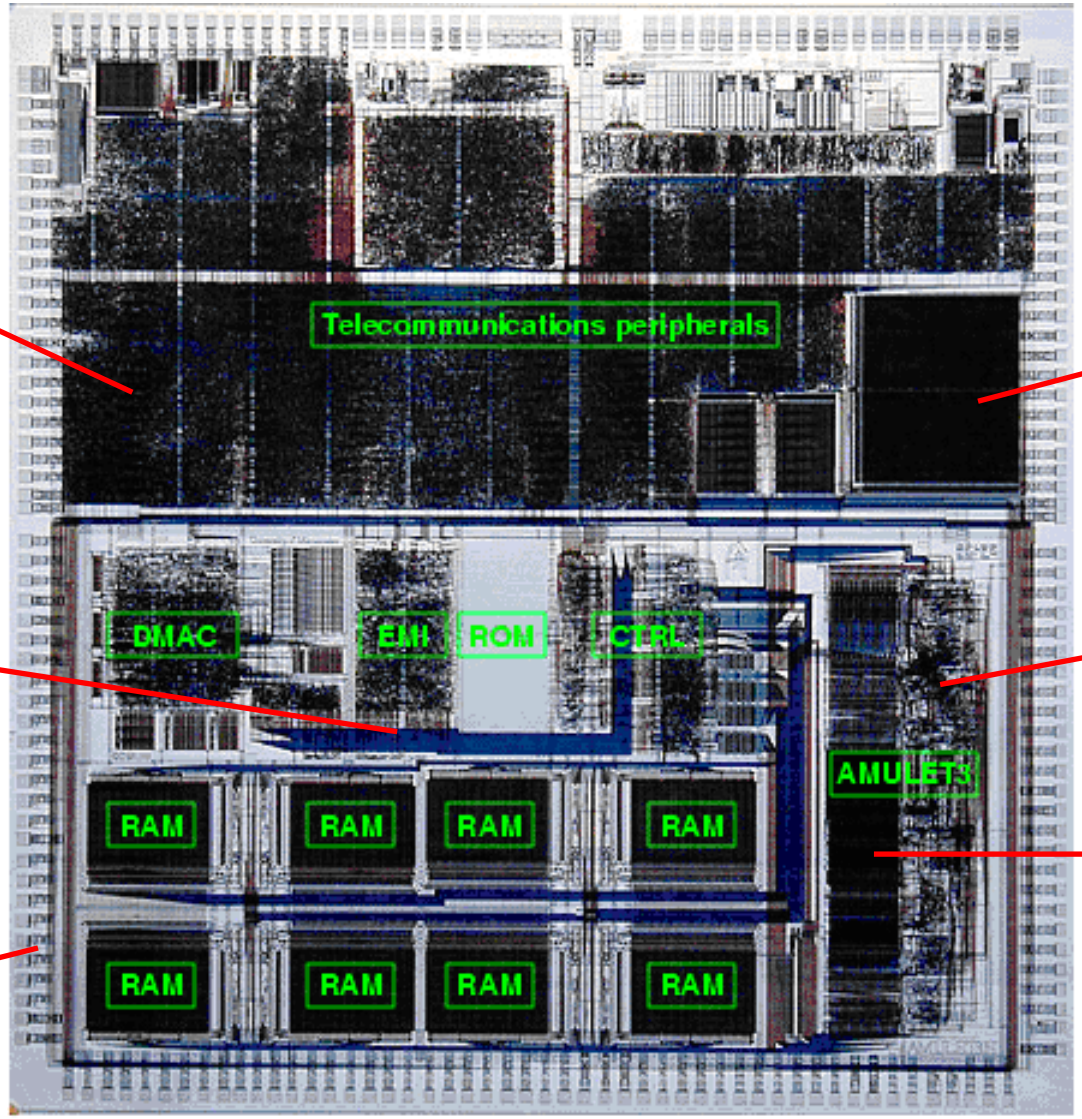
What is VLSI?

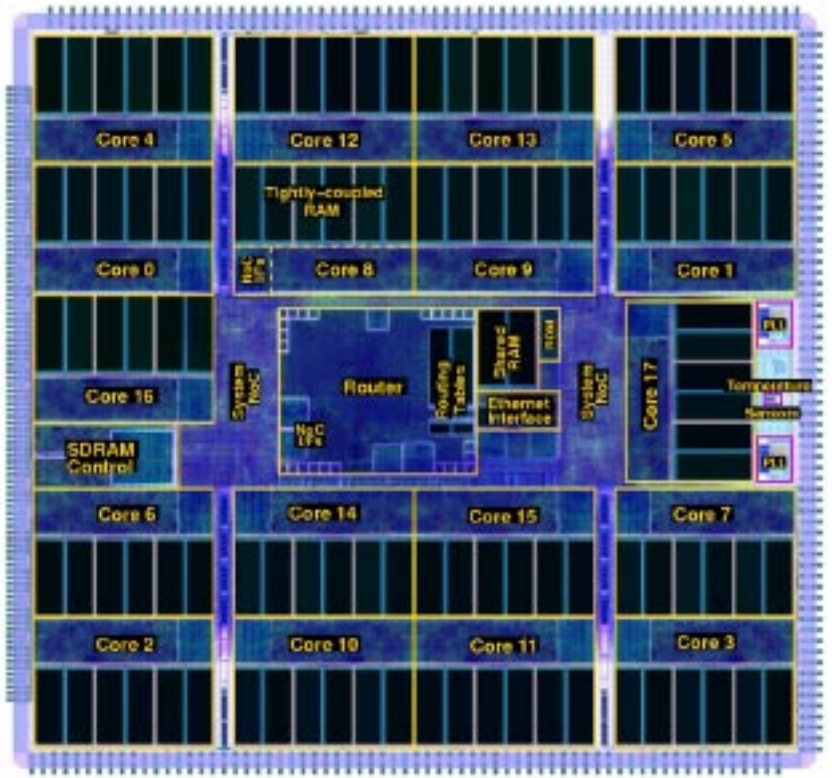


DRACO chip c. 2000: 350 nm 3LM: about 2 million transistors

The chip shown on the slide is somewhat ‘old-fashioned’ in that it has a considerable amount of hand layout, for (e.g.) the processor datapath and some of the high-level interconnect. This helps to exhibit the different design styles, however.

This is a three-layer metal process which means the wiring density is not so great that it obscures all the detail. Ten years later there may be ten metal layers so much less is visible.

The figure below shows the SpiNNaker chip, which is a more modern device.



This chip is manufactured in a 130 nm geometry and occupies about 1 cm^2. It has 18 ARM processors each with 96 KB of RAM – so RAM tends to dominate the chip area – together with on- (and off-) chip networking and a few other shared resources. (The RAM dominates the transistor count too: each bit store uses six transistors so a byte is about 50 transistors, thus the whole chip is ~100 million transistors.) The processor nodes are compiled from the RAMs and standard cells and ‘hardened’ – i.e. their layout is frozen. These are then used as macrocells in the top level composition.

