Here's a tip - aluminum foil makes a lovely hat,
and it blocks out the government's

Communications and Memories

Keep you guys outta trouble.

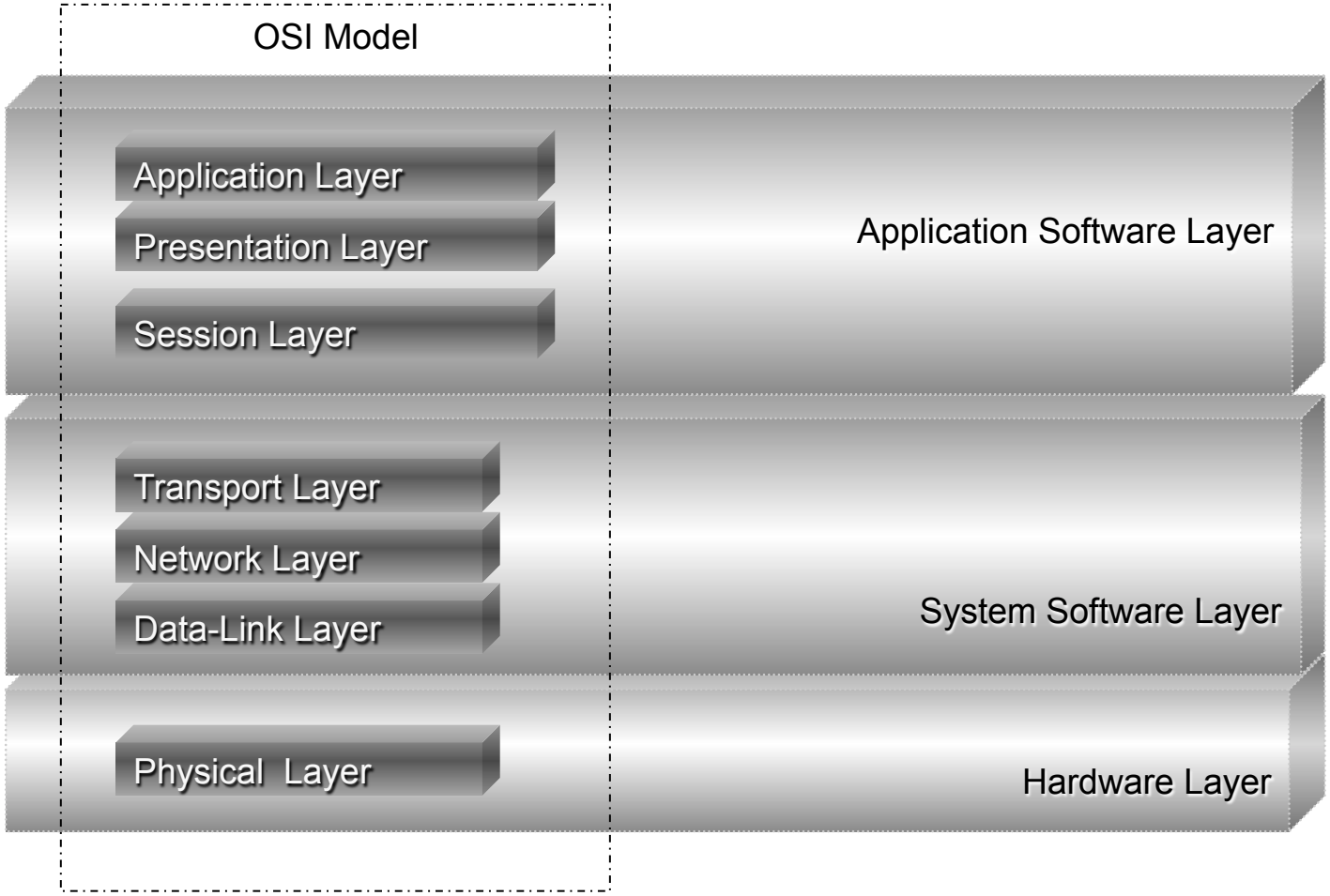
ENGN8537

Embedded Systems

Overview

- Communication Stack
- SPI, I2C
- USB
- RAM, ROM, Flash

Communication Stack



Communication Stack

Application Layer

The application layer is the uppermost in the stack. Conceptually this means that it shouldn't carry data on behalf of another protocol.

Application layer protocols are things like HTTP, NTP etc. They are 'final products'.

Of course, some application protocols have been subverted to carry other protocols; for example, web applications use HTTP to carry data associated with other protocols e.g. JSON which is in fact a presentation layer protocol itself.

Communication Stack

Presentation Layer

This ensures that both ends of the protocol agree on data formats, that is, the data is ‘presented’ in the same way.

This layer is rarely implemented separately from the application layer, as the presentation of data is dependent on the data itself which in turn is heavily dependent on what the application’s actually trying to say.

Communication Stack

Session Layer

Responsible for controlling ‘conversations’ between each end. This synchronizes the applications at each end and ensures that the data that’s transferred doesn’t fall on deaf ears.

Most of the functionality of TCP counts as a session layer.

Communication Stack

Transport Layer

Relied on for end-to-end transmission, error correction, reliable transmission.

The rest of TCP fits in here, ensuring that packets that leave one end actually arrive at the other, retransmitting as necessary.

Communication Stack

Network Layer

Logical Addressing. Allows applications to request connections to particular logical locations like IP addresses.

There isn't much that seems logical about IP addresses, but in fact they're hierarchical identifiers that originally could location particular machines in particular labs, companies, countries etc. As the number of computers wanting IP addresses outstripped the number actually available, this model was broken and the IP address is now pretty arbitrary.

Communication Stack

Data Link Layer

Physical Addressing. Logical addresses are associated with an entity like “the server for web page X”, physical addresses are associated with particular interfaces on particular machines.

In common computer networks, the physical address is the ‘MAC’ address.

Communication Stack

Physical Layer

Responsible for actually twiddling electrical/light/radio states.

Generally also able to detect connection to a network, collision between data sent from the local device to the interface medium.

