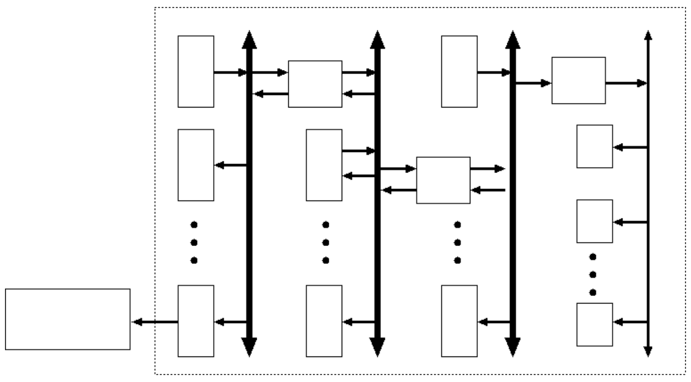
System on Chip
Design and Modelling



University of Cambridge
Computer Laboratory
Lecture Notes
FIRST HALF ONLY

Dr. David J Greaves

Part II
Computer Science Tripos
Michaelmas Term 2017/18

- **(1) Energy use in Digital Hardware.**
- **(2) Masked versus Reconfigurable Technology & Computing**
- **(3) Custom Accelerator Structures**
- **(4) RTL, Interfaces, Pipelining and Hazards**
- **(5) High-level Design Capture and Synthesis**
- **(6) Architectural Exploration using ESL Modelling**

0.0.1 SoC Design : 2017/18: Twelve Lectures for CST Part II

Over previous decades, most of the advances in computer performance have arisen from shrinking the physical size of the computer according to Moore's Law. But when we reached 100 nm transistor size, Dennard Scaling ceased and new computer architectures were required. Semiconductor physicists have provided a world where we can put much more logic on our System On Chip (SoC) that we can conveniently power up at once (Dark Silicon), meaning that application-specific accelerators are increasingly being used. How else does your mobile phone compress motion video without almost immediately flattening the battery?

In this course we examine the basic energy and performance metrics for todays chip multi-processors (CMPs), caches, busses, DRAM banks and custom accelerators and examine the need for, design of and integration of custom accelerators. We briefly visit all of the IP blocks found on a typical SoC, as used in the Raspberry Pi. We look at the future of reconfigurable computing and the role of FPGA in the datacentre.

Examples will assume knowledge of three languages, C, Verilog and assembly language but not require any degree of proficiency in these languages.

A system on a chip (SoC) consists of several different microprocessor subsystems together with memories and I/O interfaces and hardware accelerators. We shall cover the relevant design and modelling techniques leading to an overall emphasis on architectural exploration, where we can estimate energy and performance for a variety of designs without investing effort of making a detailed implementation. This is the "front end" of the design automation tool chain. (Back end material, such as design of individual gates, layout, routing and fabrication of silicon chips is not covered.)

[Note to supervisors: This year, Formal Methods and Assertion-based design are again not being lectured. Instead we have a new sections on Accelerator Structures and High-Level Synthesis. And, as per the last two years, the SystemC net-level facilities will not be lectured - we'll just do enough SystemC to cover TLM modelling in the ESL section.]

0.0.2 Recommended Reading

Subscribe for webcasts from 'Design And Reuse': www.design-reuse.com Ed Lipianski's Book

Embedded Systems Hardware for Software Engineers Ed Lipiansky. McGraw-Hill Professional (1 Feb. 2012)

Embedded Systems Hardware for Software Engineers describes the electrical and electronic circuits that are used in embedded systems, their functions, and how they can be interfaced to other devices.

Basic computer architecture topics, memory, address decoding techniques, ROM, RAM, DRAM, DDR, cache memory, and memory hierarchy are discussed. The book covers key architectural features of *widely used microcontrollers and microprocessors, including Microchip's PIC32, ATMEL's AVR32, and Freescale's MC68000. Interfacing to an embedded system is then described. Data acquisition system level design considerations and a design example are presented with real-world parameters and characteristics. Serial interfaces such as RS-232, RS-485, PC, and USB are addressed and printed circuit boards and high-speed signal propagation over transmission lines are covered with a minimum of math. A brief survey of logic families of integrated circuits and programmable logic devices is also contained in this in-depth resource.*

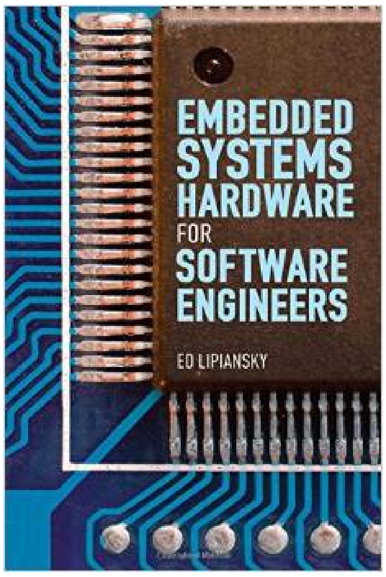


Figure 1: Embedded Systems Hardware for Software Engineers.

Ed Lipianski's Book

Mishra K. (2014) *Advanced Chip Design - Practial Examples in Verilog.* Pub Amazon Martson Gate.

Multicore field-programmable SoC: Xilinx Zync Product Brief

Atmel, ARM-based Embedded MPU AT91SAM Datasheet

OSCI. *SystemC tutorials and whitepapers* . Download from OSCI www.accelera.org or copy from course web site.

Brian Bailey, Grant Martin. *ESL Models and Their Application: Electronic System Level Design.* Springer.

The Simple Art of SoC Design - Closing the gap between ESL and RTL Michael Keating (2011), Springer ISBN 978-1-4419-8586-6

Ghenassia, F. (2006). *Transaction-level modeling with SystemC: TLM concepts and applications for embedded systems* . Springer.

Eisner, C. & Fisman, D. (2006). *A practical introduction to PSL* . Springer (Series on Integrated Circuits and Systems) — not relevant this year.

Foster, H.D. & Krolnik, A.C. (2008). *Creating assertion-based IP* . Springer (Series on Integrated Circuits and Systems) — not relevant this year.

Grotker, T., Liao, S., Martin, G. & Swan, S. (2002). *System design with SystemC* . Springer. E-BOOK (PDF)

Wolf, W. (2002). *Modern VLSI design (System-on-chip design)* . Pearson Education. Online: LINK.

0.0.3 Example: A Cellphone.

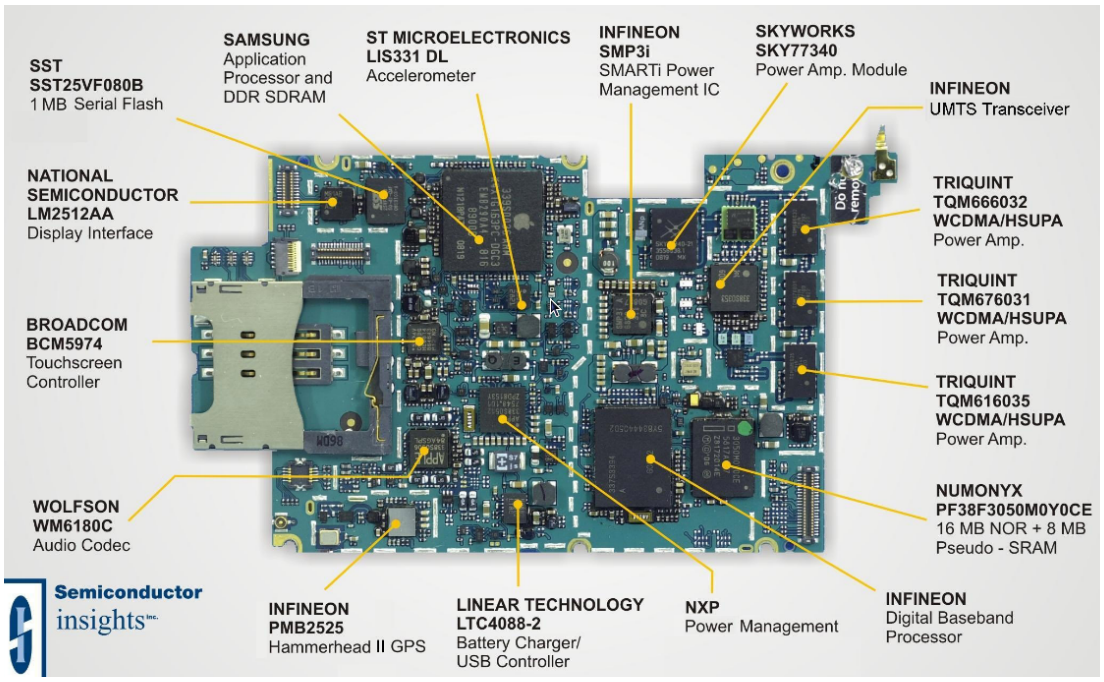


Figure 2: One of the two main PCBs of an Apple iPhone. Main SoC is top, left-centre.

A modern mobile phone contains eight or more radio tranceivers, counting the various cellphone standards, GPS, WiFi, near-field and Bluetooth. For the Apple iPhones, all use off-SoC mixers and some use on-SoC ADC/DAC. Another iPhone teardown link

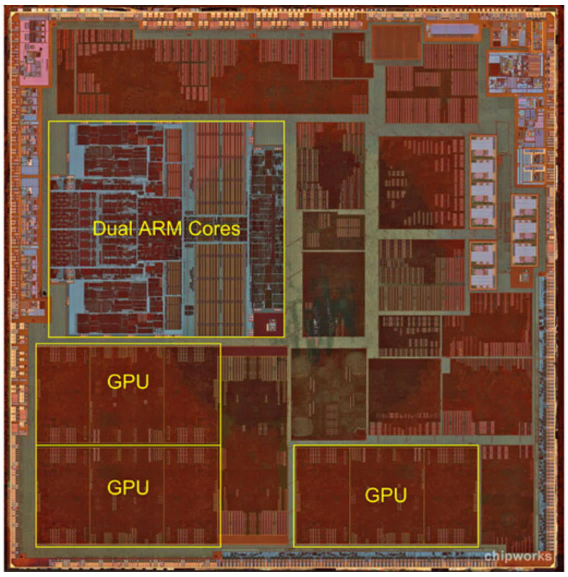


Figure 3: An Apple SoC - Two ARM and 3 GPU cores. Made by arch rival Samsung.

Further examples: iFixit Teardowns iPhone Dissected

Samsung GalaxyNumonyx Flash DatasheetCellphone physical components - bill of materials:

- Main SoC - Application Processor, Caches and **custom accelerators** for MP3 and MPEG and so on

- DRAM (dynamic random-access memory)
- Display (touch sensitive) + Keypad + Misc buttons
- Audio ringers and speakers, microphone(s) (noise cancelling),
- Infra-red IRDA port
- Multi-media codecs (A/V capture and replay in several formats)
- Radio Interfaces: GSM (three or four bands), BlueTooth, 802.11, GPS, Nearfield, ... plus antennas.
- Power Management: Battery Control, Processor Speed, on/off/flight modes.
- Front and Rear Cameras, Flash/Torch and ambient light sensor,
- Memory card slot,
- Physical connectors: USB, Power, Headset,
- Case, Battery and PCBs
- Java VM and Operating System.

0.0.4 Introduction: What is a SoC (1/2)?

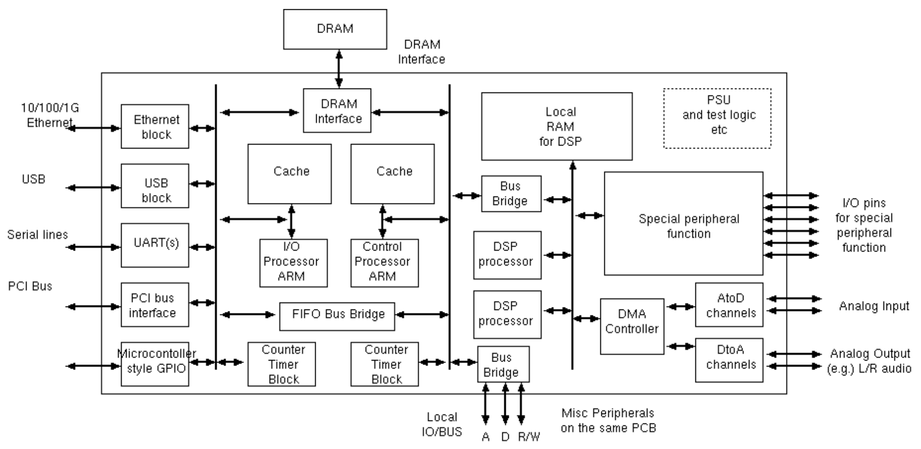


Figure 4: Block diagram of a multi-core 'platform' chip, used in a number of networking products. Programmable DSP processesors were used instead of hard-wired accelerators.

A System On A Chip: typically uses 70 to 140 mm^2 of silicon.

Multicore field-programmable SoC Xilinx Product Brief: PDFAtmel ARM-Based Platform Chip: PDF

0.0.5 Introduction: What is a SoC (2/2)?

A SoC is a complete system on a chip. A 'system' includes a microprocessor, memory and peripherals. The processor may be a custom or standard microprocessor, or it could be a specialised media processor for sound, modem or video applications. There may be multiple processors and also other generators of bus cycles, such as DMA controllers. DMA controllers can be arbitrarily complex and are only really distinguished from processors by their complete or partial lack of instruction fetching.

Processors are interconnected using a variety of mechanisms, including shared memories and message-passing hardware entities such as general on-chip networks and specialised channels and mailboxes.

SoCs are found in every consumer product, from modems, mobile phones, DVD players, televisions and iPods.

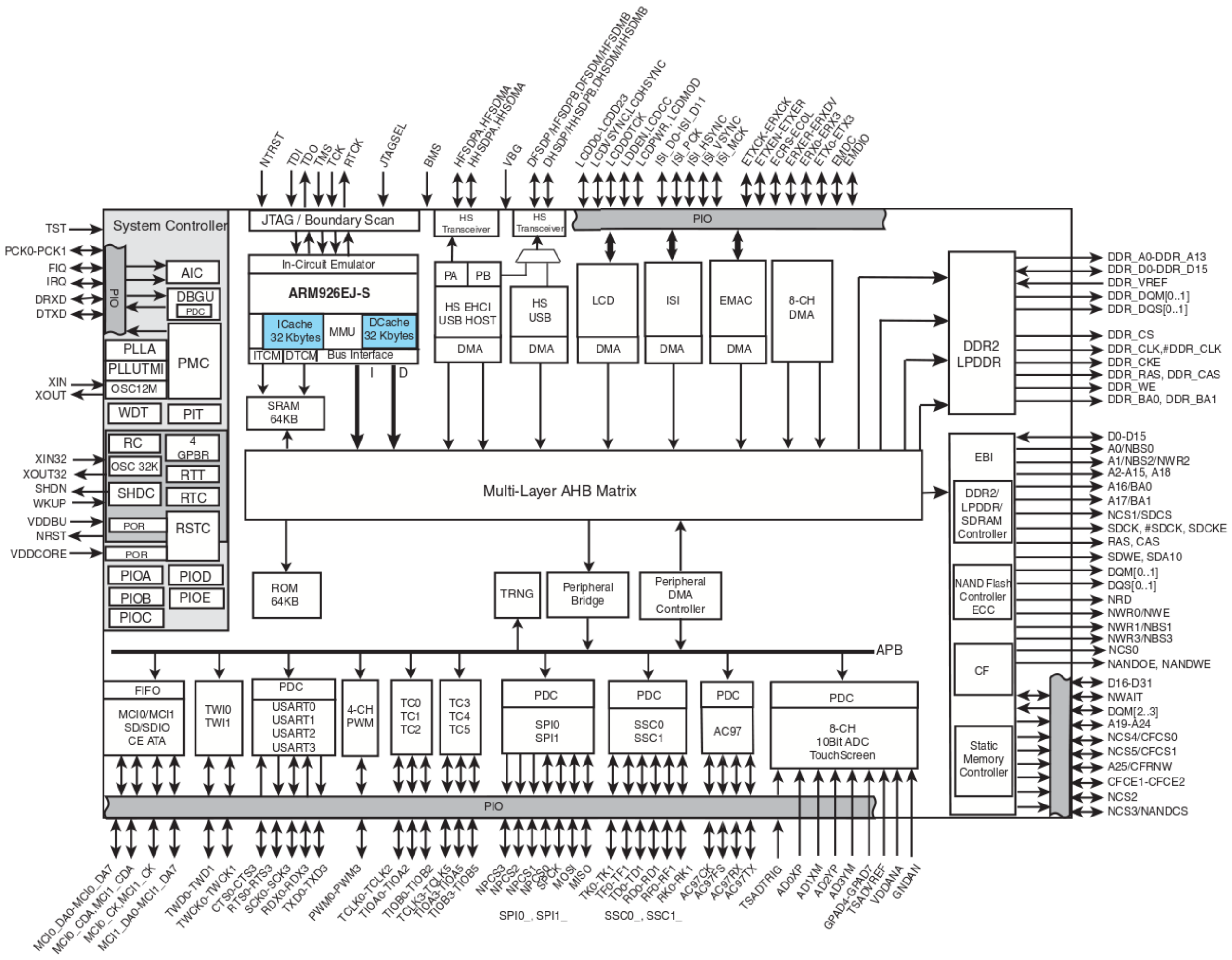


Figure 5: Platform Chip Example: Atmel SAM Series 9645.

0.1 Hardware Design Flow for an ASIC

ASIC: Application-Specific Integrated Circuit. The ASIC hardware design flow is divided by the **Structural RTL** level into:

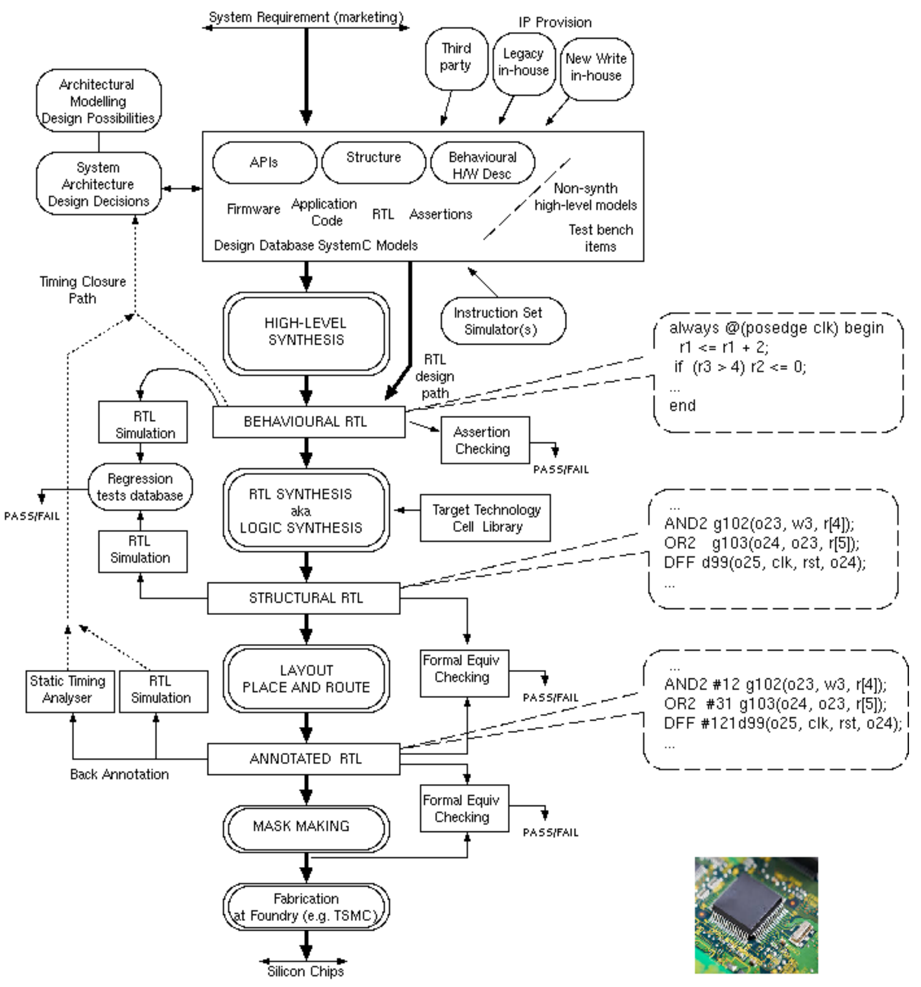
- **Front End**: specify, explore, design, capture, synthesise $\rightsquigarrow$ **Structural RTL**
- **Back End**: **Structural RTL** $\rightsquigarrow$ place, route, mask making, fabrication.

There is a companion software design flow that must mesh perfectly with the hardware if the final product is to work first time. This course will put as much emphasis on field-programmable parts (FPGAs) as on ASICs, since FPGA has now grown in importance.

Figure 6 shows a typical design and maufacturing flow that leads from design capture to ASIC fabrication. This might take six months. The FPGA equivalent can be done in half a day!

0.1.1 Front End

The design must be specified in terms of high-level requirements, such as function, throughput and power consumption.



```
always @(posedge clk) begin
  r1 <= r1 + 2;
  if (r3 > 4) r2 <= 0;
  ...
end
```

```
...
AND2 g102(o23, w3, r[4]);
OR2   g103(o24, o23, r[5]);
DFF d99(o25, clk, rst, o24);
...
```

```
...
AND2 #12 g102(o23, w3, r[4]);
OR2  #31 g103(o24, o23, r[5]);
DFF #121d99(o25, clk, rst, o24);
...
```

Figure 6: Design and Manufacturing Flow for SoC.

Design capture: it is transferred from the marketing person's mind, back of envelope or or wordprocessor document into machine-readable form.

Architectural exploration will try different combinations of processors, memories and bus structures to find an implementation with good power and load balancing. A loosely-timed high-level model is sufficient to compute the performance of an architecture.

Detailed design will select IP (interlectual property) providers for all of the functional blocks, or else they will exist from previous in-house designs and can be used without license fees, or else freshly written.

Logic synthesis will convert from behavioural RTL to structural RTL. Synthesis from formal high-level forms, including C,C++, SysML statecharts, formal specifications of interfaces and behaviour is beginning to be used.

Instruction set simulators (ISS) for embedded processors are needed: purchased from third parties such as ARM and MIPS, or as a by-product of custom processor design.

The interface specifications (register maps and other APIs) between components need to be stored: the IP-XACT format may be used.

High-level models that are never intended to be synthesisable and test bench components will also be coded, typically using SystemC.