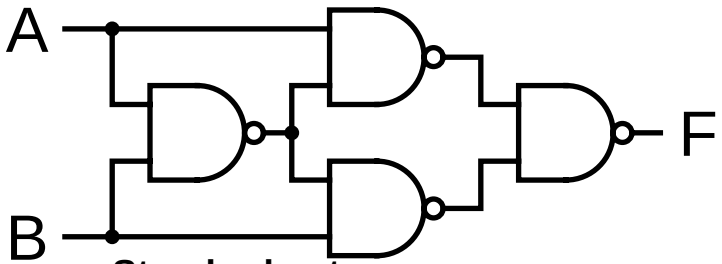
The repetition of macrocells with identical layout means that only one example needs to be simulated in detail. This is a huge time/resource saving.

Note, also, the small area of analogue components at the centre of the right hand edge.

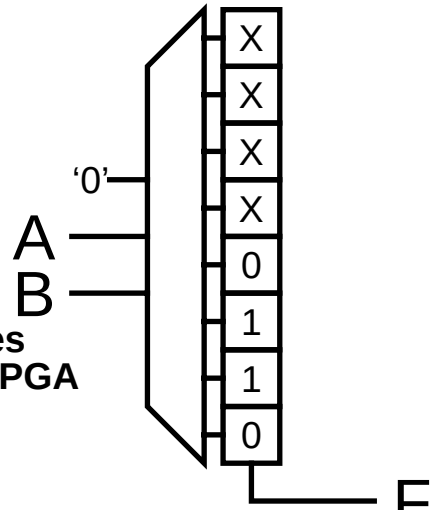
Logic Implementation

The FPGA approach illustrates that gates are not the only way to implement logic.

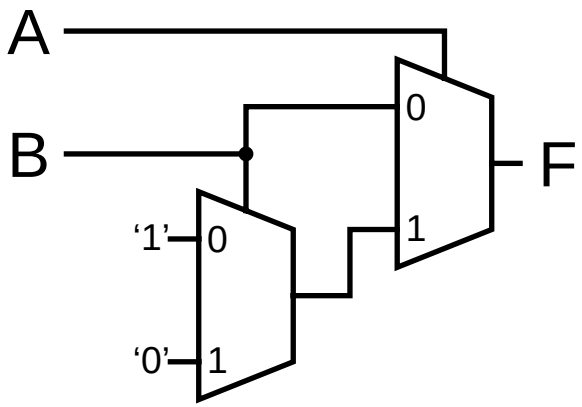
Implementation of Combinatorial Logic: F = A xor B



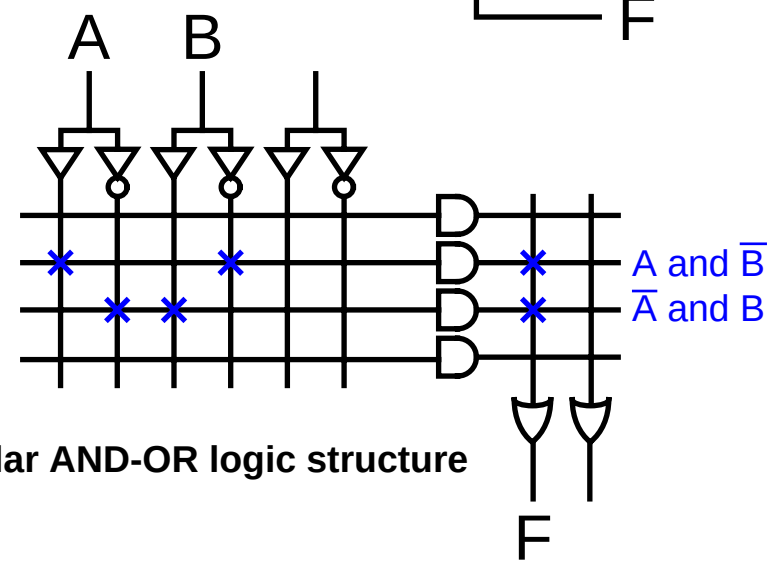
Standard gates
Typically used in ASICs



Look-up tables
e.g. used in FPGA



Multiplexer logic
Maybe suitable for optical switching



PLA
Regular AND-OR logic structure

Different implementation technology

Look-up tables

Look-up tables are used in FPGAs because a RAM look-up of a truth table is an easy way of providing a *programmable* function. These functions will typically be more complex than a single logic gate. There is also an advantage in that the logic *delay* is roughly constant for the function.

Larger look-up tables may be useful in other calculations although their is prohibitive in some cases. For example, dividing (slow operation) two 16-bit numbers could be looked-up given a $2^{16} \times 2^{16} = 2^{32}$ entry ROM but this would not fit on mast chips. However a 64 K ROM could feasibly provide a reciprocal which could be multiplied by the other operand.

DSPs (Digital Signal Processors) often contain ROMs for ‘calculating’ sines and cosines (which can use the same look-up, of course). This can also be useful in software, trading memory for speed.

PLAs

Programmable Logic Arrays (and similar) used to be used extensively as customisable logic components both as field-programmable devices and ASIC components. The technology is (largely) now superseded.

A PLA exploits the fact that any logic function can be derived with two levels of logic: an AND of an arbitrary set of inputs (some possibly inverted) followed by an OR of the AND outputs. (In practice this can be two NANDs in series.)

The drawback to this is that the ‘gates’ may be infeasibly large as there is a limit to the practical fan-in on most logic implementations.

Multiplexer logic

The slide illustrates a logic function implemented purely using 2:1 multiplexers. If those multiplexers were implemented in CMOS this would be quite inefficient.

However, it is conceivable that some future logic could provide multiplexers (switches) as its basic technology. An example might be optical computing where it has been demonstrated that a laser beam can switch another (with suitable support from a refractive medium)

More ‘exotic’ logic

This subsection is for interest rather than memorisation.

Dynamic logic

Dynamic logic cuts a normal, static CMOS gate in half, typically to just the NMOS transistors (they are the smaller, more efficient type, remember?). The output of all such gates are *precharged* with one PMOS transistor per gate and then left *floating* ‘high’. Because they are capacitive these wires will stay high for some time. The NMOS inputs are then applied and conditionally discharge the node, resulting in the logic function.

Dynamic logic can be smaller and (debatably) lower power than static CMOS; it is more complex (needs clocking) and sensitive to layout variations. Really it is only suitable for full custom applications.

