK_TIMER2	Peripheral clock selection for TIMER2.	00
15:14	PCLK_TIMER3	Peripheral clock selection for TIMER3.	00
17:16	PCLK_UART2	Peripheral clock selection for UART2.	00
19:18	PCLK_UART3	Peripheral clock selection for UART3.	00
21:20	PCLK_I2C2	Peripheral clock selection for I2C2.	00
23:22	PCLK_I2S	Peripheral clock selection for I2S.	00
25:24	PCLK_MCI	Peripheral clock selection for MCI.	00
27:26	-	Unused, always read as 0.	00
29:28	PCLK_SYSCON	Peripheral clock selection for the System Control block.	00
31:30	-	Unused, always read as 0.	00

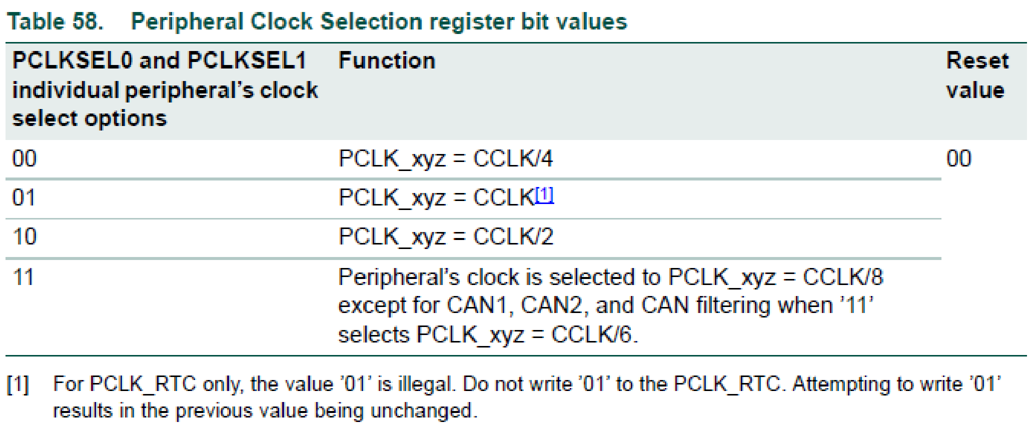


Table 58. Peripheral Clock Selection register bit values

PCLKSEL0 and PCLKSEL1 individual peripheral's clock select options	Function	Reset value
00	PCLK_xyz = CCLK/4	00
01	PCLK_xyz = CCLK[1]	
10	PCLK_xyz = CCLK/2	
11	Peripheral's clock is selected to PCLK_xyz = CCLK/8 except for CAN1, CAN2, and CAN filtering when '11' selects PCLK_xyz = CCLK/6.	

[1] For PCLK_RTC only, the value '01' is illegal. Do not write '01' to the PCLK_RTC. Attempting to write '01' results in the previous value being unchanged.

```
PCLKSEL1 &= ~(0x3 << 2);
```

The 2 bits 3:2 are cleared, selecting a clock of CCLK/4 for the GPIO

PINconnect

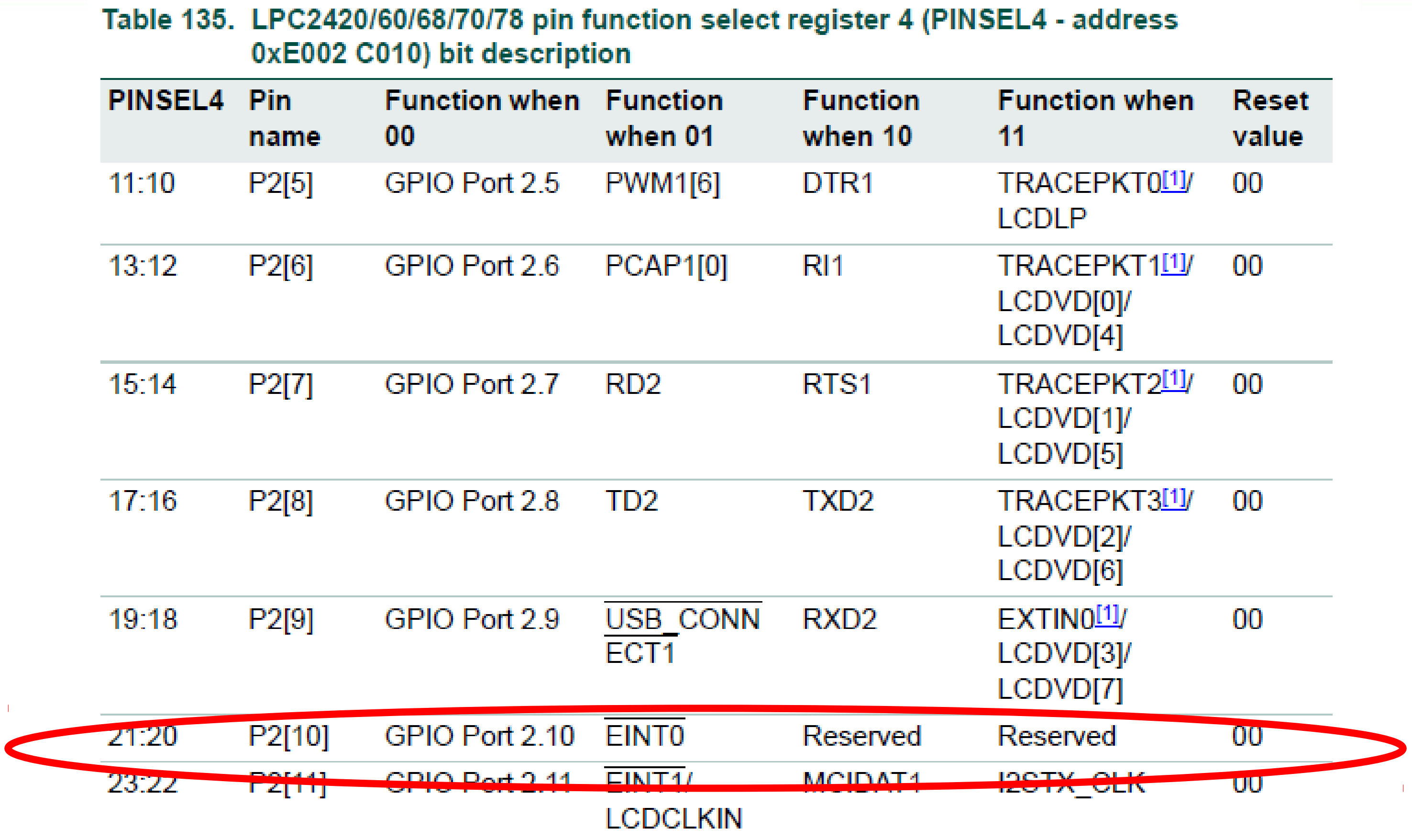


Table 135. LPC2420/60/68/70/78 pin function select register 4 (PINSEL4 - address 0xE002 C010) bit description

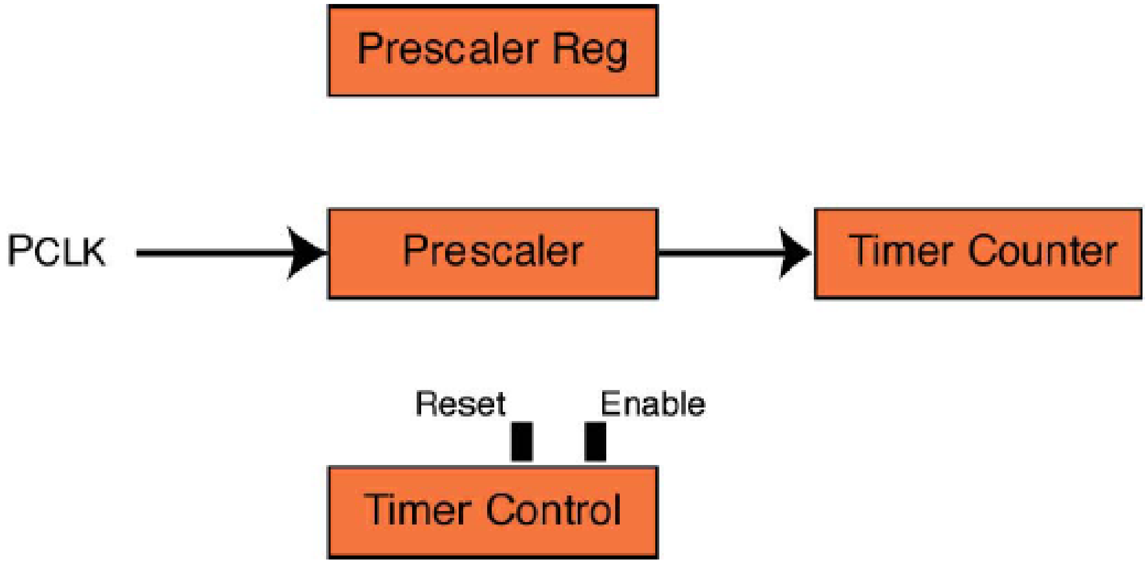
PINSEL4	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
11:10	P2[5]	GPIO Port 2.5	PWM1[6]	DTR1	TRACEPKT0[1]/ LCDLP	00
13:12	P2[6]	GPIO Port 2.6	PCAP1[0]	RI1	TRACEPKT1[1]/ LCDVD[0]/ LCDVD[4]	00
15:14	P2[7]	GPIO Port 2.7	RD2	RTS1	TRACEPKT2[1]/ LCDVD[1]/ LCDVD[5]	00
17:16	P2[8]	GPIO Port 2.8	TD2	TXD2	TRACEPKT3[1]/ LCDVD[2]/ LCDVD[6]	00
19:18	P2[9]	GPIO Port 2.9	$\overline{\text{USB\_CONNECT1}}$	RXD2	EXTIN0[1]/ LCDVD[3]/ LCDVD[7]	00
21:20	P2[10]	GPIO Port 2.10	$\overline{\text{EINT0}}$	Reserved	Reserved	00
23:22	P2[11]	GPIO Port 2.11	$\overline{\text{EINT1}}$/ LCDCLKIN	MCIDAT1	I2STX_CLK	00

```
/* function select for P2.10 (GPIO) in PINSEL4
(PINSEL4[21..20] = 0b00 ) (RW) */
  PINSEL4 &= ~(3 << 20) ;
```

General purpose timer

- The LPC2478 includes four 32-bit Timer/Counters
- Count cycles of the system derived clock or an externally-supplied clock
- Include programmable 32-bit prescaler
- Can optionally generate interrupts or perform other actions at specified timer values, based on four match registers
  - Set LOW on match, Set HIGH on match, Toggle on match, Do nothing on match.
- The Timer/Counter also includes four capture inputs to trap the timer value when an input signal transitions
  - A capture event may also optionally generate an interrupt

Prescaler

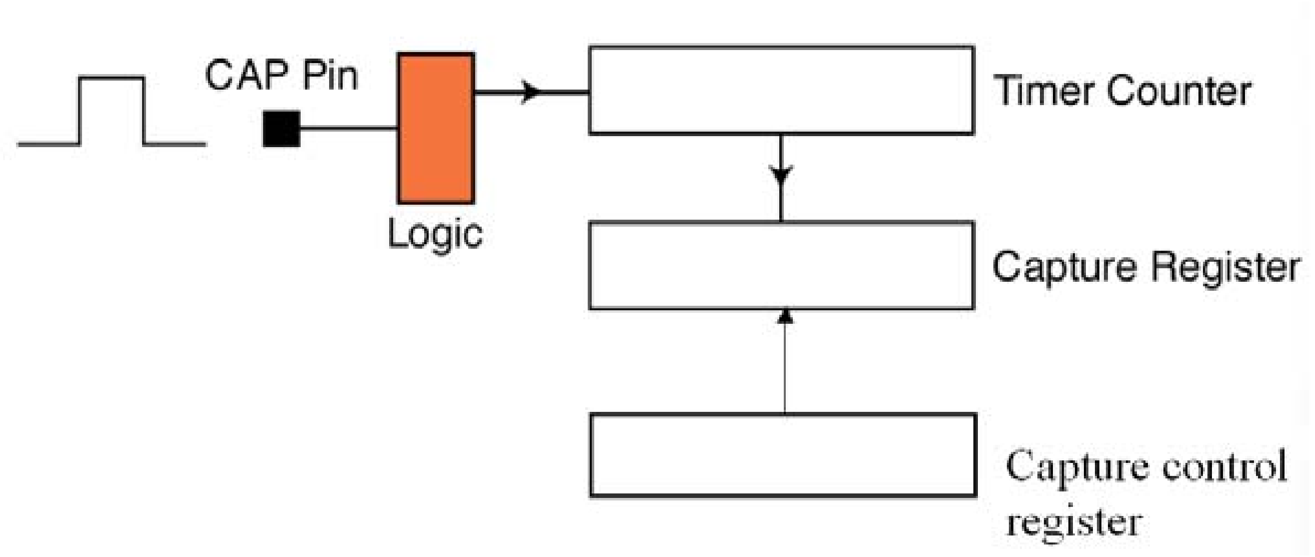

Prescaler Reg
PCLK
Prescaler
Timer Counter
Reset
Enable
Timer Control

Prescaler and Timer counter control

- Prescaler :
  - Each PCLK edge the prescaler counter is incremented
  - When the prescaler counter equals the prescaler register the timer counter is incremented and the prescaler counter is cleared
- Timer Control register :
  - Enable or disable the 2 counter (prescaler and timer)
  - reset of the timer counter and the prescaler counter

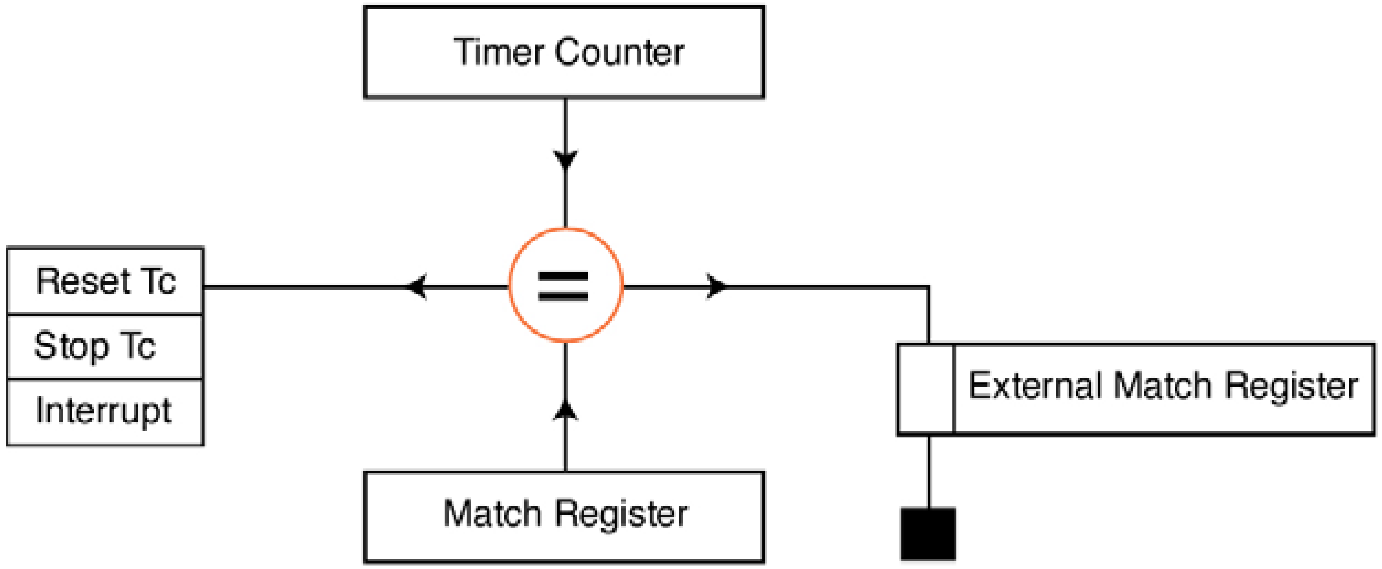
Capture mode

- Use to measure pulse duration
  - Counter captured on external events on CAP pin
    - Rising edge, falling edge, toggle
  - Interrupt request can be generated by a capture



Match mode

- Used to control the counter
  - Disable or reset the counters
- Can generate an interrupt request on match
- The pin level can be changed on match (external match register)
  - Set, cleared, toggle



Timing on match

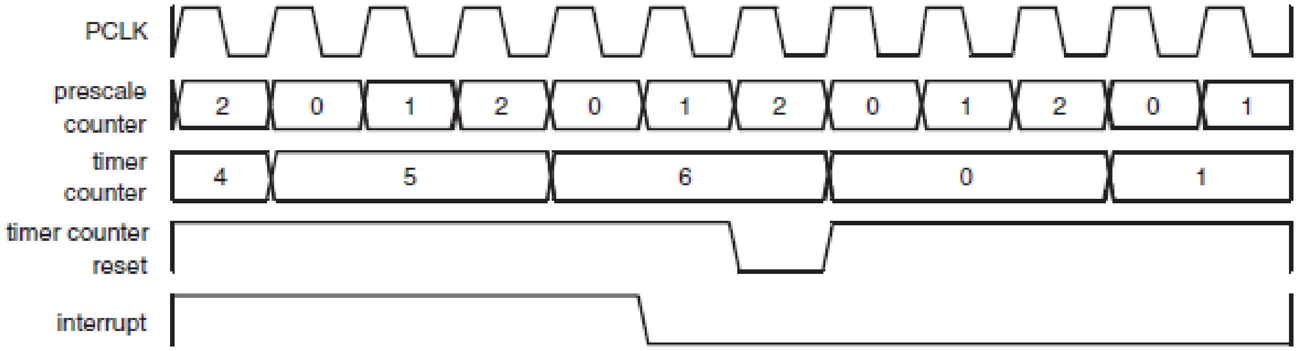


Fig 129. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled.

