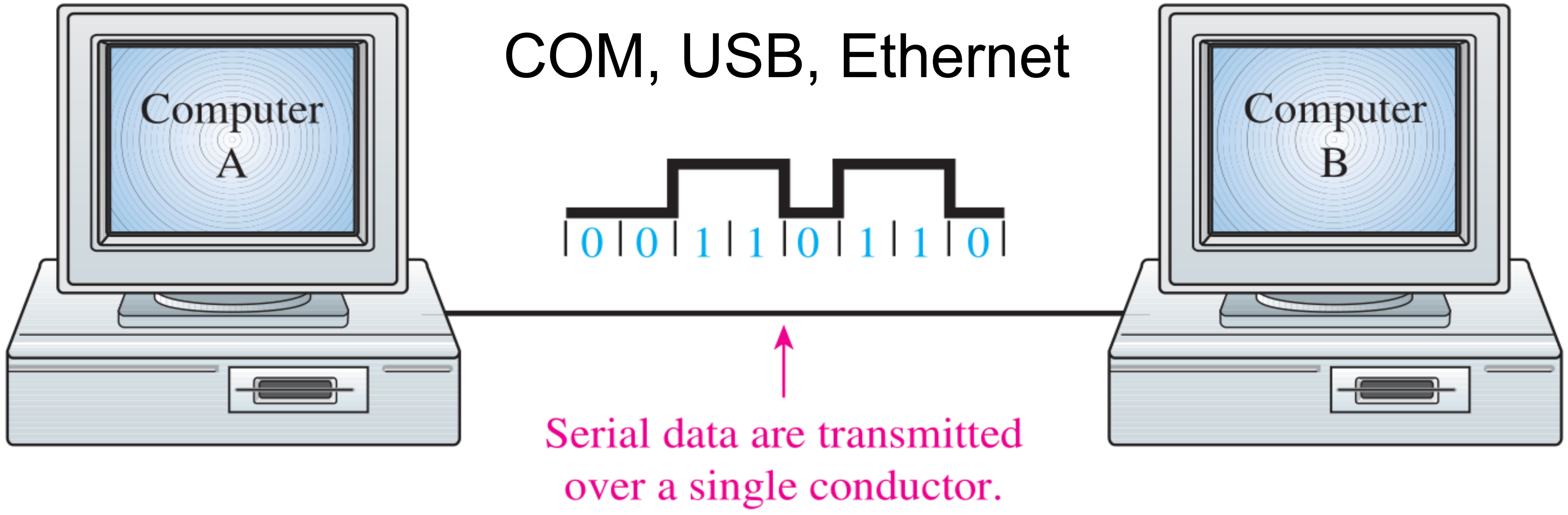
Digital Logic Design

Lecture 11

Interfacing to digital circuits

Binary information to be transmitted from one location to another will be in either serial or parallel format

COM, USB, Ethernet

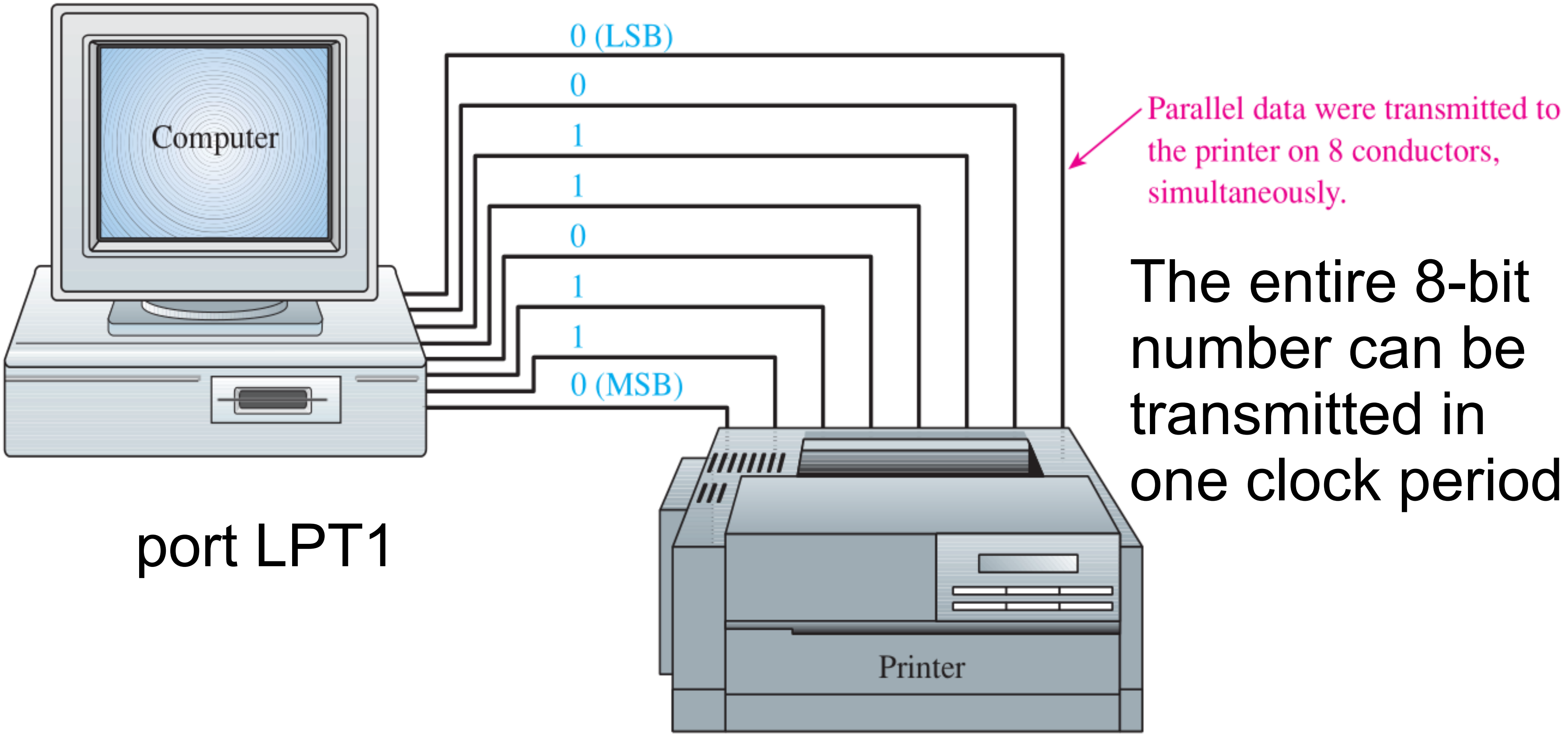


The serial format is inexpensive because it only uses a single conductor (and a common ground) and one set of input/output circuitry.

But it is slow because it can only transmit 1 bit for each clock period.

The parallel format uses a separate electrical conductor for each bit to be transmitted (and a common ground).

Inside a computer, binary data are almost always transmitted on parallel channels called the PCI data bus.

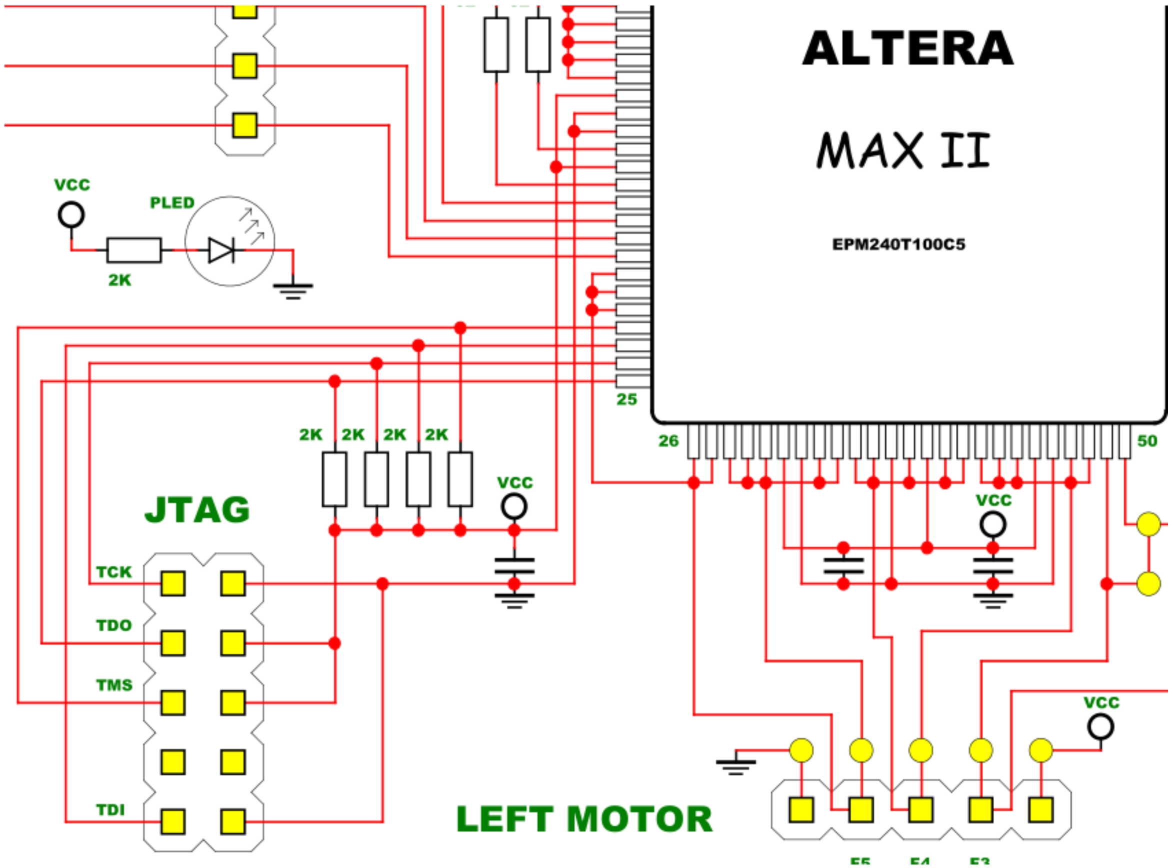

Computer
0 (LSB)
0
1
1
0
1
1
0 (MSB)
Parallel data were transmitted to the printer on 8 conductors, simultaneously.
Printer

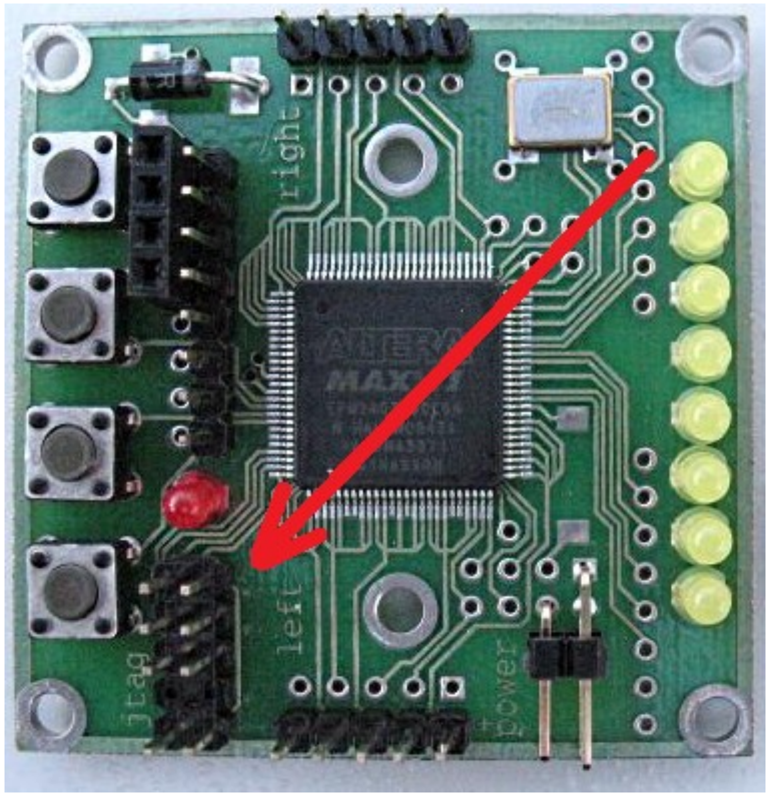
The entire 8-bit number can be transmitted in one clock period