Pass transistor logic

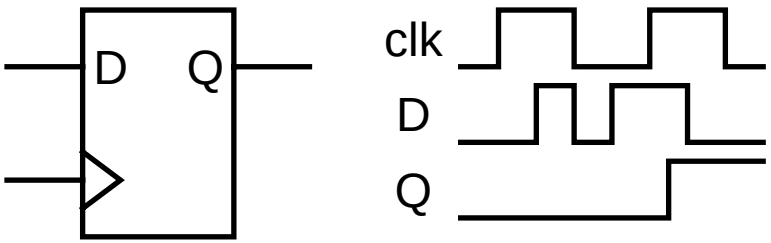
Gates need not be CMOS structures, even using MOSFET technology. There are other possible structures such as using *pass transistors* with their own advantages and disadvantages. However this is not the time so investigate these further.

D-type flip-flops

There are several different storage elements available to a designer.

{RS flip-flops, transparent latches, …}

On the whole, stick to D-type flip-flops.



Reminder:

These are **edge-triggered** devices which, when **clocked**, copy their input to their output

at other times they simply hold their data stable

- ‘D-types’ are a very simple-to-use form of memory element
- Highly suited to synchronous design
- Very easy to specify in HDLs such as Verilog
  - `always @ (posedge clk) Q <= D;`

Typically have several in parallel to form a **register**.

… and `clk` is a global signal so all flip-flops change **synchronous**ly.

Synchronous Design

[This should be revision.]

The vast majority of digital designs are synchronous.

There is a good reason for this!

In a synchronous design the next state of any subcircuit can be calculated from the current state and any inputs. This is a static evaluation. When (after) everything is ready the clock will cause a global state change and the next cycle begins.

This simplifies:

- the design
  - each block can be designed in isolation, knowing that its inputs will change only at clock edges
- the logic
  - as signals race through the logic changes can arrive at a gate's inputs at different times
  - this can cause glitching
  - glitches are blocked by the next flip-flop; the logic stabilises before this is clocked
- the analysis
  - everything can be timed by counting clock cycles
  - the minimum clock period can be determined from the logic

Asynchronous logic is possible but it is subject to many more constraints.

Because synchronous design is an attractive paradigm support has evolved targetted at this style.

[Some of the following references things you may not have met, yet.]

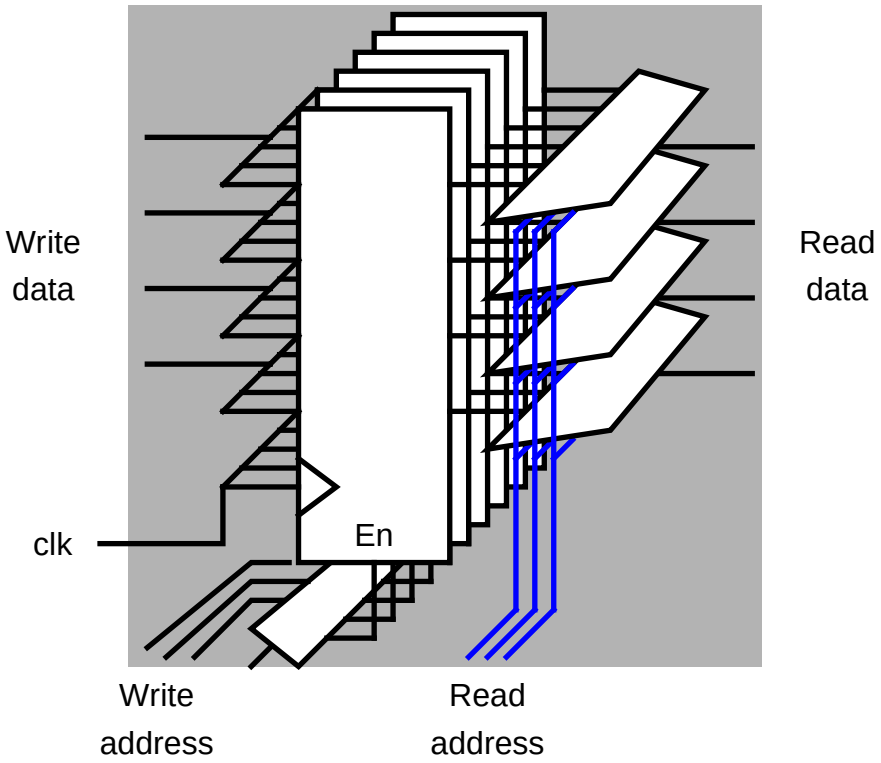
- HDLs – such as Verilog – support synchronous designs directly. Using '`posedge clk`' to initiate state changes is the normal approach.
- Timing analysers use the clock as a reference to measure other delays.
- Most 'IP blocks'[1] will already be designed this way.
- FPGAs are optimised for synchronous designs, containing many D-type flip-flops and dedicated clock distribution networks.

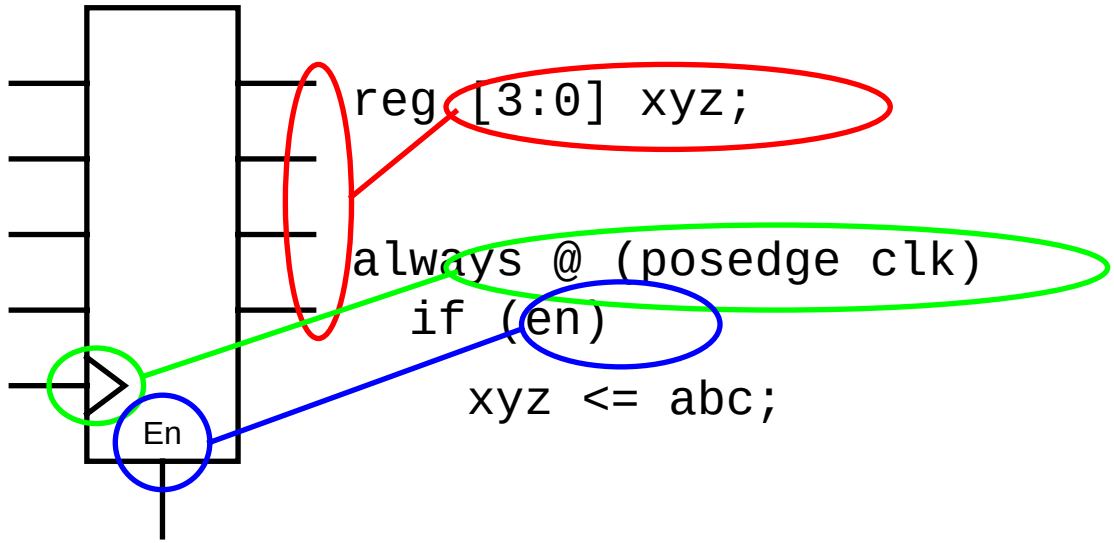
1. IP ('Intellectual Property') here would encompass 'bought in' designs which you may include in you own design. For example, when a company like Apple or Nokia design a 'phone chip they may license an ARM processor rather than building their own. Licensing a design costs money but saves time, not necessarily just on the hardware as software development tools may be available too. Also, by using a known design, incoming engineers save on having to learn something new.

Registers

"Register" is another word that means (slightly) different things depending on context.