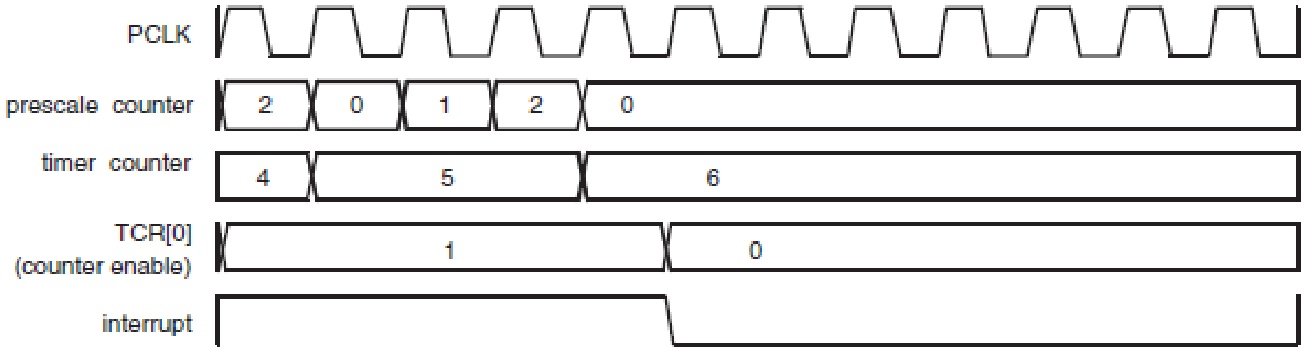


Fig 130. A timer Cycle in Which PR=2, MRx=6, and both interrupt and stop on match are enabled

User manual

Table 548: Timer Control Register (TCR, TIMERn: TnTCR - addresses 0xE000 4004, 0xE000 8004, 0xE007 0004, 0xE007 4004) bit description

Bit	Symbol	Description	Reset Value
0	Counter Enable	When one, the Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled.	0
1	Counter Reset	When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until TCR[1] is returned to zero.	0
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 550: Match Control Register (T[0/1/2/3]MCR - addresses 0xE000 4014, 0xE000 8014, 0xE007 0014, 0xE007 4014) bit description

Bit	Symbol	Value	Description	Reset Value
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		0	This interrupt is disabled	
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.	0
		0	Feature disabled.	
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.	0
		0	Feature disabled.	

Lab Example

```
static void mdelay(unsigned int ms)
{
  T1TCR = 0x02;          // stop and reset timer
  T1PR  = 0x00;          // set prescaler to zero
  T1MR0 = ms * (Fpclk / 1000); // Fpclk = 36000000
  T1MCR = 0x04;          // stop timer on match
  T1TCR = 0x01;          // start timer

  //wait until delay time has elapsed : test the 'enable' bit
  while(T1TCR & 0x01)
    ;
}
```

I2C

- Inter Integrated Circuit (Two Wire Interface)
- Two wire communication bus (synchronous serial transmission)
- Multimaster
- 400kbit/s (for slow devices)
- Each device has an address (8 or 10 bits) which is used when addressed in slave mode

I2C bus connection

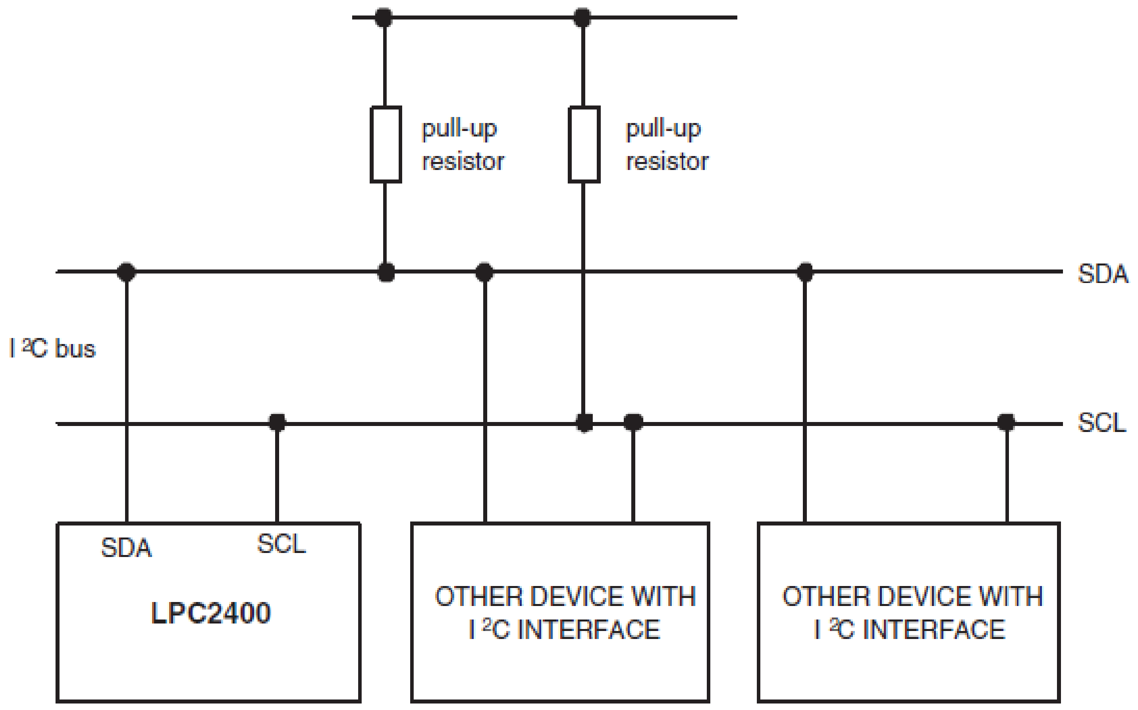

pull-up resistor
pull-up resistor
SDA
I ²C bus
SCL
SDA
SCL
LPC2400
OTHER DEVICE WITH I ²C INTERFACE
OTHER DEVICE WITH I ²C INTERFACE

Fig 111. I^{2}C bus configuration

Transfer

- Master drives the clocks and initiate transfer
- Slave respond to master request
- A Transmission is started by a "start" sequence
- Data are transferred in sequence of 8 bits (from/to master) MSb first
  - Data are changed during low edge of clock
  - Data must be stable during high edge of clock
- Transmission ends with a "stop" sequence
- For each 8 bits data receiver must acknowledge sender by sending an "ack" bit (low level)

Start and stop conditions

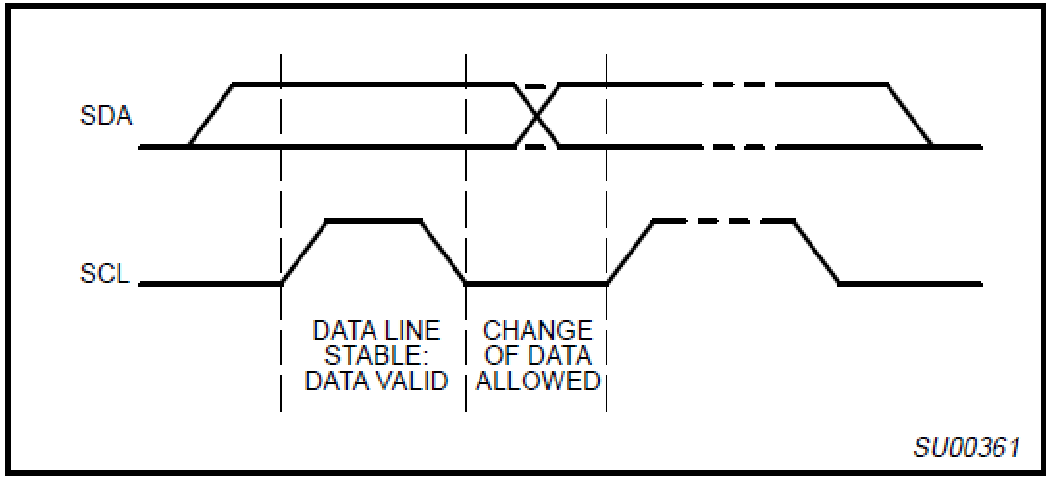
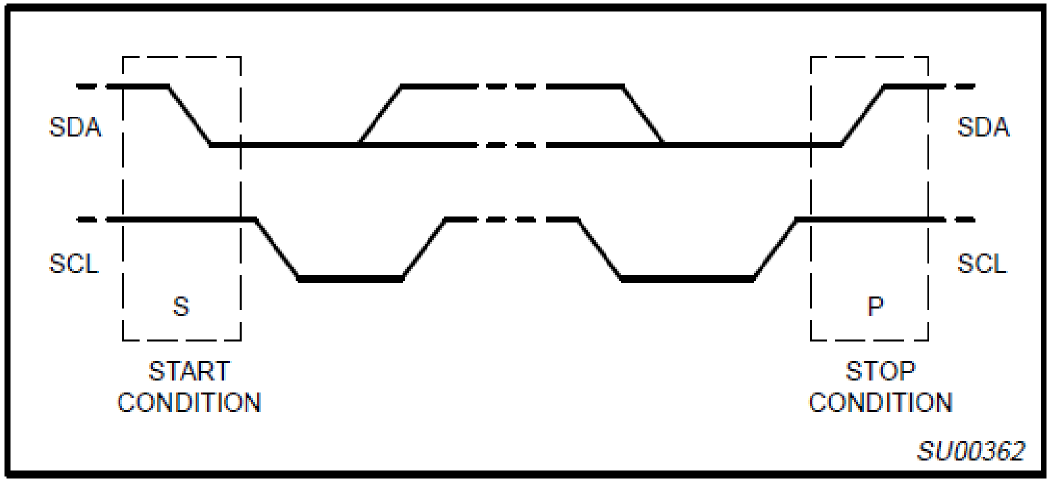


Figure 5. Bit transfer on the I^{2}C-bus



Transfers

- Transfer from master to slave
  - First byte transmitted by master is slave address (7 bits)
  - The 8th bit is low signaling a write to the device
  - Next follows a numbers of data bytes
  - Slave returns an ACK bit after each received byte

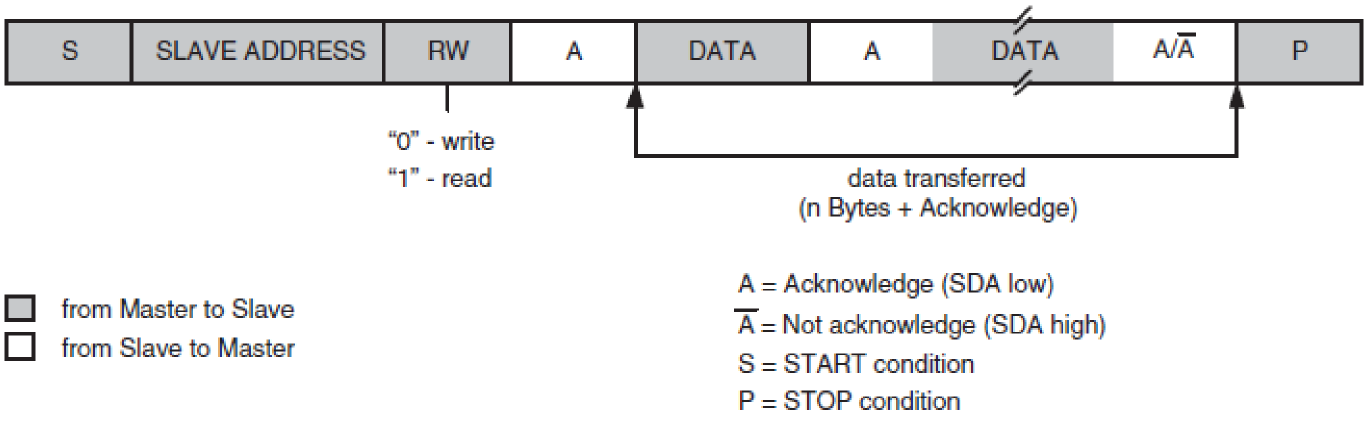


Fig 112. Format in the Master Transmitter mode

Transfers

- Transfer from slave to master
  - First byte transmitted by master is slave address (7 bits)
  - The 8th bit is high signaling a read from the device
  - Next follows a numbers of data bytes send by the slave
  - The master send a NACK to stop the reading