0.1.2 Back End

After RTL synthesis using a target technology library, we have a structural netlist that has no gate delays. Place and route gives 2-D co-ordinates to each component, adds external I/O pads and puts wiring between the components. RTL annotated with actual implementation gate delays gives a precise power and performance model. If performance is not up to par, design changes are needed.

Fabrication of masks is commonly the most expensive single step (e.g. one million pounds), so must be correct first time.

Fabrication is performed in-house by certain large companies (e.g. Intel, Samsung) but most companies use foundaries (UMC, TSMC).

At all stages (front and back end), a library of standard tests will be run every night and any changes that cause a previously-passing test to fail (regressions) will be automatically reported to the project manager.

0.1.3 Levels of Modelling Abstraction

Our modelling system must support all stages of the design process, from design entry to fabrication. But we cannot model a complete SoC in detail and expect to simulate the booting of the O/S in reasonable time. We shall use a model called an **ESL Virtual Platform**. And where we are interested in the details of a specific module, we need to mix components using different levels of modelling abstraction within a single virtual platform.

Levels commonly used are:

- **Functional Modelling:** The 'output' from a simulation run is accurate.
- **Memory Accurate Modelling:** The contents and layout of memory is accurate.
- **Untimed TLM:** No time stamps recorded on transactions.
- **Loosely-timed TLM:** The number of transactions is accurate, but order may be wrong.
- **Approximately-timed TLM:** The number and order of transactions is accurate.
- **Cycle-Accurate Level Modelling:** The number of clock cycles consumed is accurate.
- **Event-Level Modelling:** The ordering of net changes within a clock cycle is accurate.

Other terms in use are:

- **Programmer View Accurate:** The contents of visible memory and registers is as per the real hardware, but timing may be inaccurate and other registers or combinational nets that are not designated as part of the 'programmers view' may not be modelled accurately.
- **Behavioural Modelling:** Using a threads package, or other library (e.g. SystemC), hand-crafted programs are written to model the behaviour of each component or subsystem. Major hardware items such as busses, caches or DRAM controllers may be neglected in such a model.

The Programmer's View is often abbreviated as 'PV' and if timing is added it is called 'PV+T'.

The Programmer's View contains only architecturally-significant registers such as those that the software programmer can manipulate with instructions. Other registers in a particular hardware implementation, such as pipeline stages and holding registers to overcome structural hazards, are not part of the PV.

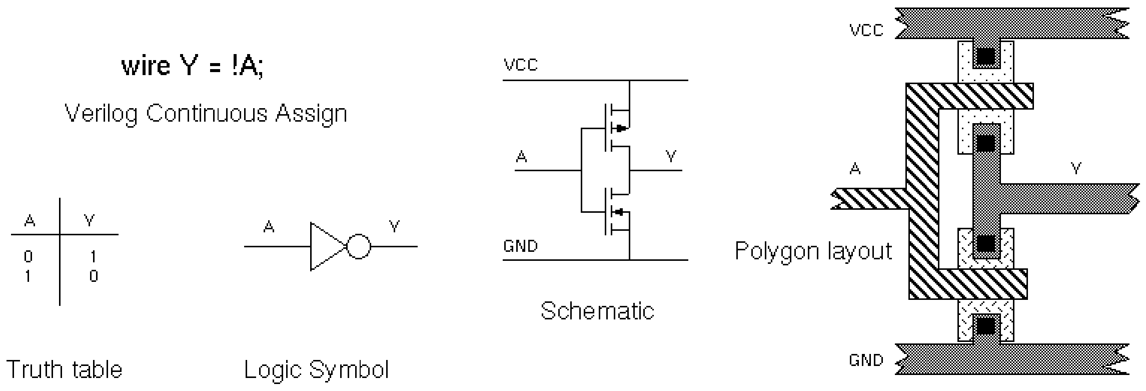


Figure 7: An inverter viewed at various levels of abstraction.

KG 1 — Energy use in Digital Hardware.

Battery life is very important for convenience. The electricity bill is sometimes the biggest cost in a data centre [citation needed!]. Saving energy in computing is always a good idea. In this section we will examine energy and performance and energy saving techniques.

Energy in an electronic device gets used in several different ways: for a Mobile Phone we might see the following budget:

- **Screen Backlight**: 1 to 2 watts
- **RF Transmisions** : via the various transmit antennae: up to 4 watts
- **Sound and Vibrations**: through the speaker and little shaker motor 300 mW
- **Heat**: 'wasted' in the electronics: up to 5 watts

A mobile phone might have a 10 Wh capacity (36 kilo Joules).

1.1 Basic Physics

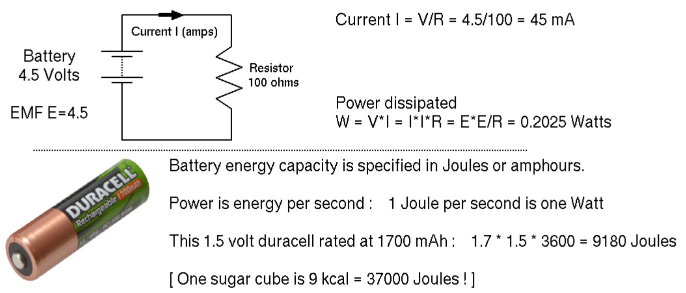

Current I (amps)
Battery
4.5 Volts
EMF E=4.5
Resistor
100 ohms
Current I = V/R = 4.5/100 = 45 mA
Power dissipated
W = V*I = I*I*R = E*E/R = 0.2025 Watts
Battery energy capacity is specified in Joules or amphours.
Power is energy per second : 1 Joule per second is one Watt
This 1.5 volt duracell rated at 1700 mAh : 1.7 * 1.5 * 3600 = 9180 Joules
[One sugar cube is 9 kcal = 37000 Joules !]
DURACELL
Rechargeable

Figure 1.1: Ohms Law, Power Law and Battery Capacity.

We need to know that power (watts) is voltage (volts) times current (amps). Power (watts) is also energy (joules) per unit time (second).

And a joule of energy is one colomb of charge dropping one volt in potential.

$P = V \times I = E \times f$

1.1.1 Energy of Computation (1)

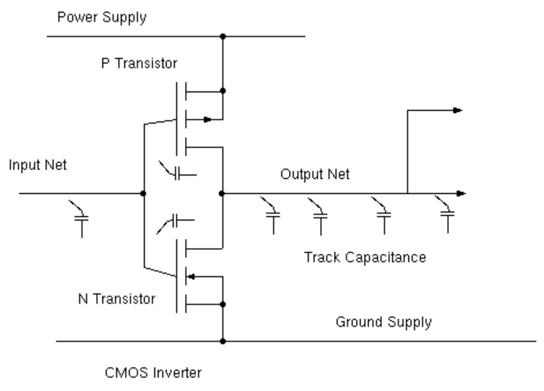
It is critical to understand where the energy goes in hardware in physical terms. All our models build up from this basis. Dynamic energy arising from stray capacitance turns out to be our main energy user today.

Gate current I = Static Current (leakage) + Dynamic Current.

Dynamic current = Short-circuit current + Dynamic charge current.

Early CMOS (VCC 5 volts): negligible static current, but today at VCC of 0.9 to 1.3 volts it can be 30 percent of consumption for some (high-leakage) designs.

The short-circuit power can generally be neglected. It is the energy wasted when both the P and N transistors are briefly conducting at once as the output. It is only significant when an input is left floating or slowly changing.



Dynamic charge current computation:

- All energy in a net is wasted each time output goes from one to zero.
- The energy in a capacitor was $E = CV^2/2$.
- The same amount of energy is wasted charging it up.
- Dominant capacitance is proportional to net length.
- Gate input and output capacitance also contribute to C.

The **toggle rate** for a net is the average frequency of a change in value, which is twice the activity ratio multipled by the clock frequency.

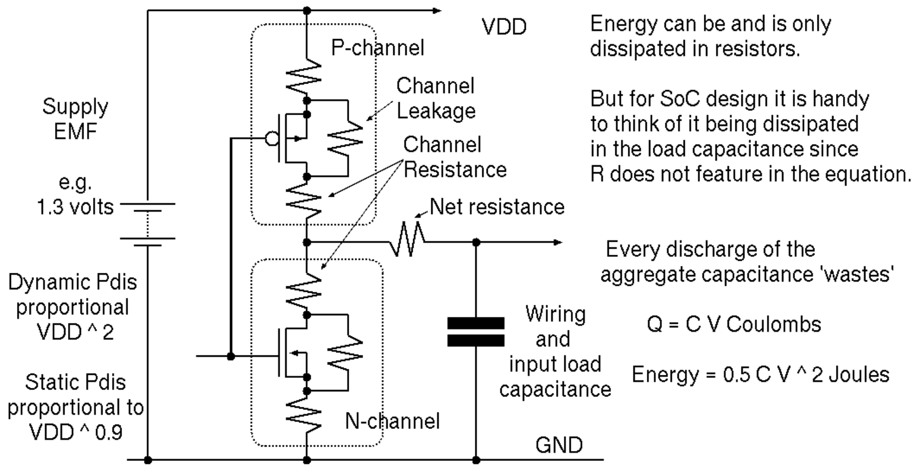

VDD
P-channel
Channel Leakage
Channel Resistance
Net resistance
Supply EMF
e.g. 1.3 volts
Dynamic Pdis proportional VDD ^ 2
Static Pdis proportional to VDD ^ 0.9
Wiring and input load capacitance
N-channel
GND
Energy can be and is only dissipated in resistors.
But for SoC design it is handy to think of it being dissipated in the load capacitance since R does not feature in the equation.
Every discharge of the aggregate capacitance 'wastes'
Q = C V Coulombs
Energy = 0.5 C V ^ 2 Joules

Figure 1.2: Dynamic energy dissipation mechanism.

Capacitors do not consume energy - they only store it temporarily. Only resistors dissipate energy in logic circuits, but their resistance does not feature in the energy use formula. The energy in the wiring capacitance is ‘wasted’ on each complete transistion cycle. If the clock frequency is f and a net has activity ratio α (the fraction of clock cycles it transitions from one to zero) then the energy used is

$$E = f * \alpha * C * V^2$$

As we increase voltage, dynamic power grows quadraticly. Static power grows a little better than (i.e. less than) linear since transistors may turn off more fully.

Note that although we divide by two as part of the standard formula for the energy in a capacitor, this quantity of energy is wasted both in the charging network on the zero-to-one transition and in the discharging network on the one-to-zero transition. So we can drop the half.

Note: static power consumption is static current multiplied by supply voltage (P=IV). Page 30 or so of this cell library has real-word examples: 90nm Cell LibrarySee also the power formula on the 7400A data sheet: 74LVC00A.pdfFurther details: Power Management in CPU Design.

1.1.2 Landauer Limit and Reversible Computation

There are theoretical limits on the energy an irreversible computation requires. Current technology is a long way off these in two resepects:

- we use too much energy representing and communicating bits and
- we use Vonn Neumann based computation which does not scale well.

Let's consider electrical computers:

- If we build a computer using a network of standard components (such as transistors), where the inteconnection pattern expresses our design intent, then the components must be at different spatial locations. The computer will have some physical volume.
- If we interconnect our components using electrical wires these will have capacitance, resistance and inductance that stop them behaving like ideal conductors. The smaller the volume, the less wire we need and the better the wires (and hence computer) will work.
- If we use transistors that need a swing of about 0.7 volts on their gates to switch them reliably between off and on, then our wires need to move through at least that much potential difference.

The capacitance of the wires turns out to be our main enemy. Given a prescribed minimum voltage swing, the energy used by switching a wire between logic levels can only be made smaller by reducing its area and hence capacitance. Hence smaller computers are better.

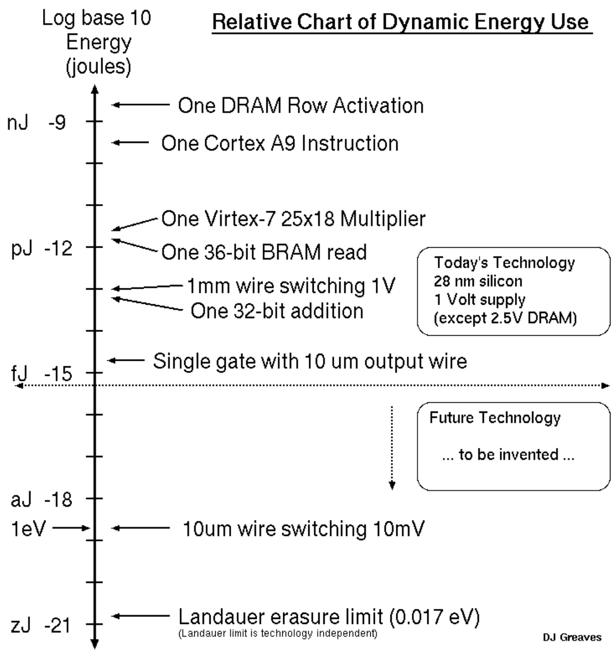


Figure 1.3: Relative chart of dynamic energy use.

The traditional approach of wasting the energy of each transition can be countered by returning the charge to the power supply (eg. Asynchrobatic logic for low-power VLSI design by David John Willingham, Univ Westminster, 2010) and reversible logic (eg. Toffoli logic) can get below the Landauer limit. Some standard algorithms such as encryption and lossless compression are reversible at the large - the trick is mooted to be to code in a way that does not delete intermediate results during the computation. Such techniques may be in wide use within a decade or perhaps two.

Figure 1.3 shows ballpark figures for dynamic energy use in today's (2016/2018) mainstream 28 nm silicon technology. We see that contemporary computers are about six orders of magnitude above the Landauer limit in terms of energy efficiency, so there is a lot of improvement still possible before we have to consider reversibility. (Log to base 18 months of one million is roughly 30 years, so 2050 is the reversible computing deadline.)

Note: Reversible computing not examinable for Part II.

1.1.3 Typical Numerical Values

Let's just run some numbers for a fictional chip made solidly of logic:

22 um geometry: Capacitance per micron 0.3 fF

Energy of discharge for a net of 1mm at VDD of 1 volt is 0.15 pJ

Simple gate area: 0.3 square um.

If we assume a sub-system of 1000 gates, its area is 1000 x 0.3 = 300 sq um

By Rent or otherwise, we have an average wiring length of 0.3 x sqrt(300) = 5.2 um.

Clocking at 200 MHz with an activity ratio of 0.1, dynamic power will be 1000 x 200e6 x 0.1 x 5.2 x 0.3e-15 = 31 uW.

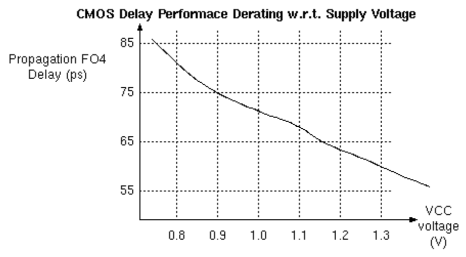
We will assume the inputs transition quickly enough for us to neglect short-circuit currents. Now we must add on the static power and the energy to drive the outputs from the sub-system:

Static power depends on leakage, but might be one fifth the dynamic power at 1 volt.

If we assume ten output nets with an activity of 0.1 and length 250 micron, their energy will be 10 x 200e6 x 0.1 x 250 x 0.3e-15 = 150 uW.

If we assume a 1 sq centimetre chip is made up of 1e8/300 = 3e5 such units, the chip power would be 3e5(150e-6 + 31e-6 + 6e-6)=56 watts. Clearly, we either need fluid cooling (water or ethanol heat pipe) or else the Dark Silicon approach. The activity ratio of 0.1 (toggle rate of 20 percent) reflects very-busy logic, such as an AES encoder. Most subsystems have lower activity ratios when in use. And the average activity ratio depends on how frequently used the block is, of course.

1.1.4 Gate Delay as a Function of Supply Voltage



Transistors have a gate threshold voltage around which they switch from off to on. This limits our lowest possible supply voltage. Above this, logic delay in CMOS is roughly inversely proportional to supply voltage. Accordingly, to operate faster, we need a higher supply voltage for a given load capacitance. CMOS Delay Versus Supply Voltage

$$\text{Gate Delay} \propto \frac{C * V}{(V - V_t)^2}$$

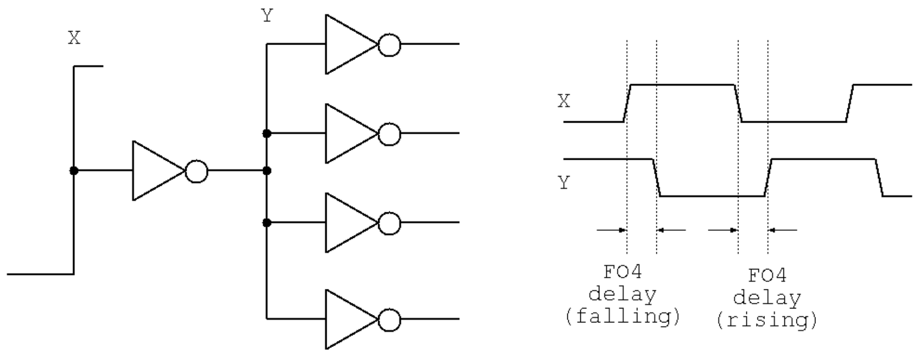


Figure 1.4: Fanout-4 delay measurement.

The **FO4 delay** is the delay through an inverter that is feeding four other nearby inverters (fan out of four). This is often quoted as a reference characteristic of a logic technology. The combinational delay of a particular design can also be expressed in a technology-independent way by quoting it in units of FO4 delay.

Here is a complete demo of simulating a CMOS inverter made from two MOSFETs using hspice. *Note: Spice simulation is not examinable for Part II CST.*

```
// spice-cmos-inverter-djg-demo.hsp
// Updated 2017 by DJ Greaves
// Based on demo by David Harris harrisd@leland.stanford.edu


///////////////////////////////////////////
// Set supply voltage



////////////////////////////////////////
// Declare global supply nets and connect them to a constant-voltage supply
.global Vdd Gnd
Vsupply Vdd Gnd DC 'VddVoltage'


///////////////////////////////////////////
// Set up the transistor geometry by defining lambda

.opt scale=0.35u  * Define lambda // This is half the minimum channel length.

// Set up some typical MOSFET parameters.
//http://www.seas.upenn.edu/~jan/spice/spice.models.html#mosis1.2um

.MODEL CMOSN NMOS LEVEL=3 PHI=0.600000 TOX=2.1200E-08 XJ=0.200000U
    +TPG=1 VTO=0.7860 DELTA=6.9670E-01 LD=1.6470E-07 KP=9.6379E-05
    +UO=591.7 THETA=8.1220E-02 RSH=8.5450E+01 GAMMA=0.5863
    +NSUB=2.7470E+16 NFS=1.98E+12 VMAX=1.7330E+05 ETA=4.3680E-02
    +KAPPA=1.3960E-01 CGDO=4.0241E-10 CGSO=4.0241E-10
    +CGBO=3.6144E-10 CJ=3.8541E-04 MJ=1.1854 CJSW=1.3940E-10
    +MJSW=0.125195 PB=0.800000


.MODEL CMOSP PMOS LEVEL=3 PHI=0.600000 TOX=2.1200E-08 XJ=0.200000U
    +TPG=-1 VTO=-0.9056 DELTA=1.5200E+00 LD=2.2000E-08 KP=2.9352E-05
    +UO=180.2 THETA=1.2480E-01 RSH=1.0470E+02 GAMMA=0.4863
    +NSUB=1.8900E+16 NFS=3.46E+12 VMAX=3.7320E+05 ETA=1.6410E-01
    +KAPPA=9.6940E+00 CGDO=5.3752E-11 CGSO=5.3752E-11
    +CGBO=3.3650E-10 CJ=4.8447E-04 MJ=0.5027 CJSW=1.6457E-10
    +MJSW=0.217168 PB=0.850000
```

```
/////////////////////////////////////////////
// Define the inverter, made of two mosfets as usual, using a subcircuit.

.subckt myinv In Out N=8 P=16 // Assumes 5 lambda of diffusion on the source/drain
m1 Out In Gnd Gnd CMOSN l=2 w=N
+ as='5*N' ad='5*N'
+ ps='N+10' pd='N+10'
m2 Out In Vdd Vdd CMOSP l=2 w=P
+ as='5*P' ad='5*P'
+ ps='P+10' pd='P+10'
.ends myinv


//////////////////////////////////////////////
// Top-level simulation netlist
//  One instance of my inverter and a load capacitor
x1 In Out  myinv          // Inverter
C1 Out Gnd 0.1pF          // Load capacitor

//////////////////////////////////////////////
// Stimulus: Create a waveform generator to drive In
// Use a  "Piecewise linear source"  PWL that takes a list of time/voltage pairs.

Vstim In Gnd PWL(0 0 1ns 0   1.05ns 'VddVoltage'   3ns VddVoltage    3.2ns 0)

//////////////////////////////////////////////
// Invoke transient simulation (that itself will first find a steady state)

.tran .01ns 6ns // Set the time step and total duration
.plot TRAN v(In) v(Out)
.end
// To get an X-windows plot, run the following interactive sequence of commands:
// $ ngspice spice-cmos-inverter-djg-demo.hsp
// ngspice 11 -> run
// ngspice 12 -> plot v(In) v(Out)

// end of file
```

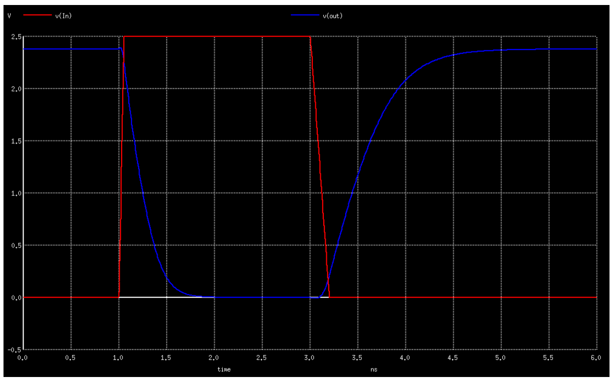


Figure 1.5: Plot when running from a VCC supply of 2.5 volts. Red is stimulus and blue is output.

