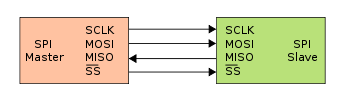
Serial Peripheral Interface

* SPI is a Synchronous Serial Communication.
* SPI is a Master/Slave Communication.
* SPI have only 1 Master and multi-Slaves.
* Totally Full-Duplex communication.
* No Bus arbitration.
* Independent slaves.
* Cascade Slave Communication



The SPI bus specifies four logic signals.

\* SCLK — Serial Clock (output from master)

\* MOSI/SIMO — Master Output, Slave Input (output from master)

\* MISO/SOMI — Master Input, Slave Output (output from slave)

\* SS — Slave Select (active low; output from master)

* SPI is most often employed in systems for communication between the central processing unit ( CPU ) and peripheral devices. It is also possible to connect two microprocessors by means of SPI.

Advantages

\* Full duplex communication

\* Higher throughput than I²C or SMBus

\* Extremely simple hardware interfacing

o Typically lower power requirements than I²C or SMBus due to less circuitry (including pullups)

o No arbitration or associated failure modes

o Slaves use the master's clock, and don't need precision oscillators

Disadvantages

\* No hardware slave acknowledgment

\* Supports only one master device

\* No error-checking protocol is defined

\* Only handles short distances compared to RS-232, RS-485, or CAN-bus

**To begin a communication**:

1. the master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 1-70 MHz.
2. The master then pulls the slave select low for the desired chip. If a waiting period is required (such as for analog-to-digital conversion) then the master must wait for at least that period of time before starting to issue clock cycles.
3. During each SPI clock cycle, a full duplex data transmission occurs:

\* the master sends a bit on the MOSI line; the slave reads it from that same line

\* the slave sends a bit on the MISO line; the master reads it from that same line

Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it, such as writing it to memory. If there is more data to exchange, the shift registers are loaded with new data and the process repeats.

Inter-Integrated Circuit(I2C)

Features:

* Multi-Master Bus.
* half duplex
* 7 bit address space Maximum 112 nodes(16 reserved).
* Normal Mode 100 Kb/s (The most common).
* Fast Mode 400 kb/s.
* High-Speed Mode 3.4 Mb/s.
* Clock Stretching.

combined: y5rg kza 7aga wra b3d l nfc el slave

master master bus: hw ely b generate clock 3la bus wy2dr yklm kol el slaves

f2wl ma master ymaster bus mmkn yb3t kza data wra b3d

**Communication protocol:**

* master

1. after the master master the bus
2. send start bit
3. address of slave

* slave

1. choose read or write
2. then slave send Ack or negative Ack(at not receiving)
3. then send or receive

* master

send stop bit and the end of communication not after every send and receive , stop bit mean master mb2ash mask el bus mosh m master el bus

* and address fixed you can see it inside datasheet
* highest priority master hwa ely 7 y master el bus

SPI vs. I2C

* Both SPI and I2C provide good support for communication with slow peripheral devices that are accessed intermittently. EEPROMs and real-time clocks are examples of such devices. But SPI is better suited than I2C for applications that are naturally thought of as data streams (as opposed to reading and writing addressed locations in a slave device). An example of a "stream" application is data communication between microprocessors or digital signal processors. Another is data transfer from analog-to-digital converters.
* SPI can also achieve significantly higher data rates than I2C. SPI-compatible interfaces often range into the tens of megahertz. SPI really gains efficiency in applications that take advantage of its duplex capability, such as the communication between a "codec" (coder-decoder) and a digital signal processor, which consists of simultaneously sending samples in and out.
* SPI devices communicate using a master-slave relationship. Due to its lack of built-in device addressing, SPI requires more effort and more hardware resources than I2C when more than one slave is involved. But SPI tends to be simpler and more efficient than I2C in point-to-point (single master, single slave) applications for the very same reason; the lack of device addressing means less overhead

|  |  |
| --- | --- |
| SPI |  |
| Control register1   * Serial Peripheral Interrupt Enable * Master/Slave Mode Select   Control register2   * Bidirectional mode   Status register   * Transfer Complete Flag   Baud rate register  3 bits , up to 7 frequencies |  |

pwm IC

A universal asynchronous receiver/transmitter (UART)

