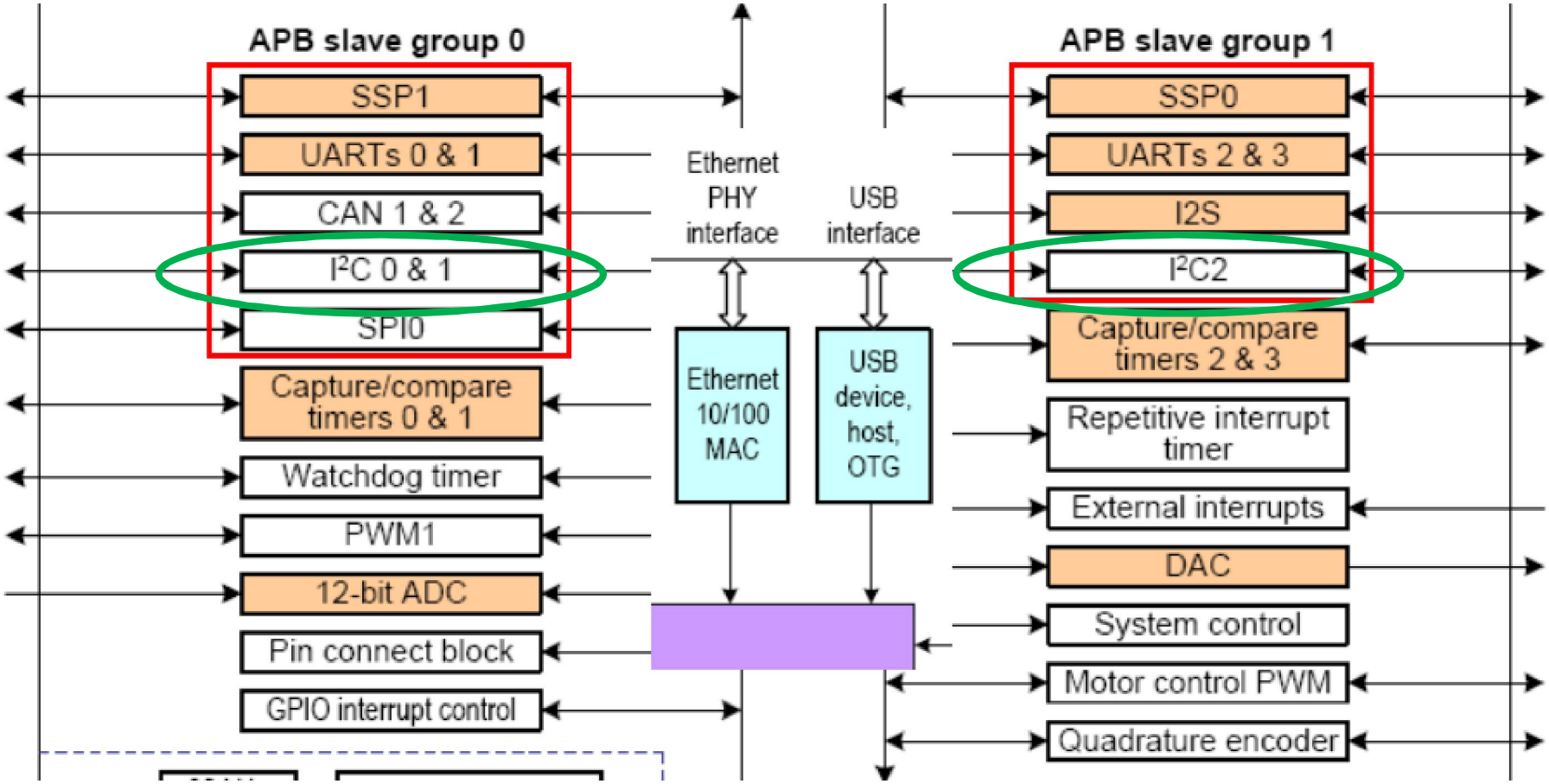
Serial Interfaces

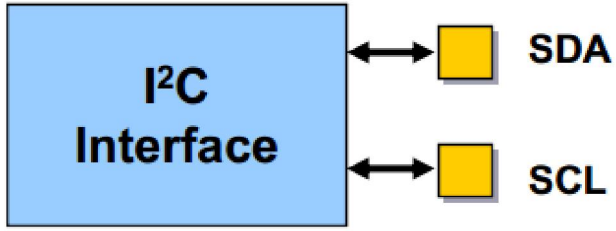
(I2C, SPI/SSP, I2S, CAN, USB)

Serial Interfaces: I2C



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**)

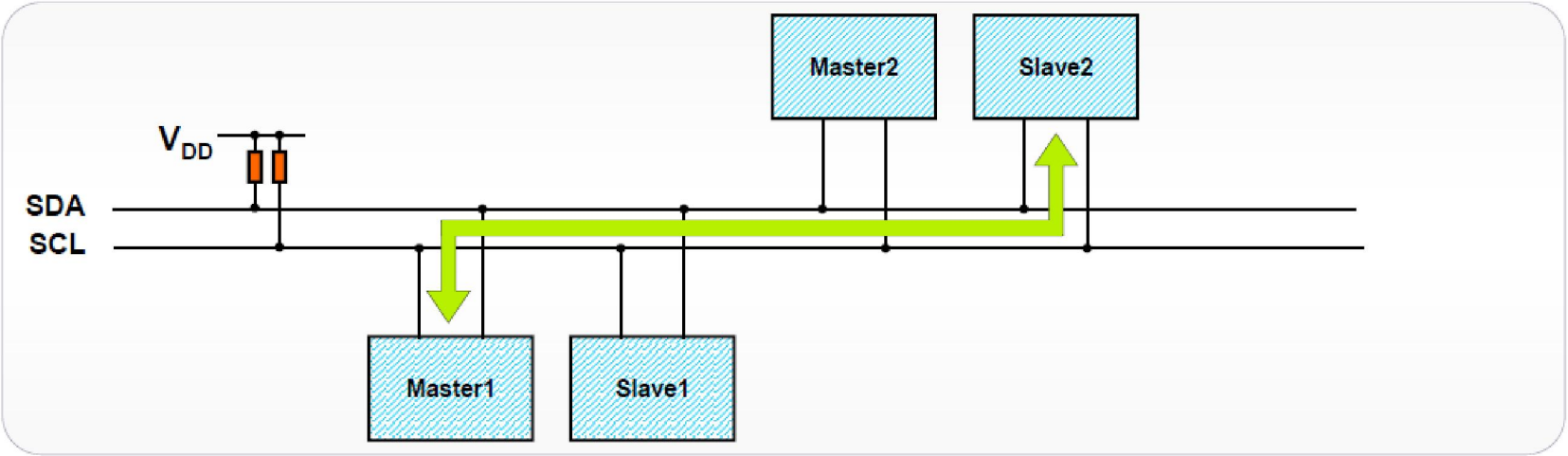


- Multi-master capable bus with arbitration feature.
- Master-Slave communication; Two-device only communication.
- Each IC on the bus is identified by its own **address code**.
- I2C compliant bus interface, and can be configured as Master, Slave, or Master/Slave.
- Bi-directional data transfer between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

Source: DesignCon 2003 TecForum I2C Bus Overview

I2C: Hardware architecture

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



I2C: Modes (Speed)

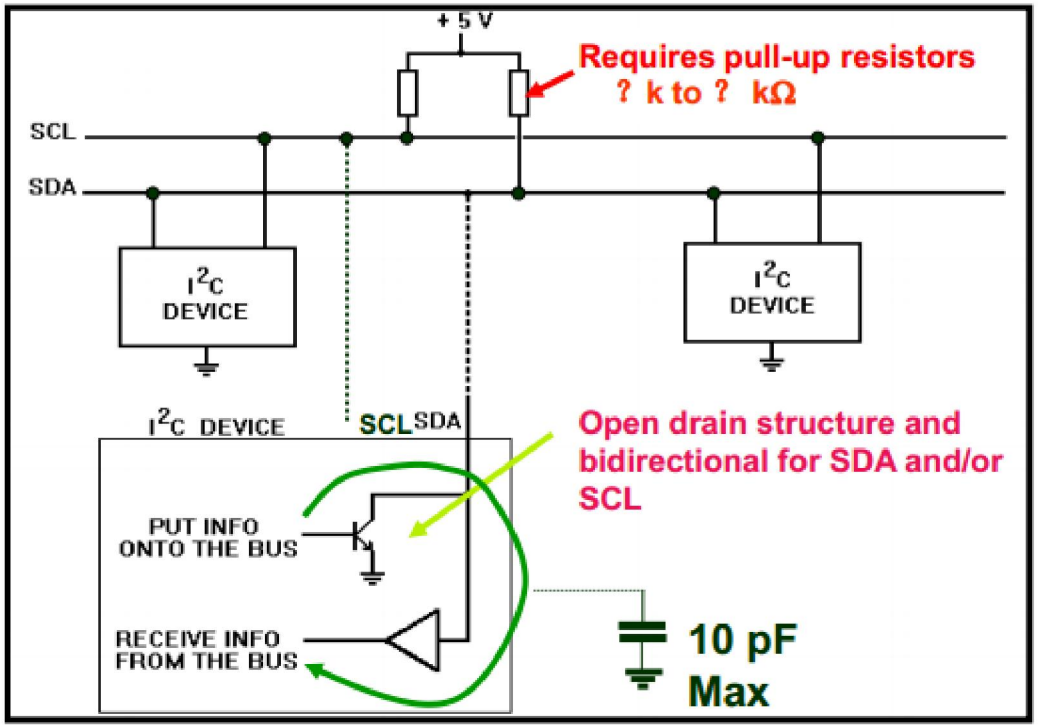
	Standard Mode	Fast Mode	Fast Mode Plus (FM+)	High Speed Mode	
Bitrate (kBit/s)	0 – 100	0 – 400	0 – 1000	0 – 1700	0 – 3400
Address (bits)	7 (10)	7 (10)	7 (10)	7 (10)	7 (10)
Capacitive Bus Load (pF)	400	400	550	400	100
Sink current (mA)	3	3	20	3	3