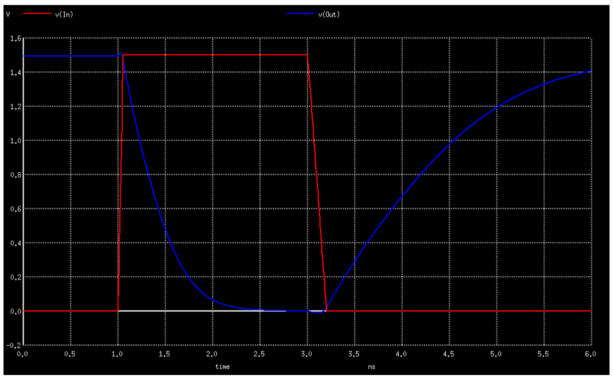
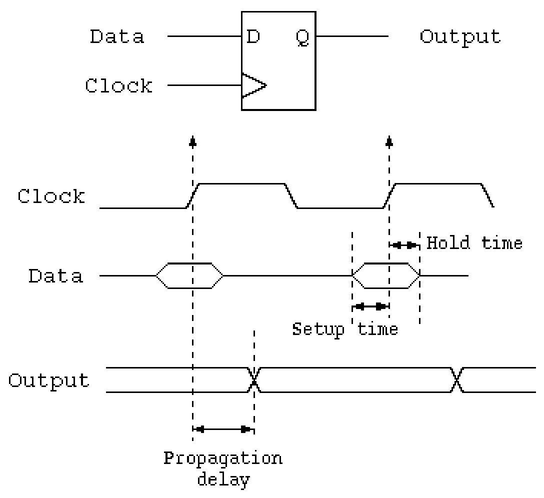


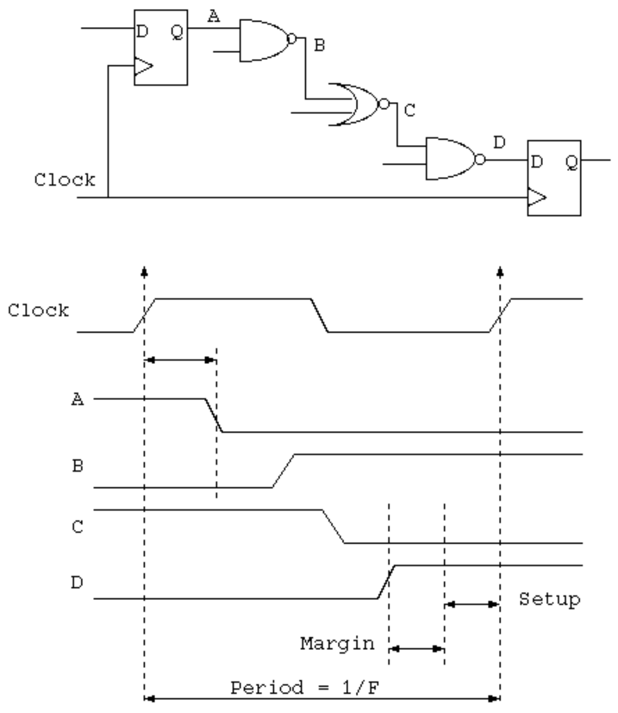
Figure 1.6: Plot when running at 1.5 volts.

The output load capacitor experiences a fairly typical exponential charge and discharge shape. The shape is not a true 1-exp(-t/CR) curve owing to some non-lineariy in the MOSFETs. But it is pretty close. If the FETs had equal on resistance at the two supply voltages, although the swing of the ouput in the two plots would be different, the delays before they cross the half-supply level would be identical. The difference arises owing to the on resistance being less when the gate voltage is less (i.e. when it is closer to the transistor threshold voltage).

1.1.5 Detailed Delay Model.

Now we have looked at the energy model, the other primary metric for a design can be considered: its delay performance that limits the clock rate.





The maximum clock frequency of a synchronous clock domain is set by its critical path. The longest path of combinational logic must have settled before the setup time of any flip-flop starts.

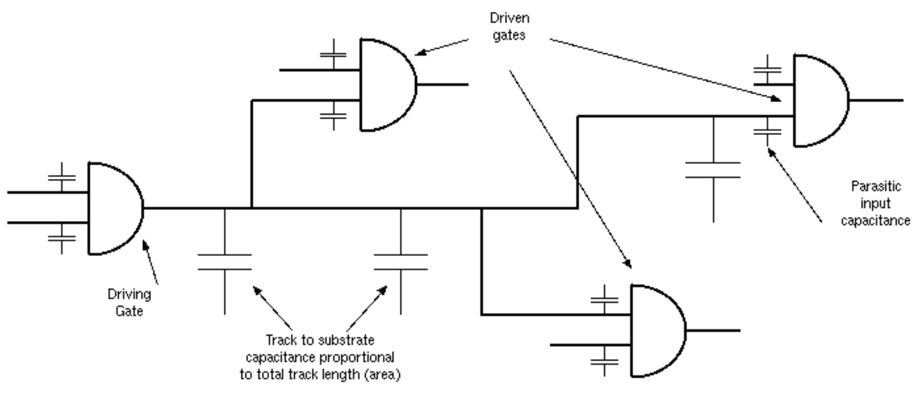

Driven gates
Parasitic input capacitance
Driving Gate
Track to substrate capacitance proportional to total track length (area)

Figure 1.7: Logic net with tracking and input load capacitances illustrated.

Both the power consumption and effective delay of a gate driving a net depend mainly on the length of the net driven.

The delay is modelled with this formula:

$$\text{device delay} = (\text{intrinsic delay}) + (\text{output load} \times \text{derating factor}).$$

The track-dependent output loading is a library constant times the track area. The load-dependent part is the sum of the input loads of all of the devices being fed. For short, non-clock nets (less than 0.1 wavelength), we just include propagation delay in the gate derating and assume the signal arrives at all points simultaneously.

Precise track lengths are only known after place and routing (Figure 6). Pre-layout and pre-synthesis we can predict net lengths from Rent's Rule and RTL-level heuristics.

Figure 1.7 shows a typical net, driven by a single source. To change the voltage on the net, the source must overcome the stray capacitance and input loads. The fanout of a gate is the number of devices that its output feeds. The term *fanout* is also sometimes used for the maximum number of inputs to other gates a given gate is allowed to feed, and forms part of the design rules for the technology.

The speed of the output stage of a gate, in terms of its propagation delay, decreases with output load. Normally, the dominant aspect of output load is capacitance, and this is the sum of:

- the capacitance proportional to the area of the output conductor,
- the sum of the input capacitances of the devices fed.

To estimate the delay from the input to a gate, through the internal electronics of a gate, through its output structure and down the conductor to the input of the next gate, we must add three things:

- the internal delay of the gate, termed the intrinsic delay
- the reduction in speed of the output stage, owing to the fanout/loading, termed the derating delay,
- the propagation delay down the conductor.

The propagation delay down a conductor obeys standard transmission line formula and depends on the distributed capacitance, inductance and resistance of the conductor material and adjacent insulators. For circuit board traces, resistance can be neglected and the delay is just the *speed of light* in the circuit board material: about 7 inches per nanosecond, or 200 metres per microsecond. On the other hand, for shorter nets on chip, less than one tenth a wavelength long, we commonly assume the signal arrives at all destinations at once and model the propagation delay as an additional inertial component of the driving gate and include this via the gate derating.

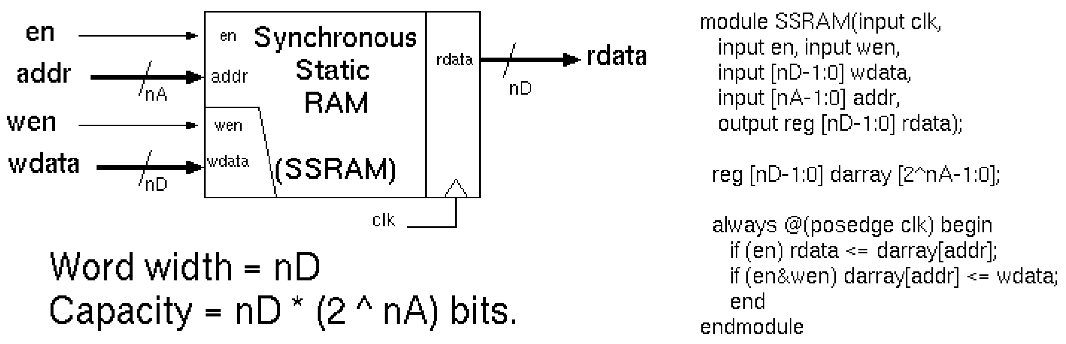
1.2 Memory Power and Performance

As well as logic gates, our main users of power will be SRAM, DRAM and I/O pads.

1.2.1 RAM - On-chip Static Random-Access Memory (Static RAM).

RAMs vary in their size and number of ports. Single-ported SRAM is the most important and most simple resource. It connects to a bus as an addressable target. It is also used inside caches for tags and data. Today's SoC designs have more than fifty percent of their silicon area devoted to SRAM for various purposes.

The illustrated RAM has a one clock cycle read latency. When a write occurs, the old value at the location is still read out, which is commonly a useful feature.



```
module SSRAM(input clk,
   input en, input wen,
   input [nD-1:0] wdata,
   input [nA-1:0] addr,
   output reg [nD-1:0] rdata);

  reg [nD-1:0] darray [2^nA-1:0];

  always @(posedge clk) begin
    if (en) rdata <= darray[addr];
    if (en&wen) darray[addr] <= wdata;
    end
endmodule
```

Figure 1.8: Synchronous static RAM with single port: logic symbol and internal RTL model.

The 'en' input signal is not striclty needed since the RAM could deliver read data on all cycles. However, this wastes power, so it would be better to hold the address input stable when not needing to read the RAM.

Owing to RAM fabrication overheads, RAMs below a few hundred bits should typically be implemented as register files made of flip-flops. But larger RAMs have better density and power consumption than arrays of flip-flops. Commonly, synchronous RAMs are used, requiring one clock cycle to read at any address. The same address can be written with fresh data during the same clock cycle, if desired.

RAMs for SoCs were normally supplied by companies such as Virage and Artizan (but these are now part of larger companies). A 'RAM compiler' tool is run for each RAM in the SoC. It reads in the user's size, shape, access time and port definitions and creates a suite of models, including the physical data to be sent to the foundry.

High-density RAM (e.g. for L2 caches) may clock at half the main system clock rate and/or might need error correction logic to meet the system-wide reliability goal.

RAM consumes static and dynamic energy. The ESL section of these notes gives high-level modelling figures that include about 10 pJ per read or write operation and a leakage of 82 nW per bit.

Additional notes:

On-chip SRAM needs a test mechanism. Various approaches:

- Can test with software running on embedded processor.
- Can have a special test mode, where address and data lines become directly controllable (JTAG or otherwise).
- Can use a built-in hardware self test (BIST) wrapper that implements 0/F/5/A and walking ones typical tests.

Larger memories and specialised memories are normally off-chip for various reasons:

- Large area: would not be cost-effective on-chip,
- Specialised: proprietary or dense VLSI technology cannot be made on chip,
- Specialised: non-volatile process (such as FLASH)
- Commodity parts: economies of scale (ZBT SRAM, DRAM, FLASH)

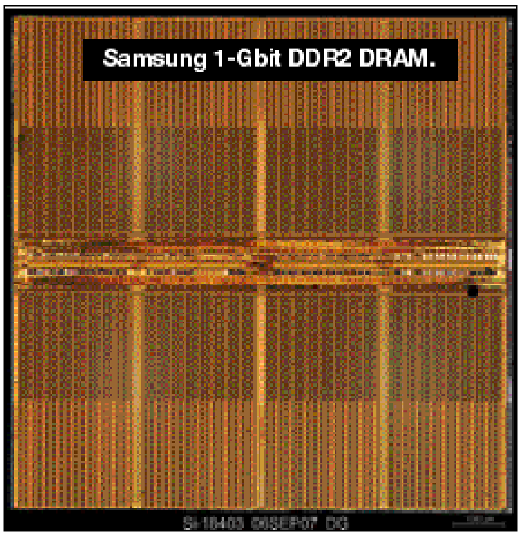
But in the last five years DRAM and FLASH have found their way onto the main SoC as maturing technology shifts the economic sweet spot.

1.2.2 Dynamic RAM : DRAM

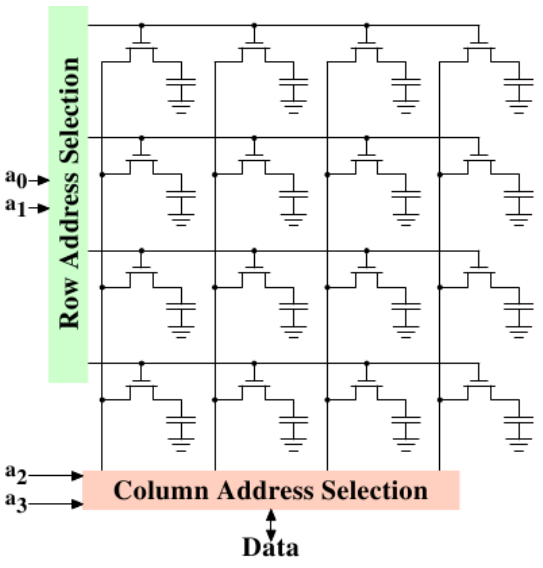
Figure 1.9: DRAM single-in-line memory module (SIMM).

DRAMs for use in PCs are mounted on SIMMS or DIMMS, but for embedded applications, often are just soldered to the main PCB. Normally one DRAM chip (or pair of chips to make D=32) is shared over many sub-systems in, say, a mobile phone. SoC DRAM compatibility might be a generation behind workstation DRAM: e.g. using DDR2 instead of DDR3 Also, the most recent SoCs embed some DRAM on the main die or flip-chip bond it right on top of the die in the same package.

Modern DRAM chip with 8 internal memory banks.



m



These are the pin connections of a typical DIMM from 2010:

Clk+/-	Clock (200MHz)	wq[7:0]	Write lane qualifiers
Ras-	Row address strobe	ds[7:0]	Data strobes
Cas-	Column address strobe	dm[7:0]	Data masks
We-	Write enable	cs-	Chip select
dq[63:0]	Data in/out	addr[15:0]	Address input
reset	Power on reset	bs[2:0]	Bank select
		spd[3:0]	Serial presence detect

DRAM performance is often quoted in MT/s which is mega-transfers per second. Our DIMM example has a 200 MHz clock and hence 400 MT/s. This is low performance by today's standards: 64 bits times 400 MHz gives 25.6 Gb/s peak (4 GB/sec). The capacity is a 1 Gbyte DIMM made of 8 chips.

The latest (Jan 2018) DDR4 memories operate at 4332 MT/sec. Each transfer caries a word the width of the data bus (e.g. 16 bits) and transfers are performed on both edges of a clock. This clock would be at 2.166 GHz. But that is the burst transfer rate. To manage a burst, DRAM timings of 19-21-21 are used, which is the number of clock cycles to send the row address, the column address and for writeback, respectively.

In the worst case, if the DRAM is currently 'open' on the wrong row, 61 clock cycles will be needed to change to the new location. Roughly the same number of clock cycles again will be used in pipeline stages through the various memory hierarchy levels of the controlling device.

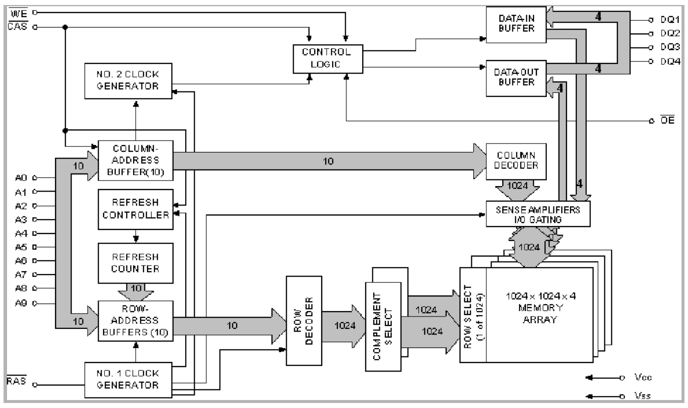
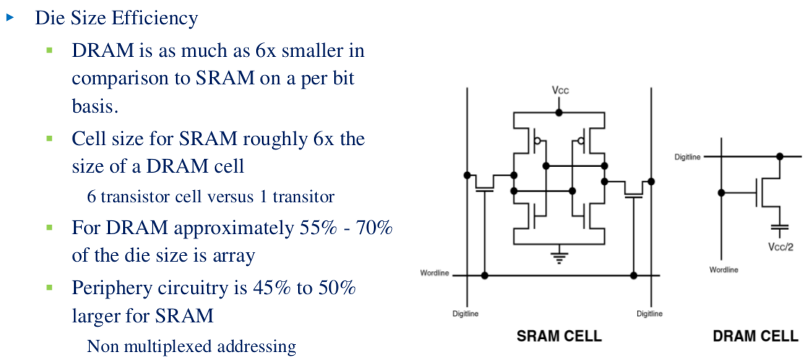


Figure 1.10: Single-bank DRAM Chip Internal Block Diagram.

This DRAM has four data I/O pins and four internal planes, so no bank select bits. (Modern, larger capacity DRAMs have multiple such structures on their die and hence additional bank select inputs select which one is addressed.)



- Die Size Efficiency
  - DRAM is as much as 6x smaller in comparison to SRAM on a per bit basis.
  - Cell size for SRAM roughly 6x the size of a DRAM cell
    - 6 transistor cell versus 1 transitor
  - For DRAM approximately 55% - 70% of the die size is array
  - Periphery circuitry is 45% to 50% larger for SRAM
    - Non multiplexed addressing

Vcc
Wordline
Digitline
Digitline
SRAM CELL
Digitline
Vcc/2
Wordline
DRAM CELL

Figure 1.11: SRAM cell complexity versus DRAM cell

Dynamic RAM keeps data in capacitors. The data will stay there reliably for up to four milliseconds and hence every location must be read out and written back (refreshed) within this period. The data does not need to leave the chip for refresh, just transferred to the edge of its array and then written back again. Hence a whole row of each array is refreshed as a single operation.

DRAM is not normally put on the main SoC chip(s) owing to its specialist manufacturing steps and large area needs. Instead a standard part is put down and wired up. (DRAM is traded as a commodity like corn and gold.)

A row address is first sent to a bank in the DRAM and then one has random access to the columns of that row using different column addresses. The DRAM cells internally have destructive read out because the capacitors get discharged into the row wires when accessed. Therefore, whenever finished with a row, the bank containing it goes busy while it writes back the data and gets ready for the next operation (charing row wires to mid-way voltage etc.).

DRAM is slow to access and certainly not 'random access' compared with on-chip RAM. A modern PC might take 100 to 300 clock cycles to access a random part of DRAM, but the ratio may not be as severe in embedded systems with lower system clocks. Nonetheless, we typically put a cache on the SoC as part of the memory controller. The controller may embody error detection or correction logic using additional bit lanes in the DRAM.

The cache will access the DRAM in localised bursts, saving or filling a cache line, and hence we arrange for cache lines to lie within DRAM rows.

The controller may keep multiple banks open at once to exploit tempro-spatial access locality.

DRAM controller is typically coupled with a cache or at least a write buffer.

DRAM: high latency and write-back overheads imply me must select a bank closing policy. The best controllers will lookahead in a pool of pending requests to assist decisions on when to do write back (aka close or deactivate). It is normal to prioritise reads over writes, but overtaking must be avoided or else reads can be served from the write queue. But a new request falling in a previously open line can arrive just after we closed it! It is best if clients can tolerate responses out-of-order (hence use bus/NoC structure that supports this).

Controller must

- set up DRAM control register programming,
- set clock frequency and calibrate delay lines,
- implement specific RAS-to-CAS latencies and many other timing details,
- and ensure refresh happens.

Controller often contains a tiny CPU to interrogate serial device data. DRAM refresh overhead has minimal impact on bus throughput. For example, if 512 refresh cycles are needed in 4 ms and the cycle rate is 200E6 the overhead is 0.1 percent.

Another design consideration is how the system address bus is mapped to the various row, bank and column physical bits. This will alter the effect of memory layout on performance. Hardware is normally programmable in terms of physical address interleave. An example is, starting with most significant in physical address space: bank, row, col, burst offset, byte lane. Byte lane and burst offset are always at the bottom and col is kept lower than row, but having bank lower than col gives interleaving of accesses to open pages which is sensible when the system workload has a lot of localised activity to one large area, whereas having bank higher makes sense when the system has various concurrent active hot spots, such as typical with heap, stack, code and static segments. etc..

Here we are using the definition of a bank as a region where only one row is active within one bank. Multiple banks may be within the same channel (which always occurs for banks within one DRAM chip) or arranged over separate channels were a channel is defined as a data bus (or DRAM controller backside port). Multiple channles gives higher data throughput owing to spatial diversity.

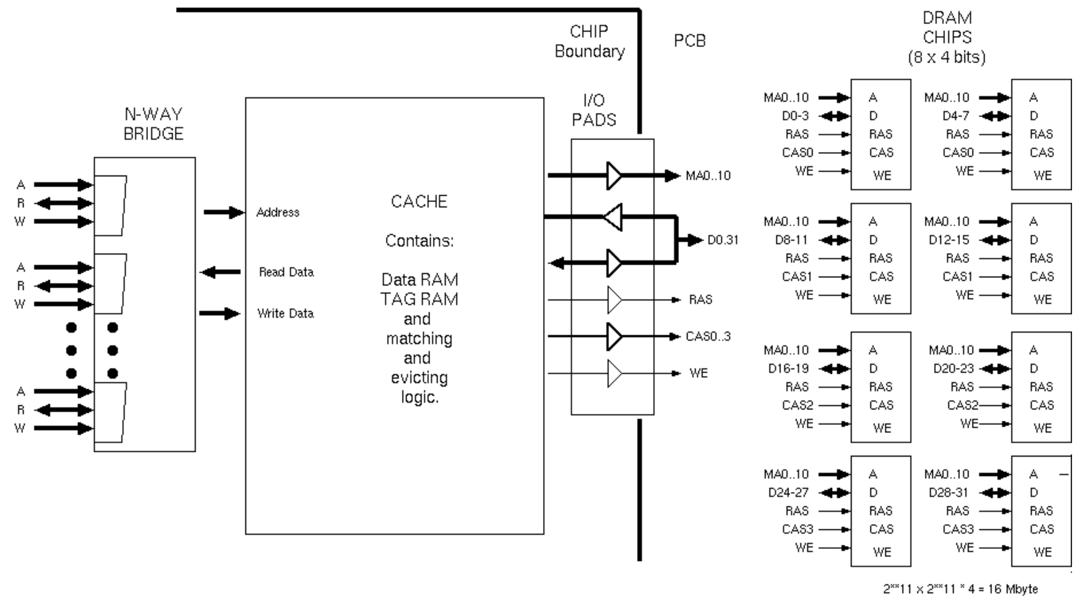


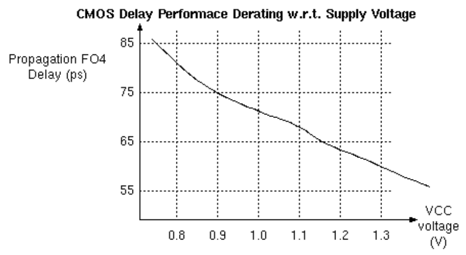
Figure 1.12: Typical structure of a small DRAM subsystem.