* (UART) controller is the key component of the serial communications subsystem of a computer.
* The UART takes bytes of data and transmits the individual bits in a sequential fashion.
* At the destination, a second UART re-assembles the bits into complete bytes. Serial transmission of digital information (bits) through a single wire or other medium is much more cost effective than parallel transmission through multiple wires.
* A UART is used to convert the transmitted information between its sequential and parallel form at each end of the link. Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms.
* 2400 baud

2 types of UART:

* UART Universal Asynchronous Receiver/Transmitter
* USART Universal Synchronous-Asynchronous Receiver/Transmitter

1.1 Synchronous Serial Transmission

Synchronous serial transmission requires that the sender and receiver share a clock with one another, or that the sender provide a strobe or other timing signal so that the receiver knows when to “read” the next bit of the data.

Synchronous communication is usually more efficient because only data bits are transmitted between sender and receiver, and synchronous communication can be more costly if extra wiring and circuits are required to share a clock signal between the sender and receiver.

1.2 Asynchronous Serial Transmission

Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which are used to synchronize the sending and receiving units.

When a word is given to the UART for Asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter.

After the Start Bit, the individual bits of the word of data are sent, with the Least Significant Bit (LSB) being sent first. Each bit in the transmission is transmitted for exactly the same amount of time as all of the other bits, and the receiver “looks” at the wire at approximately halfway through the period assigned to each bit to determine if the bit is a 1 or a 0. For example, if it takes two seconds to send each bit, the receiver will examine the signal to determine if it is a 1 or a 0 after one second has passed, then it will wait two seconds and then examine the value of the next bit, and so on.

When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates. The Parity Bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter.

When the receiver has received all of the bits in the data word, it may check for the Parity Bits (both sender and receiver must agree on whether a Parity Bit is to be used), and then the receiver looks for a Stop Bit. If the Stop Bit does not appear when it is supposed to, the UART considers the entire word to be garbled and will report a Framing Error to the host processor when the data word is read. The usual cause of a Framing Error is that the sender and receiver clocks were not running at the same speed, or that the signal was interrupted.

Regardless of whether the data was received correctly or not, the UART automatically discards the Start, Parity and Stop bits. If the sender and receiver are configured identically, these bits are not passed to the host.

If another word is ready for transmission, the Start Bit for the new word can be sent as soon as the Stop Bit for the previous word has been sent.

**Pulse width modulator(PWM)**

Features:

The PWM block includes these distinctive features:

* Eight independent PWM channels with programmable period and duty cycle
* Dedicated counter for each PWM channel
* Programmable PWM enable/disable for each channel
* Software selection of PWM duty pulse polarity for each channel
* Four clock sources (A, B, SA, and SB) provide for a wide range of frequencies
* Its output is waveform output of PWM

Modes of operation:

1. Run mode
2. Wait mode
3. Stop mode

Input capture(IC)

* Measure(read) frequency
* Input capture is a method of dealing with input signals in an embedded system.
* Embedded systems using input capture will record a timestamp in memory when an input signal is received. It will also set a flag indicating that an input has been captured. This allows the system to continue executing without interruption while an input is being received while still having the capability to trigger events based on the exact time when the input was received.

Device driver

Any driver should have some functions like:

1. Init: initialize registers, global variables, configuration parameters
2. Deinit: usually done when entering state mode and stop mode(reset mode)

Importance of deinit: derivers not used should be de-initialized to disable its effect on the whole system..

1. Set mode
2. Callback: application can’t talk to the HW directly but through the driver

There are 2 ways:

1. Application polling on the driver and ask about the change
2. Diver callback function on the application layer on any change

**Methods of configuration parameters:**

1. **Pre-compile: #** define
2. **Initialization:** configuration for parameters
3. **Function:** configuration done within the function input parameters

Link time: cons values in .obj files then linked with .c filea

Post build: pass to function pointer to structure

Embedded software components can be configured at different points in time:

1. pre-compile configuration is applied before the source code is put into the build process. This allows the implementation of some configuration settings (e.g. handling of variants) by macros or C pre-processor switches.
2. Link-time configuration is typically used to link tables with ROM constants to a library.
3. post-build configuration is applied on an existing ECU. Here the configuration data is downloaded via a flash boot loader.