PURDUE UNIVERSITY DEPARTMENT OF PHYSICS

Physics 53600

Electronics Techniques for Research

Now in PowerPoint!

Spring 2020 Semester

Prof. Matthew Jones

The usual ANNOUNCEMENT

- Obvious changes to the course:
  - No in-person lectures: you'll have to read the lecture notes yourself
  - No more labs: don't worry about it – your grade will be based on work done so far
  - Remaining assignments will try to cover topics that would have been explored in the lab
  - Second mid-term: simplest to cancel it
  - Final exam: I think it will be a 24 hour exam with written responses that can be easily sent by e-mail.
- Changes to grading scheme:
  - Old scheme: Assignments (30%) exams (40%) lab (30%)
  - New scheme: Assignments (50%) exams (25%) lab (25%)

The usual ANNOUNCEMENT

- Because there won't be any in-person lectures, you will have to read the lecture notes yourself.
- To demonstrate that you have read them, you will be required to answer *one or two simple questions* before the next lecture is posted.
- The question will probably be at the beginning and you just have to e-mail me the answer

  mjones@physics.purdue.edu
- To make this easy, please make your subject look like this:

  "PHYS53600 Lecture xx questions Your Name"
- These will be part of your assignment grade, maybe contributing 10% of your total grade.

More ANNOUNCEMENTS

- Feel free to send me questions about the lecture material if there is anything you don't understand. I'm happy to give more explanation (and I'm soooo bored.)
- Send me e-mail if you think it would be useful to arrange a time as a class to have a time where you can ask questions by video.

LECTURE 24 QUESTIONS

1. *Some time before the final exam, do you want to arrange for a Webex meeting where everyone has the opportunity to ask questions about the lecture notes or other material in the course?*

2. *For-profit companies often pay thousands of dollars per year to use FPGA design tools but the same tools are often given away for free for academic/research use.*

   Why do you suppose that is?

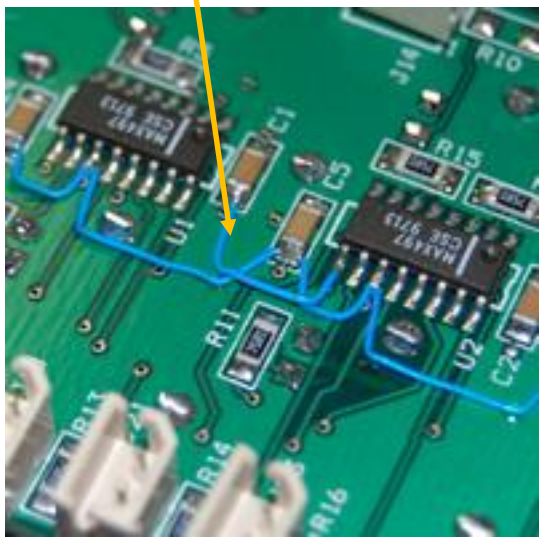
Designing with Discrete Logic Circuits

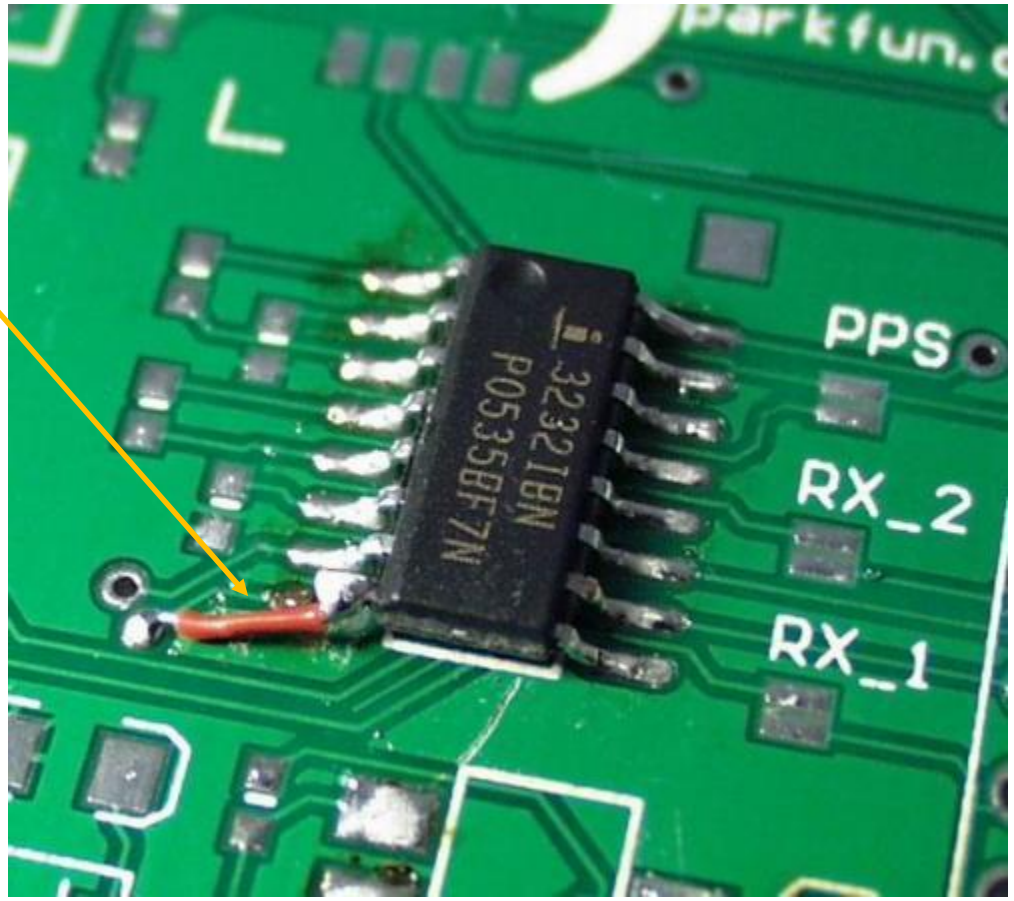
- Many of the digital circuits we have discussed so far exist as discrete integrated circuits.
- They usually (but not always) require designing and fabricating a printed circuit board to be used reliably.
- This design process can be time consuming, expensive, and error-prone.
- Simple mistakes can be fixed by cutting traces on the PCB with a razor blade, or soldering “blue wires” where new connections are needed.