SPI

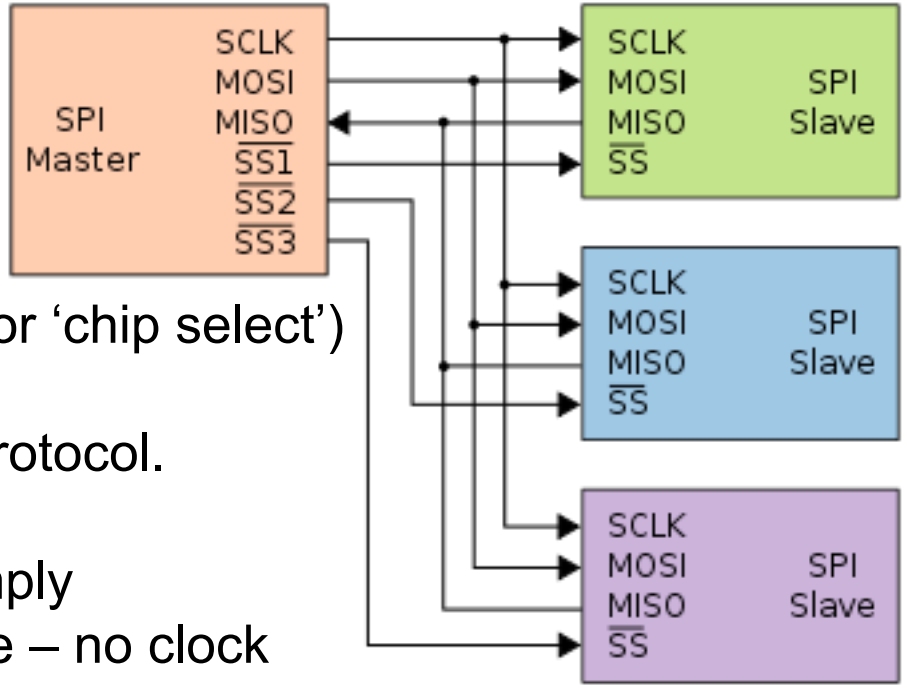
Serial Peripheral Interface: SPI

One master, many slaves.

One ‘slave select’ line per slave (or ‘chip select’)

Full-duplex, synchronous serial protocol.

SPI is in many ways the most simply communication interface out there – no clock recovery, addressing has its own, broken out wires, separate wires for data in and out etc.



SPI

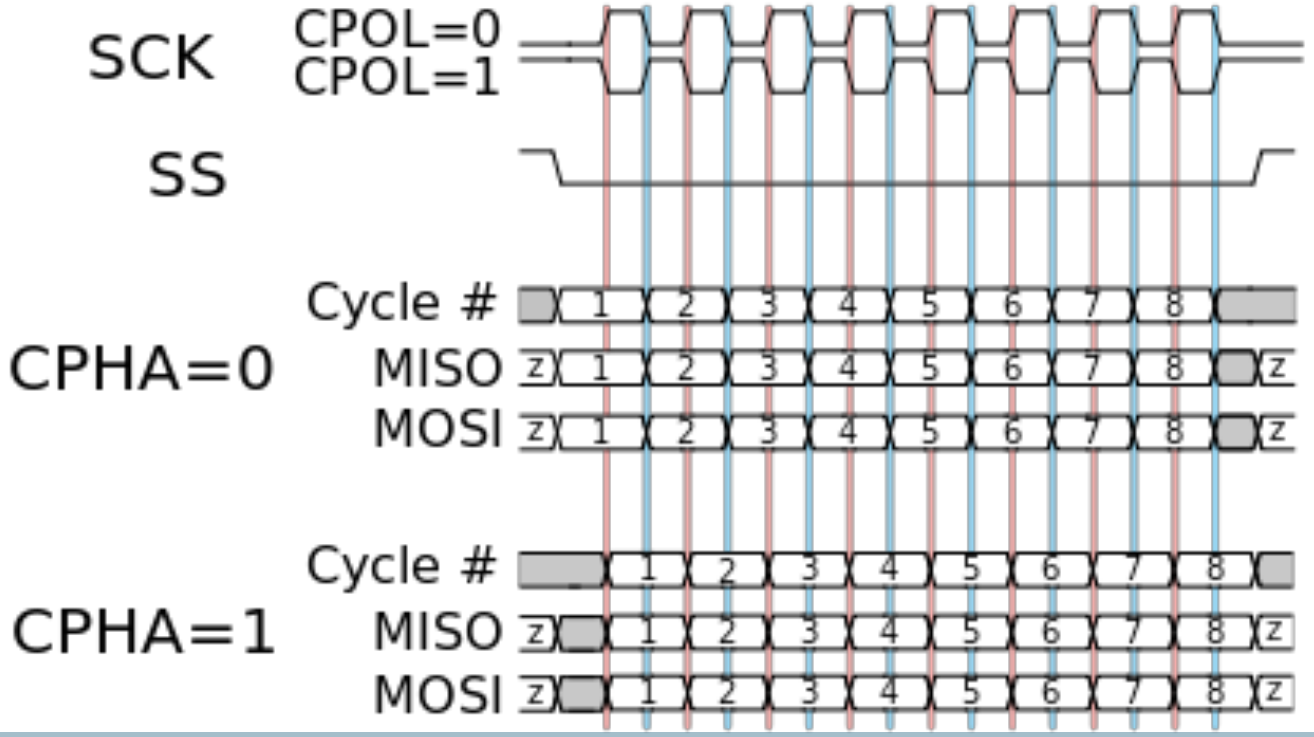
Very commonly used for simple peripherals in Embedded Systems such as

- Serial memories
- SD Cards
- Sensors
- Interface devices
- etc.

Can be run at several MHz if required, so useful for high rate ADCs and memories.

SPI

SPI defines the Physical Layer only, the data transmitted on the bus is not at all defined by the standard. The clock edge at which data is sampled is also not defined by SPI so care must be taken to ensure compatibility between devices on the bus.



I2C

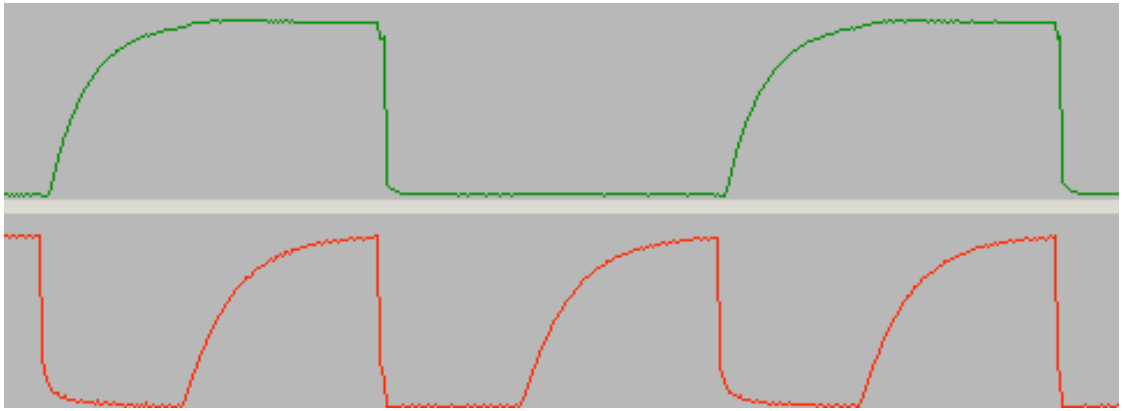
More tightly standardized than SPI, it defines the required clock rate, a multi-master arbitration protocol and even the type of data to be transferred.

