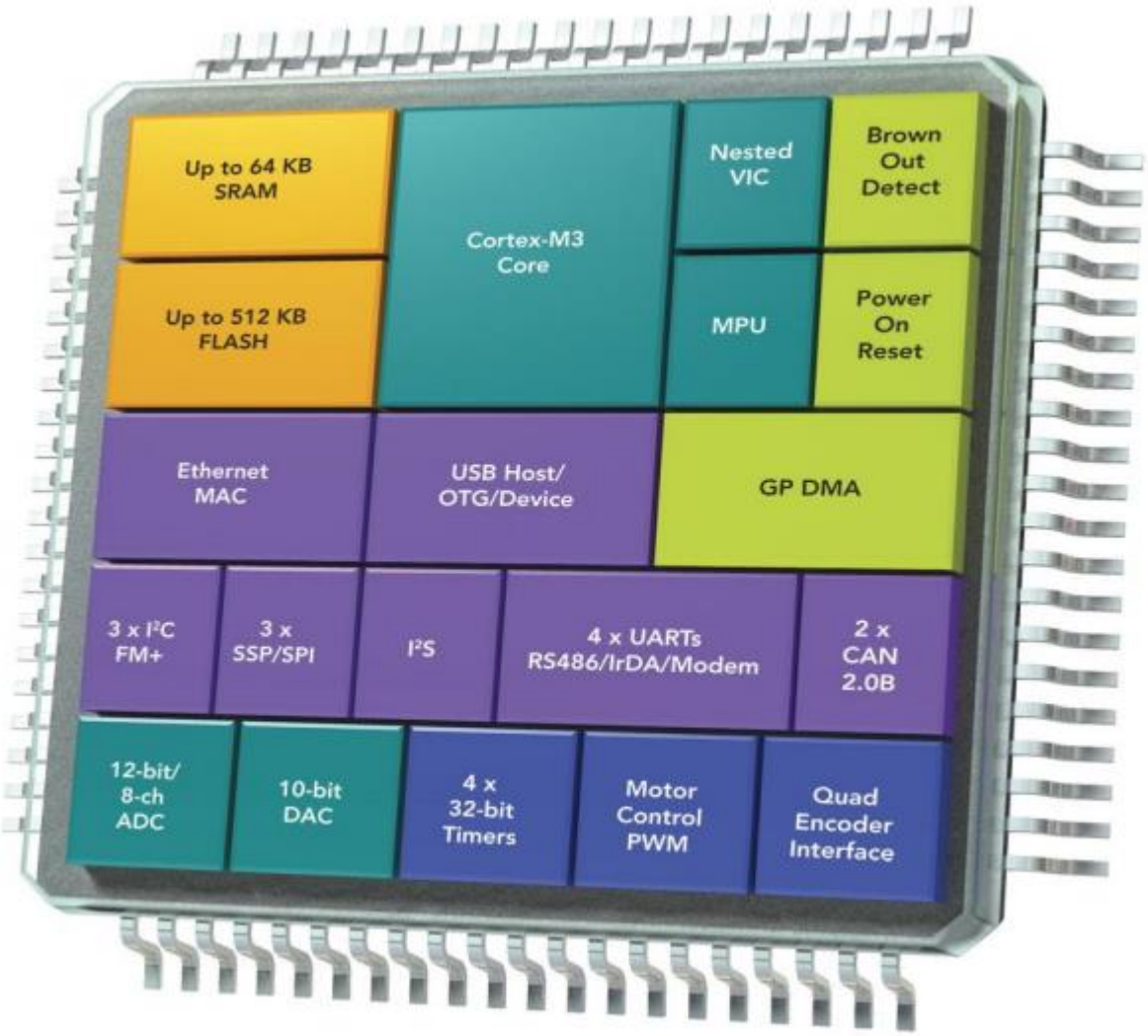
Microcontroller Peripherals: LPC1768

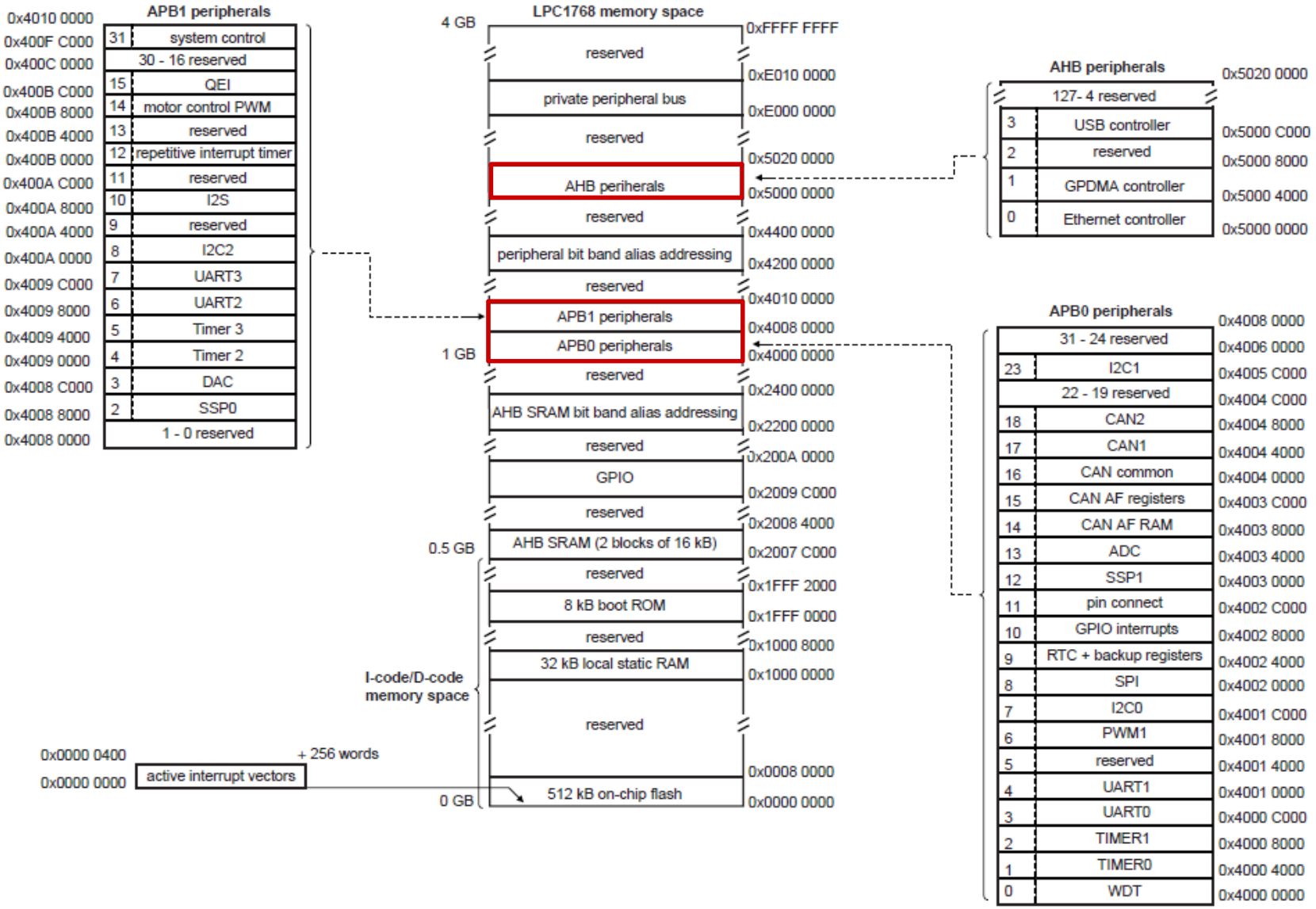
Index

- **2.1.** Block diagram. General Purpose Input/Output (GPIO).
- **2.2.** Timers: Timer 0/1/2/3. Pulse Width Modulator (PWM). Real-Time Clock (RTC). Watchdog Timer (WDT). Quadrature Encoder Interface (QEI).
- **2.3.** Analogs I/O: Analog-to-Digital Converter (ADC), Digital-to-Analog Converter (DAC).
- **2.4.** Serial Interfaces: UARTs, I2C, SPI, I2S, USB, CAN.
- **2.5.** General Purpose DMA (GPDMA).

2.1. LPC17xx: Block Diagram



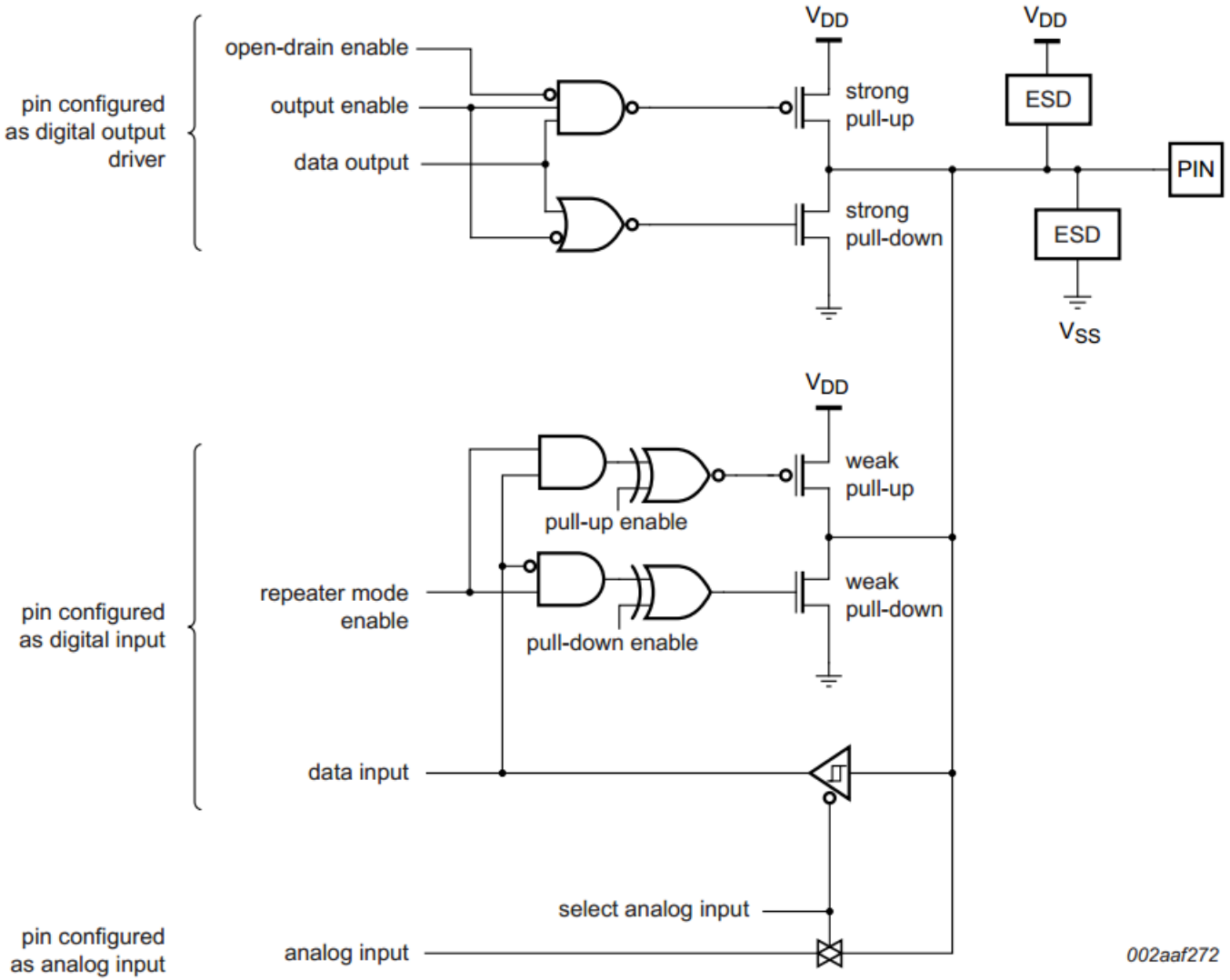
2.1. Peripherals Map



2.1. GPIOs: General Purpose I/O

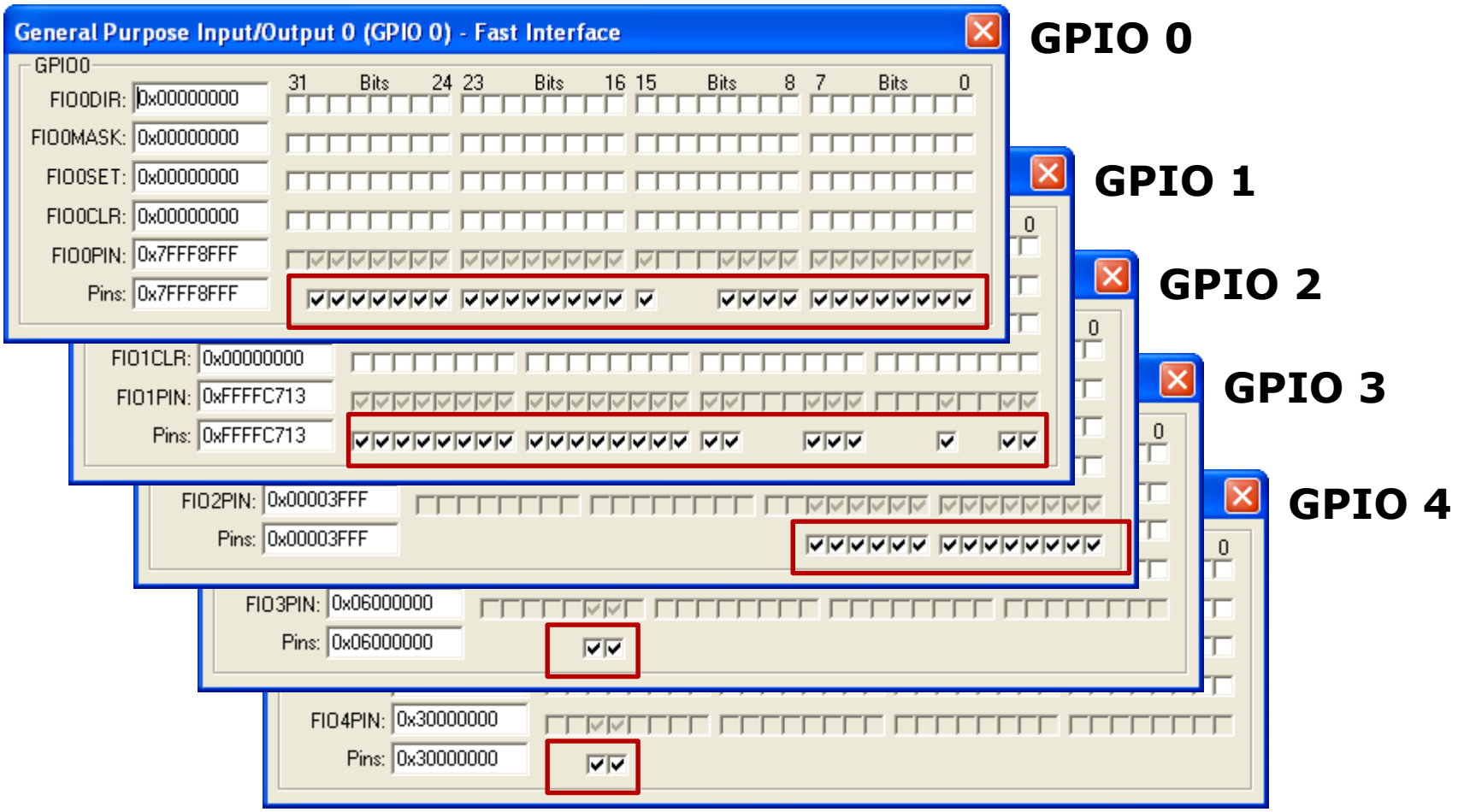
- **70** High Speed **GPIOs** (LQFP100) and 52 High Speed GPIOs (LQFP80).
- All pins have **configurable pull-ups**, **pull-downs**, or **neither**.
  - Through PINMODE and PINMODE_OD registers.
- All pins configurable as **open-drain**.
- GPIO registers are located on a **peripheral AHB bus** for fast I/O timing.
- **GPIO registers are accessible by the GPDMA**.
- External interrupt inputs:
  - 46 edge-sensitive interrupt inputs (**P0** and **P2**).
  - 4 level or edge-sensitive external interrupt inputs (**EINT0**…**3**).
  - Can optionally wake the processor from Power-down.

2.1. GPIOs: General Purpose I/O



2.1. GPIOs: General Purpose I/O

- Peripherals->GPIO Fast Interface



2.1. GPIO: Registers

- Registers to control GPIO functionality through Fast Interface AHB Bus
  - **FIODIR**: Fast GPIO Port Direction control register (0=in, 1=out). This register individually controls the direction of each port pin.
  - **FIOMASK**: Fast Mask register for port.

    Writes, sets, clears, and reads (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN) to alter or return **only the bits enabled by 0 in this register**.
  - **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.

    Writing places corresponding values in all bits enabled by 0 in FIOMASK.
  - **FIOSET**: Fast Port Output Set register using FIOMASK.

    Writing 1s produces highs at the corresponding port pins. Reading this register returns the current contents of the port output register.

    Only bits enabled by 0 in FIOMASK can be altered
  - **FIOCLR**: Fast Port Output Clear register using FIOMASK.

    Writing 1s produces lows at the corresponding port pins.

    Only bits enabled by 0 in FIOMASK can be altered

2.1. GPIO: Setting direction (I/O)

Pin direction setting:

- **Five** registers, **FIOnDIR** (n=0-4).
- In *Keil*, **LPC_GPIOn->FIODIR**[31:0] control the pin input/output where **n** is the pin group (0-4).
- To set a pin **as output**, set the corresponding bit to **1**.
- To set a pin **as input**, set the corresponding bit to **0**. By default, all pins are set as input (all bits are 0).

- Examples:
  - **LPC_GPIO1->FIODIR |= (1<<3);** // Set **P1.3** as output
  - **LPC_GPIO1->FIODIR |= (0xFF<<16);** // Set P1 (bits **16-23**) as output
  - **LPC_GPIO1->FIODIR &= ~(0xFF<<16);** // Set P1 (bits **16-23**) as input

2.1. GPIO: Read/write digital I/O

- **Pin digital read/write**:
  - **Five** registers, **FIOnPIN** (n=0-4).
  - In *Keil*, **LPC_GPIOn->FIOPIN** is used to read/write in GPIO**n**.

- Examples:
  - Write:
    - **LPC_GPIO1->FIOPIN |= (1<<3);** //set **P1.3** to digital 1
    - **LPC_GPIO1->FIOPIN |= (0xFF<<16);** // Set P1 (bits **16-23**) to digital 1
    - **LPC_GPIO1->FIOPIN &=~ (1<<3);** //set **P1.3** to digital 0
    - **LPC_GPIO1->FIOPIN &= ~(0xFF<<16);** // Set P1 (bits **16-23**) to digital 0
  - Read:
    - **if(LPC_GPIO1->FIOPIN &(1<<3));** //**P1.3 ?**

    - **if(((LPC_GPIO1->FIOPIN &(3<<20))>>20)==0);** //**P1.20-P1.21=00 ?**
    - **if(((LPC_GPIO1->FIOPIN &(3<<20))>>20)==1);** //**P1.20-P1.21=01 ?**
    - **if(((LPC_GPIO1->FIOPIN &(3<<20))>>20)==2);** //**P1.20-P1.21=10 ?**
    - **if(((LPC_GPIO1->FIOPIN &(3<<20))>>20)==3);** //**P1.20-P1.21=11 ?**

    - **switch((LPC_GPIO1->FIOPIN &(3<<20))>>20);** //**P1.20-P1.21 ?**

2.1. GPIO: Set/clear digital I/O

- **Fast Set/Clear digital pins**:
  - In *Keil*, **LPC_GPIOn->FIOSET** is used to turn a pin to **HIGH** (page 122/835).
  - To turn a pin to digital 1 (high), set the corresponding bit of LPC_GPIOn->FIOSET to 1.
    - If we set LPC_GPIOn->FIOSET **bit to 0**, there is **no effect**.
  - In *Keil*, **LPC_GPIOn->FIOCLR** is used to turn a pin to **LOW**.
  - To turn a pin to digital 0 (low), set the corresponding bit of LPC_GPIOn->FIOCLR to 1.
    - If we set LPC_GPIOn->FIOCLR **bit to 0**, there is **no effect**.
- Examples:
  - **LPC_GPIO0->FIOSET |= (1<<3);** //set **P1.3** to digital 1
  - **LPC_GPIO1->FIOSET |= (0xFF<<16);** // Set P1 (bits **16-23**) to digital 1
  - **LPC_GPIO0->FIOCLR |= (1<<3);** //set **P1.3** to digital 0
  - **LPC_GPIO1->FIOCLR |= (0xFF<<16);** // Set P1 (bits **16-23**) to digital 0

2.1. GPIOs: General Interrupt inputs

- Registers to control **GPIO (P0** and **P2) interrupt inputs functionality** through **EINT3** ISR:

Table 102. GPIO interrupt register map

Generic Name	Description	Access	Reset value[1]	PORTn Register Name & Address
IntEnR	GPIO Interrupt Enable for Rising edge.	R/W	0	IO0IntEnR - 0x4002 8090 IO2IntEnR - 0x4002 80B0
IntEnF	GPIO Interrupt Enable for Falling edge.	R/W	0	IO0IntEnR - 0x4002 8094 IO2IntEnR - 0x4002 80B4
IntStatR	GPIO Interrupt Status for Rising edge.	RO	0	IO0IntStatR - 0x4002 8084 IO2IntStatR - 0x4002 80A4
IntStatF	GPIO Interrupt Status for Falling edge.	RO	0	IO0IntStatF - 0x4002 8088 IO2IntStatF - 0x4002 80A8
IntClr	GPIO Interrupt Clear.	WO	0	IO0IntClr - 0x4002 808C IO2IntClr - 0x4002 80AC
IntStatus	GPIO overall Interrupt Status.	RO	0	IOIntStatus - 0x4002 8080

- IntStatus: GPIO overall Interrupt Status. Read-only register indicates the presence of interrupt pending on all of the GPIO ports. Only one status bit per port is required.

2.1. Config. Periph. (Software Steps)

- For using any of the LPC1768 peripherals, the general steps to be followed are:
  - 1. **Power Up** the peripheral to be used.
  - 3. Connect necessary pins using **Pin Connect Block**.
  - 2. Set the **Clock Rate** for the peripheral (No necessary)
    - Default: **cclk/4** (Reset).
  - 4. **Initialize** the registers of the peripheral.
  - 5. Config. exception **priority level** (in NVIC).
  - 6. **Enable** exception number (in NVIC).

2.2. TIMERs

- LPC1768 has 8 x 32 bits timers:
  - General use: **Timers 0**, **1**, **2**, **3**, with (at least) 2 Capture Inputs + 4 Match Outputs.
  - 2 for **PWM** generation.
  - **RTC** and **WDT**.
  - **RIT** and **SysTick**
- **Pinned out** with a choice of multiple pins for each.
- With a **programmable 32-bit Preescaler.**
- Can generate IRQs.
- Can generate sampling frequency of ADC.
- GPDMA controller support.
  - Allows for timed memory-to-memory transfers.

2.2. TIMERs 0/1/2/3

- A minimum of **two Capture inputs** and **four Match outputs** are pinned out for all four timers, with a choice of multiple pins for each.
- A 32-bit Timer/Counter with a programmable 32-bit Prescaler Counter or Timer operation.
- Up to **two 32-bit capture channels** per timer, that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- **Four 32-bit match registers that allow:**
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Up to **four external outputs** corresponding to match registers, with the following capabilities:
  - Set low, high, toggle or do nothing on match.

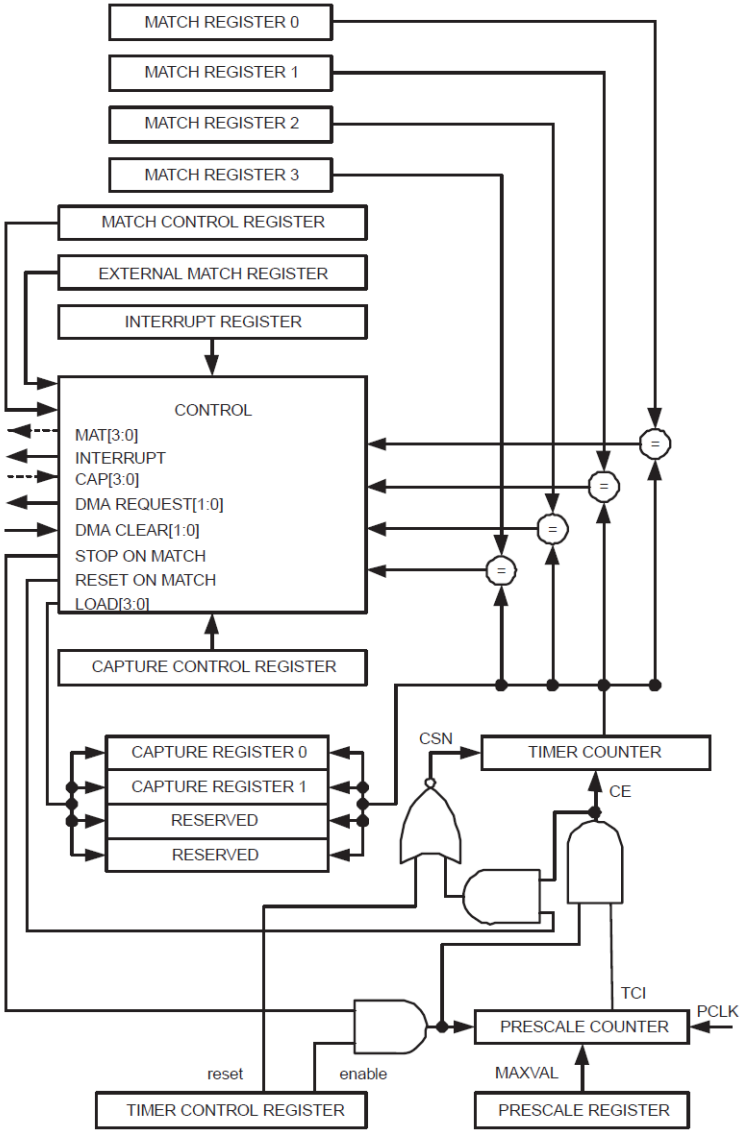
2.2. TIMERs 0/1/2/3

Input Capture (CAPn.0, CAP2.1)

- To measure external signals' timings (frequency, pulse width, etc.)
- Up to **2 32-bit capture channels per timer**, can take a snapshot of the timer value with input signal transitions
- **Capture event may also generate an interrupt**

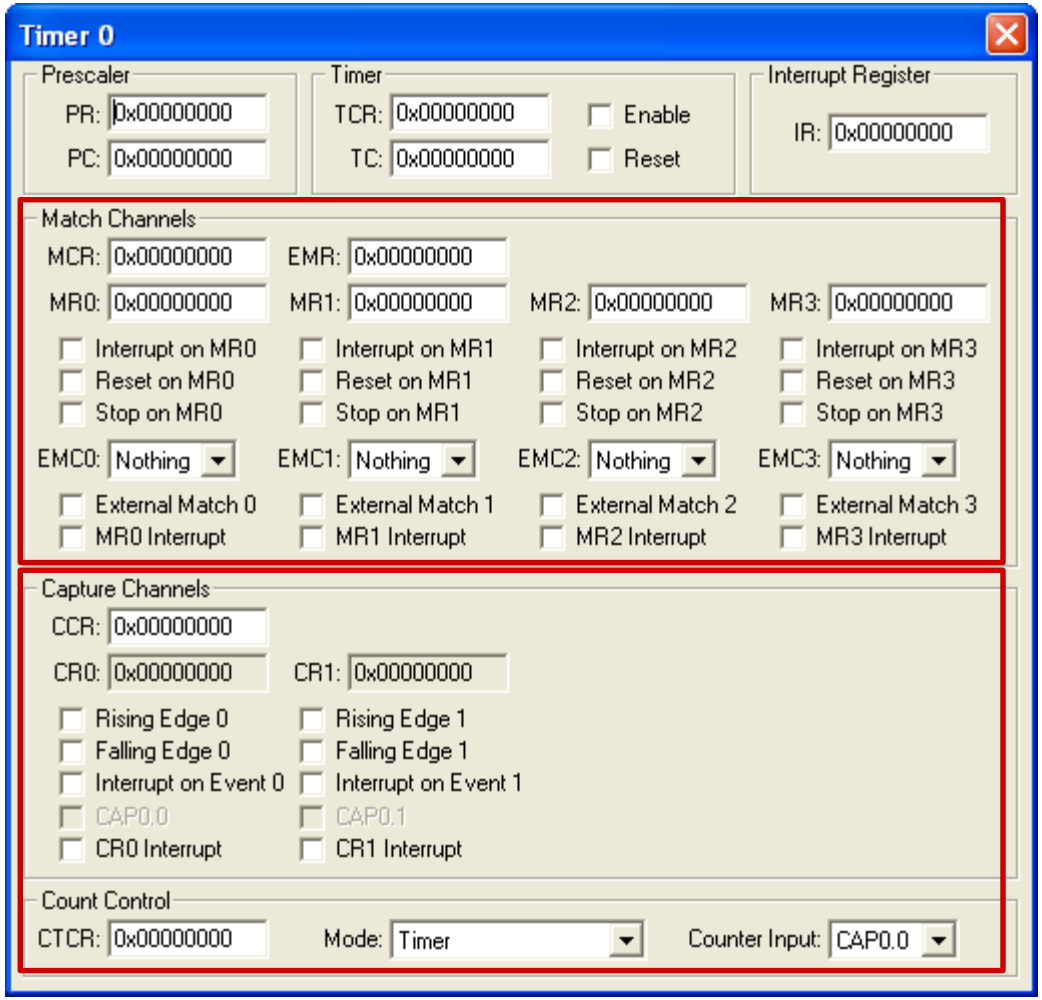
Output Compare (MATn.0 … MATn.3)

- **To generate signals (high precision, different types such as PWM)**
- 4 32-bit match registers that allow:
  - Continuous operation
  - Stop timer on match
  - Reset timer on match
- **Match may also generate an interrupt**
- Up to 4 external outputs corresponding to match registers, with the following capabilities:
  - Set low on match.
  - Set high on match.
  - Toggle on match.
  - Do nothing on match.



2.2. TIMERs: Keil ARM_MDK Debug Windows

- **Pheriferals -> Timers -> Timer 0-1-2-3**



2.2. Timer0 (code example)

- Go to page 490 in the manual and follow the links provided there
- 1. Powering the Timer0 (Table 46, page 63)

```
LPC_SC->PCONP |= 1 << 1; //Power up Timer 0
```

- 2. After enabling it you need to give the clock for Timer0 (Table 40, page 56. For Timer2/3 refer Table 41, page 57) Select clock as per your requirements.

```
LPC_SC->PCLKSEL0 |= 1 << 3; // Clock for timer = CCLK/2
```

- 3. Configuring the interrupt LPC1768 contains 4 match registers for each timer. These match registers can be used to reset the timer, generate interrupt, stop the timer, to generate timing signal on an external pin. Now we shall use Timer0 Match0 register to generate the interrupt.

```
LPC_TIM0->MR0 = Ftick/Fint; // LED blinking frequency based on the clock frequency timer
```

2.2. Timer0 (code example)

- 4. Configure it as to interrupt when the timer count matches the Match0 (Table 429, page 496)

  LPC_TIM0->MCR |= 1 << 0; // Interrupt on Match0 compare

- 5. Reset the timer (Table 427, page 494)

  LPC_TIM0->TCR |= 1 << 1; // Reset Timer0

- 6. Enable Timer interrupts

  NVIC_EnableIRQ(TIMER0_IRQn); // Enable timer interrupt

- 5. And finally start the timer (Table 427, page 494)

  LPC_TIM0->TCR |= 1 << 0; // Start timer

2.2. Timer0 (code example)

Interrupt function:

```
void TIMER0_IRQHandler (void)
{
    if((LPC_TIM0->IR & 0x01) == 0x01) //  MR0 interrupt ?
      {
       LPC_TIM0->IR |= 1 << 0; // Clear MR0 interrupt flag
       LPC_GPIO1->FIOPIN ^= 1 << 29; // Toggle the LED
      }

     if((LPC_TIM0->IR & 0x02) == 0x02) //  MR1 interrupt ?

    .....

    .....
}
```

2.2. Frequency Measurement (I)

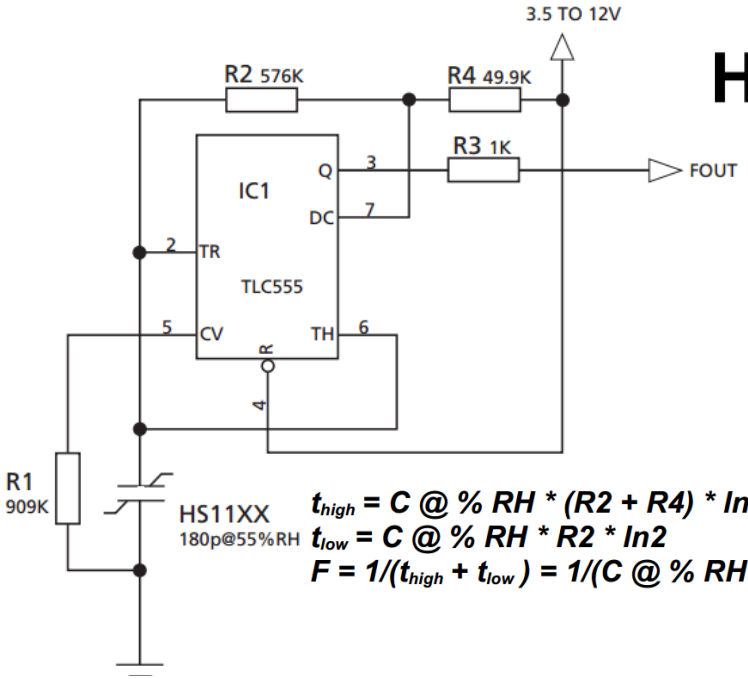
- Caudal meter:

- Working Voltage: 5 to 18VDC
- Max current draw: 15mA @ 5V
- Working Flow Rate: 1 to 30 Liters/Minute
- Working Temperature range: -25 to 80°C
- Working Humidity Range: 35%-80% RH
- Maximum water pressure: 2.0 MPa
- Output duty cycle: 50% +-10%
- Output rise time: 0.04us
- Output fall time: 0.18us
- Flow rate pulse characteristics: Frequency (Hz) = 7.5 * Flow rate (L/min)
- Pulses per Liter: 450
- Durability: minimum 300,000 cycles

- Interrupt function:

```
void TIMER0_IRQHandler (void) // Timer 0 y capture mode (CAP0.0)
{
      static uint32_t  temp, float frequency;
      LPC_TIM0->IR |= 1 << 4;                    // Clear CR0 flag interrupt
      frequency=Fpclk/(LPC_TIM0->CR0-temp);      // frequency in Hz.
      caudal=frecuency/7.5;                      // caudal (L/min).
      temp=LPC_TIM0->CR0;                         // update temp

}
```

2.2. Frequency Measurement (II)



HS1101 Relative Humidity Sensor

$t_{high} = C @ \%\ RH * (R2 + R4) * ln2$

$t_{low} = C @ \%\ RH * R2 * ln2$

$F = 1/(t_{high} + t_{low}) = 1/(C @ \%\ RH * (R4 + 2 * R2) * ln2)$

Typical Characteristics for Frequency Output Circuits

REFERENCE POINT AT 6660Hz FOR 55%RH / 25°C

RH	0	10	20	30	40	50	60	70	80	90	100
Frequency	7351	7224	7100	6976	6853	6728	6600	6468	6330	6186	6033

Typical for a 555 Cmos type. TLC555 (RH : Relative Humidity in %, F : Frequency in Hz)

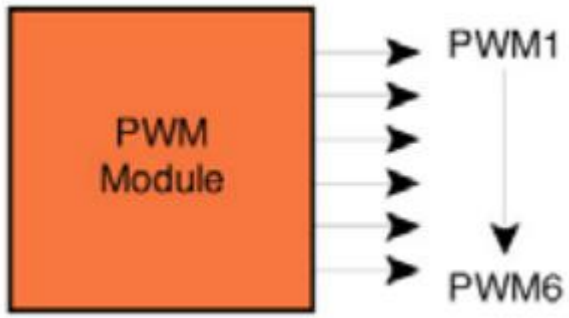
Polynomial response :

$$F_{mes(Hz)} = F55_{(Hz)}(1.1038 - 1.936810^{-3} * RH + 3.011410^{-6} * RH^2 - 3.440310^{-8} * RH^3)$$

2.2. Pulse Width Modulator (PWM)

- A 32-bit Timer/Counter with a programmable 32-bit prescaler. (may use the peripheral clock or one of the capture inputs as the clock source).

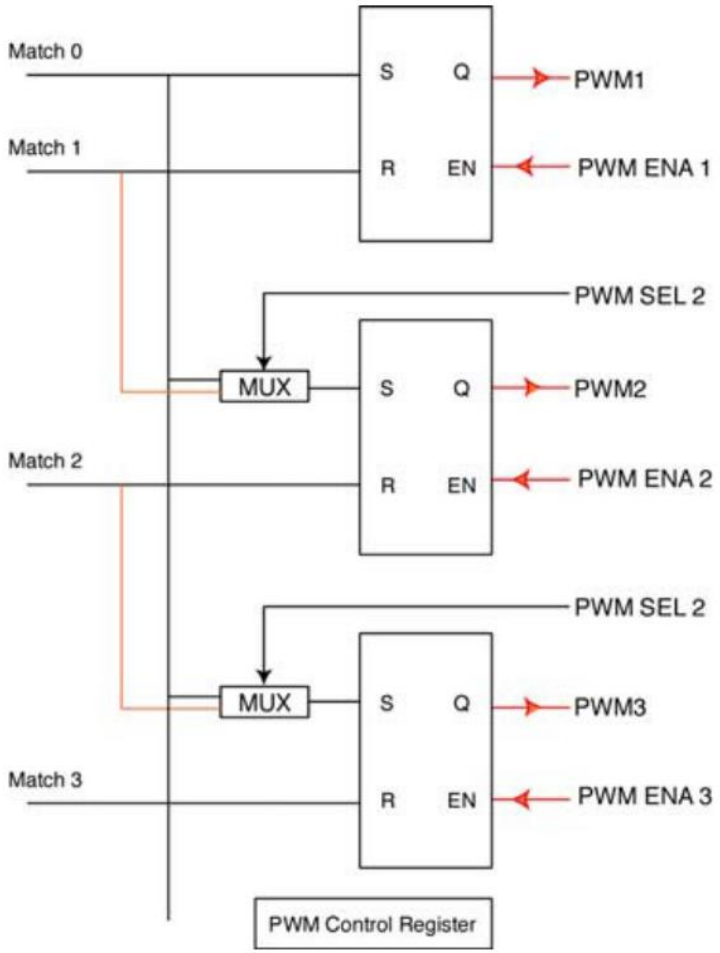
- Seven match registers allow up to **6 single edge** controlled or **3 double edge** controlled PWM outputs, or a mix of both types.



- The match registers also allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.

