Imperial College London

E 2.1 Digital Electronics 2

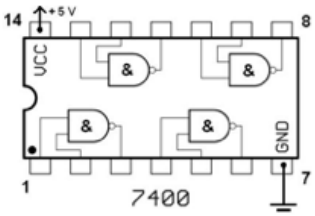
Lecture 2
Introduction to FPGAs

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

Course webpage: www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/
E-mail: p.cheung@imperial.ac.uk

In this lecture, we discuss how digital electronics has evolved over the years, from discrete logic to highly integrated circuits. For this module, the digital technology that we will be focusing on is called “Field Programmable Gate Arrays” or FPGAs. This lecture will introduce you to the idea of such digital devices, and in particular, you will learn about the particular device that you will be using in the Second Year Laboratory later in the term to support this module.

Old ways of implementing digital circuits



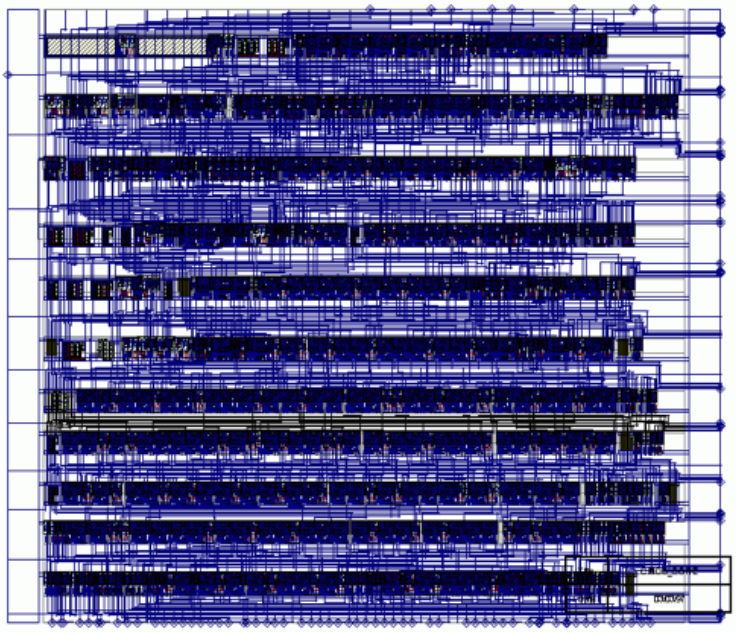
- Discrete logic – based on gates or small packages containing small digital building blocks (at most a 1-bit adder)
- De Morgan's theorem – theoretically we only need 2-input NAND or NOR gates to build anything
- Tedious, expensive, slow, prone to wiring errors

PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 2

Last year you learned about implementing digital circuits using gates such as the one shown here. You can still buy this chip with FOUR NAND gates in one package and this is known as **discrete logic**. We generally **do not** use these any more. It is slow, expensive, consumes lots of energy and very hard to use.

Nevertheless, it is good to learn about NAND and NOR gates because, using De Morgan's theorem, you could in theory design and implement an Intel i7 microprocessor using use two input NAND or NOR gates alone. NAND or NOR gates therefore could be regarded as the building block of all digital circuits. Similarly, you could in theory build a car using only basic Lego blocks. Unfortunately such a car would not be very good.

Early integrated circuits based on gate arrays



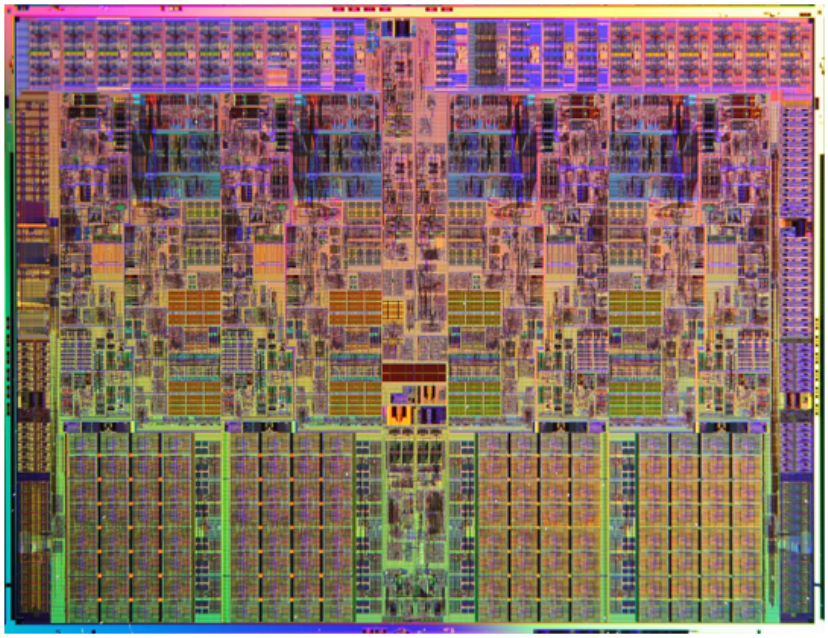
- Rows of gates – often identical in structure
- Connected to form customer specific circuits
- Can be full-custom (i.e. completely fabricated from scratch for a given design)
- Can be semi-custom (i.e. customisation on the metal layers only)
- Once fabricated, the design is fixed

PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 3

In early days of integrated circuits, designers started using rows of basic gates (shown as the dark stuff here arranged in rows). These are either completely customised (full-custom) or it is made with standard rows of gates but leaving the gates unconnected. For a specific design, the gates are connect through metal lines in the wiring channels. Therefore the customisation is only in the wiring metal layers and not the layers with transistors. This is known as "**semi-custom**" application-specific integrated circuits (ASICs).

Modern digital design – full custom IC

- Intel Core i7
- > ¾ billion trans.
- Very expensive to design
- Very expensive to manufacture
- Not viable unless the market is very large



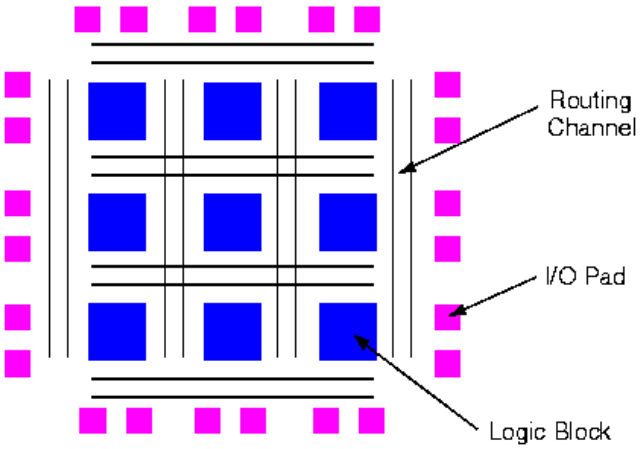
PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 4

Of course you can also customise everything – each transistor and each wiring connected in a full-custom manner. Here is the layout of Intel i7 microprocessor (with 4 cores). Designing such a circuit is very expensive, highly risky, and once designed, it cannot be changed easily.

Most applications in electronic industry cannot afford to embark on such a design. This drives the rise of the Field Programmable Gate Array.

Field Programmable Gate Arrays (FPGAs)

- Combining idea from Programmable Logic Devices (PLDs – Yr 1 Lecture 8) and gate arrays
- First introduced by Xilinx in 1985
- Arrays of logic blocks (to implement logic functions)
- Lots of programmable wiring in routing channels
- Very flexible I/O interfacing logic core to outside world
- Two dominant FPGA makers:
  - Xilinx and Altera
- Other specialist makers e.g. Actel and Lattice Logic



R1.1 p1 - p16

PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 5

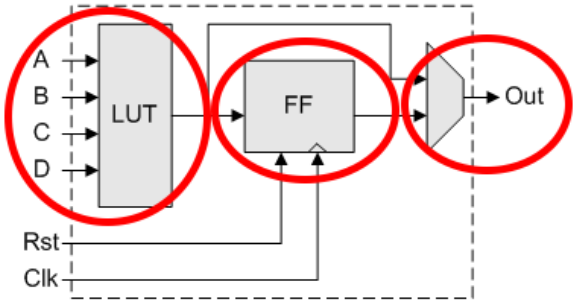
So what is an FPGA? You came across the idea of Programmable Logic Device in the first year, where the user can program what the logic gate does (be it a NAND or NOR or some form of SUM-of-PRODUCT implementation) or an adder, you as a user, can "program" the chip to perform that logic function. Now we can add another layer of user programmability – you can program how these logic gates are connected together! In that way, we have a general programmable logic chip. Unlike the microprocessor where the program is just the instruction to fix digital hardware, here you can program the hardware itself!