Fixing Mistakes on PCB's

"Blue wire"

Blue wires





Designing with Discrete Logic Circuits

- Often it is useful to budget for two (or more) PCB production runs
  - First, a small number of prototype boards to find and fix mistakes
  - Second, a larger number of boards with a MUCH higher chance of having no mistakes
  - Hopefully you don't need a third batch, but sometimes it's inevitable
- This process can be expensive in several ways
  - Design time and expertise required
  - Cost to manufacture
  - Cost of parts
  - Cost to assemble
  - Time needed to test, debug, re-design

Alternatives to Discrete Logic Circuits

- It is of great benefit to be able to fix mistakes in logic without having to re-manufacture anything
- Several ways to achieve this:
  - Programmable Logic Arrays
  - Field-Programmable Gate Arrays
  - Microprocessors/Microcontrollers
- These are general-purpose logic devices that can be re-configured after assembly if necessary

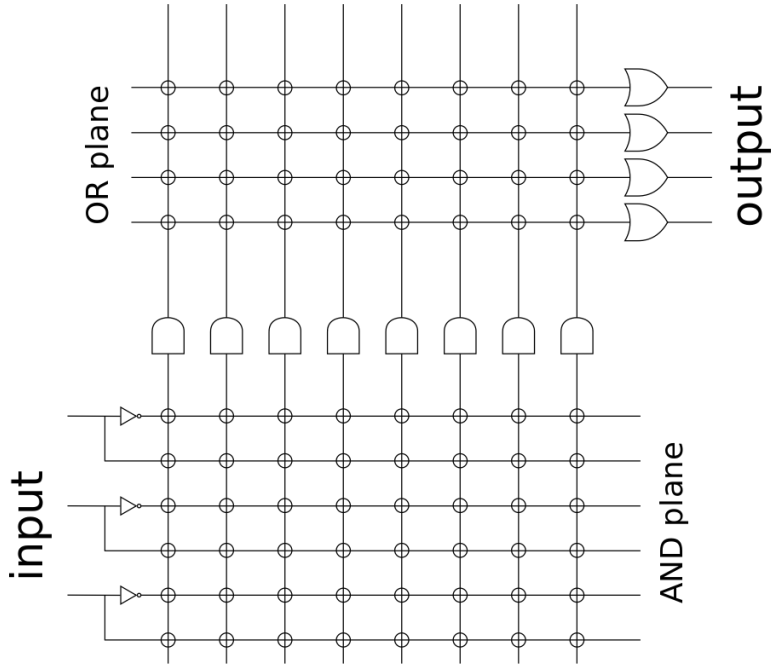
Programmable Logic Arrays

- First, consider some combinatorial logic
  - Use Boolean algebra to reduce any expression to a "sum-of-products"
  - Example: three-input "exclusive or"

$$
\begin{aligned}
q &= x \oplus y \oplus z \\
&= (x \cdot \bar{y} + \bar{x} \cdot y) \oplus z \\
&= (x \cdot \bar{y} + \bar{x} \cdot y) \cdot \bar{z} + \overline{(x \cdot \bar{y} + \bar{x} \cdot y)} \cdot z \\
&= x \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot y \cdot \bar{z} + \overline{(x \cdot \bar{y})} \cdot \overline{(\bar{x} \cdot y)} \cdot z \\
&= x \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot y \cdot \bar{z} + (\bar{x} + y) \cdot (x + \bar{y}) \cdot z \\
&= x \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot y \cdot \bar{z} + \bar{x} \cdot \bar{y} \cdot z + x \cdot y \cdot z
\end{aligned}
$$