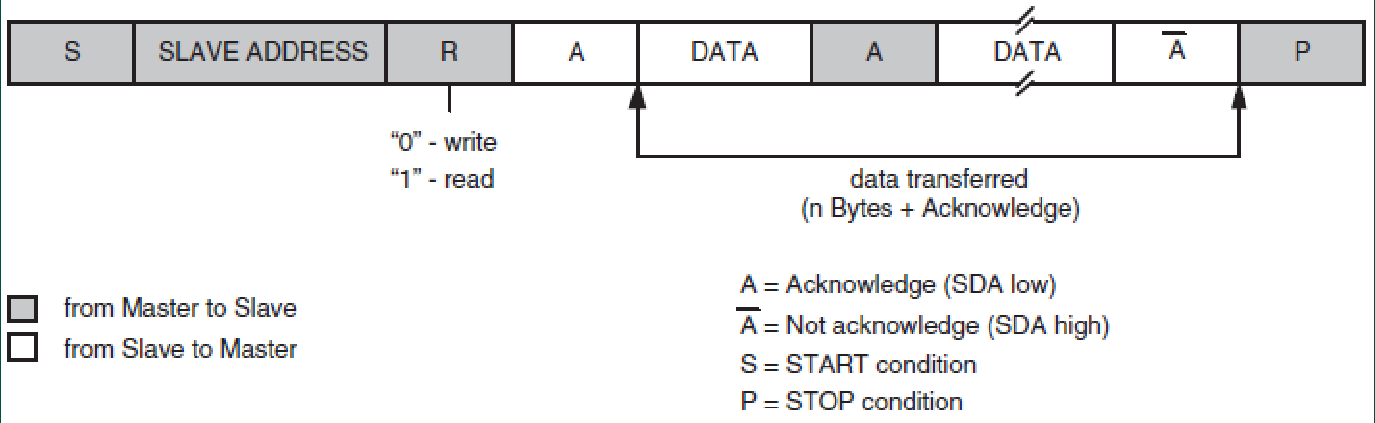


Fig 113. Format of Master Receive mode

LPC I2C interface

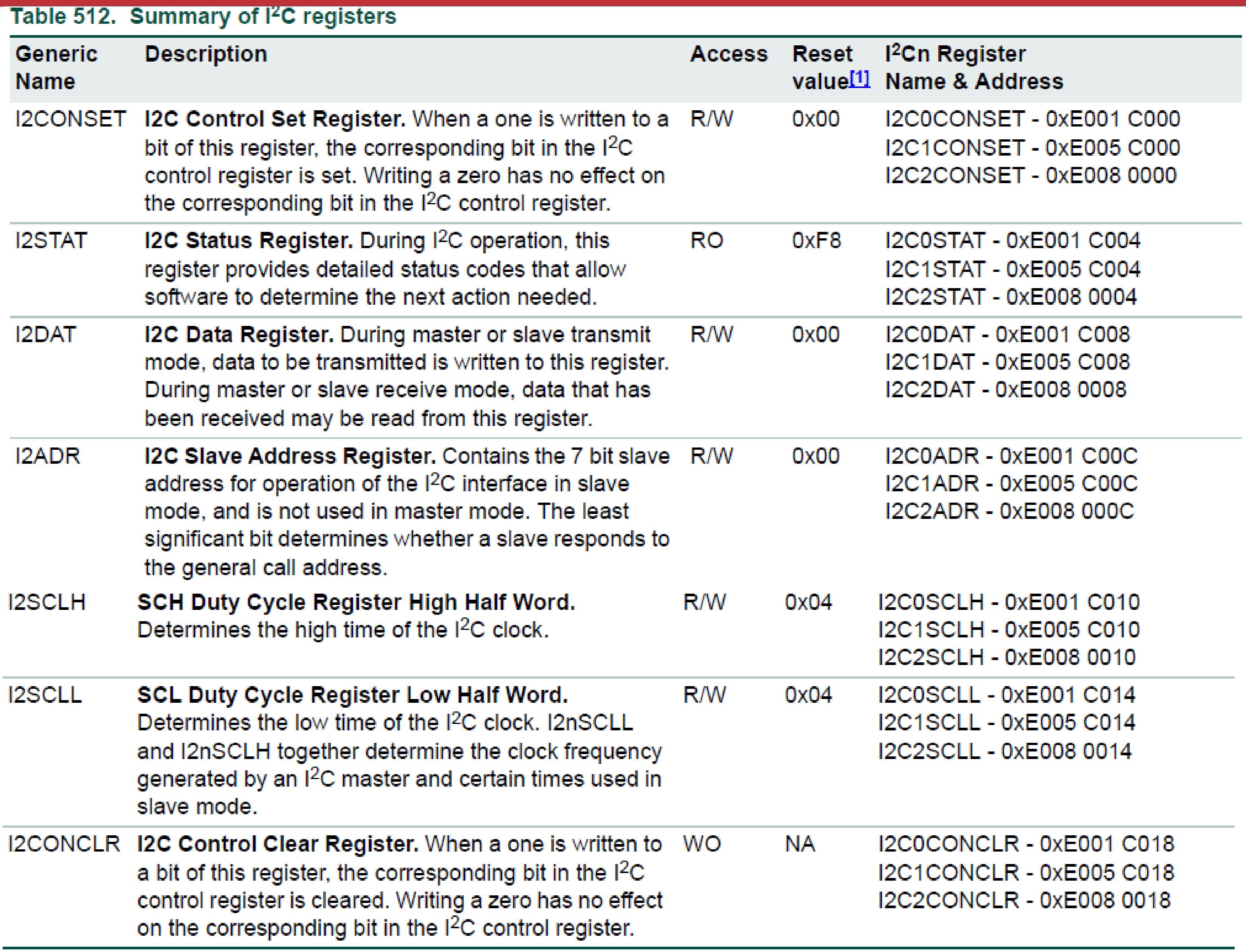
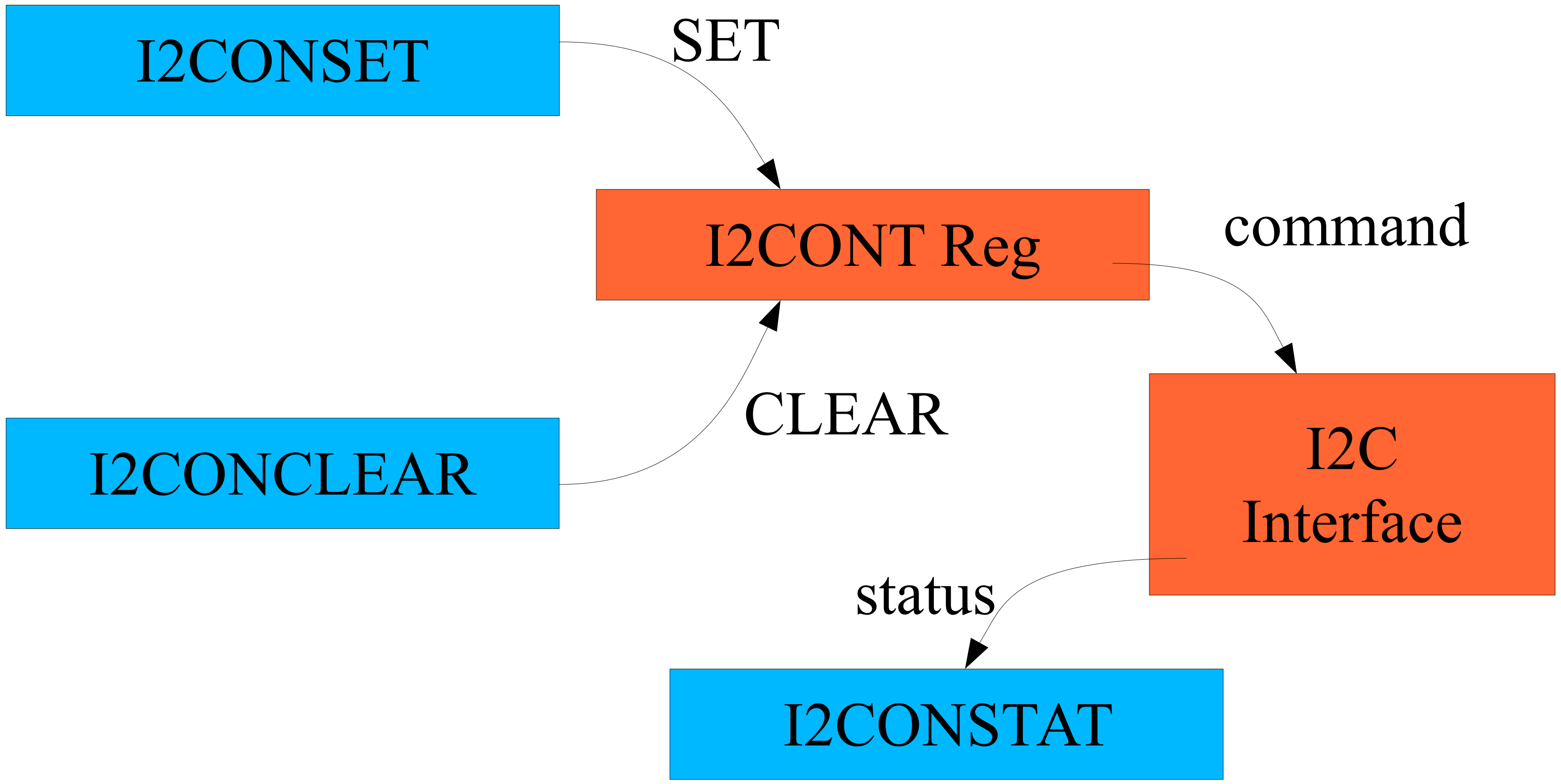


Table 512. Summary of I²C registers

Generic Name	Description	Access	Reset value[1]	I²Cn Register Name & Address
I2CONSET	**I2C Control Set Register.** When a one is written to a bit of this register, the corresponding bit in the I²C control register is set. Writing a zero has no effect on the corresponding bit in the I²C control register.	R/W	0x00	I2C0CONSET - 0xE001 C000 I2C1CONSET - 0xE005 C000 I2C2CONSET - 0xE008 0000
I2STAT	**I2C Status Register.** During I²C operation, this register provides detailed status codes that allow software to determine the next action needed.	RO	0xF8	I2C0STAT - 0xE001 C004 I2C1STAT - 0xE005 C004 I2C2STAT - 0xE008 0004
I2DAT	**I2C Data Register.** During master or slave transmit mode, data to be transmitted is written to this register. During master or slave receive mode, data that has been received may be read from this register.	R/W	0x00	I2C0DAT - 0xE001 C008 I2C1DAT - 0xE005 C008 I2C2DAT - 0xE008 0008
I2ADR	**I2C Slave Address Register.** Contains the 7 bit slave address for operation of the I²C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the general call address.	R/W	0x00	I2C0ADR - 0xE001 C00C I2C1ADR - 0xE005 C00C I2C2ADR - 0xE008 000C
I2SCLH	**SCH Duty Cycle Register High Half Word.** Determines the high time of the I²C clock.	R/W	0x04	I2C0SCLH - 0xE001 C010 I2C1SCLH - 0xE005 C010 I2C2SCLH - 0xE008 0010
I2SCLL	**SCL Duty Cycle Register Low Half Word.** Determines the low time of the I²C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I²C master and certain times used in slave mode.	R/W	0x04	I2C0SCLL - 0xE001 C014 I2C1SCLL - 0xE005 C014 I2C2SCLL - 0xE008 0014
I2CONCLR	**I2C Control Clear Register.** When a one is written to a bit of this register, the corresponding bit in the I²C control register is cleared. Writing a zero has no effect on the corresponding bit in the I²C control register.	WO	NA	I2C0CONCLR - 0xE001 C018 I2C1CONCLR - 0xE005 C018 I2C2CONCLR - 0xE008 0018


I2CONSET
SET
I2CONT Reg
command
CLEAR
I2CONCLEAR
I2C Interface
status
I2CONSTAT

Handling the interface

- Master Transmitter mode
  - Initialize ICONSET (clear SI/STA/STO in I2CONCLR)

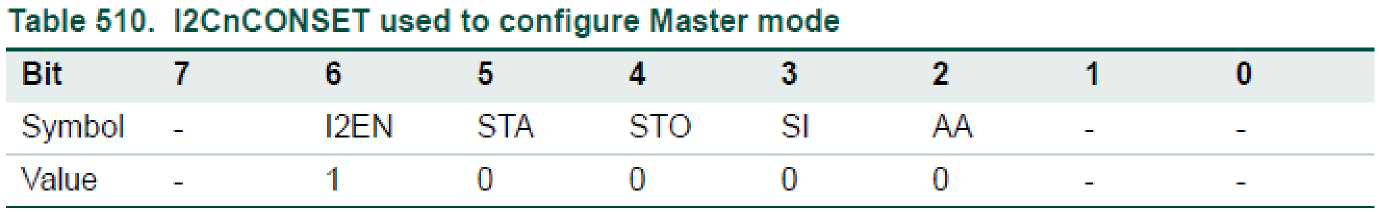


Table 510. I2CnCONSET used to configure Master mode

Bit	7	6	5	4	3	2	1	0
Symbol	-	I2EN	STA	STO	SI	AA	-	-
Value	-	1	0	0	0	0	-	-

  - Set the STA bit (SI bit is set when done and a new status code is present in I2STAT)
  - Place data in I2DAT register (Address for first byte)
  - Clear SI and STA
  - Wait for SI (set when data has been sent, new status code)
  - Place new data
  - …
  - Set STO to end transmission

Sample code

```
 I20CONSET =  I2C_STA;                            /* send START */
  while (!(I20CONSET & I2C_SI));                  /* Wait for START */
/* check status to handle error */
  I20CONCLR =  I2C_SI | I2C_STA;                  /* clear SI and STA */

  I20DAT    =  slave_address;                     /* slave address */
  while (!(I20CONSET & I2C_SI));                  /* Wait for ADDRESS send */
/* check status to handle error (nack)*/
  I20CONCLR =  I2C_SI;                            /* clear SI */

  I20DAT    =  data0;                             /* data 0*/
  while (!(I20CONSET & I2C_SI));                  /* Wait for DATA send */
/* check status to handle error (nack)*/
  I20CONCLR =  I2C_SI;                            /* clear SI */

  I20CONSET =  I2C_STO;                           /* send STOP */
  while (I20CONSET & I2C_STO);                    /* Wait for STOP  */
                                          /* note : STO is cleared automatically */
```

Note

- Controlling interface
  - For every events
    - SI is set
    - A status code is present in I2STAT
  - When SI is set, the status code can be used to take appropriate action
  - After each operation, software must wait for SI to be set (interruption can be used)
- Bit AA is used to allow interface to become slave
- Repeated STA (new start before stop) must be used with some interface (selecting register inside a device before a read by example)

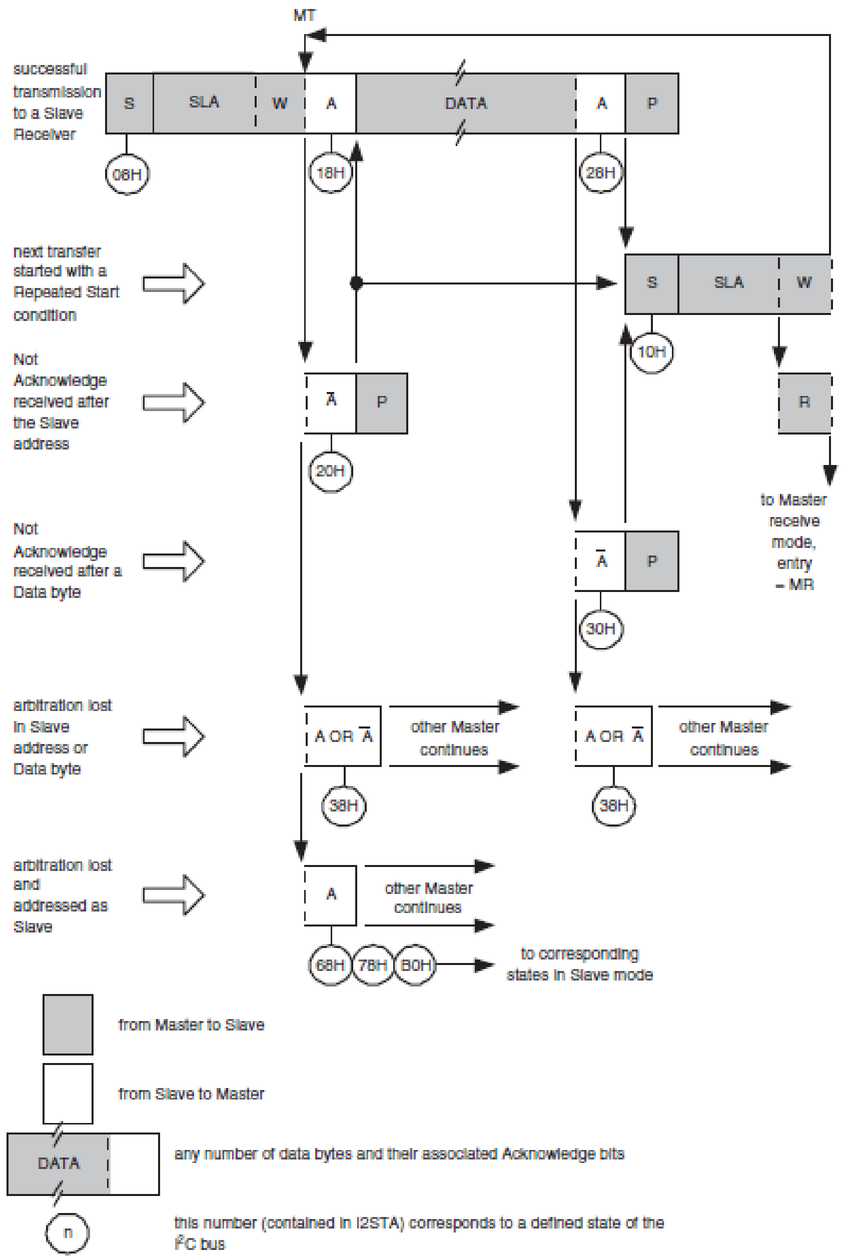

MT
successful transmission to a Slave Receiver
S
SLA
W
A
DATA
A
P
08H
18H
28H
next transfer started with a Repeated Start condition
S
SLA
W
10H
R
to Master receive mode, entry = MR
Not Acknowledge received after the Slave address
Ā
P
20H
Not Acknowledge received after a Data byte
Ā
P
30H
arbitration lost in Slave address or Data byte
A OR Ā
other Master continues
38H
A OR Ā
other Master continues
38H
arbitration lost and addressed as Slave
A
other Master continues
68H
78H
B0H
to corresponding states in Slave mode
from Master to Slave
from Slave to Master
DATA
any number of data bytes and their associated Acknowledge bits
n
this number (contained in I2STA) corresponds to a defined state of the I²C bus