- To a programmer a register will typically be an architectural feature (R0, R1, etc.)
- To an architect this may extend to include temporary variables (e.g. pipeline registers)
- To an engineer they are probably any group of flip-flops written/read together





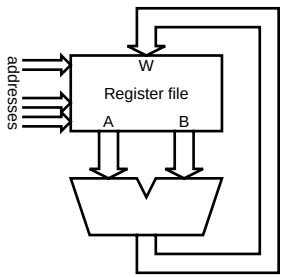
- Regardless, wherever possible keep them synchronous
  - control writing with an enable

- A typical register in an implementation will be a stand-alone structure corresponding to a single variable.
  - Therefore it doesn't have an 'address'.
- Although often potentially written to from several different sources only (at most!) one write is possible at once.
  - … obviously!
- It may be read by fanning out its value to as many places as expedient.

Register file – e.g. in a processor

A regular structure which *might* be mapped into a more compact form (for efficiency/convenience/neatness)

- Contains several/many registers
  - each has an address
- Typically fewer addresses than registers
  - therefore only a subset of the registers accessible each cycle
- In a processor (etc.) choose how many independent read and write **ports**
  - e.g. for a simple example:
    one write port
    and
    two read ports
  - the addressed registers could be the same on any/all ports



Technology mapping

A design may be described be a schematic, showing the actual components used or a HDL source which specifies functionality and only hints at structure.

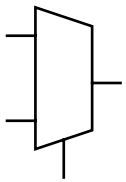
To synthesize a HDL description it is first 'elaborated', which extracts a structure for the design comprising components such as adders, memories etc.

The resultant design is then '**technology mapped**' which involves a translation of these blocks into available gates. The tech. mapping stage needs to know about the appropriate gate library.

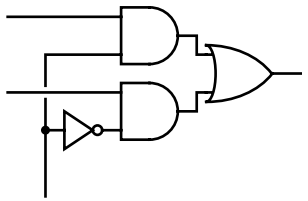
A library may specify a set of ASIC standard cells; alternatively the mapping could be into an FPGA (see later).

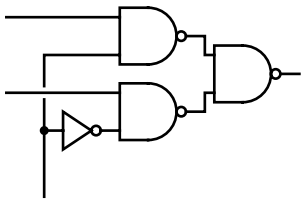
The gate mapping may differ depending on what is available. For example:

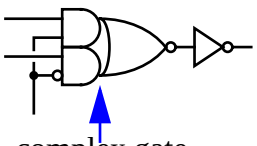
```
if (S) Q = A; else Q = B;
```



custom cell
(if available)



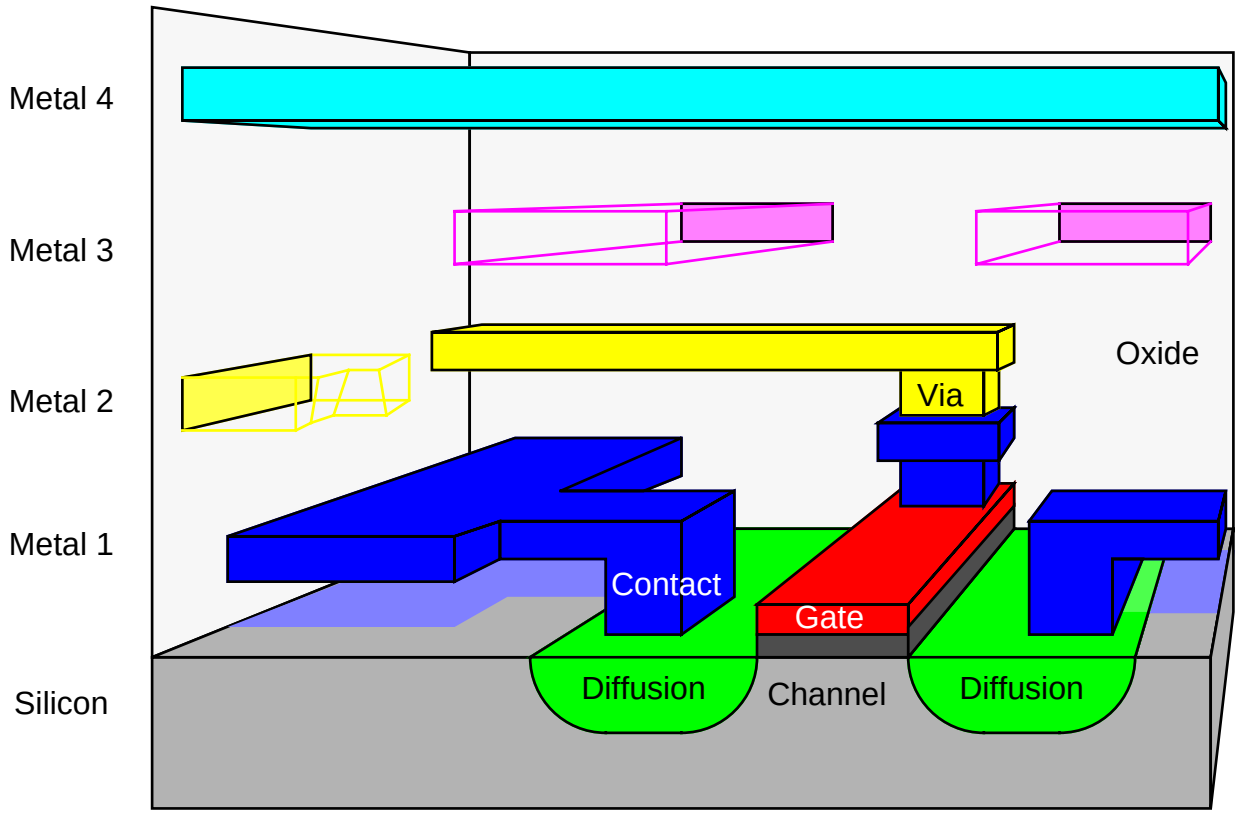




complex gate
(not covered in this module)

Layout

- An integrated circuit is built on the surface of a piece of silicon
- The active components (transistors) are basically two-dimensional
- Each gate occupies some finite physical area
- Interconnection is by wiring
  - in ~10 metal layers above the silicon surface



The Incredible Shrinking Transistor

The driving force is to get the maximum number of components into the minimum area which provides:

- ❏ Greater functionality for a given price
- ❏ Higher speed
- ❏ Reduced energy needs (for a given function)

Advances in this field are what drives the (so called) "**Moore's Law**", where the number of components which can be integrated doubles roughly every 18 months.

To a good approximation, all the advances in computing over the last fifty years have derived from these higher integration levels with the consequent increases in speed. The advances in computer architecture and so forth have probably contributed less than a ten-times improvement over this period; the other many-million-fold improvements have come from process improvement.