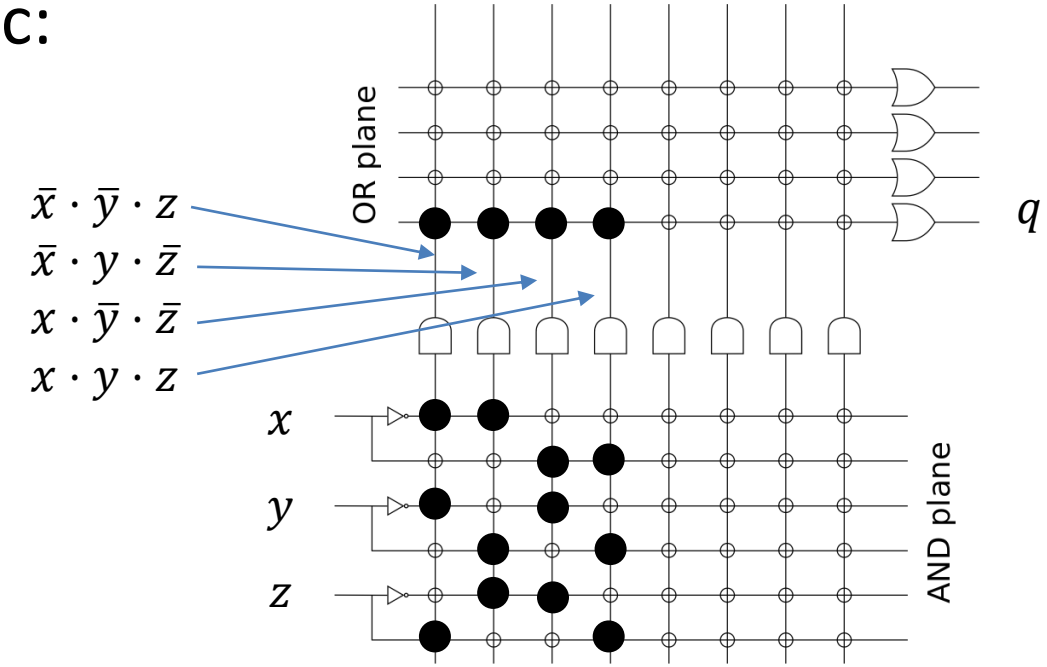
Programmable Logic Arrays

- The original PLA's (developed in the 1970's) had arrays of AND and OR operations with optional inverters at each input.



Programmable Logic Arrays

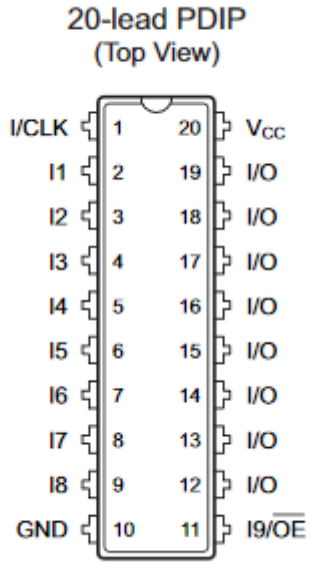
- The logic could be implemented by “burning fuses” that removed parts of the metal interconnect layer in the circuit.
- The connections left behind implement the desired logic:



Programmable Logic Arrays

- These devices could only be programmed once
- Still, if you got it wrong you could simply replace the component on a PCB with a new one
- Newer PLA's implemented combinatorial logic features in their available resources (called "macrocells")
- These require software to generate the physical configuration needed to implement the desired logic
- They also require a programmer to physically "burn in" the design.
- Then they are ready for use…
- Newer designs are "Electrically Erasable" and can be reconfigured multiple times.

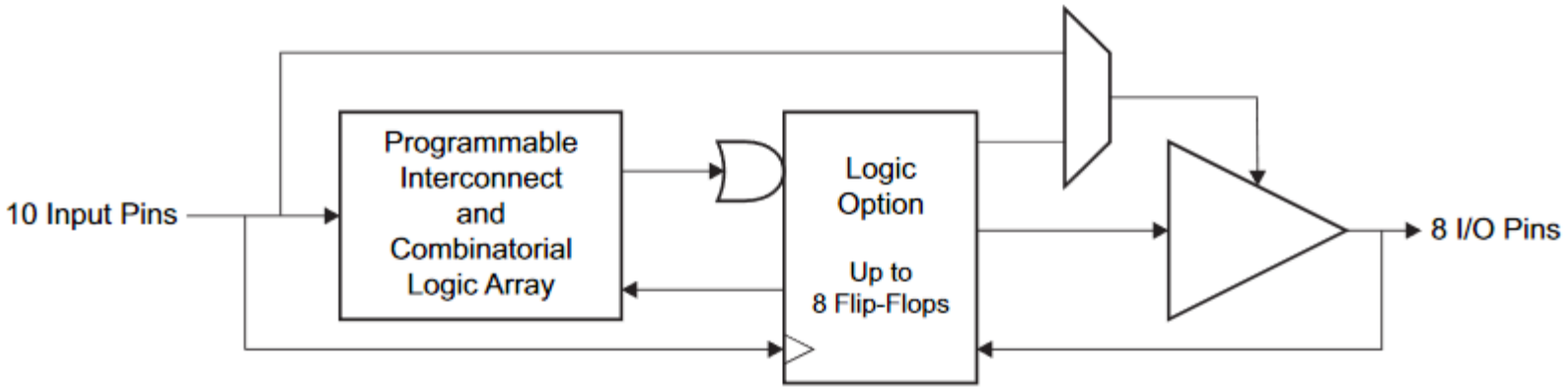
Example: Atmel ATF16V8B



Things to notice:

1. Power (V_{CC} and GND) pins are in the usual place
2. There is a dedicated CLK input pin
3. There is a dedicated output-enable pin ($\overline{\text{OE}}$) which is active low.
4. There are 8 inputs and 8 outputs
5. They are INEXPENSIVE! This one costs only 86¢

