

Electronic System Design 3

1.

The Digital Design Process

Hardware Description Language (VHDL)

Bits 'n' pieces

- Scott Roy
Rankine Building, Room 307B (between I.Thayne & Device Modelling)
S.Roy@elec.gla.ac.uk
- Course changes - already covered by Dr. Weaver
- Lecture handouts, recommended reading, syllabus, ...
- Thanks to Craig Slorach! For a fair chunk of this text.

Nobel Prize in Physics - 2000



KUNGL.
VETENSKAPSAKADEMIEN
THE ROYAL SWEDISH ACADEMY OF SCIENCES

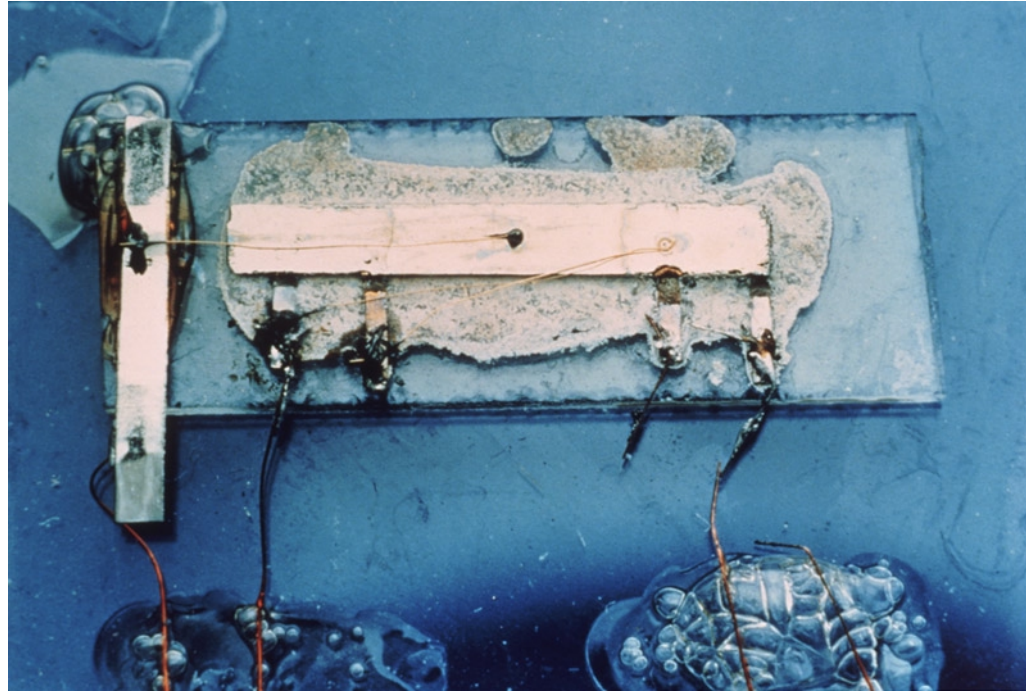
Jack S. Kilby

Texas Instruments, Dallas, Texas, USA

“for his part in the invention of the integrated circuit”

(with Zhores Alferov & Herbert Kroemer - heterostructures)

First Integrated Circuit



- Invented at TI in 1958
- Comprised of a single transistor and other components on a slice of germanium $7/16$ -by- $1/16$ -inches in size.

Summary

- Design Issues
 - IC's and Systems are becoming more and more complex and there is a need for good design analysis and practice.
 - Managing complexity by breaking down systems into manageable parts for design and implementation.
- Tools for Design & Implementation
 - Provide a brief refresher on Hardware Description Languages (HDLs) and consider how they are really used to build digital systems.