Figure 1.12 shows a 32-bit DRAM subsystem. Four CAS wires are used so that writes to individual byte lanes are possible. For large DRAM arrays, need also to use multiple RAS lines to save power by not sending RAS to un-needed destinations. [*Detailed wiring details non-examinable*]

1.3 The Voltage and Frequency Relationship

Looking at the derating graph for the standard cell libraries, we see that in the operating region, the frequency/voltage curve is roughly linear. CMOS delay is inversely proportional to supply voltage. A technique that exploits such curves is DVFS – dynamic voltage with frequency scaling.

Logic with higher-speed capabilities is smaller which means it consumes greater leakage current which is being wasted while we are halted. Also leakage energy is proportional to supply voltage (or perhaps sublinear with exponent 0.9ish : as we raise voltage, transistors are indeed turned off more, but P=IV is still increasing).



If we vary the voltage to a region dynamically, while keeping f constant, a higher supply voltage uses more power (square law) but would allow a higher f.
Let's only raise VCC when we ramp up f: classical DVFS.

For fixed task size, energy use is proportional to V squared, so the DVFS ideal method (for predictable, real-time tasks in low-leakage technology):

1. Adjust clock f for just-in-time completion (e.g. in time to decode the next frame of a video),
2. then adjust VCC so logic just works.

Hence, we aim to clock at the lowest suffcient voltage and clock frequency and for a as high as possible and minimal halt cycles.

In general, ramping voltage up linearly with clock frequency (f) results in dynamic power consumption with a cubic dependence on f. But work may be bursty, so DVFS is applied (e.g. by a laptop governor).

DVFS obtains peak performance under heavy loads, yet avoid cubic overhead when idle. We adjust VCC so that, at all times, the logic just works. However, we need to keep close track of whether we are meeting real-time and timing closure deadlines.

In a server farm processing blade we may be thermally limited, so DVFS will be throttled back by rack-level governors or Intel's RAPL.

Minor caveats (non-examinable):

Combinational logic cannot be clock gated (e.g. PAL and PLA). For large combinational blocks: can dip power supply to reduce static current when block is completely idle (detect with XORs).

So a typical SoC uses not only many dynamic clock gated islands, but also some sub-continents with automatic frequency and voltage variation. Power isolation originally used on a longer and larger scale (complete continents) but now a lot of power islands are being used.

It is possible to locally and quickly adjust supply voltage with a series transistor - but wasteful compared with an off-chip switched-mode regulator.

An off-chip power supply can be efficiently adjusted, but limited to only a few voltage islands and tens of milliseconds inertia.

1.3.1 DVFS Example

Example: core area 64 mm^2; average net length 0.1 mm; 400K gates/mm^2, $a = 0.25$.

Net capacitance = 0.1 mm × 1 fF/mm × 400K × 64 mm^2 = 2.5 nF.

Supply Voltage (V)	Clock Freq (MHz)	Static Power (mW)	Dynamic Power (mW)	Total Power (mW)
0.8	100	40	24	64
1.35	100	67	68	135
1.35	200	67	136	204
1.8	100	90	121	211
1.8	200	90	243	333
1.8	400	90	486	576

The table shows example power consumption for a circuit when clocked at different frequencies and voltages. The important thing to ensure is that the supply voltage must be sufficient for the clock frequency in use: too low a voltage means that signals do not arrive at D-type inputs in time to meet set up times.

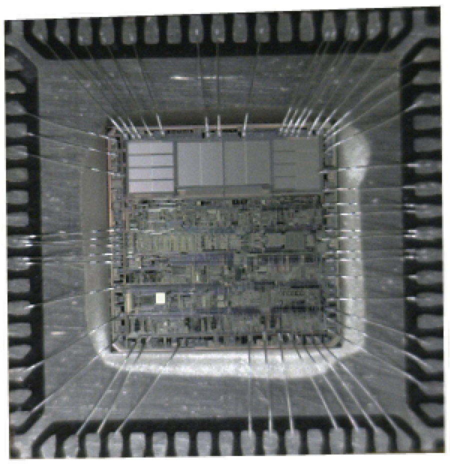
Power consumption versus frequency is worse than linear: it goes with a power law.

In the past, chips were often **core-bound** or **pad-bound**. Pad-bound meant that the chip had too many I/O signals for its core logic area: the number of I/O's puts a lower bound on the perimeter of the chip. Today's VLSI technology allows I/O pads in the middle of the chip and designs are commonly **power-bound**.

1.3.2 90 Nanometer Gate Length.

The mainstream VLSI technology in the period 2004-2008 was 90 nm. This had low leakage and very high wafer yields. Now the industry is using 22 nanometer and smaller. Parameters from a 90 nanometer standard cell library:

Parameter	Value	Unit
Drawn Gate Length	0.08	μm
Metal Layers	6 to 9	layers
Max Gate Density	400K	gates/mm^2
Finest Track Width	0.25	μm
Finest Track Spacing	0.25	μm
Tracking Capacitance	1	fF/mm
Core Supply Voltage	0.9 to 1.4	V
FO4 Delay	51	ps
Leakage current		nA/gate



Typical processor core: 200k gates + 4 RAMs: one square millimeter. Typical SoC chip area is 50-100 mm^2 $\rightsquigarrow$ 20-40 million gates (semi-custom/standard cell). Actual gate and transistor counts are higher owing to full-custom blocks (RAMs mainly).

- 2007: Dual-core Intel Itanium2: 1.6 billion transistors (90 nm).
- 2010: 8-core Intel Nehalem: 2.3 billion transistors (45 nm).
- 2010: Altera Stratix IV FPGA: 2.5 billion transistors (40 nm).
- 2015: Intel Latest CPU: circa 10 billion transistors (19 nm).

Moore's Law Transistor Count Dennard Scaling

Dimension Increase in Metal-Oxide-Semiconductor Memories and Transistors

Dennard's Rule stated that as transistors get smaller, their power density stays constant, so that the power use stays in proportion with area: both voltage and current scale (downward) with length. This meant that no new heat extraction technology was needed as VLSI capabilities improved. But once a supply voltage of 1 volt was reached, silicon CMOS cannot be run at any lower voltage without leakage (static power) greatly increasing.

The slide shows typical parameters from a 90 nanometer standard cell library. This figure refers to the width of the gate in the field effect transistors. The smaller this width, the faster the transistor can operate, but also it will consume more power as static leakage current. The 90 nm figure was the mainstream VLSI technology in the period 2004-2008, but then 40-45 nanometer technology was widely used with smaller 22 nm now mainstream.

Typical processor core: 200k gates + 4 RAMs: one square millimeter.

A typical SoC chip area is 50-100 mm^2 with 20-40 million gates. Actual gate and transistor count would be higher owing to custom blocks (RAMs mainly), that achieve a better denisty than standard cells.

Moore's Law has been tracked for the last three plus decades, but have we now reached the *Silicon End Point*? That is, can we no longer make things smaller (at the same cost)? Modern workstation processors have certainly demonstrated a departure from the previous trend of ever rising clock frequencies: instead they have several cores.

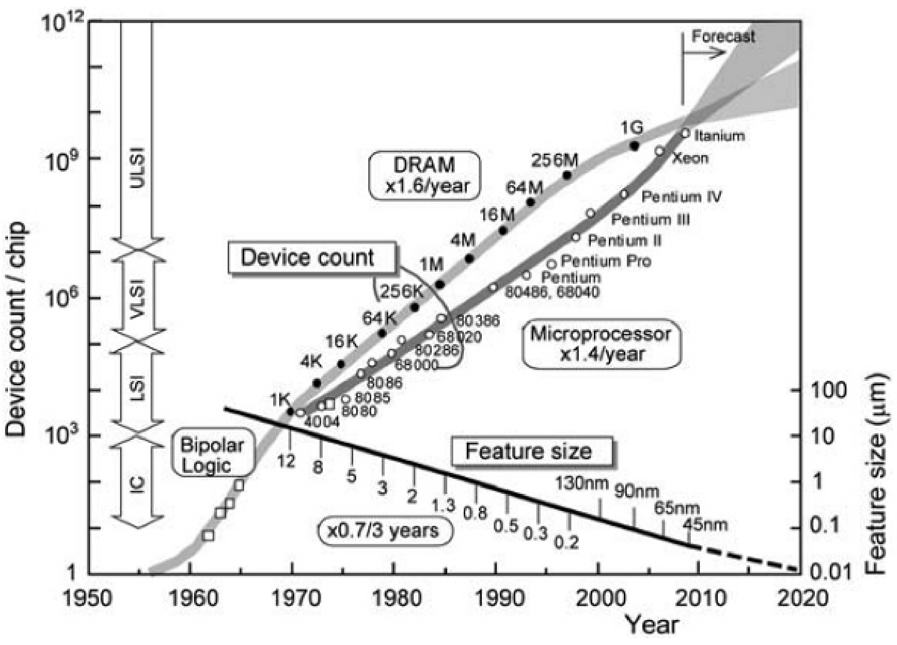


Figure 1.13: Technology Scaling Prediction.

The **Power Wall** is currently the limiting factor for practical VLSI. As Horowitz points out, the fixed threshold voltage of transistors means that supply voltages cannot be reduced further as we go to smaller and smaller geometries, hence the previous technology trajectory will change direction: Scaling, Power, and the Future of CMOS. The limiting factor for commercial products has become the cost of thermal management. We can put more-and-more transistors on our chip but we cannot use them all at once - hence **Dark Silicon**.

1.4 Further Power Saving Techniques

Turning off and slowing down are our friends when it comes to saving power.

We can save power by controlling power supplies and clock frequencies: Figure 1.17.

Frequency scaling means adjusting the clock frequency to a subsystem. The voltage will have to be scaled in tandem to ensure timing is met without wasting energy. Frequency adjustment can be instant if divider output is directly used. (But when PLLs with analog filters are used, there is inertia, e.g. 1 millisecond). Voltage adjustment also has inertia and there is design and implementation complexity supporting more than a few voltage regions.

	Clock Gating	**Supply Gating**	**DVFS**
Control:	automatic	various	software
Granularity:	register / FSM	larger blocks	macroscopic.
Clock Tree:	mostly free runs	turned off	slows down.
Response time:	instant	2 to 3 cycles	instant (or ms if PLL adjusted)
Proportionally vary voltage:	not possible	n/a	yes.

1.4.1 Save Power 2: Dynamic Clock Gating

Clock trees consume quite a lot of the power in an ASIC and considerable savings can be made by turning off the clocks to small regions. A region of logic is idle if all of the flip-flops are being loaded with their current contents, either through synchronous clock enables or just through the nature of the design. EDA

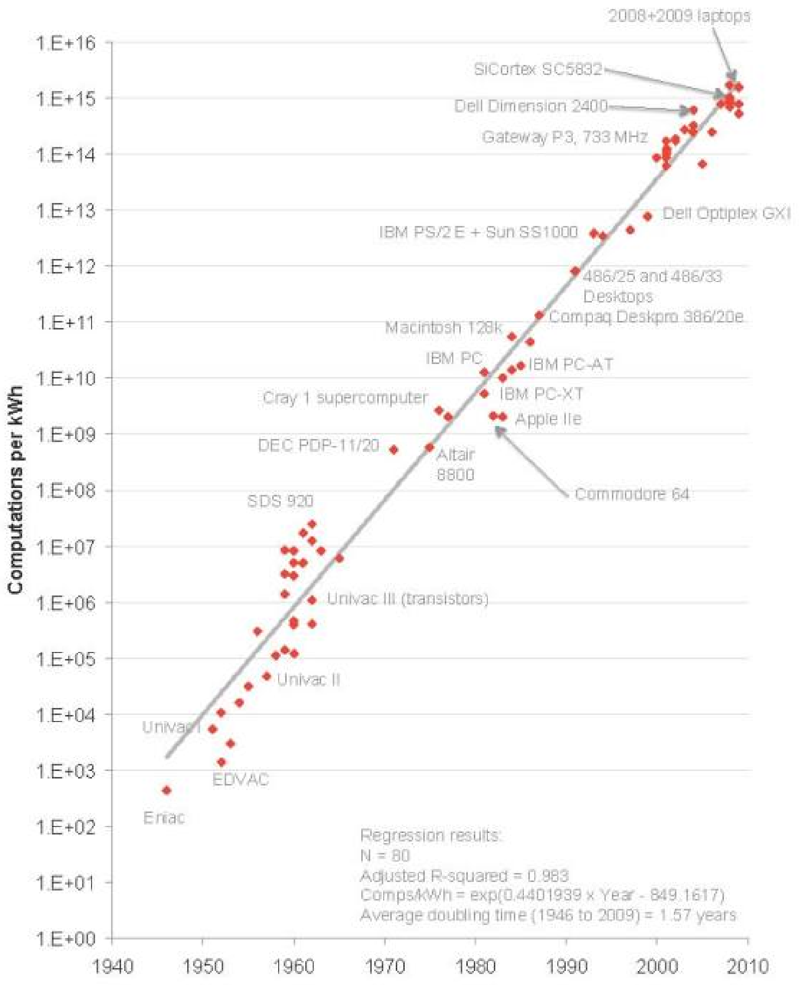


Figure 1.14: Computations per kWh. Divide by 3.6E6 for per Joule.

DESIGNLINE

Instead of using synchronous clock enables, current design practice is to use a clock gating insertion tool that gates the clock instead. One clock control logic gate serves a number of neighbouring flip-flops: state machine or broadside register.

Problem with AND gate: if CEN changes when clock is high: causes a glitch. Problem with OR gate: if CEN changes when clock is low: causes a glitch. Hence, care must be taken not to generate glitches on the clock as it is gated. Transparent latches in the clock enable signal prevent these glitches.

Care needed to match clock skew when crossing to/from non-gated domain: avoid *shoot-through* by building out the non-gated parts as well. Shoot-through occurs when a D-type is supposed to register its current D input value, but this has already changed to its new value before the clock signal arrives.

How to generate clock enable conditions ? One could have software control for complete blocks (additional control register flags, as per power gating). But today's designs automatically detect on a finer-grain basis. Synthesiser tools can automatically insert clock required conditions and insert the additional logic. Automatic tools compute 'clock needed' conditions. A clock is 'needed' if any register will change on a clock edge.

The following table shows the VDD power consumption of the E16G301 as a function of frequency and voltage with all 16 cores executing a heavy duty workload. The measurements were taken at room temperature without a heat sink and with 0 m/s airflow. The maximum operating frequency at each voltage level is specified next to the data point in the figure

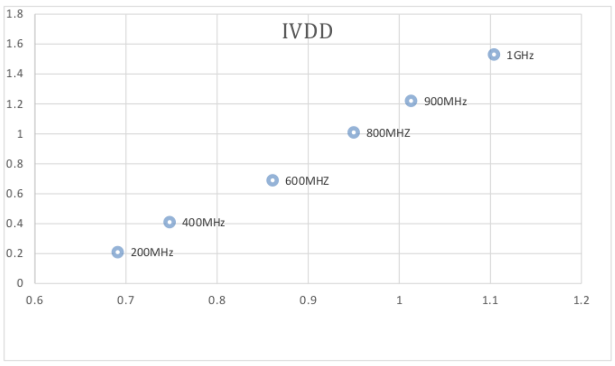


Figure 1.15: Epiphany 16 Core ‘Supercomputer’ Chip DVFS Points (a plot, not a table).

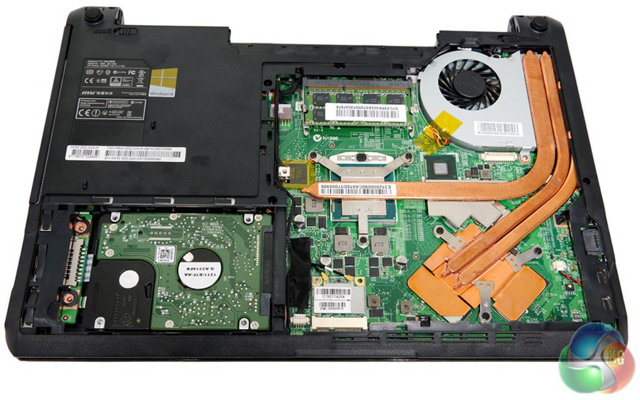


Figure 1.16: Thermal management heat pipes in a modern laptop.

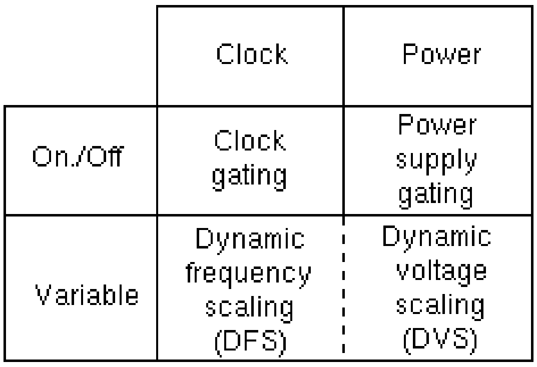
A lot of clock needed computation can get expensive, resulting in no net saving, but it can be effective if computed once at head of a pipeline.

If not a straightforward pipeline, need to be sure there are no ‘oscillating’ stages that retrigger themselves or an ‘earlier’ stage (add further runtime checks or else statically know their maximum settling time and use a counter). The maximum settling time, if it exists, is computed in terms of clock cycles using static analysis. Beyond the settling time, all registers will be being re-loaded with their current data on each clock cycle.

Beyond just turning off the clock or power to certain regions, we can consider further power saving techniques: dynamic frequency and voltage scaling.

1.4.2 Save Power 3: Dynamic Supply Gating

Increased tendency towards multi-product platform chips means large functional blocks on silicon may be off for complete product lifetime. The ‘dark silicon’ future scenario implies all chips must be mostly powered off. Battery powered devices will also use macro-scale block power down (e.g. the audio or video input and output subsystems).



	Clock	Power
On./Off	Clock gating	Power supply gating
Variable	Dynamic frequency scaling (DFS)	Dynamic voltage scaling (DVS)

Figure 1.17: Terminology and Overview of Power Saving Techniques.

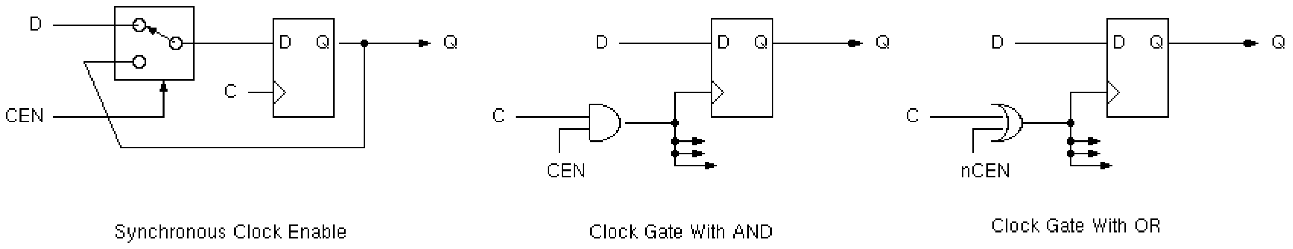
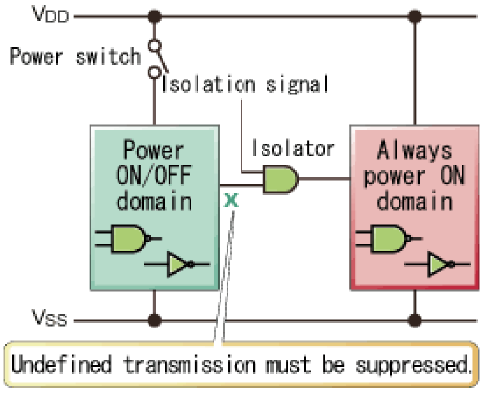
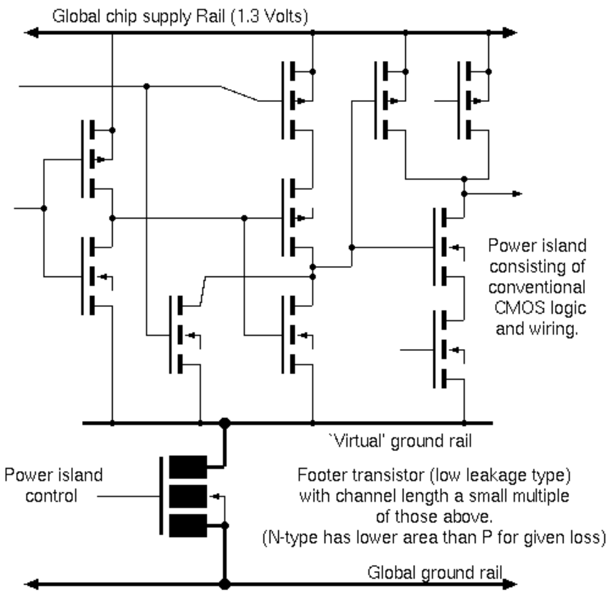


Figure 1.18: Clock enable using multiplexor, AND and OR gate.




Global chip supply Rail (1.3 Volts)
Power island consisting of conventional CMOS logic and wiring.
'Virtual' ground rail
Power island control
Footer transistor (low leakage type) with channel length a small multiple of those above.
(N-type has lower area than P for given loss)
Global ground rail

Dynamic power gating techniques typically require some sequencing: several clock cycles to power up/down a region and enable/disable isolation gates.

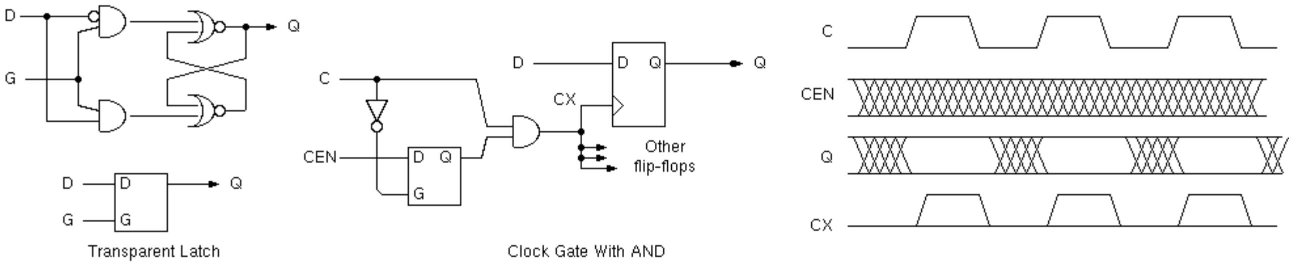


Figure 1.19: Illustrating a transparent latch and its use to suppress clock gating glitches.