2.2. Pulse Width Modulator (PWM)

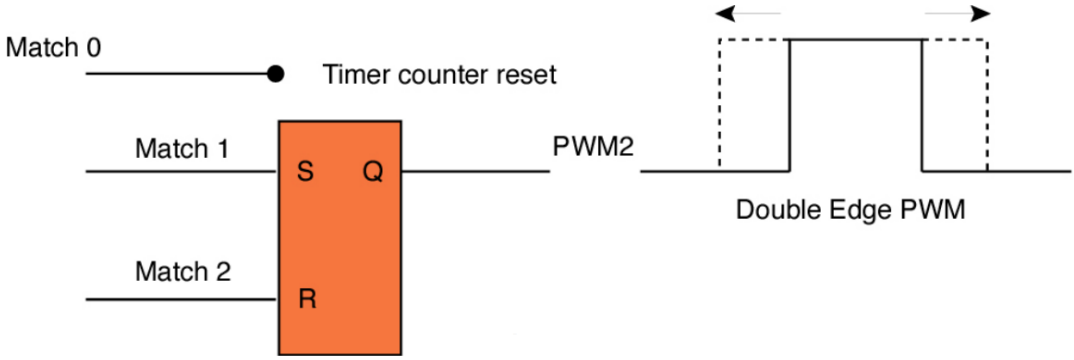
- Supports **single edge** controlled and/or **double edge** controlled PWM outputs.
  - **Single edge** controlled PWM outputs all go high at the beginning of each cycle unless the output is a constant low.
  - **Double edge** controlled PWM outputs can have either edge occur at any position within a cycle. This allows for both positive going and negative going pulses.



2.2. Pulse Width Modulator (PWM)

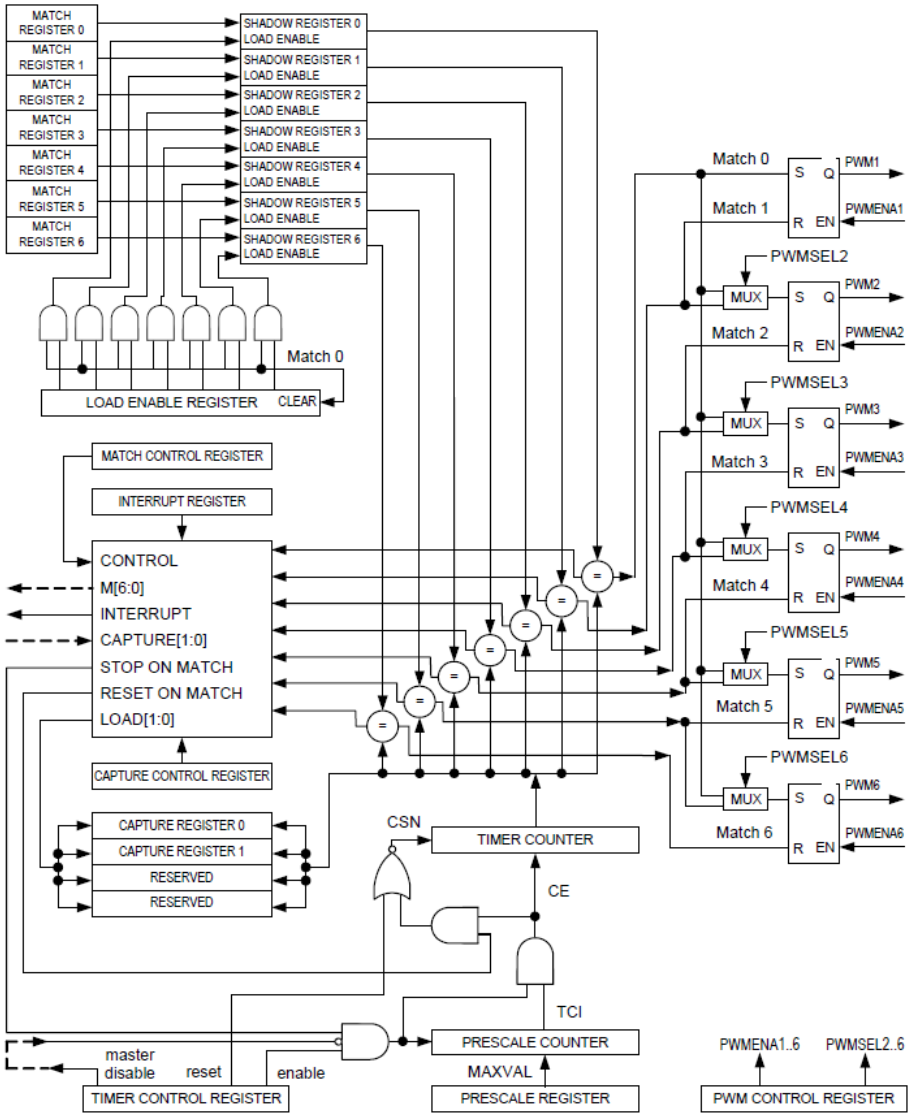
- Pulse period and width can be any number of timer counts.
  - This allows complete flexibility in the trade-off between resolution and repetition rate.
  - All PWM outputs will occur at the same repetition rate.

- **Double edge** controlled PWM outputs can be programmed to be either **positive or negative going pulses**.



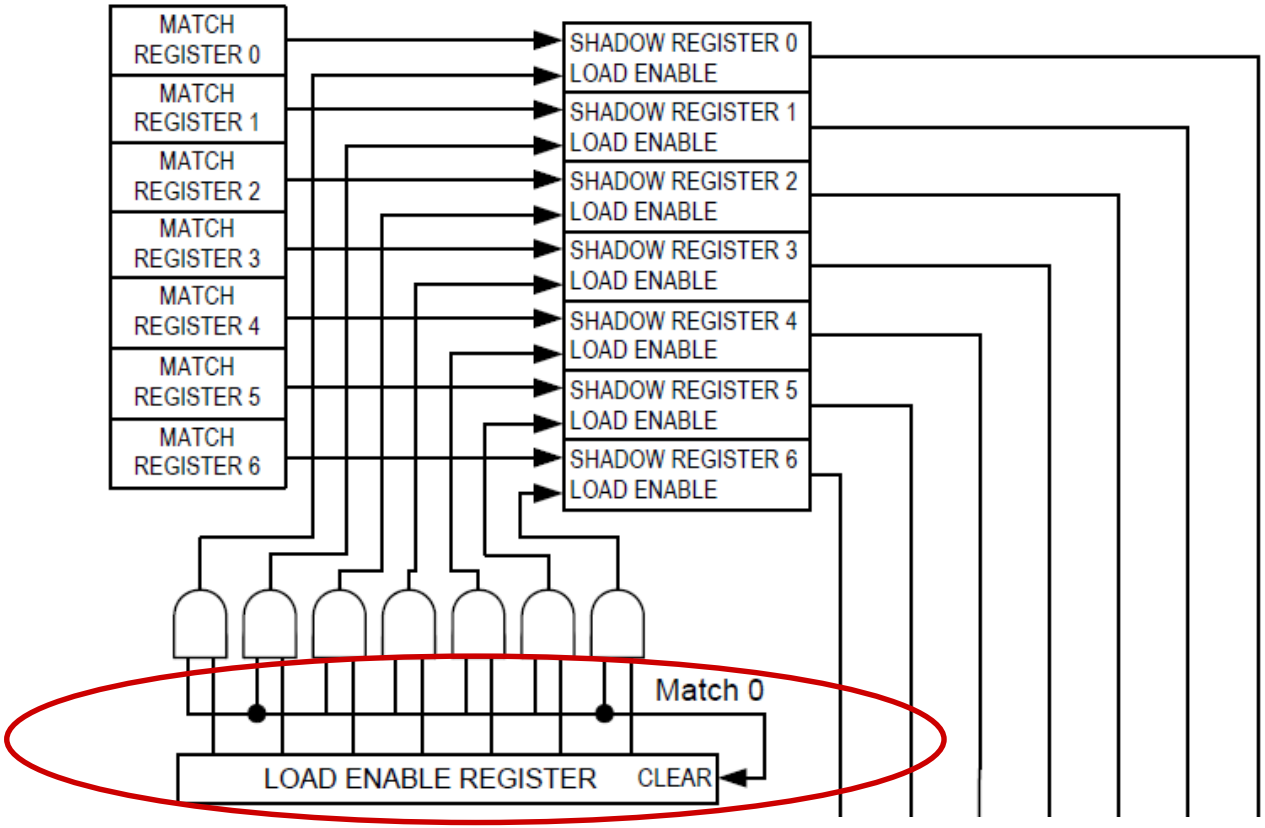
- Shadow latch mechanism.
  - Glitch-less operation.
  - Match register updates are synchronized with pulse outputs to prevent generation of erroneous pulses.
- May be used as a standard timer if the PWM mode is not enabled.

2.2. Pulse Width Modulator (PWM)



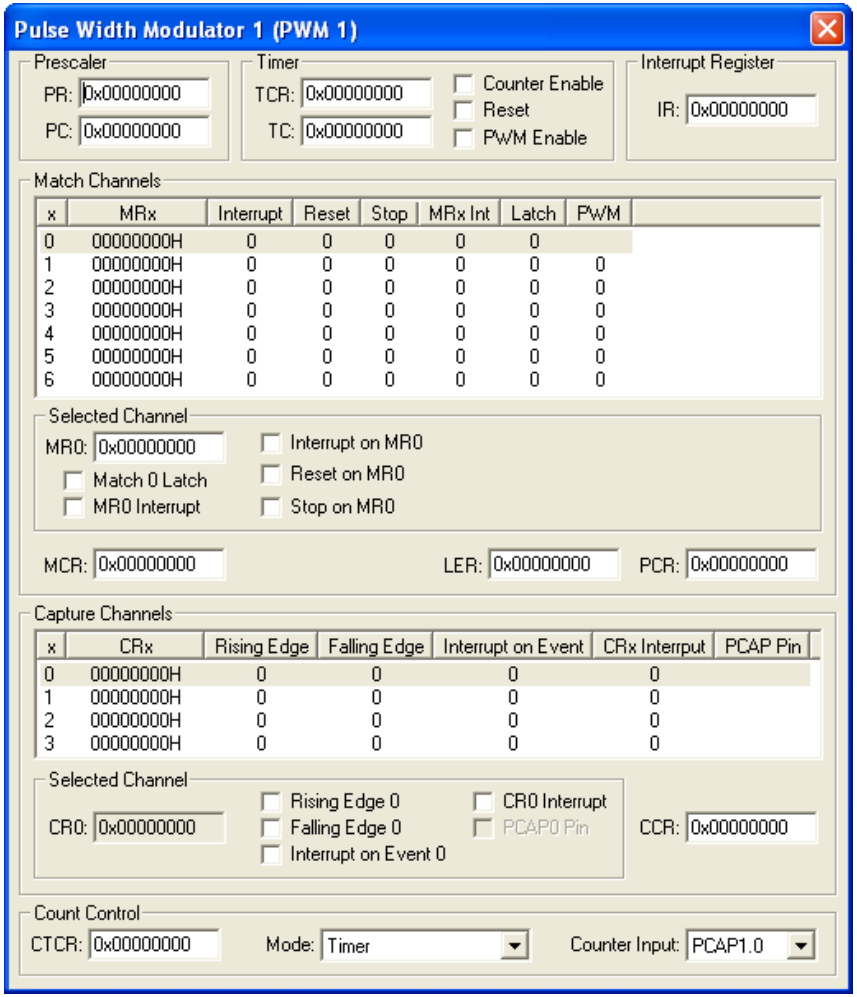
2.2. Pulse Width Modulator (PWM)

- Shadow latch mechanism:
  - When modifying a match register (**PWM1MR**n) is necessary set to "1" in PMW1LER bit corresponding.



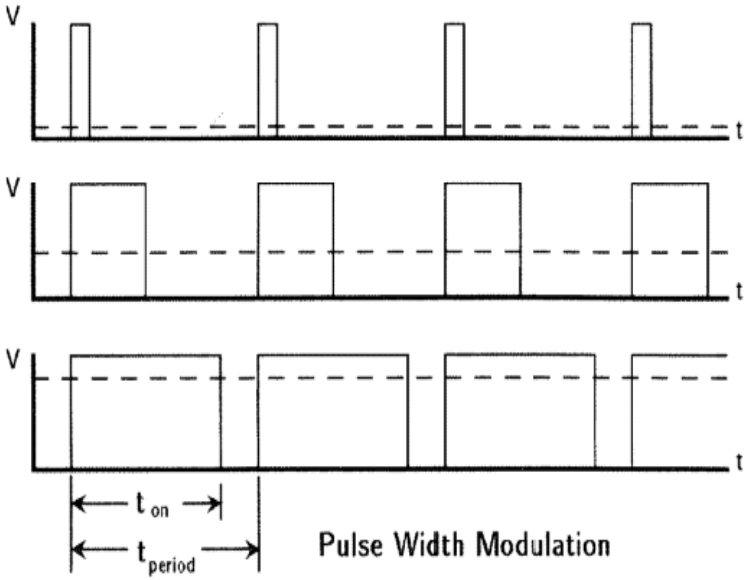
2.2. PWM: Keil ARM-MDK Debug Windows

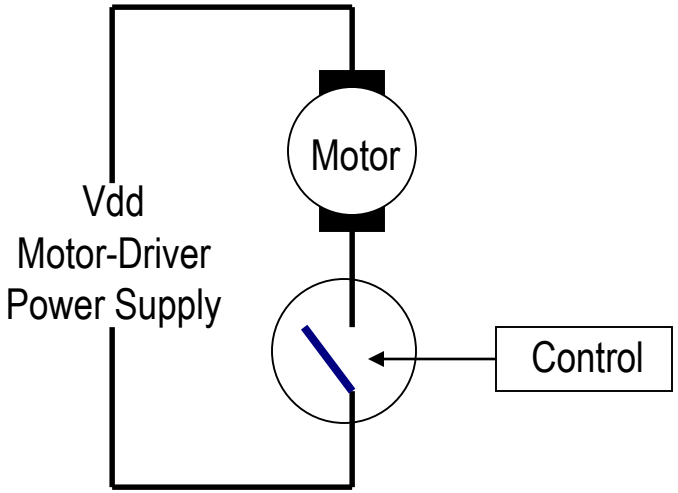
- **Peripherals-> Pulse Width Modulator 1 (PWM 1)**

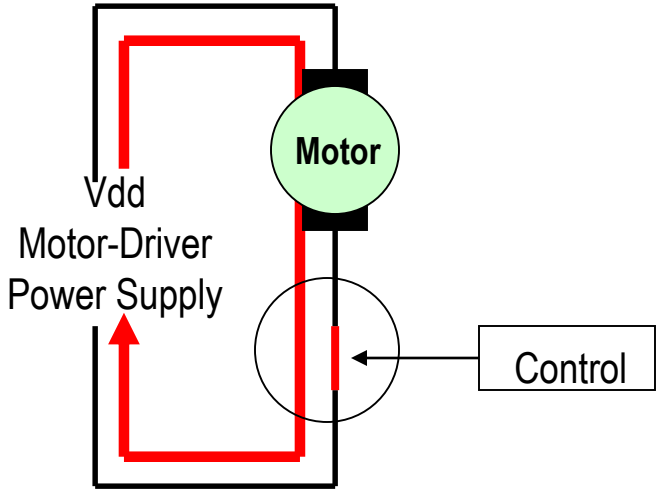


2.2. PWM: Drive DC-motors

PWM to drive DC-motors







2.2. PWM: Motor speed control

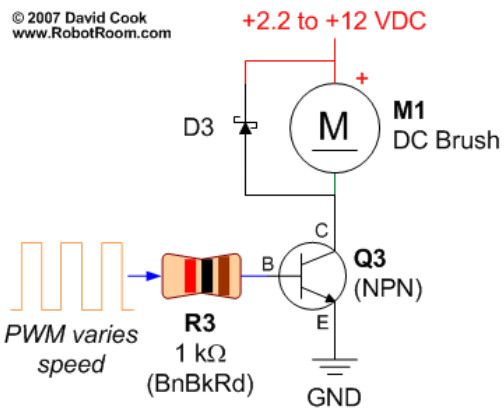
Peristaltic Liquid Pump with Silicone Tubing

- Uses approx 3/16" (4.8mm) outer diameter silicone tubing, the pump tube size has changed on us, so please measure the tubing that comes with your pump to verify!
- Working Temperature: 0°C - 40 °C
- Motor voltage: 12VDC
- Motor current: 200-300mA
- Flow rate: up to 100 mL/min
- Weight: 200 grams
- Dimensions: 27mm diameter motor, 72mm total length
- Mounting holes: 2.7mm diameter, 50mm center-to-center distance

PWM1 configuration:

```
LPC_PINCON->PINSEL4 |= 1<<0;          // P2.0 works as PWM1.1 output (table 83)
LPC_SC->PCONP |= 1<<6;                // power on
LPC_PWM1->MR0= Fpclk/Fpwm–1;          // set frequency of PWM1
LPC_PWM1->PCR|=1<<9;                  // PWMENA1=1
LPC_PWM1->MCR|=1<<1;                  // reset timer on Match0
LPC_PWM1->TCR |= (1<<0) | (1<<3);     // start timer, timer enable
```

```
void set_duty_cycle_pwm1 (float cycle)
{
      LPC_PWM1->MR1= LPC_PWM1->MR0*cycle/100;     // duty cycle
      LPC_PWM1->LER|= (1 << 1)|(1<<0);            // PWMLER[0-1]=1
}
```



2.2. PWM: Motor speed/direction control

H-Bridge circuit:

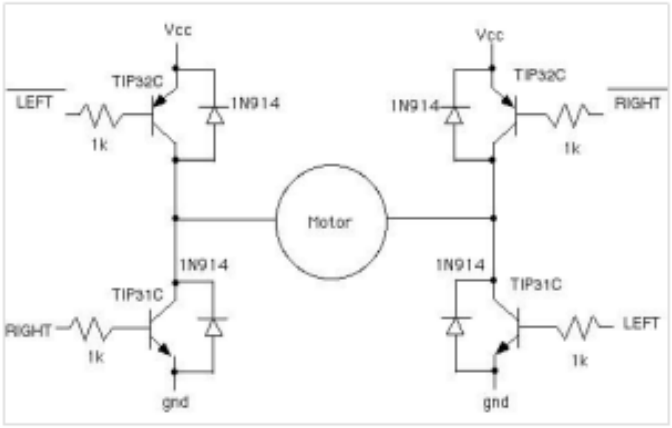
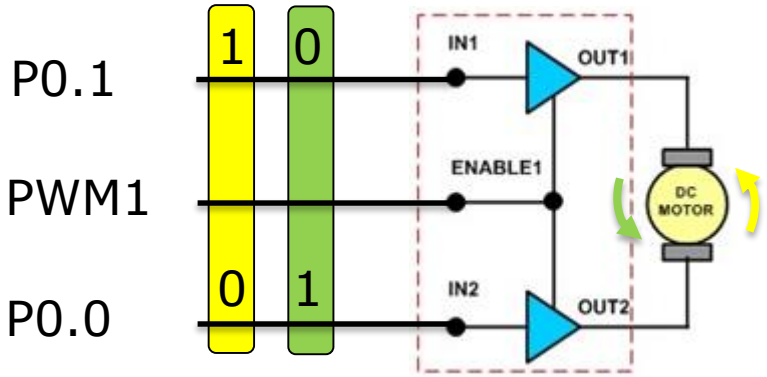
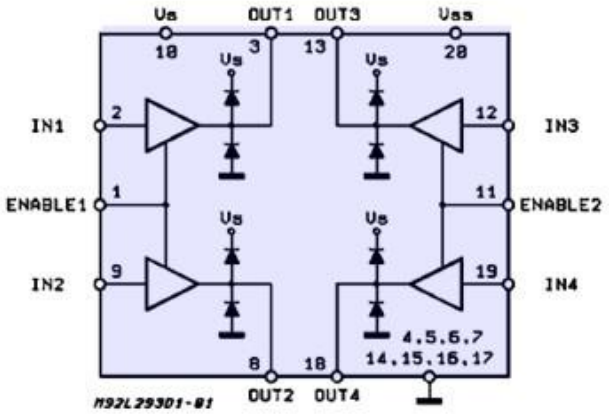
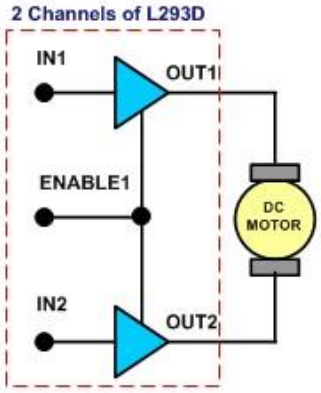


Figure 1: Typical H-bridge circuit



http://www.ermicro.com/blog





TRUTH TABLE

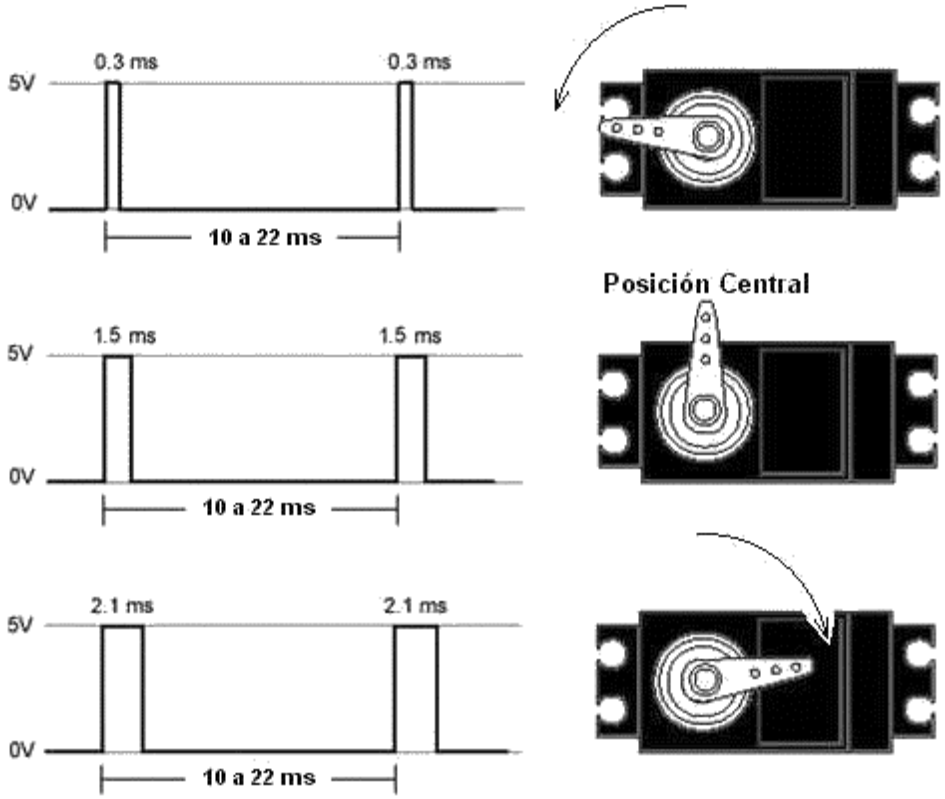
ENABLE1	IN1	IN2	OUT1	OUT2	MOTOR
H	L	L	L	L	Stop
H	H	L	H	L	Forward
H	L	H	L	H	Reverse
H	H	H	H	H	Break
L	X	X	Z	Z	Stop

H - High L - Low X - High/Low Z - High Impedance

The L293D Four Channels Motor Driver Truth Table

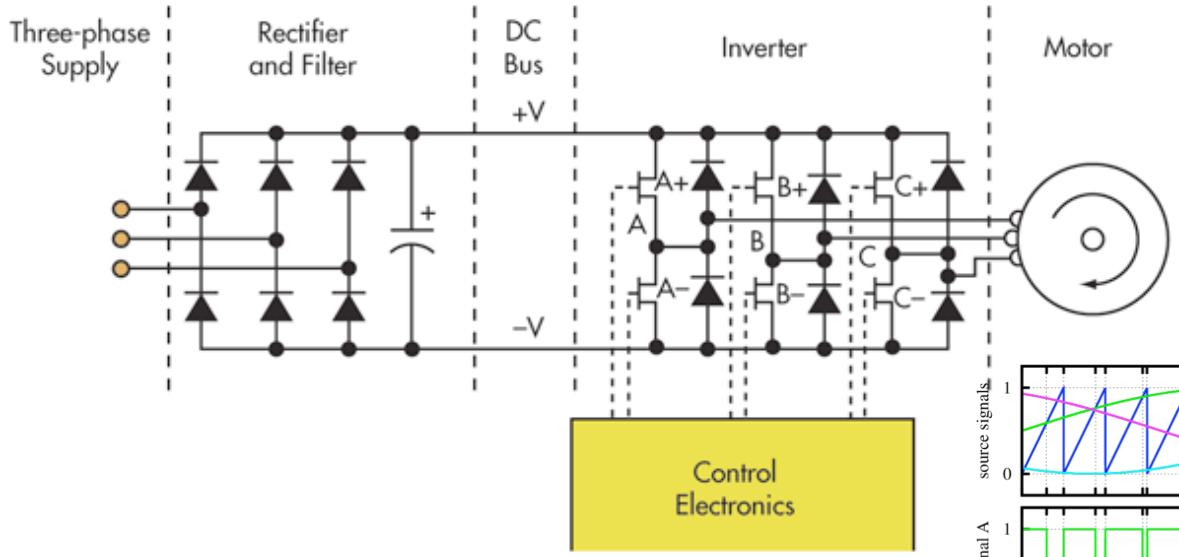
2.2. PWM: Servomotor control

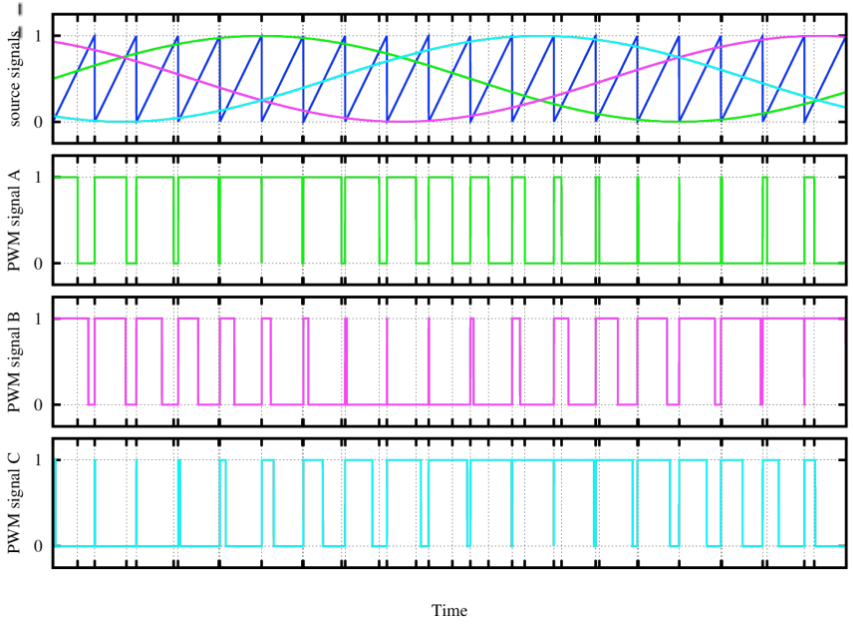
- Servomotor
  - Low power motor position-controlled.
  - Setpoint fixed with T_{on} in a fixed T signal ("PWM" type)



2.2. PWM: Three-phase motors control

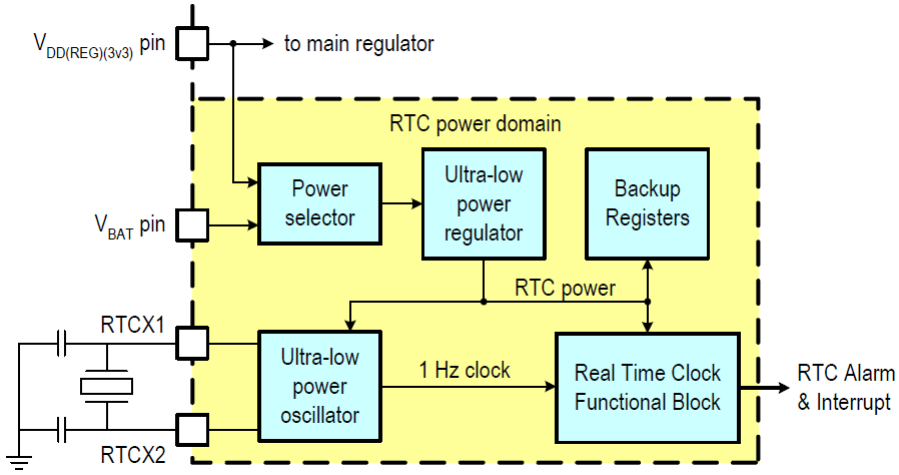
- PWM to drive three-phase motors (frequency controlled inverters)





2.2. RTC: Real-Time Clock

- Ultra-low power 32 kHz oscillator provides 1 Hz clock to the RTC.
- Separate battery power supply.Uses CPU power supply, when present.
- Calibration mechanism.
  - ± 1 second per day
- Battery-backed registers.



- Alarm function generates interrupts
  - Wakes CPU from reduced power modes.
  - 1 second resolution.
- Extremely low power consumption
  - 390 nA (typical @ 25º C)
- Calendar function does not require CPU involvement.
  - RTC works with Vbat as low as 2.1 V.

2.2. RTC: Config. registers

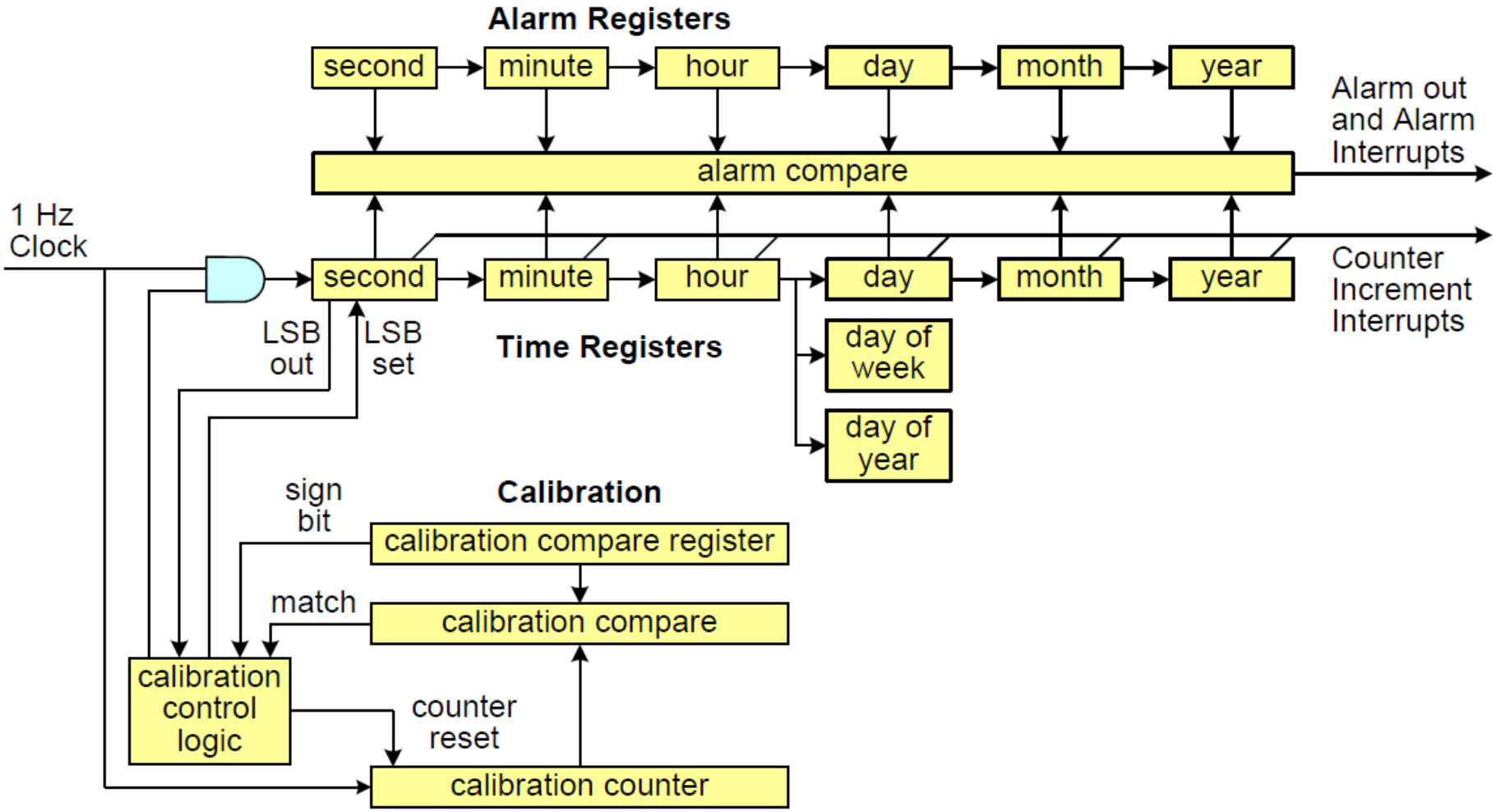
Table 508. Interrupt Location Register (ILR - address 0x4002 4000) bit description

Bit	Symbol	Description	Reset value
0	RTCCIF	When one, the Counter Increment Interrupt block generated an interrupt. Writing a one to this bit location clears the counter increment interrupt.	0
1	RTCALF	When one, the alarm registers generated an interrupt. Writing a one to this bit location clears the alarm interrupt.	0
31:21	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 509. Clock Control Register (CCR - address 0x4002 4008) bit description

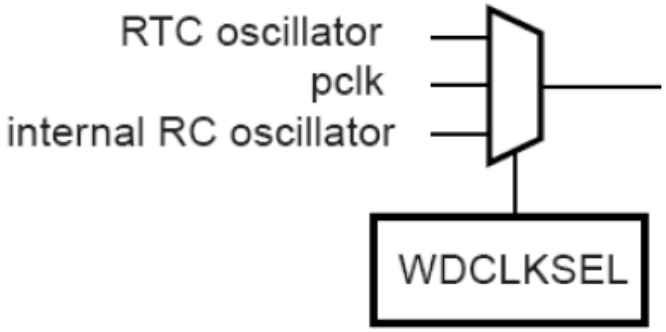
Bit	Symbol	Value	Description	Reset value
0	CLKEN		Clock Enable.	NC
		1	The time counters are enabled.	
		0	The time counters are disabled so that they may be initialized.	

2.2. RTC: Block Diagram



2.2. WDT: Watchdog Timer (I)

- Watchdog is a OS tool: **CPU grabbing management.**
- The Watchdog consists of fixed prescaler ($\mathbf{T}_{WDCLK} \times 4$) and a 32-bit time-out counter.
- Clocked from the RTC, IRC oscillator, or the peripheral clock.



- The timer decrements when clocked, and when arrives to 0, it may:
  - Reset the uC.
  - Generate an IRQ.
- Cannot be disabled by software, just in the Reset or IRQ ISRs.
- The maximum + minimum Watchdog intervals are:
  - The minimum Watchdog interval is ($\mathbf{T}_{WDCLK} \times 256 \times 4$)
  - The maximum Watchdog interval is ($\mathbf{T}_{WDCLK} \times 2^{32} \times 4$) in multiples of ($\mathbf{T}_{WDCLK} \times 4$)

2.2. WDT: Watchdog Timer (II)

- 32-bit with programmable time-out.
- Clocked from the IRC, RTC oscillator, or the peripheral clock.
- Cannot be disabled by software.

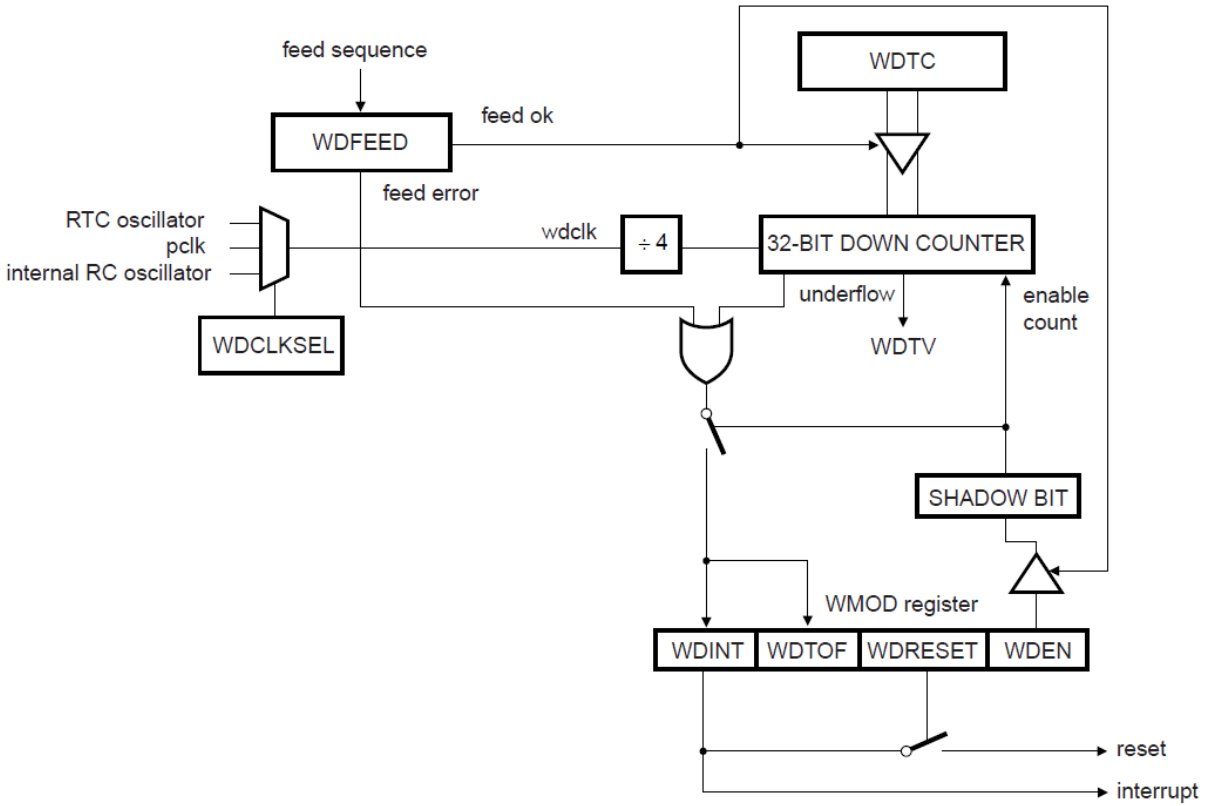
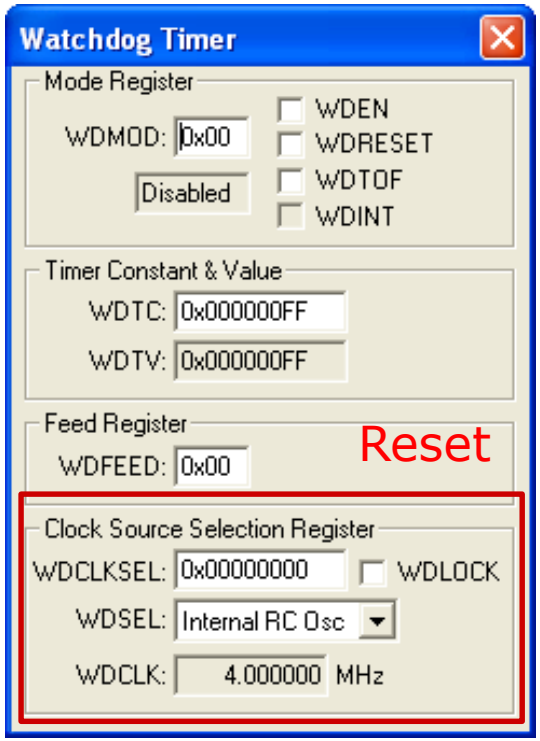


Fig 132. Watchdog block diagram



2.2. WDT: Description

- The Watchdog consists of a divide by 4 fixed prescaler and a 32-bit counter.
- The timer decrements when clocked.
- The minimum value from which the counter decrements is 0xFF.
  - The minimum Watchdog interval is ($\mathbf{T}_{WDCLK} \times 256 \times 4$)
  - The maximum Watchdog interval is ($\mathbf{T}_{WDCLK} \times 2^{32} \times 4$) in multiples of ($\mathbf{T}_{WDCLK} \times 4$).
- The Watchdog should be used in the following manner:
  - **Set the Watchdog timer constant reload value in WDTC register.**
  - **Setup the Watchdog timer operating mode in WDMOD register.**
  - **Enable the Watchdog by writing 0xAA followed by 0x55 to the WDFEED register:**

```
LPC_WDT->FEED = 0xAA;  /* Feeding sequence */
LPC_WDT->FEED = 0x55;
```

  - **The Watchdog should be fed again before the Watchdog counter underflows to prevent reset/interrupt.**

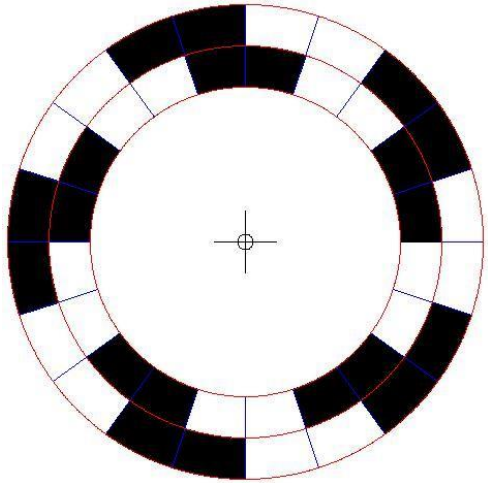
2.2. WDT: Registers description

Table 522. Watchdog register map

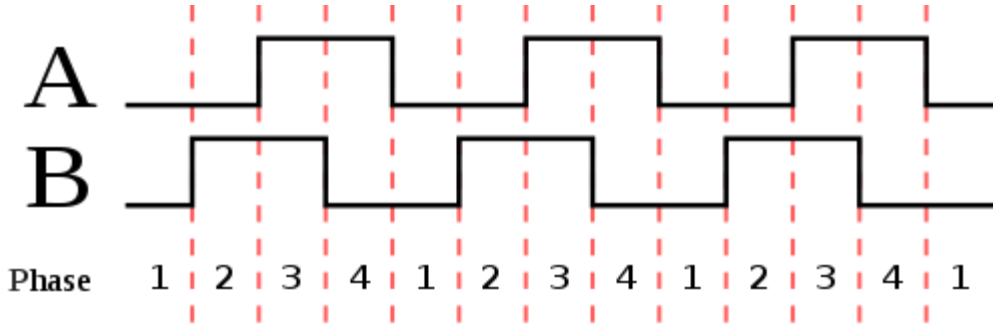
Name	Description	Access	Reset Value[1]	Address
WDMOD	Watchdog mode register. This register contains the basic mode and status of the Watchdog Timer.	R/W	0	0x4000 0000
WDTC	Watchdog timer constant register. This register determines the time-out value.	R/W	0xFF	0x4000 0004
WDFEED	Watchdog feed sequence register. Writing 0xAA followed by 0x55 to this register reloads the Watchdog timer with the value contained in WDTC.	WO	NA	0x4000 0008
WDTV	Watchdog timer value register. This register reads out the current value of the Watchdog timer.	RO	0xFF	0x4000 000C
WDCLKSEL	Watchdog clock source selection register.	R/W	0	0x4000 0010

2.2. QEI: Quadrature Encoder Interface

- A quadrature encoder consists of a disc with two tracks, containing alternating areas of reflection and non reflection, 90 degrees out of phase.