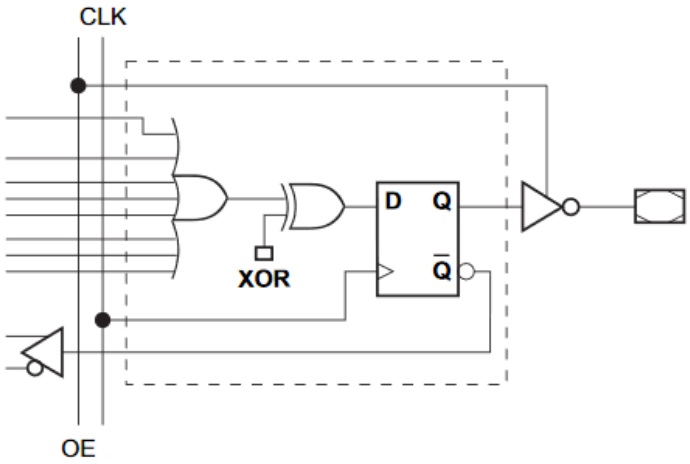
Figure 2-1. Block Diagram

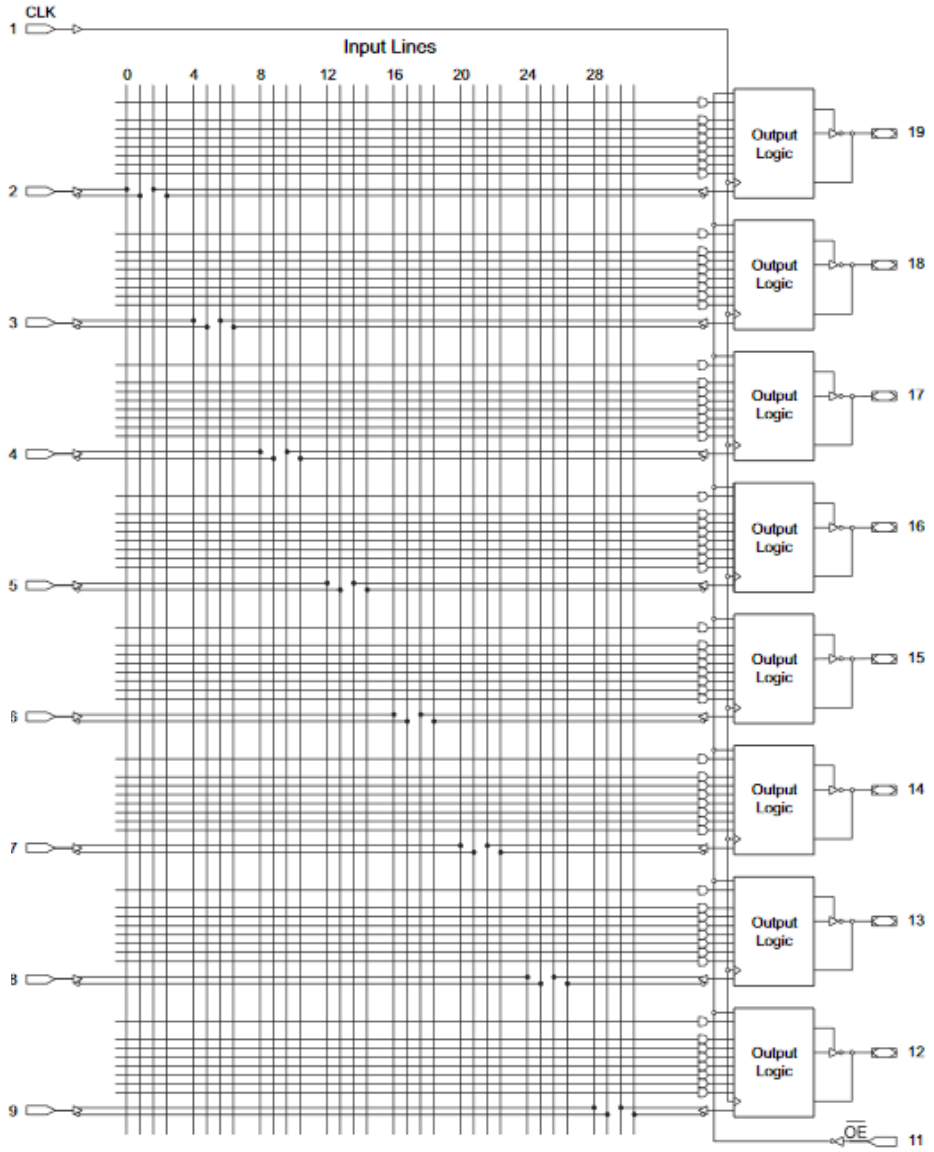


Example: Atmel ATF16V8B

- Logic array:

This part is basically the same as in the previous example. The main difference is that now there is an “output logic” macrocell:





Example: Atmel ATF16V8B

- The design software is usually free to download and install.
- The programming interface is standard enough that there are many third-party programmers available

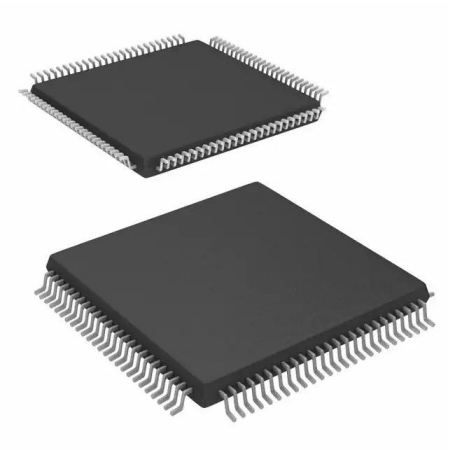
- Usually interface to a PC using USB or some other interface.