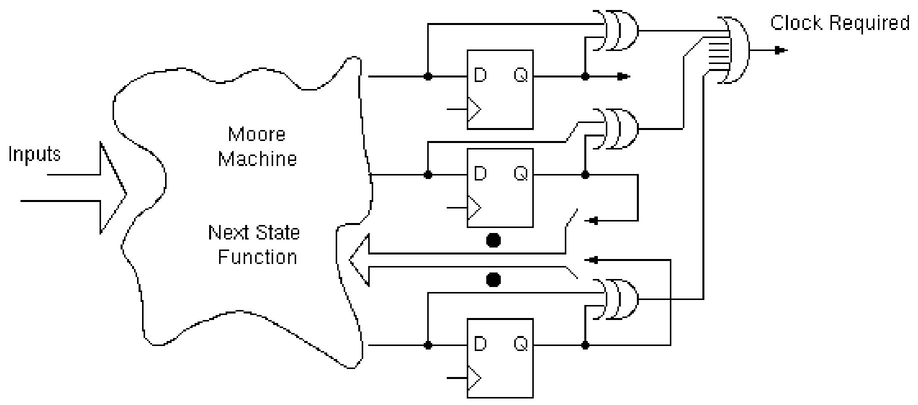


Figure 1.20: Using XOR gates to determine whether a clock edge would have any effect.

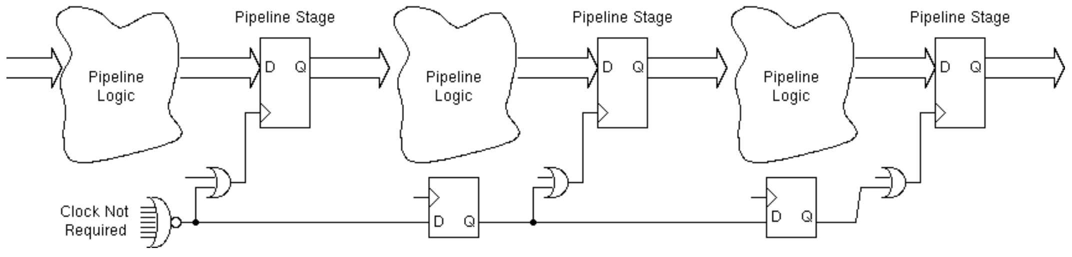


Figure 1.21: Clock needed computations forwarded down a pipeline.

Fujitsu Article: Design of low power consumption LSIs

Previously we looked at dynamic clock gating, but we can also turn off power supply to regions of a chip with fine or coarse gain, creating so-called power islands. We use power gating cells in series with supply rails. These are large, slow, low-leakage transistors. (Best to disconnect the ground supply since an N-channel transistor can be used which has smaller area for same resistance.)

Signal isolation and retention cells (t-latches) on nets that cross in and out of the region are needed. There is no register and RAM data retention in a block while the power is off. This technique is suitable at coarse grain for complete sub-systems of a chip that are not in use on a particular product or for quite a long time, such as a bluetooth tranceiver or audio input ADC. It can also be used on a fine grain with automated control similar to clock gating.

However, power gating requires some sequencing to activate the enables to the isolation cells in the correct order and hence several clock cycles or more are needed to power up/down a region. Additionally, gradual turn on over tens of milli-seconds avoids creating noise on the global power rails. Originally, power off/on was controlled by software or top-level input pads to the SoC. Today, dedicated microsequencer hardware might control a hundred power islands within a single subsystem.

A common practice is to power off a whole chip except for a one or two RAMs and register files. This was particularly common before FLASH memory was invented, when a small battery is/was used to retain contents using a lower supply (CMOS RAM data holding voltage). Today, most laptops, tablets and PCs have a second, tiny battery that maintains a small amount of running logic when the main power is off or battery removed. This runs the real-time clock (RTC).

Another technique that saves power is to half-turn-on a power gating transistor and thereby run an island at a lower voltage. This is not as efficient as adjusting standard switched-mode power supplies, since the half-turned on transistor will waste energy itself.

1.4.3 DVFS in Low Leakage Technology

Two potential strategies (Aesop: Tortoise v Hare):

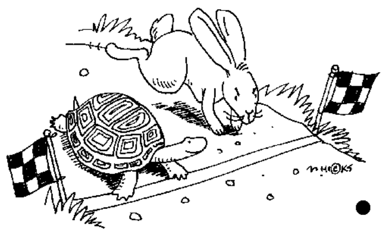


Figure 1.22: Aesop's Fable - Whose approach is better?

1. Compute quickly with halt(s), or
2. Compute more slowly and finish just in time.

To compute quickly and halt we need a higher frequency clock but consume the same number of active cycles. So the work-rate product, af, unchanged, so no power difference ? No. Running the same number of work cycles at a lower frequency requires a lower voltage and hence we save energy according to V^2.

But: current geometries only have a narrow operating voltage range Too low -> too much leakage. Too high -> tunnelling and wear and heat. So today, we operate at around one volt always and the trade off is just between high and low leakage technology - a static fab-time decision. There are further (unexaminable) prospects on the table, such as dynamic body bias ...

1.4.4 Static and Dynamic Power Tradeoff

But: for sub 45nm, voltage scaling is less viable and transistors with much higher static leakage current are commonly used: so can now be better to operate within the voltage/frequency band that works and then power off until next deadline.

```
# Trade off of Hare and Tortoise for increasingly leaky technology.
# For a hard-realtime computation we known the number of clock cycles needed but should we do them quickly
# and halt (Archilies) or slowly and finish just in time (Tortoise). For a higher leakage technology,
# switching off as much as possible tends to become preferable to running at low supply voltage.

# Unfold=1 is baseline design. Unfold=3 uses three times more silicon.
static_dynamic_tradeoff <- function(clock_freq, leakage, unfold, xx)
{
  op_count <- 1e5;
  execution_time = op_count / clock_freq / (unfold ^ 0.75);   // Model: Pollack-like unfold benefit.
  vdd <- 1 + 0.5 * (clock_freq/100e6);                        // Model: Higher supply needed for higher clk.
  static_power <- leakage * vdd ^ 0.9 * unfold * 0.4;         // Model: Leakage slightly sublinear.
  static_energy <- static_power * execution_time;             // Integrate static power
  dynamic_energy <- op_count * vdd ^ 2.0 / 0.5 * 1e-9;        // Use CV^2/2 for dynamic
}
```

For the 90nm technology, there was low static leakage and considerable scope for DVFS. With the smaller geometries performance can be traded off for greater leakage. Transistor dopant levels (and hence leakage) can be adjusted in regions or globally. We will want a low leakage, large slow transistor for power gating but may choose higher leakage transistors for logic since these will either be faster or can be run off a lower Vdd, hence reducing dynamic power.

The simple R plot illustrates the shift in operating frequency sweet spot (minimal total power) with higher leakage transistors. We considered leakage of 0.05 and 0.3 (arbitrary units). With low leakage it is best to compute slowly and finish just in time. With high leakage it is best to compute more quickly and then turn off for longer.

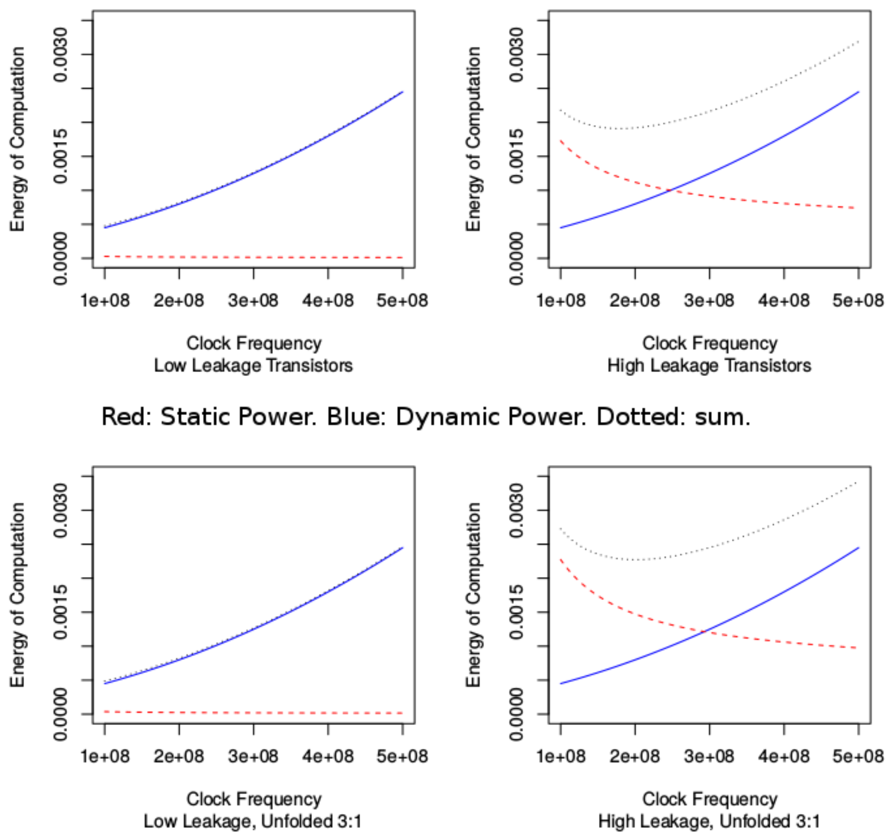


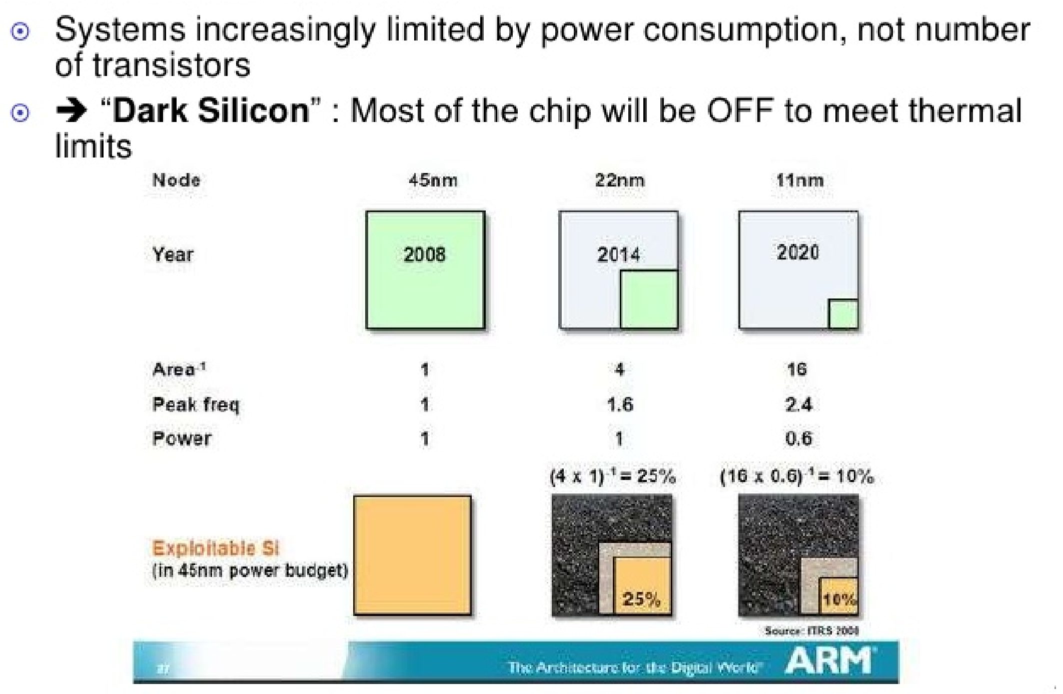
Figure 1.23: Sweetspot shift in DVFS approach for higher leakage on a real-time task.

1.4.5 Future Trends

Transistors are still being made smaller and smaller.

We have hit the Power Wall resulting in a **Dark Silicon** approach. We can no longer turn on all of the chip and get the heat out cheaply: perhaps one tenth maximum for today's 22 nanometer chips. Even less in the future. Water cooling remains an option to mitigate the Power Wall.

Insights Article



- Systems increasingly limited by power consumption, not number of transistors
- ➔ "**Dark Silicon**" : Most of the chip will be OFF to meet thermal limits

Node	45nm	22nm	11nm
Year	2008	2014	2020
Area^{-1}	1	4	16
Peak freq	1	1.6	2.4
Power	1	1	0.6
		$(4 \times 1)^{-1} = 25\%$	$(16 \times 0.6)^{-1} = 10\%$
Exploitable Si (in 45nm power budget)		25%	10%

Source: ITRS 2008

The Architecture for the Digital World® ARM

Slow, bulky power transistors will turn thousands of power islands on and off under automated or manual control.

Conservation cores: use of high-level synthesis (HLS) of standard software kernels into application-specific hardware coprocessors and putting them on the chip in case they are needed? Afterall, they have negligable cost if not turned on. Venkatesh

KG 2 — Masked versus Reconfigurable Technology & Computing

There are different types of silicon chip and different types of manufacturer. Manufacturers can be broadly classified as:

1. So-called 'IDM' or 'vertical market' chip makers such as IBM and Intel that design, manufacture and sell their chips (Integrated Device Manufacturers).
2. Fabless manufacturers such as NVIDIA and Xilinx that design and sell chips but outsource manufacturing to foundry companies.
3. The foundry companies (such as TSMC and UMC) that manufacture chips designed and sold by their customers.

The world's major foundries are SMC and TSMC: Taiwan Semiconductor Manufacturing Company Limitedbut some verticals also provide fab services, perhaps even to competitors in the application space.

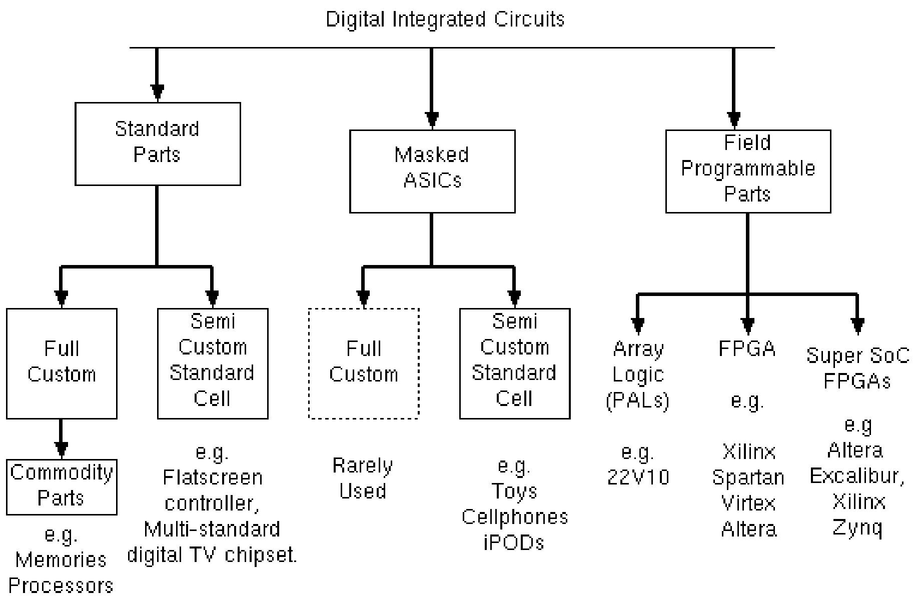

Digital Integrated Circuits
Standard Parts
Masked ASICs
Field Programmable Parts
Full Custom
Semi Custom Standard Cell
Full Custom
Semi Custom Standard Cell
Array Logic (PALs)
FPGA
Super SoC FPGAs
Commodity Parts
e.g. Memories Processors
e.g. Flatscreen controller, Multi-standard digital TV chipset.
Rarely Used
e.g. Toys Cellphones iPODs
e.g. 22V10
e.g. Xilinx Spartan Virtex Altera
e.g Altera Excalibur, Xilinx Zynq

Figure 2.1: A rough taxonomy of digital integrated circuits.

Example Standard Cell Project: 8 Bit Adder0.5 Micron Cell Library

Figure 2.1 presents a historical taxonomy of chip design approaches. The top-level division is between standard parts, ASICs and field-programmable parts. Where a standard part is not suitable, the choice between full-custom and semi-custom and field-programmable approaches has to be made, depending on performance, production volume and cost requirements.

Additional notes:

There are deviations from this taxonomy: Complex PLDs cross between PALs and FPGA with low pin-to-pin delay. Structured ASICs were mask-programmed FPGAs popular around 2005. Today (2012-16), super FPGAs such as Zync are obliterating semi-custom masked ASICs for all but very-high-volume products. When Will FPGAs Kill ASICs?

Chips can be classified by function: Analog, Power, RF, Processors, Memories, Commodity: logic, discretes, FPGA and CPLD, SoC/ASIC, Other high volume (disk drive, LCD, ...).

2.0.6 Chip Types and Classifications

An SSD drive, such as the one in figure 2.2, originally used only standard parts: a microprocessor/microcontroller with off-chip DRAM and twenty FLASH chips. But the huge volume of production means that a custom microcontroller was soon preferred, but the FLASH chips themselves remain standard parts.

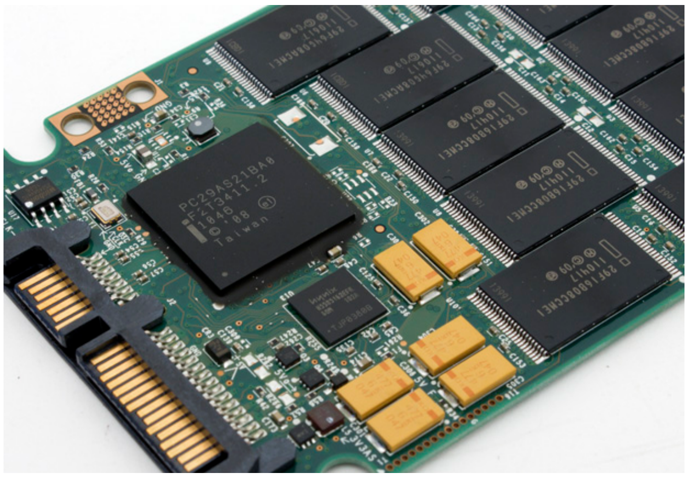


Figure 2.2: FLASH is now replacing spinning media in all but archival applications.

2.0.7 Standard Parts

A **standard part** is essentially any chip that a chip manufacturer is prepared to sell to someone else along with a datasheet and EDA (electronic design automation) models. The design may actually previously have been an ASIC for a specific customer that is now on general release. Many standard parts are general-purpose logic, memory and microprocessor devices. These are frequently full-custom designs designed in-house by the chip manufacturer to make the most of in-house fabrication line, perhaps using optimisations not made available to others who use the line as a foundry. Other standard parts include graphics controllers, digital TV chipsets, GPS receivers and miscellaneous useful chips needed in high volume.

2.0.8 Masked ASICs.

A masked ASIC (application-specific integrated circuit) is a device manufactured for a customer involving a set of masks where at least some of the masks are used only for that device. These devices include full-custom and semi-custom ASICs and masked ROMs.

A full-custom chip (or part of a chip) has had detailed, manual design effort expended on its circuits and the position of each transistor and section of interconnect. This allows an optimum of speed and density and power consumption.

Full-custom design is used for devices which will be produced in very large quantities: e.g. millions of parts where the design cost is justified. Full-custom design is also used when required for performance reasons. Microprocessors, memories and digital signal processing devices are primary users of full-custom design.

In semi-custom design, each cell has a fixed design and is repeated each time it is used, both within a chip and across many devices which have used the library. This simplifies design, but drive power of the cell is not optimised for each instance.

Semi-custom is achieved using a library of logic cells and is used for general-purpose VLSI design.

2.0.9 ASIC - Application-Specific Integrated Circuit

The cost of developing an ASIC has to be compared with the cost of using an existing part or an FPGA. The existing part may not perform the required function exactly, requiring either a design specification change, or some additional *glue logic* to adapt the part to the application.

More than one ASIC may be needed under any of the following conditions:

- application-specific functions are physically distant,
- application-specific functions require different technologies,
- application-specific functions are just too big for one ASIC,
- it is desired to split the cost and risk or reuse part of the system later on.

Factors to consider on a per-chip basis:

- power consumption limitation (powers above 5 Watts need special attention),
- die size limitation (above 11 mm on a side might escalate cost per mm^2),
- speed of operation — clock frequencies above 1 GHz raise issues,
- special considerations :
  - special static or dynamic RAM needs
  - analogue parts - what is compromised if these are integrated onto the ASIC ?
  - high power/voltage output capabilities for load control: e.g. motors.
- availability of a developed module for future reuse.

2.0.10 Semi-custom (cell-based) Design Approach

Standard cell designs use a set of well-proven logic cells on the chip, much in the way that previous generations of standard logic have been used as board-level products, such as Texas Instruments' System 74.

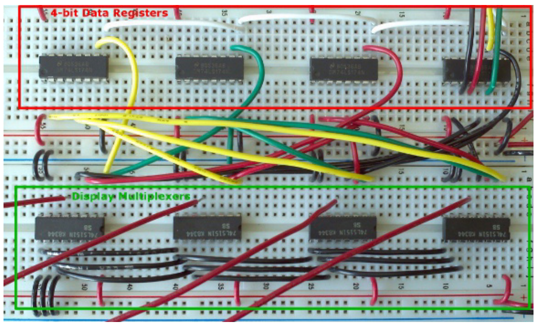
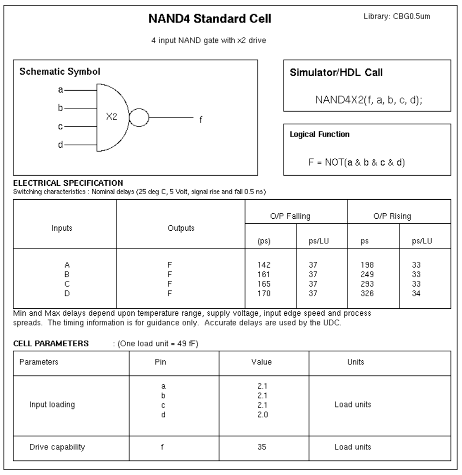


Figure 2.3: Discrete Logic Gates: Semi-custom design puts millions of them all on one die.

A library of standard logic functions is provided. Cells are placed on the chip and wired up by the user, in the same way that chips are placed on the PCB.