I2C was trademarked by Phillips, but rather than jump through licensing hoops, many manufacturers chose instead to implement exactly the same protocol but call it something different. A common alternative name to the same bus is ‘TWI’, the two-wire interface.

I2C

I2C has two wires, a clock and a data line. As such, it is half duplex.

Both lines are ‘Open Drain’, which means that they are driven low but never driven high, instead, pull-up resistors must be fitted to the bus. This means that whenever nothing is explicitly pulling the lines low, the bus idles high.



I2C

The open drain design means that when ever a device expects the bus to be high, it must check this fact. If the bus isn't in the expected state, this means that another device is driving the bus at the same time.

Either this is another bus master fighting for the bus, or an expected slave acknowledgement.

I2C

In addition to the actual data, there is a start condition, stop condition and an acknowledge slot. The data must be valid when ever the clock is high

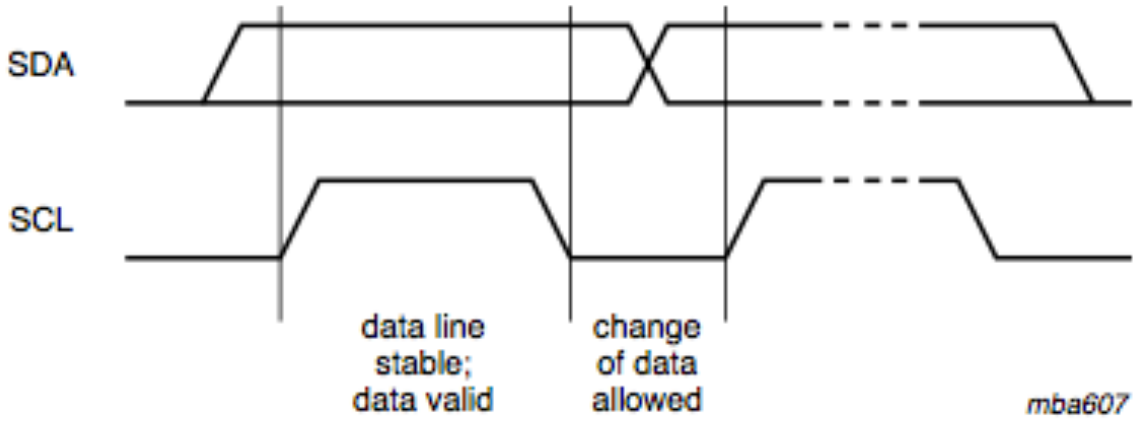


Fig 4. Bit transfer on the I^2C-bus

ref I2C specification

I2C

If the data changes while the clock is high, it is interpreted as a control condition. Every transfer starts with SDA falling while SCK is high and ends with SDA rising while SCK is high

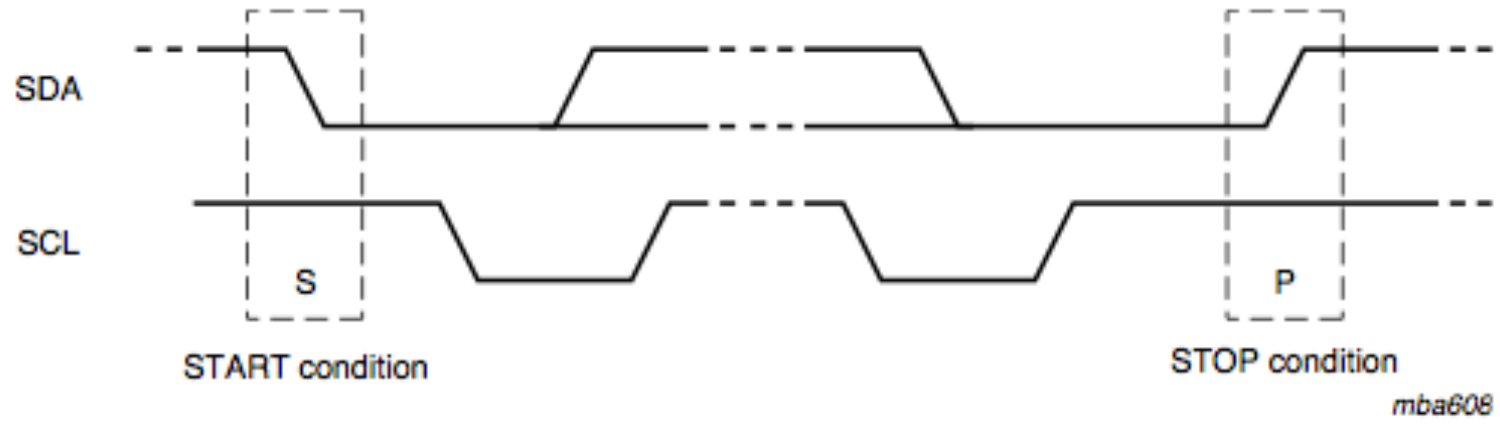


Fig 5. START and STOP conditions

ref I2C specification

I2C

Transfers may run for several bytes, separated by acknowledgements. ACKs are single bits that are driven in the opposite direction to the data. That is, if the master is writing, the slave acknowledges each write. If the master is reading (the slave is writing), the master must ACK.