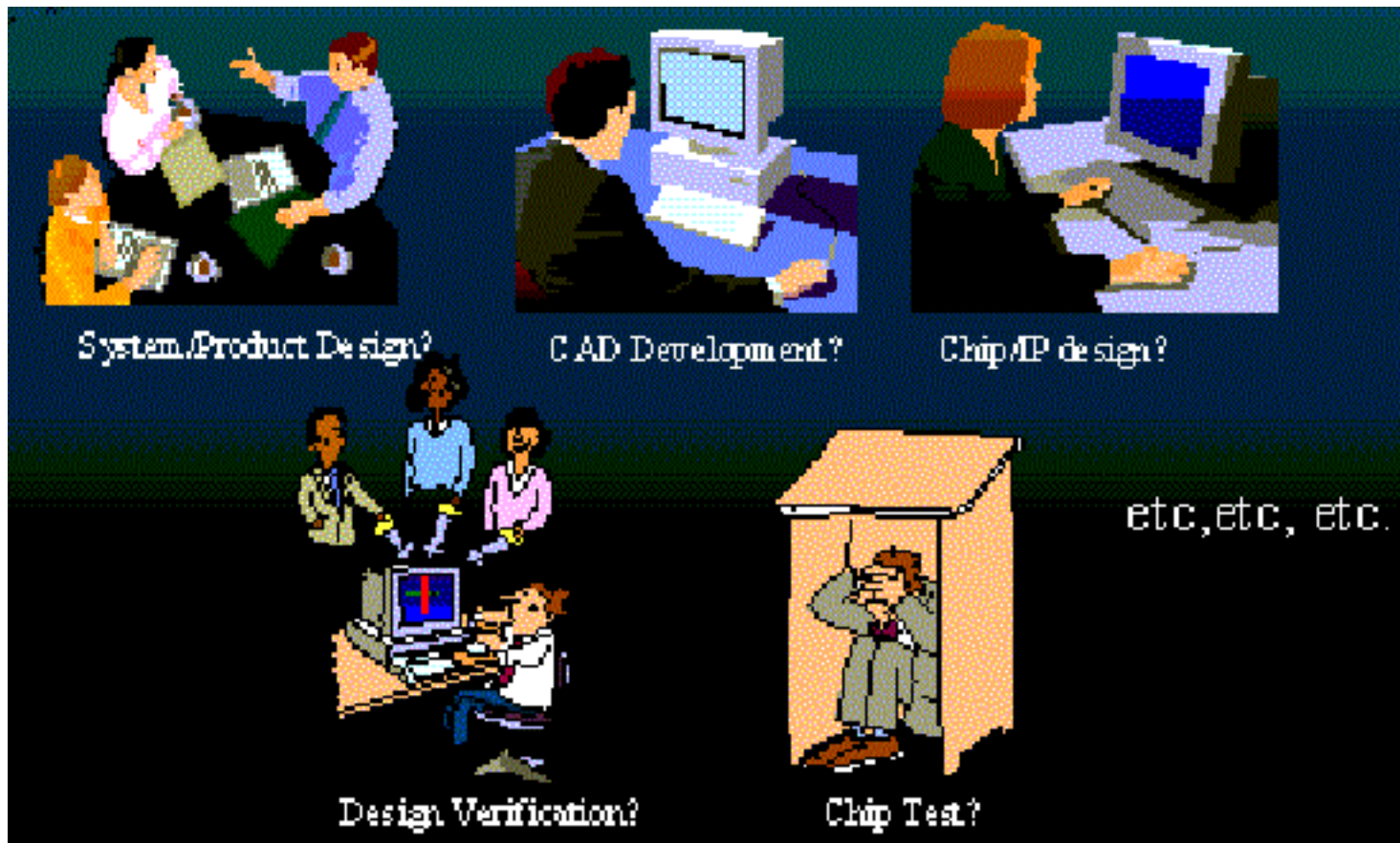
Into Context

- Center around procedures used for *digital* devices
 - Many principles applicable to all areas of electronic design.
 - Concentrate initially on all digital systems.
- Designing chips to perform some specific task
 - From a simple application (e.g. ‘Glue-Logic’)
 - To a complex device (e.g. DVD, CD, Digital TV)
 - What needs to be optimised ?
- Such ‘chips’ are now used everywhere and allow us to build electronic devices that would have been unheard of even 5 years ago !