- Standard Cell - free placement and free routing of nets,
- Gate Array - fixed placement, masked or electrical programmable wiring.



NAND4 Standard Cell Library: CBG0.5um

4 input NAND gate with x2 drive

Schematic Symbol

Simulator/HDL Call

NAND4X2(f, a, b, c, d);

Logical Function

F = NOT(a & b & c & d)

ELECTRICAL SPECIFICATION
Switching characteristics : Nominal delays (25 deg C, 5 Volt, signal rise and fall 0.5 ns)

Inputs	Outputs	O/P Falling		O/P Rising	
		(ps)	ps/LU	ps	ps/LU
A	F	142	37	198	33
B	F	161	37	249	33
C	F	165	37	293	33
D	F	170	37	326	34

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. The timing information is for guidance only. Accurate delays are used by the UDC.

CELL PARAMETERS : (One load unit = 49 fF)

Parameters	Pin	Value	Units
Input loading	a	2.1	Load units
	b	2.1	
	c	2.1	
	d	2.0	
Drive capability	f	35	Load units

Figure 2.4: Typical cell data sheet from a standard cell library.

Figure 2.4 shows a cell from the data book for a standard cell library. This device has twice the 'normal' drive power, which indicates one of the compromises implicit in standard cell over full-custom, which is that the size (driving power) of transistors used in a cell is not tuned on a per-instance basis.

Mask-programmed gate array has been mostly replaced with the field-programmed FPGA except for analog/mixed-signal niches, such the example from TRIAD

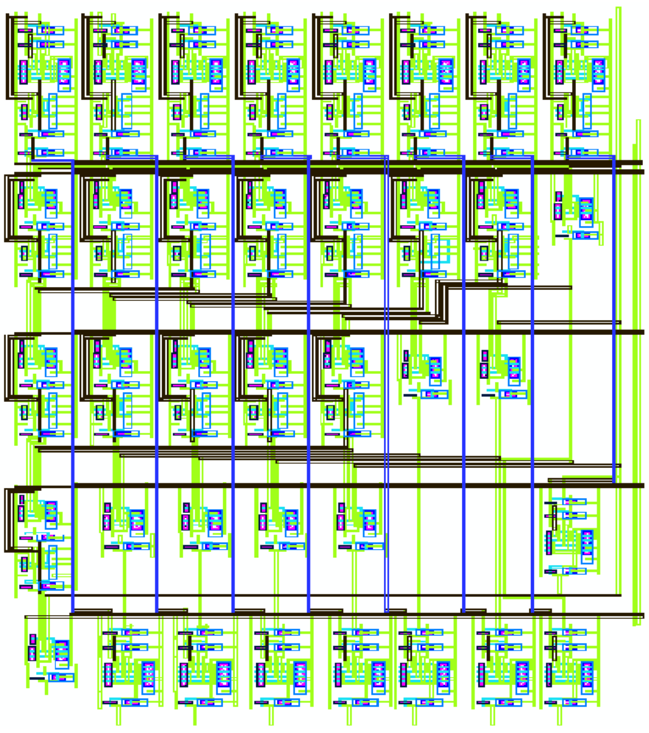


Figure 2.5: Standard cell layout for a Kogge-Stone adder. Taken from a student project (PDF on course web site).

In standard cell designs, cells from the library can freely be placed anywhere on the silicon and the number of IO pads and the size of the die can be freely chosen. Clearly this requires that all of the masks used for a chip are unique to that design and cannot be used again. Mask making is one of the largest costs in chip design. (When) Will FPGAs Kill ASICs?

2.0.11 Cell Library Tour

In the lecture we will have a look at (some of) the following documents:

Standard Cell Student Project: Kogge Stone Adder

Cell libraries in the public domain: 0.5 Micron Cell LibraryAnother 90nm Cell LibrarySome others: VLSI TECHThings to note: there's a good variety of basic gates, including quite a few multi-level gates, such as AND-OR gate. There's also I/O pads, flip-flops and special function cells. Many gates are available with various output powers. For each gate there are comprehensive figures that enable one to predict its delay and energy use, taking into account its track loading, how many other gates it is feeding and the current supply voltage.

2.0.12 ASIC Costs: RE and NRE.

The cost of a chip splits into two parts: non-recurring engineering (NRE) and per-device cost.

Item	Cost (KUSD)	Total (KUSD)
NRE: 6 months : 10 H/W Engineers	250 pa	1250
NRE: 12 months : 20 S/W Engineers	200 pa	4000
NRE: 1 Mask set (45nm)	3000	3000
RE:An 8 inch wafer	5	5n
TOTAL	5	8125 + 5n

For small quantities: share cost of masks with other designs e.g. the MOSIS programme offers multiproject wafer (MPW).

2.0.13 Chip cost versus area

The per-device cost is influenced by the yield — the fraction of working dice. The fraction of wafers where at least some of the die work is the '**wafer yield**'. Historically yields have been low, but was typically close to 100 percent for mature 90 nm fabrication processes, but has again be a problem with smaller geometries in recent years.

The fraction of die which work on a wafer (often simply the '**yield**') depends on wafer impurity density and die size. Die yield goes down with chip area. The fraction of devices which pass wafer probe (i.e. before the wafer is diced) and fail post packaging tests is very low. However, full testing of analog sections or other lengthy operations are typically skipped at the wafer probe stage.

Assume processed wafer sale price might be 5000 dollars: A six inch diameter wafer has area ($3.14r^2$) = 18000 mm^2. A chip has area A, which can be anything between 2 to 200 mm^2 (including scoring lines). Dies per wafer is $18000/A$.

Probability of working = wafer yield $\times$ die yield (assume wafer yield is 1.0 or else included in the wafer cost).

Assume 99.5 percent of square millimetres are defect free. Die yield is then

$$P(\text{All A squares work}) = 0.995^A$$

cost of working dice is

$$\frac{5000}{\frac{18000}{A}0.995^A} \text{ dollars each.}$$

Cost of a working die given a six inch wafer with a processing cost of 5000 dollars and a probability of a square millimetre being defect free of 99.55 percent.

Area	Wafer dies	Working dies	Cost per working die
2	9000	8910	0.56
3	6000	5910	0.85
4	4500	4411	1.13
6	3000	2911	1.72
9	2000	1912	2.62
13	1385	1297	3.85
19	947	861	5.81
28	643	559	8.95
42	429	347	14.40
63	286	208	24.00
94	191	120	41.83
141	128	63	79.41
211	85	30	168.78
316	57	12	427.85
474	38	4	1416.89

For a chip with regular structure, such as a memory or an FPGA, additional hidden capacity can be deployed by burning fusible straps (aka links) during wafer probe test. This increases yield despite the larger area shipped in defect-free dies. AMD marketed a range of 3-core CPUs where the 4th, present on the die, had been strapped off.

2.0.14 Gate Arrays and Field-Programmable Logic.

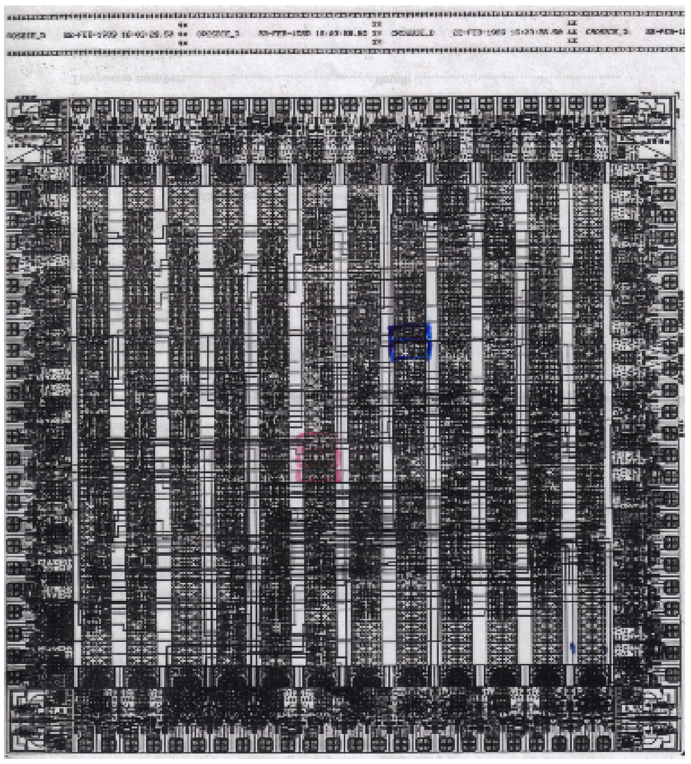


Figure 2.6: Mask for a Mask-Programmed Gate Array: (Greaves 1995, ECL for Ring Network)

Figure 2.6 reveals the regular layout of a masked gate array showing bond pads around the edge and wasted silicon area (white patches). A gate array comes in standard die sizes containing a fixed layout of configurable cells. Historically, there were two main forms of gate array:

- Mask Programmable,
- Field Programmable (FPGA).

In gate array designs, the silicon vendor offers a range of chip sizes. Each size of chip has a fixed layout and the location of each transistor, resistor and IO pad is common to every design that uses that size. Gate arrays are configured for a particular design by wiring up the transistors, gates and other components in the desired way. Many cells will be unused. For mask-programmed devices, the wiring up was done with the top two or three layers of metal wiring. Therefore only two or three custom masks were needed be made to make a new design. In FPGAs the programming is purely electronic (RAM cells control pass transistors).

The disadvantage of gate arrays is their intrinsic low density of active silicon. This arises from rounding up to the next available die size and the area overhead to support programming. The programming area overhead is especially severe for the FPGA.

2.0.15 FPGA - Field Programmable Gate Array

About 25 to 40 percent of chip sale revenue now comes from field-programmable logic devices. These are chips that can be programmed electronically on the user's site to provide the desired function. PALs and CPLDs are forms of programmable logic that are fast and small. But the most important form today is the FPGA.

Recall the Xilinx/Altera FPGA parts used in the Part IB E+A classes. Field-programmable devices may be volatile (need programming every time after power up), reprogrammable or one-time programmable. This depends on how the programming information is stored inside the devices, which can be in RAM cells or in any of the ways used for ROM, such as electrostatic charge storage (e.g. FLASH).

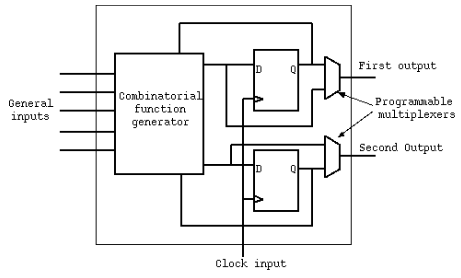
Except for niche applications (such as GaAs instead of Si), FPGAs are now always used instead of masked gate arrays and are starting to kill ASCIs (see link above).

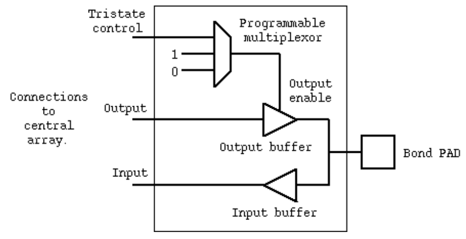
Example: The part Ib practical classes use FPGAs from Altera: ECAD and Architecture Practical Classes

Summary data for a Virtex 5 Xilinx FPGA:

Part number	XC5VLX110T-2FFG1136C
Vendor	Xilinx Inc
Category	Integrated Circuits (ICs)
Number of Gates	110000
Number of I /O	640
Number of Logic Blocks/Elements	8640
Package / Case	1136-FCBGA
Operating Temperature	0C 85C
Voltage - Supply	1.14 V 3.45 V

Circa 2009, 65 nm technology, 6-input LUT, 64 bit D/P RAMs. Today Xilinx has the Virtex 7 series that includes the Zync SoC (of which more later) wikipedia: Virtex FPGA





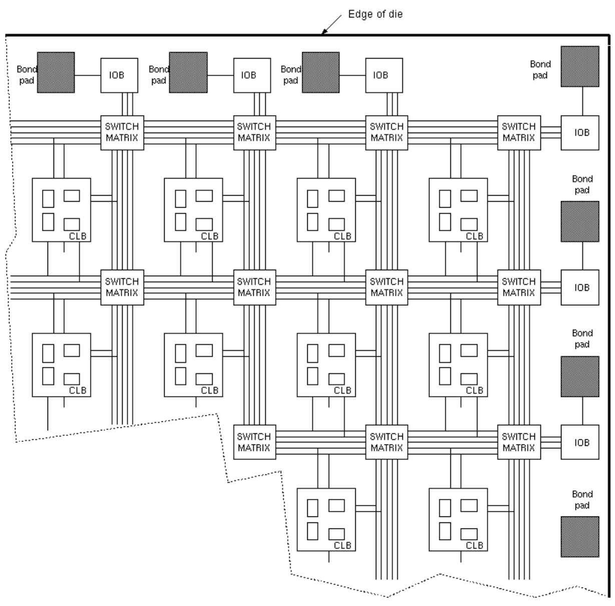


Figure 2.7: Field-programmable gate array structure, showing I/O blocks around the edge, interconnection matrix blocks and configurable logic blocks. In recent parts, the regular structure is broken up by custom blocks, including RAMs and multiplier (aka DSP) blocks.

An FPGA (field-programmable gate array) consists of an array of configurable logic blocks (CLBs), as shown in Figure 2.7. Not shown is that the device also contains a good deal of hidden logic used just for programming it. Some pins are also dedicated to programming. Such FPGA devices have been popular since about 1990.

Each CLB (configurable logic block) or slice typically contains two or four flip-flops, and has a few (five shown) general purpose inputs, some special purpose inputs (only a clock is shown) and two outputs. The illustrated CLB is of the look-up table type, where the logic inputs index a small section of pre-configured RAM memory that implements the desired logic function. For five inputs and one output, a 32 by 1 SRAM is needed. Some FPGA families now give the designer write access to this SRAM, thereby greatly increasing the amount of storage available to the designer. However, it is still an expensive way to buy memory.

FPGAs also soon started to contain RAM blocks (called block RAM or BRAM) and multiplier blocks called DSP (digital signal processing) blocks. The BRAM and DSP blocks are automatically deployed by the design tools by matching specific patterns in the user's RTL when coded appropriately. Today's FPGAs also contain many other 'hard-IP' blocks, such as PCIe, Ethernet and USB controllers that need to be manually instantiated as structural components in the RTL.

FPGAs tend to be slow, achieving perhaps one third of the clock frequency of a masked ASIC, owing to larger die area and because the signals pass through the programmable wiring junctions.

The Xilinx DSP block mostly contains a multiplier that delivers a 48 bit result and an adder for accumulating results where the output from one block has a high-performance programmable connection to a neighbour. The multiplier operands are two's complement, 25 and 18 bit operands. *Exercise: How many DSP blocks are needed for a 32x32 multiplier? What is it's latency? What differences does it make if only 32 bits of the result are needed?*

2.0.16 Circuit Switching

Much of the area of an FPGA is taken up with programmable wiring. The **pass transistor** is a cheap (in area terms) and efficient (in delay terms) form of programmable wiring, but it does not amplify the signal.

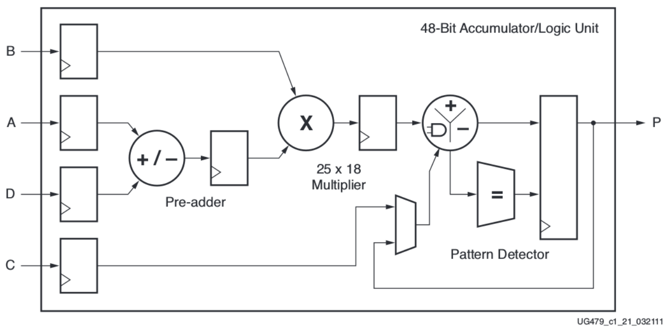


Figure 2.8: So-called DSP block in Xilinx Virtex 7 ((C) Xilinx Inc).

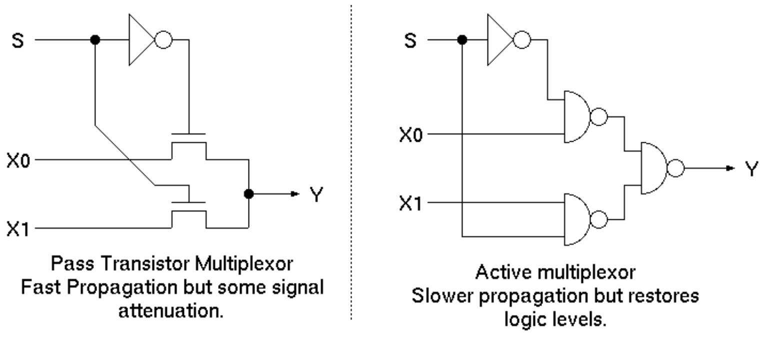

S
X0
X1
Y
Pass Transistor Multiplexor
Fast Propagation but some signal attenuation.
S
X0
X1
Y
Active multiplexor
Slower propagation but restores logic levels.

Figure 2.9: Pass transistor multiplexor compared with active path multiplexor.

FPGAs dominate the recent history of reconfigurable computing but are fine-grain owing to heritage in hardware circuits. There is an argument for having wider busses as the lowest programmable feature, which amortises the programming overhead to some extent, and yields the CGRA - coarse grain reconfigurable array.

2.0.17 Architectural Design: Partition and Exploration

A collection of algorithms and functional requirements must be implemented using one or more pieces of silicon. Each major piece of silicon contains one or more custom or standard microprocessors. Some silicon is custom for a high-volume product, some is shared over several product lines and some is third party or standard parts. The partition decisions take into account various aspects: fundamental silicon capabilities, stability of requirements, market forces, ease of reuse ...

Design Partition: Deciding on the number of processors, number of custom processors, and number of custom hardware blocks. The **system architect** must make make these decisions. SystemC helps them rapidly explore various possibilities.

Co-design and co-synthesis: two basic methods (can do different parts of the chip differently):

- Co-design: Manual partition between custom hardware and software for various processors,
- Co-synthesis: Automatic partitioning: simple ‘device drivers’ and inter-core message formats are created automatically:

Co-synthesis is still not in mainstream use (2018). Example algorithm: MPEG compression:

- A-to-D capture to framestore,

- Colour space conversion (RGB->YUV),
- DCT transform and variable Q quantisation,
- Motion detection,
- Huffman encoding.

Can any of this be best done on a general purpose (say ARM) core ?

MPEG Encoding 1MPEG algorithm 2

2.0.18 H/W Design Partition

A number of separate pieces of silicon are combined to form the product. Reasons for H/W design partition:

- Modular Engineering At Large (Revision Control/Lifetime/Sourcing/Reuse),
- Size and Capacity (chips 6-11 mm in size),
- Technology mismatch (Si/GaAs/HV/Analog/Digital/RAM/DRAM/Flash)
- Supply chain: In-house versus Standard Part.
- Isolation of sensitive RF signals,
- Cost: a new chip spin of old IP is still very expensive.

2.1 H/W versus S/W Design Partition Principles

Many functions can be realised in software or hardware. Decide what to do in hardware:

- physical I/O (line drivers/transducers/media interfaces),
- highly compute-intensive, fixed functions,

what to do on custom processors or with custom instructions/coprocessors on an extensible processor:

- bit-oriented operations,
- highly compute-intensive SIMD,
- other algorithms with custom data paths,
- algorithms that might be altered post tape out.

and what to do in S/W on standard cores:

- highly-complex, non-repetitive functions,
- low-throughput computations of any sort,
- functions that might be altered post tape out,

- generally, as much as possible.

Custom processor synthesis commercial offerings: See Tensilica - now part of Cadence

When designing a sub-system we must choose what to have as hardware, what to have as software and whether custom or standard processors are needed. When designing the complete SoC we must think about sharing of sub-system load over processors. Example: if we are designing a digital camera, how many processors should it have and can the steadicam and motion estimation processing be done in software ? Would a hardware implementation use less silicon and less battery power?

- The functions of a system can be expressed in a programming language or similar form and this can be compiled fully to hardware or left partly as software
- Choosing what to do in hardware and what to do in software is a key decision. Hardware gives speed (throughput) but software supports complexity and flexibility.
- Partitioning of logic over chips or processors is motivated by interconnection bandwidth, raw processing speed, technology and module reuse.

2.1.1 Typical Radio/ Wireless Link Structure.

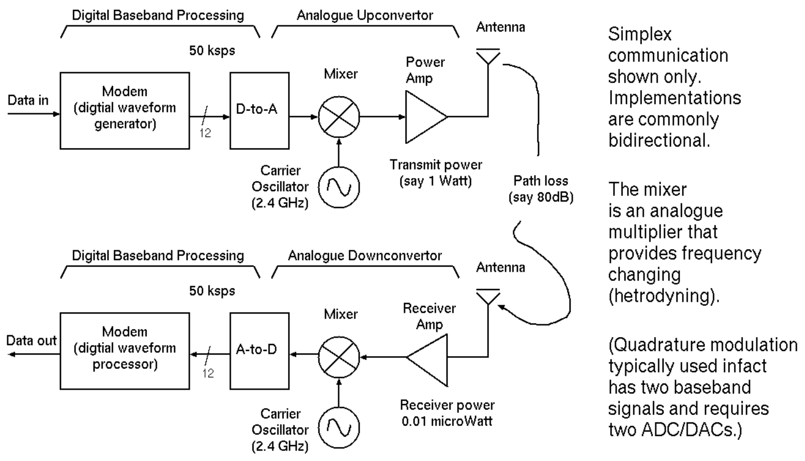

Digital Baseband Processing
Analogue Upconvertor
Antenna
50 ksps
Data in
Modem (digtial waveform generator)
12
D-to-A
Mixer
Power Amp
Carrier Oscillator (2.4 GHz)
Transmit power (say 1 Watt)
Path loss (say 80dB)
Digital Baseband Processing
Analogue Downconvertor
Antenna
50 ksps
Data out
Modem (digtial waveform processor)
12
A-to-D
Mixer
Receiver Amp
Carrier Oscillator (2.4 GHz)
Receiver power 0.01 microWatt
Simplex communication shown only. Implementations are commonly bidirectional.
The mixer is an analogue multiplier that provides frequency changing (hetrodyning).
(Quadrature modulation typically used infact has two baseband signals and requires two ADC/DACs.)

Figure 2.10: Typical structure of modern simplex radio link.

Radio communication above the VHF frequency range (above 150 MHz) uses high-frequency waveforms that cannot be directly processed by A-to-D or D-to-A technology. Hetrodyning is analogue multiplication with a sine wave carrier to perform frequency conversion. This exploits the sin(A)*sin(B) = -cos(A+B)/2 part of the standard trig identity for converting upwards and the other half for converting downwars.