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

Source: DesignCon 2003 TecForum I2C Bus Overview

I2C: SDA/SCL Driver Architecture

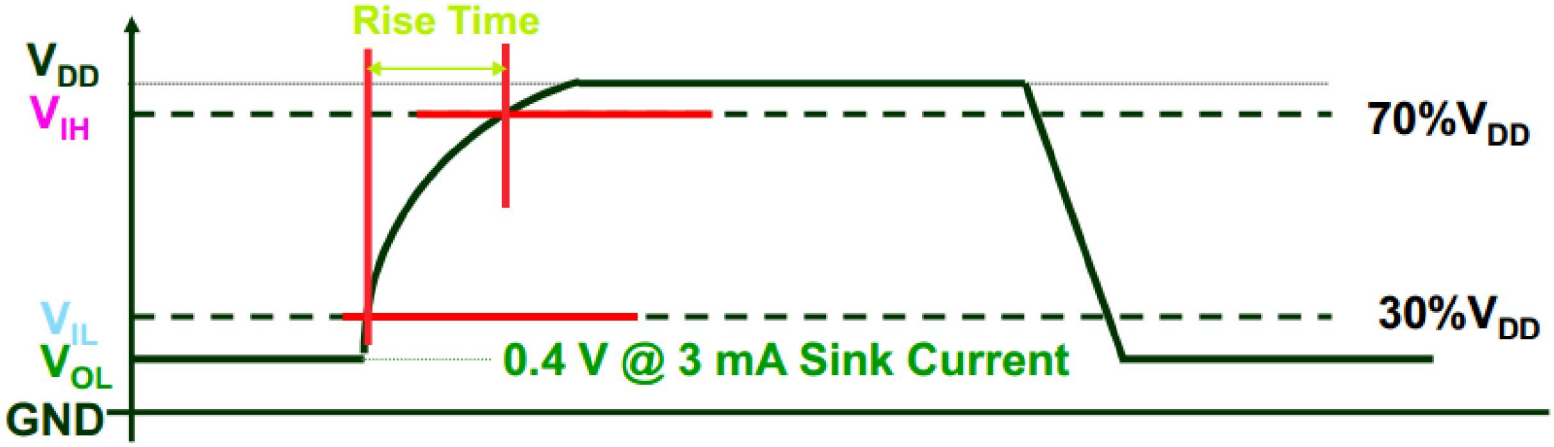
- SDA and SCL are open drain/collector
  - Required pull-up resistors to pull the line to logic "1"
- Clock stretch is possible when a device is busy



Source: DesignCon 2003 TecForum I2C Bus Overview

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

I2C: Calculating Pull-up Resistors

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

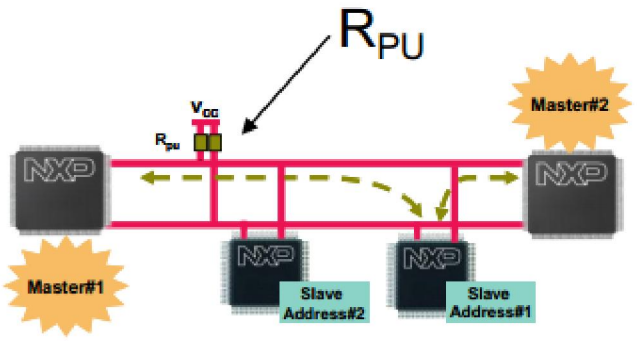
2) $R_{MIN} = (V_{DDMAX} - V_{OLMAX}) / I_{OLMAX}$

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**

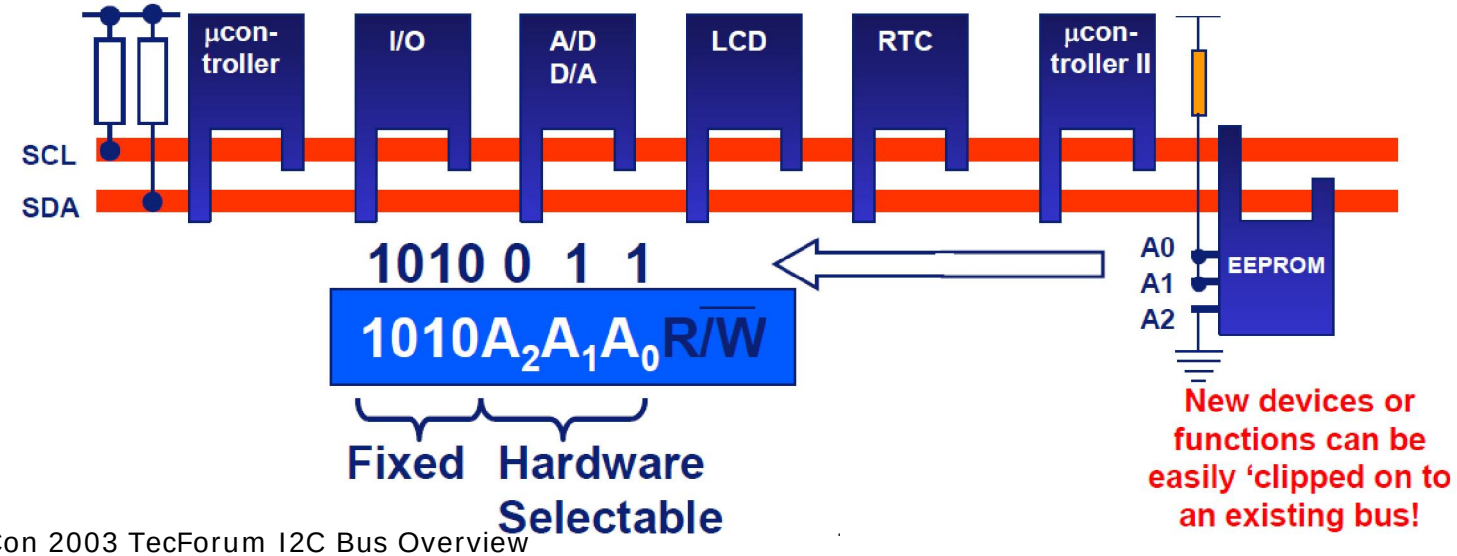


Glossary

- 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

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

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.

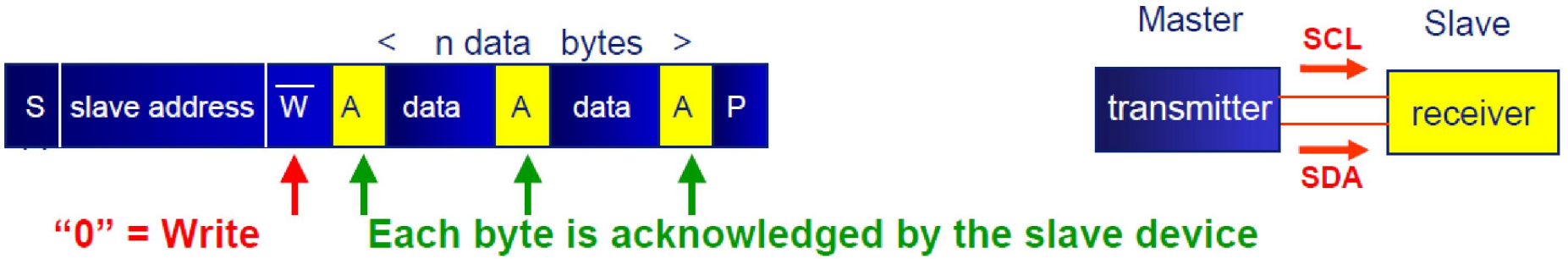
- Example:

Transmit (0 = Write)

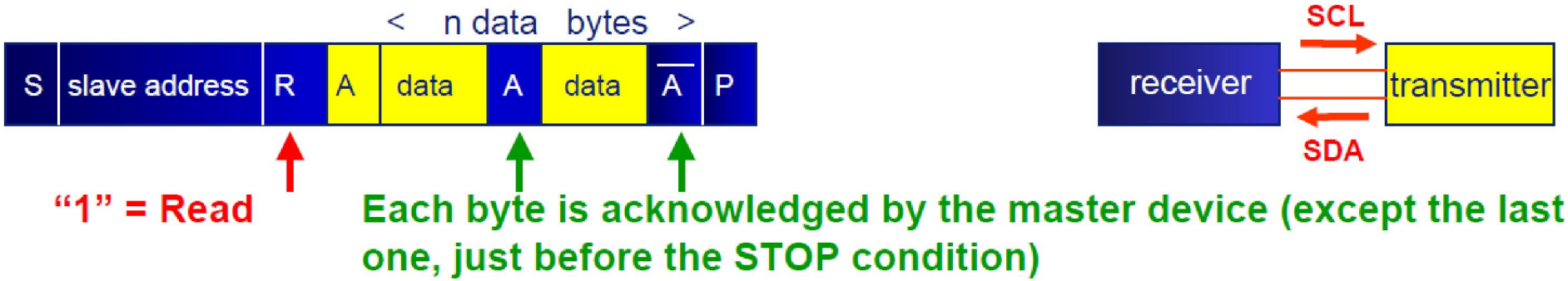
Receive (1 = Read)

I2C: Protocol (II)

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

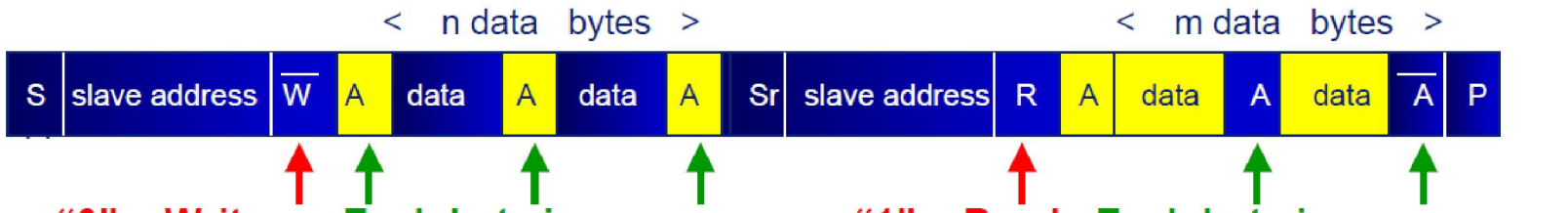


- **Read from a Slave device:**
  - The master is a "MASTER TRANSMITTER then MASTER -RECEIVER":
    - it transmits Clock all the time.
    - it sends slave address data and then becomes a receiver.