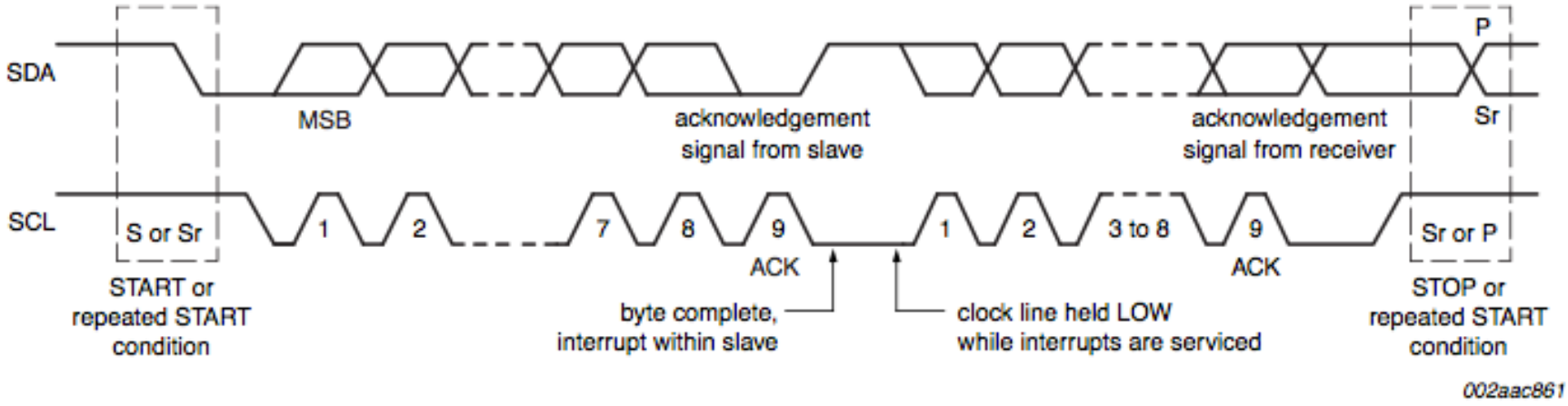


Fig 6. Data transfer on the I²C-bus

ref I2C specification

I2C

The ACK slot also plays another role. When the master is writing to the slave, the slave must ACK every byte the master sends.

If the master is reading from the slave, the ACK is used to indicate how much data should be read. If the master ACKs a byte read from the slave, then the slave must provide another byte. If the master doesn't acknowledge a byte (it "NAK"s it), then it is a signal to the slave that the transfer is complete.

In this way, a lack of ACK isn't always an error condition.

I2C

The slave can also indicate to the master that it needs more time to complete a process. The slave can force the clock line to stay low while the master expects it to be high. This is called 'clock stretching'

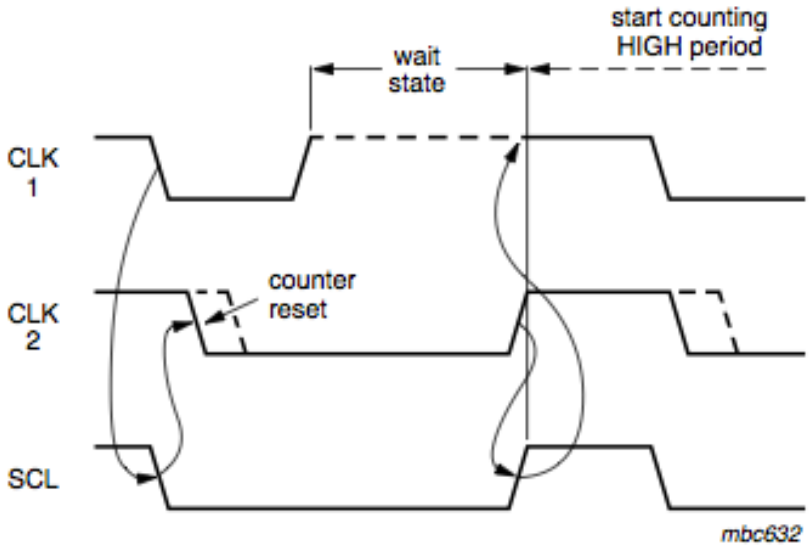


Fig 7. Clock synchronization during the arbitration procedure

ref I2C specification

I2C

I2C defines the addressing on the bus as well, though it doesn't differentiate between physical and logical addresses. The address of a slave is often programmed in to that device at the factory.

The first byte after the start condition contains a slave address and a bit indicating whether the master wishes to read or write data.

I2C

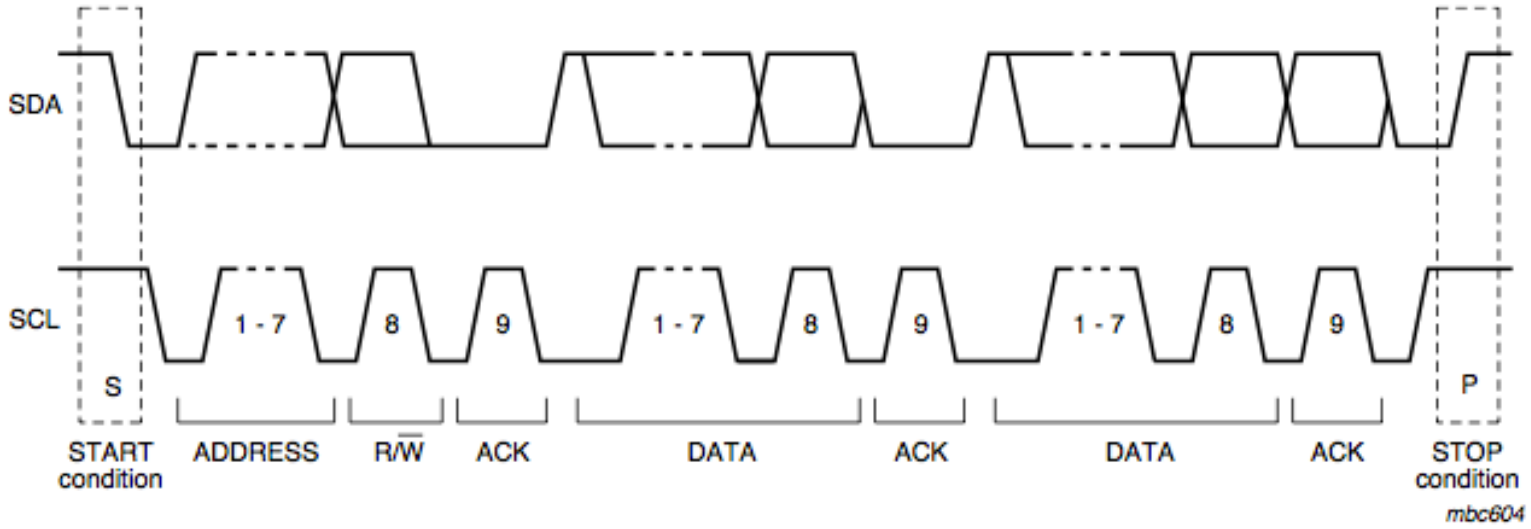


Fig 9. A complete data transfer

ref I2C specification

I2C

Many other protocols are built in top of I2C.

- PMBus: Power Management Bus is specified to carry messages regarding voltages, currents, power transfers and the like
- SMBus: The System Management bus is commonly found on computer motherboards, communicates simple messages such as ON/OFF
- DDC: Display Data Channel lives inside VGA, DVI and HDMI connectors and allows the video card to query display parameters such as resolution, supported colour depth and the like.

I2C

I2C is also used as a control channel in parallel with data channels.

- Camera modules commonly use I2C for configuration and a parallel colour bus for the picture information
- LCD screens often use I2C (or SPI) for configuration and a parallel colour bus for the picture information
- Audio CODECs use I2C for configuration and I2S for data transfer (you should know that from labs!)
- Standard computer memory modules use I2C for presence, capacity and error correction information while the parallel memory bus carries the read and write operations.