The high frequency circuity is almost always implemented on a separate chip from the digital signal processing (DSP) for the baseband logic. The radio transmitter is typically 50 percent efficient and will use a about 100 mW for most indoor purposes. A cell phone transmitter has a maximum power of 4W which will be used when a long distance from the mast. (Discuss: Having a mast in a school playground means the children are beaming far less radio signal from their own phones into their own heads.) The backlight on a mobile phone LCD may use 300mW (100 LEDs at 30 mW each).

2.1.2 Partitioning example: A Bluetooth Module.

An initial implementation of the Bluetooth radio was made of three pieces of silicon bonded onto a small fibreglass substrate...

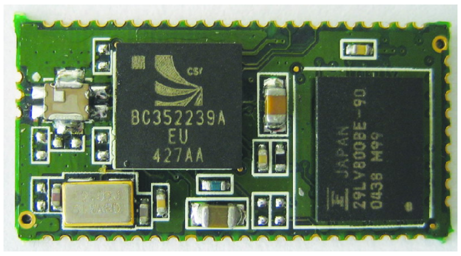


Figure 2.11: Broadcom (Cambridge Silicon Radio) Bluetooth Module circa 2000.

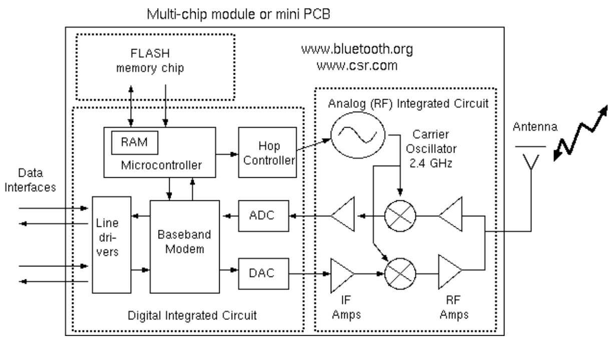


Figure 2.12: Example of a design partition — Block diagram of Bluetooth radio module (circa 2000).

An initial implementation of the Bluetooth radio was made of three pieces of silicon bonded onto a small fibreglass substrate with overall area of 4 square centimetres. The module was partitioned into three pieces of silicon partly because the overall area required would give a low yield, but mainly because the three sections used widely different types of circuit structure.

The analog integrated circuit contained amplifiers, oscillators, filters and mixers that operate in the 2.4 GHz band. This was too fast for CMOS transistors and so bipolar transistors with thin bases were used. The module amplifies the radio signals and converts them using the mixers down to an intermediate frequency of a few MHz that can be processed by the ADC and DAC components on the digital circuit.

The digital circuit had a small amount of low-frequency analog circuitry in its ADC and DACs and perhaps in its line drivers if these are analog (e.g. HiFi). However, it was mostly digital, with random logic implementations of the modem functions and a microcontroller with local RAM. The local RAM holds a system stack, local variables and temporary buffers for data being sent or received.

The FLASH chip is a standard part, non-volatile memory array that can hold firmware for the microcontroller, parameters for the modem and encryption keys and other end application functions. The flash memory is a standard 29LV800BE (Fujitsu) - 8m (1m X 8/512 K X 16) Bit

Today, the complete Bluetooth module can be implemented on one piece of silicon, but this still presents

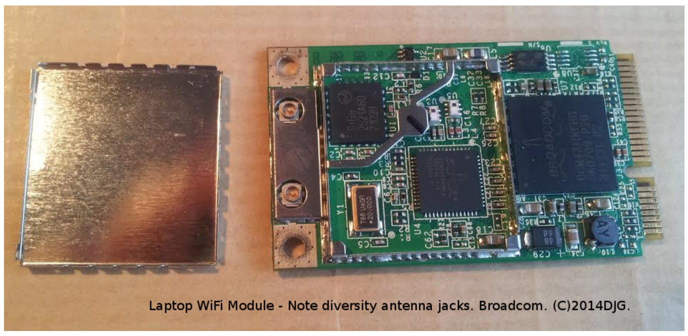

Laptop WiFi Module - Note diversity antenna jacks. Broadcom. (C)2014DJG.

Figure 2.13: WiFi laptop module. Shielding lid, shown left, has been lifted off.

a major technical challenge owing to the diverse requirements of each of the sub-components.

2.2 Super FPGAs: Example Xilinx Zynq

Connecting DRAM to FPGAs has become a requirement in the last decade and hardened DRAM controllers are now common. These work well with soft processors synthesised to programmable logic and for accelerated software applications via HLS, but their performance has lagged CPU and GPU so FPGAs are still not ideal for large data bandwidth applications to RAM.

Although it was common to put 'soft cores' in the programmable logic. Today's devices have hardened CPUs and many other hard IP blocks. The Zynq from Xilinx has two ARM cores, all the standard SoC IP blocks and an area of FPGA programmable logic, all on one die. The same DRAM bank is accessible to both the hardened ARMs and the programmable logic.

Xilinx Zynq-7000 Product Brief (PDF)

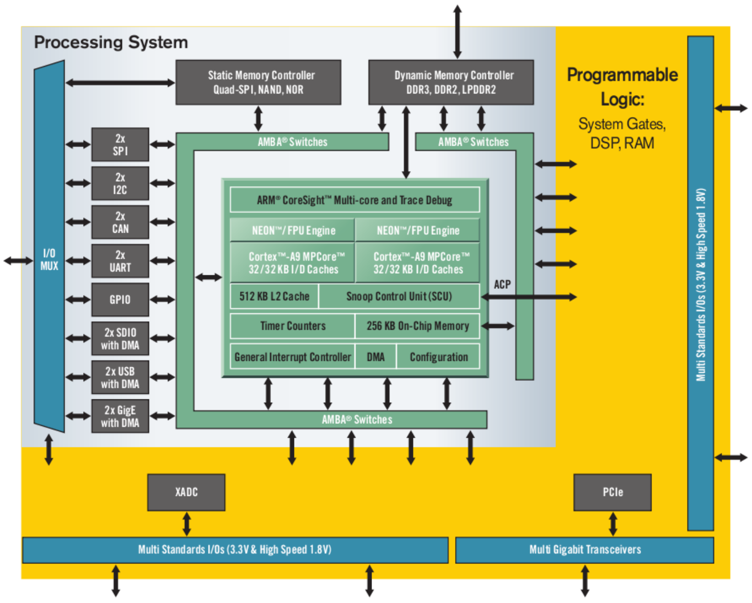
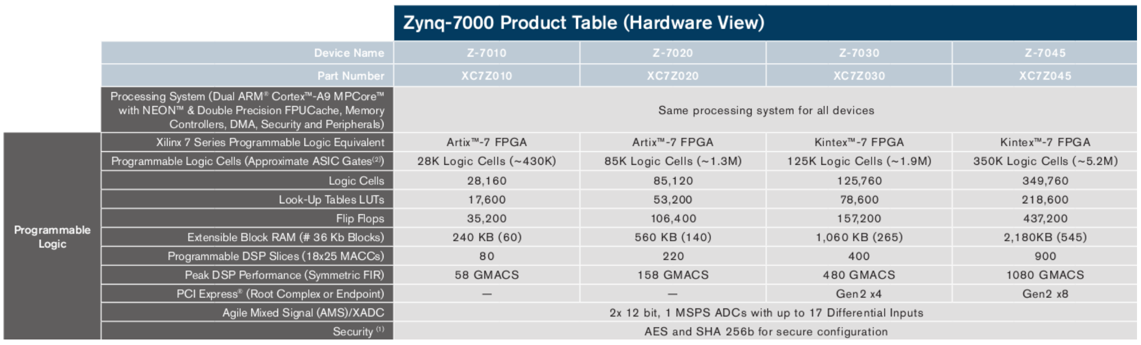

Processing System
Static Memory Controller Quad-SPI, NAND, NOR
Dynamic Memory Controller DDR3, DDR2, LPDDR2
Programmable Logic:
System Gates, DSP, RAM
I/O MUX
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Switches
AMBA® Switches
ARM® CoreSight™ Multi-core and Trace Debug
NEON™/FPU Engine
NEON™/FPU Engine
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
512 KB L2 Cache
Snoop Control Unit (SCU)
Timer Counters
256 KB On-Chip Memory
General Interrupt Controller
DMA
Configuration
ACP
AMBA® Switches
Multi Standards I/Os (3.3V & High Speed 1.8V)
XADC
PCIe
Multi Standards I/Os (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers

Figure 2.14: Xilinx Zynq 7000 Overview.

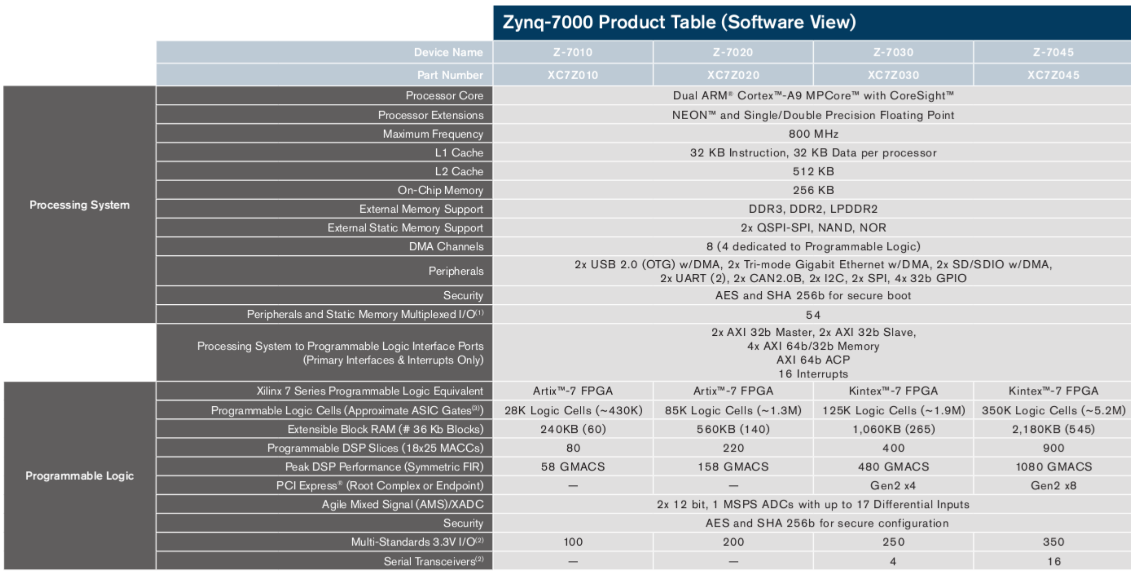


Zynq-7000 Product Table (Hardware View)

	Device Name	Z-7010	Z-7020	Z-7030	Z-7045
	Part Number	XC7Z010	XC7Z020	XC7Z030	XC7Z045
	Processing System (Dual ARM® Cortex™-A9 MPCore™ with NEON™ & Double Precision FPUCache, Memory Controllers, DMA, Security and Peripherals)	Same processing system for all devices			
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix™-7 FPGA	Artix™-7 FPGA	Kintex™-7 FPGA	Kintex™-7 FPGA
	Programmable Logic Cells (Approximate ASIC Gates[2])	28K Logic Cells (~430K)	85K Logic Cells (~1.3M)	125K Logic Cells (~1.9M)	350K Logic Cells (~5.2M)
	Logic Cells	28,160	85,120	125,760	349,760
	Look-Up Tables LUTs	17,600	53,200	78,600	218,600
	Flip Flops	35,200	106,400	157,200	437,200
	Extensible Block RAM (# 36 Kb Blocks)	240 KB (60)	560 KB (140)	1,060 KB (265)	2,180KB (545)
	Programmable DSP Slices (18x25 MACCs)	80	220	400	900
	Peak DSP Performance (Symmetric FIR)	58 GMACS	158 GMACS	480 GMACS	1080 GMACS
	PCI Express® (Root Complex or Endpoint)	—	—	Gen2 x4	Gen2 x8
	Agile Mixed Signal (AMS)/XADC	2x 12 bit, 1 MSPS ADCs with up to 17 Differential Inputs			
	Security [1]	AES and SHA 256b for secure configuration			

Figure 2.15: Xilinx Zynq 7000 FPGA Resources.

The high cost of ASIC masks now makes FPGA suitable for medium volume production. Super FPGAs, like Zynq emerge: the dark silicon trend means we can put all IP blocks on one chip provided we leave them mostly turned off. Flexible I/O routing means physical pads can be IP block bond outs, GPIOs or FPGA.



Zynq-7000 Product Table (Software View)

	Device Name	Z-7010	Z-7020	Z-7030	Z-7045
	Part Number	XC7Z010	XC7Z020	XC7Z030	XC7Z045
Processing System	Processor Core	Dual ARM® Cortex™-A9 MPCore™ with CoreSight™			
	Processor Extensions	NEON™ and Single/Double Precision Floating Point			
	Maximum Frequency	800 MHz			
	L1 Cache	32 KB Instruction, 32 KB Data per processor			
	L2 Cache	512 KB			
	On-Chip Memory	256 KB			
	External Memory Support	DDR3, DDR2, LPDDR2			
	External Static Memory Support	2x QSPI-SPI, NAND, NOR			
	DMA Channels	8 (4 dedicated to Programmable Logic)			
	Peripherals	2x USB 2.0 (OTG) w/DMA, 2x Tri-mode Gigabit Ethernet w/DMA, 2x SD/SDIO w/DMA, 2x UART (2), 2x CAN2.0B, 2x I2C, 2x SPI, 4x 32b GPIO			
	Security	AES and SHA 256b for secure boot			
	Peripherals and Static Memory Multiplexed I/O(1)	54			
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave, 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts			
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix™-7 FPGA	Artix™-7 FPGA	Kintex™-7 FPGA	Kintex™-7 FPGA
	Programmable Logic Cells (Approximate ASIC Gates(3))	28K Logic Cells (~430K)	85K Logic Cells (~1.3M)	125K Logic Cells (~1.9M)	350K Logic Cells (~5.2M)
	Extensible Block RAM (# 36 Kb Blocks)	240KB (60)	560KB (140)	1,060KB (265)	2,180KB (545)
	Programmable DSP Slices (18x25 MACCs)	80	220	400	900
	Peak DSP Performance (Symmetric FIR)	58 GMACS	158 GMACS	480 GMACS	1080 GMACS
	PCI Express® (Root Complex or Endpoint)	—	—	Gen2 x4	Gen2 x8
	Agile Mixed Signal (AMS)/XADC	2x 12 bit, 1 MSPS ADCs with up to 17 Differential Inputs			
	Security	AES and SHA 256b for secure configuration			
	Multi-Standards 3.3V I/O(2)	100	200	250	350
	Serial Transceivers(2)	—	—	4	16

Figure 2.16: Xilinx Zynq 7000 ARM Cores, RAM, DRAM and DMA summary.

KG 3 — Custom Accelerator Structures

Perhaps the first hardware accelerators added to alongside the integer execution units of early computers were for floating-point arithmetic. But accelerators can serve many different purposes and sit elsewhere within the architecture. In this section we note the design considerations.

3.1 H/W to S/W Interfacing Techniques

The system is to be divided into some number of hardware and software blocks with appropriate means of interconnection. The primary ways of connecting hardware to software are:

- CPU coprocessor and/or custom instructions,
- Packet channel connected as coprocessor or mapped to main register file,
- Programmed I/O to pin-level GPIO register,
- Programmed I/O to FIFOs,
- Interrupts (hardwired to one core or dynamically dispatched),
- Pseudo-DMA: processor generates memory addresses or network traffic and the accelerator simply snoops or interposes on the data stream,
- DMA.

Another design point is to do everything in hardware with no CPUs, but a CPU in a supervisory role is normally sensible.

3.2 Custom Accelerators on SoC or ASIC

Suppose something like the following fragment of code is a dominant consumer of power in a portable embedded mobile device:

```
for (int xx=0; xx<1024; xx++)
{
   unsigned int d = Data[xx];
   int count = 0;
   while (d > 0) { if (d&1) count ++;   d >>= 1; }
   if (!xx || count > maxcount) { maxcount = count; where = xx; }
}
```

This kernel tallies the set bit count in each word: such bit-level operations are inefficient using general-purpose CPU instruction sets.

A dedicated **hardware accelerator** avoids instruction fetch overhead and is generally more power efficient. Analysis using Amdhal's law and high-level simulation (SystemC TLM) can establish whether a hardware implementation is worthwhile. There are several feasible partitions:

1. Extend the CPU with a **custom datapath** and custom ALU (Figure 3.1a) for the inner tally function controlled by a **custom instruction**.

2. Add a tightly-coupled **custom coprocessor** (Figure 3.1b) with fast data paths to load and store operands from/to the main CPU. The main CPU still generates the address values `xx` and fetches the data as usual.
3. Place the whole kernel in a **custom peripheral unit** (Figure 3.2) with operands being transferred in and out using programmed I/O or pseudo-DMA.
4. As 3, but with the new IP block having bus master capabilities so that it can fetch the data itself, with polled or interrupt-driven synchronisation with the main CPU.
5. Just use FPGA or FPGA bank without a conventional CPU...

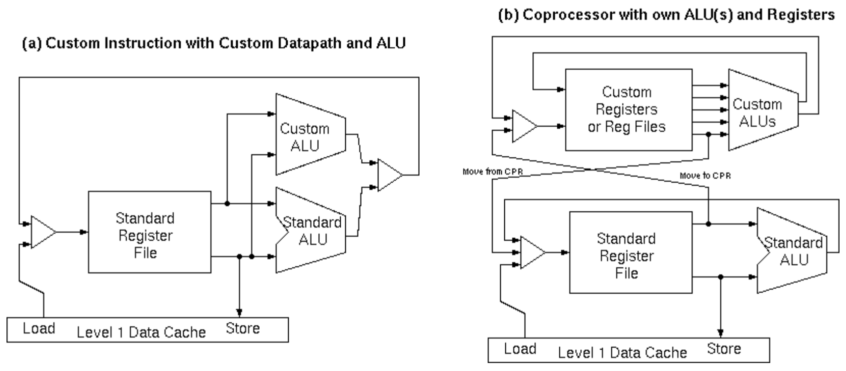


Figure 3.1: A custom ALU operation implemented in two similar ways: as a custom instruction or as a coprocessor.

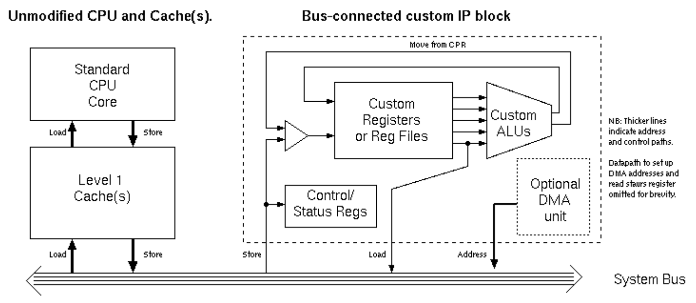

Unmodified CPU and Cache(s).
Bus-connected custom IP block
Move from CPR
Standard CPU Core
Load
Store
Level 1 Cache(s)
Custom Registers or Reg Files
Custom ALUs
Control/ Status Regs
Optional DMA unit
NB: thicker lines indicate address and control paths.
Datapath to set up DMA addresses and read staus register omitted for brevity.
Load
Store
Store
Load
Address
System Bus

Figure 3.2: A custom function implemented as a peripheral IP block, with optional DMA (bus master) capability.

The special hardware in all approaches may be manually coded in RTL or compiled using HLS from the original C implementation.

In the first two approaches, both the tally and the conditional update of the maxcount variable might be implemented in the hardware but most of the gain would come from the tally function itself and the detailed design might be different depending on whether custom instruction or coprocessor were used. The custom instruction operates on data held in the normal CPU register file. The bit tally function alone reads one input word and yields one output word, so it easily fits within the addressing modes provided for normal ALU operations. Performing the update of both the maxcount and word registers in one custom instruction would require two register file writes and this may not be possible in one clock cycle and hence, if this part of the kernel is placed in the custom datapath we might lean more towards the co-processor approach.

Whether to use the separate IP block really depends on whether the processor has something better to do in the meantime and that there is sufficient bus bandwidth for them both to operate.

With increasing available transistor count in the form of **dark silicon** (ie. switched off most of the time) in recent and future VLSI, implementing standard kernels as custom hardware cores is a possible future trend for power conservation. The **conservation cores** project Venkatesh considered implementing the inner loops of a 'mature computations' such as a selection of popular Android applications in silicon on future mobile phones.

3.3 Bump-in-Wire Reconfigurable Accelerator Architectures

FPGA is increasingly seen as a computing element alongside CPU and GPU. Energy savings of two orders of magnitude are often seen when a suitable application is accelerated on FPGA. Execution speed can also commonly increase, although this is hampered by the order-magnitude reduction in clock frequency compared with CPU (e.g 200 MHz instead of 2 GHz).

Historically, many hardware accelerator projects have ultimately been unsuccessful because: either

- The hardware development takes too long and general-purpose CPUs meanwhile progress and overtake (their development teams are vastly more resourced)
- The overhead of copying the data in and out of the accelerator exceeds the processing speed up.
- The hardware implementation is out of date, such as when the requirements or a protocol standard is changed.

But by implementing accelerators on FPGA at a place where the data is moving already, these problems can be largely mitigated. Also, until recently, FPGAs have not had hardened DRAM controllers and consequently been short of DRAM bandwidth.

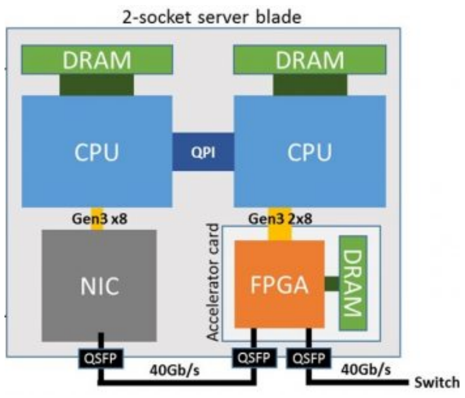


Figure 3.3: Bump-in-Wire design for Microsoft Catapult Accelerator (2016).

Microsoft have had several generations of blade design for their data centres. Recent ones have placed the FPGA in series with blade's network connection, thereby enabling copy-free pre- and post-processing of data. For instance, an index hash can be computed on database fields.

The FPGAs on neighbouring cards are also locally interconnected with a high-speed ring or mesh network, enabling them to be pooled and managed independently of the blade's CPUs. This enables systolic sorting networks and the like to be formed; e.g. for keeping the k-best Bing search results.