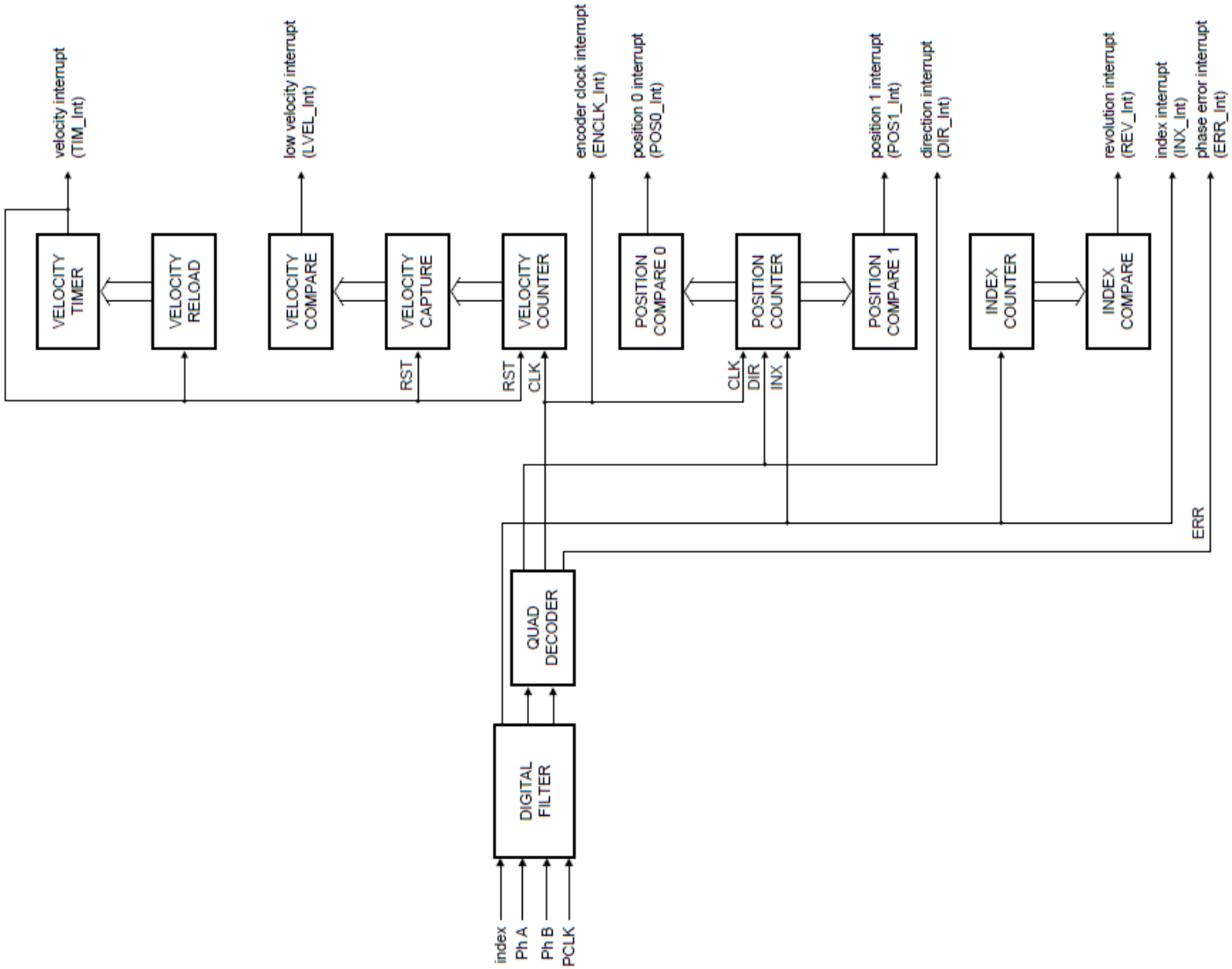
- As it rotates in front of an emitter/receiver pair for each track (which we will call channel A, and channel B), it will produce the following results.



2.2. QEI: Characteristics

- Tracks encoder position.
- Increments/ decrements depending on direction.
- Programmable for 2X or 4X position counting.
- Velocity capture using built-in timer.
- Velocity compare function with less than interrupt.
- Uses 32-bit registers for position and velocity.
- Three position compare registers with interrupts.
- Index counter for revolution counting.
- Index compare register with interrupts.
- Can combine index and position interrupts to produce an interrupt for whole and partial revolution displacement.
- Digital filter with programmable delays for encoder input signals.
- Can accept decoded signal inputs (clock and direction).

2.2. QEI: Block Diagram



2.2. QEI: Signals Operation

- The QEI module supports **two modes** of signal operation:
  - **In quadrature phase mode**, the encoder produces two clocks that are 90 degrees out of phase; the edge relationship is used to determine the direction of rotation.
  - **In clock/direction mode**, the encoder produces a clock signal to indicate steps and a direction signal to indicate the direction of rotation.

Table 479. Encoder states

Phase A	Phase B	state
1	0	1
1	1	2
0	1	3
0	0	4

Table 481. Encoder direction

DIR bit	DIRINV bit	direction
0	0	forward
1	0	reverse
0	1	reverse
1	1	forward

Table 480. Encoder state transitions[1]

from state	to state	Direction
1	2	positive
2	3	
3	4	
4	1	
4	3	negative
3	2	
2	1	
1	4	

2.2. QEI: Registers

Table 483. QEI Register summary

Name	Description	Access	Reset value	Address
Control registers				
QEICON	Control register	WO	0	0x400B C000
QEICONF	Configuration register	R/W	0	0x400B C008
QEISTAT	Encoder status register	RO	0	0x400B C004
Position, index, and timer registers				
QEIPOS	Position register	RO	0	0x400B C00C
QEIMAXPOS	Maximum position register	R/W	0	0x400B C010
CMPOS0	position compare register 0	R/W	0	0x400B C014
CMPOS1	position compare register 1	R/W	0	0x400B C018
CMPOS2	position compare register 2	R/W	0	0x400B C01C
INXCNT	Index count register	RO	0	0x400B C020
INXCMP	Index compare register	R/W	0	0x400B C024
QEILOAD	Velocity timer reload register	R/W	0	0x400B C028
QEITIME	Velocity timer register	RO	0	0x400B C02C
QEIVEL	Velocity counter register	RO	0	0x400B C030
QEICAP	Velocity capture register	RO	0	0x400B C034
VELCOMP	Velocity compare register	R/W	0	0x400B C038
FILTER	Digital filter register	R/W	0	0x400B C03C
Interrupt registers				
QEIINTSTAT	Interrupt status register	RO	0	0x400B CFE0
QEISET	Interrupt status set register	WO	0	0x400B CFEC
QEICLR	Interrupt status clear register	WO	0	0x400B CFE8
QEIIE	Interrupt enable register	RO	0	0x400B CFE4
QEIIES	Interrupt enable set register	WO	0	0x400B CFDC
QEIIEC	Interrupt enable clear register	WO	0	0x400B CFD8

2.2. QEI: Velocity measurement (rpm)

- The following equation converts the velocity counter value into an RPM value:

$$RPM = (PCLK * QEICAP * 60) \div (QEILOAD * PPR * Edges)$$

- where:
  - **PCLK** is the peripheral clock rate for the QEI block. See Section 4.7.3 for more on the possibilities for PCLK).
  - **QEICAP** is the captured velocity counter value for the last velocity timer period.
  - **QEILOAD** is the velocity timer reload value.
  - **PPR** is the number of pulses per revolution of the physical encoder used in the application.
  - **Edges** is 2 or 4, based on the capture mode set in the QEICON register (2 for CapMode set to 0 and 4 for CapMode set to 1)

2.2. QEI: Signals Operation

- When edges on PhA lead edges on PhB, the position counter is incremented.
- When edges on PhB lead edges on PhA, the position counter is decremented.
- When a rising and falling edge pair is seen on one of the phases without any edges on the other, the direction of rotation has changed.

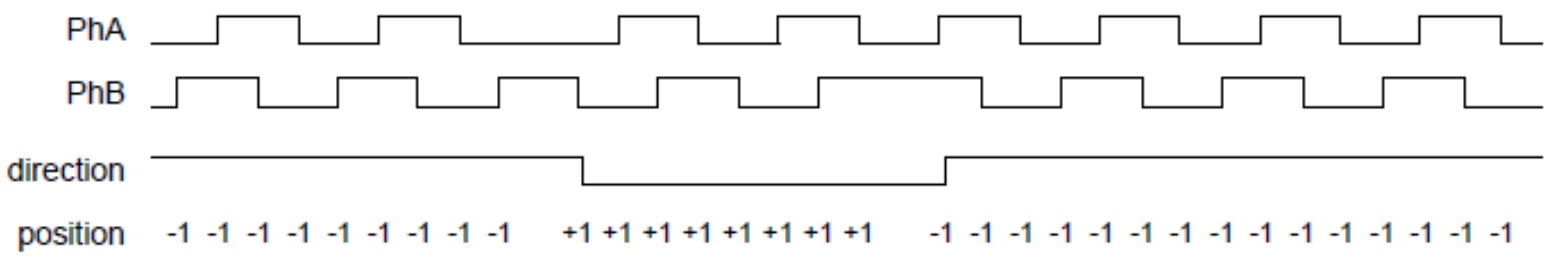


Fig 129.Quadrature Encoder Basic Operation

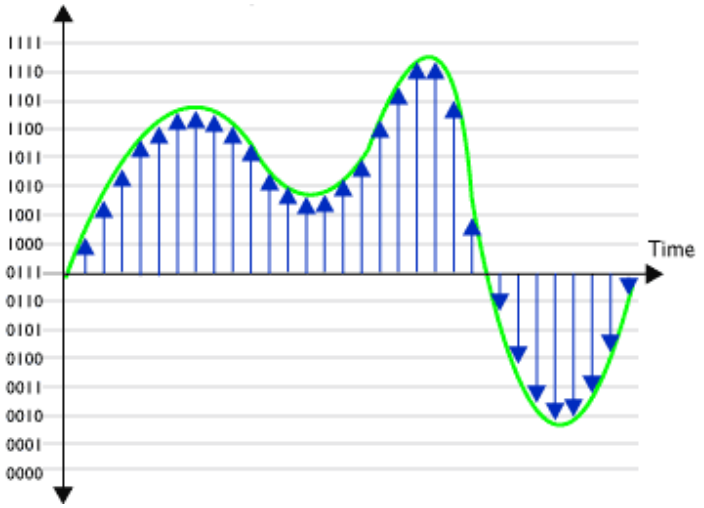
2.3. Analogs Input/Output

Analog **D**igital **C**onverter

Digital **A**nalog **C**onverter

2.3. ADC: Analog Digital Converter

- What is the ADC – Analog to Digital Conversion?
  - Signals in the nature to be controlled by any embedded system are normally ANALOG.
  - In order to work with those signals the embedded systems need a code representing the instantaneous value of each signal.



- Important parameters of the conversion:
  - Conversion code – number of bits
  - SPAN.
  - Number and type of analog inputs (differencial…)
  - Conversion time – Sampling time
  - Conversion errors: quantification, gain, lineality, offset …

2.3. ADC: Characteristics

- LPC1768 has a 12bits A/D converter:
  - SAR based conversor.
  - 8 multiplexed analog inputs.
  - Max. conversion frequency 200KHz.
    - → Conversion time 5us
    - → 65 x T_{ADC}, $f_{ADC} \leq 13MHz$
  - SPAN defined between V_{REFN} to V_{REFP} ($V_{REFP} - V_{REFN} \leq V_{DDA}$)
  - **Single** (manual) or **Burst** conversion (continual conversion)
  - Conversion can be started:
    - With a direct order (software)
    - With a transition of an external pin.
    - With a timer (output compare).

2.3. ADC: Pins and Registers

Pins:

AD0.0 to AD0.7	Analog inputs, The ADC can read the voltage in any of these 8 pins, the digital function is disabled in a pin when ADC is enabled in it.
V_{REFP}, V_{REFN}	Voltage references, Provide the work range for the convertion. It is usually 3.3v and GND.
V_{DDA}, V_{SSA}	Analog power and ground, they are also 3.3v (isolation is recommended in the user guide but not done here) and GND.

Registers:

ADCR	A/D Control Register. The ADCR register must be written to select the operating mode before A/D conversion can occur.
ADGDR	A/D Global Data Register. This register contains the ADC's DONE bit and **the result of the most recent A/D conversion**.
ADINTEN	A/D Interrupt Enable Register. This register contains enable bits that allow the DONE flag of each A/D channel to be included or excluded from contributing to the generation of an A/D interrupt.
ADDR0 to ADDR7	A/D Channel Data Register. This register contains the result of the most recent conversion completed on each from 0 to 7.
ADSTAT	A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt/DMA flag.

2.3. ADC Registers

Name	Access	Description	Address Offset
AD0CR	R/W	A/D Control Register	0x0000
AD0GDR	R/W	A/D Global Data Register	0x0004
AD0INTEN	R/W	A/D Interrupt Enable Register	0x000C
AD0DR0	R/W	A/D Channel 0 Data Register	0x0010
AD0DR1	R/W	A/D Channel 1 Data Register	0x0014
AD0DR2	R/W	A/D Channel 2 Data Register	0x0018
AD0DR3	R/W	A/D Channel 3 Data Register	0x001C
AD0DR4	R/W	A/D Channel 4 Data Register	0x0020
AD0DR5	R/W	A/D Channel 5 Data Register	0x0024
AD0DR6	R/W	A/D Channel 6 Data Register	0x0028
AD0DR7	R/W	A/D Channel 7 Data Register	0x002C
AD0STAT	RO	A/D Status Register	0x0030

2.3. ADC: Configuration

- **Basic configuration**
  - Power on ADC with PCADC (PCONP)
  - Configure PCLK_ADC in PCLKSEL0 (Default F_cclk/4).
  - Configure pins for ADC inputs in PINSEL and PINMODE.
    - No pull-up nor pull-down
  - Enable IRQ in NVIC.
  - Select Interrupt Priority in NVIC.

- **Advanced configuration**
  - **Fine tunning the ADC (ADTRIM)**
    - For tunning ADC and DAC internal offset.
  - **With DMA**
    - To send conversions directly to memory.
    - When configured, DMA starts automatically with IRQ (disable NVIC)
    - To be used in BURST mode.
    - DMA configured to transfer 1, 4 or 8 words according to the burst length.

2.3. ADC: Conversion modes

- **Manual** mode (software controlled)
  - Power on ADC: PDN(AD0CR[21]) = 1
  - Configure manual model: BURST(AD0CR[16]) = 0
  - Conversion for a single channel, selected in SEL(AD0CR[7:0])
  - Start the conversion in START(AD0CR[26:24]) + EDGE(AD0CR[27])
  - Can be started with an event in P0.10 (INT0), P1.27(CAP1.0), MAT0.1, MAT0.3, MAT1.0, MAT1.1

- **BURST** mode
  - Power on ADC: PDN(AD0CR[21]) = 1
  - Configure Burst mode: BURST(AD0CR[16]) = 1
  - Select channels to be converted in SEL(AD0CR[7:0]).
  - No manual conversion allowed!!! START(AD0CR[26:24]) = **0b000**

2.3. ADC: Results of conversions

- Results can be globally read at **AD0GDR** (Global Data Register)
  - Last conversion in RESULT bits (AD0GDR[**15:4**])
  - Number of last channel converted in CHN bits (AD0GDR[**26:24**])
  - Overrun flag in OVERRUN bit (AD0GDR[30])
    - Clear when read AD0GDR .
  - End of Conversion flag in DONE bit (AD0GDR[31])
    - Clear when read AD0GDR.
    - Can interrupt if AD0GINTEN(AD0INTEN[8]) = 1
- Results can be individually read for each channel in **AD0DRn** (Data Register of Channel **n**)
  - Last conversion for channel **n** in RESULT bits (AD0DRn[**15:4**]).
  - Overrun flag for channel **n** in OVERRUN bit (AD0DRn[**30**])
    - Clear when read AD0DRn
  - End of conversion flag for channel **n** in DONE bit (AD0DRn[**31**])
    - Clear when read AD0DRn
    - Can interrupt if AD0INTEN(ADINTEN[n]) = 1 and AD0INTEN(ADINTEN[8]) = 0

2.3. ADC: Control Register

AD0CR

Bit	Symbol	Value	Description	Reset
7:0	SEL		Selects which of the AD0.7:0 pins is (are) to be sampled and converted. For AD0, bit 0 selects Pin AD0.0, and bit 7 selects pin AD0.7. In software-controlled mode, only one of these bits should be 1. In hardware scan mode, any value containing 1 to 8 ones is allowed. All zeroes is equivalent to 0x01.	0x01
15:8	CLKDIV		The APB clock (PCLK_ADC0) is divided by (this value plus one) to produce the clock for the A/D converter, which should be less than or equal to 13 MHz.	0
16	BURST	1	The AD converter does repeated conversions at up to 200 kHz, scanning (if necessary) through the pins selected by bits set to ones in the SEL field. **Remark:** START bits must be 000 when BURST = 1 or conversions will not start.	0
		0	Conversions are software controlled and require 65 clocks.	
20:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
21	PDN	1	The A/D converter is operational.	0
		0	The A/D converter is in power-down mode.	
23:22	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
26:24	START		When the BURST bit is 0, these bits control whether and when an A/D conversion is started:	0
		000	No start (this value should be used when clearing PDN to 0).	
		001	Start conversion now.	
		010	Start conversion when the edge selected by bit 27 occurs on the P2.10 / EINT0 / NMI pin.	
		011	Start conversion when the edge selected by bit 27 occurs on the P1.27 / CLKOUT / USB_OVRCRn / CAP0.1 pin.	
		100	Start conversion when the edge selected by bit 27 occurs on MAT0.1.	
		101	Start conversion when the edge selected by bit 27 occurs on MAT0.3.	
		110	Start conversion when the edge selected by bit 27 occurs on MAT1.0.	
		111	Start conversion when the edge selected by bit 27 occurs on MAT1.1.	
27	EDGE		This bit is significant only when the START field contains 010-111. In these cases:	0
		1	Start conversion on a falling edge on the selected CAP/MAT signal.	
		0	Start conversion on a rising edge on the selected CAP/MAT signal.	
31:28	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

2.3. ADC: Global Data Register

AD0GDR

Table 532: A/D Global Data Register (AD0GDR - address 0x4003 4004) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin selected by the SEL field, as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}.	NA
23:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
26:24	CHN	These bits contain the channel from which the RESULT bits were converted (e.g. 000 identifies channel 0, 001 channel 1...).	NA
29:27	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits. This bit is cleared by reading this register.	0
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion is started.	0

2.3. ADC: Global Data Register

AD0INTEN

Table 533: A/D Status register (AD0INTEN - address 0x4003 400C) bit description

Bit	Symbol	Value	Description	Reset value
0	ADINTEN0	0	Completion of a conversion on ADC channel 0 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 0 will generate an interrupt.	
1	ADINTEN1	0	Completion of a conversion on ADC channel 1 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 1 will generate an interrupt.	
2	ADINTEN2	0	Completion of a conversion on ADC channel 2 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 2 will generate an interrupt.	
3	ADINTEN3	0	Completion of a conversion on ADC channel 3 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 3 will generate an interrupt.	
4	ADINTEN4	0	Completion of a conversion on ADC channel 4 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 4 will generate an interrupt.	
5	ADINTEN5	0	Completion of a conversion on ADC channel 5 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 5 will generate an interrupt.	
6	ADINTEN6	0	Completion of a conversion on ADC channel 6 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 6 will generate an interrupt.	
7	ADINTEN7	0	Completion of a conversion on ADC channel 7 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 7 will generate an interrupt.	
8	ADGINTEN	0	Only the individual ADC channels enabled by ADINTEN7:0 will generate interrupts.	1
		1	Only the global DONE flag in ADDR is enabled to generate an interrupt.	
31:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

2.3. ADC: Data Registers

AD0DR0…AD0DR7

Table 534: A/D Data Registers (AD0DR0 to AD0DR7 - 0x4003 4010 to 0x4003 402C) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin, as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}.	NA
29:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits.This bit is cleared by reading this register.	
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.	NA

2.3. ADC: Status Register

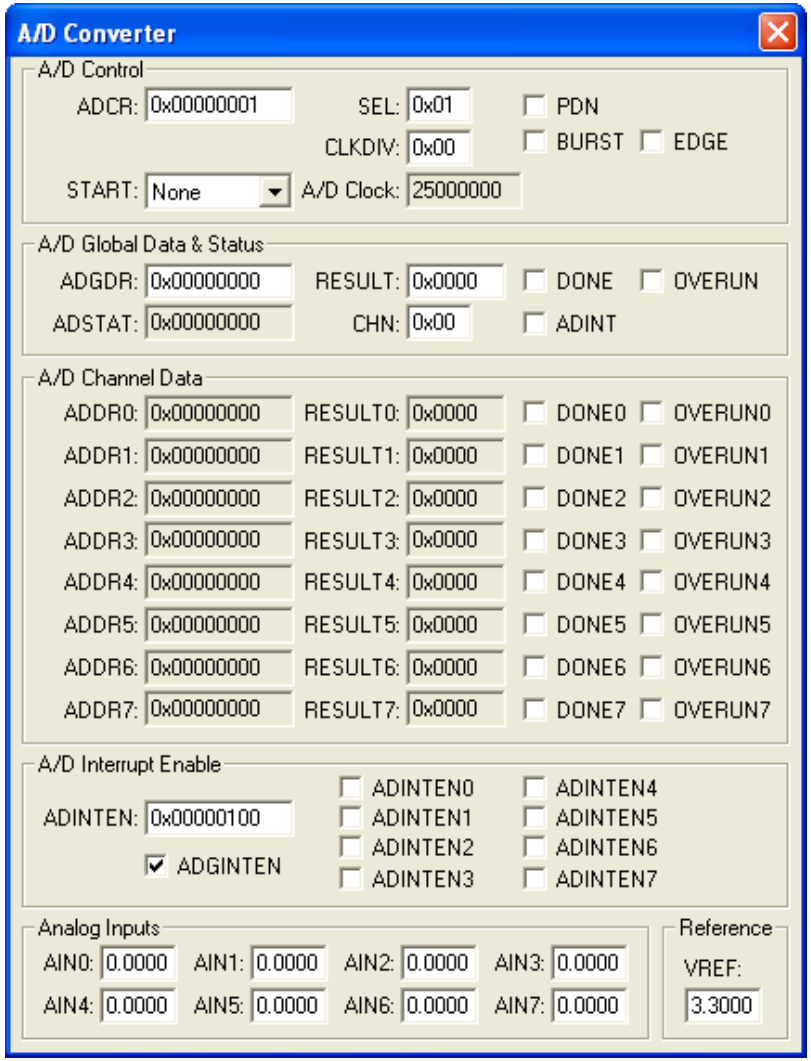
AD0STAT

Table 535: A/D Status register (AD0STAT - address 0x4003 4030) bit description

Bit	Symbol	Description	Reset value
0	DONE0	This bit mirrors the DONE status flag from the result register for A/D channel 0.	0
1	DONE1	This bit mirrors the DONE status flag from the result register for A/D channel 1.	0
2	DONE2	This bit mirrors the DONE status flag from the result register for A/D channel 2.	0
3	DONE3	This bit mirrors the DONE status flag from the result register for A/D channel 3.	0
4	DONE4	This bit mirrors the DONE status flag from the result register for A/D channel 4.	0
5	DONE5	This bit mirrors the DONE status flag from the result register for A/D channel 5.	0
6	DONE6	This bit mirrors the DONE status flag from the result register for A/D channel 6.	0
7	DONE7	This bit mirrors the DONE status flag from the result register for A/D channel 7.	0
8	OVERRUN0	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 0.	0
9	OVERRUN1	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 1.	0
10	OVERRUN2	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 2.	0
11	OVERRUN3	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 3.	0
12	OVERRUN4	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 4.	0
13	OVERRUN5	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 5.	0
14	OVERRUN6	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 6.	0
15	OVERRUN7	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 7.	0
16	ADINT	This bit is the A/D interrupt flag. It is one when any of the individual A/D channel Done flags is asserted and enabled to contribute to the A/D interrupt via the ADINTEN register.	0
31:17	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

2.3. ADC: Keil Debugger

- Peripherals→A/D converter



2.3. ADC: Electrical Characteristics (I)

ADC Electrical Characteristics

Table 18. ADC characteristics (full resolution)

V_{DDA} = 2.7 V to 3.6 V; T_{amb} = −40 °C to +85 °C unless otherwise specified; ADC frequency 13 MHz; 12-bit resolution.

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IA}	analog input voltage		0	-	V_{DDA}	V
C_{ia}	analog input capacitance		-	-	15	pF
E_D	differential linearity error	[1][2]	-	-	±1	LSB
$E_{L(adj)}$	integral non-linearity	[3]	-	-	±3	LSB
E_O	offset error	[4][5]	-	-	±2	LSB
E_G	gain error	[6]	-	-	0.5	%
E_T	absolute error	[7]	-	-	4	LSB
R_{vsi}	voltage source interface resistance	[8]	-	-	7.5	kΩ
$f_{clk(ADC)}$	ADC clock frequency		-	-	13	MHz
$f_{c(ADC)}$	ADC conversion frequency	[9]	-	-	200	kHz

[1] The ADC is monotonic, there are no missing codes.

[2] The differential linearity error (E_D) is the difference between the actual step width and the ideal step width. See Figure 26.

[3] The integral non-linearity ($E_{L(adj)}$) is the peak difference between the center of the steps of the actual and the ideal transfer curve after appropriate adjustment of gain and offset errors. See Figure 26.

[4] The offset error (E_O) is the absolute difference between the straight line which fits the actual curve and the straight line which fits the ideal curve. See Figure 26.

[5] ADCOFFS value (bits 7:4) = 2 in the ADTRM register. See *LPC17xx user manual UM10360*.

[6] The gain error (E_G) is the relative difference in percent between the straight line fitting the actual transfer curve after removing offset error, and the straight line which fits the ideal transfer curve. See Figure 26.

[7] The absolute error (E_T) is the maximum difference between the center of the steps of the actual transfer curve of the non-calibrated ADC and the ideal transfer curve. See Figure 26.

[8] See Figure 27.

[9] The conversion frequency corresponds to the number of samples per second.

2.3. ADC: Electrical Characteristics (II)

- ADC Electrical Characteristics

Table 19. ADC characteristics (lower resolution)

T_{amb} = –40 °C to +85 °C unless otherwise specified; 12-bit ADC used as 10-bit resolution ADC.

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
E_D	differential linearity error		[1][2]	-	±1	-	LSB
$E_{L(adj)}$	integral non-linearity		[3]	-	±1.5	-	LSB
E_O	offset error		[4]	-	±2	-	LSB
E_G	gain error		[5]	-	±2	-	LSB
$f_{clk(ADC)}$	ADC clock frequency	$3.0\ V \le V_{DDA} \le 3.6\ V$		-	-	33	MHz
		$2.7\ V \le V_{DDA} < 3.0\ V$		-	-	25	MHz
$f_{c(ADC)}$	ADC conversion frequency	$3\ V \le V_{DDA} \le 3.6\ V$	[6]	-	-	500	kHz
		$2.7\ V \le V_{DDA} < 3.0\ V$	[6]	-	-	400	kHz

[1] The ADC is monotonic, there are no missing codes.

[2] The differential linearity error (E_D) is the difference between the actual step width and the ideal step width. See Figure 26.

[3] The integral non-linearity ($E_{L(adj)}$) is the peak difference between the center of the steps of the actual and the ideal transfer curve after appropriate adjustment of gain and offset errors. See Figure 26.

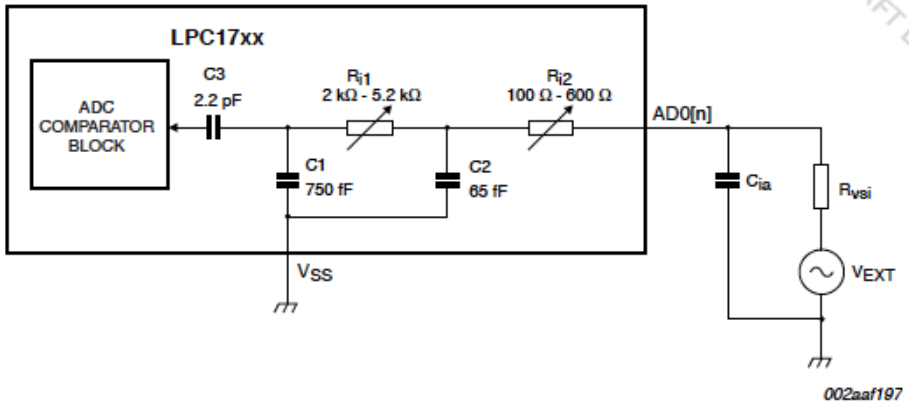
[4] The offset error (E_O) is the absolute difference between the straight line which fits the actual curve and the straight line which fits the ideal curve. See Figure 26.

[5] The gain error (E_G) is the relative difference in percent between the straight line fitting the actual transfer curve after removing offset error, and the straight line which fits the ideal transfer curve. See Figure 26.

[6] The conversion frequency corresponds to the number of samples per second.

2.3. ADC: Electrical Characteristics (III)

- ADC Electrical Characteristics



The values of resistor components R_{i1} and R_{i2} vary with temperature and input voltage and are process-dependent (see Table 20).

Parasitic resistance and capacitance from the pad are not included in this figure.

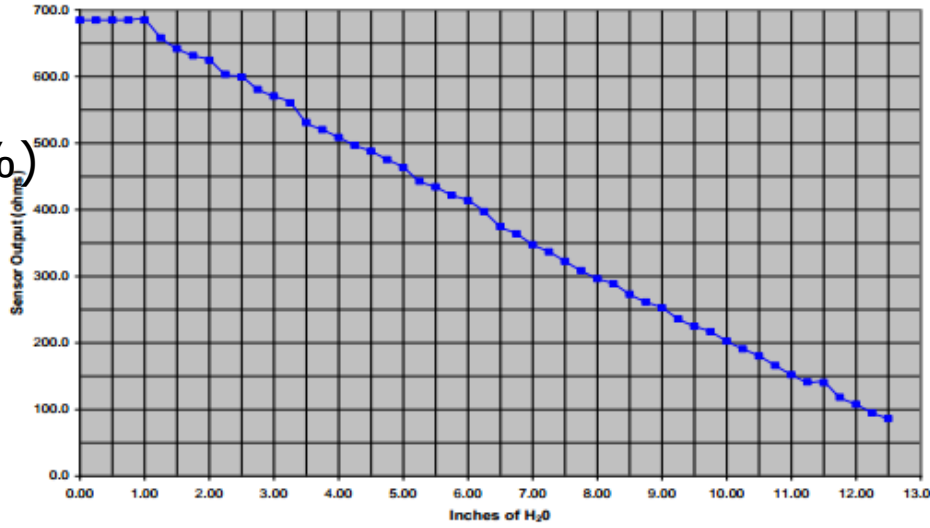
Fig 27. ADC interface to pins AD0[n]

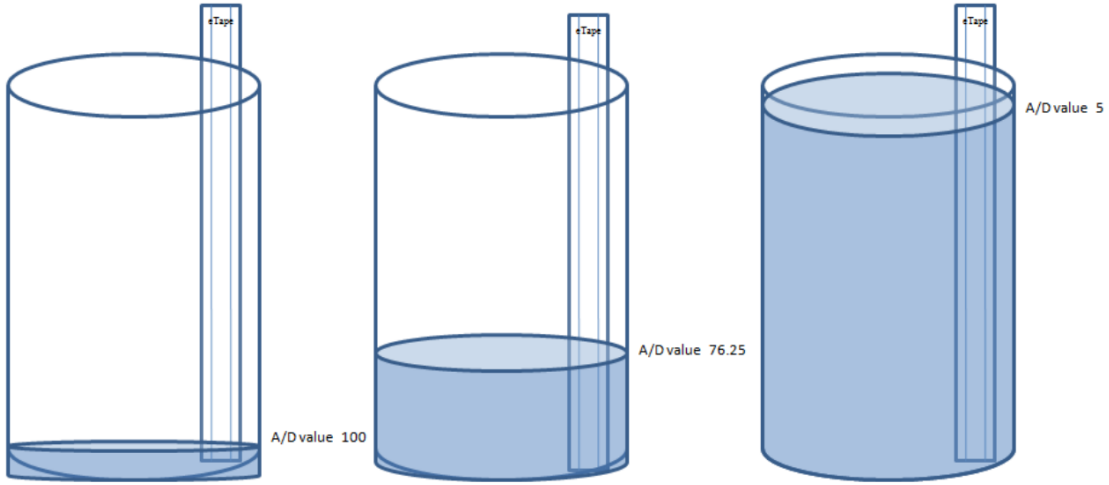
Table 20. ADC interface components

Component	Range	Description
R_{i1}	2 kΩ to 5.2 kΩ	Switch-on resistance for channel selection switch. Varies with temperature, input voltage, and process.
R_{i2}	100 Ω to 600 Ω	Switch-on resistance for the comparator input switch. Varies with temperature, input voltage, and process.
C1	750 fF	Parasitic capacitance from the ADC block level.
C2	65 fF	Parasitic capacitance from the ADC block level.
C3	2.2 pF	Sampling capacitor.

2.3. ADC: Aplicattions (sensors)

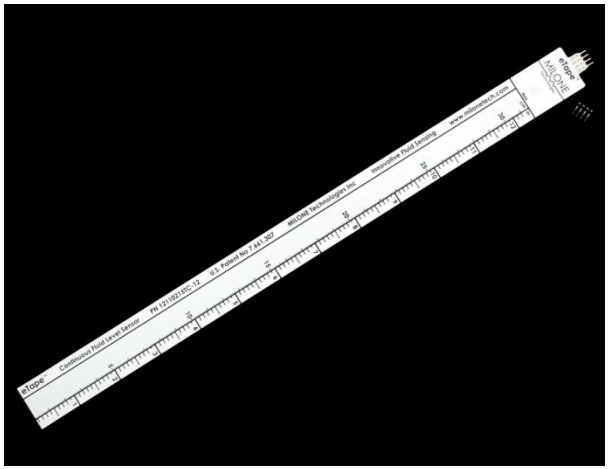
- ***Milone eTape***
- Liquid level sensor:
- Variable resistor (60-550 Ω ± 20%)





(A/D max – A/D meas) / (A/D max – A/D min) = % Full

(100 – 76.25) / (100 - 5) = .25 or 25% full



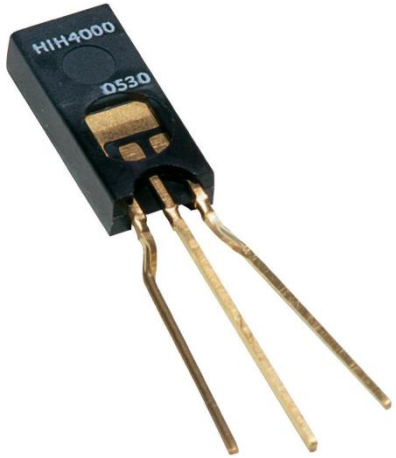
2.3. ADC: Aplicattions (sensors)

HIH-4000-001

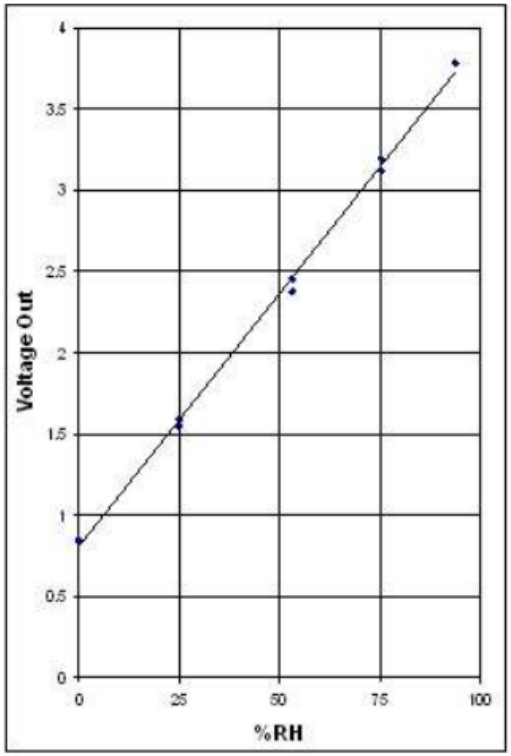
HIH-4000 Series Integrated Circuity Humidity Sensor, 2,54 mm (0.100 in) Lead Pitch SIP

Features

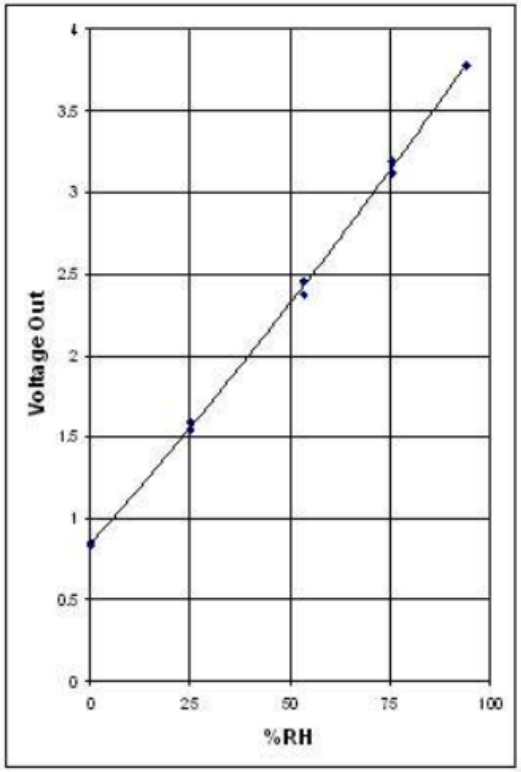
- Molded thermoset plastic housing with cover
- Linear voltage output vs %RH
- Laser trimmed interchangeability
- Low power design
- High accuracy
- Fast response time
- Stable, low drift performance
- Chemically resistant



TYPICAL BEST FIT STRAIGHT LINE



TYPICAL 2nd ORDER CURVE FIT



2.3. DAC: Digital Analog Converter

DAC Characteristics

- 10bits DAC conversor.
- Decoupled low impedance output (buffered).
- F_{DAC} max 1MHz.