Fig 120. Format and States in the Master Transmitter mode

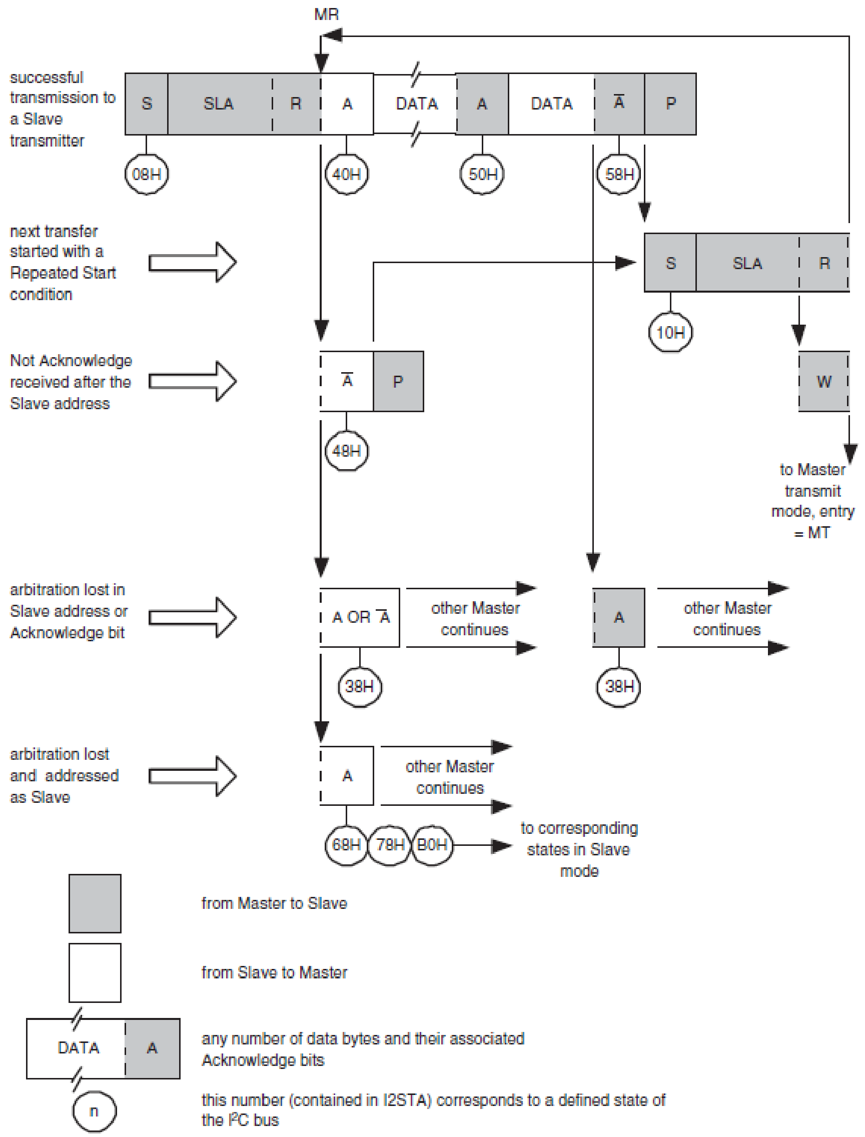

MR
successful transmission to a Slave transmitter
S
SLA
R
A
DATA
A
DATA
Ā
P
08H
40H
50H
58H
next transfer started with a Repeated Start condition
S
SLA
R
10H
W
to Master transmit mode, entry = MT
Not Acknowledge received after the Slave address
Ā
P
48H
arbitration lost in Slave address or Acknowledge bit
A OR Ā
other Master continues
38H
A
other Master continues
38H
arbitration lost and addressed as Slave
A
other Master continues
68H
78H
B0H
to corresponding states in Slave mode
from Master to Slave
from Slave to Master
DATA
A
any number of data bytes and their associated Acknowledge bits
n
this number (contained in I2STA) corresponds to a defined state of the I²C bus

Fig 121. Format and States in the Master Receiver mode

Sample 2 : using I2CISR

```
Void I2CISR(void) {
        switch(I2STAT){
                case (0x08) :                               //start bit
                        I2CONCLR = I2C_STA;
                        I2DAT = I2CAddress;                 //send address
                        break;
                case(0x18):                                 //slave address ack
                        I2DAT = I2CData;
                        break;
                case(0x20):                                 // slave address nack
                        I2DAT = I2CAddress;
                        break;
                case(0x28):                                 // data ack
                        I2CONSET = I2C_STO;
                        break
                default :
                        break;
        }
        I2CONCLR = I2C_SI;                                  // clear interrupt flag
        VICVectAddr = 0;                                    //VIC ack
}
```

Exemple of device : PC19532 (led driver)

- I2C 16 led driver
- Controlled by 10 registers
- Writing
  - Sending address of the device : 0xC0
  - Sending the number of the register
  - Sending data to the device register
- Reading
  - Sending address of the device : 0xC0
  - Sending the number of the register
  - Sending address of the device : 0xC1 with repeated STA
  - read data from the device register

Bus transaction

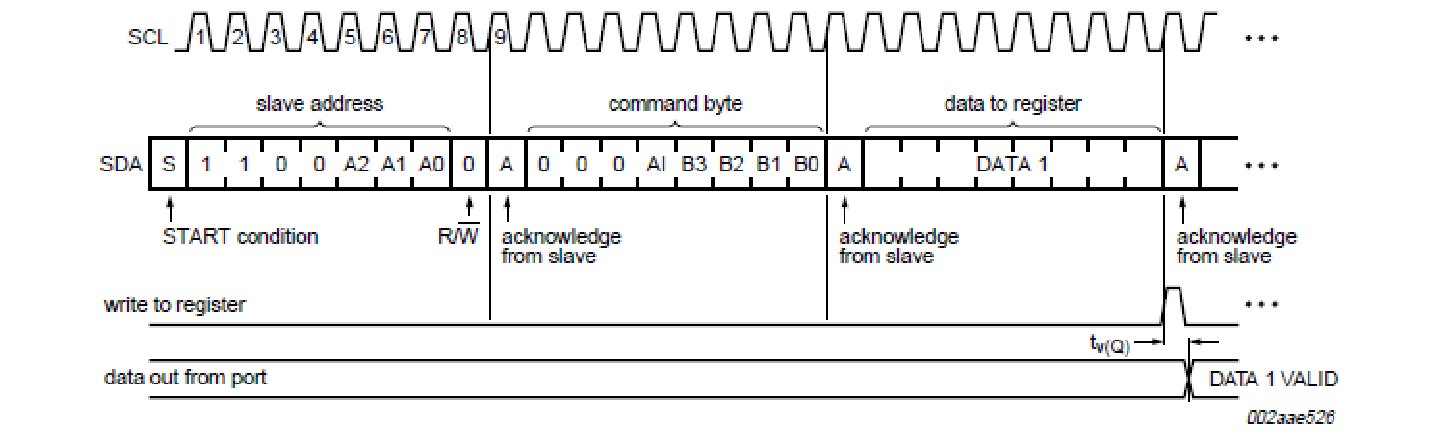


Fig 11. Write to register

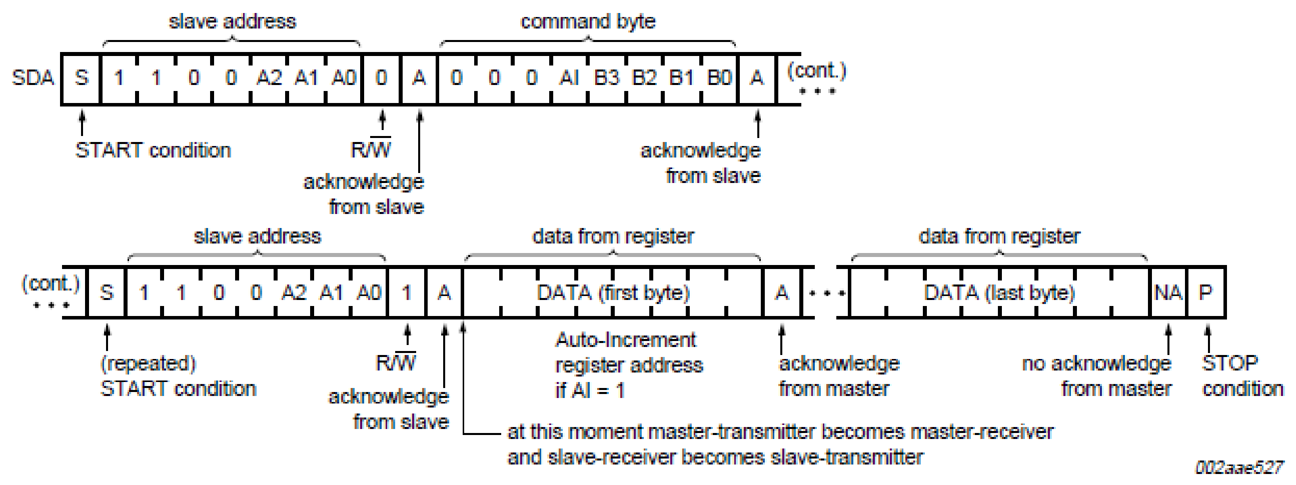


Fig 12. Read from register

PCA9532 registers

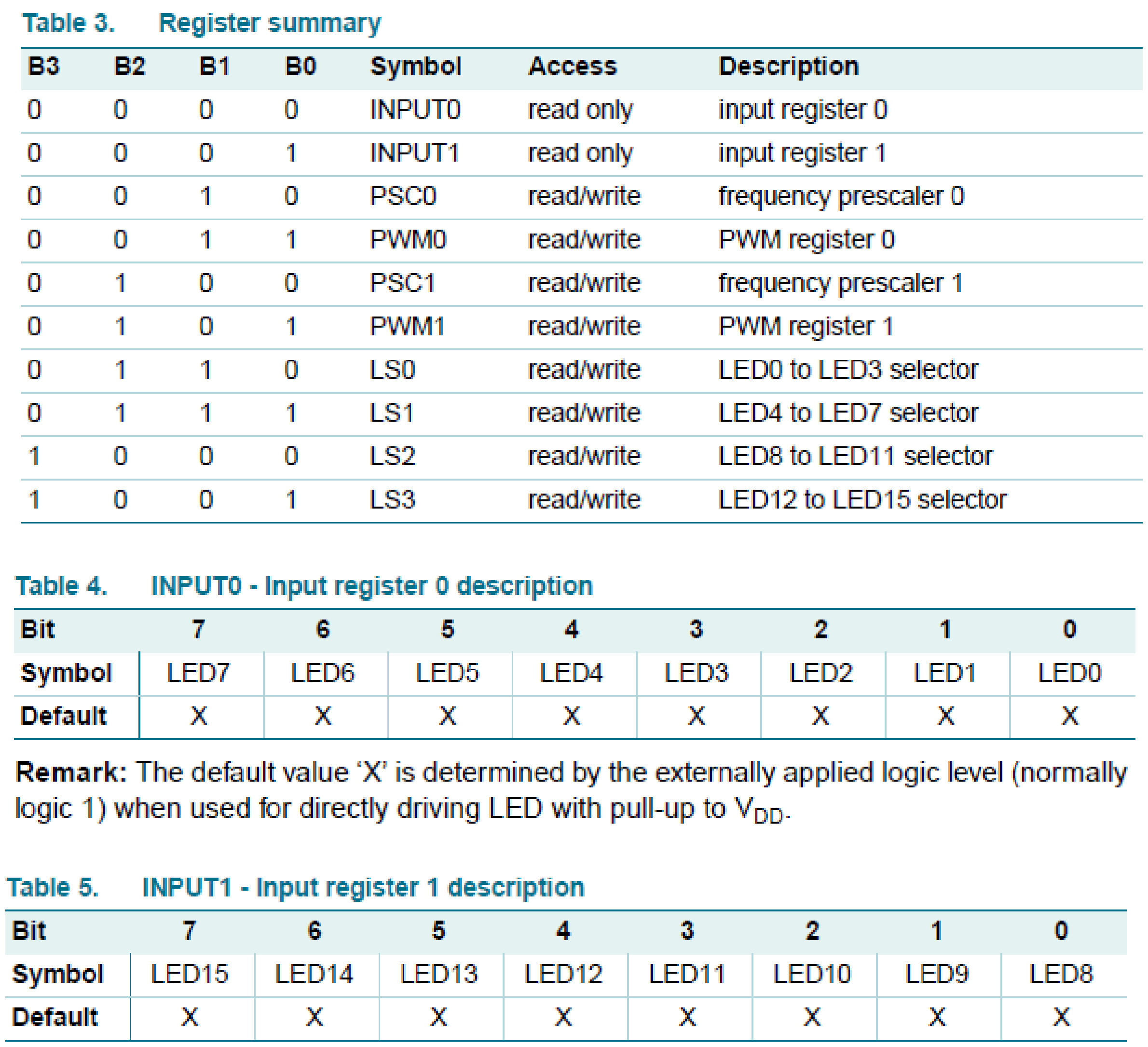


Table 3. Register summary

B3	B2	B1	B0	Symbol	Access	Description
0	0	0	0	INPUT0	read only	input register 0
0	0	0	1	INPUT1	read only	input register 1
0	0	1	0	PSC0	read/write	frequency prescaler 0
0	0	1	1	PWM0	read/write	PWM register 0
0	1	0	0	PSC1	read/write	frequency prescaler 1
0	1	0	1	PWM1	read/write	PWM register 1
0	1	1	0	LS0	read/write	LED0 to LED3 selector
0	1	1	1	LS1	read/write	LED4 to LED7 selector
1	0	0	0	LS2	read/write	LED8 to LED11 selector
1	0	0	1	LS3	read/write	LED12 to LED15 selector

Table 4. INPUT0 - Input register 0 description

Bit	7	6	5	4	3	2	1	0
Symbol	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
Default	X	X	X	X	X	X	X	X

Remark: The default value 'X' is determined by the externally applied logic level (normally logic 1) when used for directly driving LED with pull-up to V_{DD}.