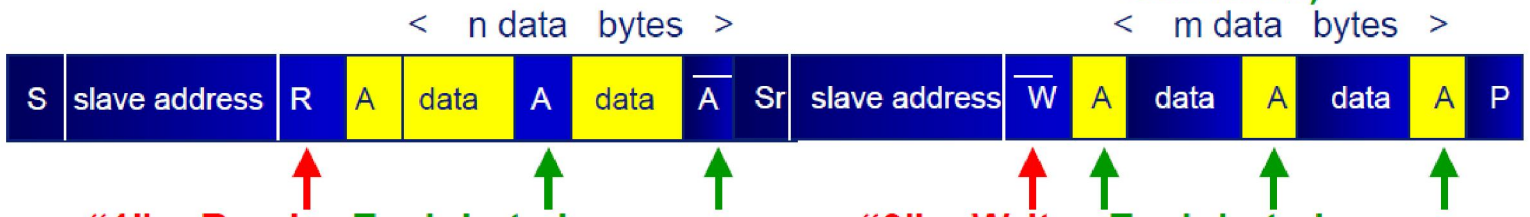


I2C: Protocol (III)

- **Combined Write and Read**



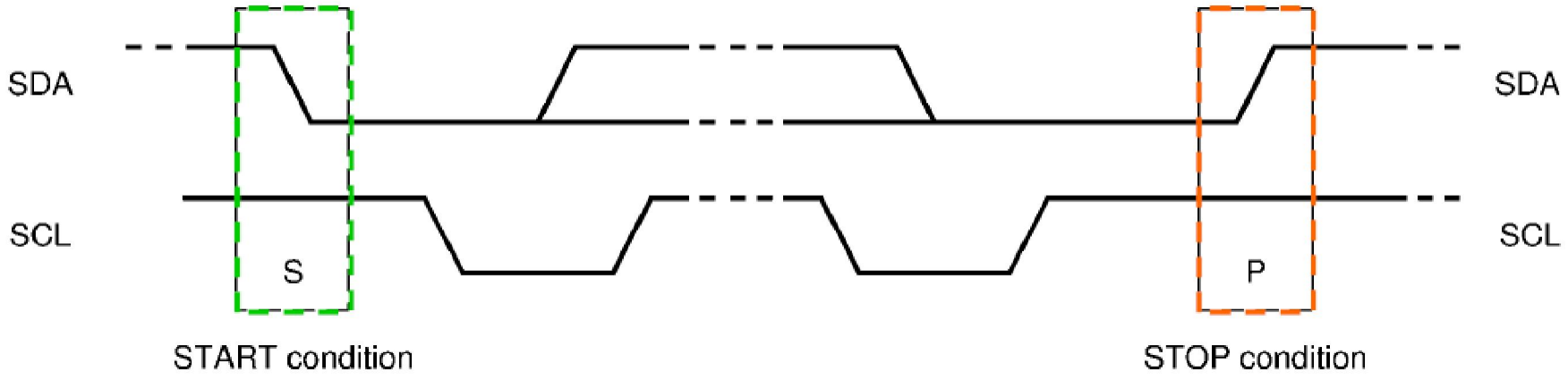
- **Combined Read and Write**



Source: DesignCon 2003 TecForum I2C Bus Overview

I2C: Protocol (IV)

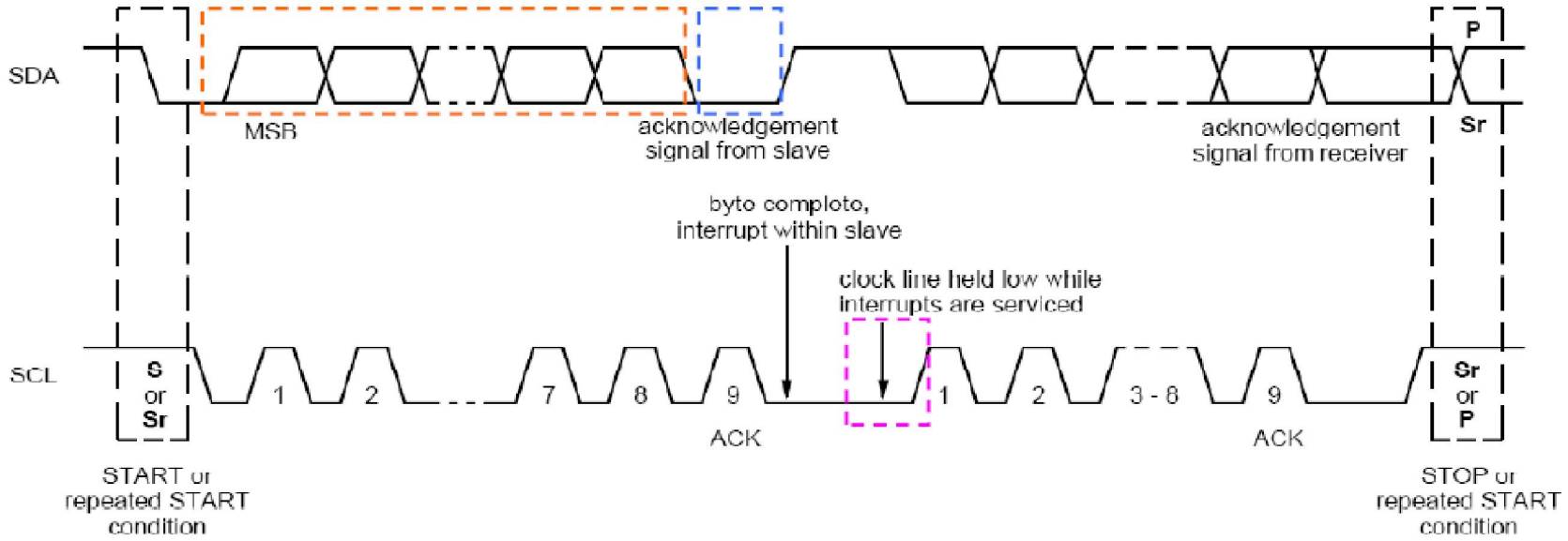
- **START & STOP conditions.**
  - **Start** condition: a HIGH to LOW transition on the SDA line while SCL is HIGH.
  - **Stop** condition: a LOW to HIGH transition on the SDA line while SCL is HIGH.



Source: DesignCon 2003 TecForum I2C Bus Overview

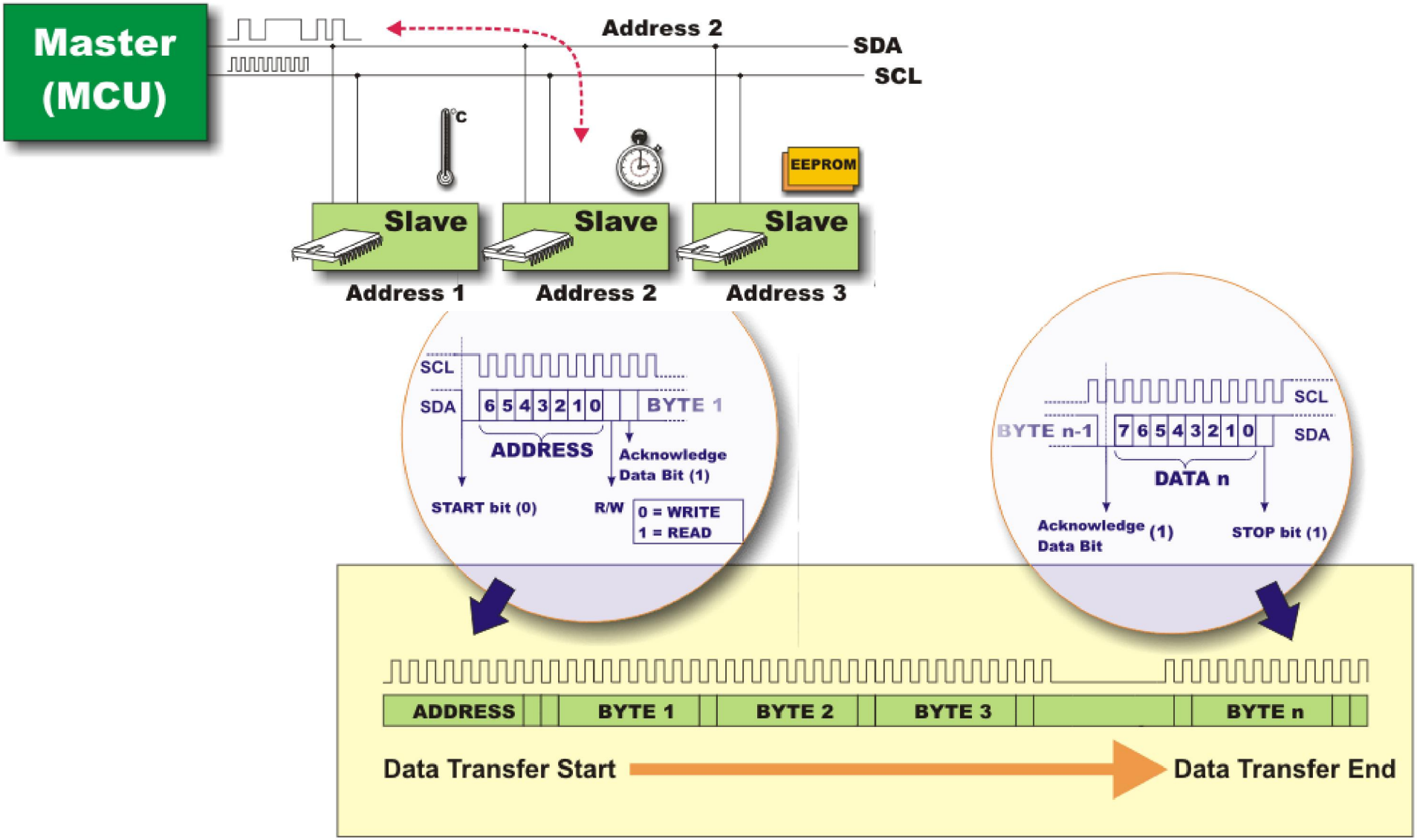
I2C: Protocol (V)

- **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.

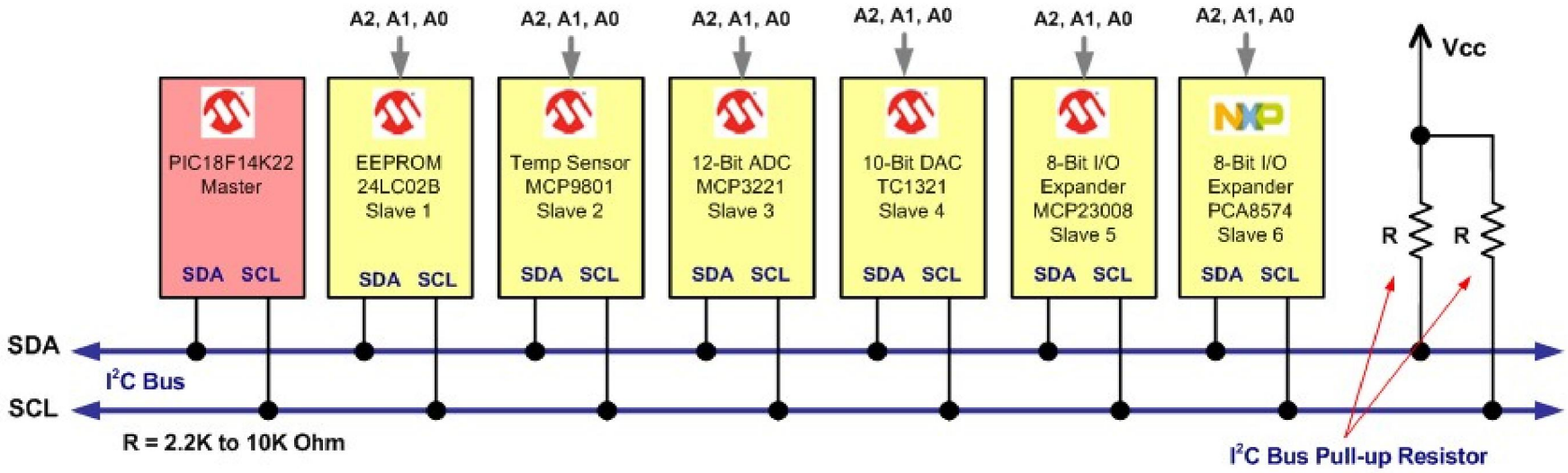


Source: DesignCon 2003 TecForum I2C Bus Overview

I2C: Data Transfer Example



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)