Some Definitions

- Design
 - The process of taking a functional specification and producing a solution for this specification
 - There may be many different solutions, but we are only ever interested in producing a single one ! (although an optimisation criterion may be extensibility).
- Implementation
 - Implementing the design in some form
 - Or, in simple terms ‘building it’

Design Issues



Motivation

- Building IC's is just another Engineering Problem !
 - Principles no different from designing a car, etc.
- Electronic designs are becoming more and more complex
 - 25 years ago designed with primitive gates (AND, OR, etc.) and connected these together to form systems.
 - With the advent of the ASICs (Application Specific Integrated Circuits) and large FPGAs (Field Programmable Gate Arrays) can design a custom IC for each application / prototype.
 - Integration of general purpose processors and dedicated hardware needs hardware & software design skills (*co-design*).
 - Rebalancing of design criteria.

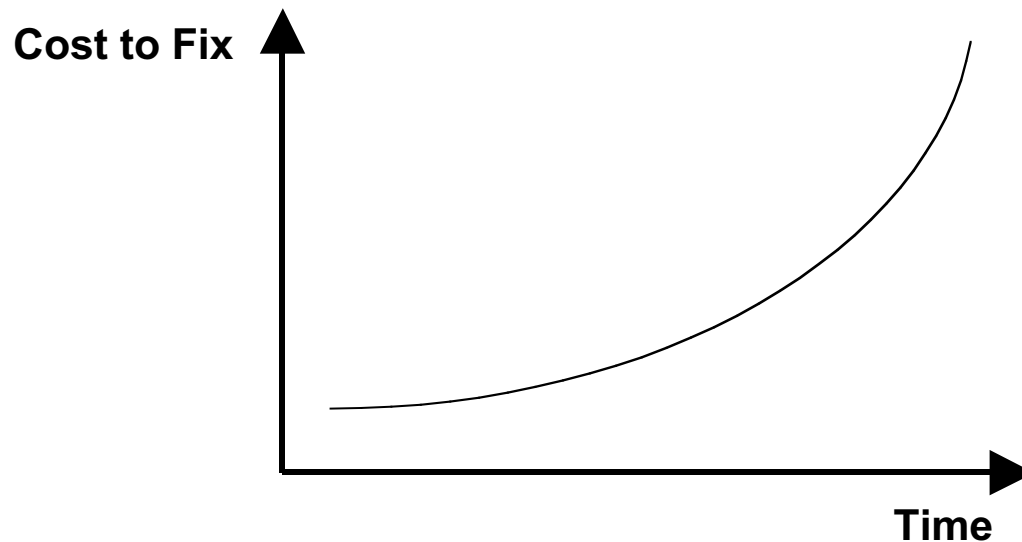
Motivation

- Commercial factors \$\$
 - Cost, speed, power, size, yield, ...
 - Drivers are now more consumer than military.
 - Consumer products have a very short lifespan so *time to market* is critical.
 - Design re-use is therefore becoming crucial.
- Must have robust methods of design verification, ...

Motivation

Have to get things right first time !

- ~~Want to get things right as early as possible~~ as the cost of change/ problem fixing increases exponentially with the stage of design



Ideal Design Process

- So, ideally our design process should...
 - Work for any size of design from the simplest (<1000 gates) design to the very large designs (>1M gates)
 - Allow new designs to be realized rapidly
 - Be structured to get the design ‘right first time’ before we commit to implementation/ manufacture.
 - » Design for test
 - » Design verification
 - Re-use parts of existing designs to speed up time to market

The Problem

- It's the first day of your new job working for a large semiconductor company

Design a 'chip' to....

- So, where do you start ????