To put some sense of scale on this, a '28 nm process' has a transistor gate length of 28 nm so a transistor will be somewhere between 100 nm and 200 nm across. A 'fairly simple' 32-bit processor like an ARM may occupy about 0.2 mm^2 and a complete embedded computer – with adequate RAM, which probably dominates the area – can fit inside 1 mm^2. Typical chip sizes may be in the range 20 mm^2 to 100 mm^2 (larger and smaller are possible).

For comparison:

- ❏ Visible light has a wavelengths around 500 nm
- ❏ Viruses are typically 10 nm - 100 nm across
- ❏ A large molecule will be a few nanometers across
  - ❍ e.g. hæmoglobin: 6-7 nm

Design Rules

When it has to be, layout is drawn using a 2D plan view with coloured shapes representing the different layers. This is usually referred to as "polygon pushing".

There are many **design rules** associated with the physical layout. Whilst there is – in principle – a lot of commonality a set of design rules will be specific to a particular process from a particular **silicon foundry** (a.k.a. '**fab**'). The design rules specify such things as the minimum (and sometimes the maximum) width of features such as wires; they will also specify minimum spacings to keep wires (and other features) sufficiently far apart.

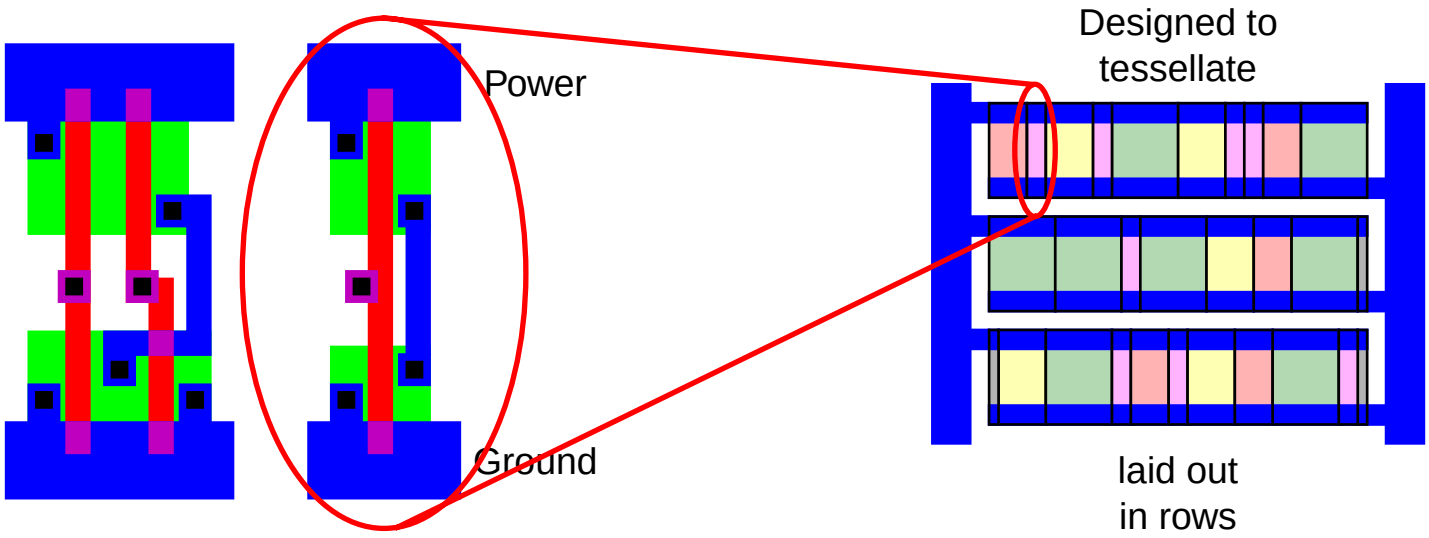
The reason for design rules is to ensure that the device can be manufactured reliably. The features when manufactured will not have sharp edges as there are uncertainties in the various chemical deposition and etching processes. By obeying the design rules wires will be thick enough not to contain gaps and spaced far enough to avoid touching.

There may also be rules about vertical placement because the surface layers are not completely smooth and the wiring layers become increasingly 'lumpy' as they are built ever higher.

Usually the design rules are confidential to the foundry and all the ASIC designer will see is a library of gate 'phantoms' which show the standard cell outlines and come with a behavioural model for simulation. For those who need to customise further the design rules may be licensed; in this case layout should be run through **DRC** (a Design Rule Check) which is a CAD tool which highlights any violations.

Before fabrication the foundry will run DRC themselves. Your design must pass this before proceeding or the foundry will take no responsibility if all your chips come out dead.

Standard cells



- The basis of most ‘random’ logic
  - Each cell is a gate, flip-flop, etc.
- Designed to be placed edge-to-edge
  - ‘Heights’ (“pitch”) identical
  - ‘Widths’ may differ
- Wiring superimposed on metal layers, above
- Libraries of cells from the silicon manufacturer

Gate implementation

It is possible to design at transistor level and draw every transistor by hand. In theory this could produce the most highly optimised designs; in practice it would take a very long time!

The normal approach is to use a library of prepared gates (and flip-flops etc.) which whose layout has already been done and checked. These are then instantiated where they are needed. A typical library will contain a familiar set of basic CMOS cells such as ANDs, NANDs, ORs and NORs.

Gates vary in (physical) size but they must all be wired together later. To facilitate this the layout of these is normally in the form of standard cells.

Standard cells are designed on a **common pitch** but may be different lengths. The length may be an integer multiple of a basic size to simplify placement on a grid.

Cells are then abutted in rows so that their power rails all connect together; this solves one of the wiring problems. The cells' layout is such that any pair of cells can be abutted without violating any rules. It is also possible to 'reflect' the layout if this makes connection easier.

Rectangular areas comprising several rows of cells are formed, padding out any gaps in rows to form a regular block. This can be fed with bigger power buses at its ends.

The cells use only the lowest layer or two of metal so can be connected into from above. The majority of the wiring is added above the cells.

Gate arrays

A chip is built in layers, each requiring one or more **masks** for etching purposes. These are high technology (expensive) products made once per design. Because of the cost of masks for a whole chip the popularity of gate arrays (*not filed-programmable*, in this case) has returned somewhat. In these a regular layout of standard cells is provided and the user customises the wiring only. This gives a device between an ASIC and an FPGA (q.v.) in performance, logic density and development cost.

Floorplanning

- Floorplanning is the layout of 'big' components on a chip.
- It is not necessary for chips which are place-and-routed in a single operation.
- It is used when the layout process is hierarchical.
  - i.e. some blocks are assembled and '**hardened**' …
  - … then used as components for later layout.

Hardening blocks:

- saves effort if blocks are instantiated several times
- can simplify simulation as blocks can be tested separately
- removes some layout freedom, making later routeing harder

A block can be compiled from standard cells and then hardened. It will typically have connectors placed around its periphery.

Other **hard macrocells** may be imported which are laid out using different tools: the most common is probably memory.