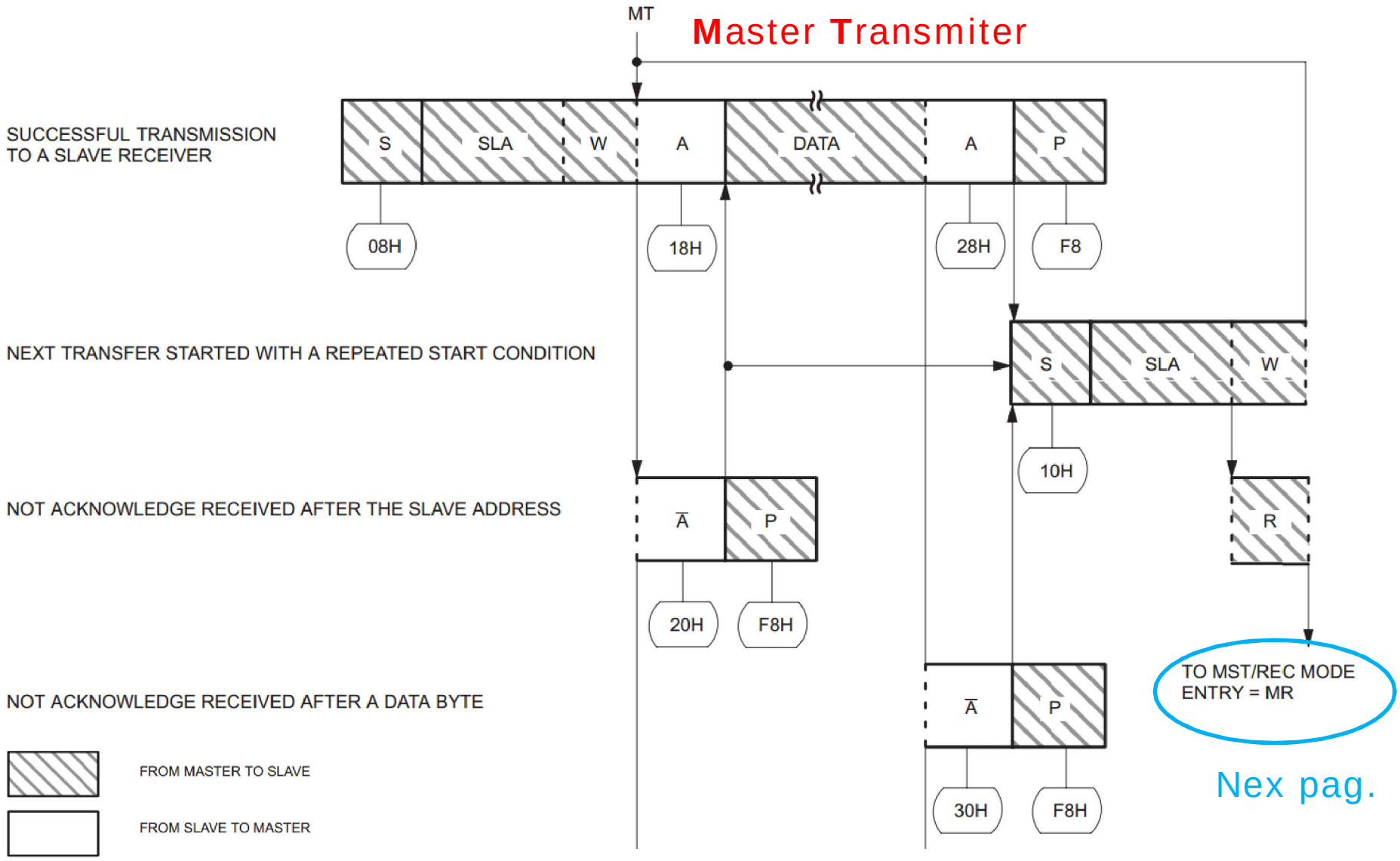
ID3	ID2	ID1	ID0	A2	A1	A0	R/W

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

I2C Registers

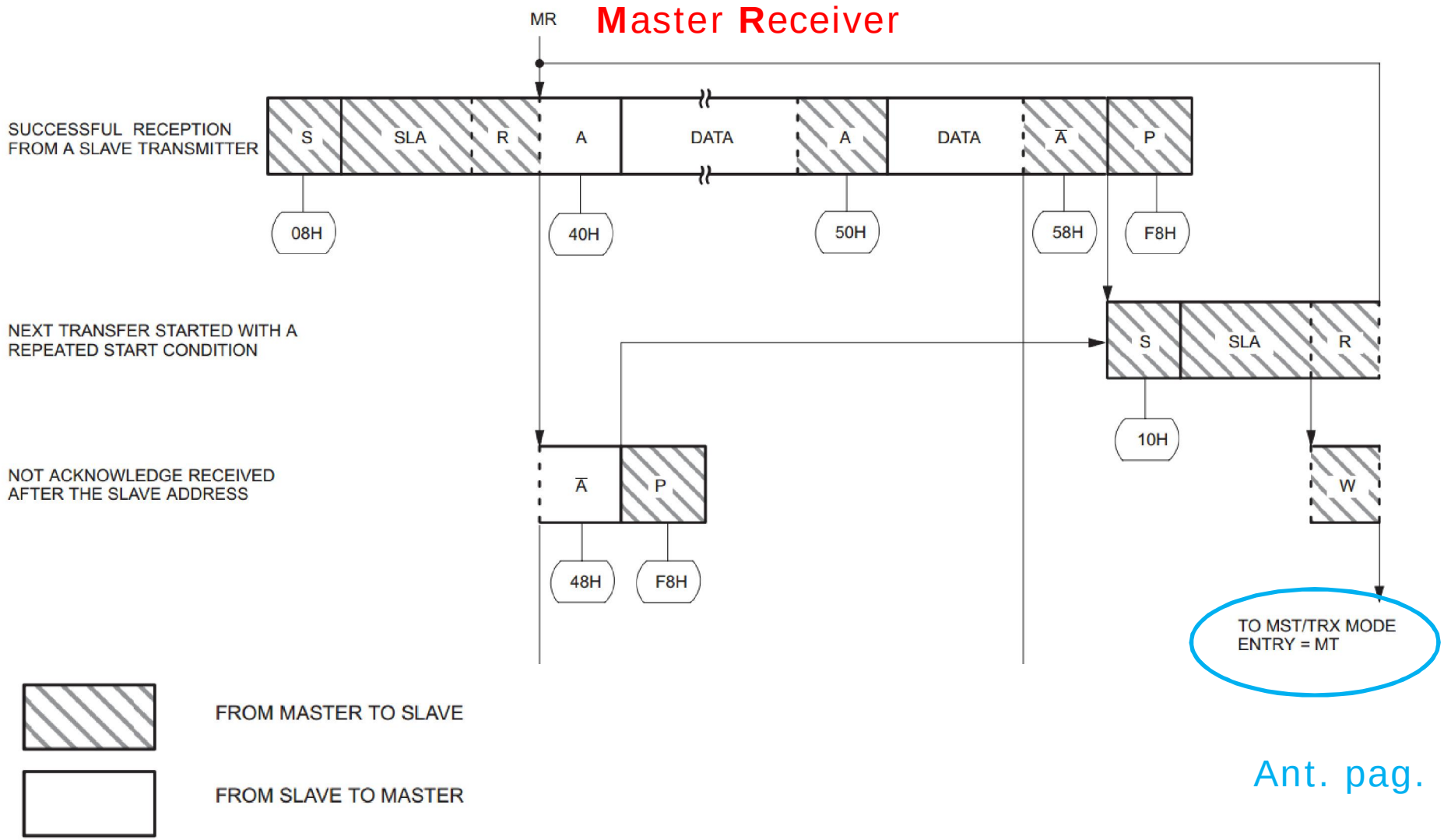
Name	Access	Description	Address Offset
I2C0CONSET	R/W	I2C Control Set Register	0x0000
I2C0STAT	RO	I2C status Register	0x0004
I2C0DAT	R/W	I2C Data Register	0x0008
I2C0ADR0	R/W	I2C Slave Address Register 0	0x000C
I2C0SCLH	R/W	SCH Duty Cycle Register High Half Word	0x0010
I2C0SCLL	R/W	SCL Duty Cycle Register 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

I2C: Format and States in MT mode



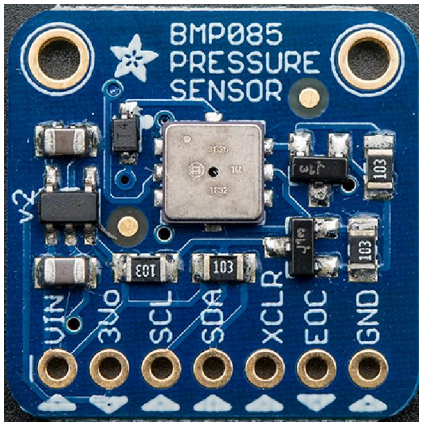
Nex pag.

I2C: Format and States in MR mode

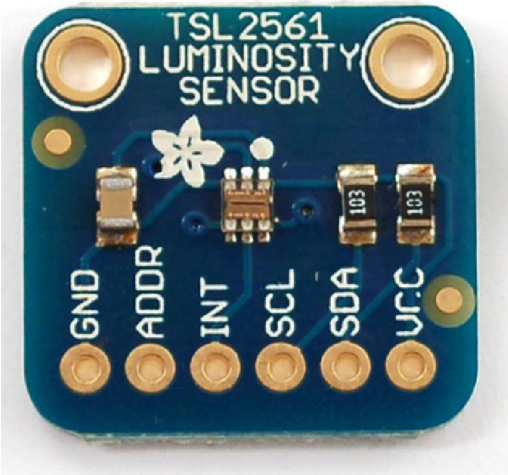


Ant. pag.