Design Flow -I

- Start with some high level functional specification

Joe Bloggs Inc.
The system should

~~~~~

~~~~~

~~~~~

- e.g. LCD Display Driver, Teletext, Telephone, etc.
- May run to several hundred pages and defines the scope of our design

# Design Flow- II

- So, how do we start implementing ?
  - Could now sit down and start building the system from logic gates
  - Would you do this if you were building a car ?
    - ... See you in 10 years time !
- We require a much more structured and organised design approach through the use of
  - Hierarchy
  - Regularity
  - Re-use

# Hierarchical Approach- Introduction

- Hierarchical Approach to System Design
  - Split the overall design into discrete *blocks* which have (typically) a single function and then connect these blocks together to form the complete system (see later)
  - The process is repeated on each of the blocks in the design until a desired level of abstraction is achieved
  - Eventually, we have a list of blocks (with inputs, outputs and required functionality)
  - These then have to be implemented (see later)

# Hierarchical Approach- Example

- In the example below, the design 'top' contains 2 blocks 'BLOCK1' and 'BLOCK2'
  - BLOCK1 is then sub-divided into another 3 blocks

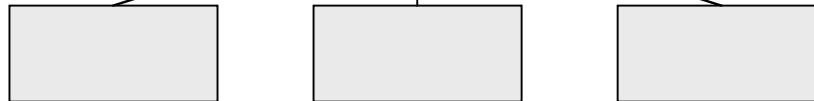
**Level 0**



**Level 1**

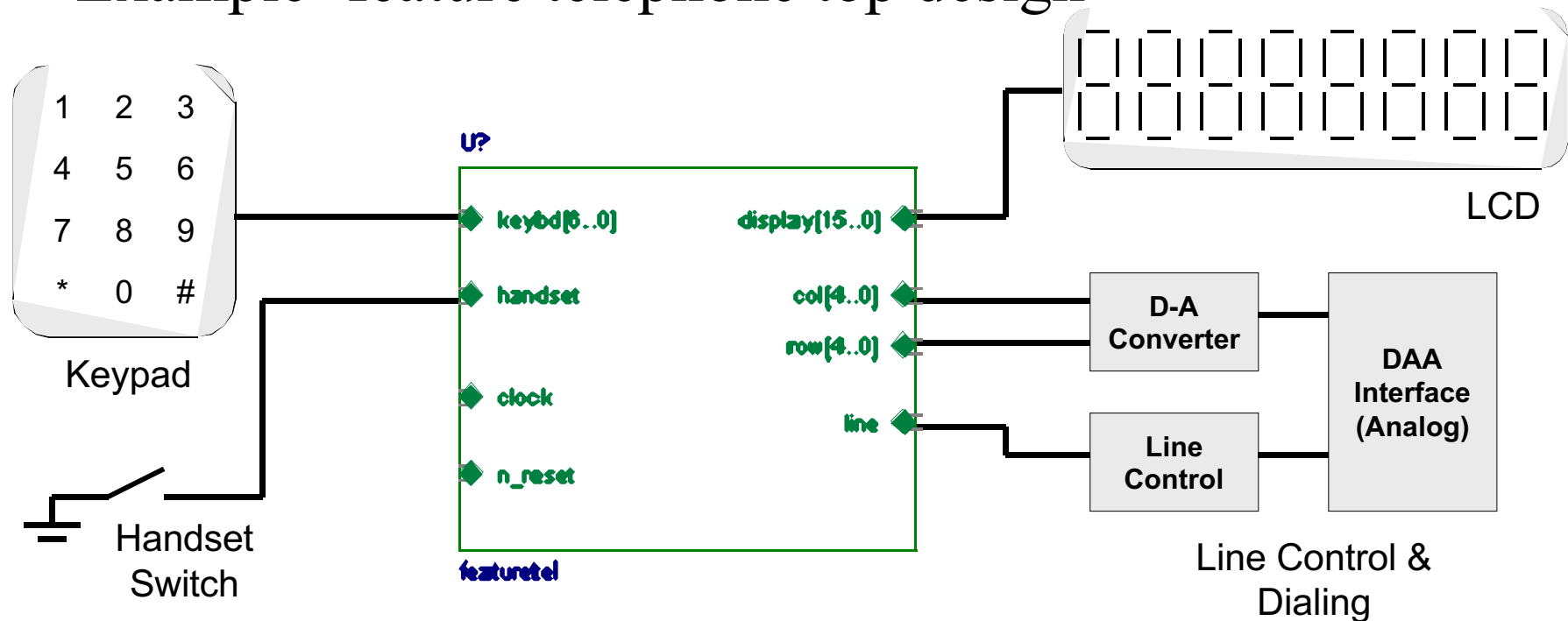


**Level 2**



# Hierarchical Approach- I

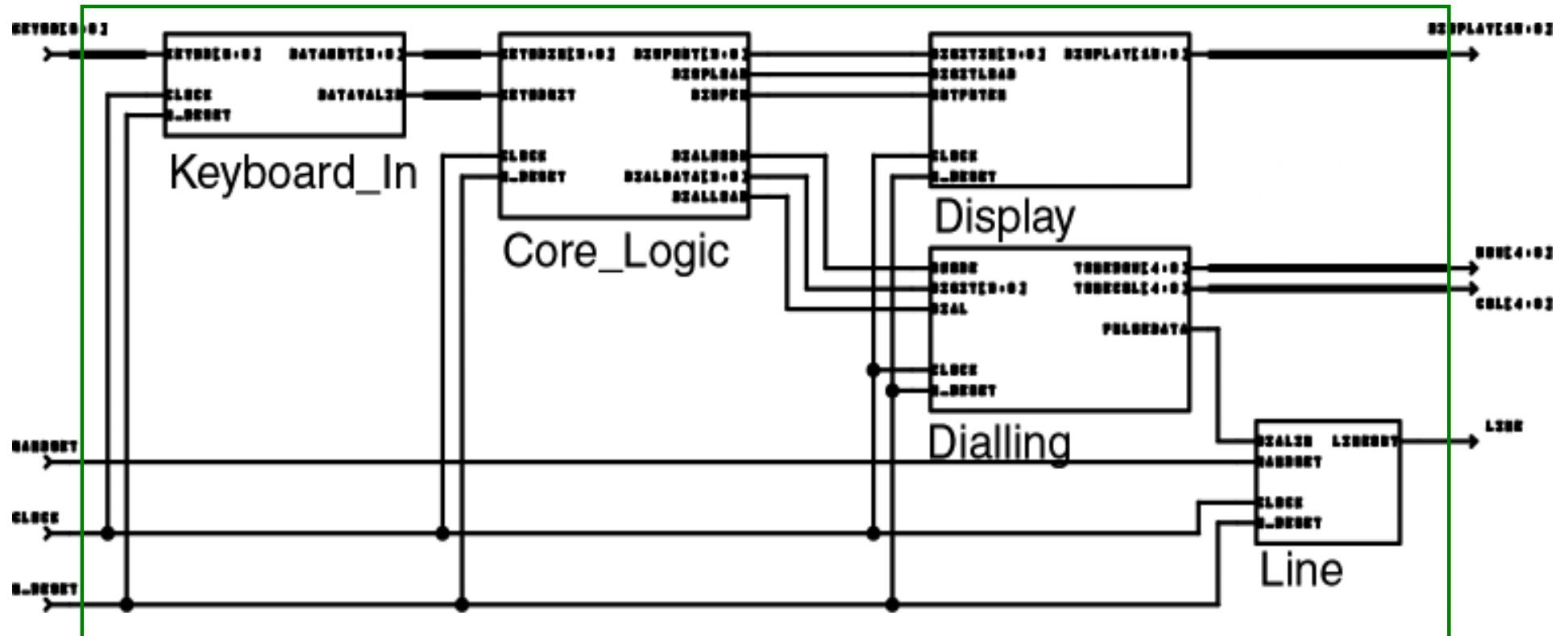
- Identify the system inputs, outputs and behaviour
  - This is in our system specification
- Example- feature telephone top design



# Hierarchical Approach- II

- We now split the system into discrete blocks which have (typically) a single function (termed *highly cohesive*) and identify the way in which they are interconnected
- An example of this is presented in **schematic 1** where the hierarchy of our 'phone has been taken down a level to give the internal details of the telephone IC
  - The system is now comprised of 5 blocks which perform a single function within the whole system (e.g. display control)
  - Could further decompose design
  - Note that the inputs and outputs are the same as the previous level (i.e. keybd[6..0] etc.)

# Feature Telephone Schematic



# Regularity

- Regularity is also useful in the design process
  - When decomposing a design attempt to form regular structures