The first FPGA was introduced by Xilinx in 1985. It has arrays of logic blocks which are programmable. It is surrounded by PROGRAMMABLE ROUTING RESOURCES, which allows the user to define the interconnections between the logic blocks. It also has lots of very flexible input and output circuits (programmable for TTL, CMOS and other interface standards).

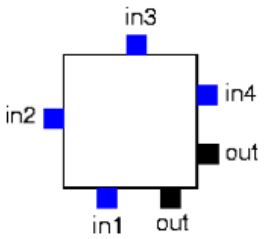
Nowadays, there are two major players in the FPGA domain: Xilinx and Altera (now part of Intel). These two company dominate 90% of the FPGA market with roughly equal share.

Configurable Logic Block (or Logic Element)

- Based around Look-up Tables (LUTs), most common with 4-inputs
- Optional D-flipflop at the output of the LUT
- 4-input LUT can implement ANY 4-input Boolean equation (truth-table)
- Special circuits for cascading logic blocks (e.g. carry-chain of a binary adder)



Logic Element (LE)



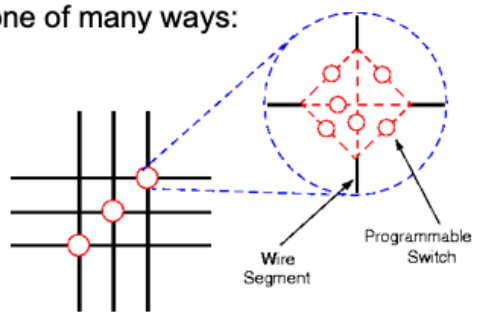
- Each logic block has pins located for easy access:

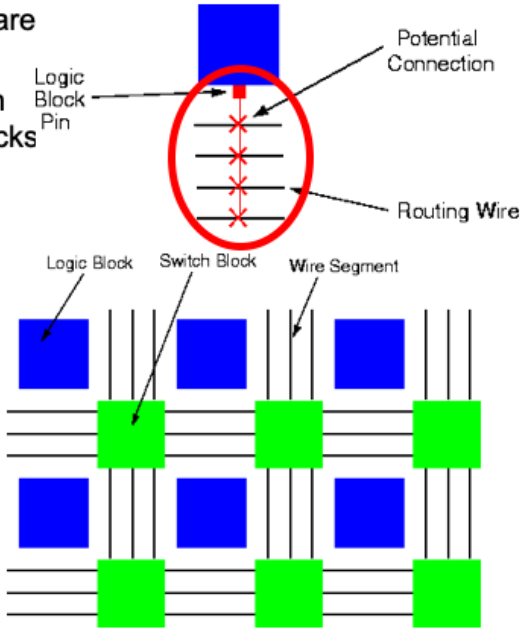
Let us look inside an FPGA. Consider the logic block shown in blue in the last slide (Altera calls their logic block a **Logic Element** (LE)). Typically an LE consists of a 4-input Look-up Table (LUT) and a D-flipflop. Let us for now NOT to worry about how the 4-LUT is implemented internally. Just treat this as a 4-input combinatorial circuit which produces one output signal as shown here. The IMPORTANT characteristic is that the 4-LUT can be user defined (or programmable) to implement ANY 4-input Boolean function.

As we will see later, the lookup table is actually implemented with a bunch of multiplexers.

Programmable Routing

- Between rows and columns of logic blocks are wiring channels
- These are programmable – a logic block pin can be connected to one of many wiring tracks through a programmable switch
- Xilinx FPGAs have dedicated switch block circuits for routing (more flexible)
- Each wire segment can be connected in one of many ways:





PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 7

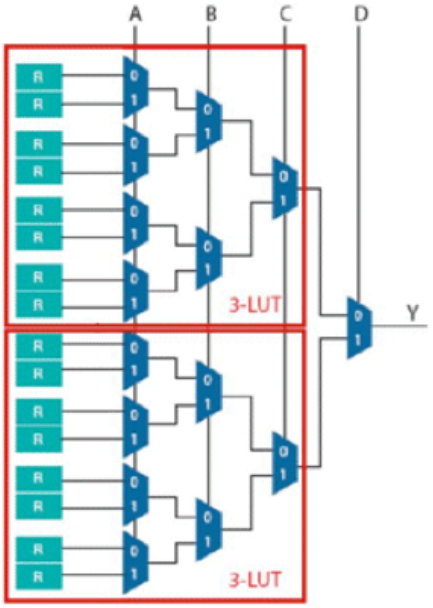
The Logic Elements are surrounded by lots of routing wires and interconnection switches. Typically a signal wire to the Logic Block or Logic Element can be connected to any of these wiring channels through a programmable connection (essentially a digital switch). Xilinx FPGAs also have dedicated switch blocks shown here. Horizontal and vertical wires can be connected through such a switch block with programmable switches (for now, don't worry how that's done).