port LPT1

JTAG

Joint Test Action Group





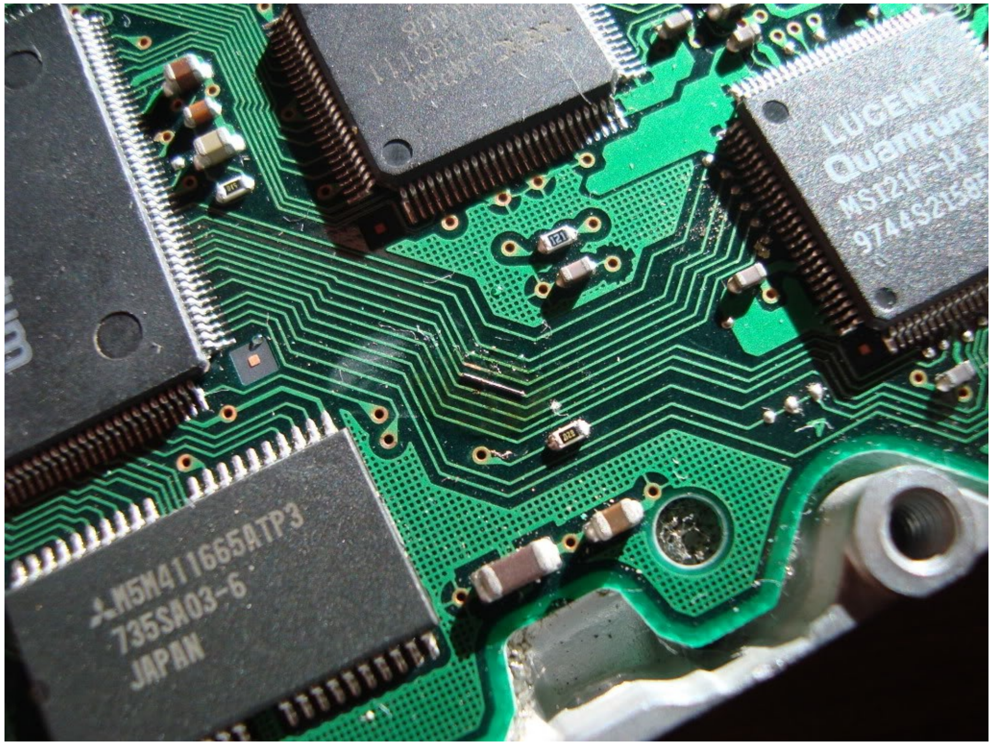
JTAG is an IEEE standard developed in the 1980s to solve electronic boards manufacturing issues.

Nowadays it is used as programming, debug and probing port.

ICs (**integrated circuits**) can have lots of pins

ICs are connected together with lots of connections (**PCB traces**)

PCB – **printed circuit board**

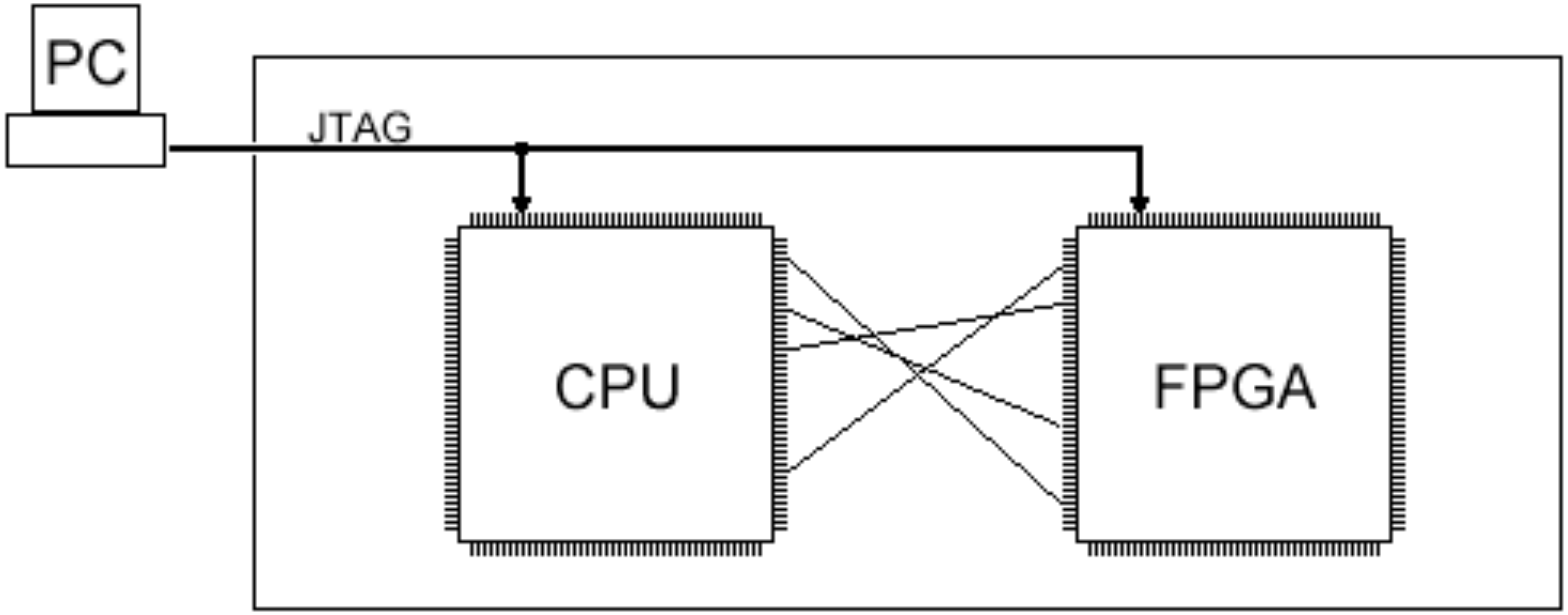


If you build a thousand boards, each with a few a thousand connections, you inevitably have a few bad boards.

JTAG was created to test all these boards and connections between ICs.

Example: a simple PCB with two ICs, a CPU and an FPGA.

Four connectons are shown.



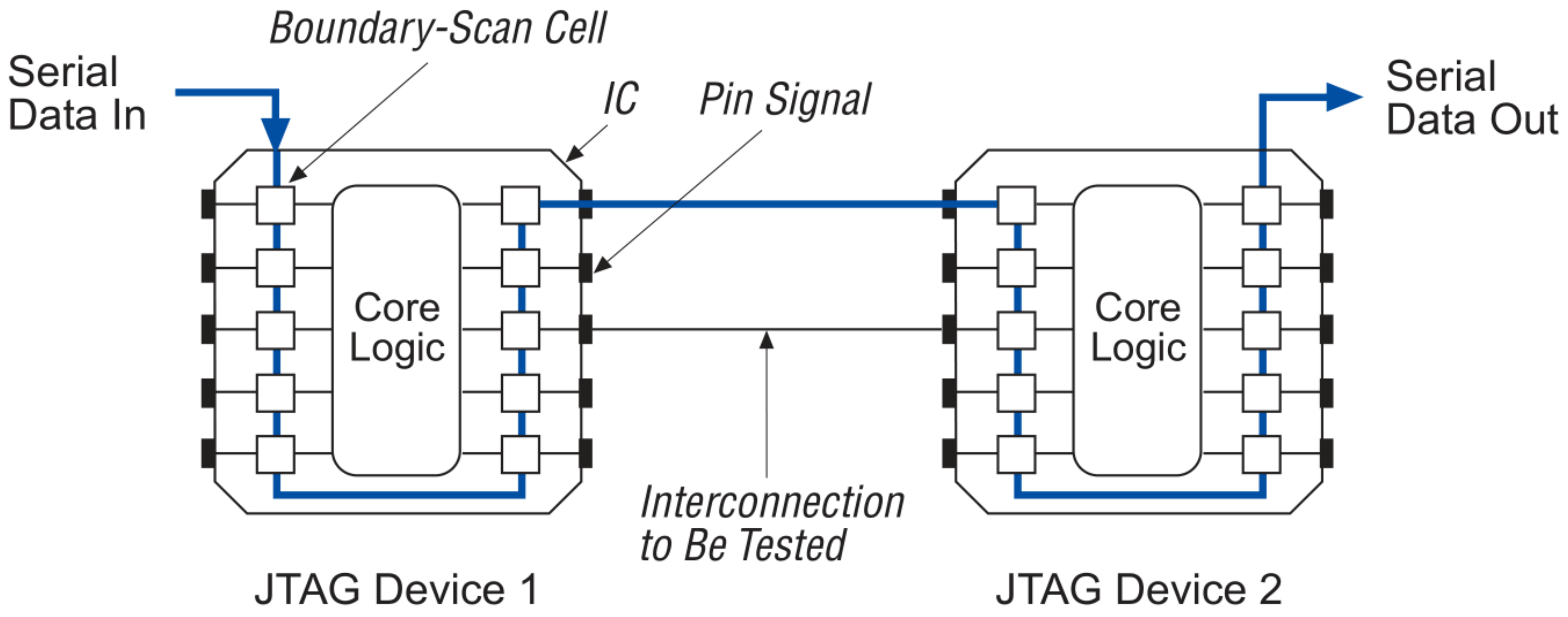
JTAG can take control (or hijack) the pins of all the ICs.

Example: JTAG is going to make all the CPU pins outputs, and all the FPGA pins inputs.

?

By sending some data from the CPU pins, and reading the values from the FPGA pins, JTAG can make sure that the board connections are fine.

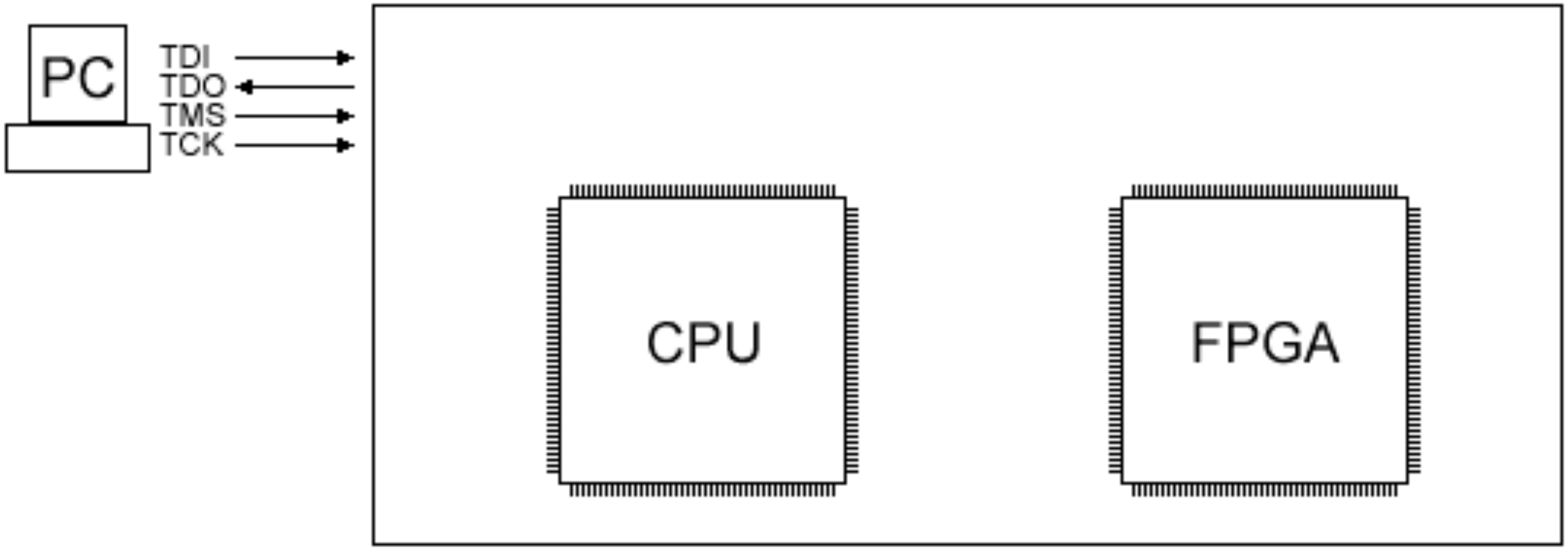
Boundary-Scan Testing



This BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally.