The QPI interconnection between CPUs is cache-consistent. Some FPGA-accelerated blade designs connect the FPGA to such a cache-consistent interconnect.

On the Zynq platform (Figure 3.7) a number of methods for connecting to the reconfigurable logic are available - they are mostly via AXI ports. They vary in cache-consistency and bandwidth and initiator/target polarity. Of the initiating ports, both provide connection to the on-chip SRAM and the single DRAM bank that is also shared with the ARM cores, But one form is cache-coherent with the ARMs and the other is not, but has higher bandwidth.

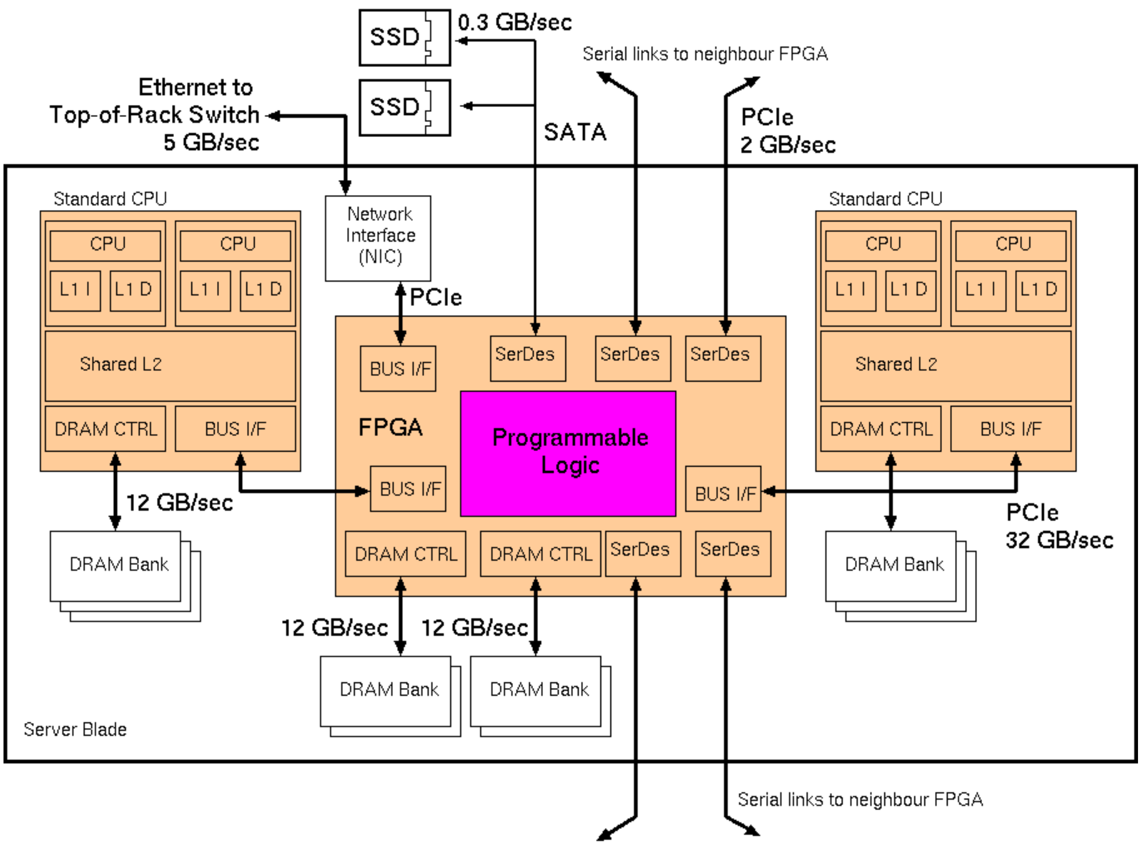


Figure 3.4: Representative 'bump-in-wire' server blade architecture that has the FPGA in series with Network Interface and Disk Drives.

3.3.1 FPGA Computing to replace Von Neumann Dominance?

The Von Neumann computer hit a wall in terms of increasing clock frequency. It is widely accepted that Parallel Computing is the most energy-efficient way forward. The FPGA is intrinsically massively-parallel and can exploit the abundant transistor count of contemporary VLSI. Andre DeHon points out that the Von Neumann architecture no longer addresses the correct problem: he writes 'Stored-program processors are about compactness, fitting the computation into the minimum area possible.'

Why is computing on an FPGA becoming a good idea ? Spatio-Parallel processing uses less energy than equivalent temporal processing (ie at higher clock rates) for various reasons. David Greaves gives nine:

1. Pollack's rule states that energy use in a Von Neumann CPU grows with square of its IPC. But the FPGA with a static schedule moves the out-of-order overheads to compile time.

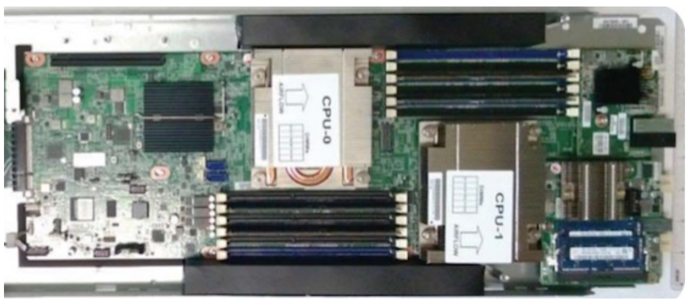
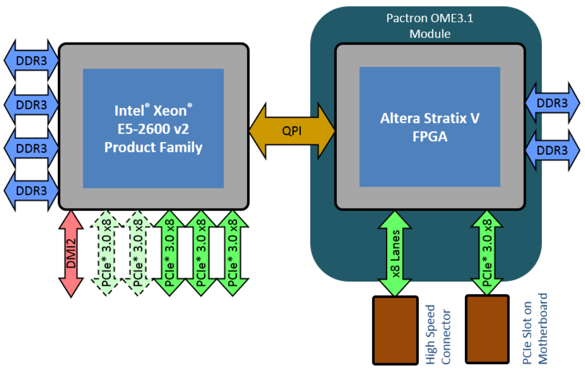
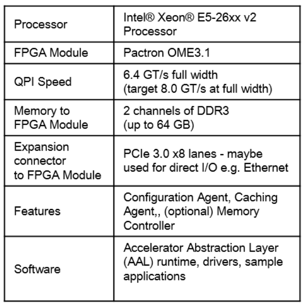


Figure 3.5: Catapult Blade - FPGA is at the lower right, its heatsink visible above its blue DRAM DIMMs.





Processor	Intel® Xeon® E5-26xx v2 Processor
FPGA Module	Pactron OME3.1
QPI Speed	6.4 GT/s full width (target 8.0 GT/s at full width)
Memory to FPGA Module	2 channels of DDR3 (up to 64 GB)
Expansion connector to FPGA Module	PCIe 3.0 x8 lanes - maybe used for direct I/O e.g. Ethernet
Features	Configuration Agent, Caching Agent,, (optional) Memory Controller
Software	Accelerator Abstraction Layer (AAL) runtime, drivers, sample applications

Figure 3.6: Cache-consistent interconnection between CPU and FPGA.

2. To clock CMOS at a higher frequency needs a higher voltage, so energy use has quadratic growth with frequency.
3. Von Neumann SIMD extensions greatly amortise fetch and decode energy, but FPGA does better, supporting precise custom word widths, so no waste at all. Standard computers support fixed data sizes only and only two encodings (integer and floating point), both of which can waste a lot of energy compared with better encodings [Constantinidies].
4. FPGA can implement massively-fused accumulate rather than re-normalising after each summation.
5. Memory bandwidth: FPGA has always had superb on-chip memory bandwidth but latest generation FPGA exceeds CPU on DRAM bandwidth too.
6. FPGA using combinational logic uses zero energy re-computing sub-expressions whose support has not changed. And it has no overhead determining whether it has changed.
7. FPGA has zero conventional instruction fetch and decode energy and its controlling micro-sequencer or predication energy can be close to zero.
8. Data locality can easily be exploited on FPGA — operands are held closer to ALUs near-data-processing (but the FPGA overall size is x10 times larger (x100 area) owing to overhead of making it reconfigurable.

   So
9. The massively-parallel premise of the FPGA is the correct way forward, as indicated by asymptotic limit studies [DeHon].

Programming an FPGA has been a problem. As we shall discuss in a later section, end users cannot be expected to be hardware or RTL experts. Instead, new compiler techniques to port software-style programming to the FPGA are developing. The main approaches today are OpenCL and HLS.

3.3.2 FPGA As The Main Or Only Processor ?

The FPGA generally needs connecting to mainstream computing resources, if only for management or file system access. Figure 3.9 shows a typical setup using a Xilinx Zynq chip, although much larger FPGAs are normally used. The so-called 'substrate' provides basic start/stop and debugging control, together with access to a programmed I/O (PIO) register file. It also provides services for a network-on-chip between the processing blocks that provides an 'FPGA Operating System' API for file server input and output. Finally, it also provides access to DRAM banks and platform-specific inter-FPGA links (not shown).

Will FPGA totally replace Von Neumann? Currently many problems remain: the static-schedule does not work well for all applications, the debugging support is in its infancy and compile times are often

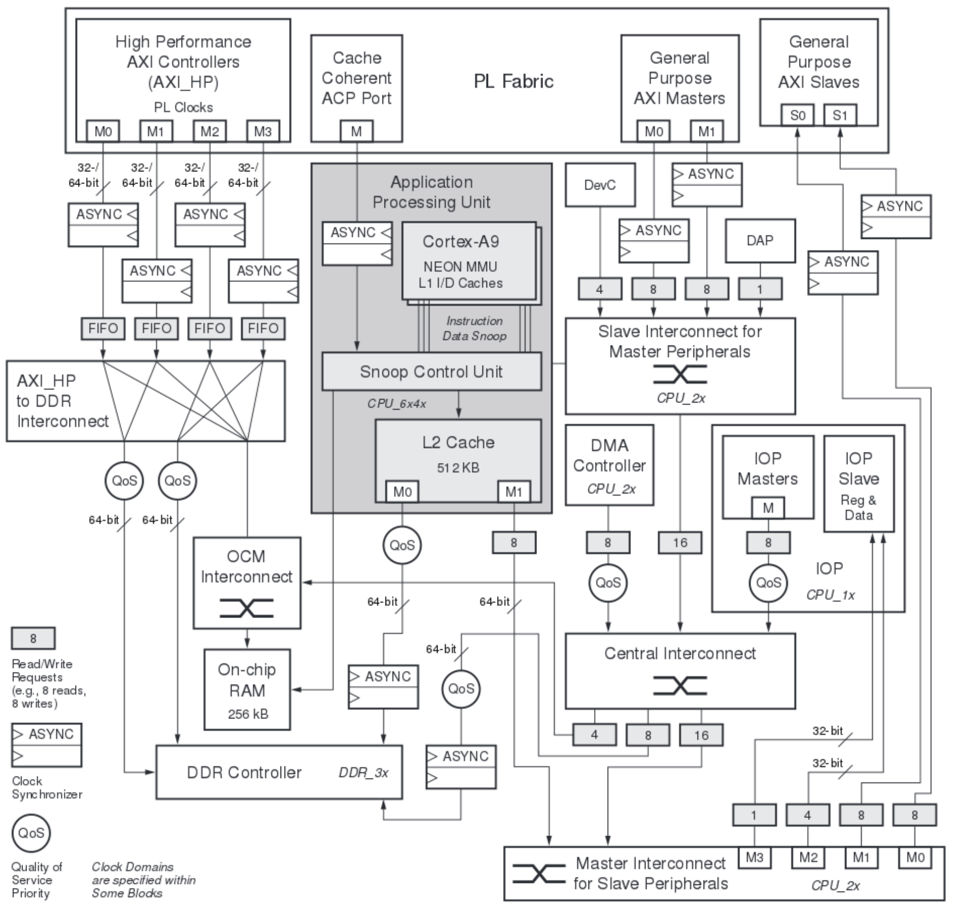


Figure 3.7: Block Diagram of the Xilinx Zynq Platform.

hours. The length overhead of programmable wiring adds a lot of capacitance, so only if the data ends up moving far less physical distance will energy be saved. Nonetheless, there are many success stories in areas such as automated trading, database indexing, deep packet inspection, genomics and cryptology. There are many further papers and PhDs to be written ...

3.3.3 Virtualising the FPGA

FPGA in the datacentre was first deployed by Microsoft for Bing acceleration. A user-programmable FPGA fabric as part of the Cloud offering was made available by Amazon at the start of 2017. Where customer designs are loadable into the FPGA, security issues arise. Overheating, denial of service and data theft must be avoided and a means to virtualise (share) the FPGA resources is needed.

The FPGA has much more user state to context switch than a conventional CPU, so the a sub-Hertz context switch rate is likely. This is at least 3-orders coarser than typical CPU time sharing. All recent FPGA families support dynamic loading of selected parts of the die while the remainder continues operating. The reconfiguration time is typically 10 to 100 ms per zone. This perhaps opens the way to finer-grained sharing, if needed.

One approach to virtualisation is to use dynamically-loaded modules under the 'Shell' and 'App' approach to FPGA real-estate (Figure 3.10) where the customer only has access to the part of the die via partial-reconfiguration. The service shim or shell provides a protected service layer. Disadvantages of this technique are that quite a lot of real estate (and especially the multiplier resource therein) is consumed with overhead. Also there are design fragmentation inefficiencies. And many of the FPGA resources, such as most of the I/O pins and hardened peripherals go unused (not a problem for a Dark Silicon

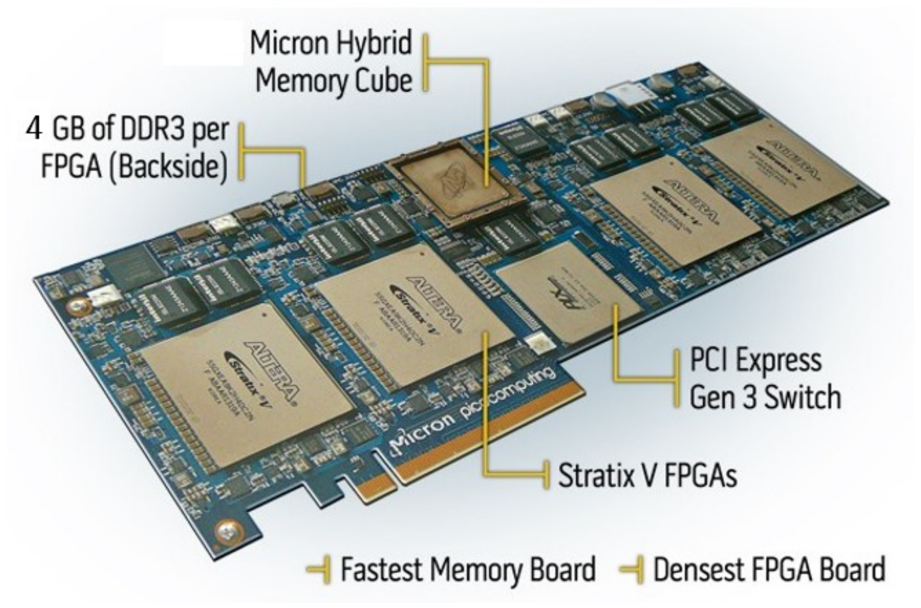


Figure 3.8: A CPU-less cloud sever blade from Pico Computing (4 FPGAs + Lots of DRAM).

future!).

3.3.4 Future FPGA Architectures for the Cloud

It is a possible that a future generation of FPGA devices will be developed specialised for server blades in the cloud. See Figure 3.11. The user logic will connect to the outside world only through a service network. Kiwi HLS uses such a NoC for O/S access, as does LEAP. But reconfigurable interconnection between adjacent zones that jointly host an application (blue) might be provided. Having the service network hardened makes it about 100-fold more area efficient, allowing much more complexity, security and monitoring. The local memory devices would also provide virtualisation assistance in their hardened controllers.

A simple NoC does not take up much FPGA resources, but the idal NoC would probably support credit-based flow control and also include support for virtualisation.

3.3.5 Links and Citations

Amazon, Microsoft, Alibaba and others are now offering FPGA in the cloud services. Some links:

Multi-FPGA System Integrator Intel/Altera Acceleration As A Service Reconfigurable Computing: Architectures and Design Methods. T.J. Todman 2005 Amazon EC2 F1 Instances

Evaluating the Energy Efficiency of Reconfigurable Computing Toward Heterogeneous Multi-Core Computing, Fabian Nowak

A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services, Andrew Putnam

Michael Dales Phd Thesis addressed this first: Managing a Reconfigurable Processor in a General Purpose Workstation Environment - Date 2003

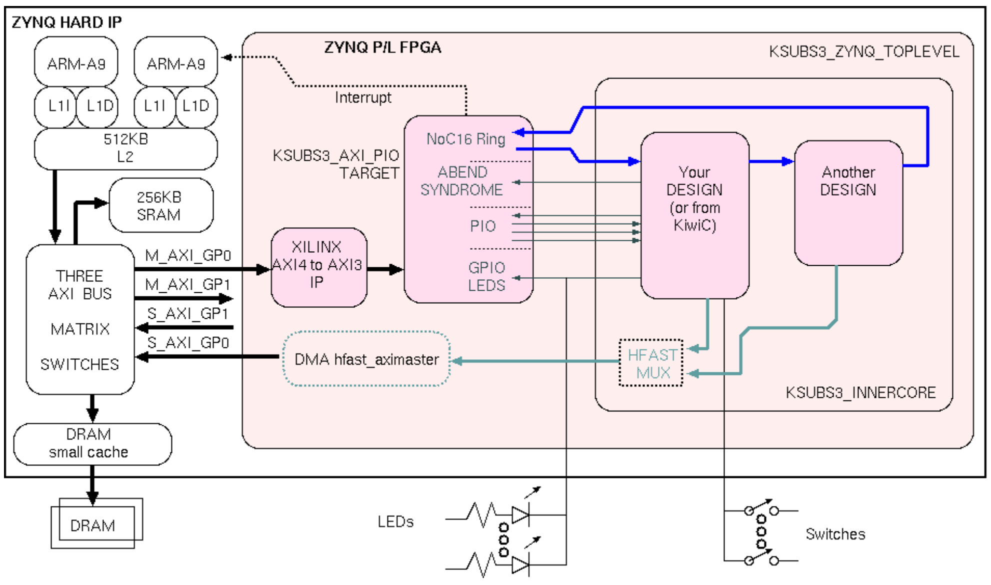


Figure 3.9: Block Diagram of the Kiwi HLS substrate, ksubs3, installed in a Zynq FPGA (all parts simplified).

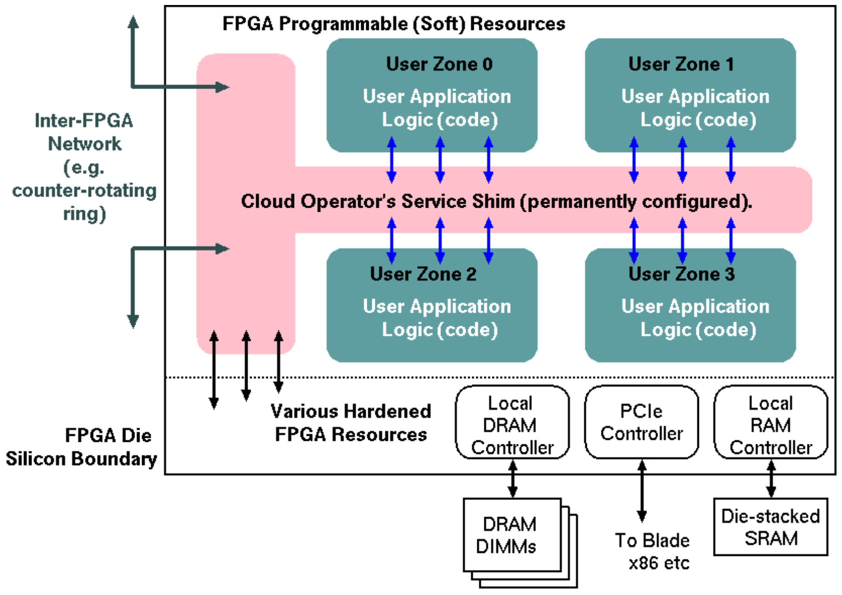

FPGA Programmable (Soft) Resources
User Zone 0
User Application Logic (code)
User Zone 1
User Application Logic (code)
Inter-FPGA Network (e.g. counter-rotating ring)
Cloud Operator's Service Shim (permanently configured).
User Zone 2
User Application Logic (code)
User Zone 3
User Application Logic (code)
FPGA Die Silicon Boundary
Various Hardened FPGA Resources
Local DRAM Controller
PCIe Controller
Local RAM Controller
DRAM DIMMs
To Blade x86 etc
Die-stacked SRAM

Figure 3.10: The reconfigurable logic resources of a conventional FPGA can be manually partitioned into user reconfigurable areas and a shell/shim for service access.

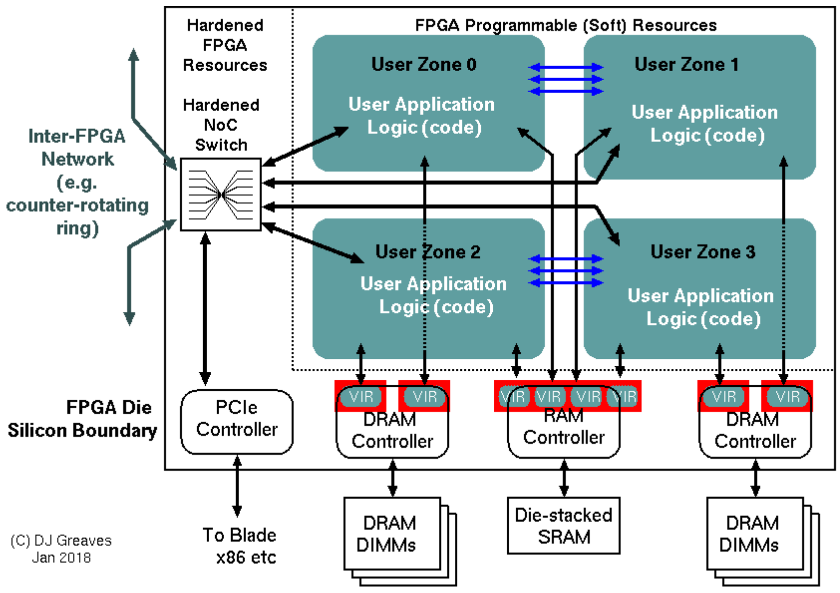


Figure 3.11: Putative block diagram for a cloud-oriented FPGA for scientific acceleration in the future.