FPGAs have huge amount of these programmable resources and switches. Typically a very small percentage of these are being connected (i.e. ON) for a given application.

The main advantage and attraction of FPGA comes from the programmable interconnect – more so than the programmable logic.

The Idea of Configuring the FPGA

- Programming an FPGA is NOT the same as programming a microprocessor
- We download a **BITSTREAM** (not a program) to an FPGA
- Programming an FPGA is known as **CONFIGURATION**
- All LUTs are configured using the BITSTREAM so that they contain the correct values to implement the Boolean logic
- Shown here is a typical implementation of a 4-LUT circuit
  - ABCD are the FOUR inputs
  - There is four level of 2-to-1 multiplexer circuits
  - The 16-inputs to the mux tree determine the Boolean function to be implemented as in a truth-table
  - These 16 binary values are stored in registers (DFF)
  - Configuration = setting the 16 registers to 1 or 0



PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 8

Programming an FPGA is called "**configuration**". In programming a computer or microprocessor, we send to the computer instruction codes as '1's and '0's. These are interpreted (or decoded) by the computer which will follow the instruction to perform tasks. The microprocessor needs to be fed these program codes continuously for it to function.

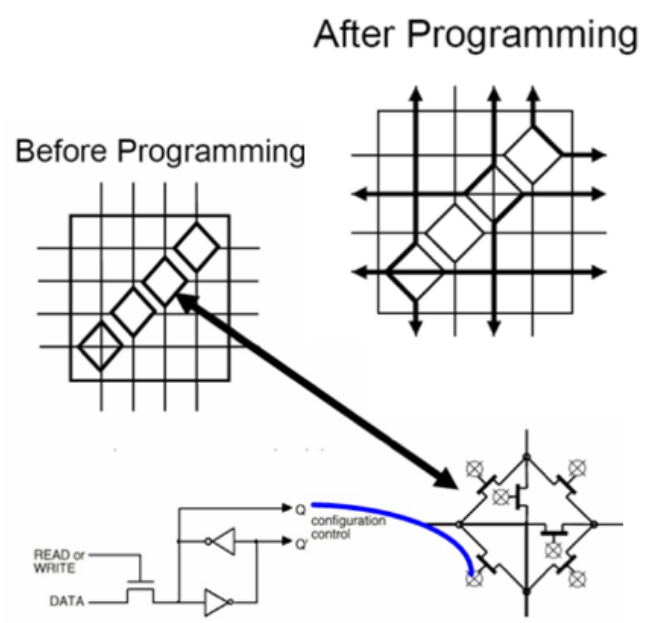
In FPGAs, you only need to **configure** the chip ONCE on power-up. You download to the chip a **BITSTREAM** (also bits in '1's and '0's), which determines the logic functions performed by the Logic Elements, and the interconnecting switches in order to connect the different LEs together to make up your circuit. Once the bitstream is received, the FPGA no longer needs to read the 1's and 0's again, very unlike a microprocessor which has to continually decoding the machine instructions. That's why we say that we **configure** an FPGA (instead of programming an FPGA, although the two words are used interchangeably).

What happens when you configure an FPGA? Let us consider the 4-input LUTs circuit. This is typically implement using a tree of four layers of 2-input to 1-output multiplexers. The entire circuit is behaving like a 16-to-1 multiplexer using the 4 inputs ABCD as the control of the MUX tree. For example, if ABCD = 0000, then the top-most input of the MUX is routed to Y output.

In this way, ABCD forms the input columns of a truth table. For 4-inputs, the truth table has 16 entries. The output Y for each of the truth table entry corresponds to the input of the MUX. Configuration involves fixing the inputs to the 16-to-1 MUX by storing '1' or '0' in the registers R. Changing the 16 values stored, you can change to truth-table to anything you want.