- Regularity is particularly useful when considering the lower-level (layout issues of a design)
  - E.g. memory design which is made from a ‘tile’ of similar cells connected together
  - If we design a single cell, then all that is required is to replicate the cells to form the complete memory array
- Datapaths also benefit from regularity
  - e.g. Adder

# Design Reuse- Introduction

- Re-Use is very important, especially with fast time to market
- If we can re-use a block then it can yield a very fast produced design
  - As part of the design has already been implemented
  - Ideally, we drop the block into our top-level design
- At the extreme, we have System on a Chip (SoC) designs which are composed of blocks from multiple authors which are simply connected together to form large devices (e.g. Set Top Boxes for Television)

# Design Reuse- Requirements

- Block must be well specified and documented
  - The author may not be the end user so the full functionality of the block should be well documented (both inline within code and external documentation)
- Block generally has to be highly cohesive to be able to be re-used
  - For example, if we are designing a peripheral block for a microprocessor (e.g. Dialing block) then it would be better if the block could be design in two sections- one general block connected to a processor specific block
- Other issues (e.g. testability etc.)

# Reuse- The Tradeoff

- Functionality Vs. *Silicon Real Estate*
  - If we are reusing a design, are we only using a small set of the functionality ?- if so then this is wasteful (needs more Silicon- so higher cost)
  - E.g. we could use a 64 bit loadable counter every time we needed a counter but this would be wasteful!
  - This not only an issue when selecting a component for a design, we also have to avoid building blocks which offer *deluxe* functionality (e.g. “I’ll make this counter have inverted inputs as well as normal inputs just in case it’s needed later”)

# **Tools for Design & Implementation**

# Introduction

- Using the previous techniques, we now have a design, but how is it implemented ?
- The outcome of the previous steps is essentially a (hierarchical) block diagram with a list of blocks each with a specification
  - List of inputs and outputs
  - Specification of the internal behaviour of the block

# The Digital Design Flow