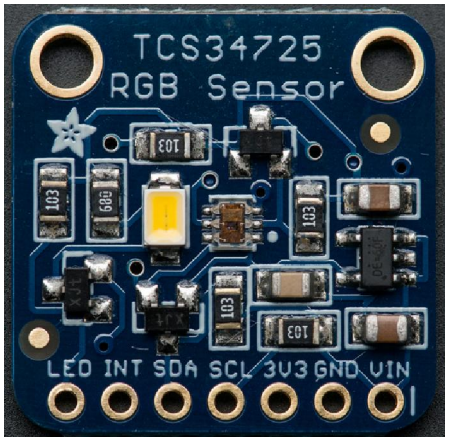
I2C: Devices with I2C interface



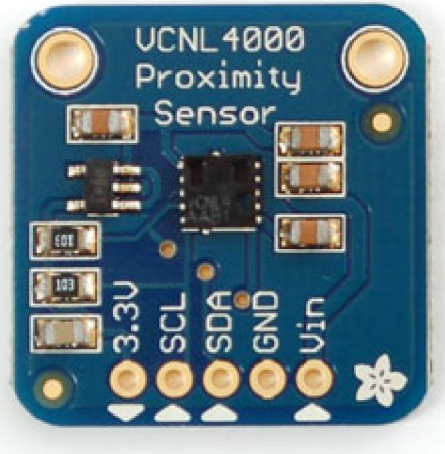
Barometric Pressure Temperature/Altitude Sensor



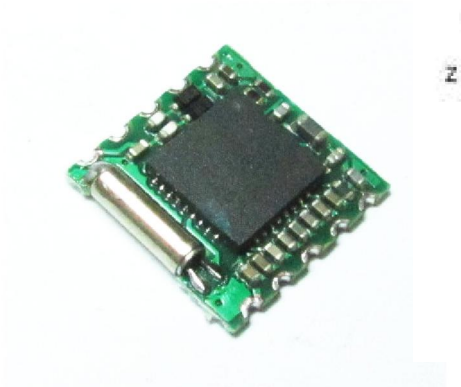
Luminosity Sensor



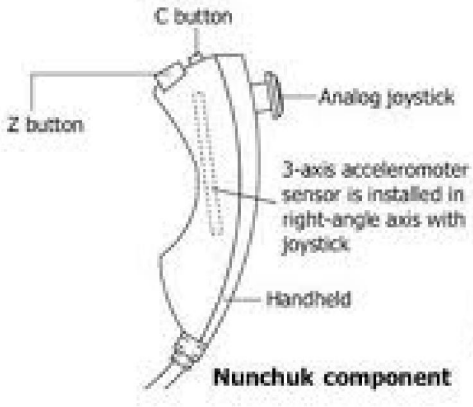
RGB Sensor

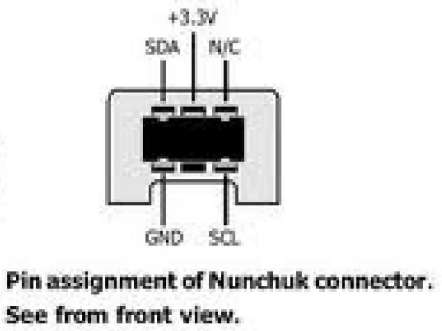


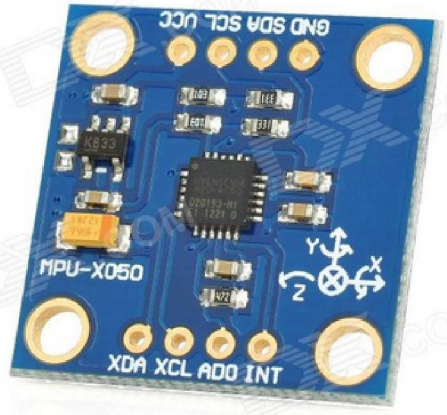
Proximity Sensor



FM Digital Tuner Module

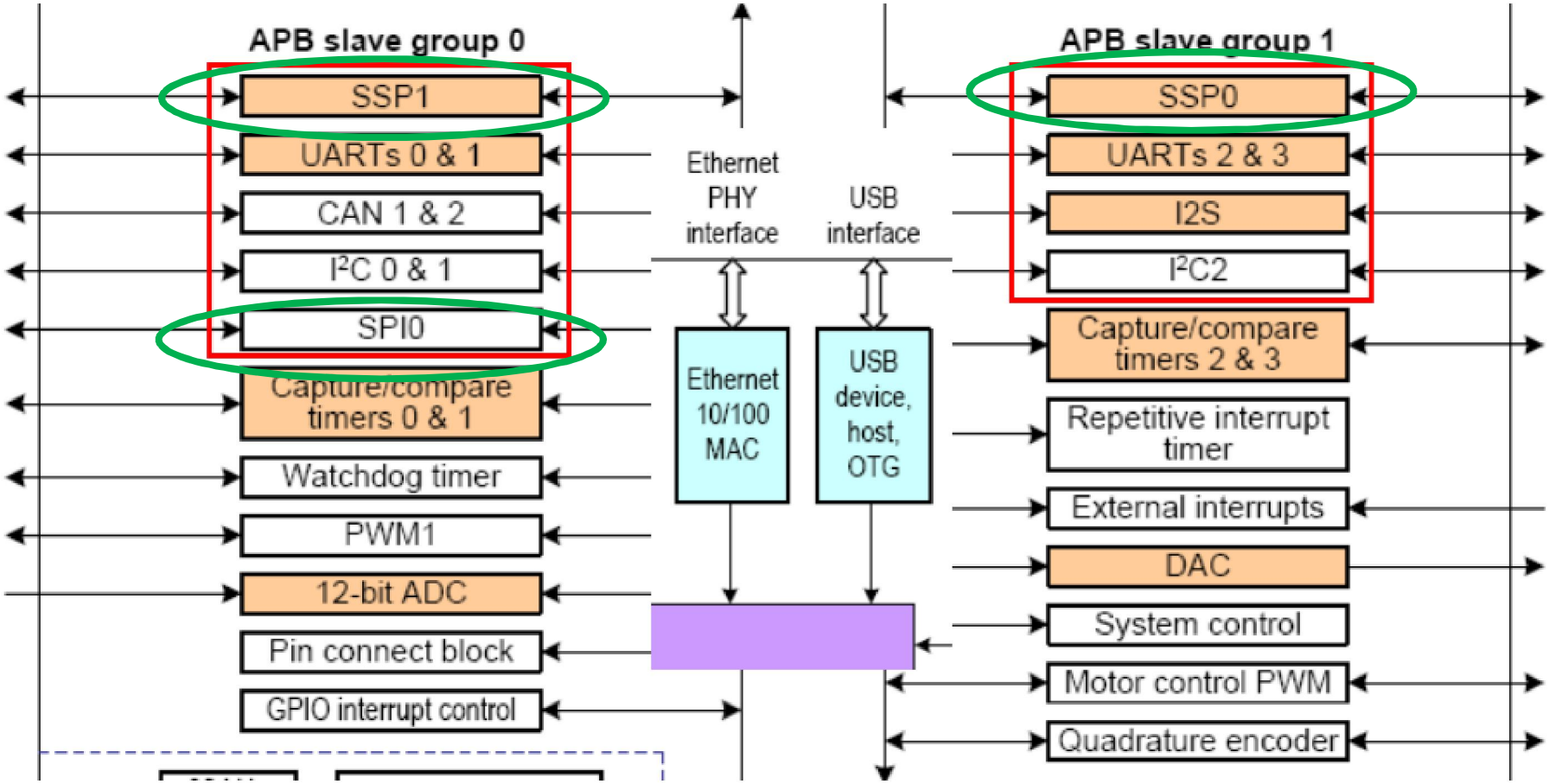






Acelerometer Sensor

Serial Interfaces: SPI, SSP

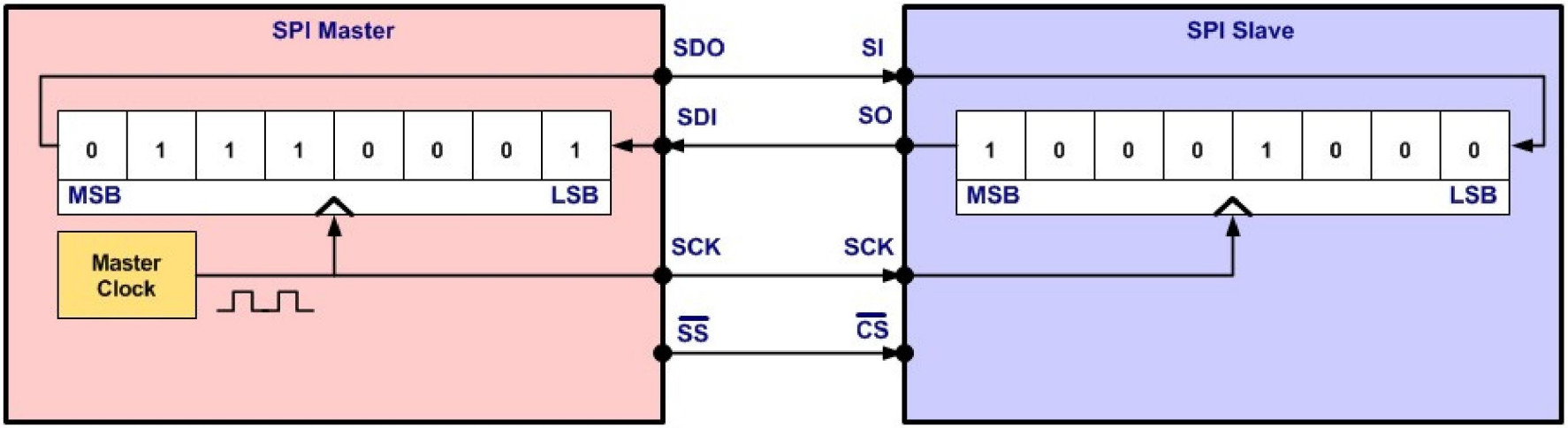


What SPI ?