Table 5. INPUT1 - Input register 1 description

Bit	7	6	5	4	3	2	1	0
Symbol	LED15	LED14	LED13	LED12	LED11	LED10	LED9	LED8
Default	X	X	X	X	X	X	X	X

PCA9532 registers

00 = output is set high-impedance (LED off; defau

01 = output is set LOW (LED on)

10 = output blinks at PWM0 rate

11 = output blinks at PWM1 rate

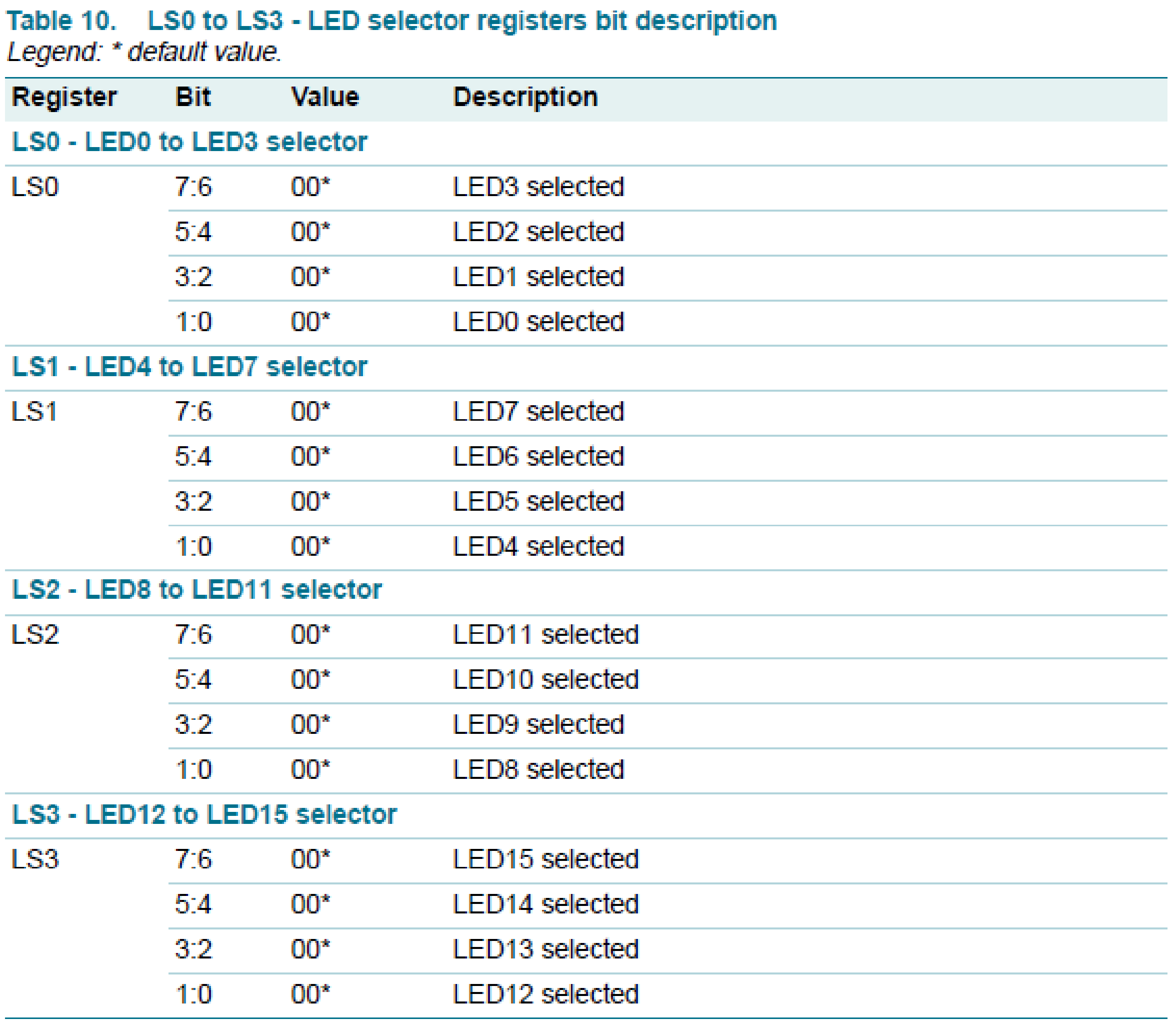


Table 10. LS0 to LS3 - LED selector registers bit description

*Legend: * default value.*

Register	Bit	Value	Description
LS0 - LED0 to LED3 selector			
LS0	7:6	00*	LED3 selected
	5:4	00*	LED2 selected
	3:2	00*	LED1 selected
	1:0	00*	LED0 selected
LS1 - LED4 to LED7 selector			
LS1	7:6	00*	LED7 selected
	5:4	00*	LED6 selected
	3:2	00*	LED5 selected
	1:0	00*	LED4 selected
LS2 - LED8 to LED11 selector			
LS2	7:6	00*	LED11 selected
	5:4	00*	LED10 selected
	3:2	00*	LED9 selected
	1:0	00*	LED8 selected
LS3 - LED12 to LED15 selector			
LS3	7:6	00*	LED15 selected
	5:4	00*	LED14 selected
	3:2	00*	LED13 selected
	1:0	00*	LED12 selected

Example

- PCA9532 is connected to pin P0.27 (SDA0) and to pin P0.28 (SCL0) (an I2C EPROM is also connected)
- Initialization
  - Power activation for I2C0 : PCONP0 |= 1 <<7
  - Clock division : PCLKSEL0[15:14] = 00 for pclk = cclk/4 = 18MHz => PCLKSEL0 &= ~(3<<14)
  - Pin : PINSEL1 [23:22] = 01 and PINSEL1 [25:24] = 01
    - PINSEL &= ~((3 <<22) | (3 << 24))
    - PINSEL1 |= (1 << 22) | (01 << 24)
  - Clock timing (100kHz)
    - High duty cycle = 90 pclk tic : I20SCLH = 90
    - Low duty cycle = 90 pclk tic : I20SCLL = 90
  - I2C0CONCLR = I2C_AA | I2C_SI | I2C_STO | I2C_STA | I2C_I2EN
  - I2C0CONSET = I2C_I2EN

Sample code to light led 8 to 11

```
 I20CONSET =  I2C_STA;                          /* send START */
  while (!(I20CONSET & I2C_SI));                /* Wait for START */
/* check status to handle error */
  I20CONCLR =  I2C_SI | I2C_STA;                /* clear SI and STA */
  I20DAT    =  0xC0;                            /* PCA address */
  while (!(I20CONSET & I2C_SI));                /* Wait for ADDRESS send */
/* check status to handle error (nack)*/
  I20CONCLR =  I2C_SI;                          /* clear SI  */
 I20DAT    =  0x18;                   /* select register LS2 and AI*/
   while (!(I20CONSET & I2C_SI));               /* Wait for DATA send */
/* check status to handle error (nack)*/
  I20CONCLR =  I2C_SI;                          /* clear SI  */
  I20DAT    =  0x01 | 0x4 | 0x10 | 0x40 ;       /* 4 leds on : led8 to 11 */
  while (!(I20CONSET & I2C_SI));                /* Wait for DATA send  */
/* check status to handle error (nack)*/
  I20CONCLR =  I2C_SI;                          /* clear SI  */
  I20CONSET =  I2C_STO;                         /* send STOP */
  while (I20CONSET & I2C_STO);                  /* Wait for STOP  */
                                      /* note : STO is cleared automatically */
```

VIC : Vectored Interrupt Controller

- 32 interrupt request inputs
- Interrupt request must be HIGH level
- VIC ORs vectored interrupt request to produce irq or fiq signal to the core
- Each interrupt can be enable or disable
- Each interrupt can be asserted by software
- Each interrupt is assigned to irq or fiq line
- Each interrupt is programmed with a priority on 4 bits
  - 0 : highest priority
  - 15 : lowest priority

Diagram

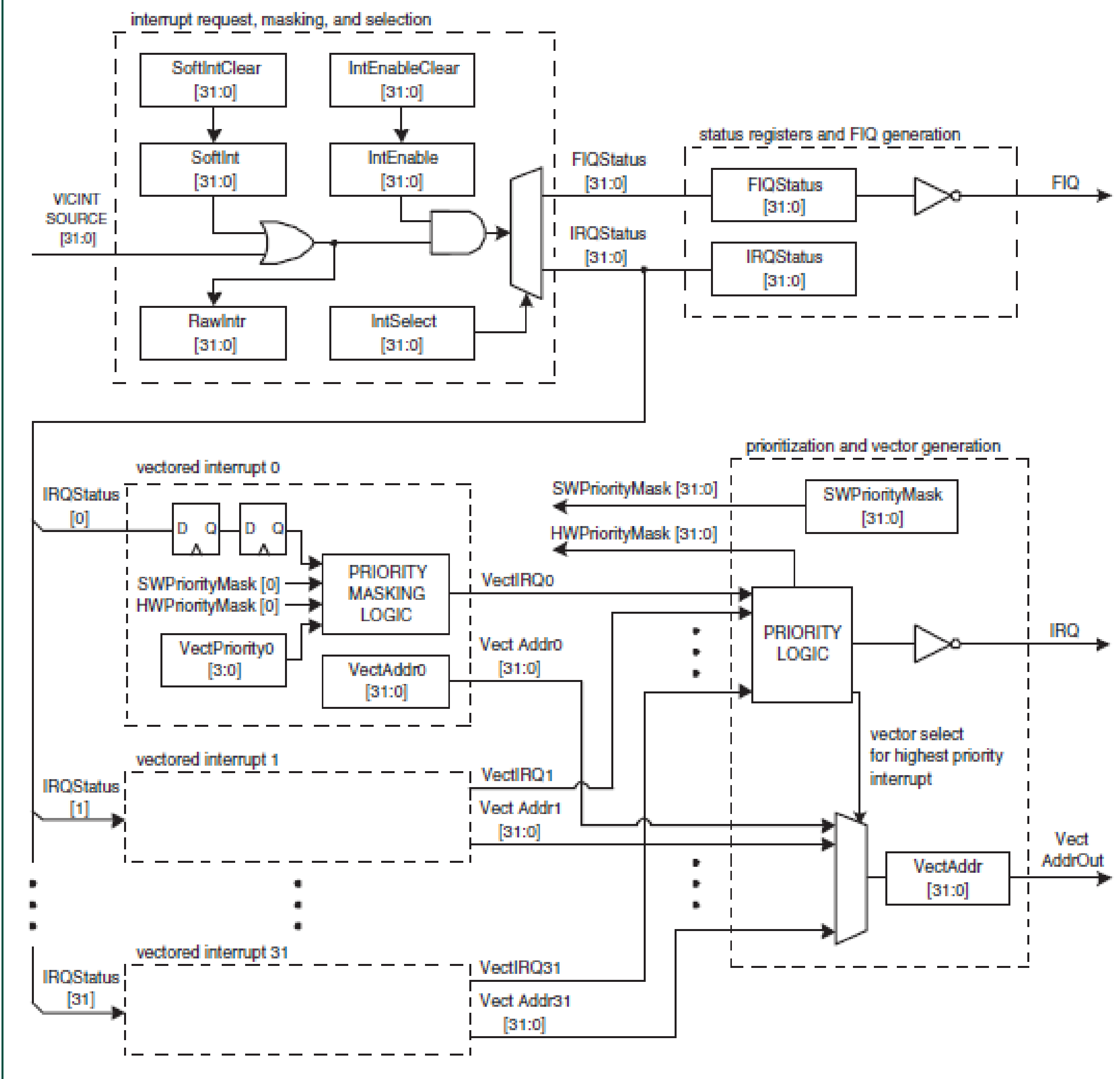


Fig 22. Block diagram of the Vectored Interrupt Controller

VIC registers

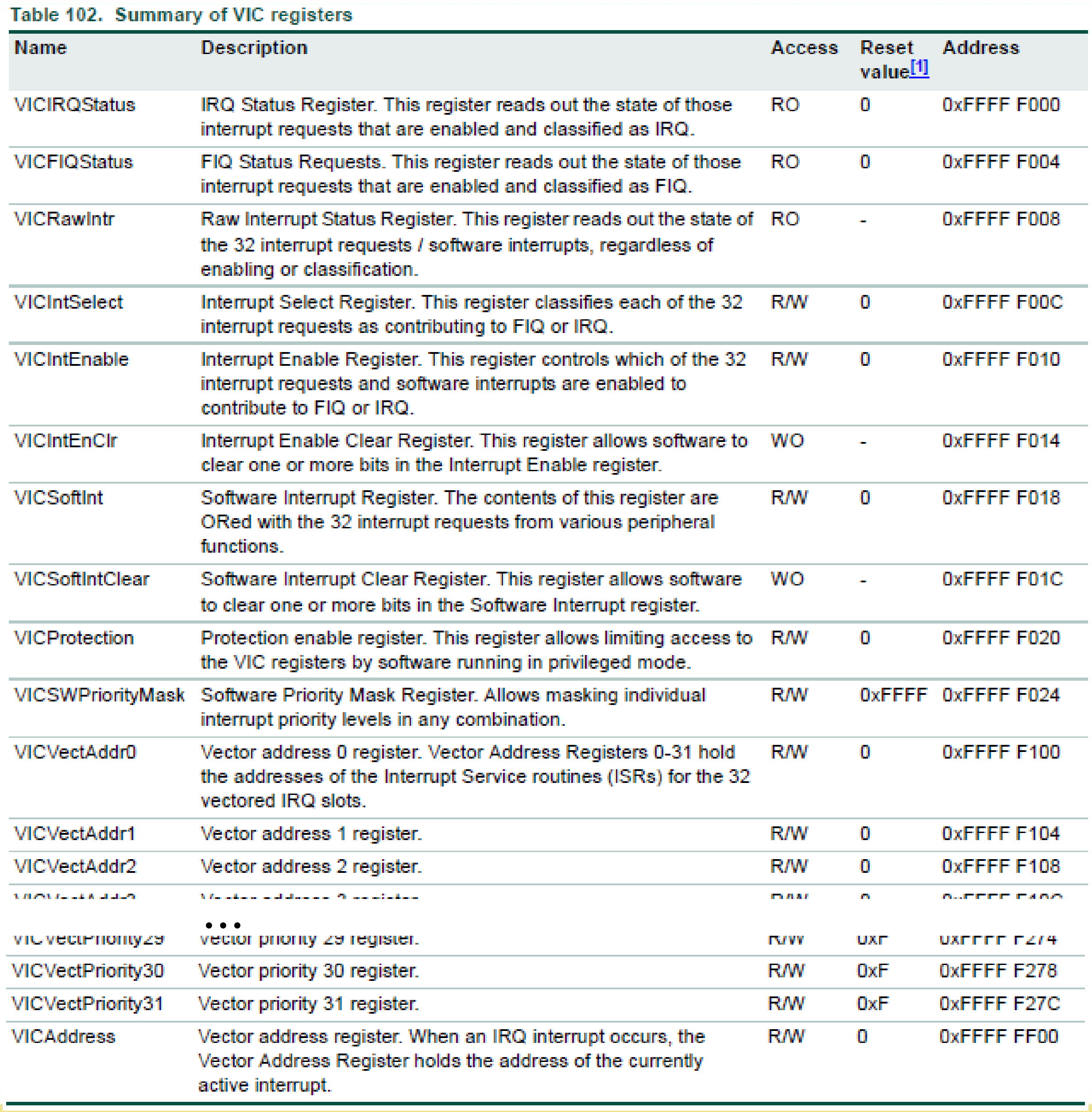


Table 102. Summary of VIC registers

Name	Description	Access	Reset value[1]	Address
VICIRQStatus	IRQ Status Register. This register reads out the state of those interrupt requests that are enabled and classified as IRQ.	RO	0	0xFFFF F000
VICFIQStatus	FIQ Status Requests. This register reads out the state of those interrupt requests that are enabled and classified as FIQ.	RO	0	0xFFFF F004
VICRawIntr	Raw Interrupt Status Register. This register reads out the state of the 32 interrupt requests / software interrupts, regardless of enabling or classification.	RO	-	0xFFFF F008
VICIntSelect	Interrupt Select Register. This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.	R/W	0	0xFFFF F00C
VICIntEnable	Interrupt Enable Register. This register controls which of the 32 interrupt requests and software interrupts are enabled to contribute to FIQ or IRQ.	R/W	0	0xFFFF F010
VICIntEnClr	Interrupt Enable Clear Register. This register allows software to clear one or more bits in the Interrupt Enable register.	WO	-	0xFFFF F014
VICSoftInt	Software Interrupt Register. The contents of this register are ORed with the 32 interrupt requests from various peripheral functions.	R/W	0	0xFFFF F018
VICSoftIntClear	Software Interrupt Clear Register. This register allows software to clear one or more bits in the Software Interrupt register.	WO	-	0xFFFF F01C
VICProtection	Protection enable register. This register allows limiting access to the VIC registers by software running in privileged mode.	R/W	0	0xFFFF F020
VICSWPriorityMask	Software Priority Mask Register. Allows masking individual interrupt priority levels in any combination.	R/W	0xFFFF	0xFFFF F024
VICVectAddr0	Vector address 0 register. Vector Address Registers 0-31 hold the addresses of the Interrupt Service routines (ISRs) for the 32 vectored IRQ slots.	R/W	0	0xFFFF F100
VICVectAddr1	Vector address 1 register.	R/W	0	0xFFFF F104
VICVectAddr2	Vector address 2 register.	R/W	0	0xFFFF F108
VICVectAddr3	Vector address 3 register.	R/W	0	0xFFFF F10C
	...			
VICVectPriority29	Vector priority 29 register.	R/W	0xF	0xFFFF F274
VICVectPriority30	Vector priority 30 register.	R/W	0xF	0xFFFF F278
VICVectPriority31	Vector priority 31 register.	R/W	0xF	0xFFFF F27C
VICAddress	Vector address register. When an IRQ interrupt occurs, the Vector Address Register holds the address of the currently active interrupt.	R/W	0	0xFFFF FF00