Configuring the routing in an FPGA

- At each interconnect site, there is a transistor switch which is default OFF (not conducting)
- Each switch is controlled by the output of a 1-bit configuration register
- Configuring the routing is simply to put a '1' or '0' in this register to control the routing switches
- Bitstream is either stored on local flash memory or download via a computer
- Configuration happens on power-up



PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 9

To configure the programmable routing, let us look at how the routing circuit works. Take Xilinx SWITCH BLOCK circuit (green blocks in slide 7). This block controls the connections between four horizontal channels and four vertical channels. The diamond shaped block is a potential interconnect site. Inside the switch block circuit, there are 6 transistor switches which are initially all OFF (or open circuit).

The gate input of EACH switch is controlled by the output of a 1-bit register (e.g. a 1-bit D-FF). If the register stores a '1', the routing transistor will have its gate driven high. Since the transistor is an nMOS transistor, it will become conducting. In this way, configuring the routing resources simply means that the correct '1's and '0's are stored in the registers that control these routing transistors.

As you would expect, typically an FPGA would have hundreds of thousands of these routing switches, most of these are OFF. Once programmed, the interconnections are made. The bold lines in the diagram above (after programming) shows the programmed connections.

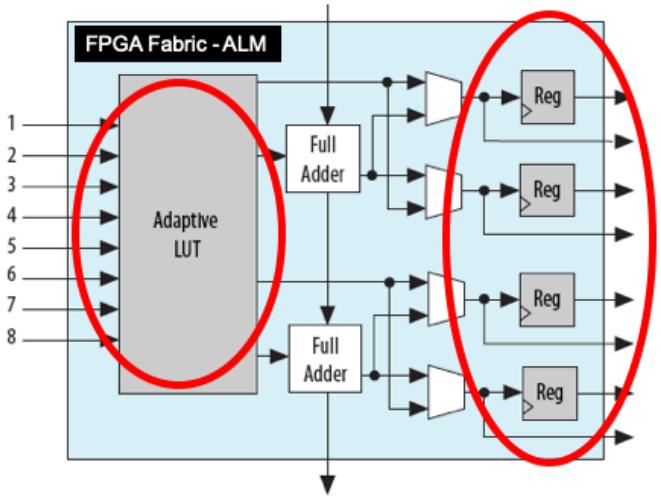
Bitsteam information used for configuration purpose are usually stored on a flash memory chip, which is download to the FPGA during power-up – similar to "booting up a computer". Once this is done, the FPGA is progammed to perform a specific user function (e.g. your design in the VERI experiment).

Alternatively you can send the bitstream to the FPGA via a computer connection to the chip. On the DE1-SOC board, it does both. Powerup DE1 will configure the Cyclone V FPGA chip to a "waiting" mode, which makes the DE1 board talk to the computer via the USB port while flashing the lights ON and OFF. You then send to the board a bitstream of your design via the USB port.

Cyclone V's Adaptive Logic Module (ALM)

- We use Altera's Cyclone V FPGA on this course
- It uses a more complex FPGA logic fabric known as Adaptive Logic Module (ALM)
- The device we use (5CSEMA5F31C6N) has 32,000 ALMs on one chip

- The logic element is more advanced than the original 4-LUT architecture
- The ALM can implement much larger logic functions, or can be broken into a number of smaller units