Operation:

- Configure pins for ADC inputs in PINSEL (no PCONP)
- Conversion sequence:
  1. Data to be converted is written to VALUE bits (DACR[15:6])
  2. DAC output settles to:

$$\mathbf{Vo = VALUE \times ((V_{REFP} - V_{REFN})/1024) + V_{REFN}}$$

  1. Settling time can be configured in BIAS bit (DACR[16])
     - 0 → 1us max. 700 uA max. f_{DAC} max 1MHz
     - 1 → 2,5 us max, 350 uA max. f_{DAC} max 400KHz

2.3. DAC: Electrical Characteristics

- DAC Electrical Characteristics

Table 21. DAC electrical characteristics

V_{DDA} = 2.7 V to 3.6 V; T_{amb} = –40 °C to +85 °C unless otherwise specified

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
E_D	differential linearity error		-	±1	-	LSB
$E_{L(adj)}$	integral non-linearity		-	±1.5	-	LSB
E_O	offset error		-	0.6	-	%
E_G	gain error		-	0.6	-	%
C_L	load capacitance		-	200	-	pF
R_L	load resistance		1	-	-	kΩ

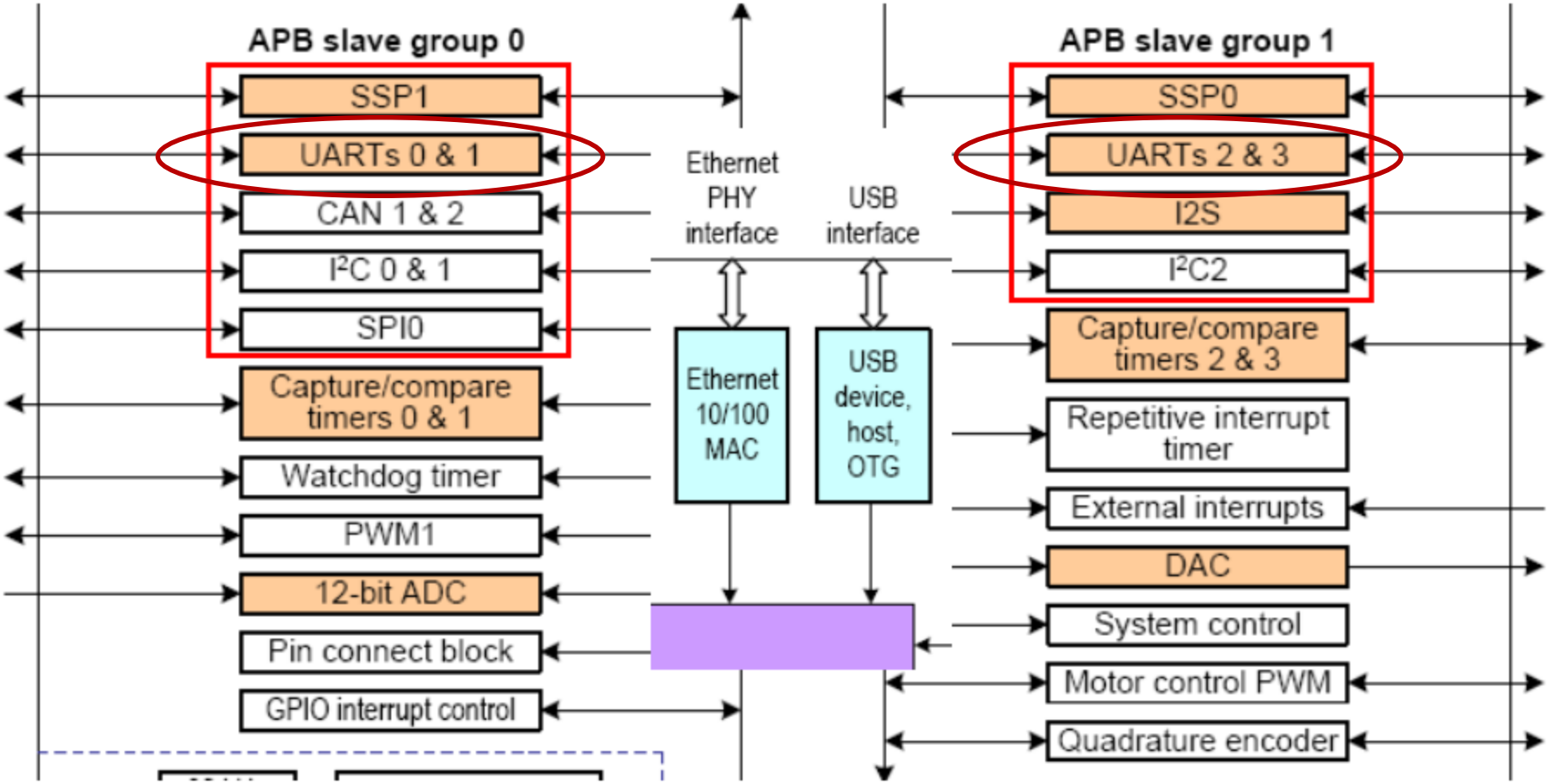
2.4. Serial Interfaces

UART: Universal **A**synchronous **R**x/**T**x

2.4. UART: Universal Asynchronous Rx/Tx

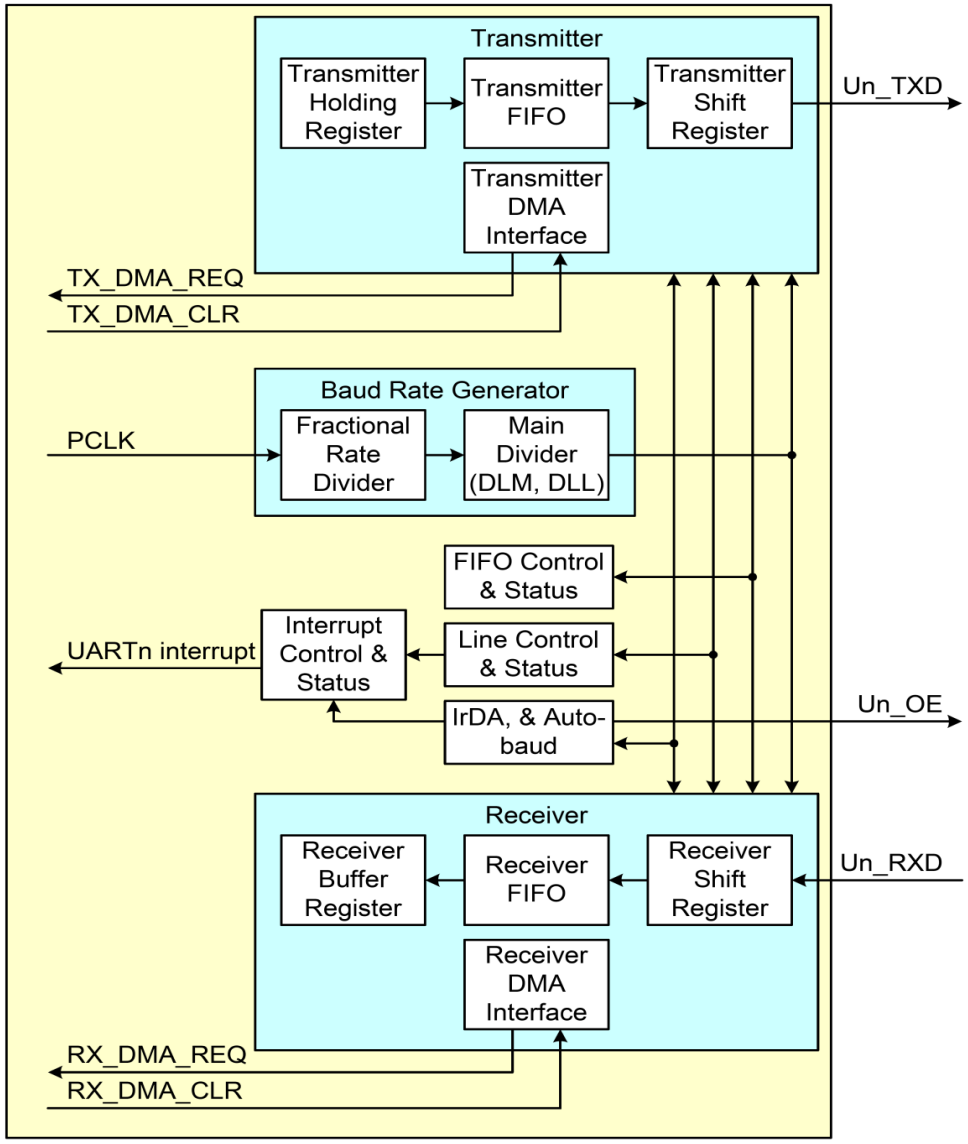
- **4 modules** (UART0, 1, 2, 3).
- 16 byte Receive and Transmit FIFOs with DMA support.
- Fractional divider for baud rate control, auto-baud capabilities, and implementation of software or hardware flow control.
- EIA-485/RS-485 and 9-bit mode support (UART1).
  - Allows both software address detection and automatic address detection using 9-bit mode.
  - Auto Direction Control.
    - Control line RTS/DTS to enable and disable the driver.
    - Address Match Register: store the station address in a RS-485 milti-drop communication.
    - Delay Value register: set the delay between the last stop-bit and the deassertion of the DIR line.
- Modem control support (UART1).
- IrDA support for infrared communication (UART3).
- Maximum possible speed of the UART ~ 6 Mbps.

2.4. UARTs 0...3


APB slave group 0
SSP1
UARTs 0 & 1
CAN 1 & 2
I²C 0 & 1
SPI0
Capture/compare timers 0 & 1
Watchdog timer
PWM1
12-bit ADC
Pin connect block
GPIO interrupt control
Ethernet PHY interface
USB interface
Ethernet 10/100 MAC
USB device, host, OTG
APB slave group 1
SSP0
UARTs 2 & 3
I2S
I²C2
Capture/compare timers 2 & 3
Repetitive interrupt timer
External interrupts
DAC
System control
Motor control PWM
Quadrature encoder

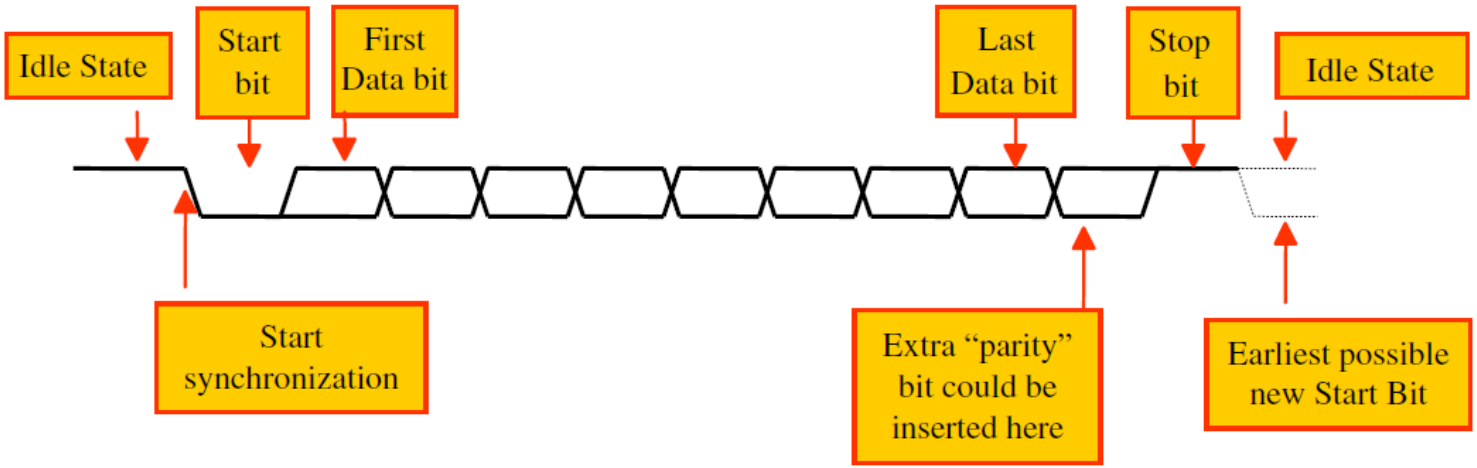
2.4. UART: Block Diagram

UART0,2,3

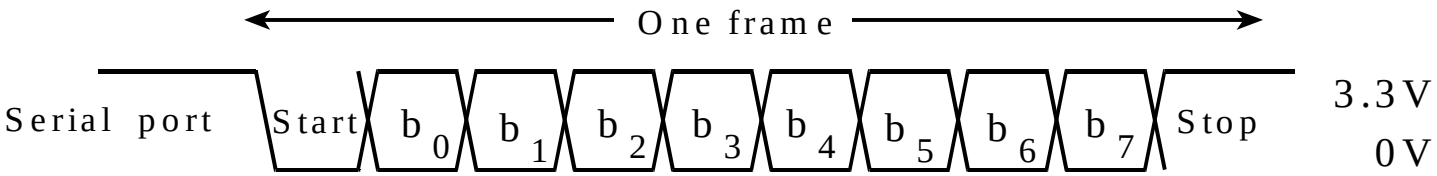


2.4. UART: TX Character Framing

- Start Bit.
- Data Bits of 5, 6, 7 or 8.
- Parity Bit.
- Stop Bit of 1, 1.5 or 2.

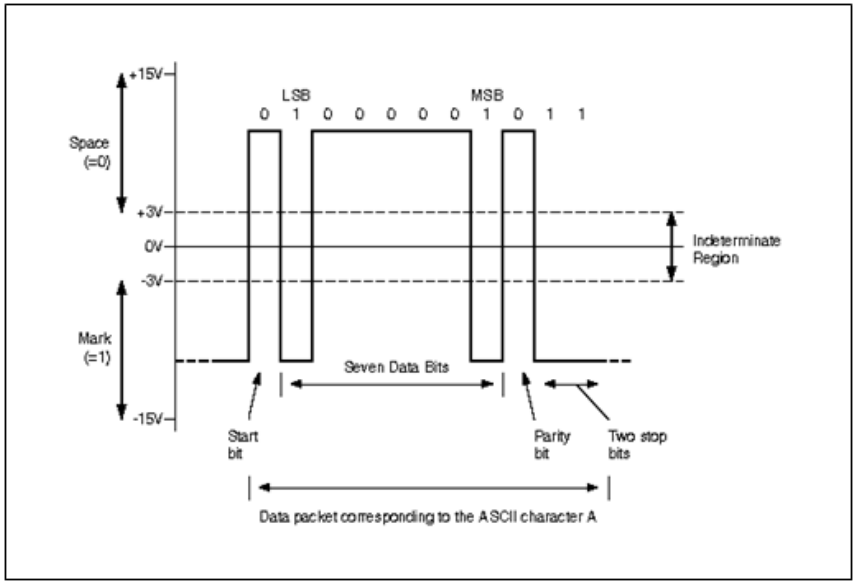


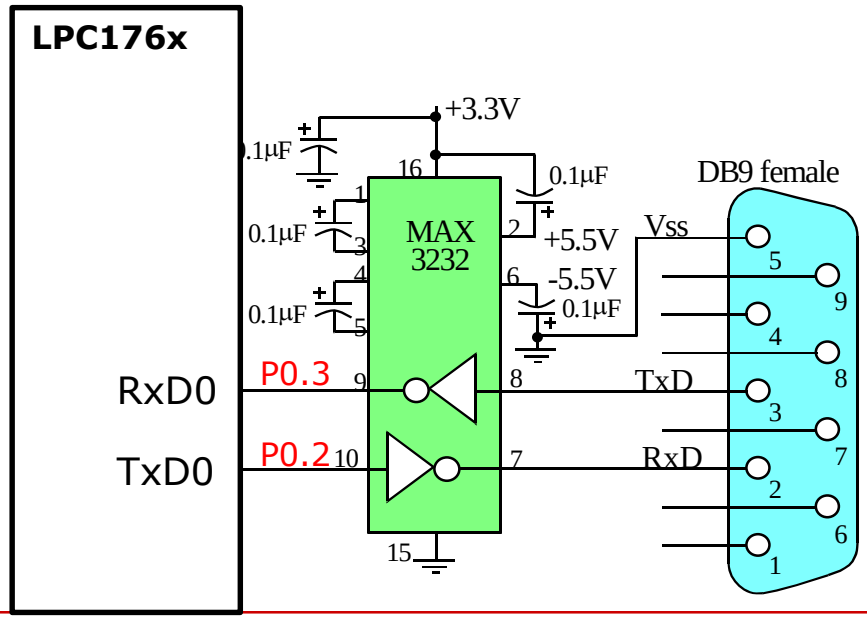
- Example: 8 bits data, No Parity, 1 stop bit



2.4. UART: Interface RS-232

- RS-232 is a popular communications interface for connecting modems and data acquisition devices (i.e. GPS receivers, electronic balances, data loggers, ...) to computers.
- RS-232 can be plugged straight into the computer's serial port (know as COM or Comm port).
- **RS-232 Line Driver**
  - Each signal (TxD, RxD) on the interface connector with reference to a signal ground.
  - The "idle" state (MARK) has the signal level negative with respect to common whereas the active state (SPACE) has the signal level positive respect to the same reference.





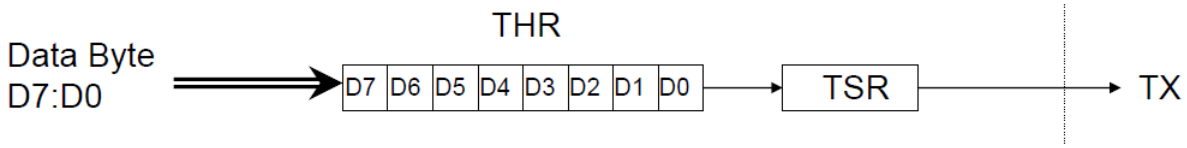
2.4. UART: Transmitter

- Parallel-to-serial conversion
- Non-FIFO Mode
  - Transmit Holding Register (THR) and Transmit Shift Register (TSR)
- FIFO Mode
  - Transmit (TX) FIFO and Transmit Shift Register (TSR)
- 16x timing for bit shifting.
- Character Framing.
- Parity Insertion.
- TX FIFO interrupt and status.

2.4. UART: Transmitter

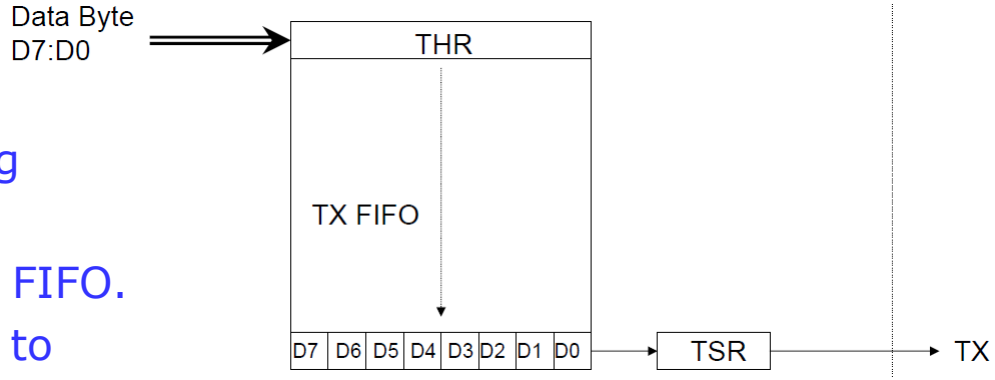
Non FIFO mode:

- Write Data to Transmit Holding Register (THR).
- Data in THR is transferred to Transmit Shift Register (TSR) when TSR is empty.
- TSR shifts the data out on the TX output pin.



FIFO mode:

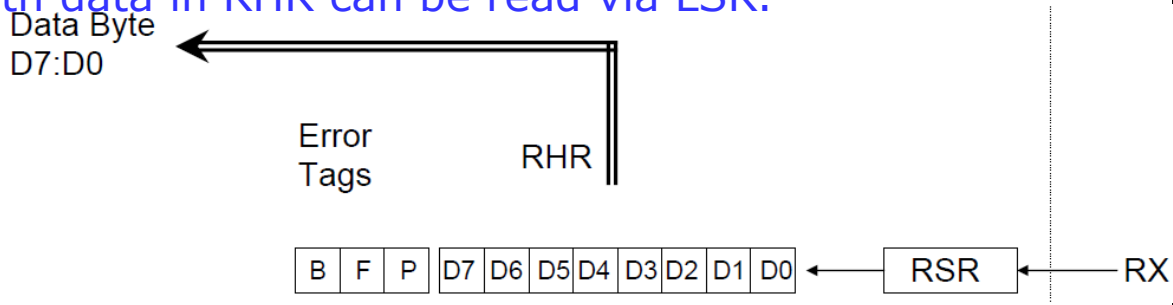
- Write Data to Transmit Holding Register (THR).
- Transmit data is queued in TX FIFO.
- Data in TX FIFO is transferred to Transmit Shift Register (TSR) when TSR is empty.
- TSR shifts data out on TX output pin.



2.4. UART: Receiver

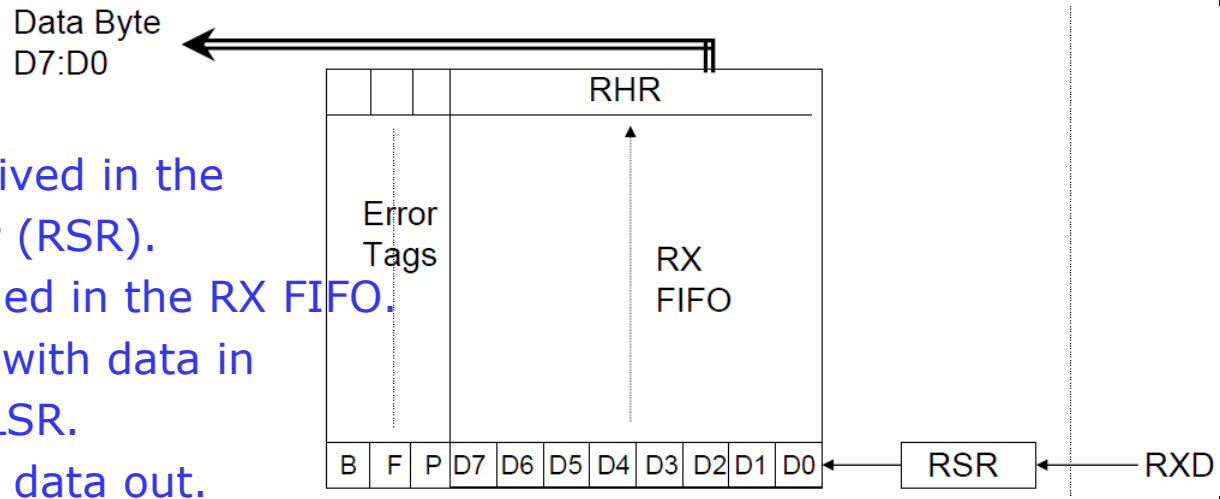
Non FIFO mode:

- Incoming data is received in the Receive Shift Register (RSR).
- Received data is transferred to the RHR.
- Error tags associated with data in RHR can be read via LSR.
- Read RHR to read the data out.



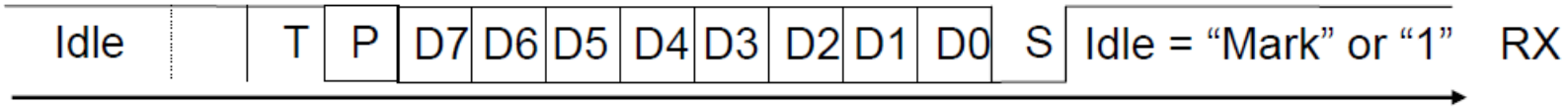
FIFO mode:

- Incoming data is received in the Receive Shift Register (RSR).
- Received data is queued in the RX FIFO.
- Error tags associated with data in RHR can be read via LSR.
- Read RHR to read the data out.



2.4. UART: RX Character Validation

- Start bit detection and validation:
  - HIGH to LOW transition indicates a start bit.
  - Start bit validated if RX input is still LOW during mid bit sampling.
- Data, parity and stop bits are sampled at mid bit.
- A valid stop bit is HIGH when the stop bit is sampled.



2.4. UART: RX Error Reporting

- **Line Status errors**
  - Error tags are associated with each byte:
    - Framing error if stop bit is not detected.
    - Parity error if parity bit is incorrect.
    - Break detected if RX input is LOW for duration of one character time and stop bit is not detected.
- Overrun error if character is received in RSR when RX FIFO is full.
  - Non-FIFO mode:
    - RHR has a data byte and data received in RSR.
    - RSR data overwrites RHR data.
  - FIFO mode:
    - RX FIFO is full and data is received in RSR.
    - Data in RX FIFO is not overwritten by data in RSR.

2.4. UART: Baudrate

Configurable baudrate:

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

- **UnDLM, UnDLL** are **8 bits data**.
- 1 ≤ **MULVAL** ≤15
- 0 ≤ **DIVADDVAL** ≤14
- DIVADDVAL ≤ MULVAL

Table 285: UARTn Fractional Divider Register (U0FDR - address 0x4000 C028, U2FDR - 0x4009 8028, U3FDR - 0x4009 C028) bit description

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

Reset

Table 286. Fractional Divider setting look-up table

FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal
1.000	0/1	1.250	1/4	1.500	1/2	1.750	3/4
1.067	1/15	1.267	4/15	1.533	8/15	1.769	10/13
1.071	1/14	1.273	3/11	1.538	7/13	1.778	7/9
1.077	1/13	1.286	2/7	1.545	6/11	1.786	11/14
1.083	1/12	1.300	3/10	1.556	5/9	1.800	4/5
1.091	1/11	1.308	4/13	1.571	4/7	1.818	9/11
1.100	1/10	1.333	1/3	1.583	7/12	1.833	5/6
1.111	1/9	1.357	5/14	1.600	3/5	1.846	11/13
1.125	1/8	1.364	4/11	1.615	8/13	1.857	6/7
1.133	2/15	1.375	3/8	1.625	5/8	1.867	13/15
1.143	1/7	1.385	5/13	1.636	7/11	1.875	7/8
1.154	2/13	1.400	2/5	1.643	9/14	1.889	8/9
1.167	1/6	1.417	5/12	1.667	2/3	1.900	9/10
1.182	2/11	1.429	3/7	1.692	9/13	1.909	10/11
1.200	1/5	1.444	4/9	1.700	7/10	1.917	11/12
1.214	3/14	1.455	5/11	1.714	5/7	1.923	12/13
1.222	2/9	1.462	6/13	1.727	8/11	1.929	13/14
1.231	3/13	1.467	7/15	1.733	11/15	1.933	14/15

- Example -> UART0 (19200 baudios)

```
LPC_UART0->LCR |= DLAB_ENABLE; // importante poner a 1
LPC_UART0->DLM =0;
LPC_UART0->DLL =81;
LPC_UART0->LCR &= ~DLAB_ENABLE;// importante poner a 0
```

$$U0DL_{16} = \frac{PCLK[Hz]}{16 \cdot V_t[baudios]} = \frac{100^6/4}{16 \cdot 19200} = 81.38$$

2.4. UART: Register Map (I)

Table 270. UART0/2/3 Register Map

Generic Name	Description	Access	Reset value[1]	UARTn Register Name & Address
RBR (DLAB =0)	Receiver Buffer Register. Contains the next received character to be read.	RO	NA	U0RBR - 0x4000 C000 U2RBR - 0x4009 8000 U3RBR - 0x4009 C000
THR (DLAB =0)	Transmit Holding Register. The next character to be transmitted is written here.	WO	NA	U0THR - 0x4000 C000 U2THR - 0x4009 8000 U3THR - 0x4009 C000
DLL (DLAB =1)	Divisor Latch LSB. Least significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0x01	U0DLL - 0x4000 C000 U2DLL - 0x4009 8000 U3DLL - 0x4009 C000
DLM (DLAB =1)	Divisor Latch MSB. Most significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0x00	U0DLM - 0x4000 C004 U2DLM - 0x4009 8004 U3DLM - 0x4009 C004
IER (DLAB =0)	Interrupt Enable Register. Contains individual interrupt enable bits for the 7 potential UART interrupts.	R/W	0x00	U0IER - 0x4000 C004 U2IER - 0x4009 8004 U3IER - 0x4009 C004
IIR	Interrupt ID Register. Identifies which interrupt(s) are pending.	RO	0x01	U0IIR - 0x4000 C008 U2IIR - 0x4009 8008 U3IIR - 0x4009 C008
FCR	FIFO Control Register. Controls UART FIFO usage and modes.	WO	0x00	U0FCR - 0x4000 C008 U2FCR - 0x4009 8008 U3FCR - 0x4009 C008
LCR	Line Control Register. Contains controls for frame formatting and break generation.	R/W	0x00	U0LCR - 0x4000 C00C U2LCR - 0x4009 800C U3LCR - 0x4009 C00C
LSR	Line Status Register. Contains flags for transmit and receive status, including line errors.	RO	0x60	U0LSR - 0x4000 C014 U2LSR - 0x4009 8014 U3LSR - 0x4009 C014

2.4. UART: Register Map (II)

Table 270. UART0/2/3 Register Map

Generic Name	Description	Access	Reset value[1]	UARTn Register Name & Address
SCR	Scratch Pad Register. 8-bit temporary storage for software.	R/W	0x00	U0SCR - 0x4000 C01C U2SCR - 0x4009 801C U3SCR - 0x4009 C01C
ACR	Auto-baud Control Register. Contains controls for the auto-baud feature.	R/W	0x00	U0ACR - 0x4000 C020 U2ACR - 0x4009 8020 U3ACR - 0x4009 C020
ICR	IrDA Control Register. Enables and configures the IrDA mode.	R/W	0x00	U0ICR - 0x4000 C024 U2ICR - 0x4009 8024 U3ICR - 0x4009 C024
FDR	Fractional Divider Register. Generates a clock input for the baud rate divider.	R/W	0x10	U0FDR - 0x4000 C028 U2FDR - 0x4009 8028 U3FDR - 0x4009 C028
TER	Transmit Enable Register. Turns off UART transmitter for use with software flow control.	R/W	0x80	U0TER - 0x4000 C030 U2TER - 0x4009 8030 U3TER - 0x4009 C030

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

2.4. UART: Rx Buffer Register ,Tx Holding Register

Table 271: UARTn Receiver Buffer Register (U0RBR - address 0x4000 C000, U2RBR - 0x4009 8000, U3RBR - 04009 C000 when DLAB = 0) bit description

Bit	Symbol	Description	Reset Value
7:0	RBR	The UARTn Receiver Buffer Register contains the oldest received byte in the UARTn Rx FIFO.	Undefined
31:8	-	Reserved, the value read from a reserved bit is not defined.	NA

Table 272: UARTn Transmit Holding Register (U0THR - address 0x4000 C000, U2THR - 0x4009 8000, U3THR - 0x4009 C000 when DLAB = 0) bit description

Bit	Symbol	Description	Reset Value
7:0	THR	Writing to the UARTn Transmit Holding Register causes the data to be stored in the UARTn transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available.	NA
31:8	-	Reserved, user software should not write ones to reserved bits.	NA

2.4. UART: Divisor Latch LSB, MSB Registers

Table 273: UARTn Divisor Latch LSB register (U0DLL - address 0x4000 C000, U2DLL - 0x4009 8000, U3DLL - 0x4009 C000 when DLAB = 1) bit description

Bit	Symbol	Description	Reset Value
7:0	DLLSB	The UARTn Divisor Latch LSB Register, along with the UnDLM register, determines the baud rate of the UARTn.	0x01
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 274: UARTn Divisor Latch MSB register (U0DLM - address 0x4000 C004, U2DLM - 0x4009 8004, U3DLM - 0x4009 C004 when DLAB = 1) bit description

Bit	Symbol	Description	Reset Value
7:0	DLMSB	The UARTn Divisor Latch MSB Register, along with the U0DLL register, determines the baud rate of the UARTn.	0x00
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

- Example set baudrate (FR=1):

$$U0DL_{16} = \frac{PCLK[Hz]}{16 \cdot V_t[baudios]}$$

```
DL=F_pclk/16*baud; // Round to the nearest whole!!!!
LPC_UART0->DLL= DL%256; //LSB
LPC_UART0->DLM= DL/256;  //MSB
```

2.4. UART: Interrupt Enable Register

Table 275: UARTn Interrupt Enable Register (U0IER - address 0x4000 C004, U2IER - 0x4009 8004, U3IER - 0x4009 C004 when DLAB = 0) bit description

Bit	Symbol	Value	Description	Reset Value
0	RBR Interrupt Enable		Enables the Receive Data Available interrupt for UARTn. It also controls the Character Receive Time-out interrupt.	0
		0	Disable the RDA interrupts.	
		1	Enable the RDA interrupts.	
1	THRE Interrupt Enable		Enables the THRE interrupt for UARTn. The status of this can be read from UnLSR[5].	0
		0	Disable the THRE interrupts.	
		1	Enable the THRE interrupts.	
2	RX Line Status Interrupt Enable		Enables the UARTn RX line status interrupts. The status of this interrupt can be read from UnLSR[4:1].	0
		0	Disable the RX line status interrupts.	
		1	Enable the RX line status interrupts.	
7:3	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
8	ABEOIntEn		Enables the end of auto-baud interrupt.	0
		0	Disable end of auto-baud Interrupt.	
		1	Enable end of auto-baud Interrupt.	
9	ABTOIntEn		Enables the auto-baud time-out interrupt.	0
		0	Disable auto-baud time-out Interrupt.	
		1	Enable auto-baud time-out Interrupt.	
31:10	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

2.4. UART: Interrupt Identification Register

Table 276: UARTn Interrupt Identification Register (U0IIR - address 0x4000 C008, U2IIR - 0x4009 8008, U3IIR - 0x4009 C008) bit description

Bit	Symbol	Value	Description	Reset Value
0	IntStatus		Interrupt status. Note that UnIIR[0] is active low. The pending interrupt can be determined by evaluating UnIIR[3:1].	1
		0	At least one interrupt is pending.	
		1	No interrupt is pending.	
3:1	IntId		Interrupt identification. UnIER[3:1] identifies an interrupt corresponding to the UARTn Rx or TX FIFO. All other combinations of UnIER[3:1] not listed below are reserved (000,100,101,111).	0
		011	1 - Receive Line Status (RLS).	
		010	2a - Receive Data Available (RDA).	
		110	2b - Character Time-out Indicator (CTI).	
		001	3 - THRE Interrupt	
5:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	FIFO Enable		Copies of UnFCR[0].	0
8	ABEOInt		End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled.	0
9	ABTOInt		Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled.	0
31:10	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

2.4. UART: Interrupt Handling

Table 277: UARTn Interrupt Handling

U0IIR[3:0] value[1]	Priority	Interrupt Type	Interrupt Source	Interrupt Reset
0001	-	None	None	-
0110	Highest	RX Line Status / Error	OE[2] or PE[2] or FE[2] or BI[2]	UnLSR Read[2]
0100 ★	Second	RX Data Available	Rx data available or trigger level reached in FIFO (UnFCR0=1)	UnRBR Read[3] or UARTn FIFO drops below trigger level
1100	Second	Character Time-out indication	Minimum of one character in the Rx FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times). The exact time will be: [(word length) × 7 - 2] × 8 + [(trigger level - number of characters) × 8 + 1] RCLKs	UnRBR Read[3]
0010 ★	Third	THRE	THRE[2]	UnIIR Read (if source of interrupt) or THR write[4]

2.4. UART: FIFO Control Register

Table 278: UARTn FIFO Control Register (U0FCR - address 0x4000 C008, U2FCR - 0x4009 8008, U3FCR - 0x4007 C008) bit description

Bit	Symbol	Value	Description	Reset Value
0	FIFO Enable	0	UARTn FIFOs are disabled. Must not be used in the application.	0
		1	Active high enable for both UARTn Rx and TX FIFOs and UnFCR[7:1] access. This bit must be set for proper UART operation. Any transition on this bit will automatically clear the related UART FIFOs.	
1	RX FIFO Reset	0	No impact on either of UARTn FIFOs.	0
		1	Writing a logic 1 to UnFCR[1] will clear all bytes in UARTn Rx FIFO, reset the pointer logic. This bit is self-clearing.	
2	TX FIFO Reset	0	No impact on either of UARTn FIFOs.	0
		1	Writing a logic 1 to UnFCR[2] will clear all bytes in UARTn TX FIFO, reset the pointer logic. This bit is self-clearing.	
3	DMA Mode Select		When the FIFO enable bit (bit 0 of this register) is set, this bit selects the DMA mode. See Section 14.4.6.1.	0
5:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	RX Trigger Level		These two bits determine how many receiver UARTn FIFO characters must be written before an interrupt or DMA request is activated.	0
		00	Trigger level 0 (1 character or 0x01)	
		01	Trigger level 1 (4 characters or 0x04)	
		10	Trigger level 2 (8 characters or 0x08)	
		11	Trigger level 3 (14 characters or 0x0E)	
31:8	-		Reserved, user software should not write ones to reserved bits.	NA

2.4. UART: Line Control Register

Table 279: UARTn Line Control Register (U0LCR - address 0x4000 C00C, U2LCR - 0x4009 800C, U3LCR - 0x4009 C00C) bit description

Bit	Symbol	Value	Description	Reset Value
1:0	Word Length Select	00	5-bit character length	0
		01	6-bit character length	
		10	7-bit character length	
		11	8-bit character length	
2	Stop Bit Select	0	1 stop bit.	0
		1	2 stop bits (1.5 if UnLCR[1:0]=00).	
3	Parity Enable	0	Disable parity generation and checking.	0
		1	Enable parity generation and checking.	
5:4	Parity Select	00	Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.	0
		01	Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.	
		10	Forced "1" stick parity.	
		11	Forced "0" stick parity.	
6	Break Control	0	Disable break transmission.	0
		1	Enable break transmission. Output pin UARTn TXD is forced to logic 0 when UnLCR[6] is active high.	
7	Divisor Latch Access Bit (DLAB)	0	Disable access to Divisor Latches.	0
		1	Enable access to Divisor Latches.	
31:8	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

2.4. UART: Line Status Register

Table 280: UARTn Line Status Register (U0LSR - address 0x4000 C014, U2LSR - 0x4009 8014, U3LSR - 0x4009 C014) 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	The UARTn receiver FIFO is empty.	
		1	The UARTn receiver FIFO is not empty.	
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.	