- **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

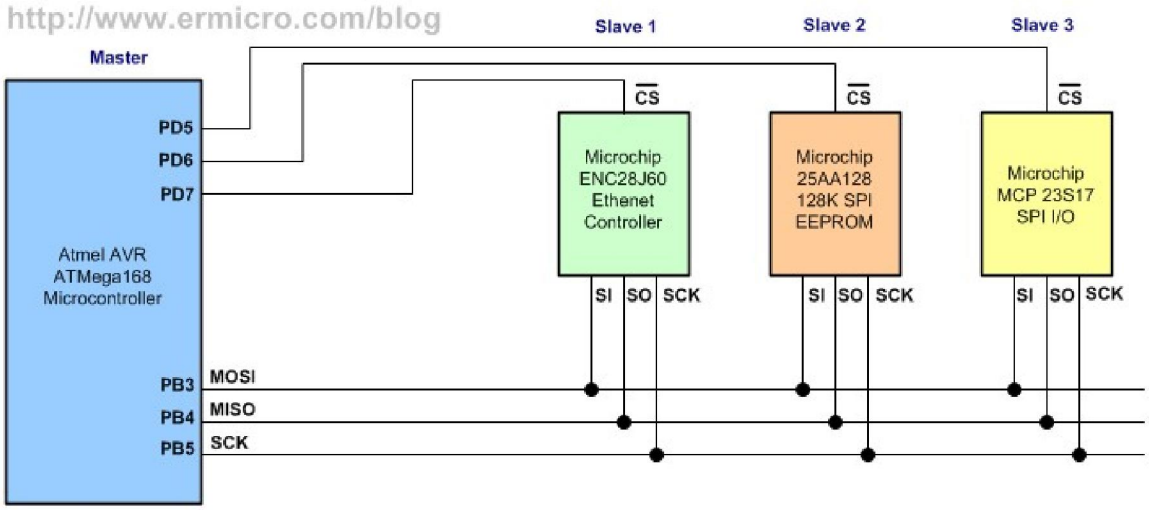
SPI: Master-Slave basic connection



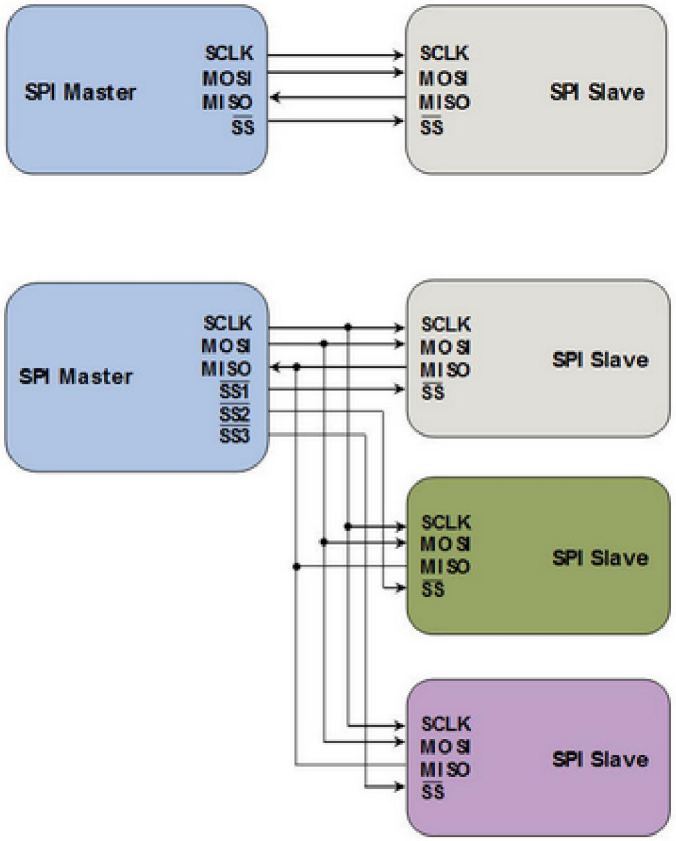
PIN	Description	Direction
SDO	Serial Data Out, This pin also called as MOSI (Master Out Serial In) in AVR Type Microcontroller	Out
SDI	Serial Data In, This pin also called as MISO (Master In Serial Out) in AVR Type Microcontroller	In
SCK	Serial Clock (SPI Master)	Out
SS	Serial Select (could be any I/O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

SPI: Master-Slave basic connection

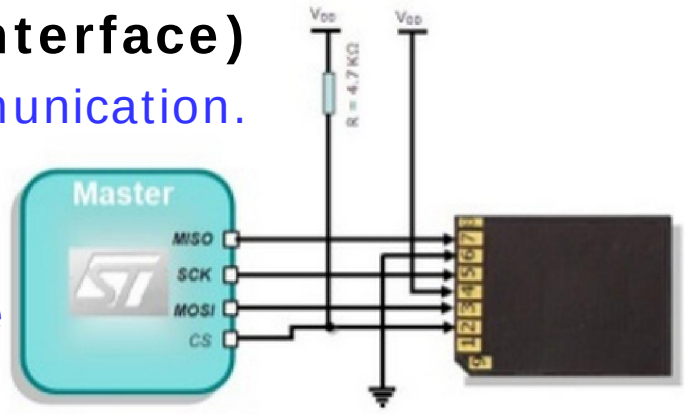


Typical SPI Master with Multiple SPI Slave Device Connection



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)

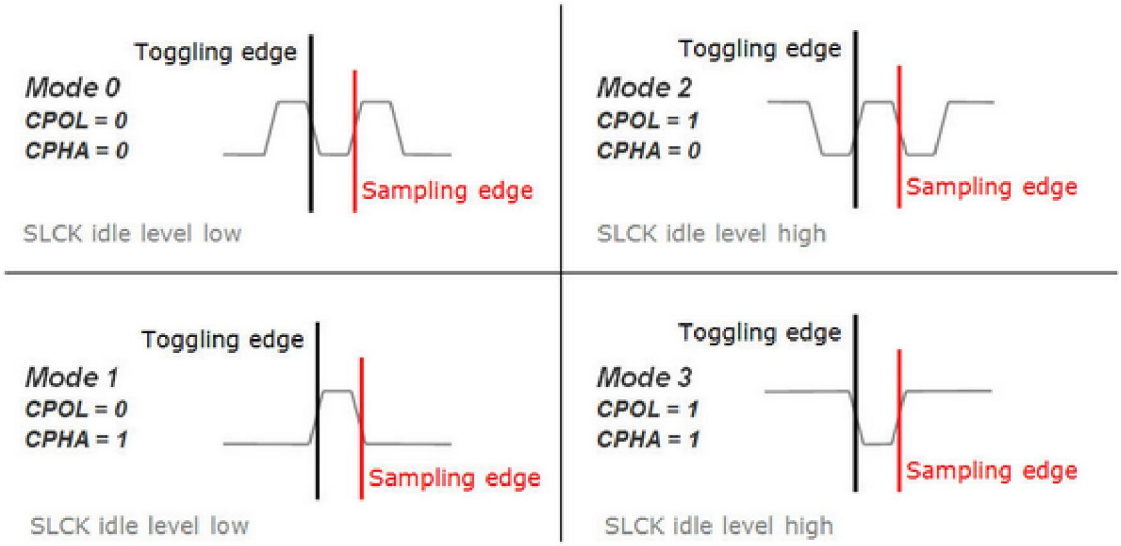
SPI: pins

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.

SPI: Modes

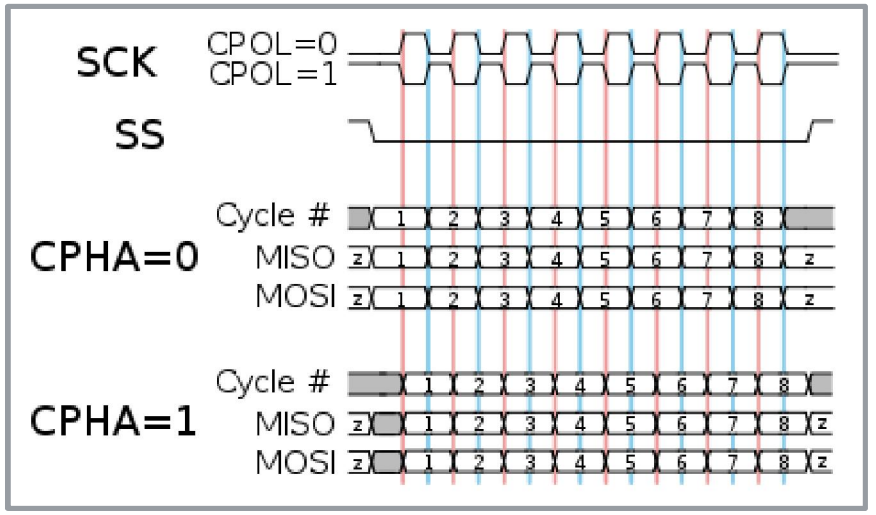
- **Four** communication modes are available.
- Basically define the SCLK edge on which the MOSI line toggles, the SCLK edge on which the master samples the MISO line and the SCLK signal steady level (that is the clock level, high or low, when the clock is not active).
- Each mode is formally defined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



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

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

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

SPI: Status Register (S0SPSR)

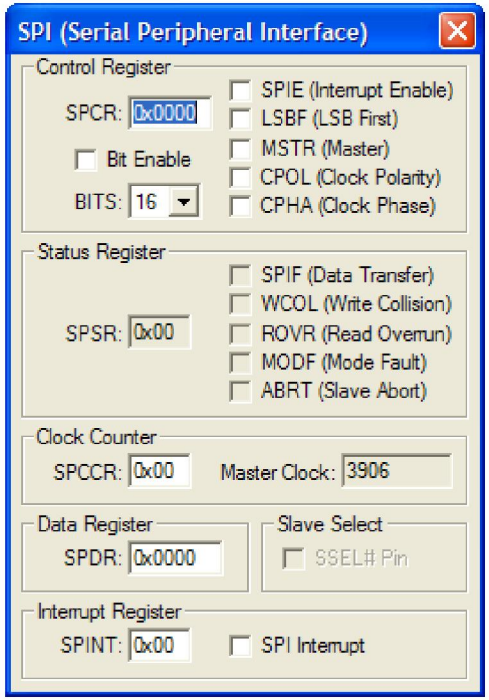
Table 362: SPI Status Register (S0SPSR - address 0x4002 0004) bit description