Boundary-scan cells (BSCs) in a device can force signals onto pins, or capture data from pin or core logic signals.

JTAG consists of 4 logic signals: TDI, TDO, TMS, TCK



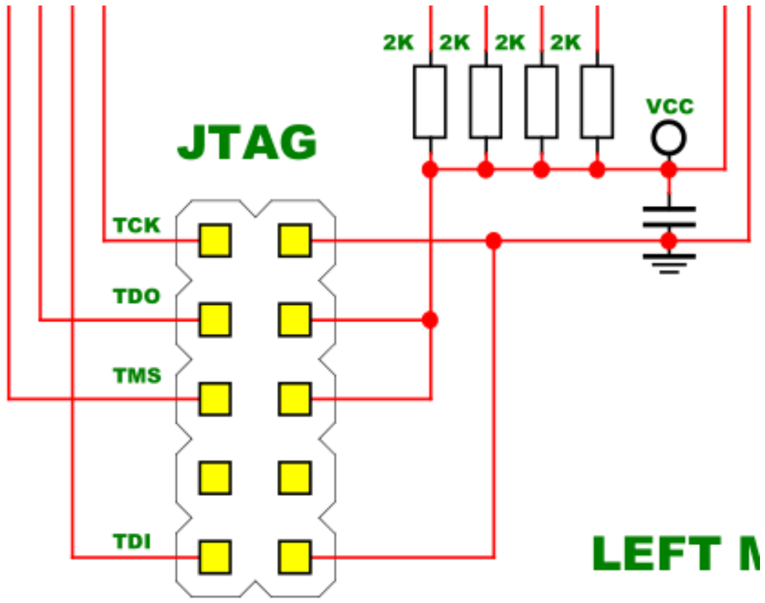
TCK – Test clock input

The other JTAG signals are synchronous to TCK:

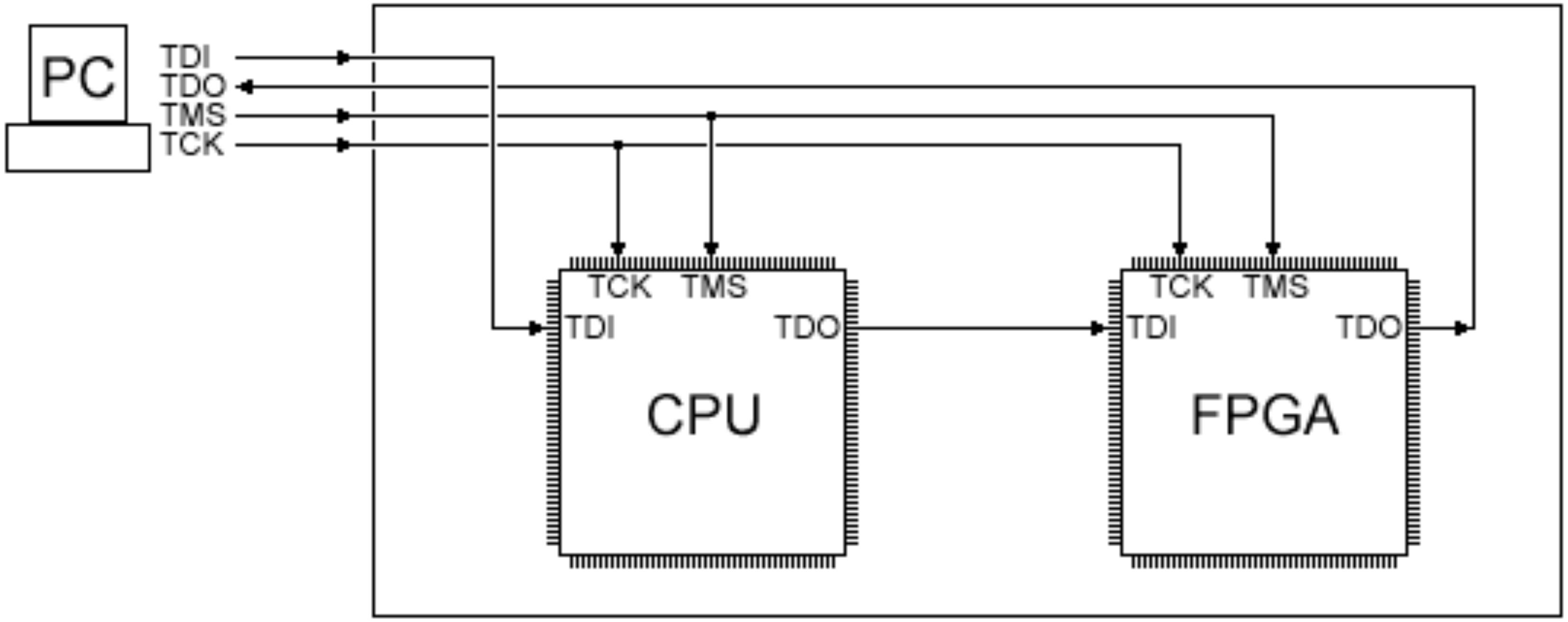
TDI – Test data input

TDO – Test data output

TMS – Test mode select



TMS and TCK are wired in parallel to all JTAG ICs.



TDI and TDO and connected to form a chain.

Each JTAG compliant IC has four pins used for JTAG (three inputs, and one output).

A fifth pin named TRST is optional (JTAG reset).

The JTAG pins are usually dedicated (not shared for other purposes).

JTAG is used for more purposes than just testing.

CPU and FPGA manufacturers allow JTAG to be used as debug port.

FPGA manufacturers also allow configuring the FPGA through JTAG, and use the JTAG signals inside the FPGA core.

Use a **JTAG cable** to control a JTAG bus from a PC.

The JTAG cable might connect to a PC's

- Parallel (printer) port
- USB port
- Ethernet port

Inside each JTAG IC, there is a **JTAG TAP controller**

TAP – Test Access Port

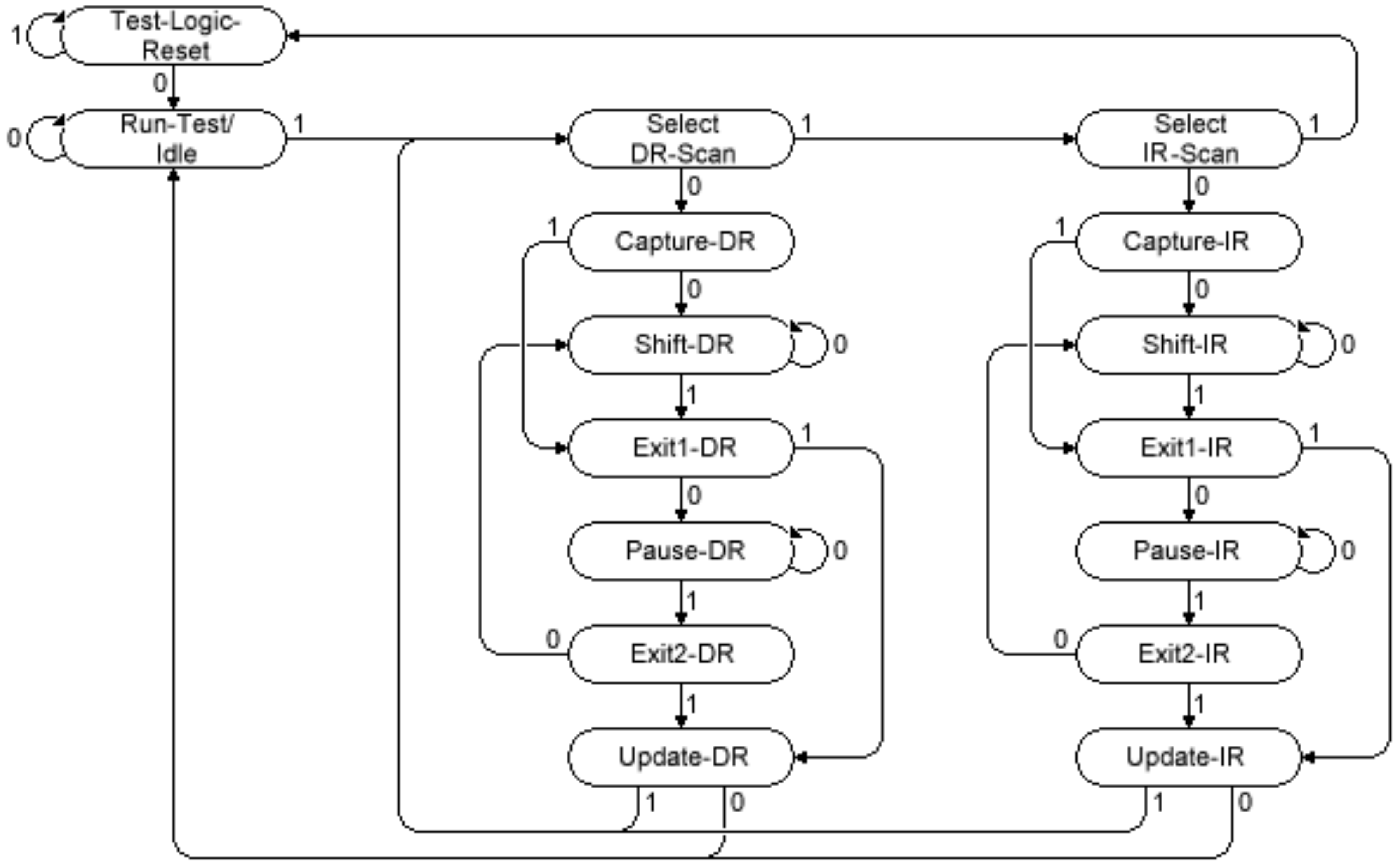
The TAP controller is a finite state machine with 16 states.

TMS is the signal that controls the TAP controller.

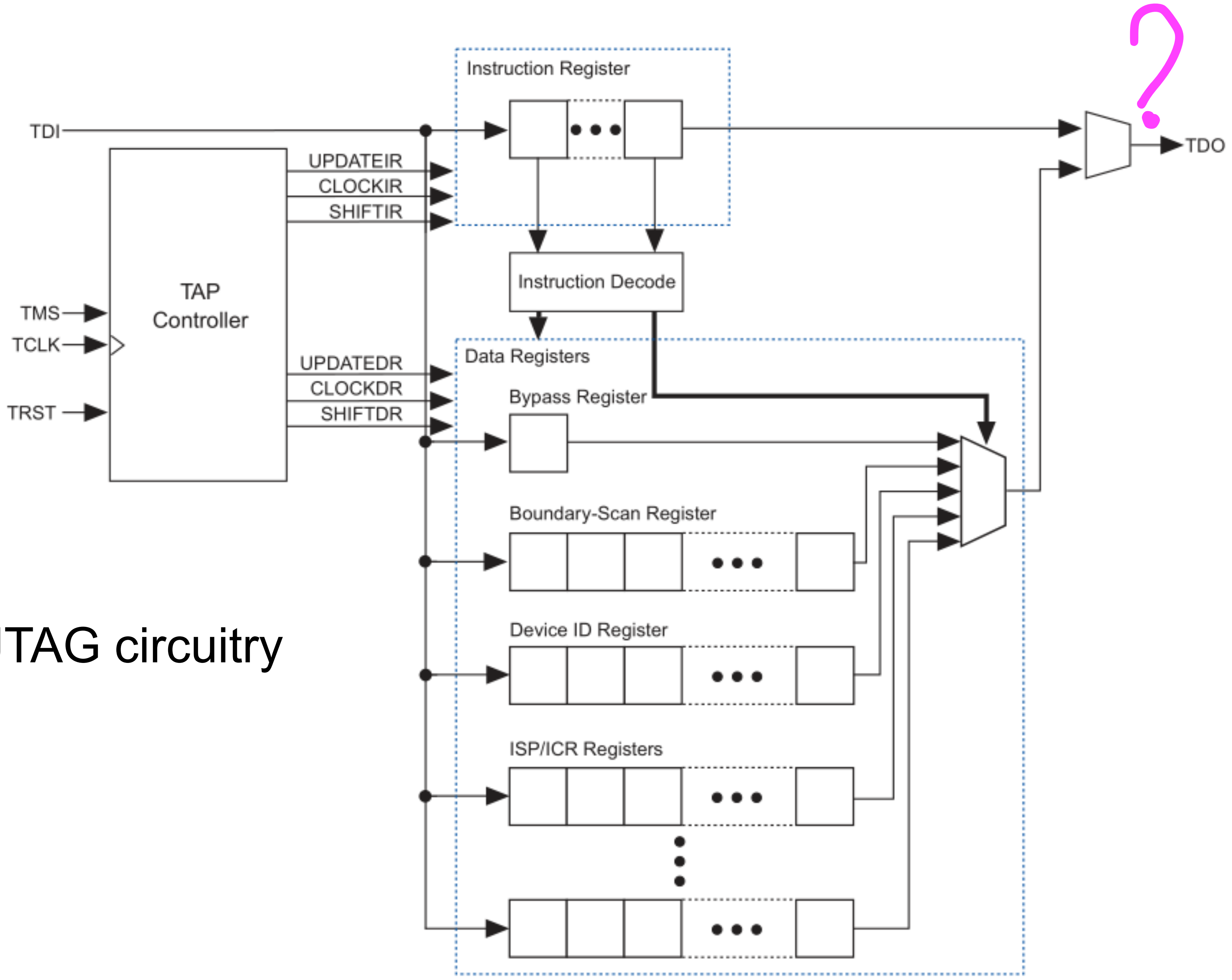
Since TMS is connected to all the JTAG ICs in parallel, all the TAP controllers move together (to the same state).