Memories

Building a memory out of D-type flip-flops may be convenient but it is expensive …

… because D-types are relatively large/bit stored

- Suitable for ‘a few’ bits – such as processor registers
- Useful if several parallel operations required
  - e.g. a processor register bank
- Inappropriate for larger stores

Next step is usually to use Static RAM (SRAM)

- Much smaller per bit
- Somewhat slower access
- Usually single ported
  - i.e. either one read *or* one write per cycle
- Sometimes multi-ported
  - at significant area/speed cost

Memories

A bit stored in SRAM will occupy roughly 10% of the chip area of a bit stored in a register.

Static RAMs are *asynchronous* structures – like transparent latches they can be write-enabled at which point the addressed word may change – but on-chip are often given a synchronous 'wrapper' so that reads and writes (appear to) occur on clock edges. This can simplify their interaction with other logic in HDLs etc.

RAM is (fairly) easy to specify in Verilog:

```
reg [31:0] my_RAM [0:1023];
reg [31:0] data_read;
...
data_read = my_RAM[address];
```

although access to *part* of the data requires an extra alias since the basic Verilog syntax will not allow explicit multi-dimensional arrays.

On an ASIC a RAM will probably be provided as a separate block and connected explicitly. The Xilinx FPGA synthesizer will recognise a RAM specification and attempt to build it into the netlist; however it will only be able to use the on-chip resources if the description matches what is available.

Dual-port RAM

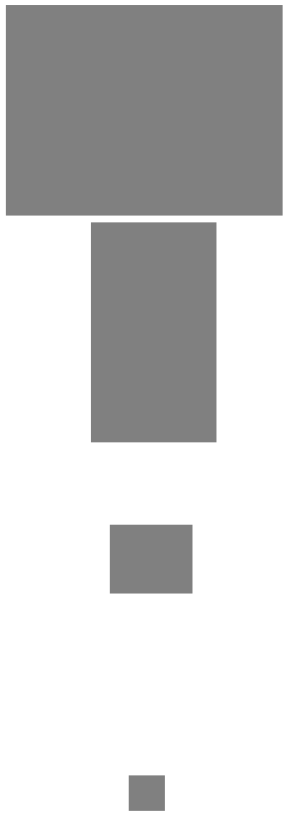
It is possible to build extra access ports into SRAM although the expense – a rough halving of the area efficiency – means that it is not done very often. This allows two independent operations simultaneously, often, but not always, a read and a write. A dual-port RAM will have two address ports and two data ports (of each type). Because wiring space is already limited and extra wires are needed a dual-port RAM will occupy about twice the area of a single port one. Because of the increased wire capacitance it will also be slower. Multi-port memory is therefore often an inefficient solution to an ASIC problem.

ROM

- Much more compact than most RAM (order of DRAM cell size)
- Fixed contents
- Typically used for programmes or data tables
- May be used as logic element.

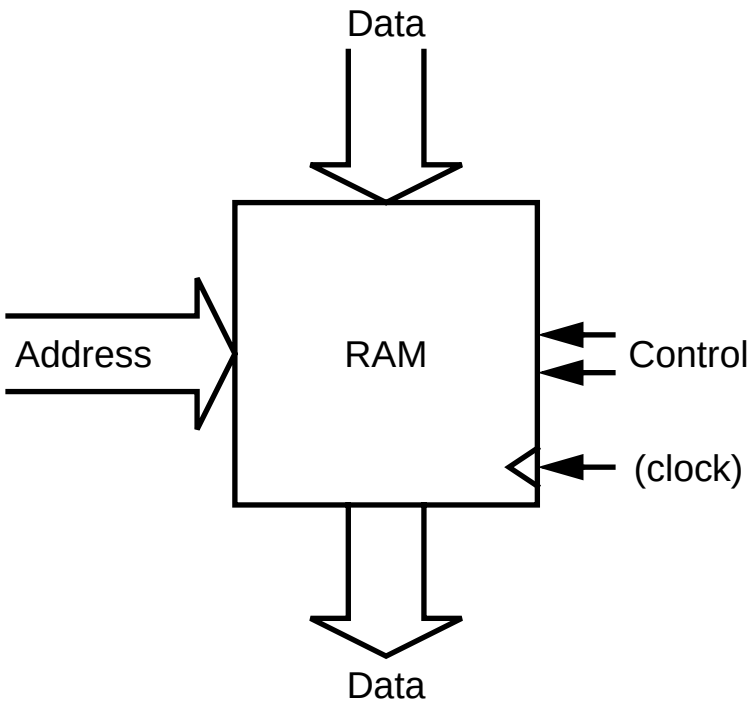
Memory cell sizes

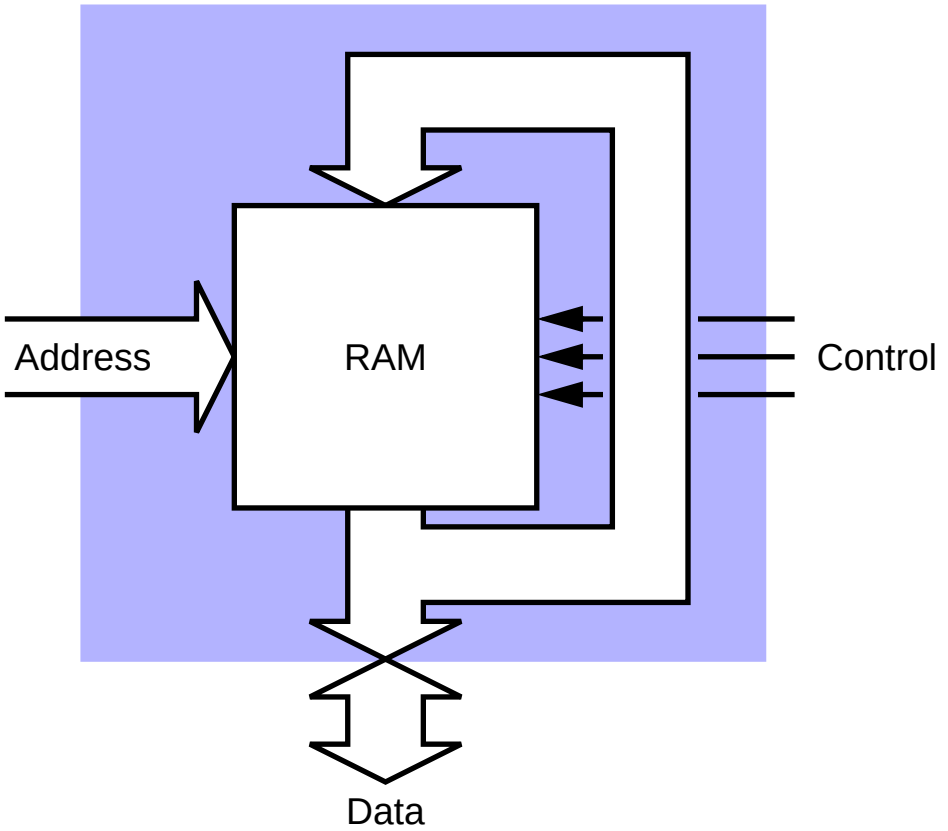
- D-type flip-flops
  - Convenient for synchronous logic (e.g. FSMs)
  - Very large area per bit
- Transparent latches
  - Okay for logic but not as convenient
  - Smaller than D-types, but still large
- SRAM
  - Small area per bit
  - Need (shareable) interface logic
  - Simple to use
- DRAM
  - Very small area per bit
  - Need considerable interface logic
  - Many awkward timing constraints



The grey squares give some impression of relative sizes.