Registers

- **VICSoftInt** : ORed with interrupt request
  - **VICSoftIntClear** : to clear one or more bit in VICSoftInt
- **VICIntEnable** : enable soft and hard irq
  - **VICIntEnClear** : to clear one or more bit in VICIntEnable
- **VICProtect** : allow usr mode to access VIC register
- **VICIntSelect** : contribue to irq(0) or fiq (1)
- **VICIrqStatus/VICFiqStatus** : show active irq/fiq request
- **VICVectAddr0-31** : isr address for each request lines
- **VICVectPriority0-31** : priority for each request lines, 0 to 15 with 15 lowest priority
- **VICAddress** : address of isr that is to be serviced
  - Musts be written at end of isr to acknowledge the IRQ

Interrupt flow

- When interrupt N occurs, if interrupt is enable the irq line is asserted
- If the interrupt line is not masked
  - Bit N in VICIntEnable set
  - The current priority is lower than the priority assigned to the corresponding IRQ N
- The VICVectAddr*N* of associated interrupt is copied in VICAddress register to be read by software (most of time this is the isr address)
- The IRQ (or FIQ) line connected to the core is asserted
- …

- When software read **VICAddress** :
  - the irq (fiq) line to the core is de-asserted
  - Hardware priority in VIC is set to the highest priority irq pending (here N)
- During the time of the irq is serviced by software
  - If an irq M with lower priority appears : nothing occurs
  - If an irq M with higher priority appears : same stages as described before (activation of the irq line, copy of VICVectAddrM, ...)
- When interrupt is serviced, software must write a dummy value in **VICAddress**
  - This signal the end of the treatment and the Hardware priority in VIC is lowered to the higher pending irq priority

Interrupt lines

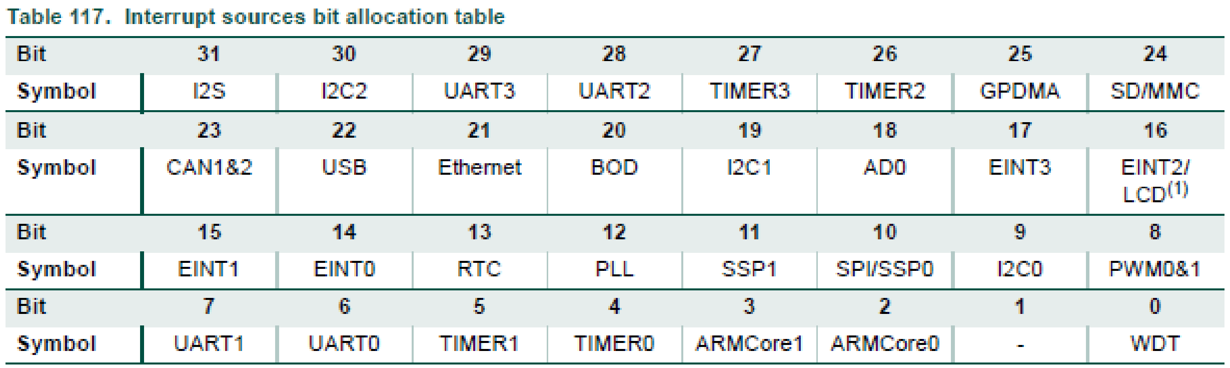


Table 117. Interrupt sources bit allocation table

Bit	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**
Symbol	I2S	I2C2	UART3	UART2	TIMER3	TIMER2	GPDMA	SD/MMC
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	CAN1&2	USB	Ethernet	BOD	I2C1	AD0	EINT3	EINT2/ LCD[1]
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	SSP1	SPI/SSP0	I2C0	PWM0&1
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT

Example of vectored irq usage

- For vectorized irq, each interrupt routine address (isrx) must be written in VICVectAddrx
- At irq vector address (0x18) instruction load pc with VIC address and so jump to the appropriate isr :

```
Vectors
        LDR PC, Reset_Addr
        LDR PC, Undef_Addr
        LDR PC, SWI_Addr
        LDR PC, PAbt_Addr
        LDR PC, DAbt_Addr
        NOP                    ; Reserved Vector
        LDR PC, [PC, #-0x0120] ; Vector from VicVectAddr
        LDR PC, FIQ_Addr
```

Exemple of configuration

- Configuring VIC for UART0

```
VICIntSelect &= ~(1<<6) ;                    /* IRQ contribution */
VICVectAddr6 = (unsigned long) uart_isr ; /* isr address */
VICVectPriority6 = 10;               /* priority = 10*/
VICIntEnable |= 1<<6 ;               /* enable uart0 IRQ */
```

Vectored interrupt flow

- 1. An interrupt occurs.
- 2. The ARM processor branches to either the IRQ or FIQ interrupt vector.
- 3. If the interrupt is an IRQ, read the VICVectAddr Register and branch to the interrupt service routine. You can do this using an LDR PC instruction. Reading the VICAddress Register updates the interrupt controllers hardware priority register.
- 4. Stack any registers that will be used to avoid any register corruption
- 5. Execute the service
- 6. Clear the requesting interrupt in the peripheral, or write to the VICSoftIntClear register if the request was generated by a software interrupt.
- 7. Restore the previously saved register
- 8. Write to the VICAddress Register. This clears the respective interrupt in the internal interrupt priority hardware.
- 9. Return from the interrupt. This re-enables the interrupts.

Vectored interrupt example code

```
0x18        LDR pc, [pc, #-0x120]            @ Load Vector into PC
@ ..................................................
vector_handler
            @Code to enable interrupt nesting
            STMFD r13!, {r0-r3, r12, lr} @ stack registers that will be corrupted by a function call

            @ Interrupt service routine...
            BL 2nd_level_handler             @ this corrupts lr_irq and r0-r3 and r12
@...
@Add code to clear the interrupt source;

@Code to exit handler
            LDMFD r13!, {r0-r3, r12, r14}    @ unstack lr_irq and r0-r3, r12
            LDR r1, =VectorAddr
            STR r0, [r1]             @ Acknowledge VIRQ serviced with a dummy write
            SUBS pc, lr, #4          @ Return from ISR
```

Vectored interrupt flow with nested interrupts

- 1. An interrupt occurs.
- 2. The ARM processor branches to either the IRQ or FIQ interrupt vector.
- 3. If the interrupt is an IRQ, read the VICVectAddr Register and branch to the interrupt service routine. You can do this using an LDR PC instruction. Reading the VICAddress Register updates the interrupt controllers hardware priority register.
- 4. Stack the workspace so that you can re-enable IRQ interrupts.
- 5. Enable the IRQ interrupts so that a higher priority can be serviced.
- 6. Execute the Interrupt Service Routine (ISR).
- 7. Disable the interrupts and restore the workspace.
- 8. Clear the requesting interrupt in the peripheral, or write to the VICSoftIntClear register if the request was generated by a software interrupt.
- 9. Write to the VICAddress Register. This clears the respective interrupt in the internal interrupt priority hardware.
- 10. Return from the interrupt. This re-enables the interrupts.

Vectored interrupt example code

```
0x18        LDR pc, [pc, #-0x120]              @ Load Vector into PC
@ ......................................................
vector_handler
            @ Code to enable interrupt nesting
            STMFD r13!, {r12, r14} @stack lr_irq and r12 [plus other regs used below, if appropriate]
            MRS r12, spsr             @ Copy spsr into r12...
            STMFD r13!, {r12}         @ and save to stack
            MSR cpsr_c, #0x1f         @ Switch to SYS mode, re-enable IRQ
            STMFD r13!, {r0-r3, r14}           @stack lr_sys and r0-r3

            @ Interrupt service routine...
            @Add code to clear the interrupt source; Code to exit handler
            BL 2nd_level_handler               @ this corrupts lr_sys and r0-r3

            LDMFD r13!, {r0-r3, r14}           @ unstack lr_sys and r0-r3
            MSR cpsr_c, #0x92         @ Disable IRQ, and return to IRQ mode
            LDMFD r13!, {r12}         @ unstack r12...
            MSR spsr_cxsf, r12        @ and restore spsr...
            LDMFD r13!, {r12, r14}             @ unstack registers
            LDR r1, =VectorAddr
            STR r0, [r1]              @ Acknowledge VIRQ serviced
            SUBS pc, lr, #4           @ Return from ISR
```

Interrupt using a library (to avoid asm)

- Some toolchains can provide entry/exit code of a interrupt routine
- Basic entry/exit code
  - To use more complex code (to allow nested interrupt or to switch context, ...) you still have to write the entry/exit code in asm
- With gcc you can use the “attribute” keyword to modify the entry/exit code of a function.
  - For a interrupt use “interrupt” attribute :

```
void myISR(void) __attribute__ ((interrupt));

Pour ARM :  __attribute__ ((interrupt ("IRQ")));
```

UART

- Universal Asynchronous Receiver/transmitter
  - Standard PC serial line
- Serial :data are transmitted bit after bit (lsb first)
- Asynchronous
  - No clock to synchronize symbol detection
  - Transmitter and receiver must use the same baud rate
  - Synchronization with start/stop bit
  - Automatic baud rate detection capable
- Full duplex via two different lines (RX and TX)

Asynchronous transmission detection

Figure 44. Asynchronous Mode: Start Bit Detection

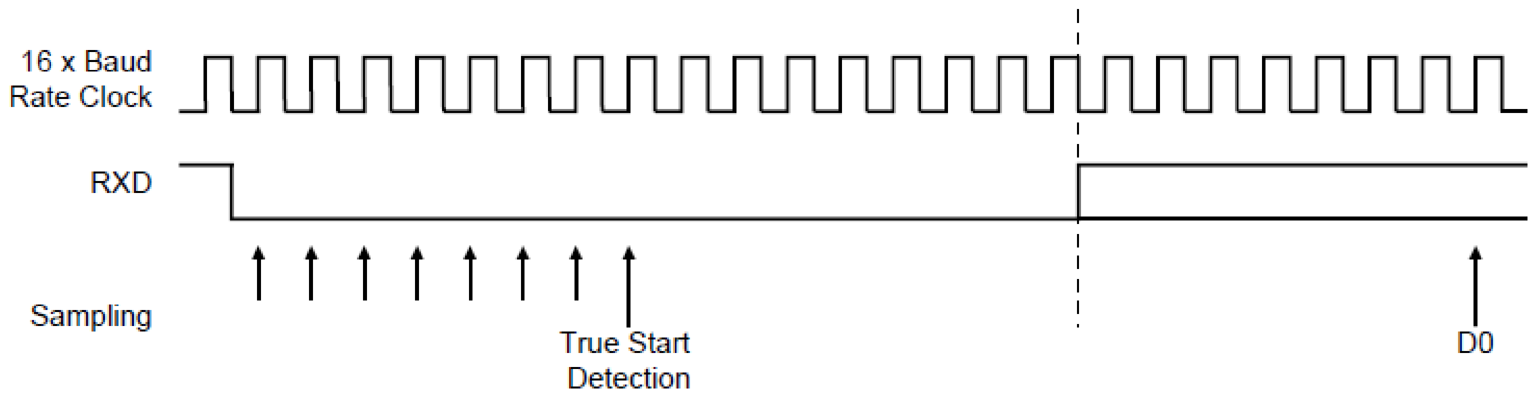


Figure 45. Asynchronous Mode: Character Reception

Example: 8-bit, parity enabled 1 stop

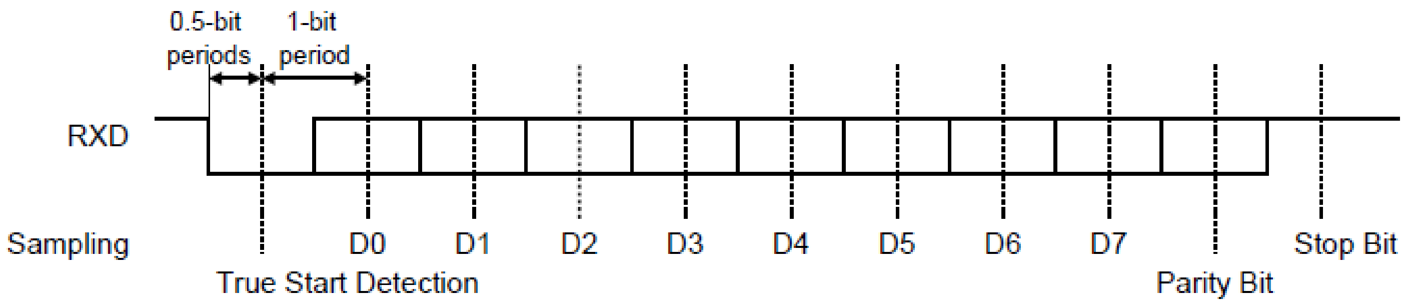
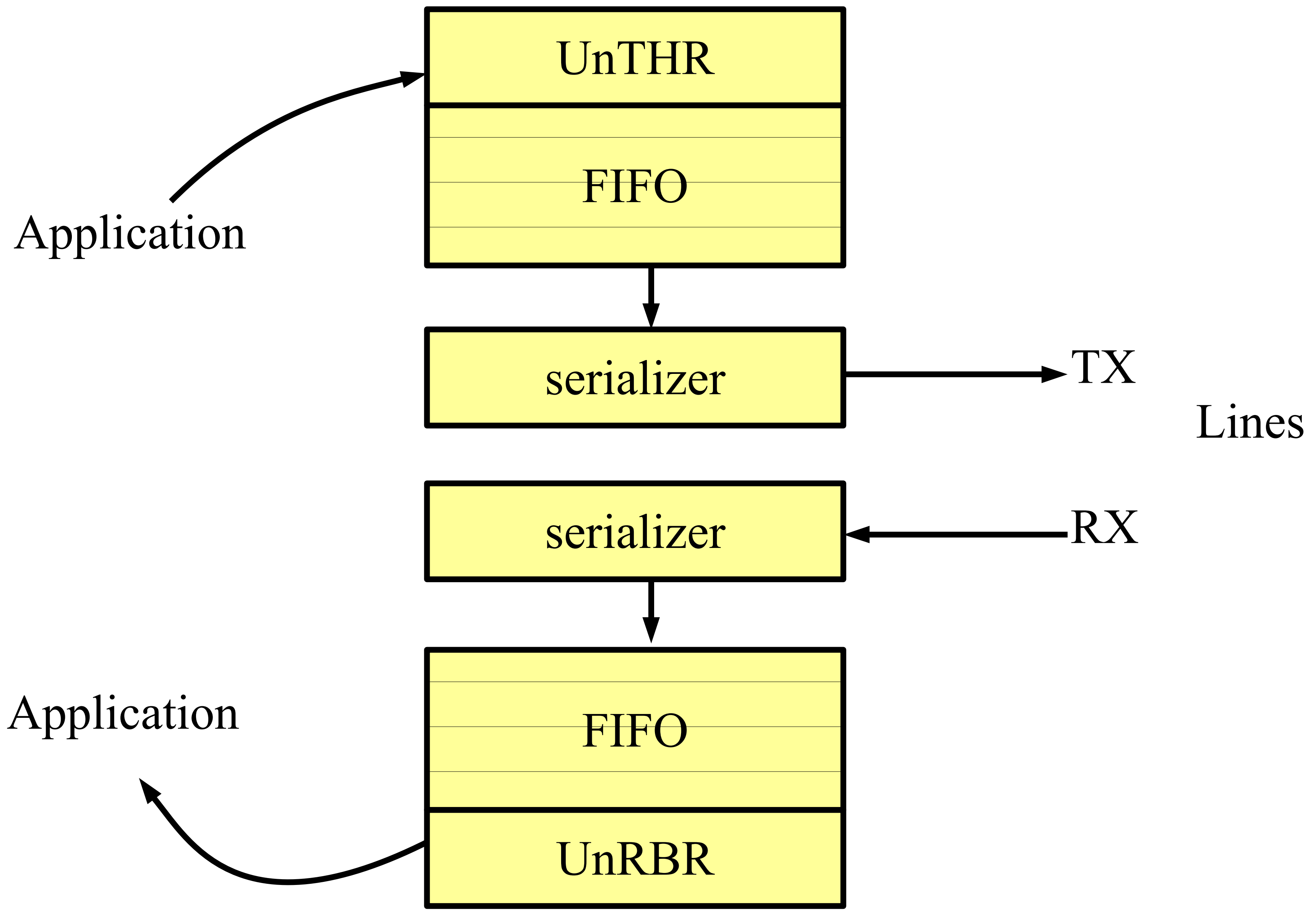