The TAP controller can execute commands specific to a particular IC.

JTAG TAP controller FSM



The numbers (0 or 1) are the value of TMS to change state



JTAG circuitry

For JTAG to work properly, the tap controllers of a JTAG chain must always be in the same state.

After power-up, they may not be in sync

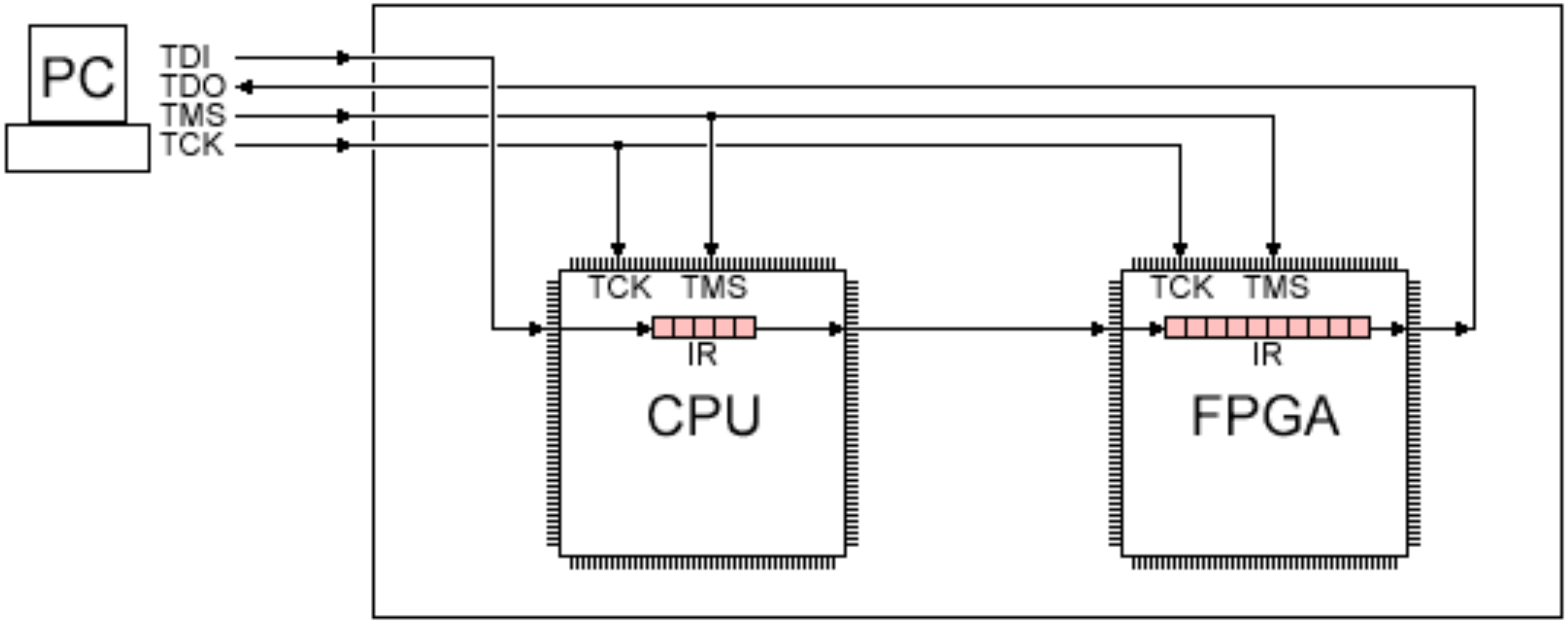
Trick: if TMS stays at 1 for five clocks, a TAP controller goes back to the state "Test-Logic-Reset".

That is used to synchronize the TAP controllers.

You write a value into IR, the instruction register, that corresponds to what you want to do with JTAG.

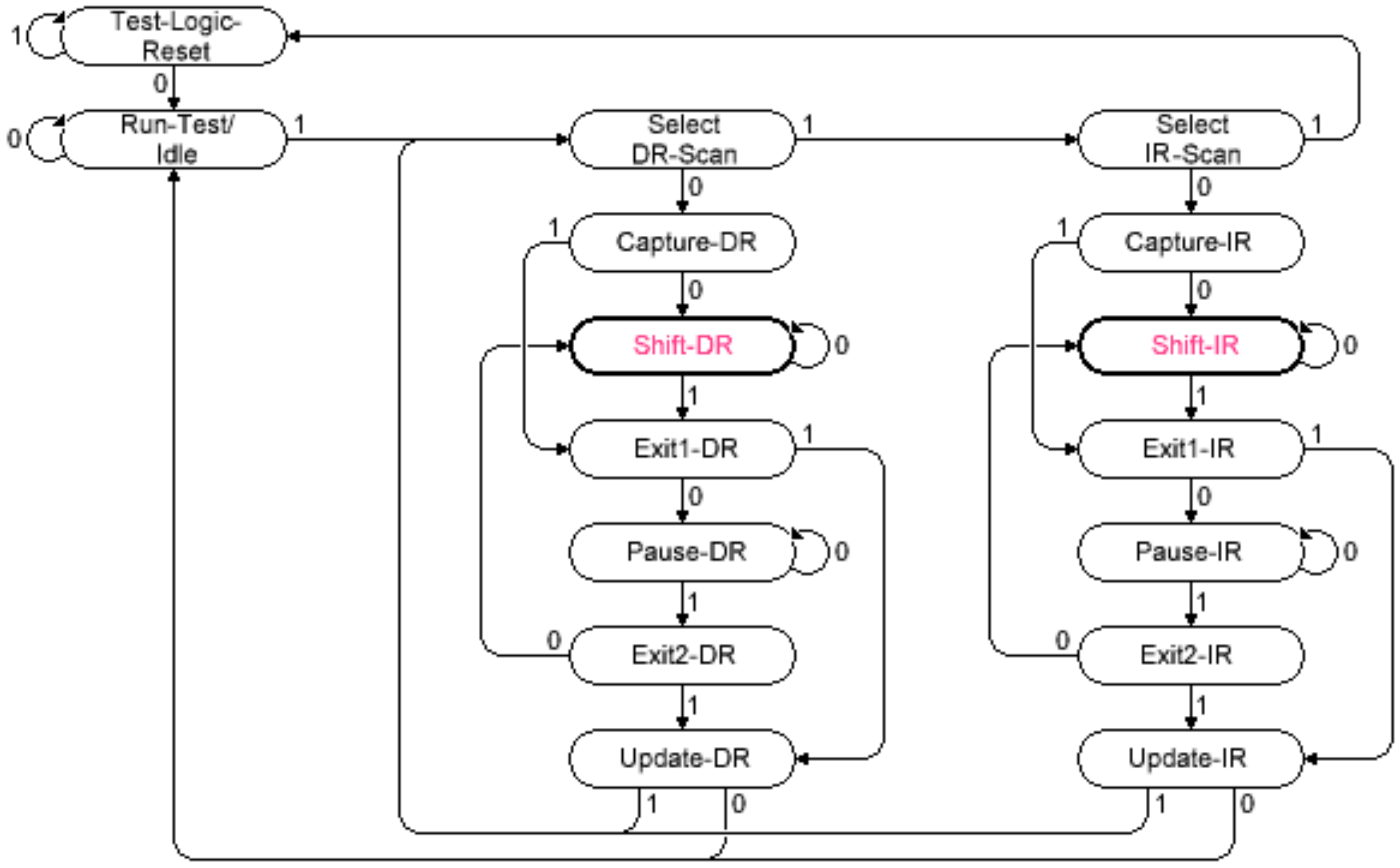
Each IC has a list of possible instructions.

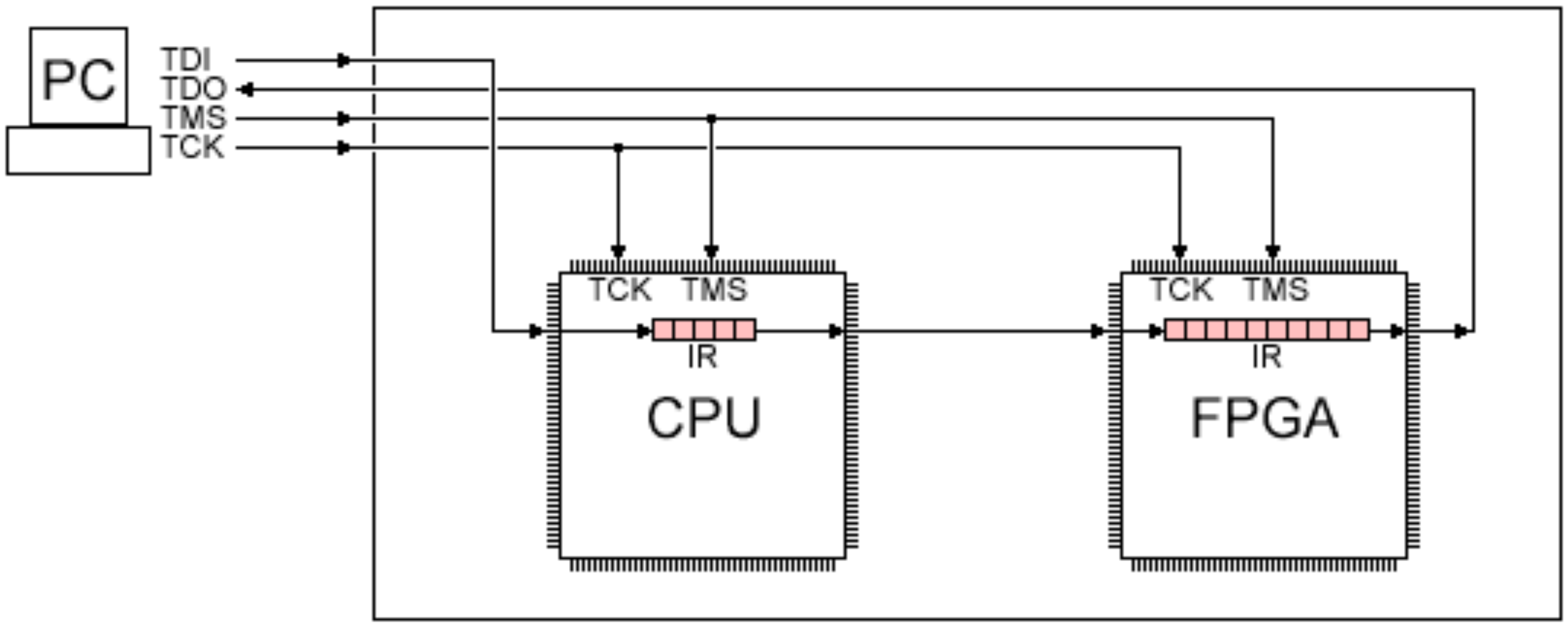
Each IC IR register has a specific length.