USB

USB is by far the most commonly used interface for computer peripherals and devices, although its use in Embedded Systems is somewhat limited.

USB is fairly complex to implement and, though the spec allows for guaranteed bandwidth or latency allocations, few devices take advantage of this.

USB

Everyone is familiar with the architecture of USB, with devices connected through hubs in a tree structure.

Inside each physical device there are a number of logical ‘endpoints’. Endpoints in each device are logically connected by ‘pipes’.

There’s always a pipe between Endpoint 0 at each end, referred to as the Control Pipe. This carries set up information at start up, including information about how the other endpoints are to be wired up.

USB

Pipes between endpoints can either by message pipes, like the control pipe, or stream pipes. Stream pipes are used for most data transfers.

Stream pipes in turn are divided in to three classes: Interrupt, Isochronous and Bulk.

Stream pipes are unidirectional while message pipes are bi-directional.

USB

Interrupt transfers can provide bounded latency assuming a complete knowledge of the devices on the bus and that all other devices are well-behaved.

Interrupt transfers are necessarily small amounts of data that must be delivered, such as information from USB mice and keyboards.

Note that bounded latency isn't low latency, it's still up to the system designer to ensure that the guaranteed response times are actually guaranteed responsive enough.

USB

Isochronous transfers can tolerate data loss more than they can tolerate latency. USB speakers, webcams and the like use Isochronous transfers.

Isochronous transfers have a guaranteed rate (which is generally, but not always, as fast as possible). This rate must be requested by the USB device when it is plugged in, in order for the OS to confirm that the guarantees can be satisfied.

In order to make these guarantees, the computer must reserve the right to reject a device's application for bandwidth; there isn't an infinite amount of it around! In practice, it's rare for one to try and connect so many webcams that the bandwidth is consumed and new devices get rejected, but it can happen.

USB

Bulk transfers take up the ‘rest’. Any bandwidth that isn’t reserved for the other two classes is allocated to bulk transfers.

This is designed to be used by things like USB storage devices, but in practice many devices that should use other types end up using bulk transfers anyway.

For example, a USB to Serial converter

USB

A common USB peripheral in Embedded Systems are “USB to Serial Converters”. These replace the standard serial ports that were present on most older computers with USB versions. This same concept is extended to “USB to I2C Converters”, “USB to SPI” etc.

These should probably use an isochronous channel in order to guarantee that there is sufficient bandwidth to move the required data. The problem is that the amount of bandwidth required is related to the baud rate and the baud rate is known when the converter is used, not when the converter is first connected. Recall isochronous packets must declare their bandwidth when they’re connected; as such USB to Serial converters are generally bulk devices.

USB

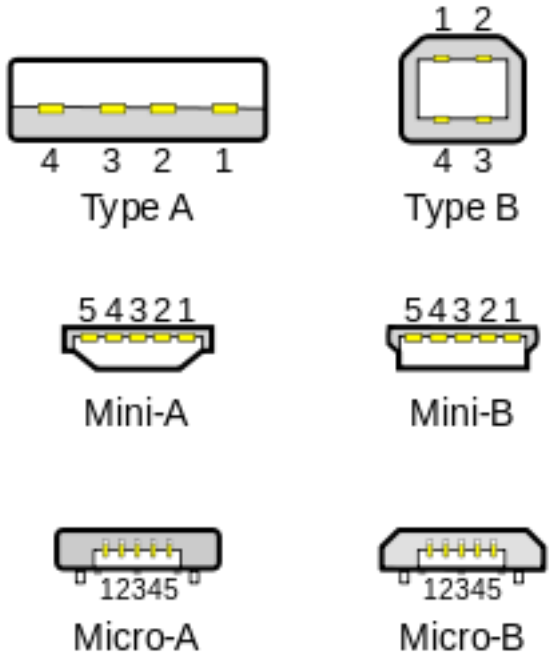
Because they use bulk transfers, they have neither guaranteed latency or bandwidth. This means that an Embedded System using such a converter for communications can't make any real time guarantees about delivery.

In practice, the simple act of plugging in a mouse can ruin your communications channel.

USB

Unlike the other busses here, USB also specifies the connectors to be used.

Upstream (Master) Downstream (Device)



USB

USB also specifies an amount of power that must be available to a down-stream device through the connector.

A device must draw no more than 500mA from a USB 1.x or 2.0 port or 900mA from a USB 3.0 port, all at 5 volts.

This simple requirement has meant that USB is specified as more than a communications bus, many devices use it just for power!

Memories

There are many different ways to categorize memories:

- Volatile or Non-Volatile
- Interface type (parallel, serial, differential)
- Writable or read-only

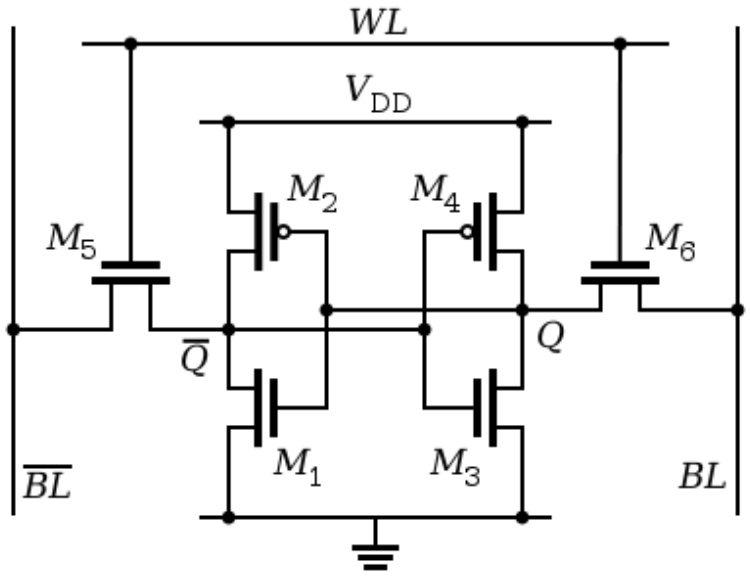
Memories

There are many different ways to categorize memories:

Memory	Interface	Volatility	Read Only
SRAM	Parallel	Yes	No
DRAM	Parallel	Super	No
EPROM	Parallel/Serial	No	Yes
EEPROM	Parallel/Serial	No	No
FLASH	Parallel/Serial	No	No
FRAM	Parallel/Serial	No	No