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

SPI: Software functions

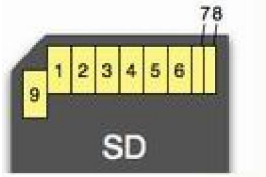
```
unsigned char  spi_transfer(unsigned char data)
{
//SLAVE_ENABLE;                          // SS=0
LPC_SPI->SPDR = data;                     // send a byte
while ((LPC_SPI->SPSR & (1<<7)) == 0)  ;// WAIT until data reg. is empty
return  LPC_SPI->SPDR;                    // return the result
//SLAVE_DISABLE;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

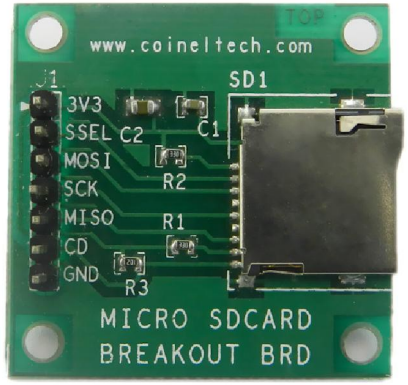


SPI: Devices

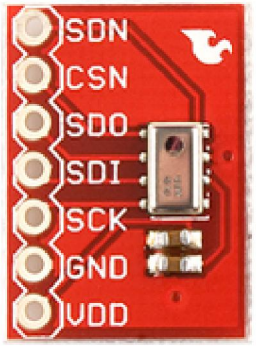
SDCard holder



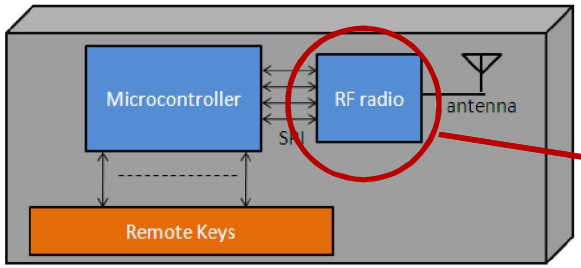
Pin	SD	SPI
1	CD/DAT3	CS
2	CMD	DI
3	VSS1	VSS1
4	VDD	VDD
5	CLK	SCLK
6	VSS2	VSS2
7	DAT0	DO
8	DAT1	X
9	DAT2	X



Micro SD holder



Barometric Pressure

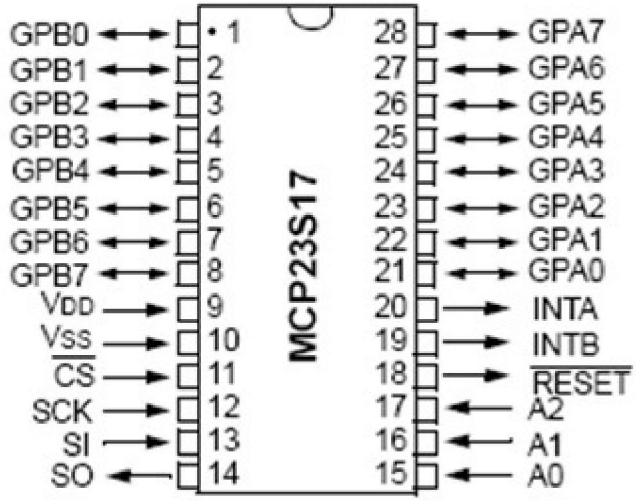


NRF905 Wireless TX/RX

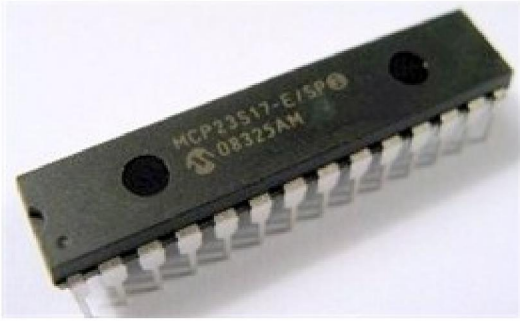
7-seg. Display module

MCP23S17 : 16-bit SPI I/O Expander

- **Transfer format**



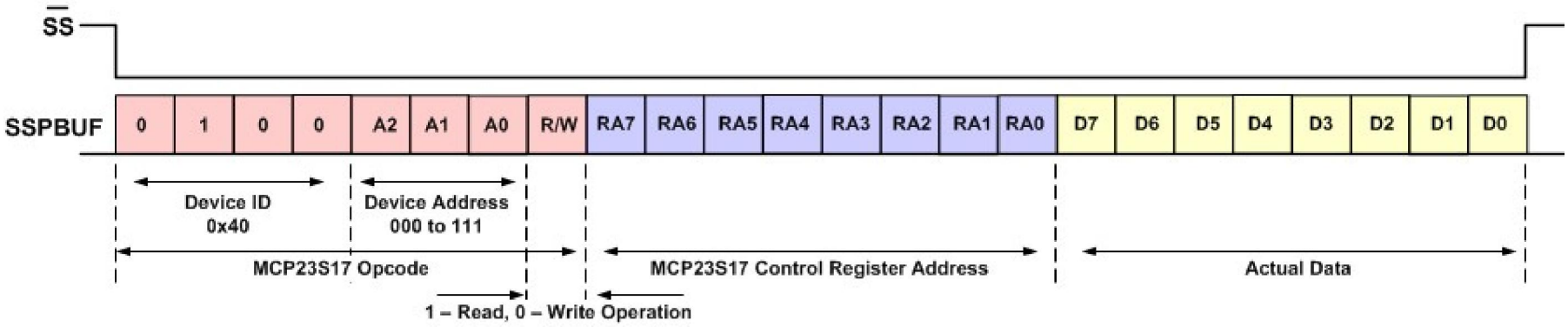
Pin Name	Description
GPA0..7	8-bit General I/O Port A
GPB0..7	8-bit General I/O Port B
INTA	Port A Interrupt Signal
INTB	Port B Interrupt Signal
RESET	Reset Signal
A0,A1,A2	Configurable Address
CS	Chip Select (active Low)
CSK	Synchronous Clock
SI	Slave In
SO	Slave Out
V_{DD}	+5 Volt
V_{SS}	GND



Microchip MCP23S17 SPI 16-bit I/O Expander

MCP23S17 : 16-bit SPI I/O Expander

Transfer format



MCP23S17 Control Registers Address for IOCONA.BANK=0

Register	Description	Address	Register	Description	Address
IODIRA	I/O Direction Register A	0x00	IOCONB	I/O Expander Configuration	0x0B
IODIRB	I/O Direction Register B	0x01	GPPUA	GPIO Pull-up Resistor A	0x0C
IPOLA	Input Polarity Port Register A	0x02	GPPUB	GPIO Pull-up Resistor B	0x0D
IPOLB	Input Polarity Port Register B	0x03	INTFA	Interrupt Flag Register A	0x0E
GPINTENA	Interrupt on Change Pins A	0x04	INTFB	Interrupt Flag Register B	0x0F
GPINTENB	Interrupt on Change Pins B	0x05	INTCAPA	Interrupt Capture Value A	0x10
DEFVALA	Default Value Register A	0x06	INTCAPB	Interrupt Capture Value B	0x11
DEFVALB	Default Value Register B	0x07	GPIOA	General Purpose I/O A	0x12
INTCONA	Interrupt On Change Cont . A	0x08	GPIOB	General Purpose I/O B	0x13
INTCONB	Interrupt On Change Cont . B	0x09	OLATA	Output Latch Register A	0x14
IOCONA	I/O Expander Configuration	0x0A	OLATB	Output Latch Register B	0x15

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

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);
}
```

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));
}
```

I)

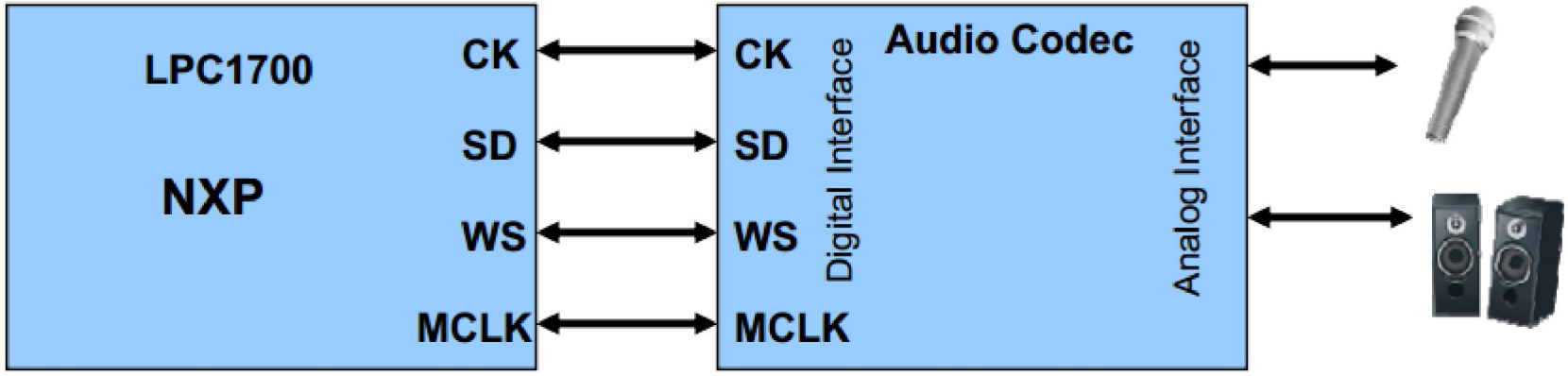
```
void LCD_WriteData( unsigned short data)
{
        SPI_CS_LOW;

        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;
}
```

SPI: GLCD library example

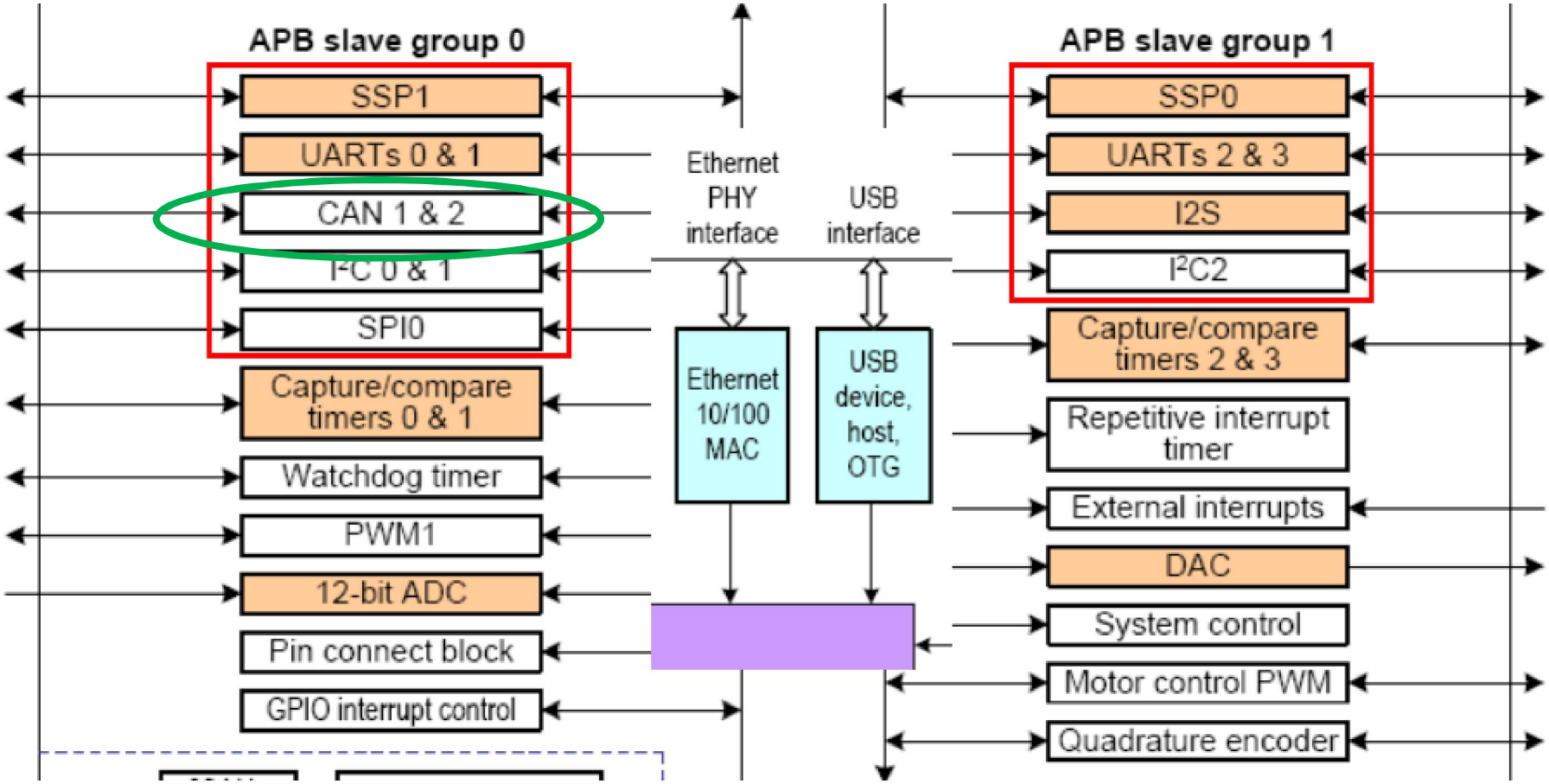
```
unsigned short LCD_ReadData(void)
{
        unsigned short value;