Diagram of a serial interface



LPC2478 UART0/1/2/3 register overview

- 16 bytes FIFO for receiver and transmitter
  - Write and read from a unique register : Read Buffer Register (UnRBR) and Transmit Holding Register (UnTHR)
  - Trigger point 1, 4, 8, 14 for receiver
- Line Status Register (UnLSR) for status information
  - Data received or transmitted, error on received data, ...
- Control
  - UARTn Line Control Register (UnLCR) : Number of bits, stop bit, parity enable , parity type, divisor latch access(DLAB)
  - FIFO Control Register (UnFCR) to reset emitter or transmitter and to chose receiver's fifo trigger

LPC2478 UART0/1/2/3 register overview

- Baud rate
  - Divisor latch (UnDLL and UnDLM)
  - Fractional Divider Register (UnFDR) for baud rate
- Interrupt
  - Interrupt Enable Register (UnIER) to allow interrupt source request to the system (data received, transmitter's fifo empty, received data error or time-out)
  - Interrupt Identification Register (UnIIR) to identify the interrupt source

UART registers

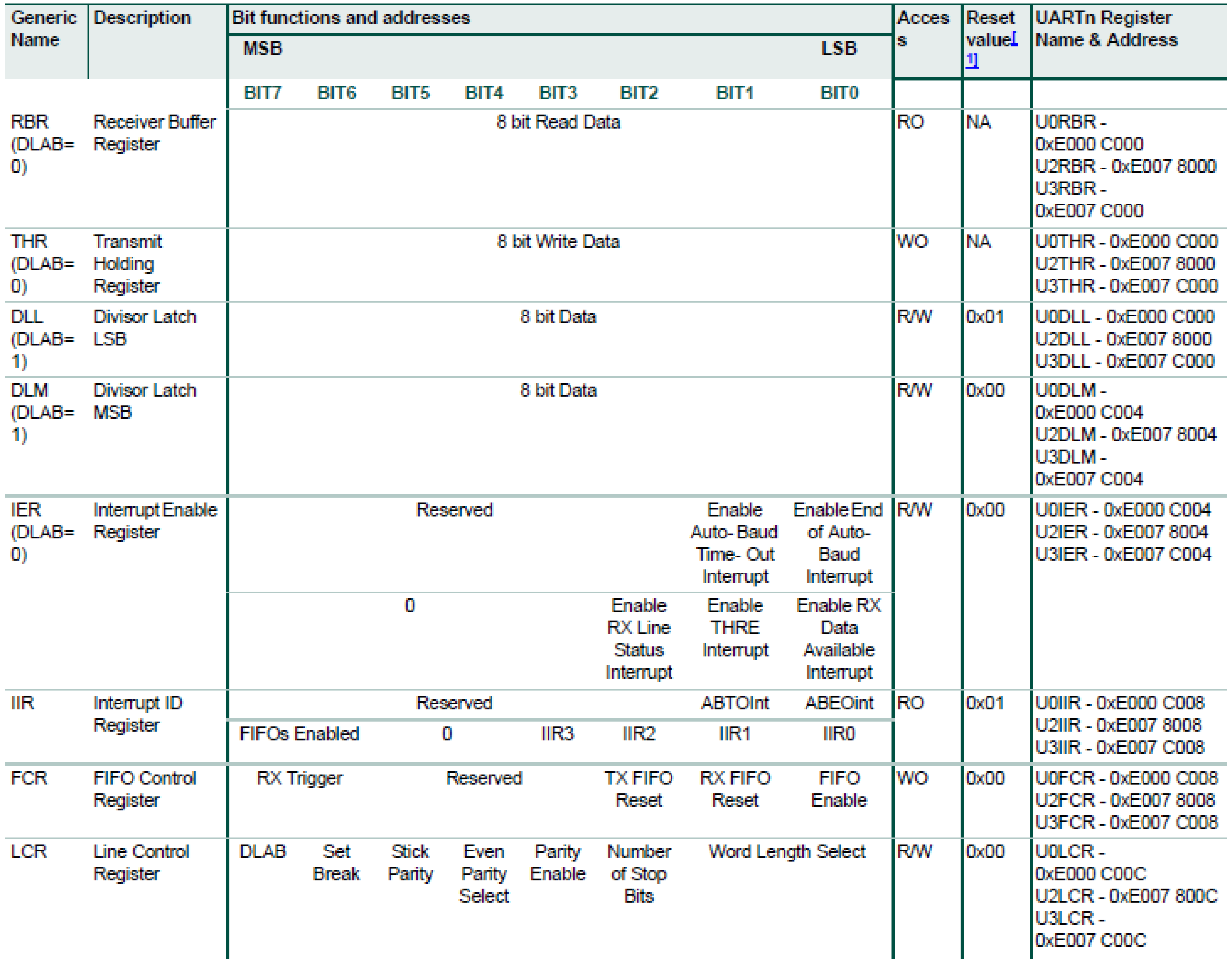


Table 377. UART Register Map

Generic Name	Description	Bit functions and addresses								Access	Reset value[1]	UARTn Register Name & Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
RBR (DLAB= 0)	Receiver Buffer Register	8 bit Read Data								RO	NA	U0RBR - 0xE000 C000 U2RBR - 0xE007 8000 U3RBR - 0xE007 C000
THR (DLAB= 0)	Transmit Holding Register	8 bit Write Data								WO	NA	U0THR - 0xE000 C000 U2THR - 0xE007 8000 U3THR - 0xE007 C000
DLL (DLAB= 1)	Divisor Latch LSB	8 bit Data								R/W	0x01	U0DLL - 0xE000 C000 U2DLL - 0xE007 8000 U3DLL - 0xE007 C000
DLM (DLAB= 1)	Divisor Latch MSB	8 bit Data								R/W	0x00	U0DLM - 0xE000 C004 U2DLM - 0xE007 8004 U3DLM - 0xE007 C004
IER (DLAB= 0)	Interrupt Enable Register	Reserved						Enable Auto- Baud Time- Out Interrupt	Enable End of Auto- Baud Interrupt	R/W	0x00	U0IER - 0xE000 C004 U2IER - 0xE007 8004 U3IER - 0xE007 C004
		0					Enable RX Line Status Interrupt	Enable THRE Interrupt	Enable RX Data Available Interrupt			
IIR	Interrupt ID Register	Reserved						ABTOInt	ABEOint	RO	0x01	U0IIR - 0xE000 C008 U2IIR - 0xE007 8008 U3IIR - 0xE007 C008
		FIFOs Enabled		0		IIR3	IIR2	IIR1	IIR0			
FCR	FIFO Control Register	RX Trigger		Reserved			TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	U0FCR - 0xE000 C008 U2FCR - 0xE007 8008 U3FCR - 0xE007 C008
LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0x00	U0LCR - 0xE000 C00C U2LCR - 0xE007 800C U3LCR - 0xE007 C00C

UART registers

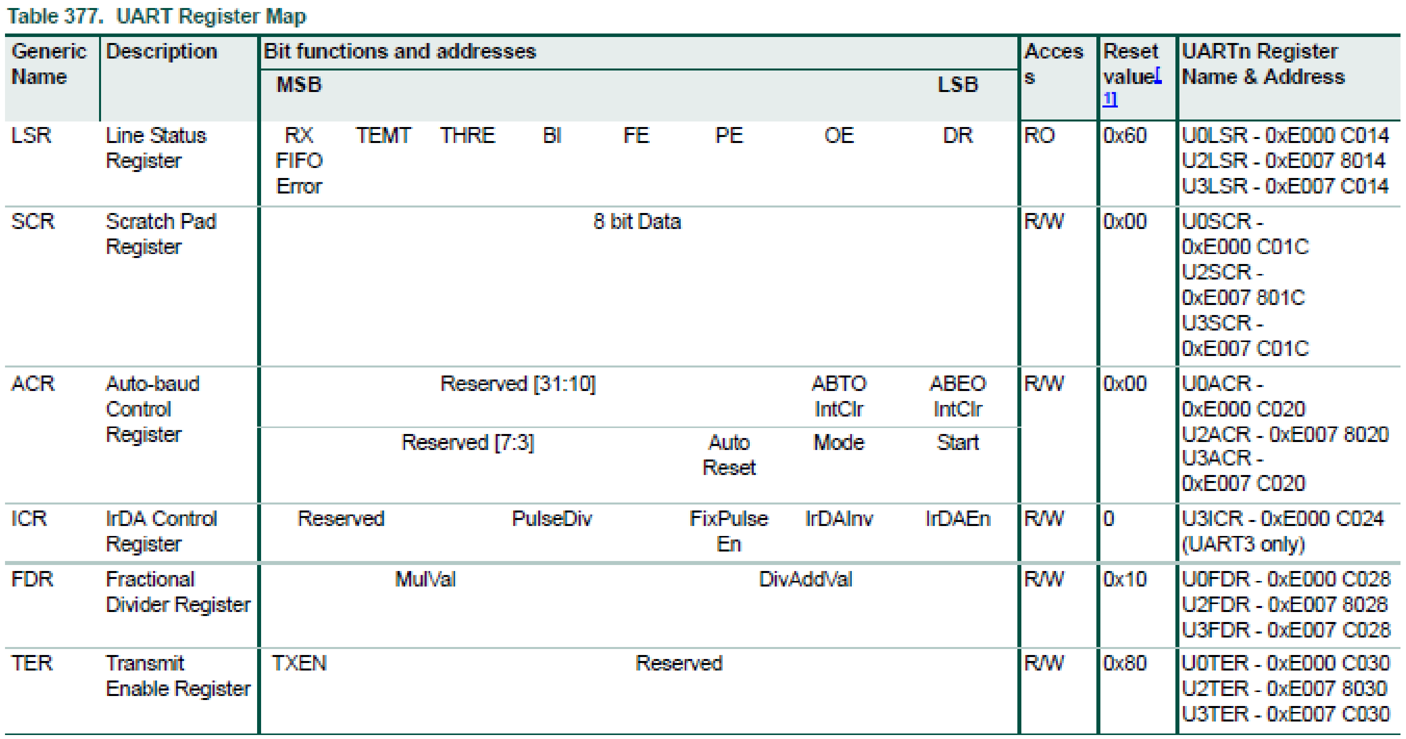
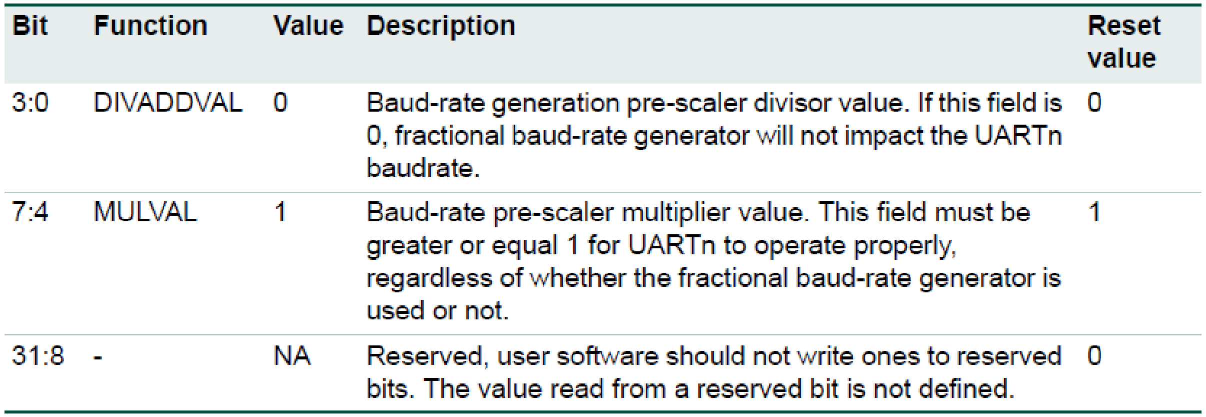


Table 377. UART Register Map

Generic Name	Description	Bit functions and addresses								Access	Reset value[1]	UARTn Register Name & Address
		MSB							LSB			
LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	U0LSR - 0xE000 C014 U2LSR - 0xE007 8014 U3LSR - 0xE007 C014
SCR	Scratch Pad Register	8 bit Data								R/W	0x00	U0SCR - 0xE000 C01C U2SCR - 0xE007 801C U3SCR - 0xE007 C01C
ACR	Auto-baud Control Register	Reserved [31:10]						ABTO IntClr	ABEO IntClr	R/W	0x00	U0ACR - 0xE000 C020 U2ACR - 0xE007 8020 U3ACR - 0xE007 C020
		Reserved [7:3]					Auto Reset	Mode	Start			
ICR	IrDA Control Register	Reserved			PulseDiv		FixPulse En	IrDAInv	IrDAEn	R/W	0	U3ICR - 0xE000 C024 (UART3 only)
FDR	Fractional Divider Register	MulVal				DivAddVal				R/W	0x10	U0FDR - 0xE000 C028 U2FDR - 0xE007 8028 U3FDR - 0xE007 C028
TER	Transmit Enable Register	TXEN	Reserved							R/W	0x80	U0TER - 0xE000 C030 U2TER - 0xE007 8030 U3TER - 0xE007 C030

Baud rate

- To allow a working serial line the baud rate must be set to match both emitter/transmitter device
- The baud rate is selected with 2 registers
  - UnDLM an UnDLL which are respectively at UnTHR and UnRBR address location when DLAB in UnLCR is set
  - Fractional Divider Register (UnFDR)



Bit	Function	Value	Description	Reset value
3:0	DIVADDVAL	0	Baud-rate generation pre-scaler divisor value. If this field is 0, fractional baud-rate generator will not impact the UARTn baudrate.	0
7:4	MULVAL	1	Baud-rate pre-scaler multiplier value. This field must be greater or equal 1 for UARTn to operate properly, regardless of whether the fractional baud-rate generator is used or not.	1
31:8	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

Example of configuration

- For a 8,1,N configuration and 115 200 baud
- Baud rate :
  - Pclk = 72/4 = 18MHz => DL = 18e6/16/115200 = 9,76
  - DL is calculated to have FR near 1,5
    - DL = 9,76/1,5 = 6 and FR = 1.628
  - FR is chosen from tab p393 => DIVADDVAL = 5; MULVAL = 8 and real FR = 1.625
  - Baud rate = 115 384 (diff = 0,1%)

```
UnFDR    = 0x85;              /* Fractional divider   */
UnLCR    = 0x83;              /* 8 bits, no Parity, 1 Stop bit DLAB = 1    */
UnDLL    = 6;                 /* 115200 Baud Rate @ 18 MHZ PCLK      */
UnDLM    = 0;                 /* High divisor latch = 0          */
UnLCR    = 0x03;              /* DLAB = 0 (& 8 bits, no Parity, 1 Stop bit) */
```