SRAM implementation





On-chip have separate read and write data buses.

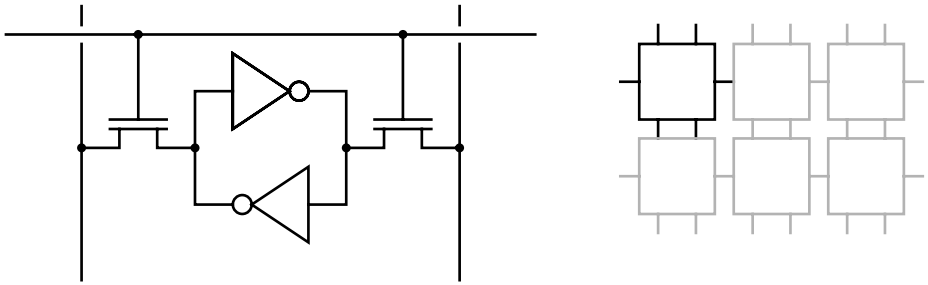
They are often clocked: i.e. the RAM operations will be synchronised to a provided clock

A separate SRAM chip will often use a multiplexed, bidirectional data bus to save pins; sometimes serial access to save even more!

SRAM chips are typically asynchronous.

Static RAM (SRAM)

The usual SRAM cell requires six transistors. Thus a kilobyte of SRAM requires about 50 000 transistors. Typically the transistor count in SRAM accounts for most of the transistors on an SoC.



Time and space preclude delving into the details of an SRAM circuit here. Sufficient to say that the (slightly odd looking?) circuit shown is built from small (probably minimum width) transistors and will tessellate for easy (compact) layout on the silicon surface.

Commodity SRAMs come in a range of sizes and speeds. Chips with a capacity of 16 Mb are available and 64 Mb may be coming soon (2011).
*NOTE: RAM sizes are always quoted in **bits**.*

On-chip the size of a SRAM is customisable. Typically a RAM compiler will be available whereby you can specify exactly the word length and – to a power of two – the number of words.

The logical size of the RAM will obviously affect its physical size but it may also influence its *aspect ratio* (i.e. the relative 'width' and 'height' on the chip surface). A logical size might be 1K × 32 but this could yield a very long, thin rectangle if implemented. In practice, as the bit cells will be roughly square, such a RAM would probably yield a block with (say) 256 words of 128 bits each containing multiplexers and demultiplexers to allow access to parts of each word.

Some Other Memory Technology

In a quest to get more (or 'better') storage in a smaller area other technologies may be employed. Even though these may be fabricated on silicon the technology is often somewhat different and they are not, typically, integrated with much processing logic. Some brief notes are included here purely for interest.

Dynamic RAM

Dynamic RAM (DRAM) has long been the choice for maximising storage/area. A bit is stored by charging (or not) a small capacitor. When the bit is read it discharges (or not) which is sensed as the bit state; the state then has to be restored to the cell. It has about 4x the area efficiency of SRAM although access is slower.

The capacitors, being imperfect, also leak away charge anyway. Thus the RAM is 'dynamic' in that it has to be 'refreshed' periodically (many times per second). This process increases the complexity of operation, demands extra power and may interfere with user access.

DRAM now comes with additional support circuits as **SDRAM** (Synchronous DRAM: note, still unlike SRAM). This has a clocked state machine to assist with some of the complexity of access *but still cannot be treated as synchronised with a 'system' clock due to skews*. Enough said!

Flash memory

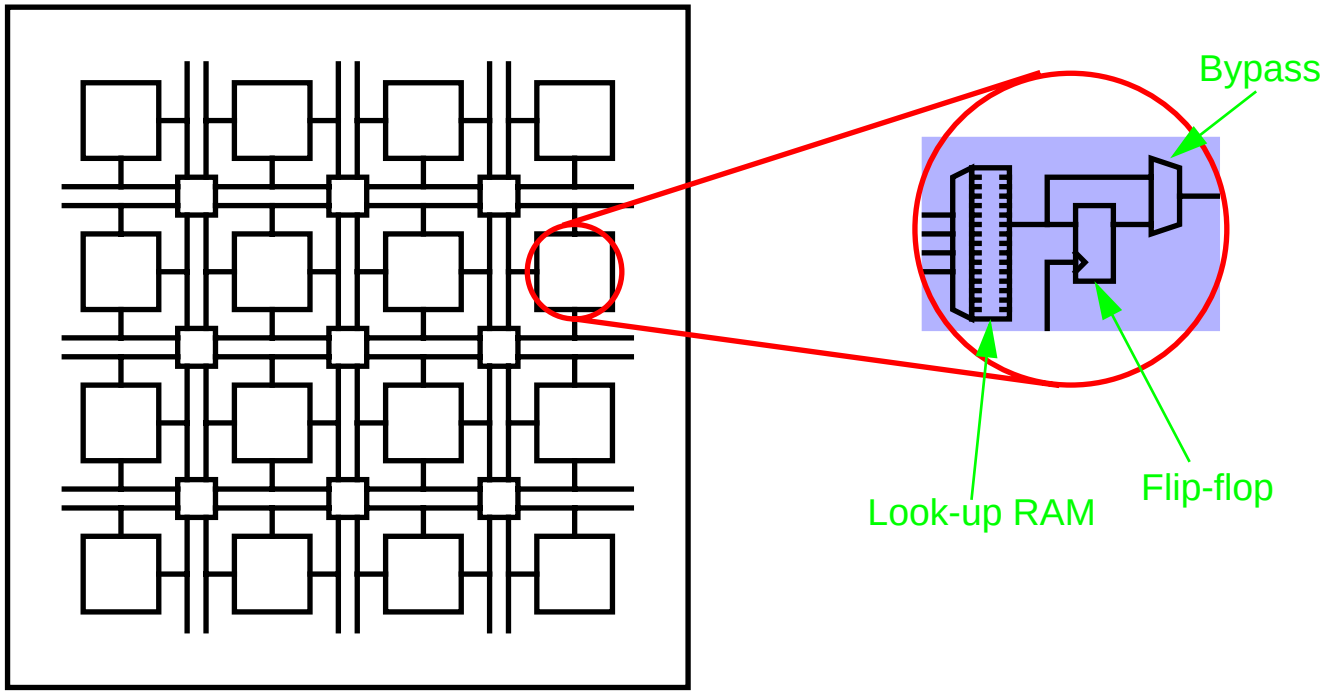
'Flash' memory is a form of EEPROM (Electrically Erasable, Programmable Read Only Memory) which is sometimes integrated onto devices such as microcontrollers to provide 'permanent' storage. It is non-volatile in that it holds data 'indefinitely', even without power. Storage is achieved using capacitance but these are (normally) 'unconnected' being a 'floating gate' on a transistor. The charge leakage is extremely low, facilitating data retention.