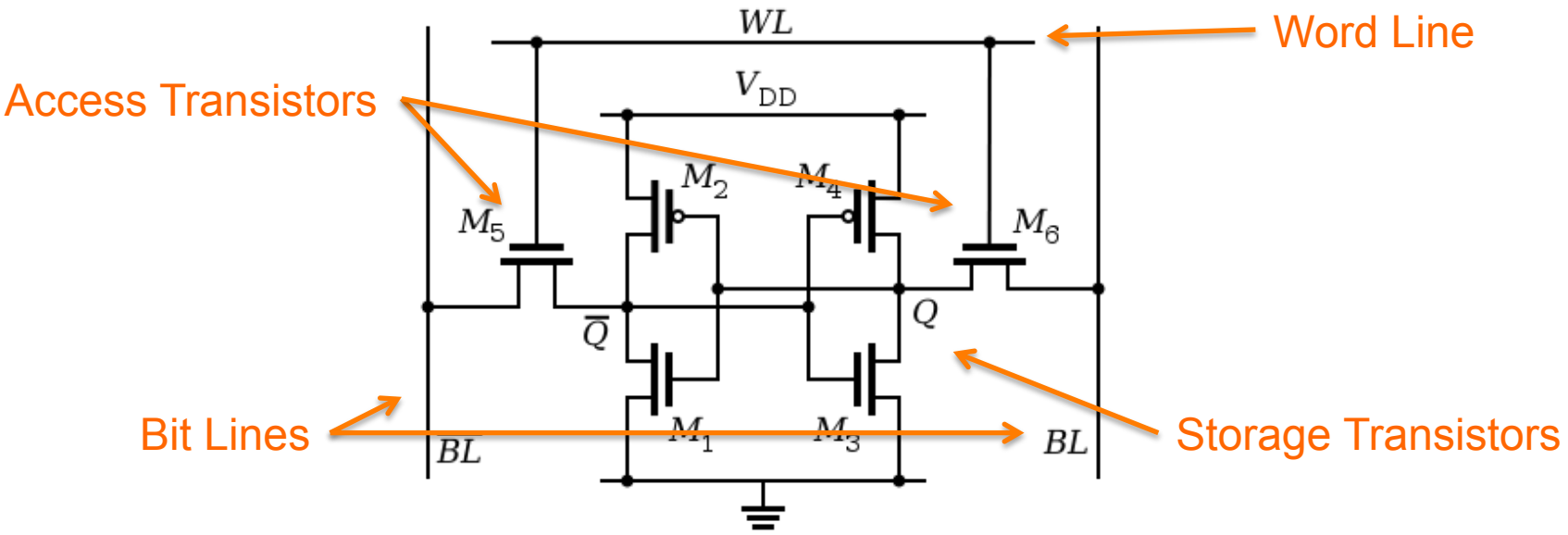
SRAM

The simplest type of volatile memory. It takes at least six transistors per bit which is quite a few compared to DRAMs. Consequently SRAMs are typically lower capacity than DRAMs, usually no more than 32- or 64-Mbit (4-8MB).



SRAM

The simplest type of volatile memory. It takes at least six transistors per bit which is quite a few compared to DRAMs. Consequently SRAMs are typically lower capacity than DRAMs, usually no more than 32- or 64-Mbit (4-8MB).



DRAM

Stores the bit values in capacitors inside the IC rather than in the states of cross-coupled inverters. This takes fewer transistors, but capacitors self-discharge so the data inside has to be regularly refreshed.

This is the ‘dynamic’ in DRAM.

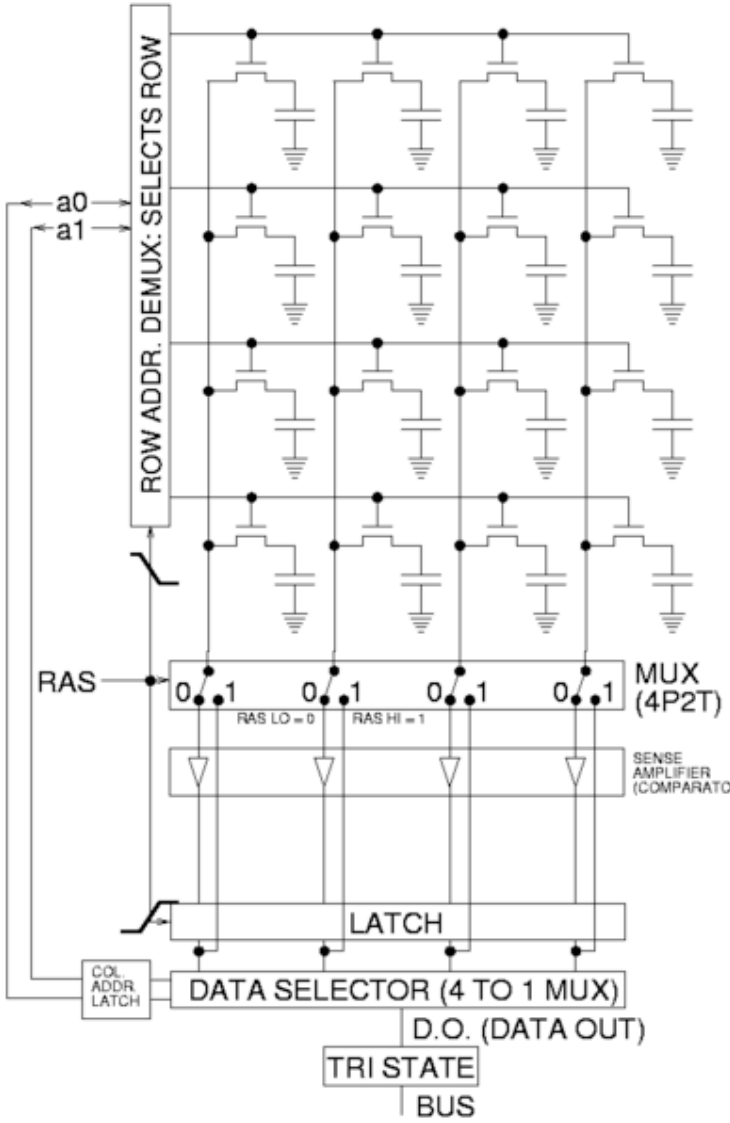
This makes DRAM more power hungry than SRAM in general, as energy is constantly being poured in to capacitors and wasted through their self-discharge.

DRAM

DRAMs are formed as matrices internally and this is reflected through the external drive scheme.

DRAM requires that both a row and column address be latched in, rather than having a single wide address bus.

This also means that fewer address lines need to be routed between RAM and processor for the same capacity.