The IR registers form a chain that is loaded through the TDI and TDO pins.

To load IR values, the PC has to make sure the TAP controllers are in the Shift-IR state.





In our example, the CPU's IR is 5 bits long.

That means it can support up to 32 JTAG instructions.

To load IR values, the PC has to make sure the TAP controllers are in the Shift-IR state, and send 15 bits through TDI.

Once shifted, the first 10 bits actually end up into the FPGA IR, and the last 5 bits into the CPU IR.

DR registers

Each TAP controller has only one IR register, but has multiple DR registers.

The DR registers are shifted the same way the IR registers are but using the Shift-DR state instead of the Shift-IR.

Each IR value selects a different DR register.

The bypass register is a 1-bit-long data register used to provide a minimum-length serial path between TDI and TDO.

The boundary-scan register is a shift register composed of all the BSCs of the device.

In our example, the CPU could have up to 32 DR registers (if all IR instructions were implemented).

Serial Vector Format (SVF) file

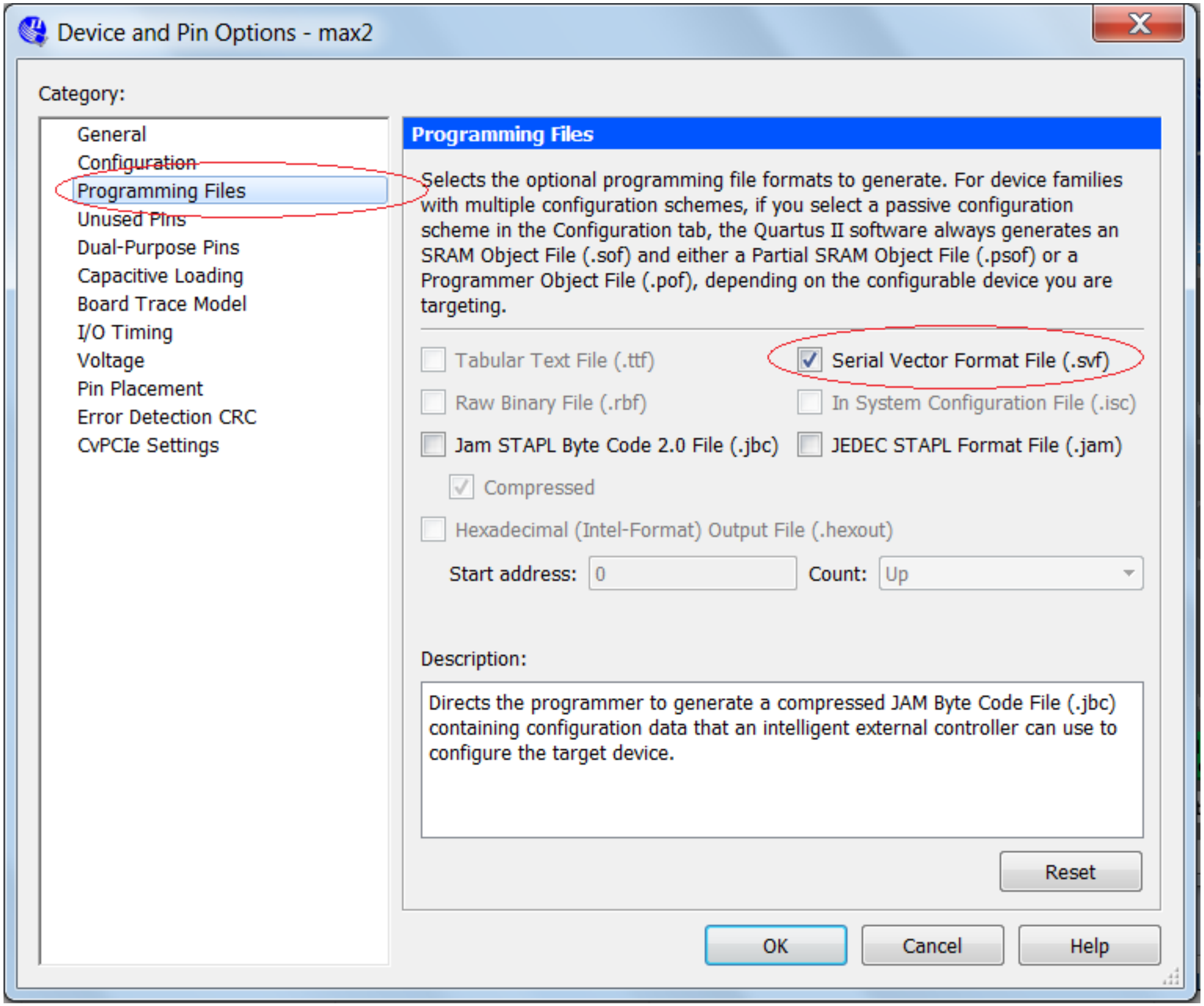
JTAG commands for programming Altera FPGA are written in the Serial Vector Format Specification.

SVF is the media for exchanging descriptions of high-level JTAG bus operations.

The SVF file is defined as an ASCII file that consists of a set of SVF statements.

An example of an SVF file:

```
!Begin Test Program
TRST OFF;                                          !Disable Test Reset line
ENDIR IDLE;                                        !End IR scans in IDLE
ENDDR IDLE;                                        !End DR scans in IDLE
HIR  8  TDI (00);                                  !8-bit IR header
HDR 16  TDI (FFFF) TDO (FFFF) MASK (FFFF);!16-bit DR header
TIR 16  TDI (0000);                                !16-bit IR trailer
TDR  8  TDI (12);                                  !16-bit DR trailer
SIR  8  TDI (41);                                  !8-bit IR scan
SDR 32  TDI (ABCD1234) TDO (11112222);             !32-bit DR scan
STATE   DRPAUSE;                                   !Go to stable state DRPAUSE
RUNTEST 100 TCK ENDSTATE IRPAUSE;                  !RUNBIST for 100 TCKs
!End Test Program
```


Device and Pin Options - max2

Category:

- General
- Configuration
- Programming Files
- Unused Pins
- Dual-Purpose Pins
- Capacitive Loading
- Board Trace Model
- I/O Timing
- Voltage
- Pin Placement
- Error Detection CRC
- CvPCIe Settings

Programming Files

Selects the optional programming file formats to generate. For device families with multiple configuration schemes, if you select a passive configuration scheme in the Configuration tab, the Quartus II software always generates an SRAM Object File (.sof) and either a Partial SRAM Object File (.psof) or a Programmer Object File (.pof), depending on the configurable device you are targeting.

- [ ] Tabular Text File (.ttf)
- [✓] Serial Vector Format File (.svf)
- [ ] Raw Binary File (.rbf)
- [ ] In System Configuration File (.isc)
- [ ] Jam STAPL Byte Code 2.0 File (.jbc)
- [ ] JEDEC STAPL Format File (.jam)
- [✓] Compressed
- [ ] Hexadecimal (Intel-Format) Output File (.hexout)

Start address: 0 Count: Up

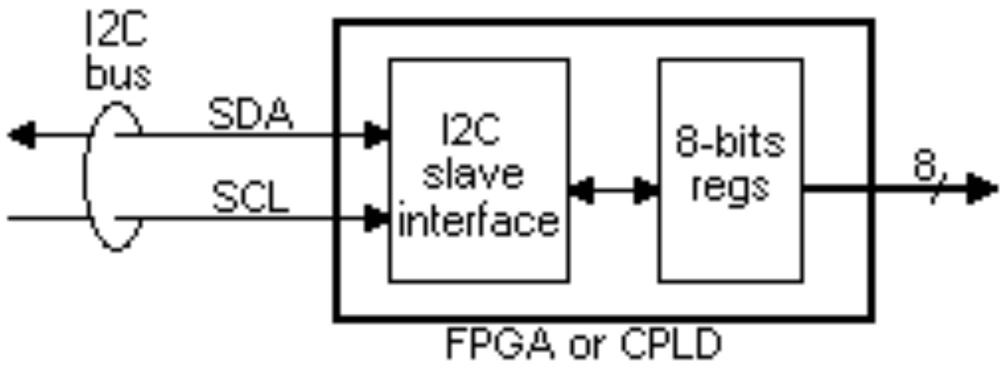
Description:

Directs the programmer to generate a compressed JAM Byte Code File (.jbc) containing configuration data that an intelligent external controller can use to configure the target device.

I^2C = Inter-IC

pronounced I-squared-C

The I^2C bus is a simple way to connect multiple chips together, in particular FPGAs/CPLDs.



Uses 2 wires (SDA and SCL) in addition to V_{DD} and GND

SDA is the Serial DAta line

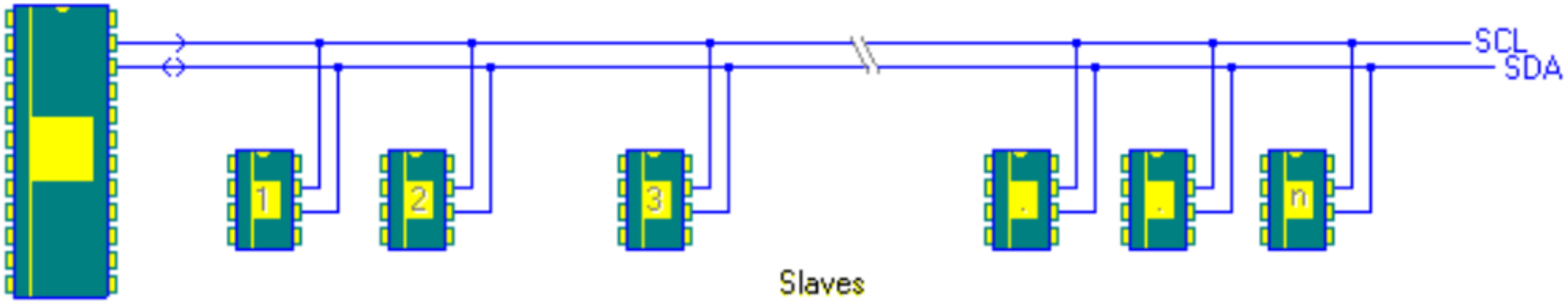
SCL is the Serial CLock line

SDA and SCL are both bi-directional

An I2C bus needs an I2C master and an I2C slave.

The I2C master is a transaction initiator (a master can write-to or read-from a slave).

The I2C slave is a transaction recipient (a slave can be written-to or read-from a master).



Can support over 100 devices on the same bus

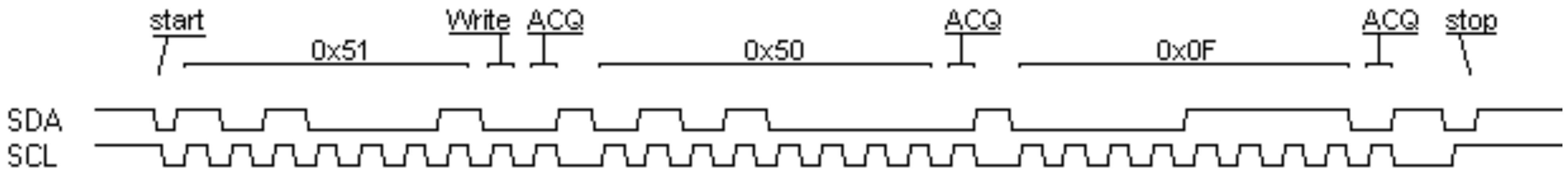
Each device on the bus has an address to be individually accessible

Multi-master (for example, two CPUs can easily share the same I2C devices)

7-bits slave addresses: each device connected to the bus has got such a unique address

Data divided into 8-bit bytes

Example: a write to an EEPROM at address 0x51, with 2 data bytes 0x50 and 0x0F



An I^{2}C transaction begins with a "start" condition, followed by the address of the device we wish to speak to, a bit to indicate if we want to read or write, the data written or read, and finally a "stop".