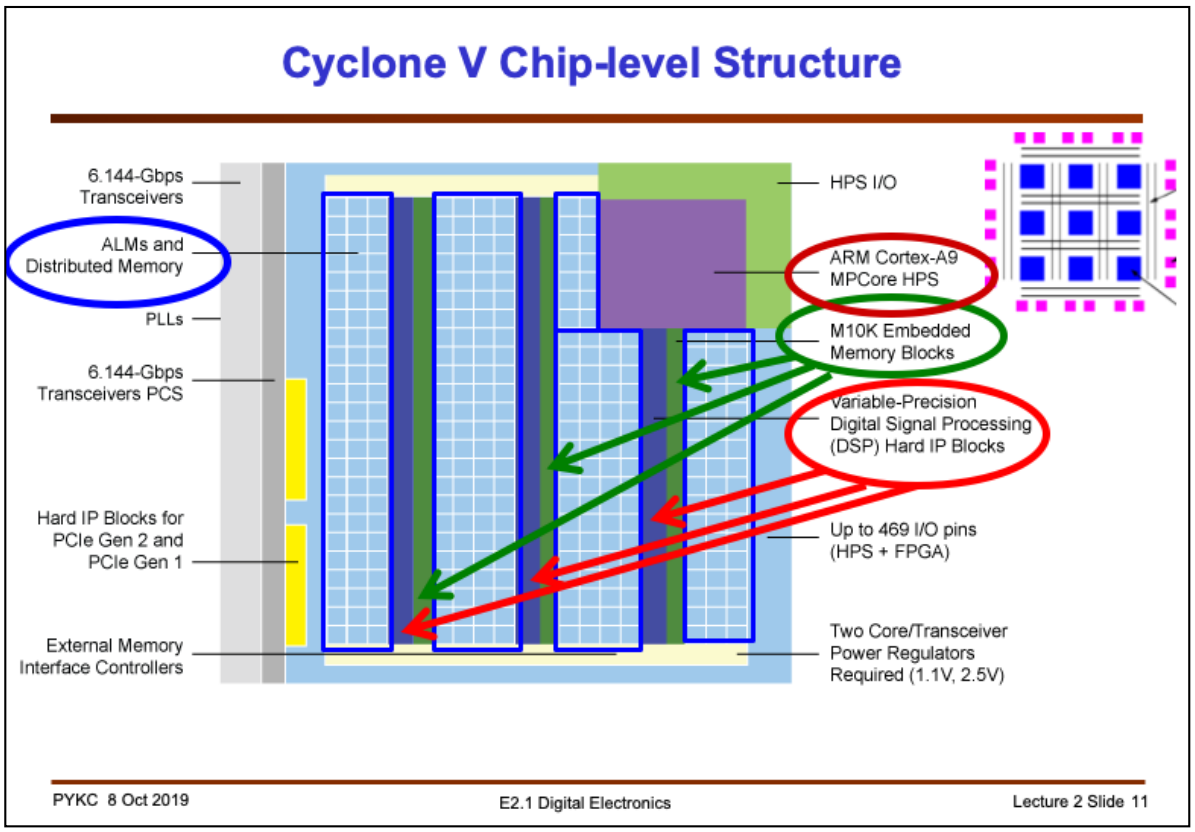
PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 10

Let us now look at the FPGA that you will use for this course. The Altera Cyclone V FPGA has a more advanced programmable logic element than the simple 4-input LUT that we have considered up to now. The call this a Adaptive Logic Module or ALM.

An ALM can take up to 8 Boolean input signals and produces four outputs with or without a register. Additionally, each ALM also can perform the function of a 2-bit binary full adder.

As a user of the Cyclone V FPGA, you don't actually need to worry too much about exactly how the ALM is configured to implement your design. The CAD software will take care of the mapping between your design and the physical implementation using the ALMs. It is however useful to know that as the technology evolves, more and more complicated programmable logic elements are being developed by the manufacturers in order to improve the area utilization of the FPGAs.

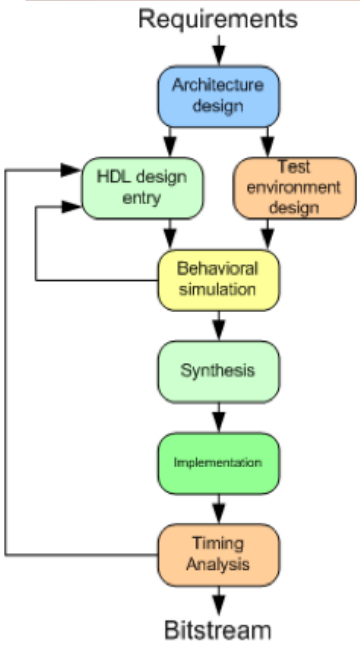
The Cyclone V on the DE1-SOC board has 32,000 ALMs, which could be estimated to be equivalent to 85K+ the old style LEs. Putting this in context, you could put onto this one chip 2,000 32-bit binary adder circuits!



The Cyclone V is much more than just an FPGA with a bunch of Logic Elements (or ALMs). Our chip in the DE1 board has 32,000 ALMs, which is around 85K old style 4-input LUT LEs. On top of that, it also has over 4Mbit of embedded memory, 87 DSP blocks (to do multiply-accumulate operations needed for signal processing), and even a dual-core ARM microprocessor!

It has hard-logic to implement PCIe interface (to fast peripherals) and external memory interface to connect to external memory. It is a truly powerful chip onto which one could implement an entire digital electronic system. Therefore Altera call this Cyclone V System-on-Chip (SoC).