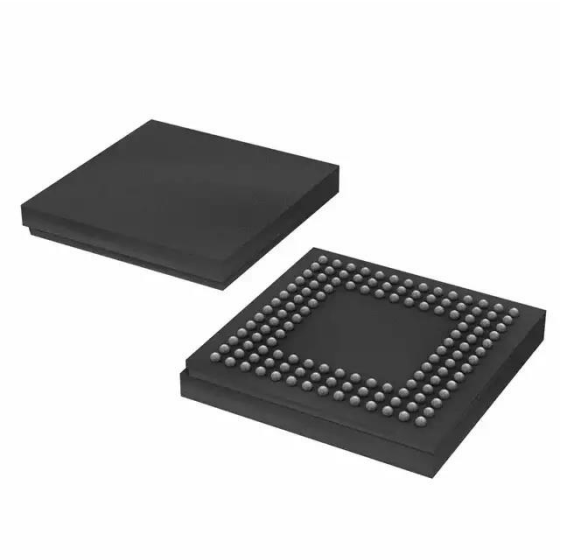
Complex Programmable Logic Devices

- The same idea has been extended to devices that have much more complex logic resources
  - Large numbers of inputs/outputs
  - Dedicated block memory
  - Large numbers of macrocells with sophisticated combinatorial/sequential logic
  - Dedicated networks for routing of clock signals
- Many are “in-circuit” reprogrammable

Example: Xilinx CoolRunner-II Series

- These often have MANY inputs/outputs
  - They typically can have of the order of 100 pins
  - Various types of packages (leads on all sides, or ball grid arrays)





Example: Xilinx CoolRunner-II Series

- Features: XC2C128 example ($9.05 each)
  - Separate core voltage (1.8 volts) and I/O voltages (1.5, 1.8, 2.5, 3.3 volts)
  - Dedicated pins for programming/debugging
  - Sophisticated macrocell architecture:

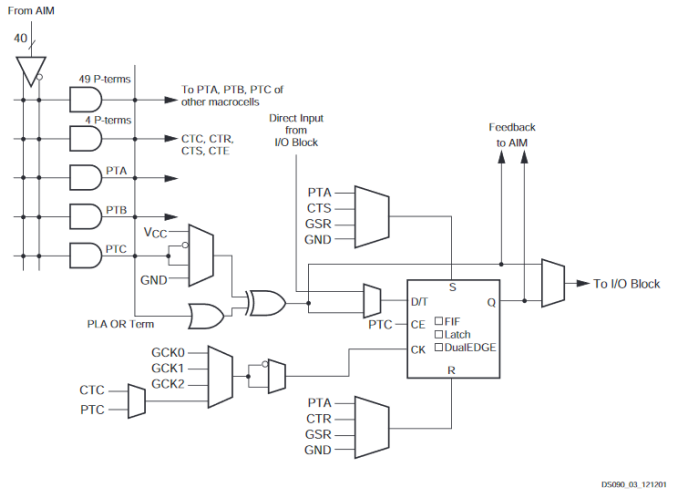
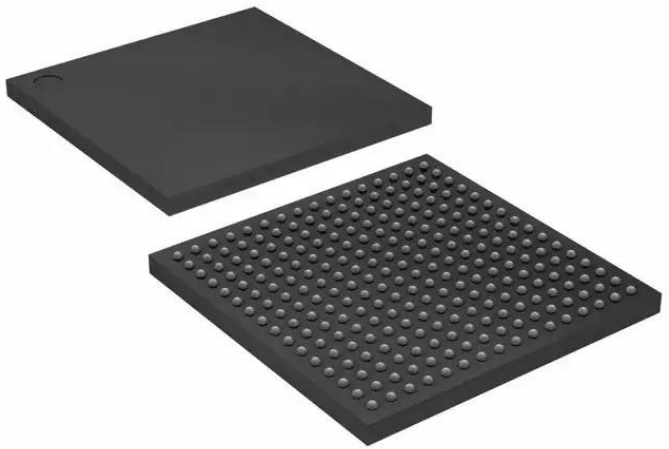


Figure 3: **CoolRunner-II CPLD Macrocell**

Field-Programmable Gate Arrays

- These are sort of like CPLD's, except that they download their configuration each time power is applied
- These can have very sophisticated resources that are available for use:
  - High speed serial transceivers
  - Clock synthesizer circuits
  - Large block memory resources
  - Large numbers of macrocells
  - Dedicated clock routing fabric