        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;
}
```

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.

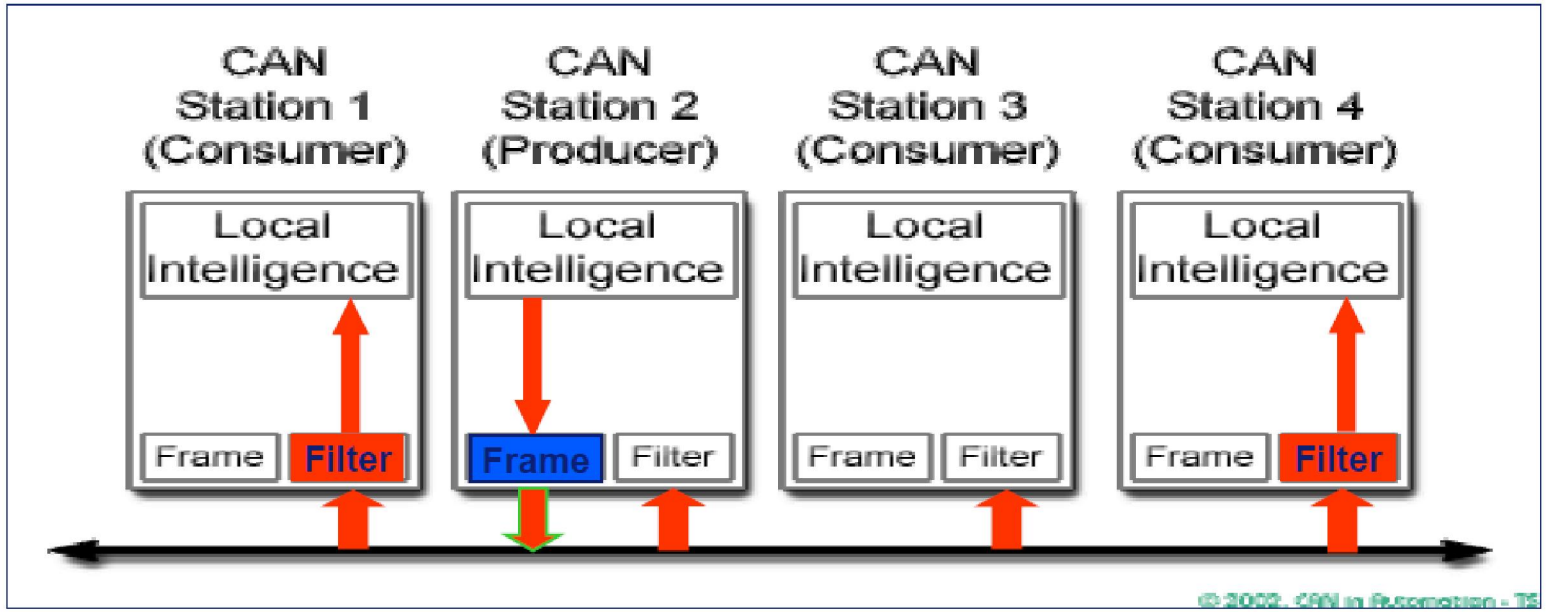


Serial Interfaces: CAN



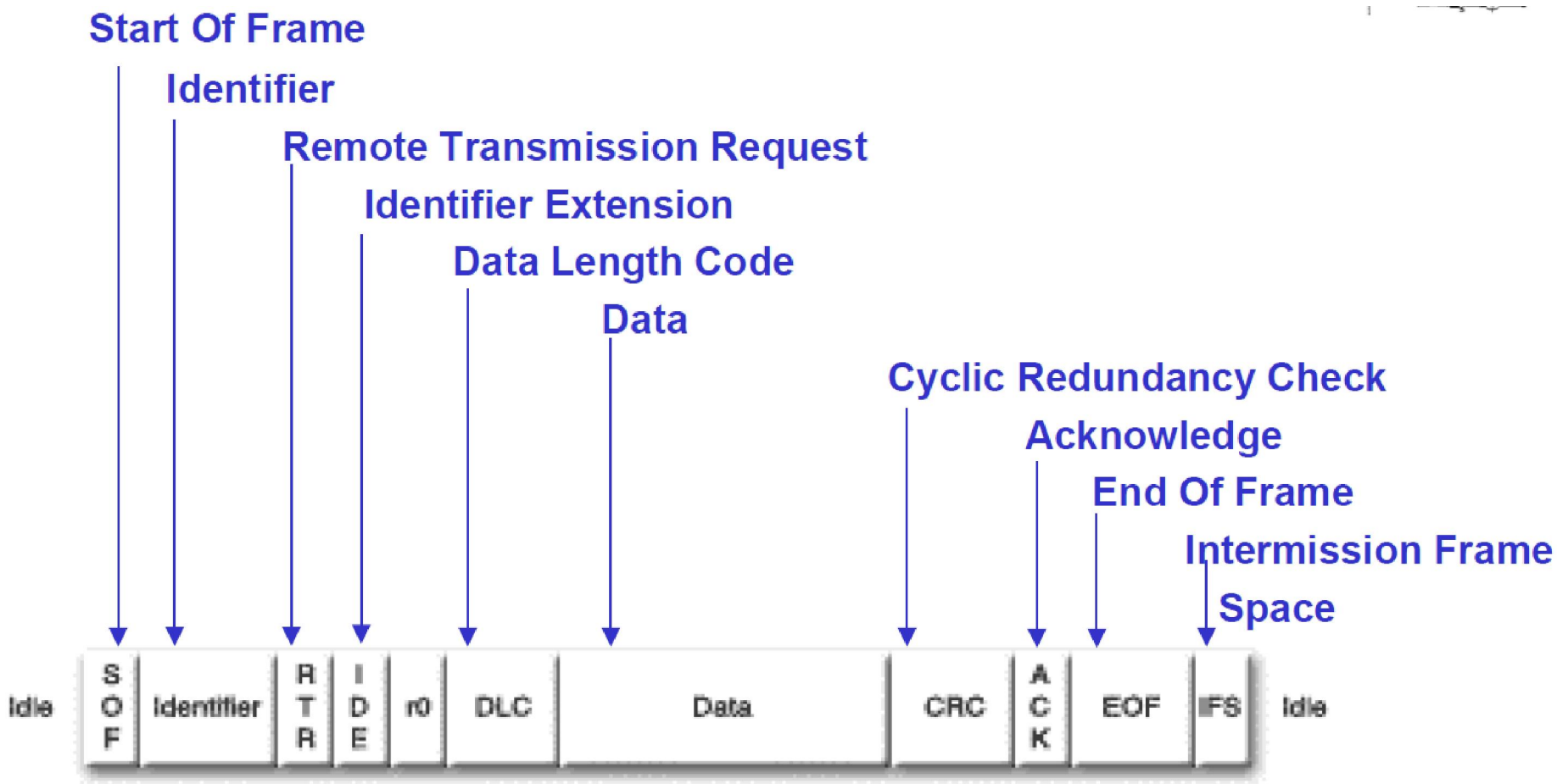
CAN: Controller Area Network

- 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

CAN: Protocol



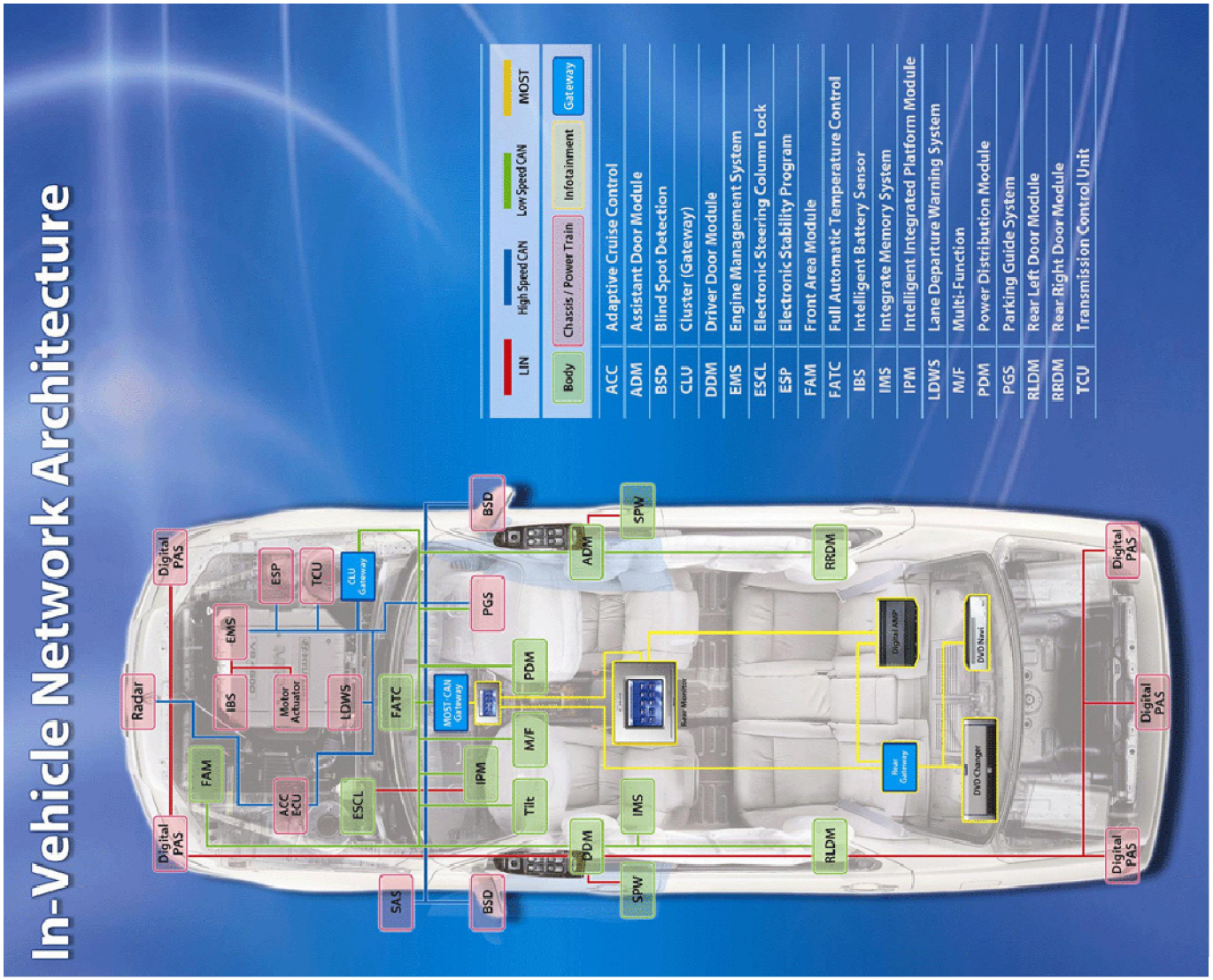
Source: DesignCon 2003 TecForum I2C Bus Overview

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

CAN: In-Vehicle Network Architecture



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

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

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

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

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