Design Tools – Altera Quartus Prime



- Quartus Prime – a comprehensive design tools for Altera FPGAs
- Special web edition free to download from (need registration):
  - http://fpgasoftware.intel.com
  - Features include (see introduction to Quartus II):
    - design entry
    - compilation from Hardware Description Languages (HDL)
    - synthesis
    - simulation
    - timing analysis
    - power analysis
    - project management

R1.3

PYKC 8 Oct 2019 E2.1 Digital Electronics Lecture 2 Slide 12

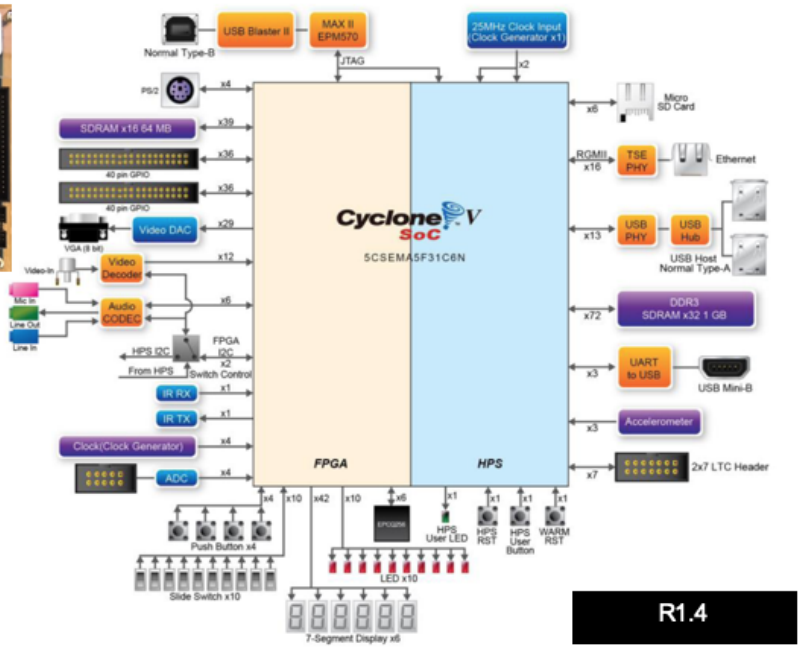
For this course, you will be designing circuits using the free version of the design suite known as Quartus Prime Lite from Intel/Altera. You can download your own copy onto your notebook machine, or you can use the versions that are installed in any PCs located anywhere in the department.

This very powerful design tool contains everything you need to design a complex digital system ON YOUR OWN COMPUTER! However, the software only runs on either a MS Windows or a Linux operating system. If you are using a Mac, you would need to run a Virtual Machine applications (such as Virtual Box) and install Windows or Linux before installing Quartus software.

Beware that the software is very large – you need to have several GB of free disk space. The minimum required RAM is 4GB, and 8GB is recommended.

If your laptop is suitable, do download this software and play with it at home.

DE1-SOC Board

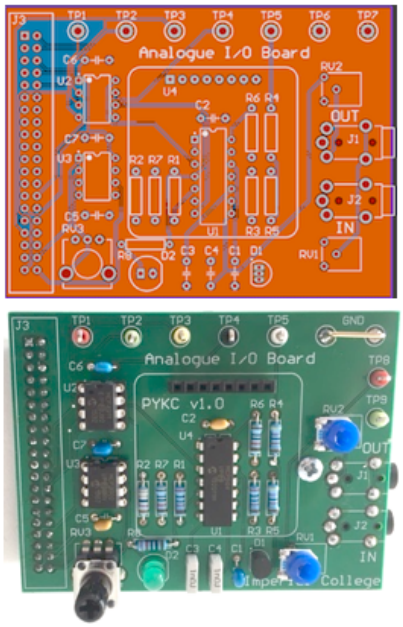


R1.4

This slide shows you the functional blocks of the DE1-SoC board. This has everything you need test basic designs involving switches, 7-segment displays and even a VGA output.

Add-on Board

- Provides analogue inputs and outputs
- Contains 2 channels ADC, one from microphone & one from a socket
- Has 2 channel analogue output with one driven by a DAC and another by a digital signal
- Includes built-in filter and operational amplifier
- Will be using this board for your 2nd year Lab Experiment: VERI



I also provide a purpose-built ADC/DAC board to support the lab experiment. This add-on board in only needed in week 3 onwards during the laboratory sessions. So for now, you can ignore it.