FPGA Example: Xilinx Spartan-3 series



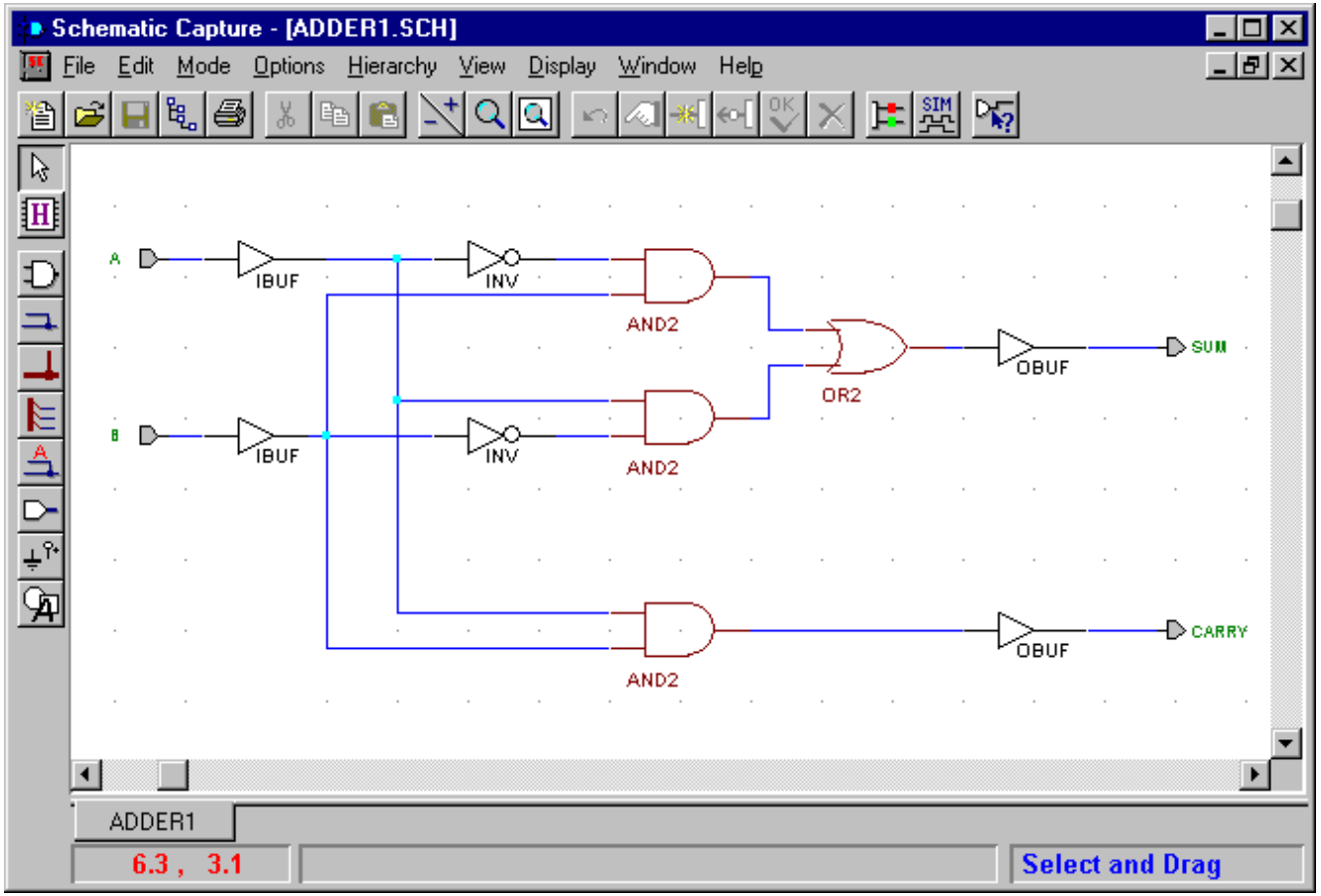
- Example: XC3S400-4FTG256C
  - 173 I/O pins
  - Core voltage: 1.2 volts
  - 896 logic blocks
  - 8064 logic elements
  - 400,000 logic gates
  - 294,912 bits of RAM
  - Costs $34.95 each

Hardware Synthesis Tools

- Now we have a new problem:
  - The resources are so abundant and sophisticated that it is essentially impossible for a human to specify each and every interconnection
- Instead we rely on synthesis tools:
  - Emphasis on “describing” what we want the hardware to accomplish
  - This can be done using schematics and elementary logic blocks
  - More commonly done using a “Hardware Definition Language”
- Rely on vendor-supplied software to translate the hardware “description” into configuration data

Hardware Synthesis Tools

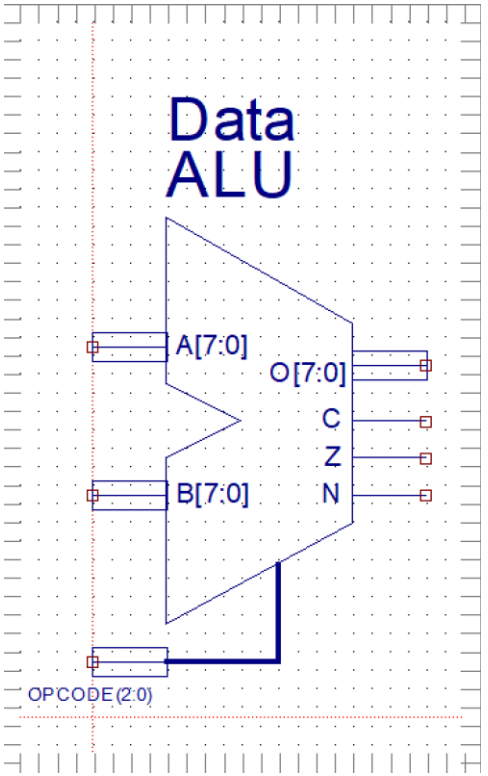
- Schematic capture using basic logic elements:



Hardware Synthesis Tools

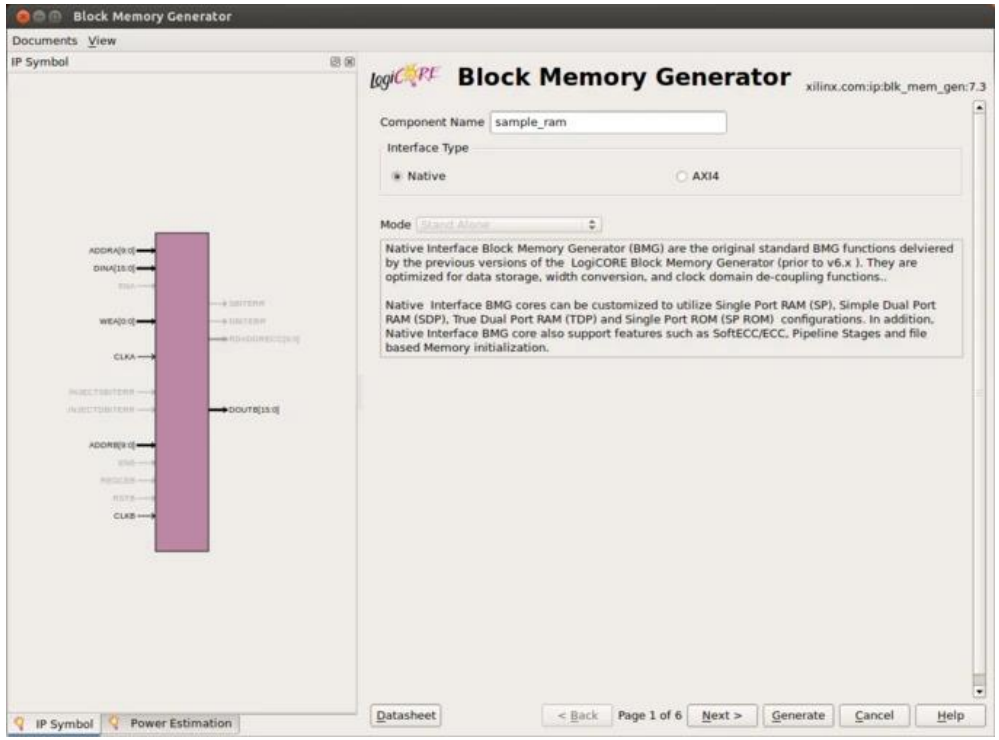
- You can also use libraries of sophisticated design elements:

For example, this is an Arithmetic Logic Unit (ALU) which could, for example, add or subtract the two 8-bit inputs based on the OPCODE inputs.



Hardware Synthesis Tools

- You can also use libraries of sophisticated design elements which can be customized using the software tools provided.



This is an example of a block memory interface. If you look carefully you can see that it has the same signals as the dual-port RAM we discussed previously (address bus, data bus, write enable, et…)

Hardware Synthesis Tools

- The important thing to learn is that these tools are used to DESCRIBE how the design should function
- They are not usually used to SPECIFY how the design should function
- There are sophisticated optimization algorithms that are used to translate the design into the actual implementation

Hardware Constraints

- Equally important is the way in which hardware constraints are specified
- These can include
  - Which physical pins map to the signals in the schematic description
  - Timing constraints
    - Input constraints: when does data arrive at the inputs?
    - Timing constraints: frequencies of internal clock signals
    - Output constraints: relative timing of outputs

Hardware Constraints

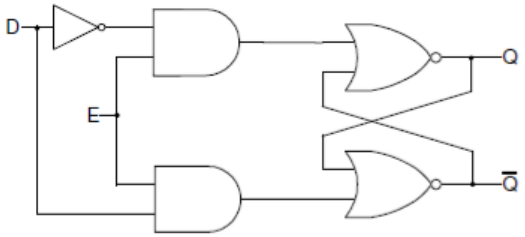
- The synthesis tools typically perform the following steps:
  - Logic reduction and mapping to resources
  - Placement of resources in configurable logic blocks
  - Routing of signals between the resources
- This exercise usually turns into a big minimization problem
  - Any “solution” that satisfies the constraints is considered acceptable
- Relies on detailed simulation and measurement of device properties

Hardware Definition Languages

- Eventually, describing everything using schematics is no longer efficient
- Today, we usually use high-level languages to describe the hardware we want to implement
- Very common examples: Verilog and VHDL
- They resemble programing languages, but they are used to describe hardware, not algorithms.