UART Status register UnLSR

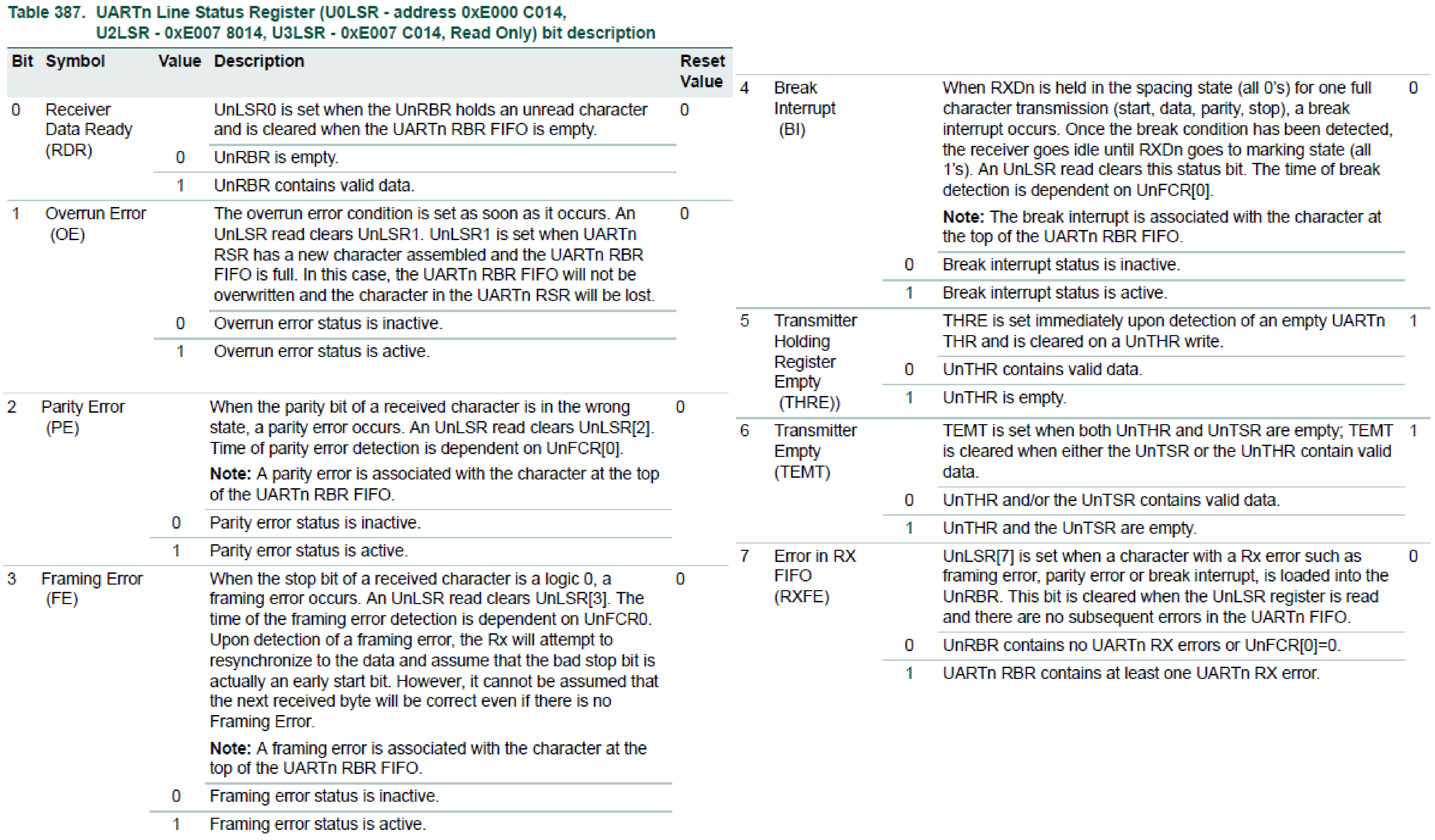


Table 387. UARTn Line Status Register (U0LSR - address 0xE000 C014, U2LSR - 0xE007 8014, U3LSR - 0xE007 C014, Read Only) bit description

Bit	Symbol	Value	Description	Reset Value
0	Receiver Data Ready (RDR)		UnLSR0 is set when the UnRBR holds an unread character and is cleared when the UARTn RBR FIFO is empty.	0
		0	UnRBR is empty.	
		1	UnRBR contains valid data.	
1	Overrun Error (OE)		The overrun error condition is set as soon as it occurs. An UnLSR read clears UnLSR1. UnLSR1 is set when UARTn RSR has a new character assembled and the UARTn RBR FIFO is full. In this case, the UARTn RBR FIFO will not be overwritten and the character in the UARTn RSR will be lost.	0
		0	Overrun error status is inactive.	
		1	Overrun error status is active.	
2	Parity Error (PE)		When the parity bit of a received character is in the wrong state, a parity error occurs. An UnLSR read clears UnLSR[2]. Time of parity error detection is dependent on UnFCR[0]. **Note:** A parity error is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Parity error status is inactive.	
		1	Parity error status is active.	
3	Framing Error (FE)		When the stop bit of a received character is a logic 0, a framing error occurs. An UnLSR read clears UnLSR[3]. The time of the framing error detection is dependent on UnFCR0. Upon detection of a framing error, the Rx will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error. **Note:** A framing error is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Framing error status is inactive.	
		1	Framing error status is active.	
4	Break Interrupt (BI)		When RXDn is held in the spacing state (all 0's) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXDn goes to marking state (all 1's). An UnLSR read clears this status bit. The time of break detection is dependent on UnFCR[0]. **Note:** The break interrupt is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Break interrupt status is inactive.	
		1	Break interrupt status is active.	
5	Transmitter Holding Register Empty (THRE))		THRE is set immediately upon detection of an empty UARTn THR and is cleared on a UnTHR write.	1
		0	UnTHR contains valid data.	
		1	UnTHR is empty.	
6	Transmitter Empty (TEMT)		TEMT is set when both UnTHR and UnTSR are empty; TEMT is cleared when either the UnTSR or the UnTHR contain valid data.	1
		0	UnTHR and/or the UnTSR contains valid data.	
		1	UnTHR and the UnTSR are empty.	
7	Error in RX FIFO (RXFE)		UnLSR[7] is set when a character with a Rx error such as framing error, parity error or break interrupt, is loaded into the UnRBR. This bit is cleared when the UnLSR register is read and there are no subsequent errors in the UARTn FIFO.	0
		0	UnRBR contains no UARTn RX errors or UnFCR[0]=0.	
		1	UARTn RBR contains at least one UARTn RX error.	

Using UART

- Configure the baud rate
- Configure bit number, parity, stop bit, ...
- Optionally reset emitter and transmitter and enable fifo
- Transmitting data
  - Write in UnTHR (up to 16 byte)
  - Wait for THE (transmitter's fifo empty) flag to set or for Transmitter empty (TEMT) flag signaling serializer empty (last byte completely transfered)
  - Write other data in UnTHR
  - ....
  - THE can be source of IRQ

Using UART

- Receiving data
  - RDR in UnLSR is set when an unread data is present in the RBR FIFO
  - Software waiting for data must poll RDR bit (wait for RDR to be set)
  - Software must read data from UnRBR until RDR is cleared
- Using interrupt
  - UnIER allow interrupt request on THRE, RBR, RX line status (Overrun error (OE), Parity Error (PE), break (BI)) and time out on reception
  - UnIIR allow software to identify interrupt source

Simple receiving/transmitting (polling)

```
void init_serial (void)  {
 PCONP   |= (1 <<3);                /* Enable UART0 power     */
 PCLKSEL0 &= 0xFFFFFF3F;            /*Pclock uart0 = Cclock/4 */
 PINSEL0 &= ~0x000000F0;
 PINSEL0 |=  0x00000050;            /* Enable TxD0 and RxD0  */

  U0FDR    = 0x85;                  /* Fractional divider      */
  U0LCR    = 0x83;                  /* 8 bits, no Parity, 1 Stop bit , DLAB = 1     */
  U0DLL    = 6;                     /* 115200 Baud Rate @ 18 MHZ PCLK */
  U0DLM    = 0;                     /* High divisor latch = 0          */
  U0LCR    = 0x03;                  /* 8 bits, no Parity, 1 Stop bit , DLAB = 0 */
}

int sendchar (int ch)  {            /* Write character to Serial Port    */
  while (!(U2LSR & 0x20));          /* Wait for transmitt buffer empty    */
  return (U2THR = ch);
}

int getkey (void)  {                /* Read character from Serial Port   */
  while (!(U2LSR & 0x01));          /* Wait for receive buffer not empty    */
  return (U2RBR);
}
```

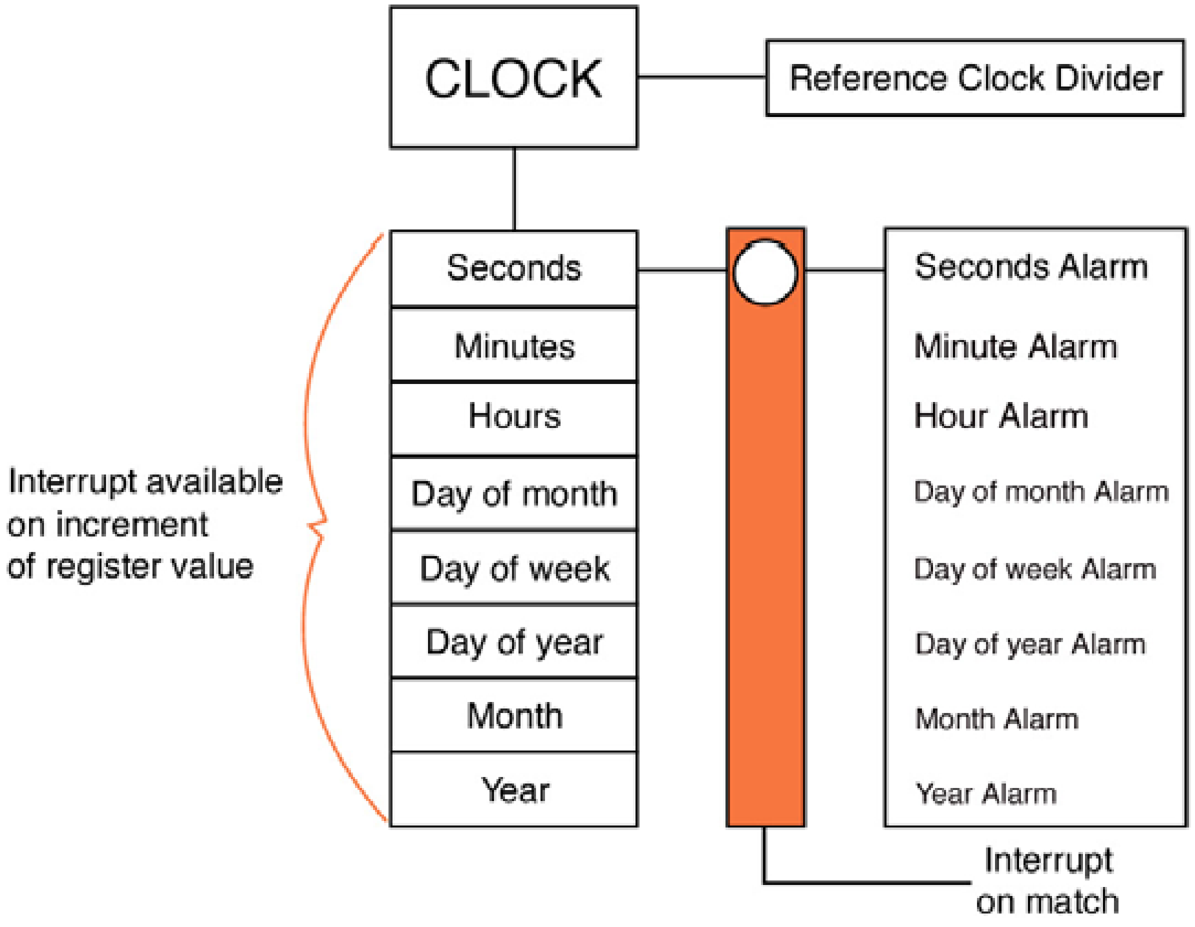
Real time clock

- The RTC is a set of counters for measuring time when system power is on, and optionally when it is off
- RTC can be clocked by a separate 32.768 kHz oscillator or by a programmable prescaler divider based on PCLK
- RTC and battery SRAM have a separate power domain supplying 3.3V to the Vbat pin
- Provides Seconds, Minutes, Hours, Day of Month, Month, Year, Day of Week, and Day of Year.

RTC Interrupt

- An alarm output pin is included to assist in waking up when the chip has had power removed to all functions except the RTC and Battery RAM.
- Periodic interrupts can be generated from increments of any field of the time registers, and selected fractional second values.
  - This enhancement enables the RTC to be used as a System Timer.
- The alarm registers allow the user to specify a date and time for an interrupt to be generated

RTC Interrupt



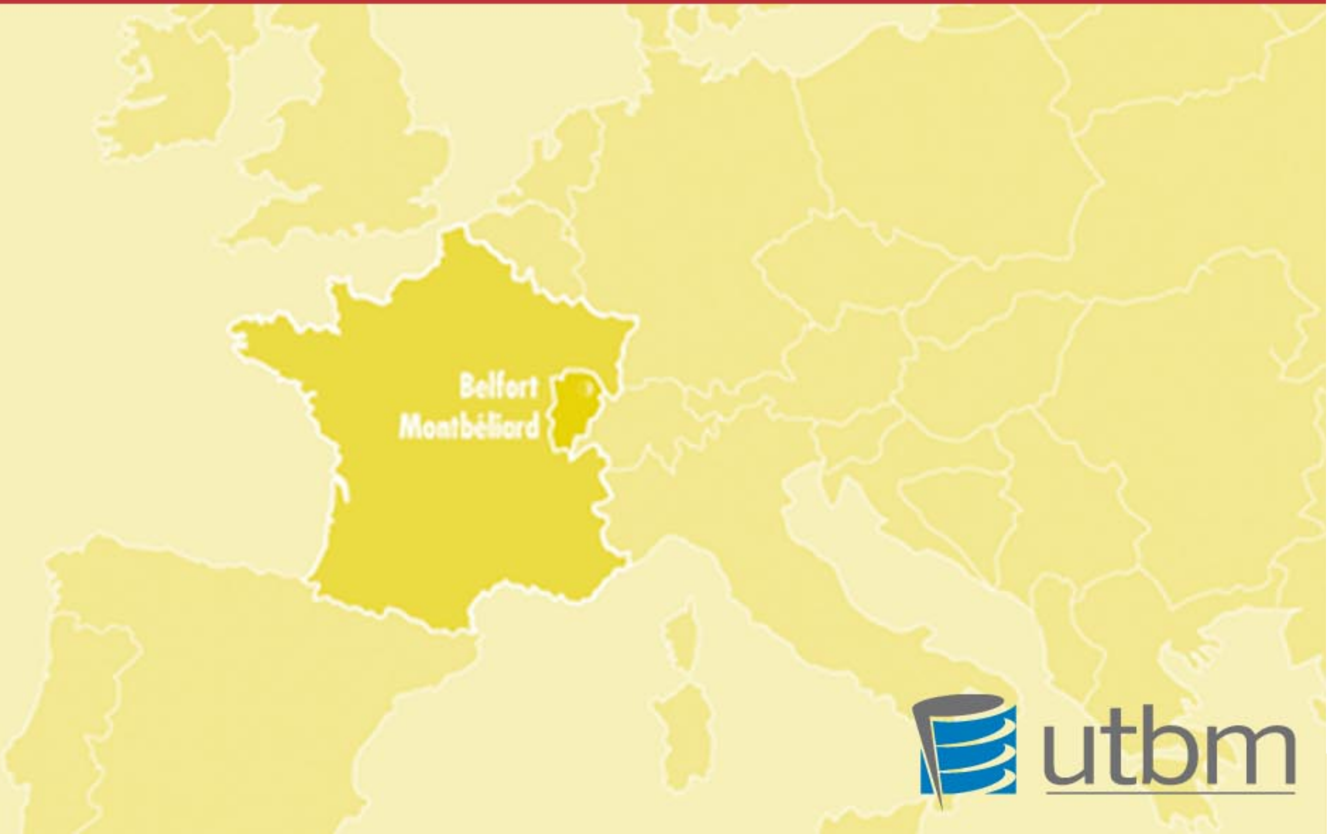
Watchdog

- Provide a method of recovering control of a crashed program
- Timer that can produce
  - Interrupt
  - Reset
- Watchdog timer must be “feeded” (reloaded) within a predetermined amount of time
  - From few µsec to few minutes

Other Interface

- PWM : Pulse Width Modulation
- I2S-bus : inter integrated circuit sound interface
- SSP : Synchronous Serial Peripheral
- ADC and DAC : Analog/digital conversion
- SD-MMC Card Interface
- CAN : Controller Area Network
- Ethernet
- USB host and device
- LCD controller

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD



utbm