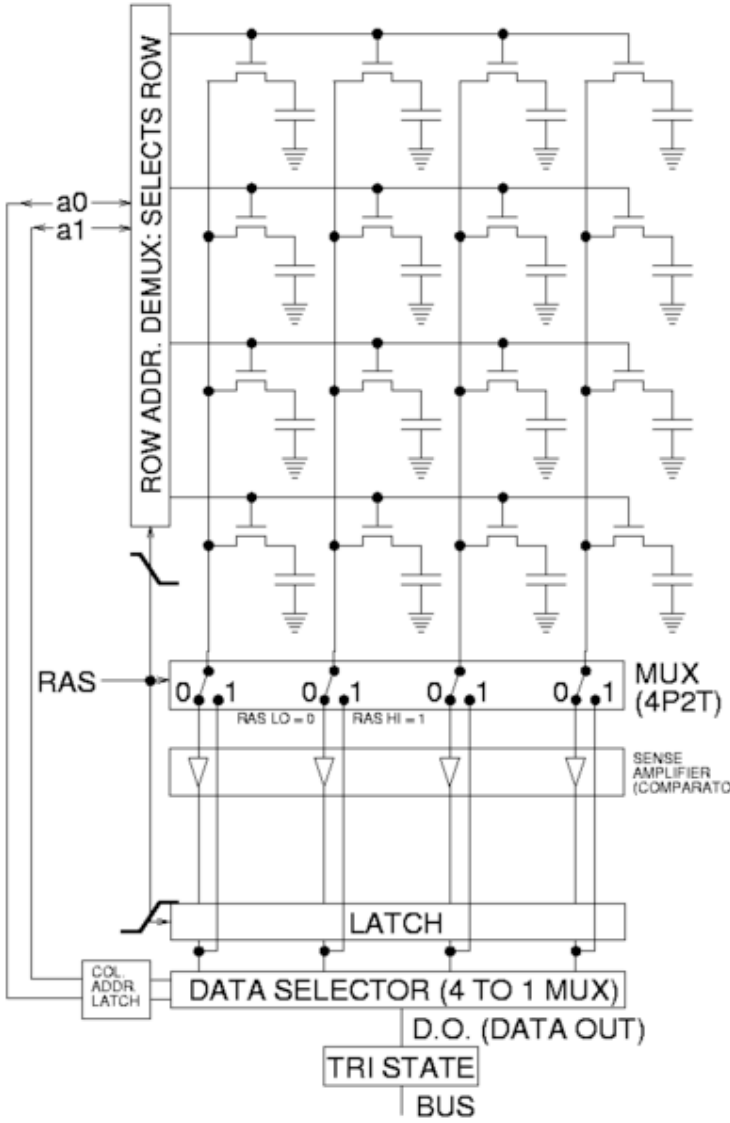


DRAM

The act of reading a row discharges that row's storage capacitors, destroying the data. As such, a read is actually a multi-stage operation internally:

1. Address a row
2. Amplify the small charges that have been read off the capacitors
3. Latch the values in
4. Write the latched values back on to the row's storage capacitors

This means that a read operation is actually how you do a data refresh as well.



PROM

Programmable Read Only Memory is made of a number of fuses or antifuses which are blown during programming.

The ‘blank’ state of most PROMs is logical ‘1’, not ‘0’, therefore blank bytes are 0xFF in hexadecimal.

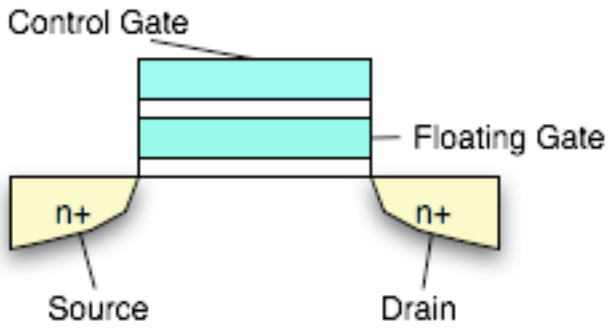
It isn’t possible to ‘unblow’ fuses, so zeros can’t be written back to ones, but it’s generally possible to blow more fuses after the original programming. As such, during multiple program cycles, each bit becomes the logical AND of the old and new bits.

PROM is one of the original ways that programs were stored in early computers and microcontrollers but is almost entirely obsolete now.

EPROM

EPROM uses floating gates on FETs to store the bit state. In effect, the gate of the FET is turned in to a capacitor which can discharge over time leading to a finite data retention time, though it's typically of the order of 100 years.

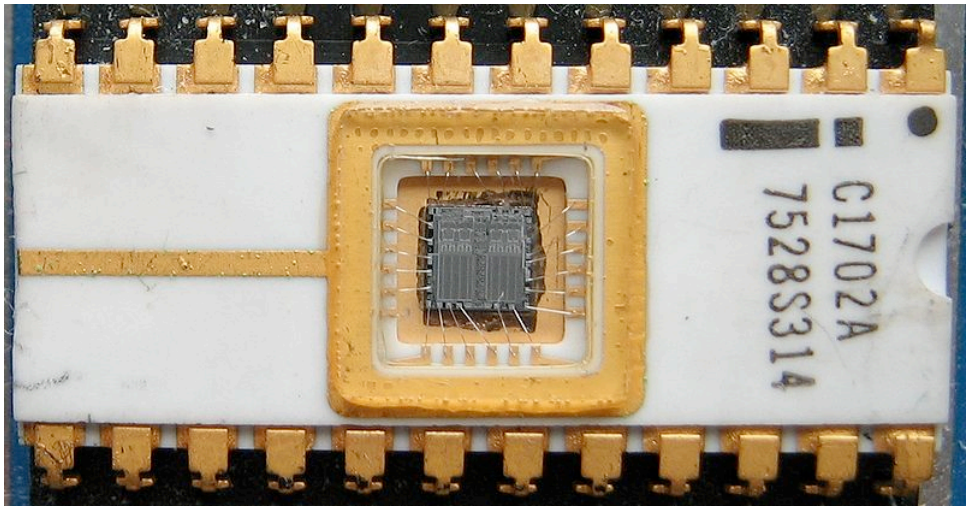
These are erasable by ultraviolet light as the high energy photons ionize the silicon, allowing the charge in the gate to dissipate



EPROM

Quartz windows in the top of the device allow the UV to enter.

The whole chip is exposed at once, therefore the chip cannot be partially erased.



from Wikipedia User ‘Poil’

EEPROM

Built in the same way to EPROM but with an extra silicon layer in the gate allowing the charge to be dissipated electrically. The dissipation slightly damages the silicon which limits the total number of times a bit can be written, typically to around 10,000 times.

While PROM and EPROM are rarely used, EEPROM is still used extensively for rarely-changed data, such as configuration parameters, serial numbers and the like. Many microcontrollers include some amount of EEPROM for non-volatile data storage.