Hardware Definition Languages

- The HDL gives the synthesis tools “hints” that can be easily mapped into hardware resources
- Example: A simple D latch:



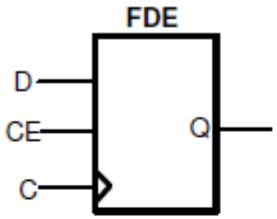
Enable	D	Q	$\overline{Q}$
0	0	Latch	
0	1	Latch	
1	0	0	1
1	1	1	0

D-latches can be modeled in behavioral modeling as shown below.

```
architecture behavior of D_latch is begin
process (D, Enable) begin
      if (Enable = '1') then
            Q <= D;
            Qbar <= not(D);
      end if;
end process;
end behavior;
```

Hardware Definition Language

- Many design elements rely on clock signals to synchronize the sampling of data
- Example: An edge-sensitive D-type flip-flop:




```
architecture behavior of D_ff_with_ce is begin
process (clk) begin
      if rising_edge(clk) then
            if (ce = '1') then
                  Q <= D;
            end if;
      end if;
end process;
end behavior;
```

Hardware Definition Language

- Here's a more complex example showing the description of block memory:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ram_infer IS
   PORT
   (
      clock: IN   std_logic;
      data:  IN   std_logic_vector (31 DOWNTO 0);
      write_address:  IN   integer RANGE 0 to 31;
      read_address:   IN   integer RANGE 0 to 31;
      we:    IN   std_logic;
      q:     OUT  std_logic_vector (31 DOWNTO 0)
   );
END ram_infer;
ARCHITECTURE rtl OF ram_infer IS
   TYPE mem IS ARRAY(0 TO 31) OF std_logic_vector(31 DOWNTO 0);
   SIGNAL ram_block : mem;
BEGIN
   PROCESS (clock)
   BEGIN
      IF (clock'event AND clock = '1') THEN
         IF (we = '1') THEN
            ram_block(write_address) <= data;
         END IF;
         q <= ram_block(read_address);
      END IF;
   END PROCESS;
END rtl;
```

Interface description

Architecture description

Practical Details

- So suppose you have a problem that can be conveniently solved using an FPGA to implement all the logic
- It can be very time consuming and expensive to design a printed circuit board with an FPGA on it
  - Usually needs at least a couple years of practical electrical engineering experience
- But you can often BUY something that does most, if not all of what you need
  - Then you can program it using your custom logic design

Demonstration Boards

- Many companies sell “demonstration boards” that can be adapted to serve a specific application:

DIGILENT®

1300 Henley Court
Pullman, WA 99163
509.334.6306
www.digilentinc.com

Cmod™ Board Reference Manual

Revised October 26, 2012
This manual applies to the Cmod rev. D

Overview

Cmod boards combine a Xilinx CPLD, a JTAG programming port, and power supply circuits in a convenient 600-mil, 40-pin DIP package. Cmods are ideally suited for breadboard or other prototype circuit designs where the use of small surface mount packages is impractical. All Cmod boards include:

The Cmod board.

Features include:

- A single 3.3V supply voltage (voltage regulation provided on Cmod board where required);
- Adequate bypass capacitance on all CPLD voltage supply pins;
- All available user I/O signals brought out to DIP pins;
- Once programmed, CPLD designs are non-volatile;
- Designs can easily be ported between Cmods using different CPLD device families;
- All Cmod boards are compatible with the free Xilinx WebPack tools.

This one costs $25.00

Demonstration Boards

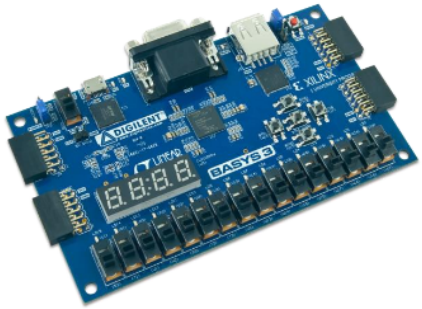
Basys 3™ FPGA Board Reference Manual

Revised April 8, 2016
This manual applies to the Basys 3 rev. C

Overview

The Basys 3 board is a complete, ready-to-use digital circuit development platform based on the latest Artix®-7 Field Programmable Gate Array (FPGA) from Xilinx®. With its high-capacity FPGA (Xilinx part number XC7A35T-1CPG236C), low overall cost, and collection of USB, VGA, and other ports, the Basys 3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs, and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits.

The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance, and more resources than earlier designs. Artix-7 35T features include:



The Basys 3.

- 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)
- 1,800 Kbits of fast block RAM
- Five clock management tiles, each with a phase-locked loop (PLL)
- 90 DSP slices
- Internal clock speeds exceeding 450MHz
- On-chip analog-to-digital converter (XADC)

This cost $149.00

Demonstration Boards

Xilinx - Adaptable. Intelligent. › Boards › Xilinx Kintex-7 FPGA KC705 Evaluation Kit

Xilinx Kintex-7 FPGA KC705 Evaluation Kit

Price: $1,695
Part Number: EK-K7-KC705-G
Lead Time: 6 Weeks
Device Support: Kintex-7

XILINX.

Buy Online ⌄ | Contact Sales ⌄

- This can be (but doesn't have to be) plugged into the PCIe bus in a PC.
- It provides high-speed network interfaces that connect directly to the FPGA.

Dedicated FPGA Modules

- You often have to decide whether it is more economical (in terms of time and money) to design and build yourself or to just buy already existing hardware.
- Especially if you have to pay your own electrical engineer, the net expense is often lower when you buy from a company.