There are other details, like the need to have an "acknowledge" bit after each byte transmitted

All I^{2}C-bus compatible devices incorporate an on-chip interface which allows them to communicate directly with each other via the I^{2}C-bus.

This design concept solves the many interfacing problems encountered when designing digital control circuits.

Examples:

- microcontrollers
- general-purpose circuits

  e.g. LCD drivers, remote I/O ports, memories

- application-oriented circuits

  e.g. digital tuning and signal processing circuits for radio and video systems

I^{2}C is slow

standard mode – 100 kbps

fast mode – 400 kbps

high speed mode – 3.4 Mbps

SPI

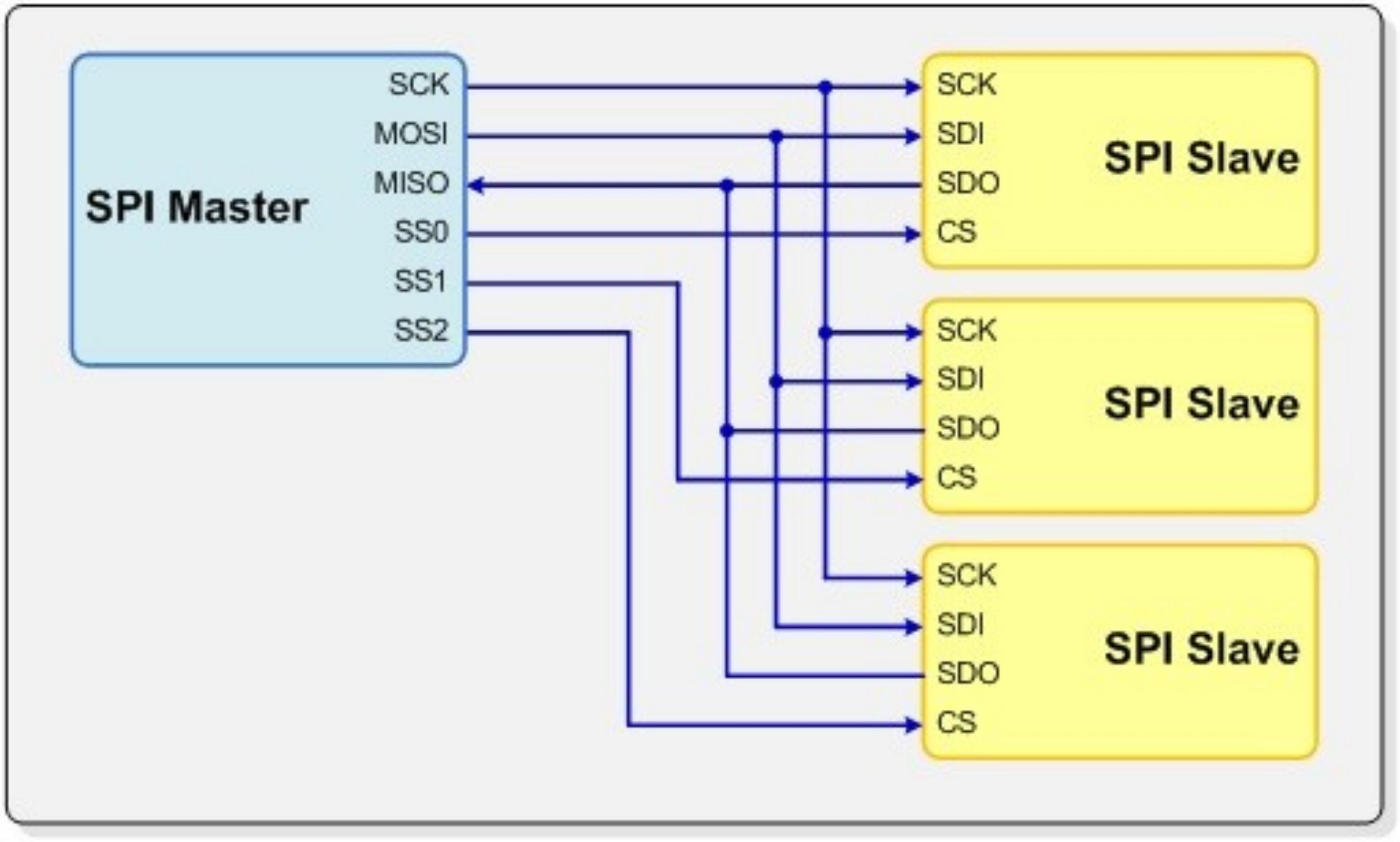
The Serial Peripheral Interface

Synchronous, serial, full-duplex, not plug-and-play.

There is one (and only one) master, and one or more slaves.

The SPI bus is commonly used for communication with

- flash memory
- sensors
- real-time clocks (RTCs)
- analog-to-digital converters
- and more



SCK – serial clock

MOSI – master out slave in

MISO – master in slave out

SS – slave select

The SCK, MOSI, and MISO signals can be shared by slaves while each slave has a unique SS line.

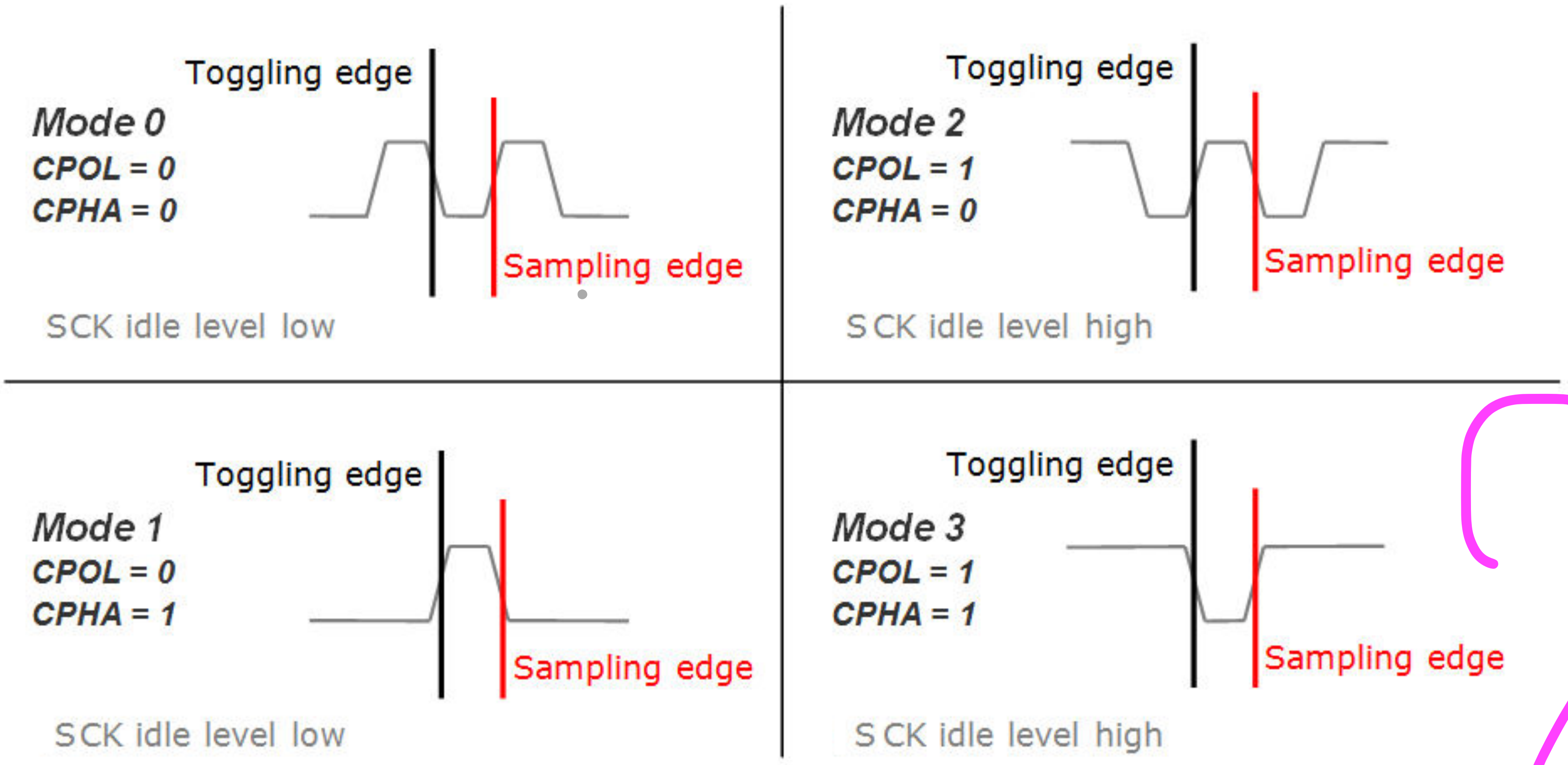
A clock signal is sent from the bus master to all slaves

All the SPI signals are synchronous to this clock signal

When the SPI master wishes to send data to a slave and/or request information from it, it selects slave by pulling the corresponding SS line low and it activates the clock signal at a clock frequency usable by the master and the slave.

The master generates information onto MOSI line while it samples the MISO line.

Four communication modes are available (MODE 0, 1, 2, 3) – that 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).



?

CPOL – clock polarity

CPHA – clock phase

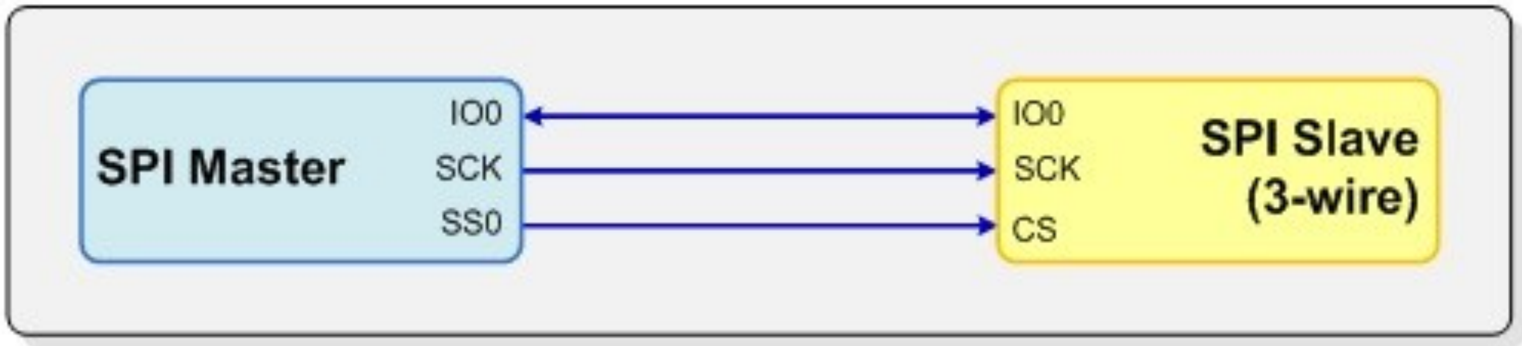
A master/slave pair must use the same set of parameters – SCK frequency, CPOL, and CPHA for a communication to be possible.

SPI does not define any maximum data rate, not any particular addressing scheme; it does not have a acknowledgement mechanism to confirm receipt of data and does not offer any flow control.

The SPI master has no knowledge of whether a slave exists, unless ‘something’ additional is done outside the SPI protocol.