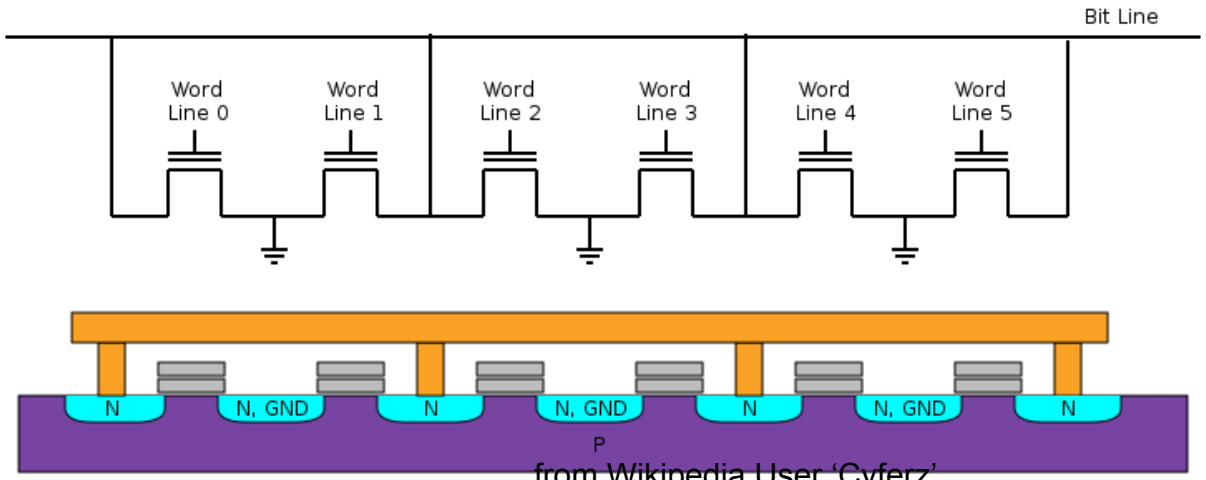
FLASH

Flash Memory is a specific type of EEPROM. The difference is in the erasure process; the mechanism is outside the scope of this course, however the effect is that Flash memory may be erased as many as 100,000 or 1M times.

Flash comes in two types, NOR and NAND.

FLASH

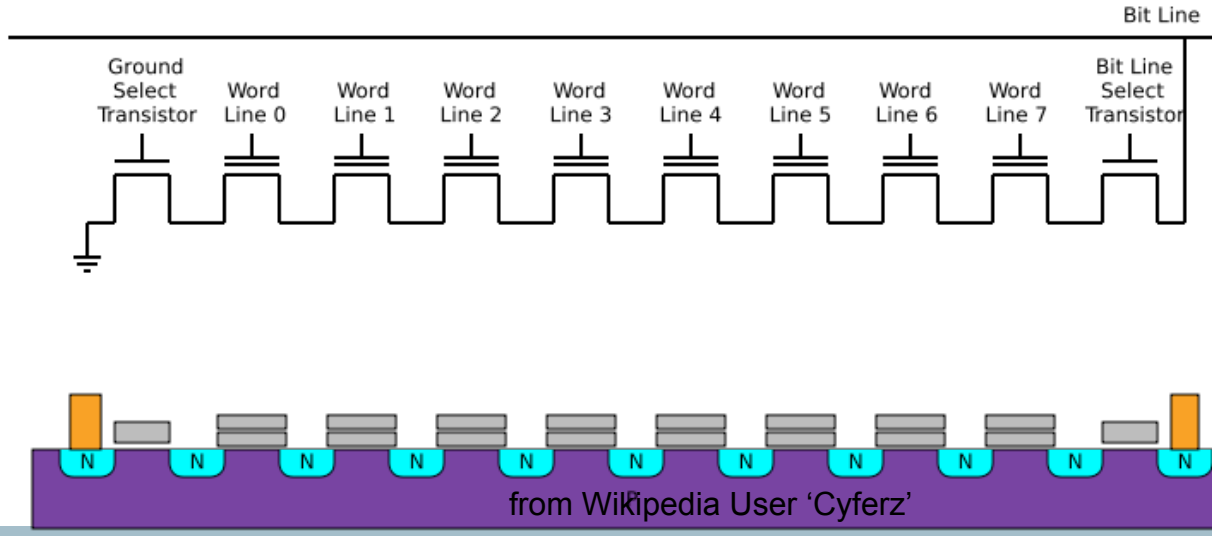
NOR flash is low density but relatively high reliability. It's simple to drive so often used for storage of bootloader or first-execute code which can't have any drivers loaded first. It works as the amount of voltage required to turn a transistor on (connect the bit line to ground) depends on whether there is charge on the floating gate. If the cell has been programmed then there is charge on the floating gate and the transistor will be able to pull the bit line low.



from Wikipedia User 'Cyferz'

Flash

NAND flash is higher density. Error rates are higher, as a fault in any one transistor in the chain causes a failure of all. In this Flash, the transistors each may be on either because it has been programmed (there is charge on its floating gate) or because it has been turned on manually by application of voltage to its word line. Say you wanted to read bit 2 from the chain, you would manually turn on transistors 0-1, 3-7 and then see whetherthe bit line was connected to ground (transistor 2 is programmed) or not.



from Wikipedia User 'Cyferz'

Flash

Like EPROM and EEPROM, flash memory can only be programmed from a logical ‘1’ to a ‘0’ – floating gate goes from no charge to fully charged. In order to set the bit the other way, the memory must be erased.

Flash memory is typically arranged in ‘erase blocks’ around 64KBit; if any bit in any byte in the block is to be erased, all must be erased.

Flash

NOR flash can typically be accessed byte at a time.

NAND flash can theoretically be accessed bit at a time, however due to the larger capacity of NAND in general, addressing individual bits is not practical.

In fact, NAND flash may only be able to be accessed page at a time, where a page may be between 512 bytes and 8 kilobytes.

Flash

EEPROM and Flash memories must be ‘wear leveled’. Each erasure causes damage which in turn limits the number of times each bit can be erased.

If a particular piece of data changes often, it can wear out its piece of memory very quickly. A memory driver should differentiate then between the ‘logical’ address and the ‘physical’ address. The same logical bit of data should be written to different physical locations each time so as not to stress one part of the flash unduly.

Flash

Flash memory is used inside things like SD cards and USB memory sticks which implement wear leveling internally.

Many such devices cheat at this by assuming the device is only ever going to be formatted with the FAT filesystem. In FAT, one part of the disk is written often (the file allocation table after which the filesystem is named) and the rest is written occasionally.

As such, many SD cards only wear level the blocks in which the FAT is expected to live. The upshot: Formatting an SD card with anything other than FAT can reduce its lifespan by hundreds of times!