```
entity ANDGATE is
  port ( A,B: in std_logic;
        Z: out std_logic );
end ANDGATE;

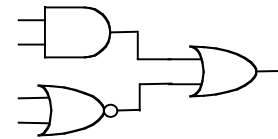
architecture MYARCH of ANDGATE is
  begin
    Z <= A and B;
  end MYARCH;
```

Hardware Description  
Language

Design



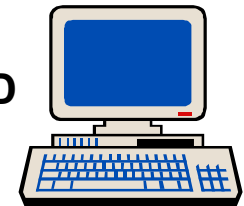
Synthesis Tool



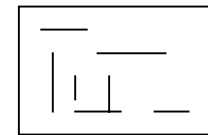
Schematic

Netlist

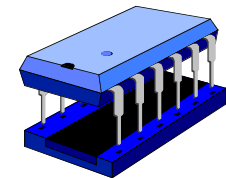
CAD



Masks



Complete  
Working IC



# HDL's- Motivation

- HDL's = Hardware Description Languages
- Isn't our Silicon solution just some highly complex program (c.f. a software implementation !)?
- So, wouldn't it be nice to write our hardware in some program type form ?
- HDL's allow us to do this !
  - We have a text based description of the desired functionality of the blocks that we're trying to design
  - No longer symbolic at the design level- **so we're no longer targeting a specific final architecture**

# HDL's- Background

- Broadly 2 different types commercially available
- Simple State Machine
  - e.g. Orcad HDL, Abel, etc.
  - With these we simply describe state machines
  - Gets a bit messy when dealing with complex designs
  - Still useful for some systems (e.g. making up a PLD)
- Full Functionality
  - e.g. VHDL & Verilog
  - Works very well for complex designs
  - We're going to look at VHDL