2.4. UART: Line Status Register (continued)

Table 280: UARTn Line Status Register (U0LSR - address 0x4000 C014, U2LSR - 0x4009 8014, U3LSR - 0x4009 C014) bit description ...continued

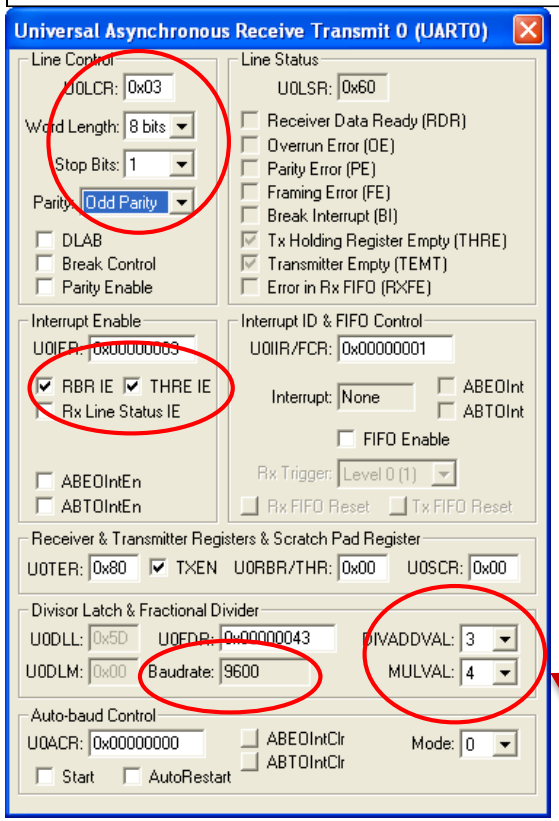
Bit	Symbol	Value	Description	Reset Value
4	Break Interrupt (BI)		When RXDn is held in the spacing state (all zeroes) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXDn goes to marking state (all ones). 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.	
31:8	-		Reserved, the value read from a reserved bit is not defined.	NA

2.4. UART: Software example (I)

```
void uart0_init(int baudrate) {

    LPC_PINCON->PINSEL0 = (1 << 4) | (1 << 6);     // Change P0.2 and P0.3 mode to TXD0 and RXD0

    // Set 8N1 mode (8 bits/dato, sin paridad, y 1 bit de stop)
    LPC_UART0->LCR |= CHAR_8_BIT | STOP_1_BIT | PARITY_NONE;

    uart0_set_baudrate(baudrate);                  // Set the baud rate

    LPC_UART0->IER = THRE_IRQ_ENABLE | RBR_IRQ_ENABLE; // Enable UART TX and RX interrupt (for LPC17xx UART)
    NVIC_EnableIRQ(UART0_IRQn);                    // Enable the UART interrupt (for Cortex-CM3 NVIC)

}
```

```
LPC_UART0->LCR |= DLAB_ENABLE; // importante poner a 1
LPC_UART0->DLM = DL / 256; // parte alta
LPC_UART0->DLL  = DL % 256; // parte baja
LPC_UART0->LCR &= ~DLAB_ENABLE; // importante poner a 0
```

if (FR=1)
DL=162.73
Baudrate= 9585 bps



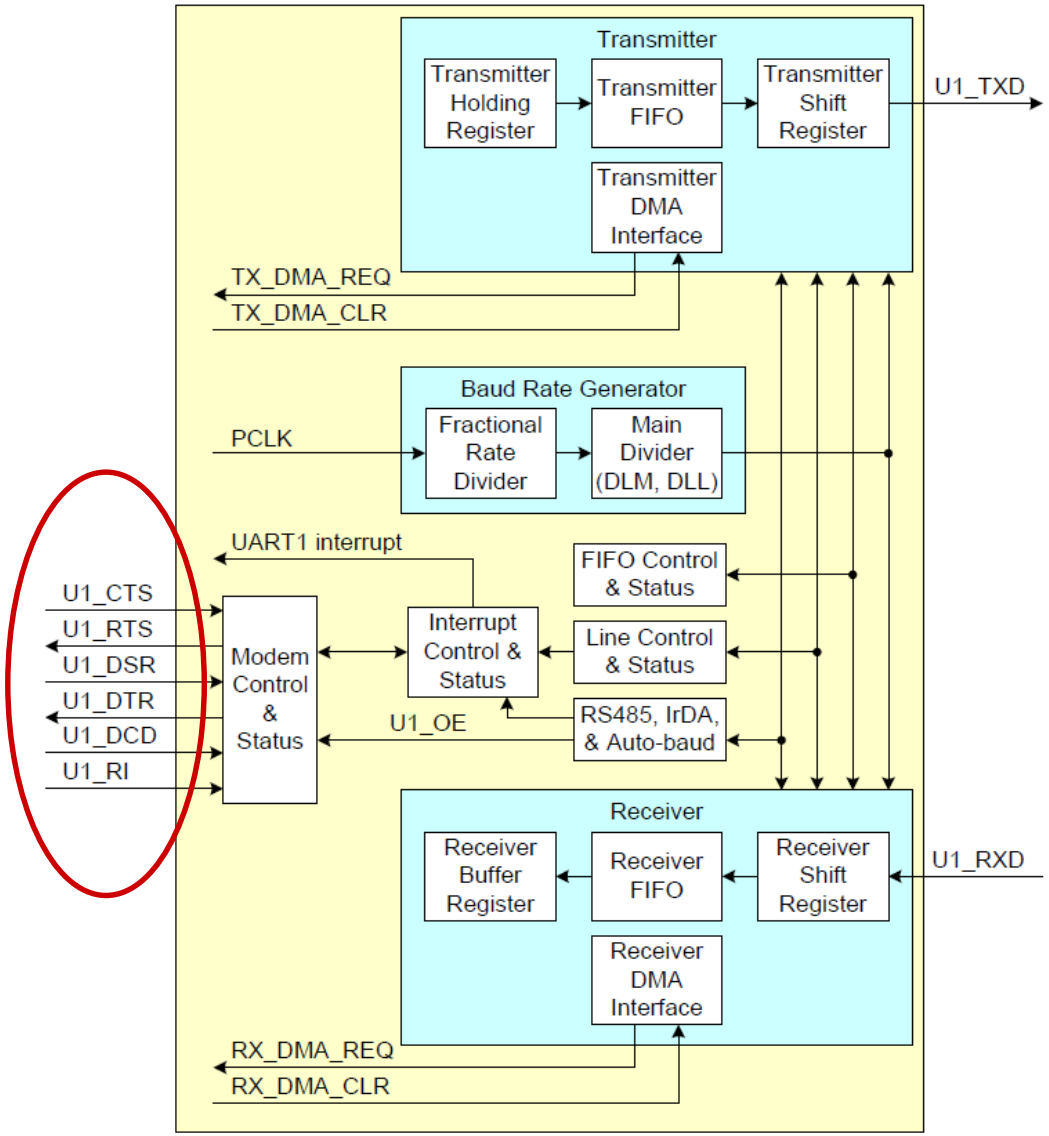

```
static int uart0_set_baudrate(unsigned int baudrate) {
    int errorStatus = -1; //< Failure

    // UART clock (FCCO / PCLK_UART0)
    // unsigned int uClk = SystemFrequency / 4;
    unsigned int uClk =SystemCoreClock/4;
    unsigned int calcBaudrate = 0;
    unsigned int temp = 0;

    unsigned int mulFracDiv, dividerAddFracDiv;
    unsigned int divider = 0;
    unsigned int mulFracDivOptimal = 1;
    unsigned int dividerAddOptimal = 0;
    unsigned int dividerOptimal = 0;

    unsigned int relativeError = 0;
    unsigned int relativeOptimalError = 100000;

    uClk = uClk >> 4; /* div by 16 */

    /*
     * The formula is :
     * BaudRate= uClk * (mulFracDiv/(mulFracDiv+dividerAddFracDiv) / (16 * DLL)
     *
     * The value of mulFracDiv and dividerAddFracDiv should comply to the following expressions:
     * 0 < mulFracDiv <= 15, 0 <= dividerAddFracDiv <= 15
     */
    for (mulFracDiv = 1; mulFracDiv <= 15; mulFracDiv++) {
        for (dividerAddFracDiv = 0; dividerAddFracDiv <= 15; dividerAddFracDiv++) {
            temp = (mulFracDiv * uClk) / (mulFracDiv + dividerAddFracDiv);

            divider = temp / baudrate;
            if ((temp % baudrate) > (baudrate / 2))
```

2.4. UART: Software example (I)

```
void tx_cadena_UART0(char *cadena)
{
   ptr_tx=cadena;
   tx_completa=0;
   LPC_UART0->THR = *ptr_tx;   // IMPORTANTE: Introducir un carácter al comienzo para iniciar TX o
}                              // activar flag interrupción por registro transmisor vacio
```

```
void UART0_IRQHandler(void) {

   switch(LPC_UART0->IIR&0x0E) {

   case 0x04:                  /* RBR, Receiver Buffer Ready */
     *ptr_rx=LPC_UART0->RBR;   /* lee el dato recibido y lo almacena */
     if(*ptr_rx++ ==13){       /* Caracter return --> Cadena completa */
       *ptr_rx=0;              /* Añadimos el caracter null para tratar los datos recibidos como una cadena*/
       rx_completa = 1;        /* rx completa */
       ptr_rx=buffer;          /* puntero al inicio del buffer para nueva recepción */
     }
     break;

   case 0x02:                      /* THRE, Transmit Holding Register empty */
     if(*ptr_tx!=0)
       LPC_UART0->THR = *ptr_tx++; /* carga un nuevo dato para ser transmitido */
     else
       tx_completa=1;
     break;
   }
}
```

2.4. UART1: Modem



2.4. UART1: Interface RS-485

What is RS-485?

- RS-485 is a EIA standard interface which is very common in the data acquisition world.
- RS-485 provides balanced transmission line which also can be shared in Multi-drop mode.
- It allows high data rates communications over long distances in real world environments.

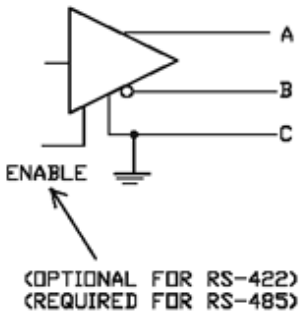
How fast can RS-485 be?

- RS-485 was designed for greater distance and higher baudrates than RS-232.
- According to the standard, 100kbit/s is the maximum speed and distance up to 4000 feet (**1200 meters**) can be achieved.

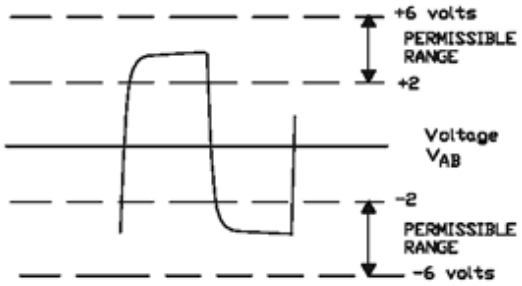
2.4. UART1: RS-485 Line Driver

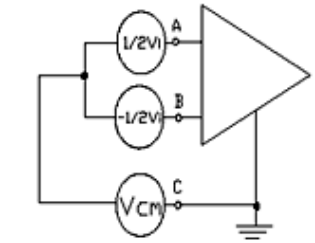
Balanced Line Drivers

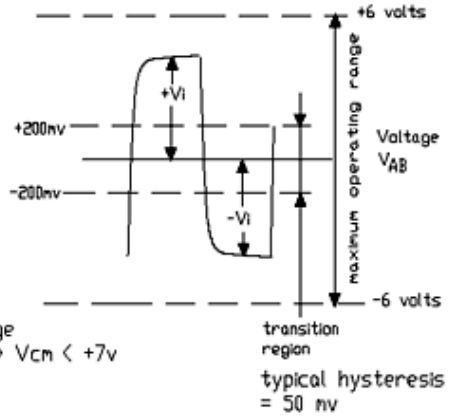
- Voltage produced by the driver appears across **a pair of signal wires** that transmit only one signal. Both wires are driven opposite.
- RS-485 driver has always the "Enable" direction control signal.
- **Differential system provides noise immunity**, because much of the common mode signal can be rejected by the receiver. So ground shifts and induced noise signals can be nullified.



BALANCED DIFFERENTIAL OUTPUT
LINE DRIVER





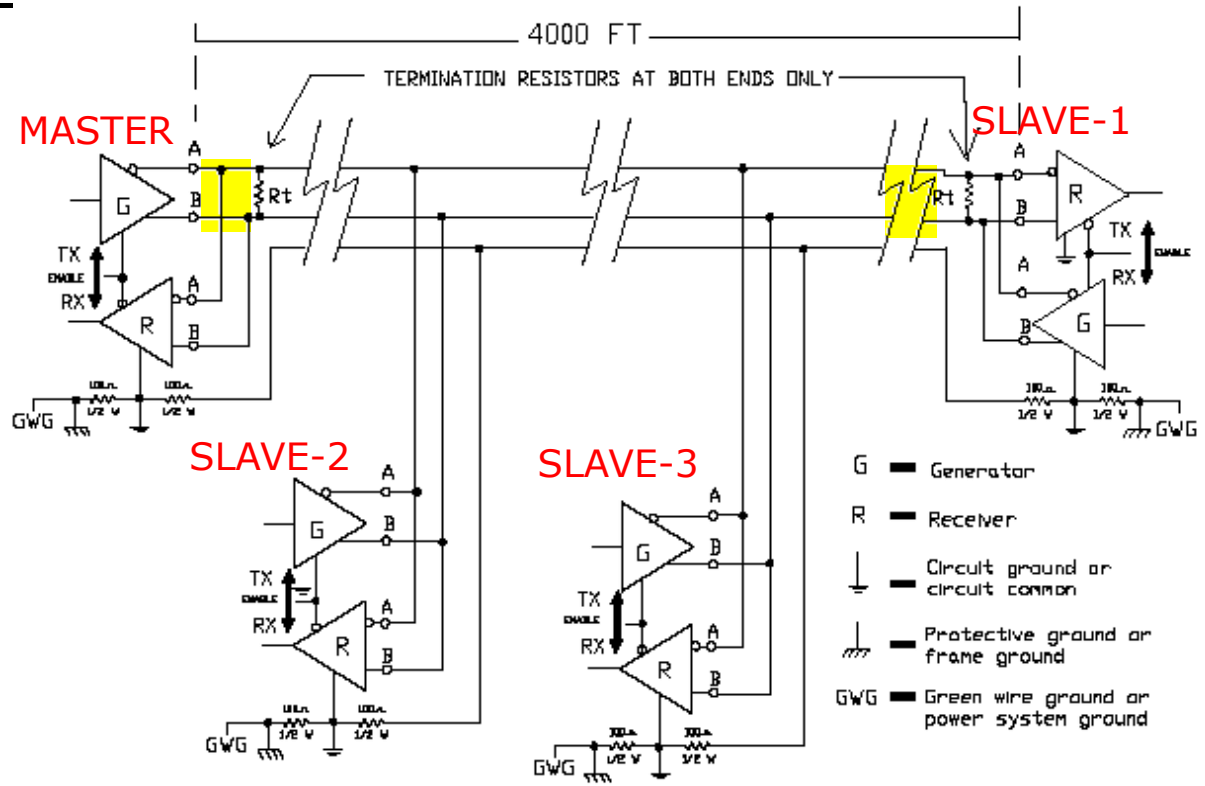


Vcm = Input Common Mode Voltage
Permissible Range for Vcm: -7v > Vcm < +7v

BALANCED DIFFERENTIAL INPUT
LINE RECEIVER

2.4. UART1: RS-485 Network

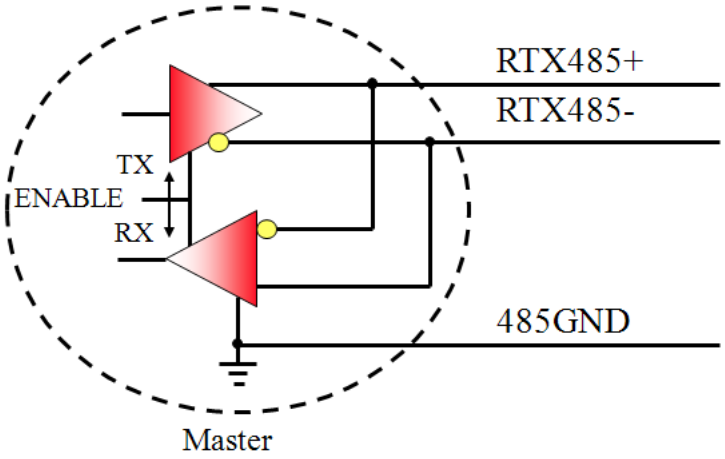
- RS-485 provides Half-Duplex, Multi-drop communications over a single twisted pair cable.
  - The standard specifies up to 32 drivers and 32 receivers can share a multidrop network.
  - Terminator resistors avoid reflected signal

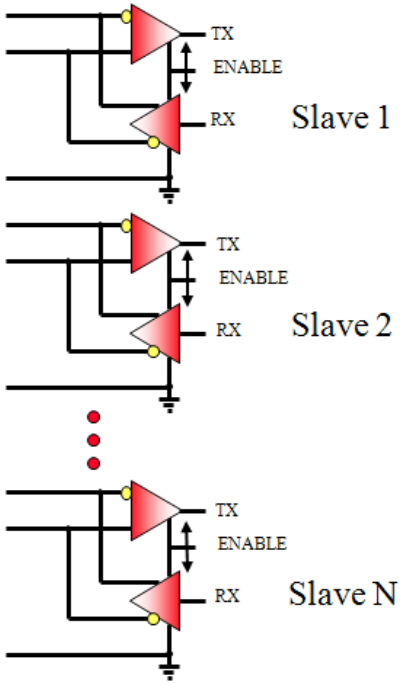


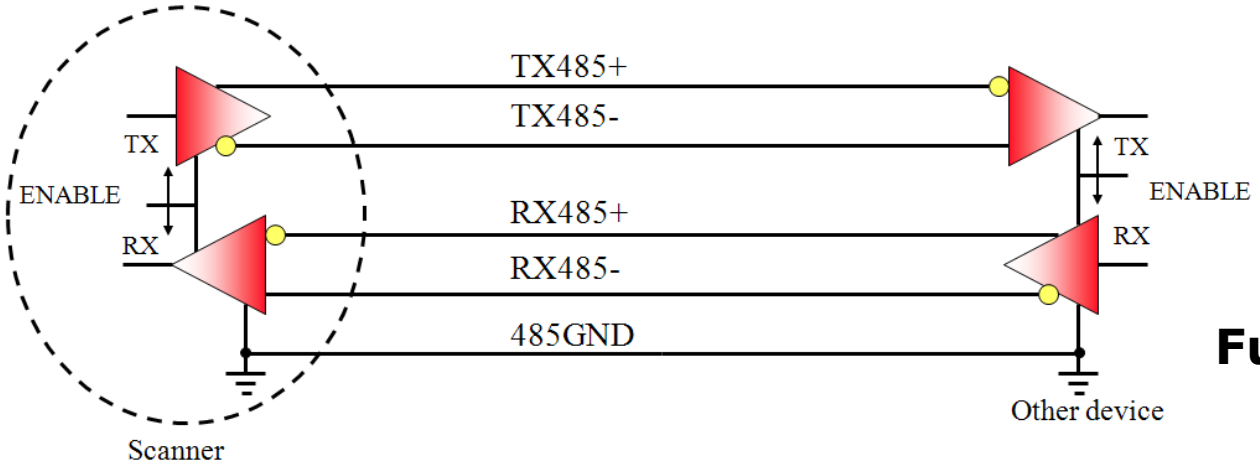
TYPCIAL RS-485 TWO WIRE MULTIDROP NETWORK

2.4. UART1: RS-485 Half and Full Duplex

Half-Duplex







Full-Duplex

2.4. UART1: RS-485 vs RS-232

	RS-232	RS-485
Mode of Operation	SINGLE-ENDED	DIFFERENTIAL
Total Number of Drivers and Receivers on One Line	1 DRIVER 1 RECEIVER	32 DRIVER 32 RECEIVER
Maximum Cable Length	50 FEET	4000 FEET
Maximum Data Rate @Max length	20kb/s	100kb/s
Driver Output Signal Level (Loaded Min.) Loaded	+/-5V to +/-15V	+/-1.5V
Driver Output Signal Level (Unloaded Max) Unloaded	+/-25V	+/-6V
Driver Load Impedance	3kΩ to 7kΩ	54Ω
Max. Driver Current in High Z State Power On	N/A	N/A
Max. Driver Current in High Z State Power Off	+/-6mA @ +/-2v	+/-100uA
Slew Rate (Max.)	30V/μS	N/A
Receiver Input Voltage Range	+/-15V	-7V to +12V
Receiver Input Sensitivity	+/-3V	+/-200mV
Receiver Input Resistance	3kΩ to 7kΩ	≥ 12kΩ

2.4. UART1: RS-485 modes of operation

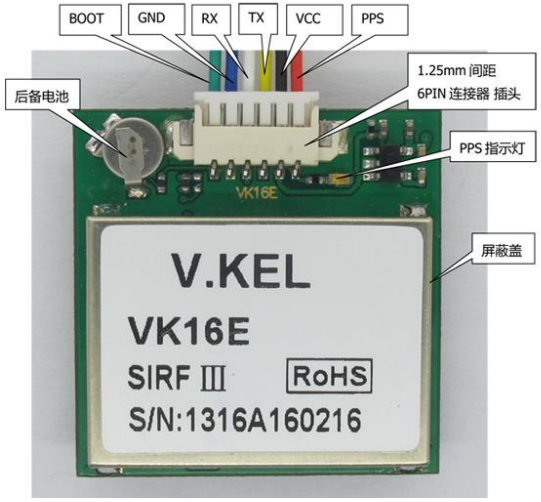
- Normal Multi-drop Mode (NMM)
- Auto Address Detection (AAD)
- Auto Direction Control
- Output inversion

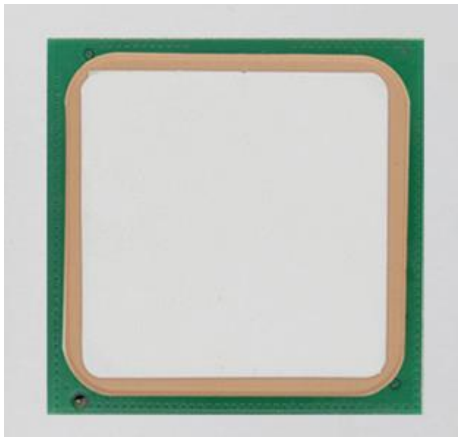
Table 308: UART1 RS485 Control register (U1RS485CTRL - address 0x4001 004C) bit description

Bit	Symbol	Value	Description	Reset value
0	NMMEN	0	RS-485/EIA-485 Normal Multidrop Mode (NMM) is disabled.	0
		1	RS-485/EIA-485 Normal Multidrop Mode (NMM) is enabled. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt.	
1	RXDIS	0	The receiver is enabled.	0
		1	The receiver is disabled.	
2	AADEN	0	Auto Address Detect (AAD) is disabled.	0
		1	Auto Address Detect (AAD) is enabled.	
3	SEL	0	If direction control is enabled (bit DCTRL = 1), pin $\overline{\text{RTS}}$ is used for direction control.	0
		1	If direction control is enabled (bit DCTRL = 1), pin DTR is used for direction control.	
4	DCTRL	0	Disable Auto Direction Control.	0
		1	Enable Auto Direction Control.	

2.4. UART: Devices with RS-232 interface

- **GPS Module** (VK16E GMOUSE GPS Module SIRF III)





- Built with fast positioning and the ability to track 20 satellites SIRF III generation chip.
- Built-in backup battery.
- Built-in high gain LNA.
- With selectable baud rate: 4800,**9600**,19200,38400
- NMEA format (ASCII):
  - RMC, GSV, GSA, GGA, VTG, GLL.

    Data example: in Polytechnic School !!!

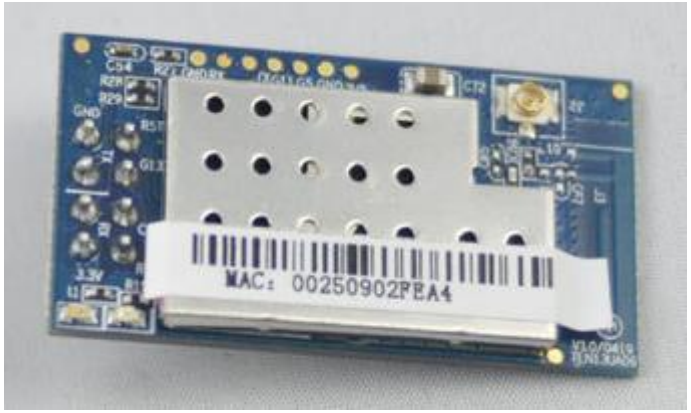
```
$GPGGA,104951.000,4030.8105,N,00321.0170,W,1,04,3.9,584.8,M,51.7,M,,0000*47
$GPRMC,104951.000,A,4030.8105,N,00321.0170,W,0.00,182.93,190214,,,A*77
$GPVTG,182.93,T,,M,0.00,N,0.0,K,A*0C
$GPGGA,104952.000,4030.8105,N,00321.0170,W,1,04,3.9,584.8,M,51.7,M,,0000*44
$GPRMC,104952.000,A,4030.8105,N,00321.0170,W,0.00,182.93,190214,,,A*74
$GPVTG,182.93,T,,M,0.00,N,0.0,K,A*0C
```

- The default output for the 9600 baud standard.

2.4. UART: Devices with RS-232 interface

2.4G Wireless WIFI Module (TLG10UA03)

- Full support for serial transparent data transfer mode, really Positive Serial Plug and Play.
- The new AT command set, all based on ASCII.
- Complete TCP / IP protocol stack to support DHCP protocol and DNS dynamic IP address assignment domain name resolution function.
- Built-in WEB server, implemented using IE browser.
- Remote configuration via the wireless network module parameters.
- Supports Frequency range: 2.412 ~ 2.484 GHz.
- Supports two types of wireless networks:
  Infrastructure Network (Infra) and ad hoc networks (Adhoc).
- Support multiple security authentication mechanisms:
  WEP64/WEP128 / TKIP / CCMP (AES)
  WEP/WPA-PSK/WPA2-PSK.
- Support for fast networking.
- Supports wireless roaming.



- Support for multiple network protocols: TCP / UDP / ICMP / DHCP / DNS / HTTP
- Supports automatic two operating mode and command support transparent transmission mode.
- Support AT command set controls.
- Support for a variety of parameter configuration mode: Serial / WEB server / Wireless connection.

2.4. UART: Devices with RS-232 interface

Digital Compass (HMR3200/HMR3300)

• The HMR3200 is a two-axis precision compass with three orthogonal magnetoresistive sensors, and can be used in either vertical or horizontal orientations.

• The HMR3300 includes a MEMS accelerometer for a horizontal three-axis, tilt compensated precision compass for performance up to a ±60° tilt range.

Optical Fingerprint Sensor (EM404)

• EM404 integrated fingerprint verification module is the latest release of HF&CCTV. It consists of optical fingerprint sensor, high performance DSP processor and Flash.

• It boasts of functions such as fingerprint enrollment, fingerprint deletion, fingerprint verification, fingerprint upload, fingerprint download, etc.

2.4. UART: Devices with RS-232 interface

Camera module (CMOS 1/4 inches Image Sensor JPG)

• **Command instruction:**

•1 Reset command: 56002600

•2 Photographing command: 5600360100

•3 Read the data length of the captured image: 5600340100
Return: 76 00 34 00 04 00 00 XX YY
XX YY ------- picture data length, XX is the high byte, YY is the low byte.

•4 Read the picture data: 56 00 32 0C 00 0A 00 00 XX XX 00 00 YY YY ZZ ZZ
Return: 7600320000 (interval) FF D8. . . . . FF D9 (interval) 7600320000
00 00 XX XX ------ starting address (starting address must be a multiple of 8, generally 00 00)
00 00 YY YY ------ picture data length (high byte first, then low byte) ZZ ZZ -------- interval (= XX XX * 0.01 milliseconds, preferably a small number, such as 00 0A)
Note: JPEG picture file data must begin with FF D8, and end with FF D9.

•5 Stop shooting: 5600360103

•6 Set camera image compression command: 56 00 31 05 01 01 12 04 XX
XX generally is 36 (range: 00 ---- FF)

•7 Set the camera image size: (default size: 320 * 240)
320 * 240: 56 00 31 05 04 01 00 19 11
640 * 480: 56 00 31 05 04 01 00 19 00

•8 Get into sleep mode: 56 00 3E 03 00 01 01

•9 Modify baud rate: 56 00 24 03 01 XX XX

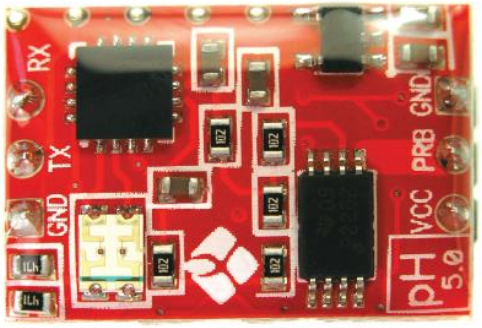
2.4. UART: Devices with RS-232 interface

MLX90614 Infrared Thermometer Module

Bluetooth to Serial Slave

Fingerprint Module Capacitive Sensor Em401

TC35 SMS Module Board



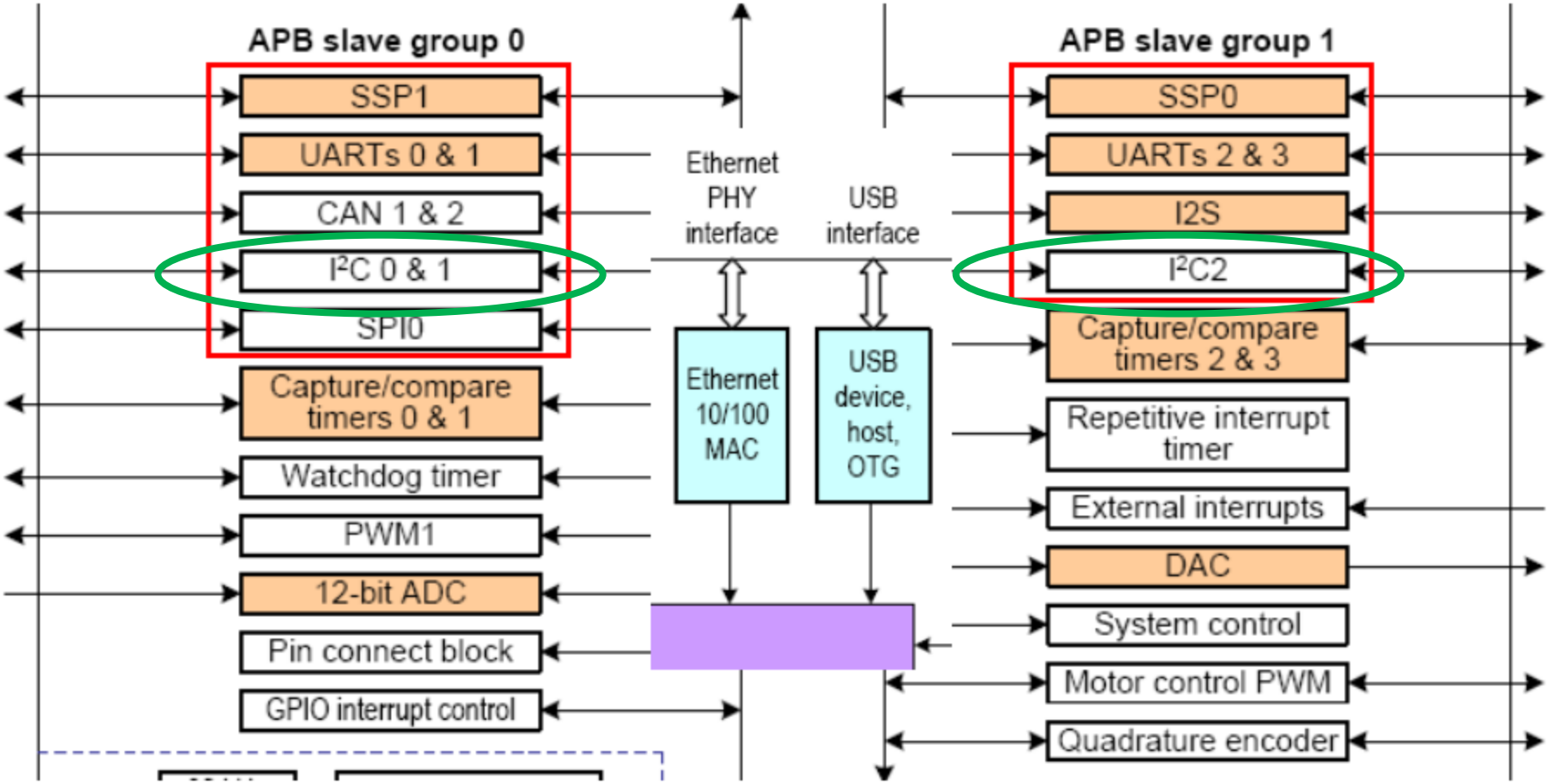
pH sensor Module (AtlasScientific)

NFC Reader Module (pn532)

2.4. Serial Interfaces

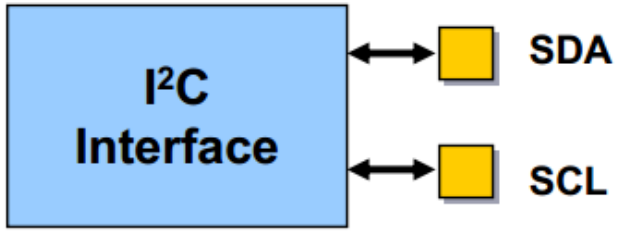
I2C, SPI/SSP, I2S, CAN, USB

2.4. Serial Interfaces: I2C



2.4. I2C introduction

- I2C bus = **I**nter-**IC** bus.
- Bus developed by Philips in the 80's
- Simple bi-directional 2-wire bus:
  - **S**erial **DA**ta (**SDA**)
  - **S**erial **CL**ock (**SCL**)

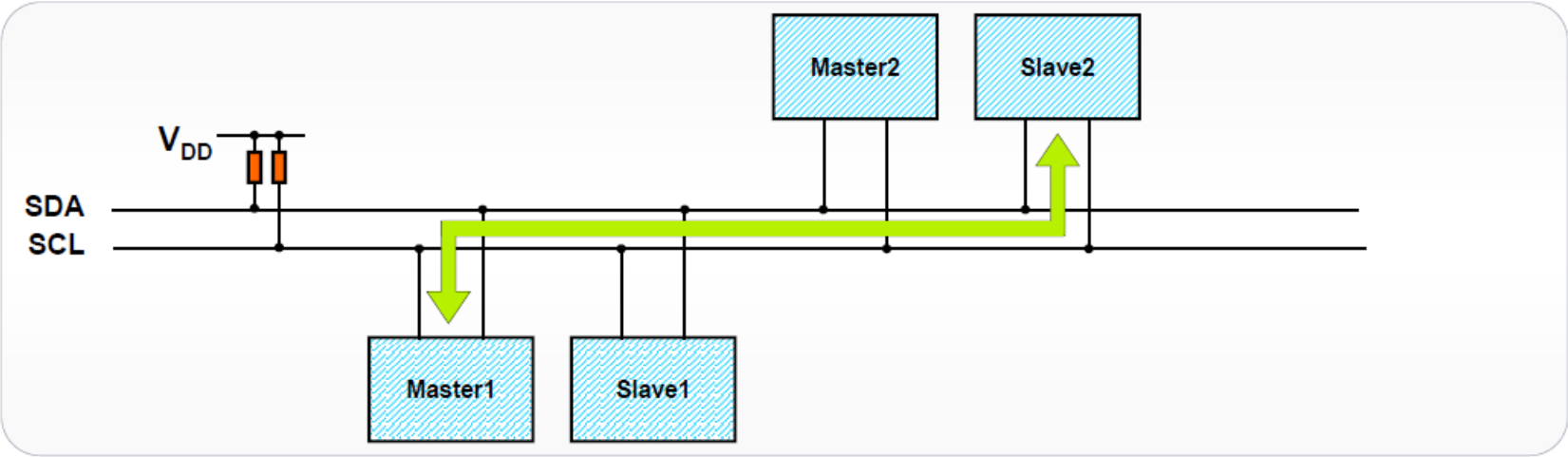


- Multi-master capable bus with arbitration feature.
- Master-Slave communication; Two-device only communication.
- Each IC on the bus is identified by its own **address code**.
- I2C compliant bus interface, and can be configured as Master, Slave, or Master/Slave.
- Bi-directional data transfer between masters and slaves.
- Programmable clock to allow adjustment of I2C transfer rates.

Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. I2C: Hardware architecture

- Multiple master.
- Multiple slave.
- Bi-directional
  - Master-transmitter
  - Master-receiver
  - Slave-transmitter
  - Slave-receiver
- Data collision is taken care off.