Reading Flash memory is reasonably fast but **writing** involves 'punching' charge through to the floating gates and **is very slow** (and also slightly damaging, sometimes resulting in a maximum guaranteed number of write cycles).

Whereas EEPROM is randomly writeable, Flash memory bits are only individually writeable in one direction, thus a block of the memory must be erased and then rewritten to incorporate a change.

FPGAs

- Field Programmable: Can be programmed in the ‘field’ – i.e. by the user
- Gate Array: matrix of logic elements



A typical FPGA comprises

- numerous programmable logic blocks, using small RAM look-up tables

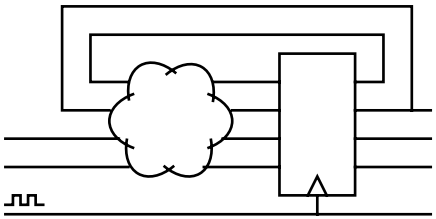
linked by

- a programmable switch matrix

FPGA architecture

The basic logic in an FPGA is a **look-up table**. Sizes vary, but this may be a 16x1 RAM which can thus contain the **truth table** for an arbitrary function of up to four inputs.

There will normally also be the *possibility* of latching the output with a D-type flip-flop. This reflects the structure of (a part of) a generic synchronous FSM (Finite State Machine).



These are interconnected using wires linked with programmable switches.

There are some other interconnections such as specialised clock distribution networks to allow synchronous operation without clock skew.

Most large FPGAs now also contain specialist logic blocks embedded in the matrix. These are much more efficient at implementing their single function – assuming that function was wanted. Examples include:

- ❏ RAM blocks
- ❏ Adder (carry) support
- ❏ Multiplier blocks
- ❏ Clock scaling
- ❏ Microprocessors
- ❏ Fast peripheral interfaces
- ❏ …

For many designs it is acceptable to allow the Place And Route (PAR) CAD tool to map a design into an FPGA. However the programmable interconnection can prove slow. For high performance applications it is possible to compile – automatically or by hand – **hard macrocells** which are pre-assembled blocks which can be placed into the array when the final design is routed.

Example FPGA

Xilinx Spartan-6 XC6SLX150

- ❏ 23 038 ‘slices’, each containing
  - ❍ four 6-input LUTs (total: 92 152)
  - ❍ eight flip flops (total: 184 304)
- ❏ 268 18 Kb RAMs (total: 4 824 Kb or about $\frac{1}{2}$ Mbyte)
- ❏ 1 355 18x18 multiplier-accumulators
- ❏ 6 clock managers, each with a PLL
- ❏ 4 memory controllers
- ❏ (up to) 576 user I/O pins

This is far from the biggest device around.

To give some sense of scale, a ‘back of envelope’ estimate would suggest an ARM 9 32-bit microprocessor would probably require about 5000 LUTs so, assuming around 50% utilization, this could encompass a 10-core ARM system each with about 50 KB of memory.

FPGA Pros …

- ❏ Fast development turnaround
- ❏ Can fix bugs
- ❏ Retargettable: easy ‘stock control’
- ❏ May contain useful IP (e.g. high-speed comms.)
- ❏ Available in ‘latest’ technologies
- ❏ …

…and Cons

- ❏ Lower capacity than ASICs
- ❏ Slower than equivalent ASIC
- ❏ More expensive (when in high volume)

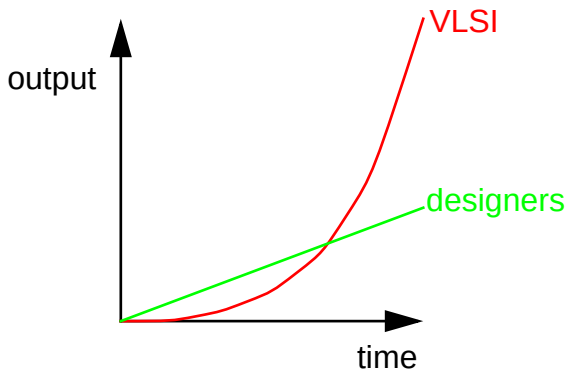
Conclusions

- VLSI devices are *very* complex – and increasingly so
  - Almost certainly the most complex systems ever devised
- Complexity can be managed through hierarchy and abstraction
  - There're a lot of steps between a logical design and a physical chip
  - Overcoming the physical limitations is increasingly hard
  - CAD tools provide a lot of assistance here
- Designer productivity is the limiting factor
- The gap is going to get worse!

Moore's Law revisited

"Moore's Law" has moved from an observation to a planning guide or 'road-map'. The industry somehow expects the shrinkage to continue and implicitly use this as a guide as to what should be available at which future dates. It has therefore become a self-fulfilling prophesy.

The growth in integration has been exponential. Designer productivity cannot match this. Silicon designers have become better, partly through increasing knowledge, partly by having more people and mostly through better tools. However there is an increasing 'design gap' between what is achievable and what can be made practically.



Moore's Law will not continue forever. There are physical limitations – such as the size of atoms – which must stop this in the fairly near future. This point has been "about ten years ahead" for the last couple of decades! Nevertheless, overcoming the problems is increasingly difficult and therefore increasingly expensive; it is probably that the economic barrier will be the limiting factor.

Future Possibilities

A single chip is now capable of holding a sizeable electronic system (System on Chip (SoC)) or a powerful multicore processor. There will soon be chips with hundreds of processor cores; exploiting these is an interesting challenge. **Parallel programming** may be forced to become a reality for lots of software.

As transistors shrink the number of atoms making up each transistor also shrinks. These are diffused in en-masse and so are subject to statistical variation. A couple of atoms difference is not much in a population of millions but starts to become significant when there are only hundreds. The **variation** of transistor properties is therefore becoming proportionately larger. Combined with the much greater number of transistors on a chip the assumption that every transistor will work *within acceptable parameters* is increasingly dubious. Future designs may be forced to include **fault tolerance** on the grounds that they will not work otherwise. This is a new design challenge.

Although there is an ever-increasing number of metal wiring layers above the chip the silicon is still basically a 2D structure. Space can be saved by making these very thin – **back grinding** after manufacture is now commonplace – and stacking them in a package. Connecting two chips together is *relatively* easy, in a 'flip-chip' combination; connecting more is harder although vias *through* the silicon are now being made. This allows systems of several chips without the speed and power penalties of driving PCB tracks.

This list is not meant to be exhaustive. Add your own examples here.