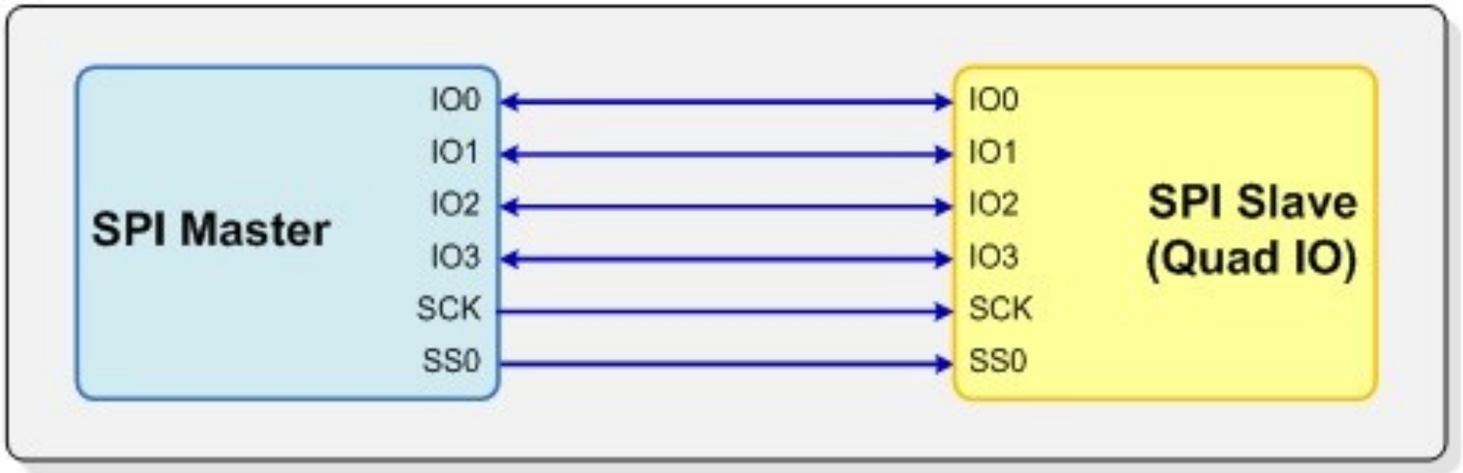
SPI does not care about the physical interface characteristics like the I/O voltages and standard used between the devices.

In 3-wire mode, MOSI and MISO lines are combined



Reducing the number of data lines and operating in half-duplex mode also decreases maximum possible throughput

Multi I/O variants such as dual I/O and quad I/O add additional data lines to the standard for increased throughput.



SPI versus I^{2}C

I²C needs 2 lines, while SPI needs at least 4 signals and more, if you add slaves.

SPI requires additional work, logic and/or pins if a multi-master architecture has to be built on SPI.

I²C has a limited device address space on 7 bits, but it can be overcome with the 10-bits extension.

I²C is a winner over SPI in sparing pins, board routing and how easy it is to build an I²C network.

SPI is faster then I²C

SPI is full-duplex; I²C is not

SPI does not define any speed limit; implementations often go over 10 Mbps.

I²C and SPI are often considered as ‘little’ communication protocols compared to Ethernet, USB, SATA, PCI-Express and others, that present throughput in the x100 megabit per second range or gigabit per second.

Ethernet, USB, SATA are meant for ‘outside the box communications’ and data exchanges between whole systems.

When there is a need to implement a communication between integrated circuit such as a microcontroller and a set of relatively slow peripheral, there is no point at using any excessively complex protocols.

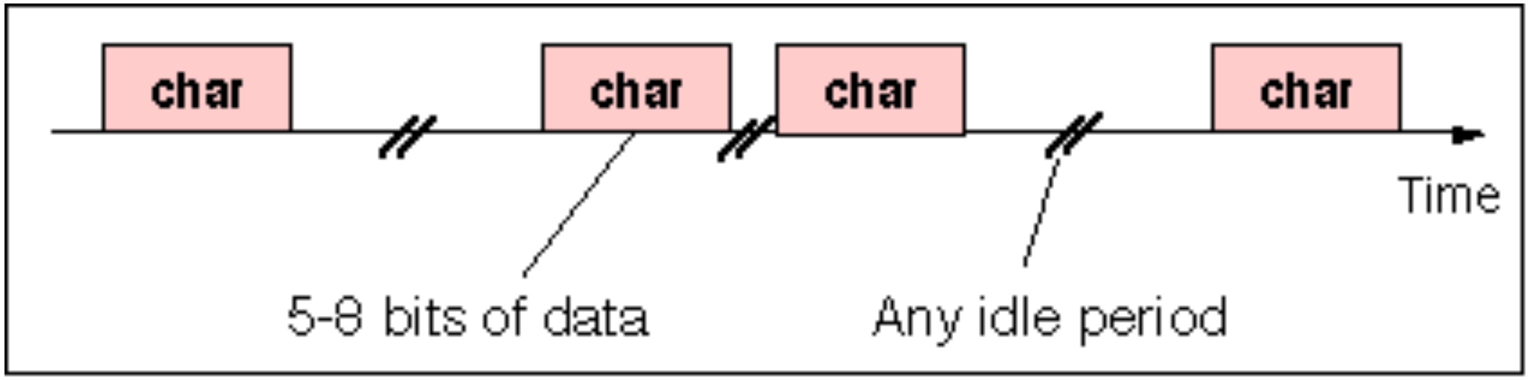
There, I²C and SPI have become popular.

Asynchronous communication

The transmitter and receiver clock are independent and are not synchronised.

There need be no timing relationship between successive characters (or bytes of data).

Individual characters may be separated by any arbitrary idle period.



An asynchronous link communicates data as a series of characters of fixed size and format.

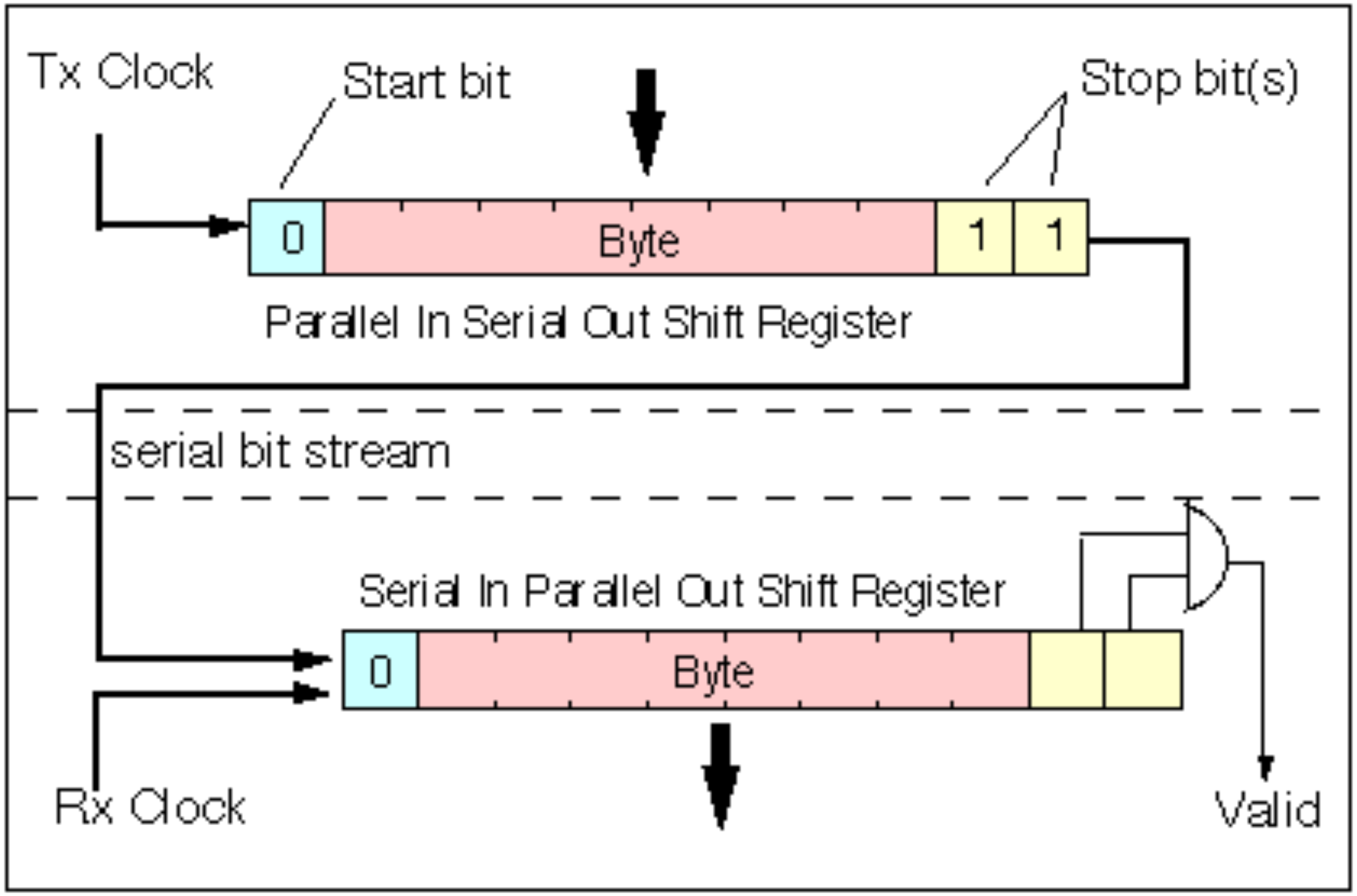
Each character is preceded by a start bit and followed by 1-2 stop bits.

Parity is often added to provide some limited protection against errors occurring on the link.

?

The use of independent transmit and receive clocks constrains transmission to relatively short characters (<8 bits)

The asynchronous transmitter delimits each character by a start sequence and a stop sequence.


Tx Clock
Start bit
Stop bit(s)
0
Byte
1
1
Parallel In Serial Out Shift Register
serial bit stream
Serial In Parallel Out Shift Register
0
Byte
Rx Clock
Valid

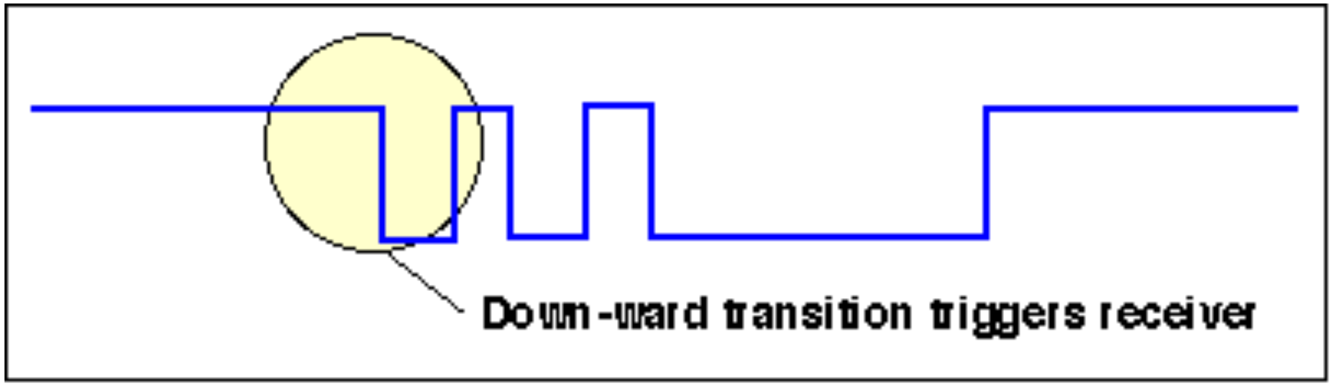
The start bit (0), data (usually 8 bits plus parity) and stop bit(s) (1) are transmitted using a shift register clocked at the nominal data rate.

At the receiver, a clock of the same nominal frequency is constructed and used to clock-in the data to the receive shift register.

Only data that are bounded by the correct start and stop bits are accepted.