2.4. I2C: Modes (Speed)

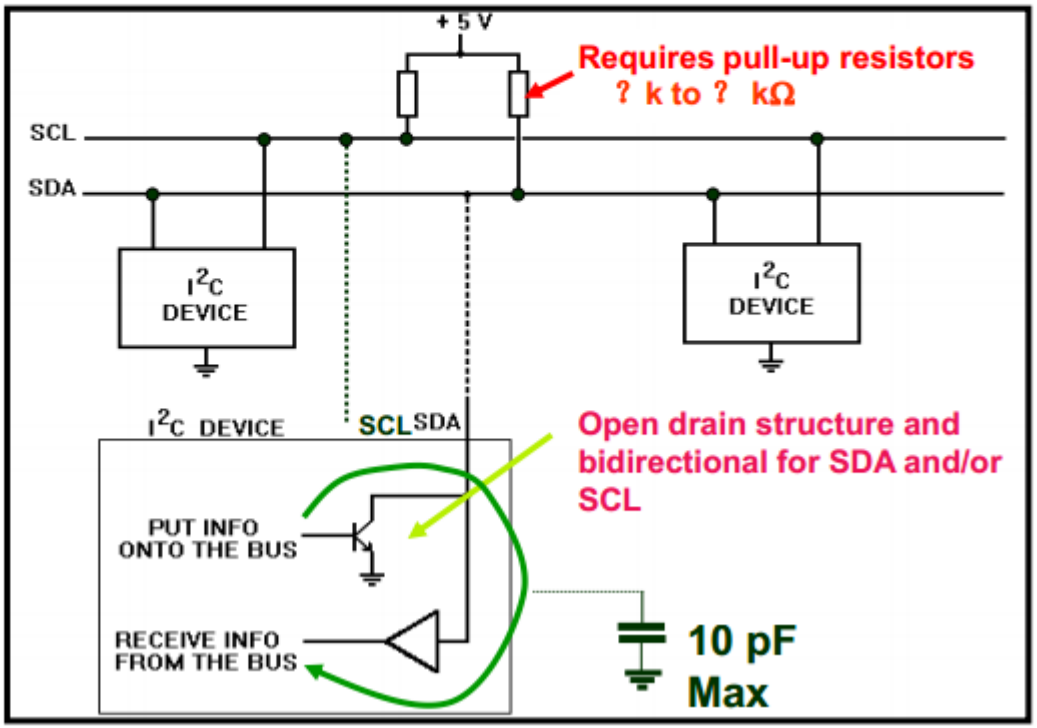
	Standard Mode	Fast Mode	Fast Mode Plus (FM+)	High Speed Mode	
Bitrate (kBit/s)	0 – 100	0 – 400	0 – 1000	0 – 1700	0 – 3400
Address (bits)	7 (10)	7 (10)	7 (10)	7 (10)	7 (10)
Capacitive Bus Load (pF)	400	400	550	400	100
Sink current (mA)	3	3	20	3	3

- Fast mode Plus (FM+):
  - Increased bandwidth
  - Increased transmission distance (at reduced bandwidth: >> 550 pF bus load)

Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. I2C: SDA/SCL Driver Architecture

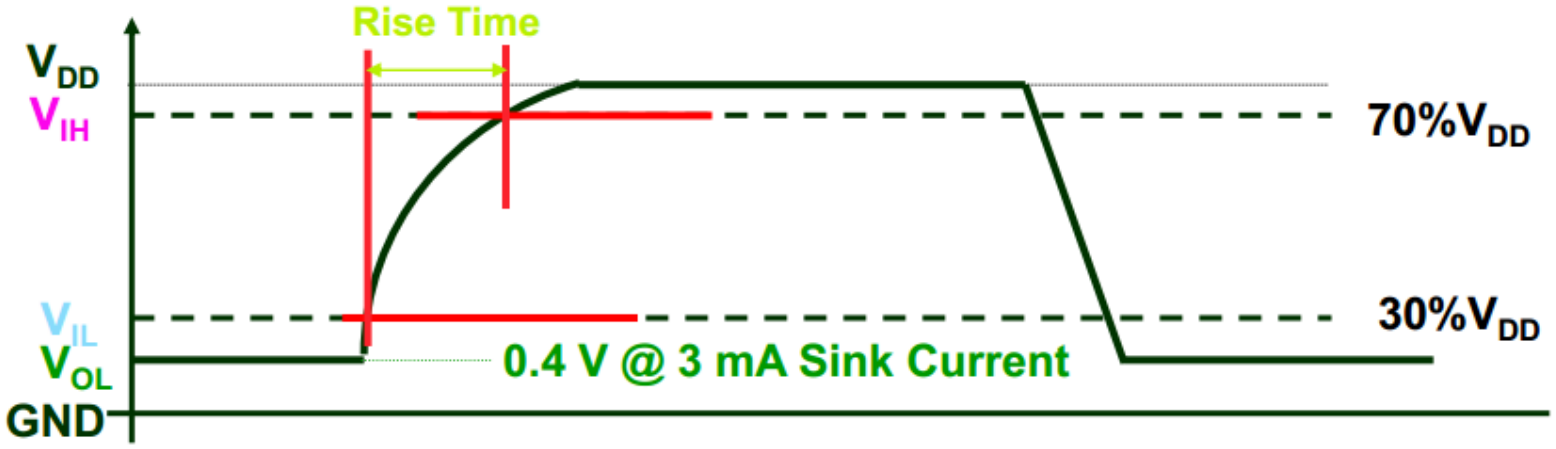
- SDA and SCL are open drain/collector
  - Required pull-up resistors to pull the line to logic “1”
- Clock stretch is possible when a device is busy



Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. I2C: Key Electrical Parameters

	Standard Mode	Fast Mode	Fast Mode Plus	High Speed Mode	
Bit Rate (kb/s)	0 to 100	0 to 400	0 to 1000	0 to 1700	0 to 3400
Max Load (pF)	400	400	560	400	100
Rise time (ns)	1000	300	120	160	80
Noise filter (ns)	-	50	10	10	10



Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. I2C: Calculating Pull-up Resistors

1) $R_{MIN} < R_{PU} < R_{MAX}$

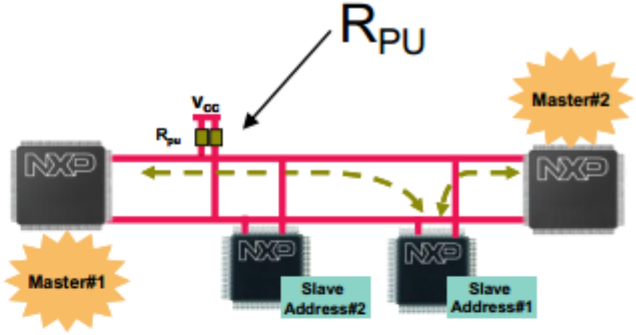
2) $R_{MIN} = (V_{DDMAX} - V_{OLMAX}) / I_{OLMAX}$

V_{DDMAX}	V_{OLMAX}	R_{MIN}		
		I_{OLMAX} = 3mA	I_{OLMAX} = 6mA*	I_{OLMAX} = 30mA**
3.6 V	0.4 V	**1.1 kΩ**	**533 Ω**	**106 Ω**
5.5 V	0.4 V	**1.7 kΩ**	**850 Ω**	**170 Ω**

*With a buffer; **Fast-mode Plus

3) $R_{MAX} * C_{MAX} = 1.18 * t_r$

MODE	Frequency	t_r	C_{MAX}	R_{MAX}
Standard	100 kHz	1000 ns	400 pF	**2.96 kΩ**
Fast Mode	400 kHz	300 ns	400 pF	**885 Ω**
Fast Mode Plus	1000 kHz	120 ns	560 pF	**252 Ω**

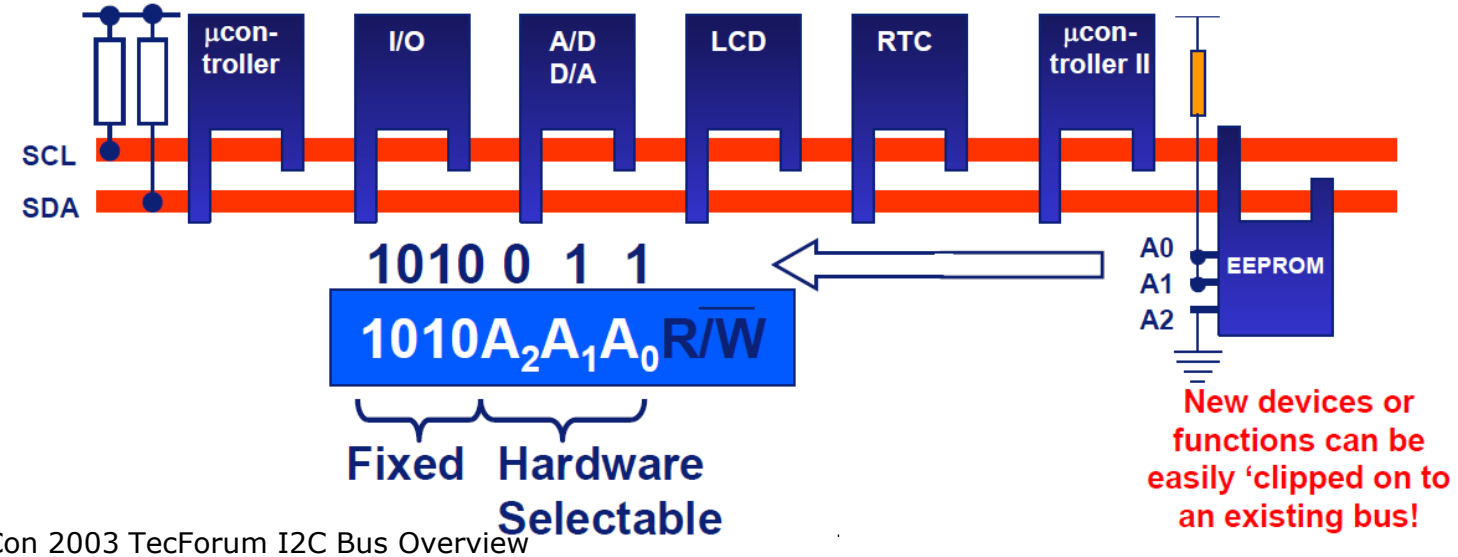


Glossary

- R_{PU}: Pull-up resistor
- R_{MIN}: Minimum pull-up resistor
- R_{MAX}: Maximum pull-up resistor
- V_{DDMAX}: Maximum supply rail
- V_{OLMAX}: Maximum output voltage low
- I_{OLMAX}: Maximum sink current
- C_{MAX}: Maximum load capacitance
- t_r: Rise time

2.4. I2C: Address, Basics

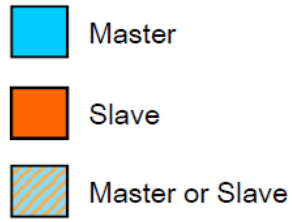
- Each device is addressed individually by software.
- Unique address per device: fully fixed or with a programmable part through hardware pin(s).
- Programmable pins mean that several same devices can share the same bus.
- Address allocation coordinated by the I2C-bus committee.
- 112 different types of devices max with the 7-bit format (others reserved)



Source: DesignCon 2003 TecForum I2C Bus Overview

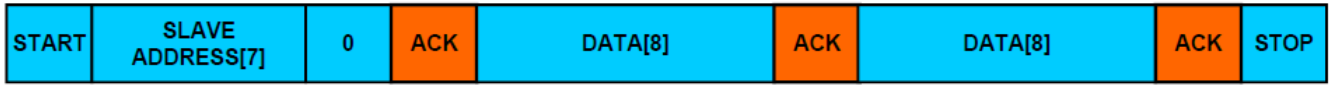
2.4. I2C: Protocol (I)

- Communication must start with: **START** condition.
- Start bit is always followed by **slave address**.
- Slave address is followed by **READ** or **NOT-WRITE** bit.
- The receiving device (either master or slave) must send an **Acknowledged** bit (**ACK**).
- Communication must end with: **STOP** condition.



- Example:

Transmit (0 = Write)

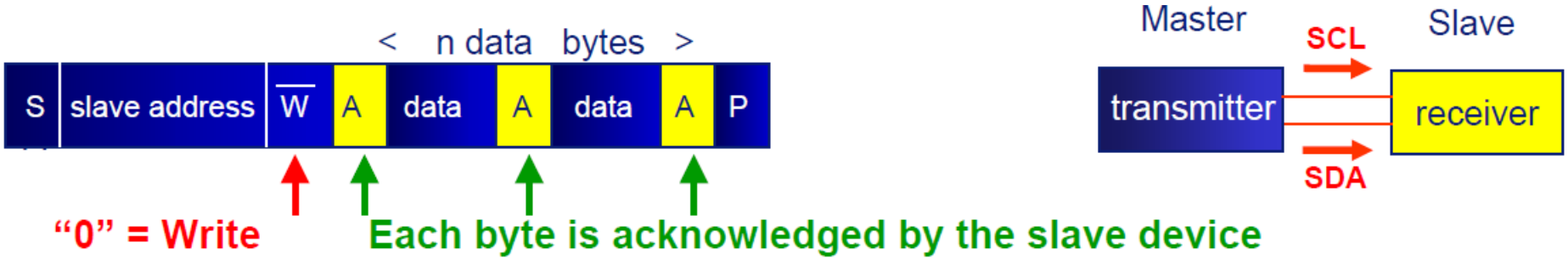


Receive (1 = Read)

2.4. I2C: Protocol (II)

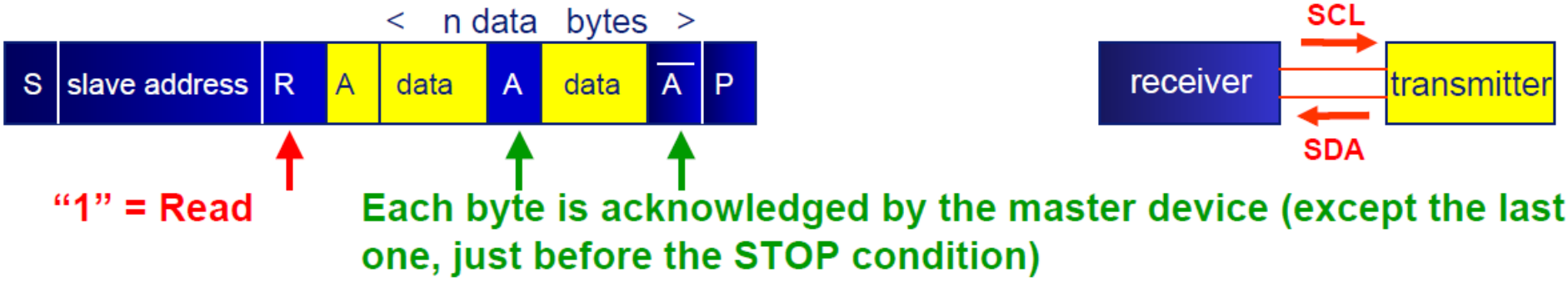
Write to a Slave device:

- The master is a "MASTER -TRANSMITTER":
  - it transmits both Clock and Data during the all communication.



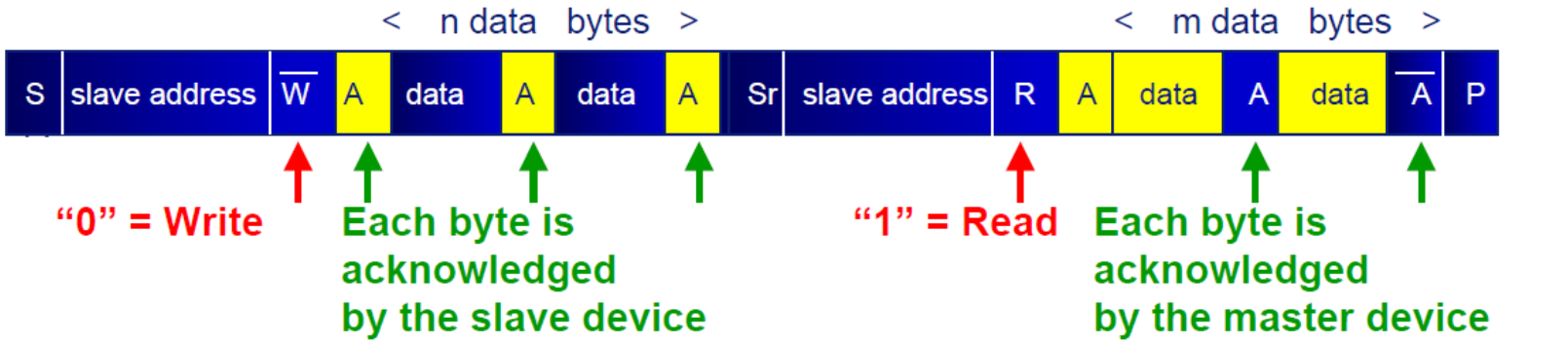
Read from a Slave device:

- The master is a "MASTER TRANSMITTER then MASTER -RECEIVER":
  - it transmits Clock all the time.
  - it sends slave address data and then becomes a receiver.

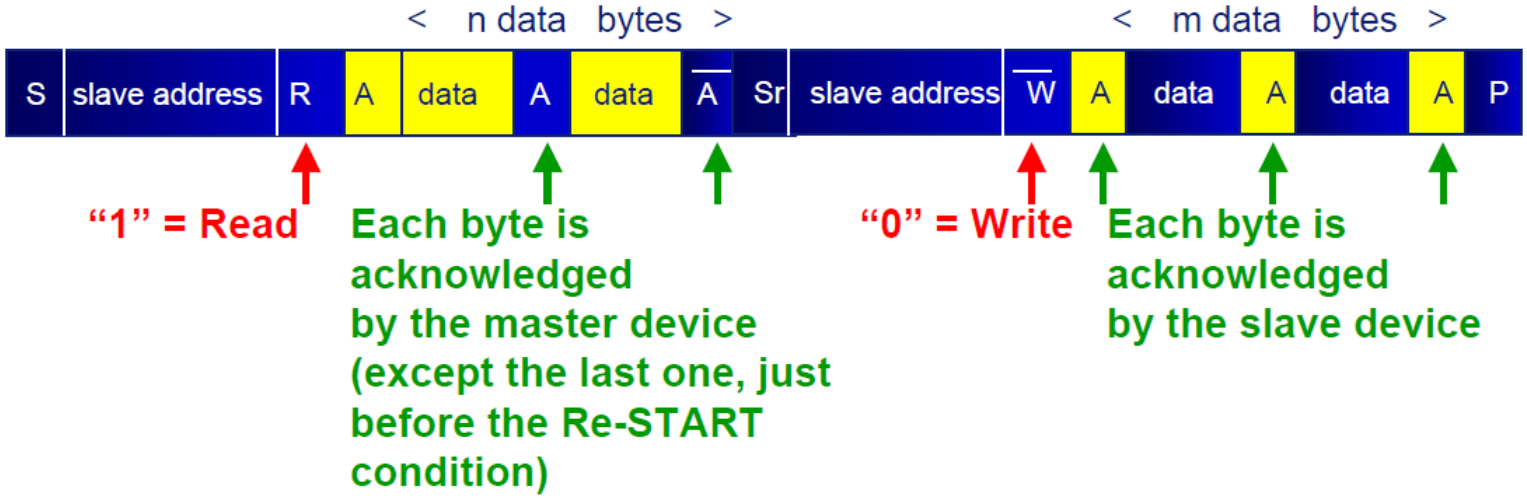


2.4. I2C: Protocol (III)

- **Combined Write and Read**



- **Combined Read and Write**

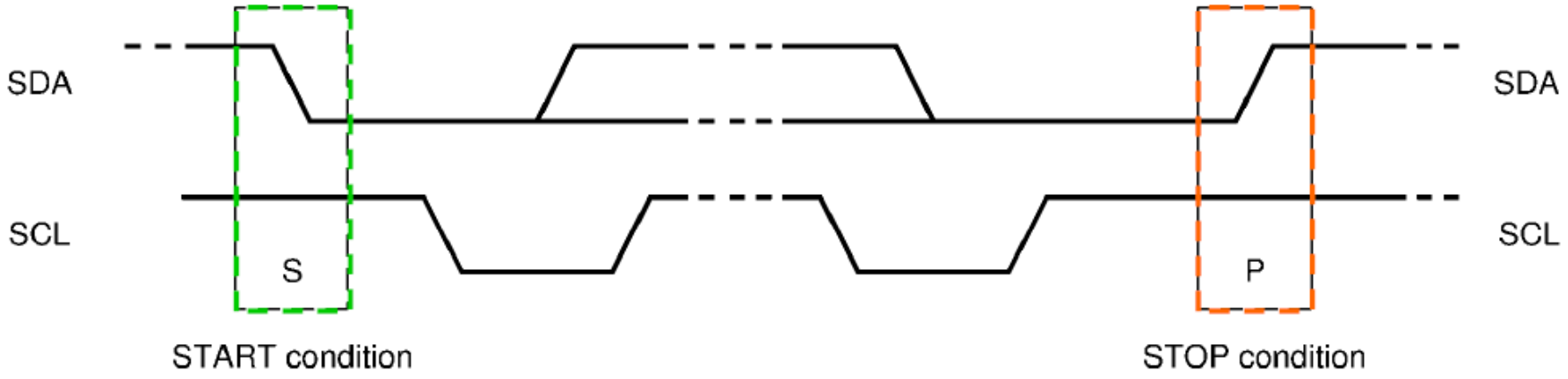


Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. I2C: Protocol (IV)

START & STOP conditions.

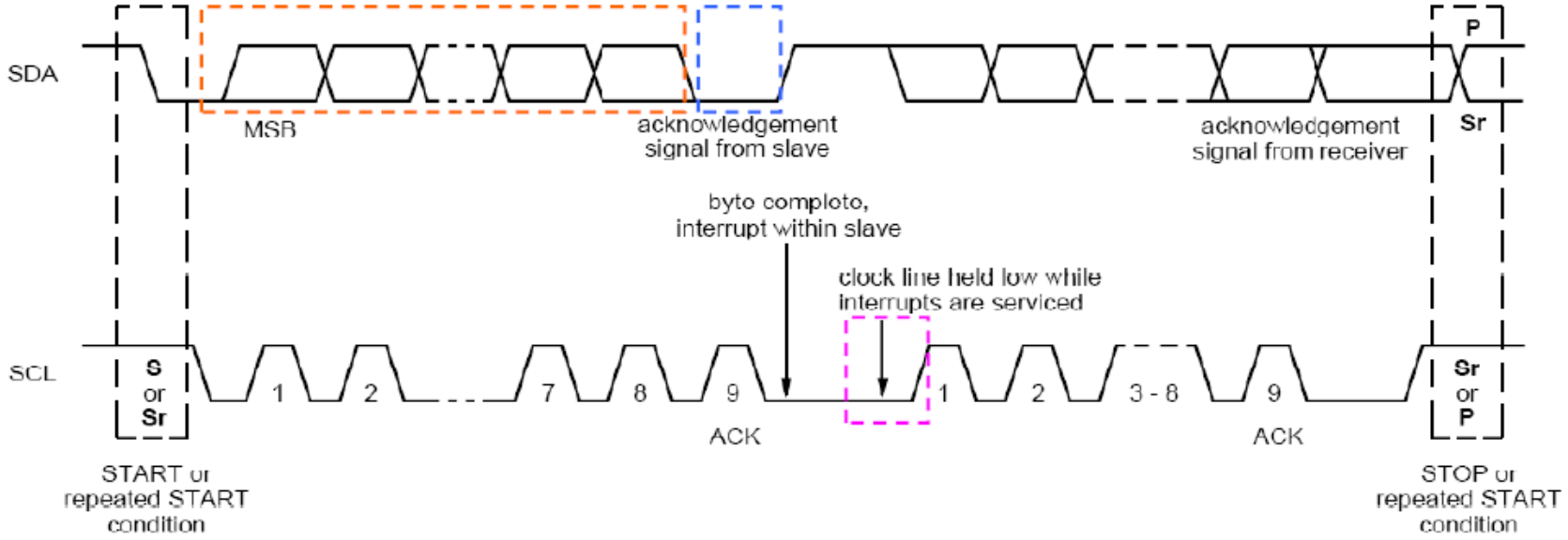
- **Start** condition: a HIGH to LOW transition on the SDA line while SCL is HIGH.
- **Stop** condition: a LOW to HIGH transition on the SDA line while SCL is HIGH.



Source: DesignCon 2003 TecForum I2C Bus Overview

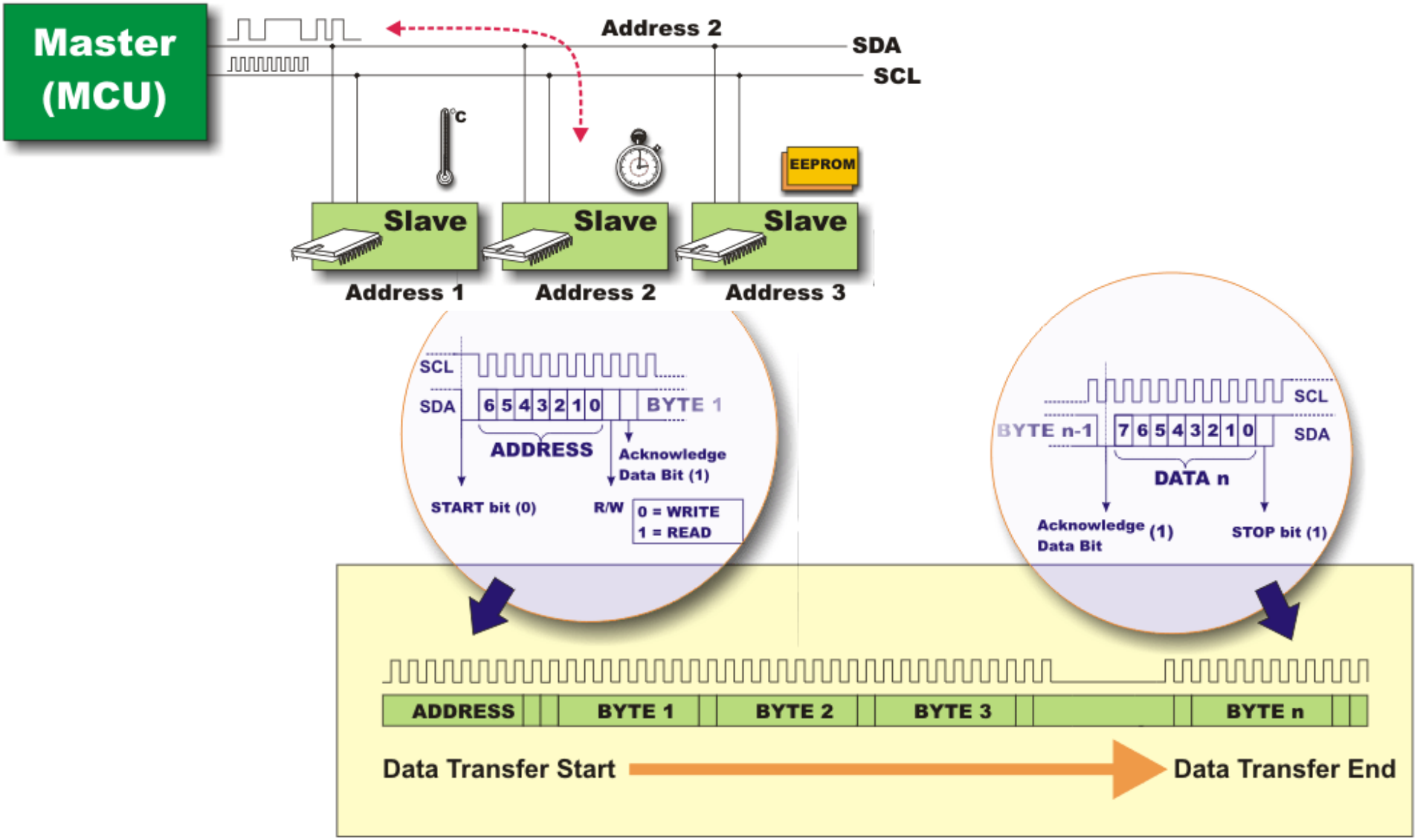
2.4. I2C: Protocol (V)

- **Data Transfer.**
  - Each byte has to be followed by an acknowledge bit (ACK).
  - Number of data bytes transmitted per transfer is unrestricted.
  - If a slave can't receive or transmit another complete byte of data, it can hold the clock line SCL LOW (clock stretching) to force the master into a wait state.

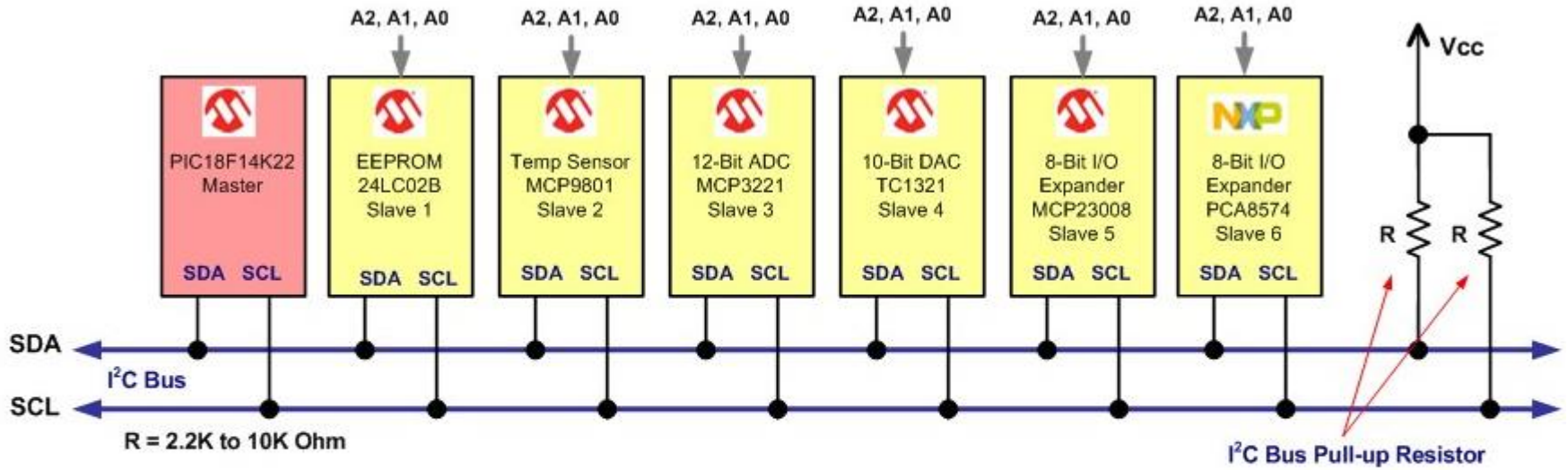


Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. I2C: Data Transfer Example



2.4. I2C: HW Slaves address configurable



7-Bit I²C Slave Address: 4-Bit Device ID (ID3, ID2, ID1, and ID0) + 3-Bit Configurable Address (A2, A1, and A0)

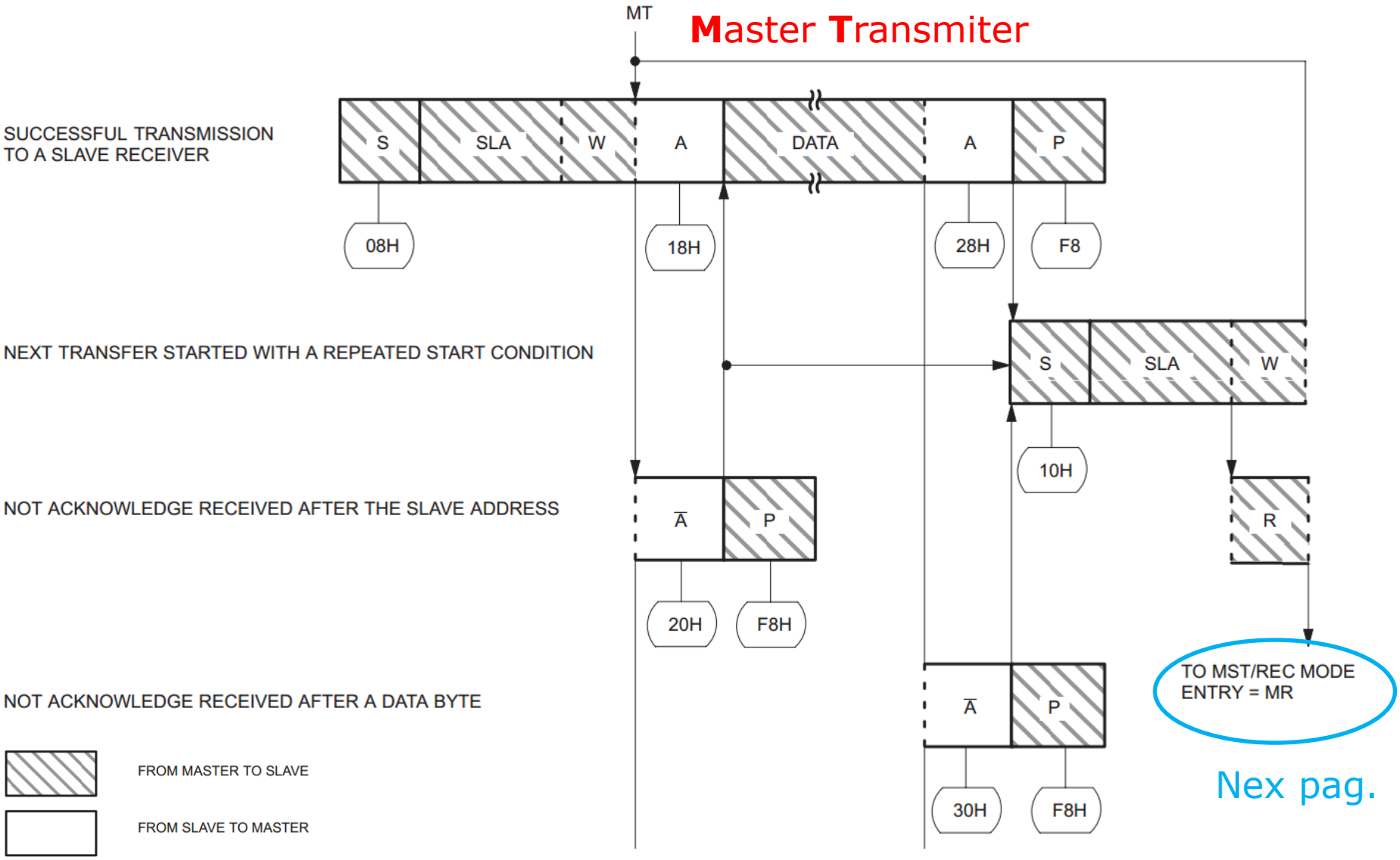
ID3	ID2	ID1	ID0	A2	A1	A0	R/W

R/W – I²C Bus Transfer Direction Command: WRITE = 0 and READ = 1

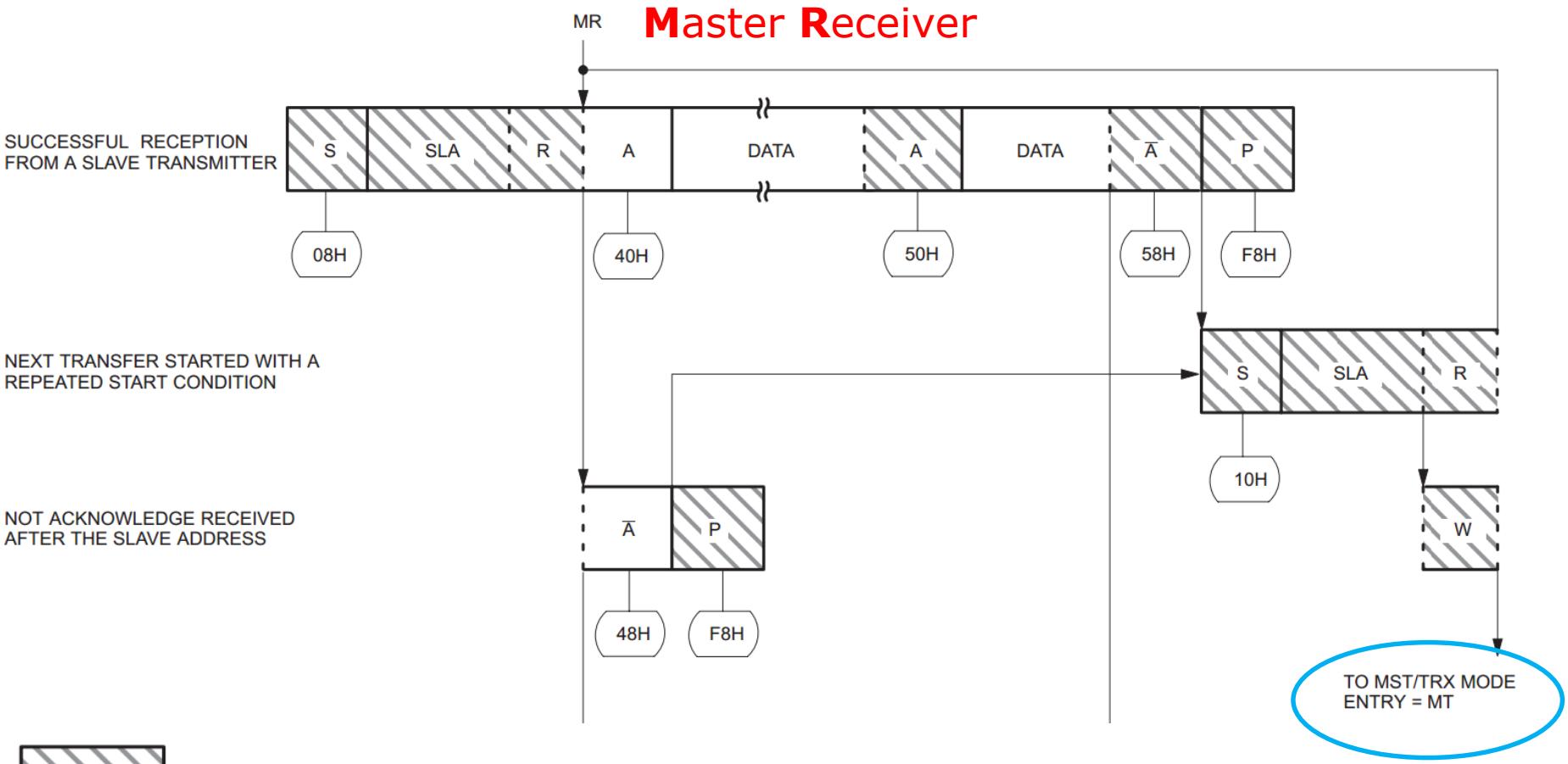
2.4. I2C Registers

Name	Access	Description	Address Offset
I2C0CONSET	R/W	I2C Control Set Register	0x0000
I2C0STAT	RO	I2C status Register	0x0004
I2C0DAT	R/W	I2C Data Register	0x0008
I2C0ADR0	R/W	I2C Slave Address Register 0	0x000C
I2C0SCLH	R/W	SCH Duty Cycle Register High Half Word	0x0010
I2C0SCLL	R/W	SCL Duty Cycle Register Low Half Word	0x0014
I2C0CONCLR	WO	I2C Control Clear Register	0x0018
I2C0MMCTRL	R/W	Monitor Mode Control Register	0x001C
I2C0ADR1	R/W	I2C Slave Address Register 1	0x0020
I2C0ADR2	R/W	I2C Slave Address Register 2	0x0024
I2C0ADR3	R/W	I2C Slave Address Register 3	0x0028
I2C0DATA_BUFFER	RO	I2C Data Buffer Register	0x002C
I2C0MASK0	R/W	I2C Slave Address Mask Register 0	0x0030
I2C0MASK1	R/W	I2C Slave Address Mask Register 1	0x0034
I2C0MASK2	R/W	I2C Slave Address Mask Register 2	0x0038
I2C0MASK3	R/W	I2C Slave Address Mask Register 3	0x003C

2.4. I2C: Format and States in MT mode

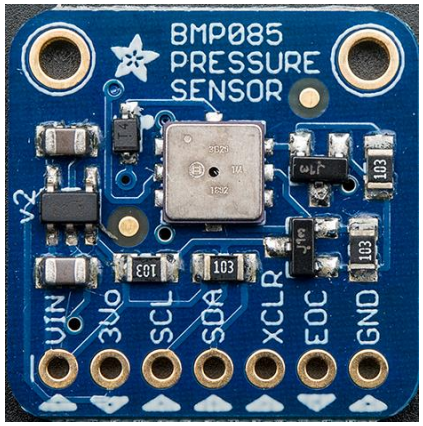


2.4. I2C: Format and States in MR mode

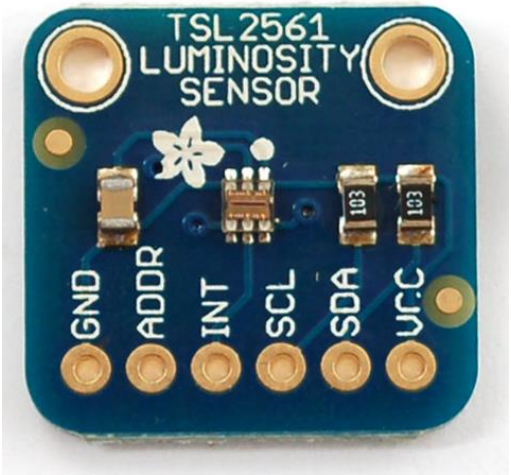


Ant. pag.

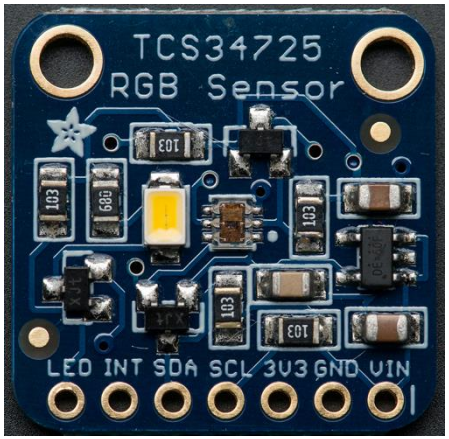
2.4. I2C: Devices with I2C interface



Barometric Pressure Temperature/Altitude Sensor



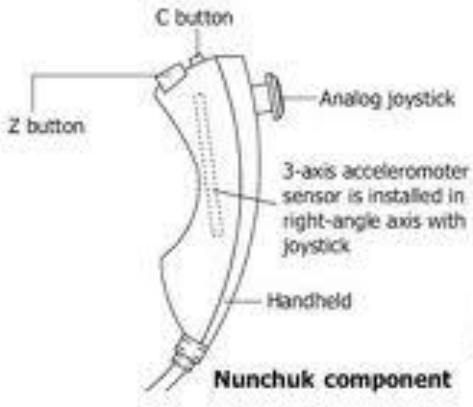
Luminosity Sensor

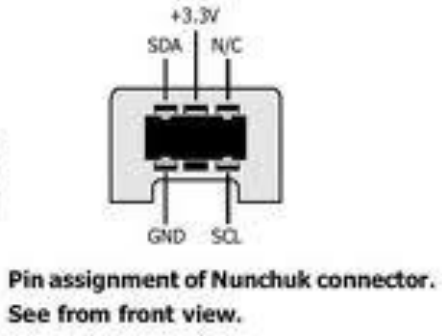


RGB Sensor

Proximity Sensor

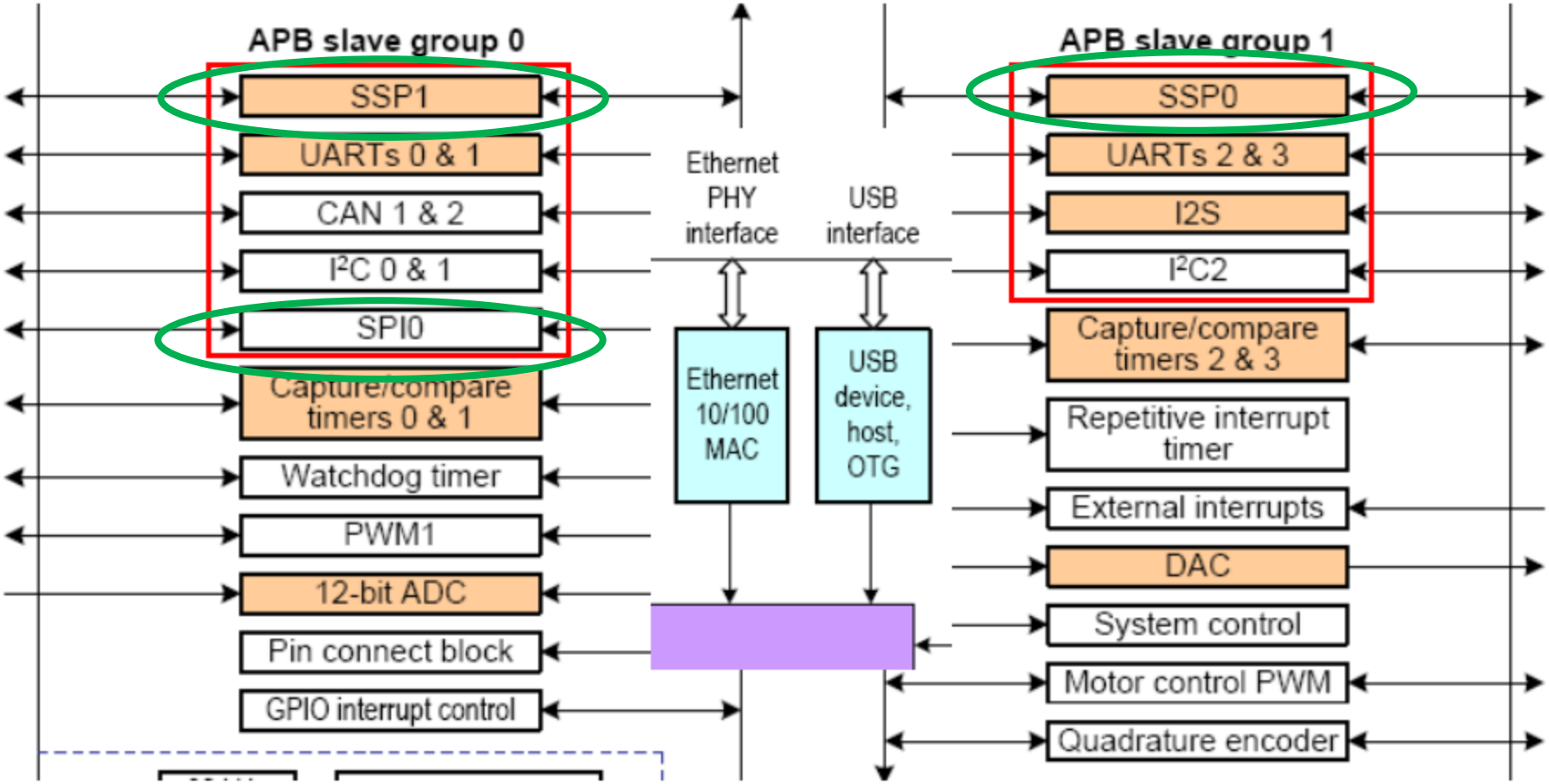
FM Digital Tuner Module





Acelerometer Sensor

2.4. Serial Interfaces: SPI, SSP

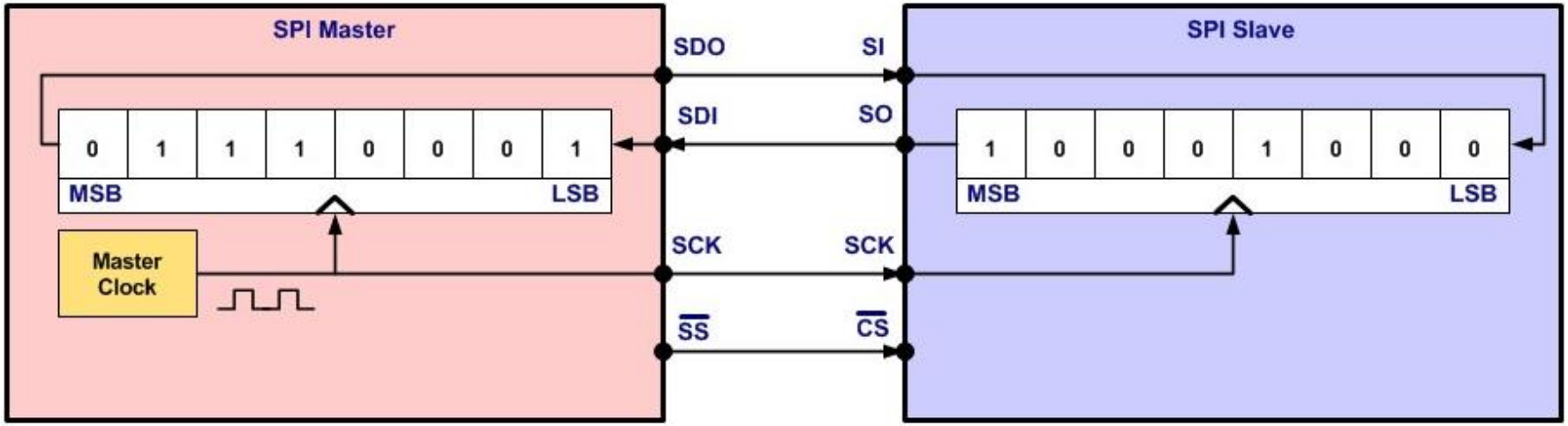


2.4. What SPI ?

- **S**erial **P**eripheral **I**nterface (SPI) is a 4-wire full-duplex **synchronous serial** data link:
  - **SCLK**: Serial Clock
  - **MOSI**: Master Out Slave In -Data from Master to Slave
  - **MISO**: Master In Slave Out -Data from Slave to Master
  - **SS**: Slave Select
- Originally developed by *Motorola*.
- Used for connecting peripherals to each other and to microprocessors.
- Shift register that serially transmits data to other SPI devices.
- Actually a "3 + **n**" wire interface with **n** = *number of devices*.
- Only one master active at a time.
- Various Speed transfers (function of the system clock)

Source: DesignCon 2003 TecForum I2C Bus Overview

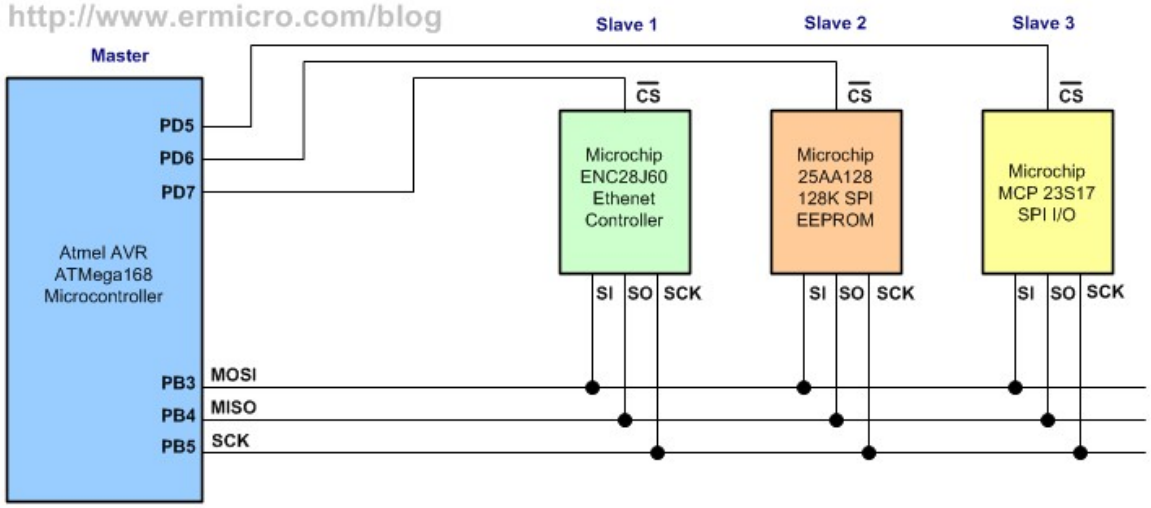
2.4. SPI: Master-Slave basic connection



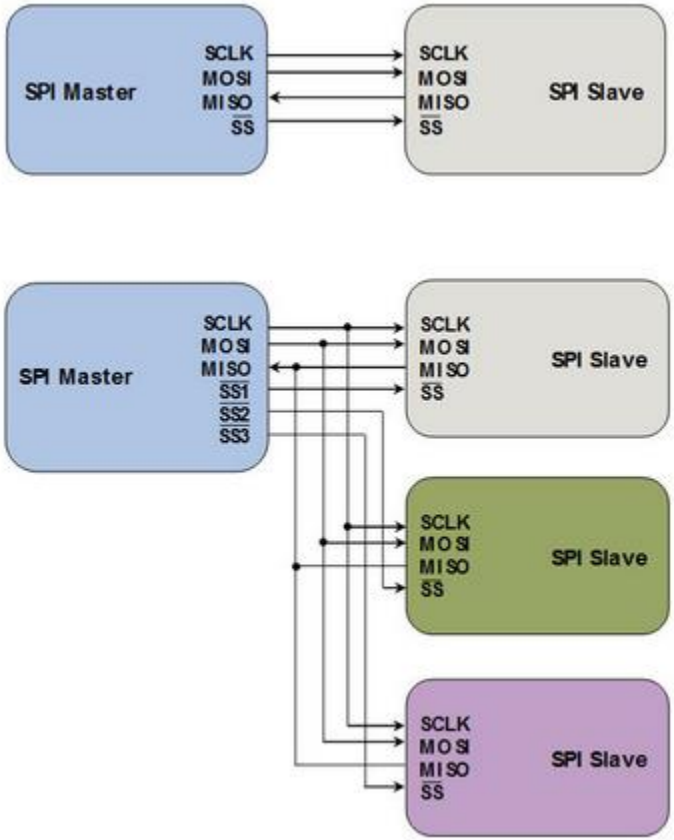
PIN	Description	Direction
SDO	Serial Data Out, This pin also called as MOSI (Master Out Serial In) in AVR Type Microcontroller	Out
SDI	Serial Data In, This pin also called as MISO (Master In Serial Out) in AVR Type Microcontroller	In
SCK	Serial Clock (SPI Master)	Out
SS	Serial Select (could be any I/O Port)	Out

PIN	Description	Direction
SI	Serial In	In
SO	Serial Out	Out
SCK	Serial Clock (SPI Slave)	In
CS	Chip Select (usually active low)	In

2.4. SPI: Master-Slave basic connection

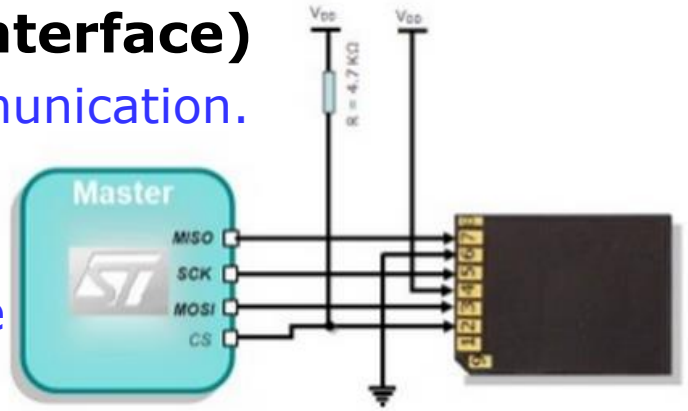


Typical SPI Master with Multiple SPI Slave Device Connection



2.4. SPI (0) and SSP (0, &1)

- **SPI controller (Serial Peripheral Interface)**
  - Synchronous, Serial, Full Duplex Communication.
  - SPI master or slave.
  - 8 to 16 bits per transfer.
  - Programmable clock Polarity and Phase for data transmit/receive operations.
  - Maximum speed (master/slave) 12.5 Mbps.



- **SSP controller (Synchronous Serial Communication)**
  - 8-frame FIFOs for both Transmit and Receive and multi-protocol capabilities.
  - 4 to 16-bits data transfers.
  - DMA support.
  - Maximum speed.
    - 50 Mbps (Master Mode)
    - 8 Mbps (Slave Mode)

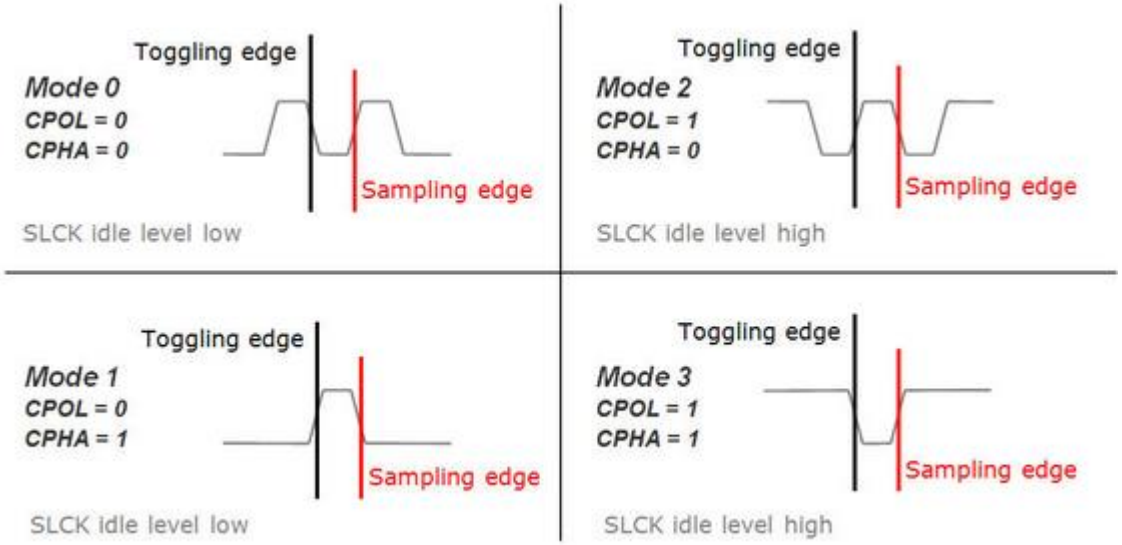
2.4. SPI: pins

Table 358. SPI pin description

Pin Name	Type	Pin Description
SCK	Input/ Output	**Serial Clock.** The SPI clock signal (SCK) is used to synchronize the transfer of data across the SPI interface. The SPI is always driven by the master and received by the slave. The clock is programmable to be active high or active low. The SPI is only active during a data transfer. Any other time, it is either in its inactive state, or tri-stated.
SSEL	Input	**Slave Select.** The SPI slave select signal (SSEL) is an active low signal that indicates which slave is currently selected to participate in a data transfer. Each slave has its own unique slave select signal input. The SSEL must be low before data transactions begin and normally stays low for the duration of the transaction. If the SSEL signal goes high any time during a data transfer, the transfer is considered to be aborted. In this event, the slave returns to idle, and any data that was received is thrown away. There are no other indications of this exception. This signal is not directly driven by the master. It could be driven by a simple general purpose I/O under software control.
MISO	Input/ Output	**Master In Slave Out.** The SPI Master In Slave Out signal (MISO) is a unidirectional signal used to transfer serial data from an SPI slave to an SPI master. When a device is a slave, serial data is output on this pin. When a device is a master, serial data is input on this pin. When a slave device is not selected, the slave drives the signal high-impedance.
MOSI	Input/ Output	**Master Out Slave In.** The SPI Master Out Slave In signal (MOSI) is a unidirectional signal used to transfer serial data from an SPI master to an SPI slave. When a device is a master, serial data is output on this pin. When a device is a slave, serial data is input on this pin.

2.4. SPI: Modes

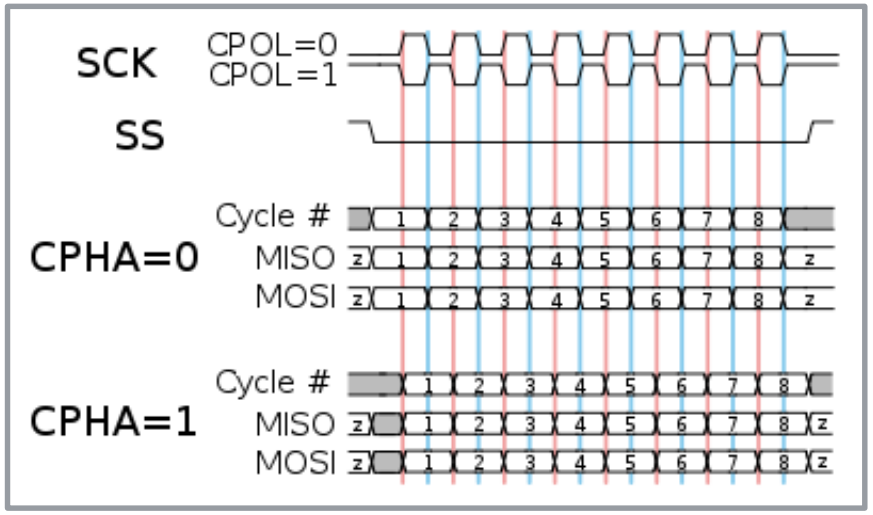
- **Four** communication modes are available.
- Basically define the SCLK edge on which the MOSI line toggles, the SCLK edge on which the master samples the MISO line and the SCLK signal steady level (that is the clock level, high or low, when the clock is not active).
- Each mode is formally defined with a pair of parameters called '**clock polarity**' (**CPOL**) and '**clock phase**' (**CPHA**).



2.4. SPI: Timing (Clock Polarity and Clock Phase)

- The Clock Polarity (**CPOL**) determines the idle state of the clock: If CPOL=0, the clock is low when it's idle and high when it's active. If CPOL=1 the clock is high when it's idle and low when it's active.

- The Clock Phase (**CPHA**) determines edge at which the data is sampled:
  - for CPHA=0 the data is sampled at the first edge.
  - for CPHA=1 the data is sampled on second edge.



- The four combinations of the CPOL and CPHA are called SPI modes, you need to select a mode when configuring SPI, depending on your hardware.
  - Note that if CPOL = 0 and CPHA = 0 the clock transitions form low to high when active and the data is sampled on the raising edge of the clock this is the same as when CPOL =1 and CPHA = 1 because data will still be sampled on the raising edge of the clock

2.4. SPI Register map

Table 360. SPI register map

Name	Description	Access	Reset Value[1]	Address
S0SPCR	SPI Control Register. This register controls the operation of the SPI.	R/W	0x00	0x4002 0000
S0SPSR	SPI Status Register. This register shows the status of the SPI.	RO	0x00	0x4002 0004
S0SPDR	SPI Data Register. This bi-directional register provides the transmit and receive data for the SPI. Transmit data is provided to the SPI0 by writing to this register. Data received by the SPI0 can be read from this register.	R/W	0x00	0x4002 0008
S0SPCCR	SPI Clock Counter Register. This register controls the frequency of a master's SCK0.	R/W	0x00	0x4002 000C
S0SPINT	SPI Interrupt Flag. This register contains the interrupt flag for the SPI interface.	R/W	0x00	0x4002 001C

2.4. SPI: Control Register (S0SPCR)

Table 361: SPI Control Register (S0SPCR - address 0x4002 0000) bit description

Bit	Symbol	Value	Description	Reset Value
1:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	BitEnable	0	The SPI controller sends and receives 8 bits of data per transfer.	0
		1	The SPI controller sends and receives the number of bits selected by bits 11:8.	
3	CPHA		Clock phase control determines the relationship between the data and the clock on SPI transfers, and controls when a slave transfer is defined as starting and ending.	0
		0	Data is sampled on the first clock edge of SCK. A transfer starts and ends with activation and deactivation of the SSEL signal.	
		1	Data is sampled on the second clock edge of the SCK. A transfer starts with the first clock edge, and ends with the last sampling edge when the SSEL signal is active.	
4	CPOL		Clock polarity control.	0
		0	SCK is active high.	
		1	SCK is active low.	
5	MSTR		Master mode select.	0
		0	The SPI operates in Slave mode.	
		1	The SPI operates in Master mode.	
6	LSBF		LSB First controls which direction each byte is shifted when transferred.	0
		0	SPI data is transferred MSB (bit 7) first.	
		1	SPI data is transferred LSB (bit 0) first.	
7	SPIE		Serial peripheral interrupt enable.	0
		0	SPI interrupts are inhibited.	
		1	A hardware interrupt is generated each time the SPIF or MODF bits are activated.	
11:8	BITS		When bit 2 of this register is 1, this field controls the number of bits per transfer:	0000
		1000	8 bits per transfer	
		1001	9 bits per transfer	
		1010	10 bits per transfer	
		1011	11 bits per transfer	
		1100	12 bits per transfer	
		1101	13 bits per transfer	
		1110	14 bits per transfer	
		1111	15 bits per transfer	
		0000	16 bits per transfer	
31:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

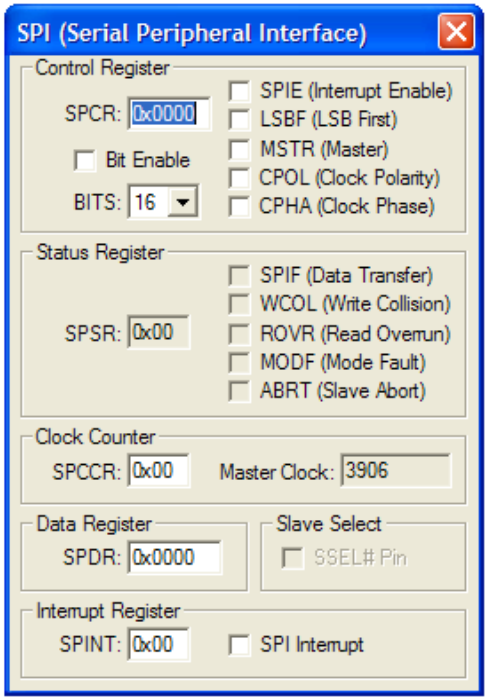
2.4. SPI: Status Register (S0SPSR)

Table 362: SPI Status Register (S0SPSR - address 0x4002 0004) bit description

Bit	Symbol	Description	Reset Value
2:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
3	ABRT	Slave abort. When 1, this bit indicates that a slave abort has occurred. This bit is cleared by reading this register.	0
4	MODF	Mode fault. when 1, this bit indicates that a Mode fault error has occurred. This bit is cleared by reading this register, then writing the SPI0 control register.	0
5	ROVR	Read overrun. When 1, this bit indicates that a read overrun has occurred. This bit is cleared by reading this register.	0
6	WCOL	Write collision. When 1, this bit indicates that a write collision has occurred. This bit is cleared by reading this register, then accessing the SPI Data Register.	0
7	SPIF	SPI transfer complete flag. When 1, this bit indicates when a SPI data transfer is complete. When a master, this bit is set at the end of the last cycle of the transfer. When a slave, this bit is set on the last data sampling edge of the SCK. This bit is cleared by first reading this register, then accessing the SPI Data Register. **Note:** this is not the SPI interrupt flag. This flag is found in the SPINT register.	0
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

2.4. SPI: Software functions

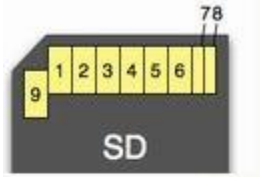
```
unsigned char  spi_transfer(unsigned char data)
{
//SLAVE_ENABLE;                          // SS=0
LPC_SPI->SPDR = data;                    // send a byte
while ((LPC_SPI->SPSR & (1<<7)) == 0)   ;// WAIT until data reg. is empty
return  LPC_SPI->SPDR;                   // return the result
//SLAVE_DISABLE;                         // SS=1
}
```




```
#define  SLAVE_CS_BIT            16  // EXAMPLE
#define  SLAVE_CS_MASK           (1<<SLAVE_CS_BIT)
#define  SLAVE_ENABLE            (LPC_GPIO0->FIOCLR = SLAVE_CS_MASK)
#define  SLAVE_DISABLE           (LPC_GPIO0->FIOSET = SLAVE_CS_MASK)

void spi_config(void)
{
    LPC_SC->PCONP |= (1<<8);             // apply power to the SPI interface
    LPC_SC->PCLKSEL0 |= (1<<16);         // use CCLK as SPI clock
    LPC_SPI->SPCCR = 24;                 // now divide SPI clock (must be >= 8, must be even!)
    LPC_PINCON->PINSEL0 |= (3<<30);      // set P0.15 as SCK
    LPC_PINCON->PINSEL1 |= (3<<2);       // set P0.17 as MISO
    LPC_PINCON->PINSEL1 |= (3<<4);       // set P0.18 as MOSI
    LPC_PINCON->PINMODE1 |= (2<<2);      // set MISO with no pull-up or pull-down
    LPC_SPI->SPCR = (1<<5);              // set SPI to master mode
    LPC_GPIO0->FIODIR |= SLAVE_CS_MASK;  // use SSEL as GPIO slave select line;it an output
    SLAVE_DISABLE;                       // pull the chip-select line high
}
```

2.4. SPI: Devices

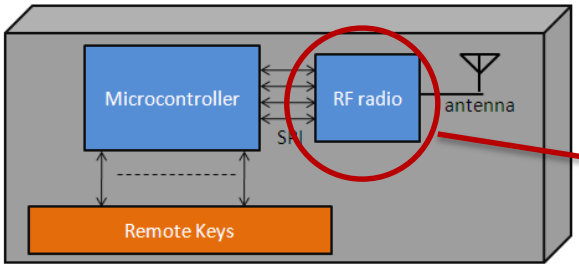
SDCard holder



Pin	SD	SPI
1	CD/DAT3	CS
2	CMD	DI
3	VSS1	VSS1
4	VDD	VDD
5	CLK	SCLK
6	VSS2	VSS2
7	DAT0	DO
8	DAT1	X
9	DAT2	X

Micro SD holder

Barometric Pressure

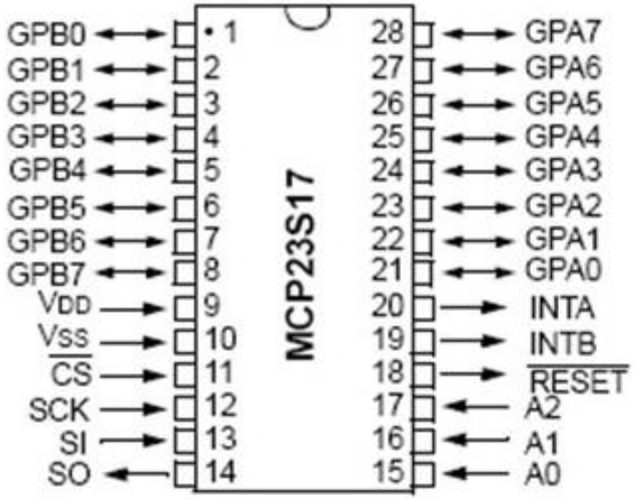


NRF905 Wireless TX/RX

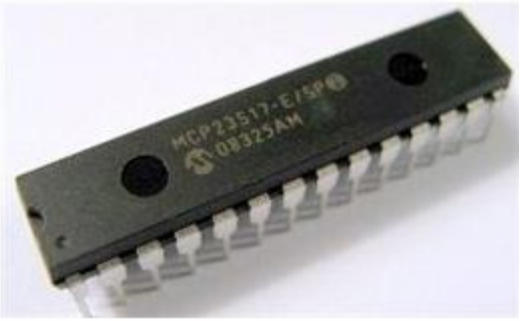
7-seg. Display module

2.4. MCP23S17 : 16-bit SPI I/O Expander

Transfer format



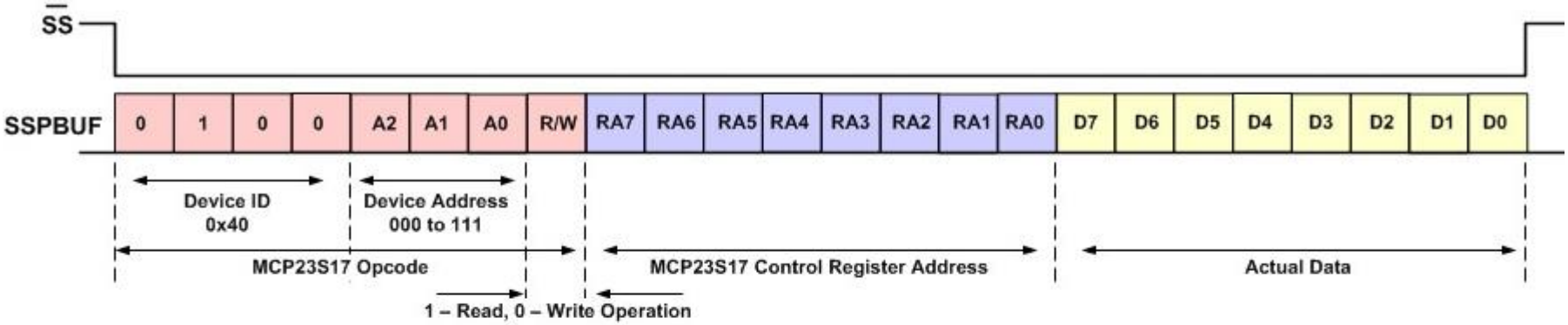
Pin Name	Description
GPA0..7	8-bit General I/O Port A
GPB0..7	8-bit General I/O Port B
INTA	Port A Interrupt Signal
INTB	Port B Interrupt Signal
RESET	Reset Signal
A0,A1,A2	Configurable Address
CS	Chip Select (active Low)
CSK	Synchronous Clock
SI	Slave In
SO	Slave Out
V_{DD}	+5 Volt
V_{SS}	GND



Microchip MCP23S17 SPI 16-bit I/O Expander

2.4. MCP23S17 : 16-bit SPI I/O Expander

Transfer format



MCP23S17 Control Registers Address for IOCONA.BANK=0

Register	Description	Address
IODIRA	I/O Direction Register A	0x00
IODIRB	I/O Direction Register B	0x01
IPOLA	Input Polarity Port Register A	0x02
IPOLB	Input Polarity Port Register B	0x03
GPINTENA	Interrupt on Change Pins A	0x04
GPINTENB	Interrupt on Change Pins B	0x05
DEFVALA	Default Value Register A	0x06
DEFVALB	Default Value Register B	0x07
INTCONA	Interrupt On Change Cont . A	0x08
INTCONB	Interrupt On Change Cont . B	0x09
IOCONA	I/O Expander Configuration	0x0A

Register	Description	Address
IOCONB	I/O Expander Configuration	0x0B
GPPUA	GPIO Pull-up Resistor A	0x0C
GPPUB	GPIO Pull-up Resistor B	0x0D
INTFA	Interrupt Flag Register A	0x0E
INTFB	Interrupt Flag Register B	0x0F
INTCAPA	Interrupt Capture Value A	0x10
INTCAPB	Interrupt Capture Value B	0x11
GPIOA	General Purpose I/O A	0x12
GPIOB	General Purpose I/O B	0x13
OLATA	Output Latch Register A	0x14
OLATB	Output Latch Register B	0x15

2.4. SSP Registers

Name	Access	Description	Address Offset
SSP0CR0	R/W	Control Register 0	0x0000
SSP0CR1	R/W	Control Register 1	0x0004
SSP0DR	R/W	Data Register	0x0008
SSP0SR	RO	Status Register	0x000C
SSP0CPSR	R/W	Clock Prescaler Register	0x0010
SSP0IMSC	R/W	Interrupt Mask Set and Clear Register	0x0014
SSP0RIS	RO	Raw Interrupt Status Register	0x0018
SSP0MIS	RO	Mask Interrupt Status Register	0x001C
SSP0ICR	WO	SSPICR Interrupt Clear Register	0x0020

2.4. SSP: Software functions

```
void    SPI_Init(void)
{
LPC_SC->PCONP |= 1 <<21;    //enable POWER to SSP0 (redundant following reset)
LPC_SC->PCLKSEL1 |= 1<<10;   //pclk = cclk
LPC_SSP0->CPSR |= 8;     //internal divider
LPC_PINCON->PINSEL0 |= 0x80000000;   //Pin P0.15 allocated to function 3 SCLK
LPC_PINCON->PINSEL1 |= 0x02<<0;  //Pin P0.16 allocated to function 3 NSS/SSEL
LPC_PINCON->PINSEL1 |= 0x02<<2;  //Pin P0.17 allocated to function 3 MISO
LPC_PINCON->PINSEL1 |= 0x02<<4;  //Pin P0.18 allocated to function 3 MOSI
LPC_SSP0->CR0 |= 3<<6;   //clock phase
LPC_SSP0->CR0 |= 7<<0;   // 8 bits
LPC_SSP0->CR1 |= 1<<1;  //enable SSP
LPC_GPIO0->FIODIR |= (1<<23);    //make output
}
char    spi_transfer(char data) {
LPC_GPIO0->FIOCLR = 1<<23;   //select slave
LPC_SSP0->DR = data;
while (!(LPC_SSP0->SR & (1<<2)));
LPC_GPIO0->FIOSET = 1<<23;   //release slave
return (LPC_SSP0->DR);
}
```

2.4. SSP: GLCD library example

```
unsigned char LPC17xx_SPI_SendRecvByte (unsigned char byte_s)
{
        /* wait for current SSP activity complete */
        while (SSP_GetStatus(LPC_SSP0, SSP_STAT_BUSY) ==  SET);

        SSP_SendData(LPC_SSP0, (unsigned short) byte_s);

        while (SSP_GetStatus(LPC_SSP0, SSP_STAT_RXFIFO_NOTEMPTY) == RESET);
        return (SSP_ReceiveData(LPC_SSP0));
}
```

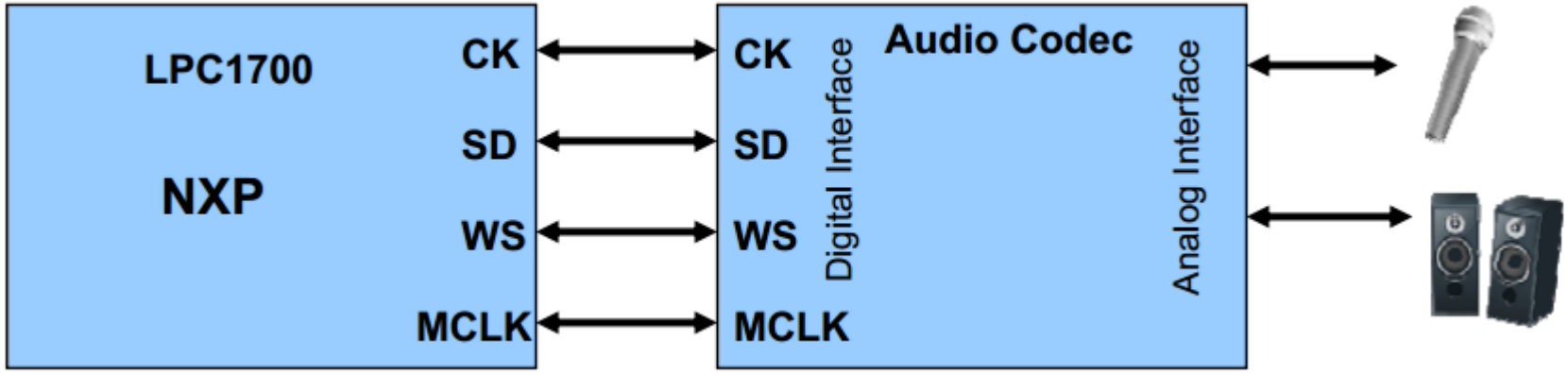
```
void LCD_WriteData( unsigned short data)
{
        SPI_CS_LOW;

        LPC17xx_SPI_SendRecvByte(SPI_START | SPI_WR | SPI_DATA);    /* Write : RS = 1, RW = 0 */
        LPC17xx_SPI_SendRecvByte((data >>   8));                    /* Write D8..D15          */
        LPC17xx_SPI_SendRecvByte((data & 0xFF));                    /* Write D0..D7           */

        SPI_CS_HIGH;
}
```

2.4. SPI: GLCD library example

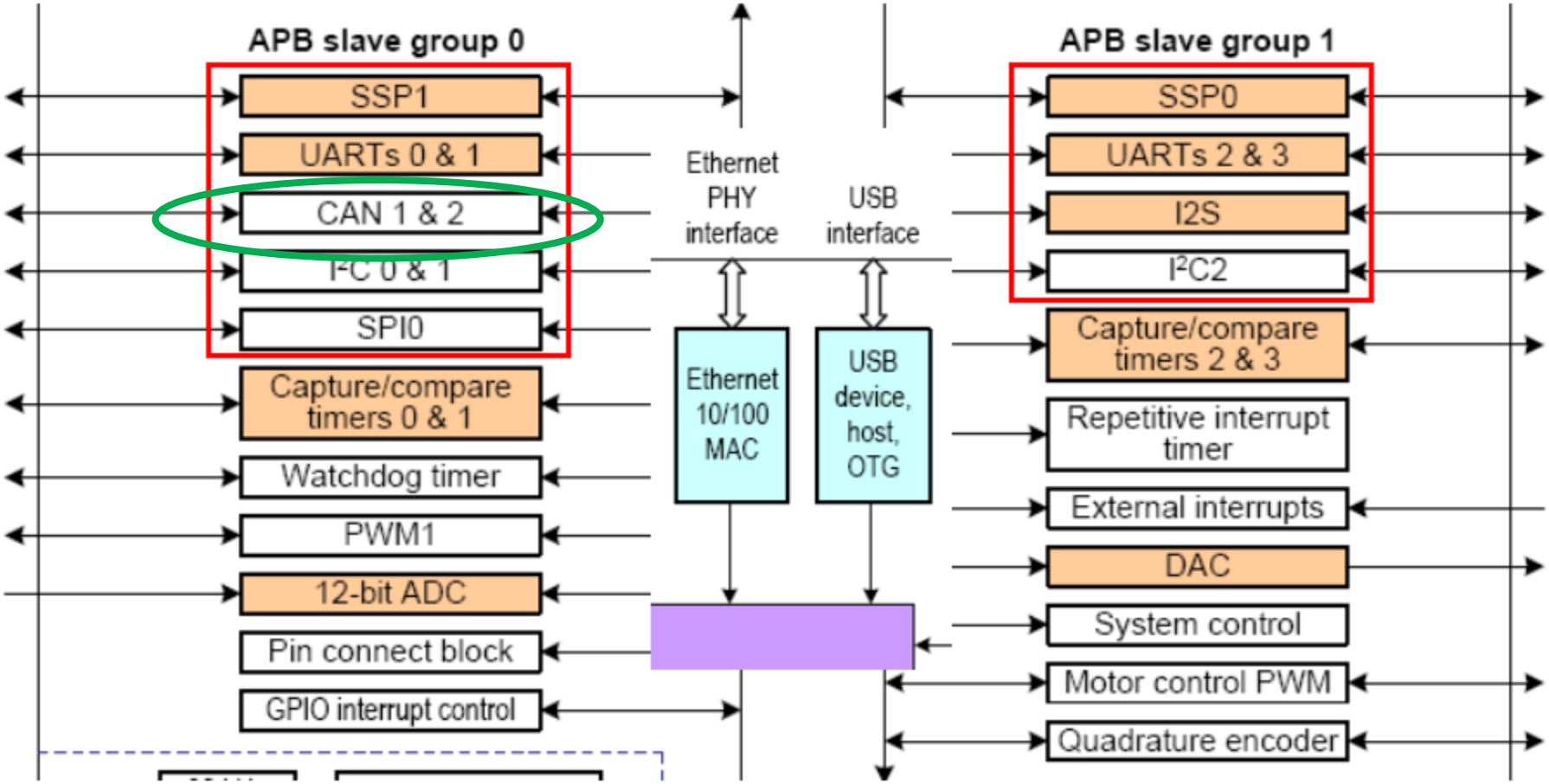
```
unsigned short LCD_ReadData(void)
{
        unsigned short value;

        SPI_CS_LOW;

        LPC17xx_SPI_SendRecvByte(SPI_START | SPI_RD | SPI_DATA);     /* Read: RS = 1, RW = 1
        LPC17xx_SPI_SendRecvByte(0);                                 /* Dummy read 1
        value   = LPC17xx_SPI_SendRecvByte(0);                       /* Read D8..D15
        value <<= 8;
        value  |= LPC17xx_SPI_SendRecvByte(0);                       /* Read D0..D7

        SPI_CS_HIGH;

        return value;
}
```

2.4. I2S

- Supports 3-wire data transmit and receive or 4-wire combined transmit and receive connections.
- Audio Master Clock input/output (used on many I2S codecs)
- The I2S input and output can each operate independently in both master and slave mode.
- Both mono and stereo data streams are supported over wide range of sampling frequencies which can range from 16 - 96 kHz.
- GPDMA support – allows streaming audio data over I2S interface.