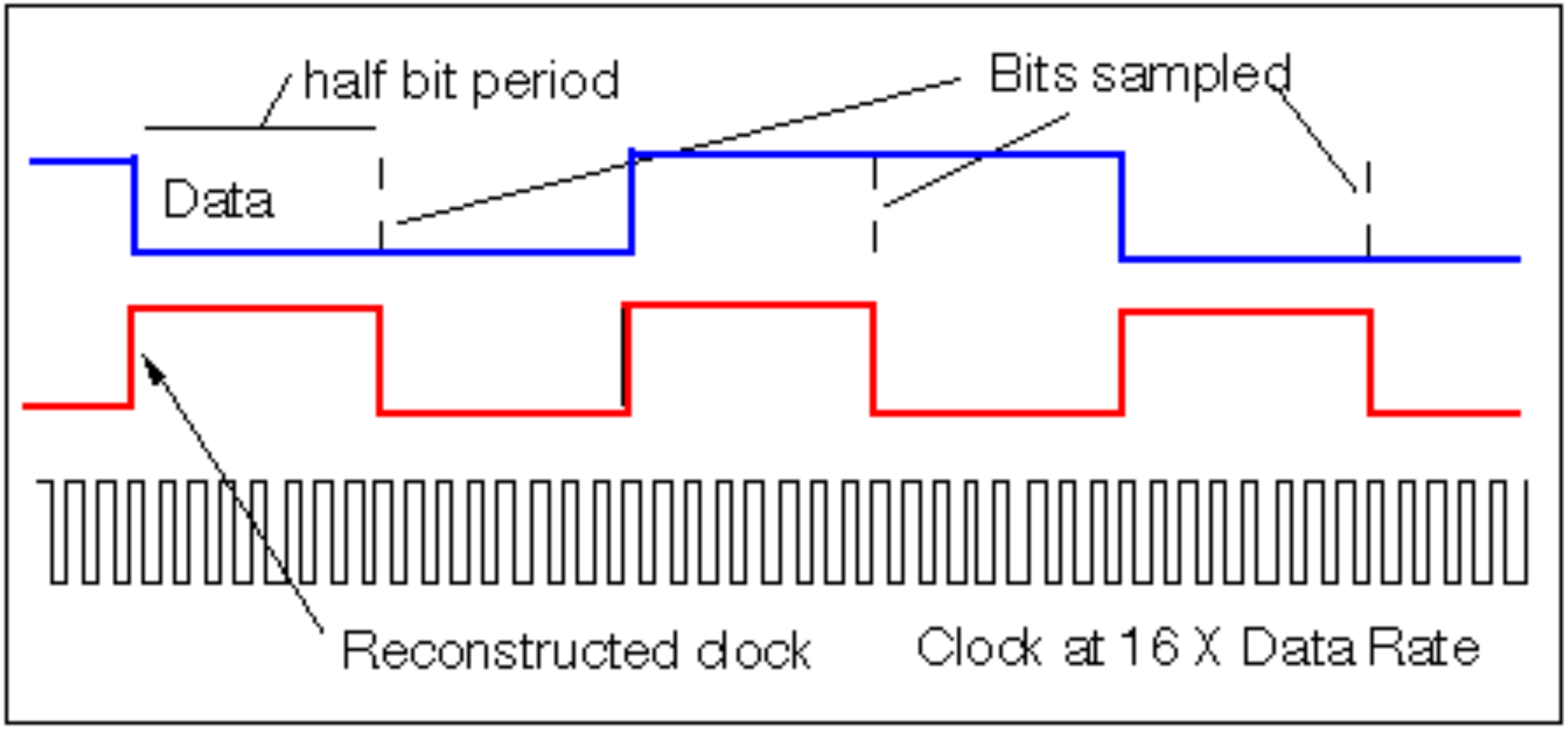
This operation is normally performed using a **UART - Universal Asynchronous Receiver Transmitter**.

The receiver is started by detecting the edge of the first start bit

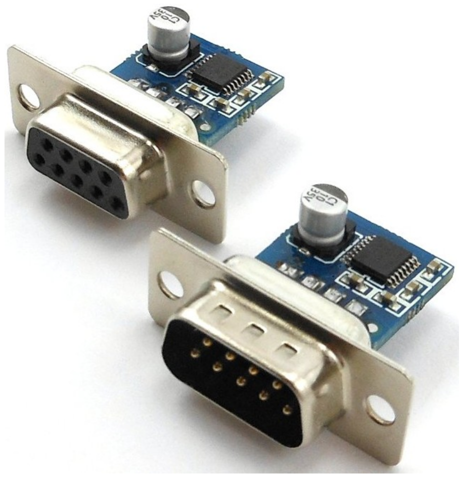


The reconstructed receive clock (receive (rx) clock) is generated using a local stable high rate clock, frequently operating at 16 or 32 times the intended data rate.

Clock generation proceeds by detecting the edge of the start bit and counting sufficient clock cycle from the high frequency clock to identify the mid position of the start bit.



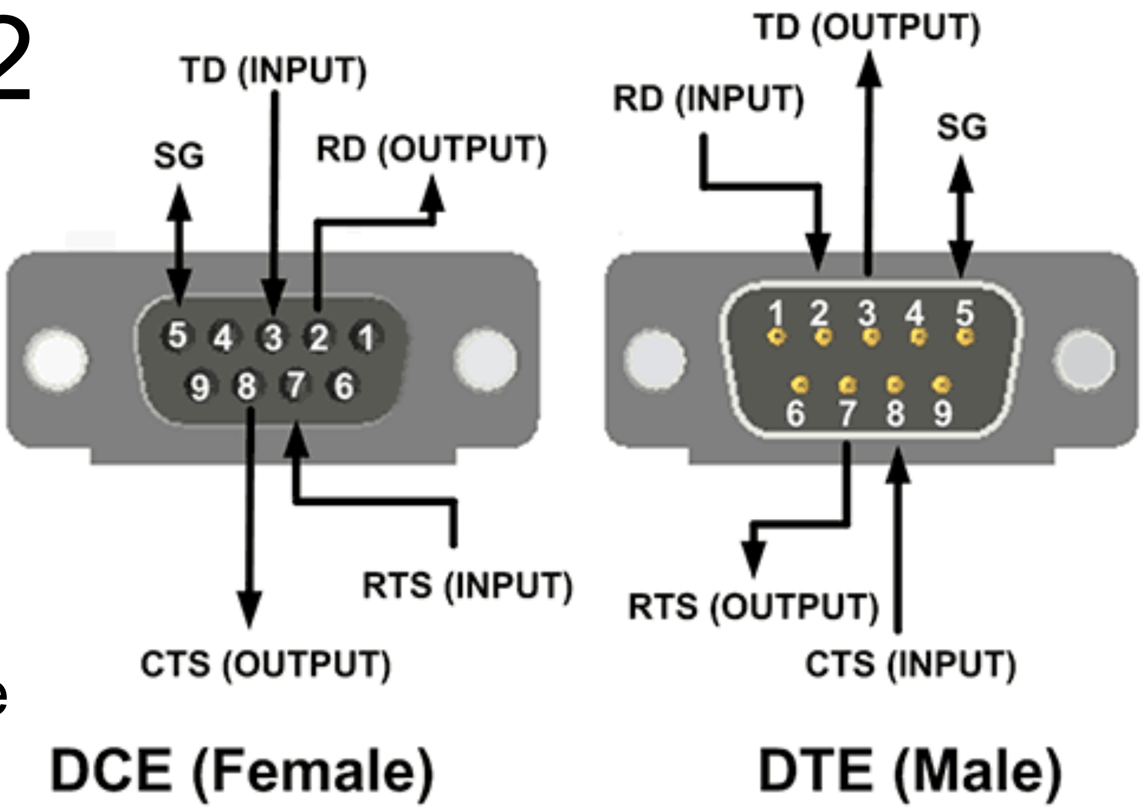
RS-232



The **DTE** (**Date Terminal Equipment**) is the computer or terminal that serves as the data source or the data sink.

It also provides the control functions.

The **DCE** (**Data Communication Equipment**) is a peripheral device.



RS-232 Function	Pin Number		Input to	Input to
	DB25	DB9	DTE	DCE
Shield	1			
Transmit Date (TD)	2	3		■
Receive Data (RD)	3	2	■	
Request to Send (RTS)	4	7		■
Clear to Send (CTS)	5	8	■	
DCE Ready (DSR)	6	6	■	
Signal Ground (SG)	7	5		
Received Line Signal Detector (DCD)	8	1	■	
DTE Ready (DTR)	20	4		■
Ring Indicator (RI)	22	9		

RS-232 allows bidirectional full-duplex communication (the PC can send and receive data at the same time).

Data is commonly sent by chunks of 8 bits (a byte) and is serialized: the LSB (data bit 0) is sent first, then bit 1, ... and the MSB (bit 7) last.

Transmission sequence:

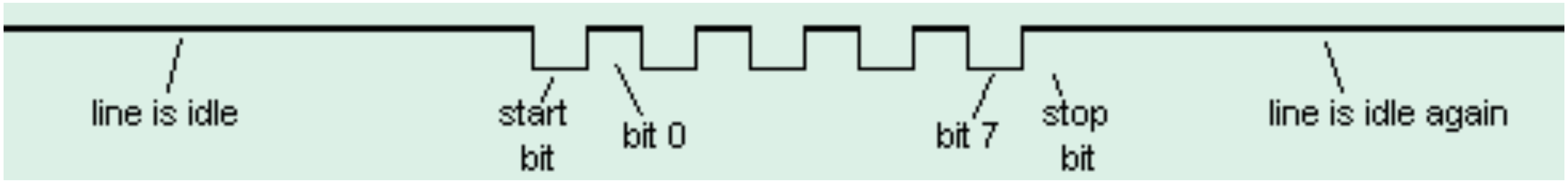
The transmitter sends "idle" (1) when and as long as the line is idle.

The transmitter sends "start" (0) before each byte transmitted, so that the receiver can figure out that a byte is coming.

The 8 bits of the byte data are sent.

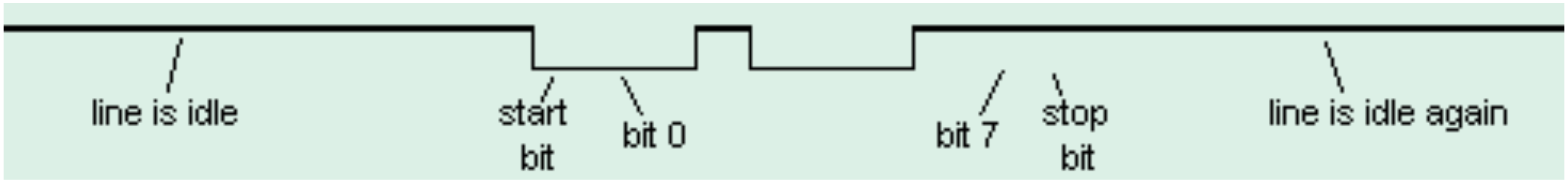
The transmitter sends "stop" (1) after each byte.

85 (01010101) is sent



Since it is transmitted LSB (bit-0) first, the line toggles like that: 1-0-1-0-1-0-1-0.

196 (11000100) is sent:



The bits are harder to see.

The receiver has to know at which speed the data is sent.

Common implementations of the RS-232 interface allow some standard speed values:

1200 bauds (bits per second)

9600 bauds

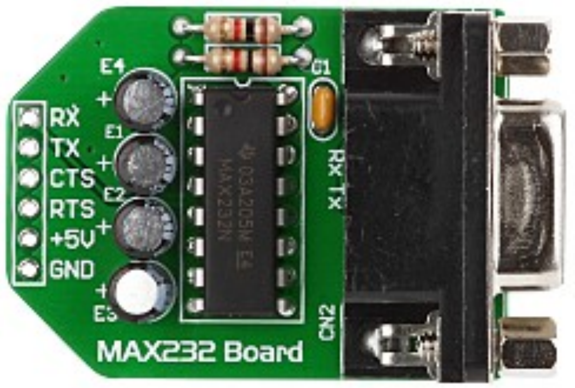
38400 bauds

115200 bauds

A voltage of +12 volts (usually +3 to +10 volts) represents a binary 0 and -12 volts (-3 to -10 volts) is a binary 1.

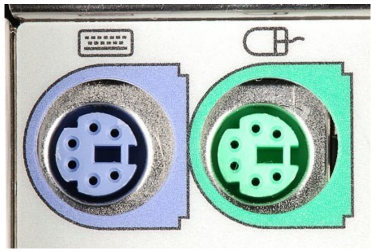
You need a translating circuit to communicate with a 5V or 3.3V logic.

A translating circuit can be based on ICs like HIN202, MAX232, SP230, and other.



Other protocols

PS/2 http://www.computer-engineering.org/



USB

EPP (Enhanced Parallel Port)

SD

HDMI

You can use them with FPGA: http://fpga4fun.com/

Other protocols

PCI

PCI Express

Ethernet

You can use them with FPGA: http://fpga4fun.com/