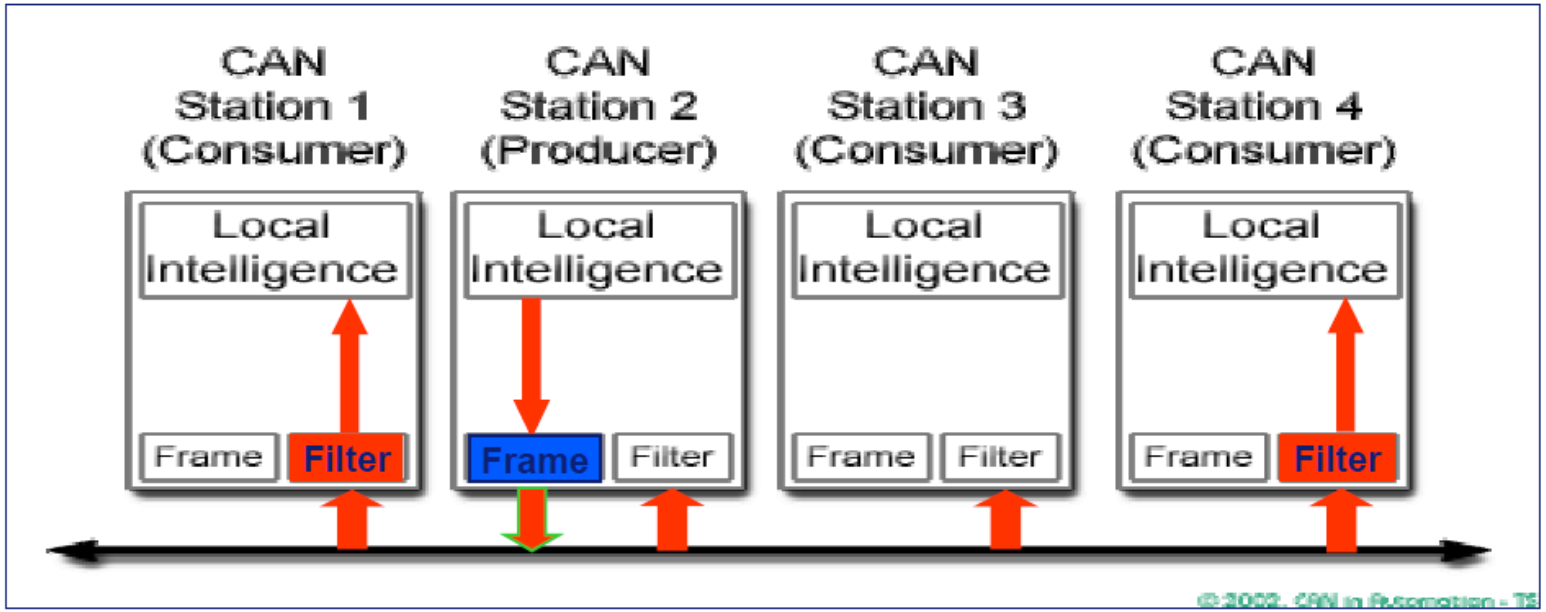


2.4. Serial Interfaces: CAN



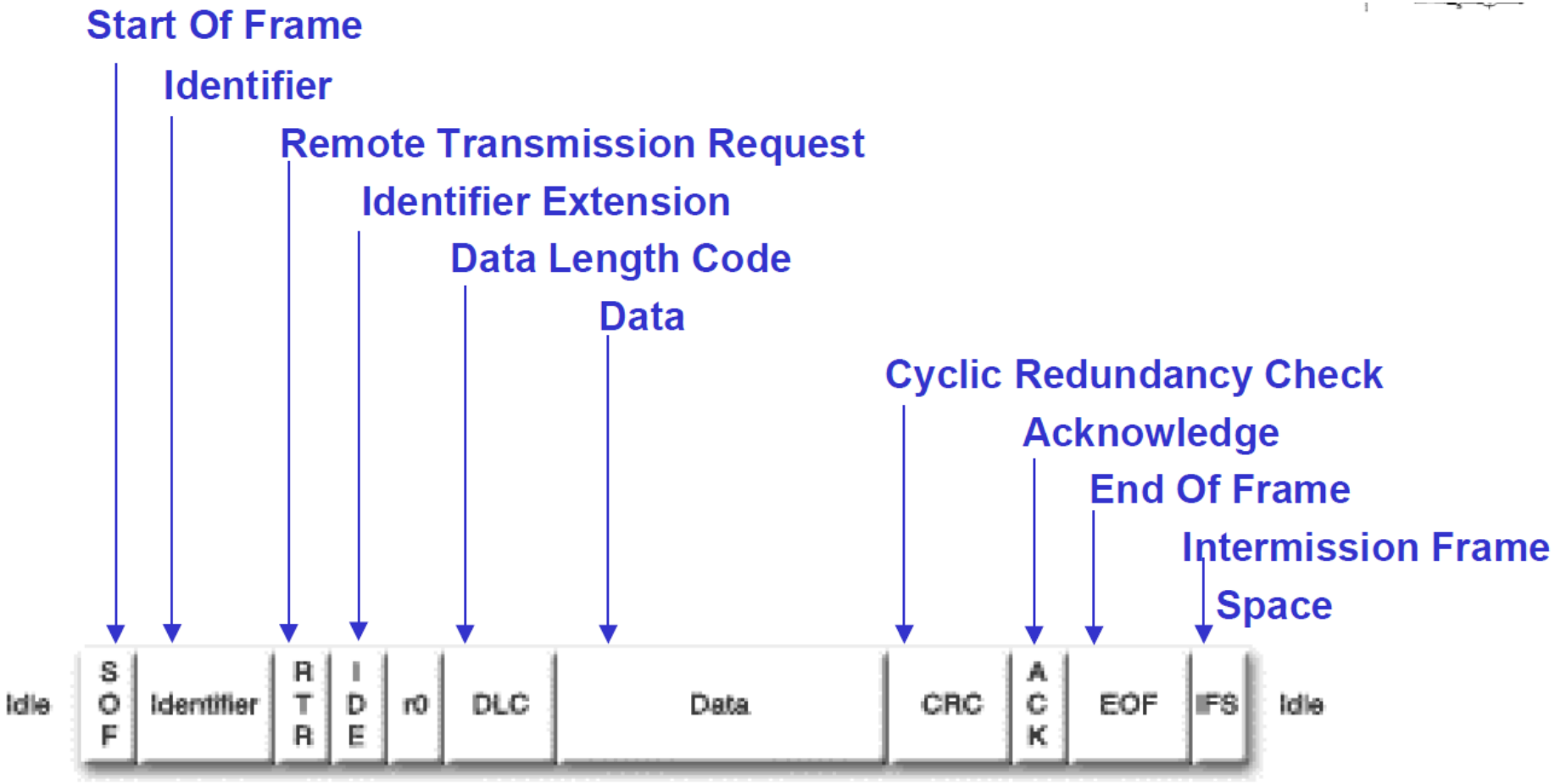
2.4. CAN: Controller Area Network

- Proposed by Bosch with automotive applications in mind (and promoted by CIA -of Germany -for industrial applications)
- Relatively complex coding of the messages.
- Relatively accurate and (usually) fixed timing.
- All modules participate in every communication.
- Content-oriented (message) addressing scheme.



Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. CAN: Protocol



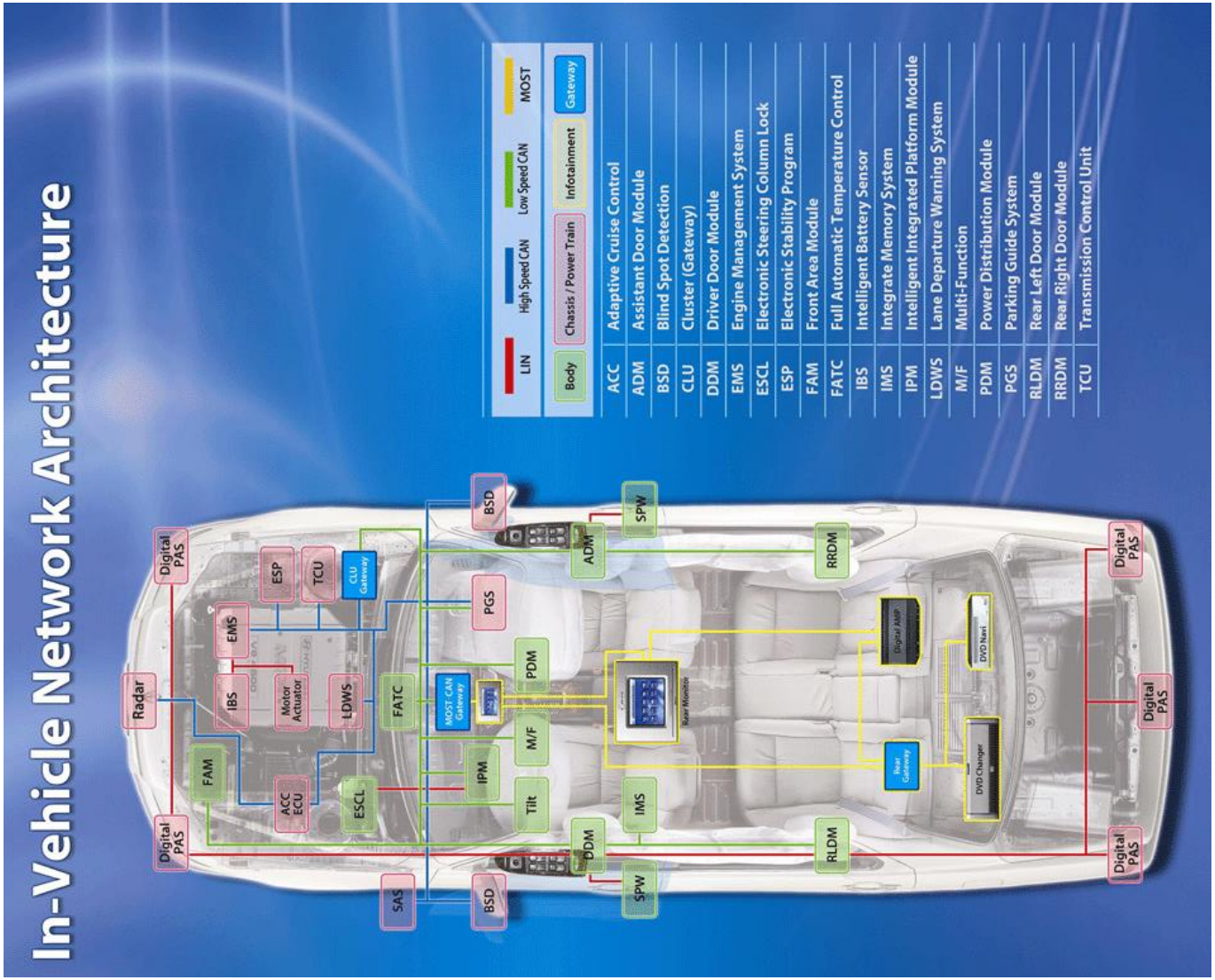
Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. CAN: Advantages

- Accepted standard for Automotive and industrial applications
  - Interfacing between various vendors easier to implement.
- Freedom to select suitable hardware
  - differential or 1 wire bus.
- Secure communications, high Level of error detection
  - 15 bit CRC messages (Cyclic Redundancy Check)
  - Reporting / logging
  - Faulty devices can disconnect themselves
  - Low latency time
  - Configuration flexibility
- High degree of EMC immunity (when using Si-On-Insulator technology)

Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. CAN: In-Vehicle Network Architecture



2.4. What is USB ?

- Originally a standard for connecting PCs to peripherals
- Defined by Intel, Microsoft, …
- Intended to replace the large number of legacy ports in the PC.
- Single master (= Host) system with up to 127 peripherals.
- Simple plug and play; no need to open the PC.
- Standardized plugs, ports, cables.
- Has over 99% penetration on all new PCs.
- Adapting to new requirements for flexibility of Host function.
  - New Hardware/Software allows dynamic exchanging of Host/Slave roles
  - PC is no longer the only system Host. Can be a camera or a printer.

Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. USB: Topology

- Host
  - One PC host per system.
  - Provides power to peripherals

  .
- Hub
  - Provides ports for connecting more peripheral devices.
  - Provides power, terminations
  - External supply or Bus Powered

- Device, Interfaces and Endpoints
  - Device is a collection of data interface(s)
  - Interface is a collection of endpoints (data channels).
  - Endpoint associated with FIFO(s) for data I/O interfacing.

Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. USB: Advantages

- Hot pluggable, no need to open cabinets.
- Automatic configuration.
- Up to 127 devices can be connected together.
- Push for USB to become THE standard on PCs.
  - standard for iMac, supported by Windows, now on > 99%of PCs
- Interfaces (bridges) to other communication channels exist.
  - USB to serial port (serial port vanishing from laptops).
  - USB to IrDA or to Ethernet.
- Extreme volumes force down IC and hardware prices.
- Protocol is evolving fast.

Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. USB: Versions of specification

- USB 1.1
  - Established, large PC peripheral markets.
  - Well controlled hardware, special 4-pin plugs/sockets.
  - 12MBits/sec (normal) or 1.5Mbits/sec (low speed) data rate.

- USB 2.0
  - Challenging IEEE1394/Firewirefor video possibilities.
  - 480 MHz clock for Hi-Speed means it's real "UHF" transmission.
  - Hi-Speed option needs more complex chip hardware and software.
  - Hi-Speed component prices about x 2 compared to full speed.

- USB "OTG" (On The Go) Supplement
  - New hardware -smaller 5-pin plugs/sockets.
  - Lower power (reduced or no bus-powering).

Source: DesignCon 2003 TecForum I2C Bus Overview

2.4. Pros and Cons of the different buses

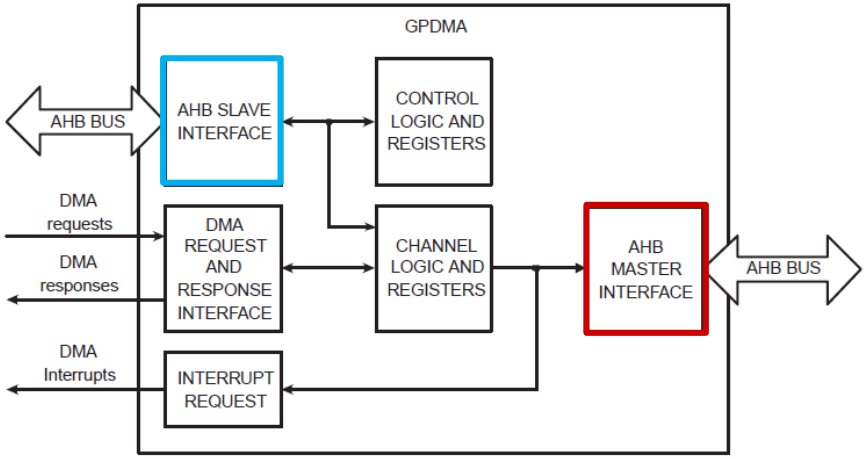
UART	CAN	USB	SPI	I²C
• Well Known • Cost effective • Simple	• Secure • Fast	• Fast • Plug&Play HW • Simple • Low cost	• Fast • Universally accepted • Low cost • Large Portfolio	• Simple • Well known • Universally accepted • Plug&Play • Large portfolio • Cost effective
• Limited functionality • Point to Point	• Complex • Automotive oriented • Limited portfolio • Expensive firmware	• Powerful master required • No Plug&Play SW - Specific drivers required	• No Plug&Play HW • No “fixed” standard	• Limited speed

Source: DesignCon 2003 TecForum I2C Bus Overview

GPDMA

2.5. GPDMA: General Purpose DMA Controller

- Direct Memory Access is a feature which allows the peripherals to access the memory directly to transfer data to other peripherals without the help of CPU.

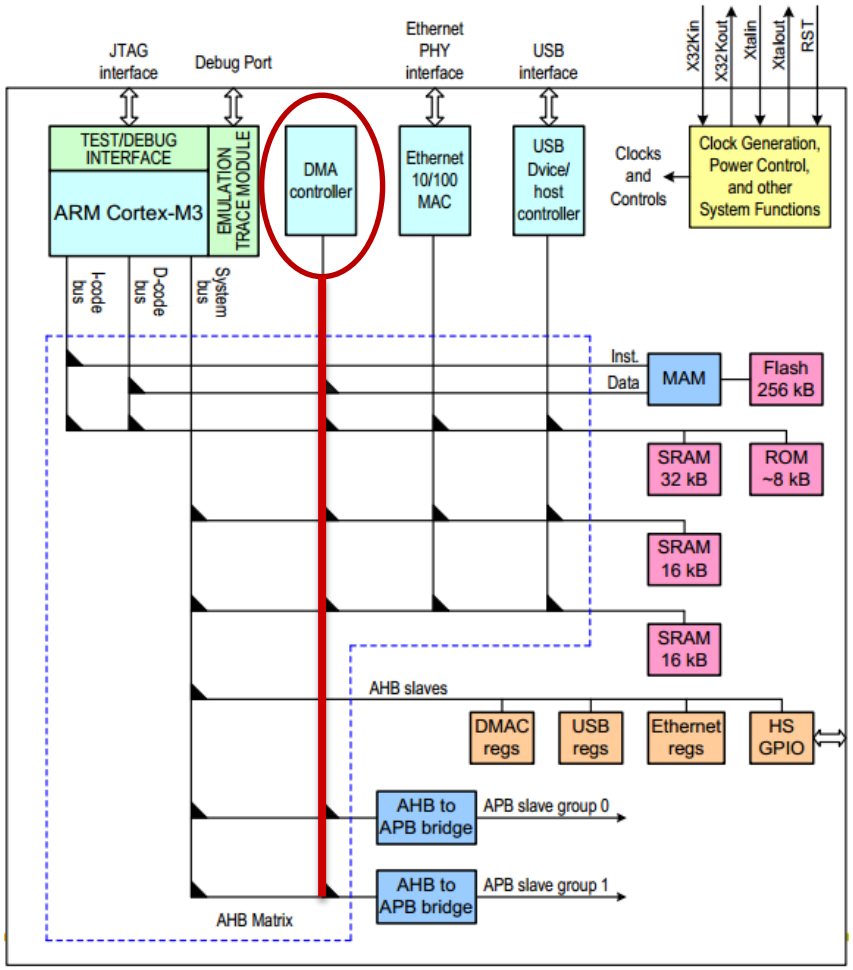


- So whenever we are using DMA, we can be doing computations on the CPU while the data transfer is taking place.

- LPC17xx has a General Purpose DMA (GPDMA) which has **8 channels** of which each can perform **unidirectional data transfer**.
  - All types of data transfers like **memory-memory**, **memory-peripheral**, **peripheral-peripheral** are supported.
  - And we can prioritize the DMA transfers as we wish. It can also perform **8-bit, 16-bit and 32-bit wide transactions**.
  - If we want to transfer a stream of data that is **not stored in contiguous** manner, then we can use the **scatter** and **gather** feature of the DMA which uses linked list.

2.5. GPDMA: General Purpose DMA Controller

- Supports high-speed peripherals as well as memory-to-memory transfers.
  - 32-bit master bus width (support 8-, 16-, or 32-bit transfers).
  - Eight DMA channels, each with a four-word FIFO.
  - 16 peripheral DMA request lines.
  - Memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral transfers are supported.
- DMA support for following peripherals:
  - All UARTs.
  - 12-bit ADC and 10-bit DAC.
  - Timer match conditions (trigger only).
  - SSP/ I2S.
  - GPIOs.
- Single DMA and burst DMA request signals.



2.5. GPDMA: Register Map

Table 544. GPDMA register map

Name	Description	Access	Reset state	Address
General registers				
DMACIntStat	DMA Interrupt Status Register	RO	0	0x5000 4000
DMACIntTCStat	DMA Interrupt Terminal Count Request Status Register	RO	0	0x5000 4004
DMACIntTCClear	DMA Interrupt Terminal Count Request Clear Register	WO	-	0x5000 4008
DMACIntErrStat	DMA Interrupt Error Status Register	RO	0	0x5000 400C
DMACIntErrClr	DMA Interrupt Error Clear Register	WO	-	0x5000 4010
DMACRawIntTCStat	DMA Raw Interrupt Terminal Count Status Register	RO	0	0x5000 4014
DMACRawIntErrStat	DMA Raw Error Interrupt Status Register	RO	0	0x5000 4018
DMACEnbldChns	DMA Enabled Channel Register	RO	0	0x5000 401C
DMACSoftBReq	DMA Software Burst Request Register	R/W	0	0x5000 4020
DMACSoftSReq	DMA Software Single Request Register	R/W	0	0x5000 4024
DMACSoftLBReq	DMA Software Last Burst Request Register	R/W	0	0x5000 4028
DMACSoftLSReq	DMA Software Last Single Request Register	R/W	0	0x5000 402C
DMACConfig	DMA Configuration Register	R/W	0	0x5000 4030
DMACSync	DMA Synchronization Register	R/W	0	0x5000 4034
DMAREQSEL	Selects between UART and timer DMA requests on channels 8 through 15	R/W	0	0x400F C1C4
Channel 0 registers				
DMACC0SrcAddr	DMA Channel 0 Source Address Register	R/W	0	0x5000 4100
DMACC0DestAddr	DMA Channel 0 Destination Address Register	R/W	0	0x5000 4104
DMACC0LLI	DMA Channel 0 Linked List Item Register	R/W	0	0x5000 4108
DMACC0Control	DMA Channel 0 Control Register	R/W	0	0x5000 410C
DMACC0Config	DMA Channel 0 Configuration Register	R/W	0 [1]	0x5000 4110

2.5. GPDMA: General Configuration

- The essential things that you need to do while configuring the DMA are select the channel to be used for data transfer based on the priority.
  - Set the bus **transfer width** (8-bit, 16-bit or 32-bit).
  - Set the **source** and **destination**.
  - Set the DMA **request signal** (The signal that will start the DMA transfer).
  - Set the endian behaviour (Little endian or Big endian).
  - Set the **burst size** (Number of bytes that need to be transferred continuously in one DMA transaction).
  - Set the **block transfer size** (max. 4095)
  - Set the **type of transfer** (Mem-Mem, Mem-Peripheral or Peripheral-Peripheral).

2.5. GPDMA: Memory to GPIO transfers example (I)

- Now let us go step by step and setup the GPDMA to perform a simple task of toggle a group of LEDs connected to GPIO:
  - Lets store a pattern series 1's and 0's in an array (100 bytes) and then transfer it to the GPIO port for which the LEDs are connected through channel 0 of the DMA.
  - Timer 1 **generates the DMA request signal** every 0.5 seg.

- **Setup of the timer which generates the DMA request signal:**

```
LPC_SC->PCONP |= 1 << 2;          // Power up DMA
LPC_TIM1->MR0 = F_tick/2 - 1;     // 0.5 seg. (F_tick=F_pclk, PR=0)
LPC_TIM1->MCR = 1 << 1;           // reset on Match Compare 0
LPC_TIM1->TCR = 0x01;             // start timer after DMA init.
```

- **Now lets configure the DMA:**
- 1. Power up the GPDMA (LPC17xx manual table 46, pg 64)

```
LPC_SC->PCONP |= 1 << 29;
//There is no need to setup the clock for DMA as we were doing for other peripherals.
 //The system clock will be directly given to the DMA.
```

- 2. Enable the GPDMA (LPC17xx manual table 557, pg 599)

```
LPC_GPDMA->DMACConfig |= 1 << 0;
```

2.5. GPDMA: Memory to GPIO transfers example (II)

- 3. Set the Match Compare 0 (MAT1.0 signal) as the DMA request signal.
  - LPC_GPDMA->DMACSync &= ~(1 << 10); // use MAT1.0 for Sync (LPC17xx manual table 558, pg 599) Not needed becouse DMACSync register will be zero on reset.
  - LPC_SC->DMAREQSEL |= 1 << 2; // Timer1 Match Compare 0 as DMA request (LPC17xx manual table 559, pg 600)
- 4. **Clear the Interrupt Terminal Count Request and Interrupt Error Status register**. (Writing 1 to the clear registers will clear the entry in the main register)
  - LPC_GPDMA->DMACIntErrClr |= 0xff; // (LPC17xx manual table 549, pg 596)
  - LPC_GPDMA->DMACIntTCClear |= 0xff; // (LPC17xx manual table 547, pg 595)
- 5. Set the **source and destination addresses** (LPC17xx manual table 560 and 561, pg 601)
  - LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_GPIO1->FIOPIN3); // LED is connected to P1[24…31].
  - LPC_GPDMACH0->DMACCSrcAddr = (uint32_t) &data[0]; // data[] is the array where I have stored alternating 1's and 0's

2.5. GPDMA: Memory to GPIO transfers example (III)

- 6. Clear the **L**inked **L**ist **I**tem as we are not using it. (LPC17xx manual table 562, pg 602)
  - LPC_GPDMACH0->DMACCLLI = 0;
- 7. Set the burst transfer size, source burst and destination burst size, source and destination width, source increment, destination increment (LPC17xx manual table 563, pg 603)
  - In our case, let the **transfer size be 100 bytes**, source burst and destination **burst sizes are 1**, source and destination **width are 8-bits** and we need to do **source increment** and there is **no need for destination increment**.
  - LPC_GPDMACH0->DMACCControl = 100 | (1 << 26);
- 8. Set the type of transfer as **Memory to Peripheral** and the destination **request line as MAT1.0** (LPC17xx manual table 564, pg 605)
  - LPC_GPDMACH0->DMACCConfig = (10 << 6) | (1 << 11); // 10 corresponds to MAT1.0 and it is selected as the destination request peripheral (LPC17xx manual table 543, pg 592)
- 9. Enable the channel (LPC17xx manual table 564, pg 605)
  - LPC_GPDMACH0->DMACCConfig |= 1; //enable CH0
- 10. Setup an interrupt for DMA transfer completion (optional), so that after the DMA transfer is complete and interrupt request is generated, and in the interrupt service routine, you need to disable the channel by writing:
  - LPC_GPDMACH0->DMACCConfig = 0; // stop CH0 DMA
- 10. Repeat all the steps of configuration to setup another transaction.

2.5. GPDMA: DAC to Memory transfer example (I)

- Programa que demuestra el funcionamiento del ADC por DMA.
- El inicio de conversión del ADC mediante MAT1.0.
- Canal 0 del ADC (AD0.0) es P0.23.
- Necesario hab. la interrup. ADC canal 0 (No habilitar int. ADC en el NVIC)
- DMA en modo transferencia de 8 bits (Acceder a ADDR0[8…15])

```
#include <LPC17xx.H>
#define F_cpu 100e6        // Defecto Keil (xtal=12Mhz)
#define F_pclk F_cpu/4     // Defecto despues del reset
#define F_muestreo 8000    // Frecuencia de muestreo del ADC

 //Configuracion del DMA
#define WIDTH_VALUE 0      // Size Mode Transfer = 8 bits
#define OFFSET_ADC 0x11    // Para acceder a ADDR0[8..15]

//#define WIDTH_VALUE 1      // Size Mode Transfer = 16 bits
//#define OFFSET_ADC 0x10    // Para acceder a ADDR0[0..15]

#define SBURST 0x00
#define DBURST 0x00
#define TAM_BLOCK_DMA 1000 // Tamaño del bloque

unsigned char buffer[TAM_BLOCK_DMA]; // destino de las muestras
```

2.5. GPDMA: DAC to Memory transfer example (II)

- Configuración del Timer 0 (Trigger del DMA)

```
/*  Timer 0 en modo Output Compare (reset TOTC on Match 1)
  Counter clk: 25 MHz   MAT0.1 : On match, Toggle pin/output (P1.29)
  Cada 2 Match se provoca el INICIO DE CONVERSIÓN DEL ADC
  Habilitamos la salida (MAT0.1) para observar la frecuencia de muestreo del ADC */

void init_TIMER0(void)
{
    LPC_SC->PCONP|=(1<<1);              //
    LPC_PINCON->PINSEL3|= 0x0C000000; //
    LPC_TIM0->PR = 0x00;                //
    LPC_TIM0->MCR = 0x10;               //
    LPC_TIM0->MR1 = (F_pclk/F_muestreo/2)-1; // DOS Match para iniciar la conversión!!!!
    LPC_TIM0->EMR = 0x00C2;             //
    LPC_TIM0->TCR = 0x01;               //
}
```

2.5. GPDMA: DAC to Memory transfer example (III)

- Configuración del ADC

```
void init_ADC(void)
{
  LPC_SC->PCONP|= (1<<12);            // POwer ON
  LPC_PINCON->PINSEL1|= (1<<14);      // ADC input= P0.23 (AD0.0)
  LPC_PINCON->PINMODE1|= (2<<14);     // Deshabilita pullup/pulldown
  LPC_SC->PCLKSEL0&=~(3<<8);          // CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
  LPC_ADC->ADCR= 0;
  LPC_ADC->ADCR= (0x01<<0)|           // Canal 0
                 (0x01<<8)|           // CLKDIV=1   (Fclk_ADC=25Mhz /(1+1)= 12.5Mhz)
                 (4<<24)|             // Inicio de conversión con el Match 1 del Timer 0
                 (0x01<<21);          // PDN=1
  LPC_ADC->ADINTEN= (1<<0);           // Hab. interrupción fin de conversión canal 0
}
```

2.5. GPDMA: DAC to Memory transfer example (IV)

- Configuración del DMA

```
void init_DMA(void)
{

   //POWER-ON
   LPC_SC->PCONP |= 1 << 29;
   LPC_GPDMA->DMACConfig    |= (1<<0); // Hab. DMA

  //Borramos flags (interrupts. pendientes?)
   LPC_GPDMA->DMACIntTCClear = 0xff;
   LPC_GPDMA->DMACIntErrClr = 0xff;

  //Ponemos a 0 CCControl y CCConfig.
   LPC_GPDMACH0->DMACCConfig = 0;
   LPC_GPDMACH0->DMACCControl =0;

   //Origen y destino.
   LPC_GPDMACH0->DMACCDestAddr = (uint32_t)buffer;
   LPC_GPDMACH0->DMACCSrcAddr = LPC_ADC_BASE+OFFSET_ADC; // ADR0 (0x40003420)

   //Linked list item register, solo una fuente.
   LPC_GPDMACH0->DMACCLLI      = 0;

  //Periferico de origen: ADC y Transfer type - Desde periferico a memoria (P2M)

                              //| S.BURST=1  | D.Burst=1  |  S.Width        |     D.Width     |  Incr. Dest | TC interrupt enable.
   LPC_GPDMACH0->DMACCControl|= (SBURST<<12) |(DBURST<<15)|(WIDTH_VALUE<<18)|(WIDTH_VALUE<<21)| ( 1 << 27 ) | (1 << 31);
   LPC_GPDMACH0->DMACCControl|= TAM_BLOCK_DMA;//Tamaño maximo.

                              //ORIGEN: ADC|   Dest=mem. |    P2M    |    ERROR/TC MASK     | Inicio
   LPC_GPDMACH0->DMACCConfig =   ( 4 << 1 ) | (0x00 << 6) | (2 << 11) | (1 << 15) | (1 << 14)| 1;

  //Habilita la interrupcion del DMA en el NVIC
   NVIC_EnableIRQ(DMA_IRQn);
}
```

2.5. GPDMA: DAC to Memory transfer example (V)

- Función Interrupción del DMA (Error, o fin de la transferencia)

```
void DMA_IRQHandler(void)
{
static uint32_t contador;
  contador++;
  if (LPC_GPDMA->DMACIntStat & 1) //Interrupcion?
   {
        if (LPC_GPDMA->DMACIntTCStat & 1)//Terminal Count Interrupt Request?
          {
          LPC_GPDMA->DMACIntTCClear = 1; //Borramos interrupcion;
          send_buffer_N(buffer,TAM_BLOCK_DMA);
          }
       if (LPC_GPDMA->DMACIntErrStat & 1)//Error Interrupt Request?
          {
          LPC_GPDMA->DMACIntErrClr = 1; //Borramos interrupcion;
          }
   }

}
```

2.5. GPDMA: Basic functions

- LPC17xx.h

```
/*------------- General Purpose Direct Memory Access (GPDMA) ------------------*/
/** @brief  General Purpose Direct Memory Access (GPDMA) register structure definition */
typedef struct                          /* Common Registers                     */
{
  __I  uint32_t DMACIntStat;
  __I  uint32_t DMACIntTCStat;
  __O  uint32_t DMACIntTCClear;
  __I  uint32_t DMACIntErrStat;
  __O  uint32_t DMACIntErrClr;
  __I  uint32_t DMACRawIntTCStat;
  __I  uint32_t DMACRawIntErrStat;
  __I  uint32_t DMACEnbldChns;
  __IO uint32_t DMACSoftBReq;
  __IO uint32_t DMACSoftSReq;
  __IO uint32_t DMACSoftLBReq;
  __IO uint32_t DMACSoftLSReq;
  __IO uint32_t DMACConfig;
  __IO uint32_t DMACSync;
} LPC_GPDMA_TypeDef;

/** @brief  General Purpose Direct Memory Access Channel (GPDMACH) register structure definition */
typedef struct                          /* Channel Registers                    */
{
  __IO uint32_t DMACCSrcAddr;
  __IO uint32_t DMACCDestAddr;
  __IO uint32_t DMACCLLI;
  __IO uint32_t DMACCControl;
  __IO uint32_t DMACCConfig;
} LPC_GPDMACH_TypeDef;
```

2.5. GPDMA: Basic functions

- Example of structure

```
typedef union {
  struct {
    uint32_t TRANSFER_SIZE              : 12;
    uint32_t SOURCE_BURST_SIZE          : 3;
    uint32_t DESTINATION_BURST_SIZE     : 3;
    uint32_t SOURCE_TRANSFER_WIDTH      : 3;
    uint32_t DESTINATION_TRANSFER_WIDTH : 3;
    uint32_t                            : 2;  // reserved and must be written 0
    uint32_t SOURCE_INCREMENT           : 1;  // after each transfer, increment source address(1)/don't increment(0)
    uint32_t DESTINATION_INCREMENT      : 1;  // after each transfer, increment destination address(1)/don't increment(0)
    uint32_t                            : 3;  // Prot1..Prot3, not used in LPC17xx
    uint32_t TERMINAL_COUNT_INT_ENABLE  : 1;  // terminal count interrupt enable
  };
  uint32_t reg;
} LPC_DMAC_CxCONTROL_TypeDef;

typedef struct                          /* Channel Registers                */
{
  __IO uint32_t DMACCSrcAddr;
  __IO uint32_t DMACCDestAddr;
  __IO uint32_t DMACCLLI;
  __IO LPC_DMAC_CxCONTROL_TypeDef DMACCControl;
  __IO LPC_DMAC_CxCONFIG_TypeDef DMACCConfig;
       uint32_t RESERVED[3];
} LPC_GPDMACH_TypeDef;

#define LPC_GPDMACH0_BASE     (LPC_AHB_BASE  + 0x04100)
#define LPC_GPDMACH           ((LPC_GPDMACH_TypeDef    *) LPC_GPDMACH0_BASE )  // to be used as an array
```

2.5. GPDMA: Basic functions

- dma.c dma.h

```
// initialise the DMA controller
void DMA_init(void) {

    /* Select MAT0.0 instead of UART0 Tx */
    LPC_SC->DMAREQSEL = 0x01;

    LPC_SC->PCONP |= 1<<29;     //Power GPDMA module

    /* Enable the GPDMA controller */
    LPC_GPDMA->DMACConfig = 1;

    /* Enable synchro logic request 8, MAT0.0 */
    LPC_GPDMA->DMACSync   = 1 << 8;
}
```

```
#ifndef DMA_H_
#define DMA_H_

#include "LPC17xx.h"
#include <defs.h>

void DMA_init(void);
void DMA_config(signed long* outputBuffer);
void DMA_waitForTransfer(void);

#endif /* DMA_H_ */
```

2.5. GPDMA: Basic functions

```
// Configure the DMA controller
void DMA_config(signed long* outputBuffer) {
    LPC_GPDMACH0->DMACCSrcAddr  = (uint32_t) outputBuffer;
    LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR);
    LPC_GPDMACH0->DMACCLLI      = 0;            // linked lists for ch0
    LPC_GPDMACH0->DMACCControl  = bufferSize// transfer size (0 - 11) = 32
            | (0 << 12)         // source burst size (12 - 14) = 1
            | (0 << 15)         // destination burst size (15 - 17) = 1
            | (2 << 18)         // source width (18 - 20) = 32 bit
            | (2 << 21)         // destination width (21 - 23) = 32 bit
            | (0 << 24)         // source AHB select (24) = AHB 0
            | (0 << 25)         // destination AHB select (25) = AHB 0
            | (1 << 26)         // source increment (26) = increment
            | (0 << 27)         // destination increment (27) = no increment
            | (0 << 28)         // mode select (28) = access in user mode
            | (0 << 29)         // (29) = access not bufferable
            | (0 << 30)         // (30) = access not cacheable
            | (0 << 31);        // terminal count interrupt disabled

    LPC_GPDMACH0->DMACCConfig   =  1            // channel enabled (0)
            | (0 << 1)          // source peripheral (1 - 5) = none
            | (8 << 6)          // destination request peripheral (6 - 10) = MAT0.0
            | (1 << 11)         // flow control (11 - 13) = mem to per
            | (0 << 14)         // (14) = mask out error interrupt
            | (0 << 15)         // (15) = mask out terminal count interrupt
            | (0 << 16)         // (16) = no locked transfers
            | (0 << 18);        // (27) = no HALT
}
```

```
void DMA_waitForTransfer(void) {
    // wait for the DMA to finish
    while (LPC_GPDMACH0->DMACCConfig & 1);
}
```