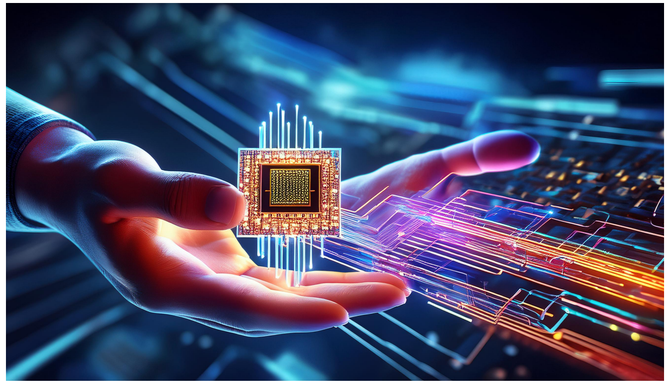
18.10.24 14:52

CSCI 250
Introduction to Computer Organisation
Lecture 3: CPU Architecture II



Jetic Gū

2024 Fall Semester (S3)

Overview

- Architecture: von Neumann
- Textbook: LCD: 9.7; CO: 2.1
- Core Ideas:
    1. Data bus ports in LogicWorks
    2. Lab 3 stuff

Changes in CSCI250

- Original Plan: more VHDL heavy implementations
- Reality:
  - The intended software (Vivado, ISE) are not properly installed on the college computer.
  - The old software LogicWorks, as I’ve found out, has very very limited VHDL support, insufficient for us to design a full CPU efficiently.
- Conclusion
  - We’ll have to draw a lot of circuits. I of course will make sure there’s at least one way of designing every component within reasonable time. This will mean that the final CPU design will be greatly simplified.

Future

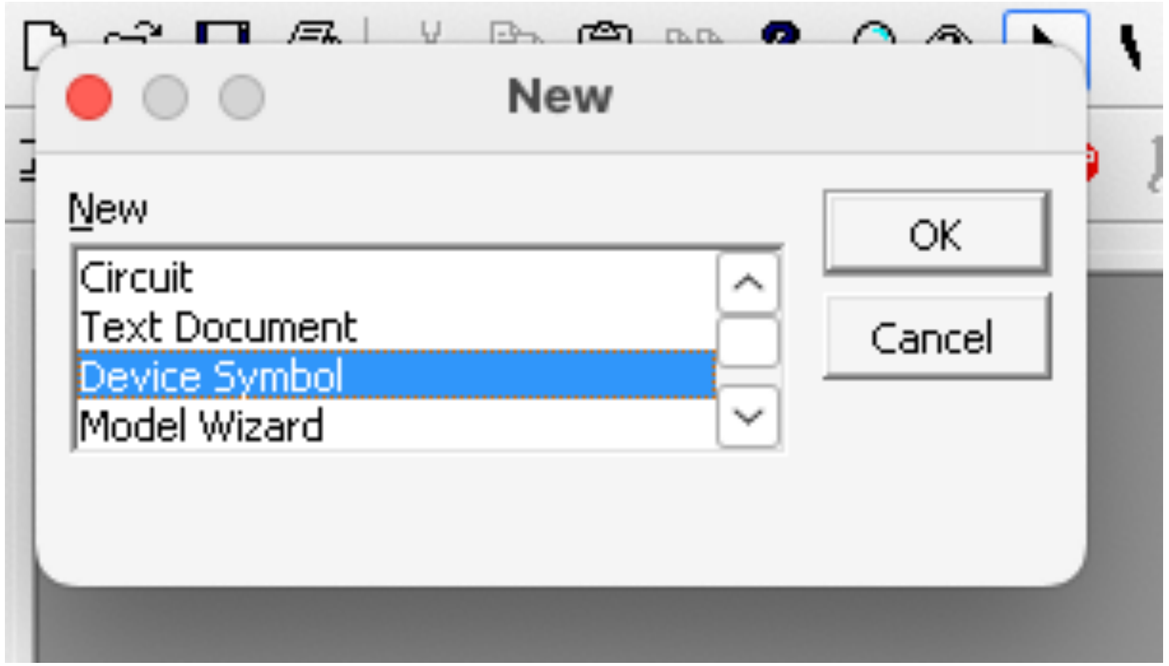
Data Bus Ports

Why no Bus Ports by Default?

- There’s no bus ports on any microchips, just individual 1bit ports for obvious reasons.
- LogicWorks’ built-in bus ports for VHDL is very difficult to use, you are very restricted when it comes to what can be connected.
- Fortunately, it can be resolved using custom bus ports. In CSCI250 Lab 3, you will need 2 bus ports: D16 and D3.

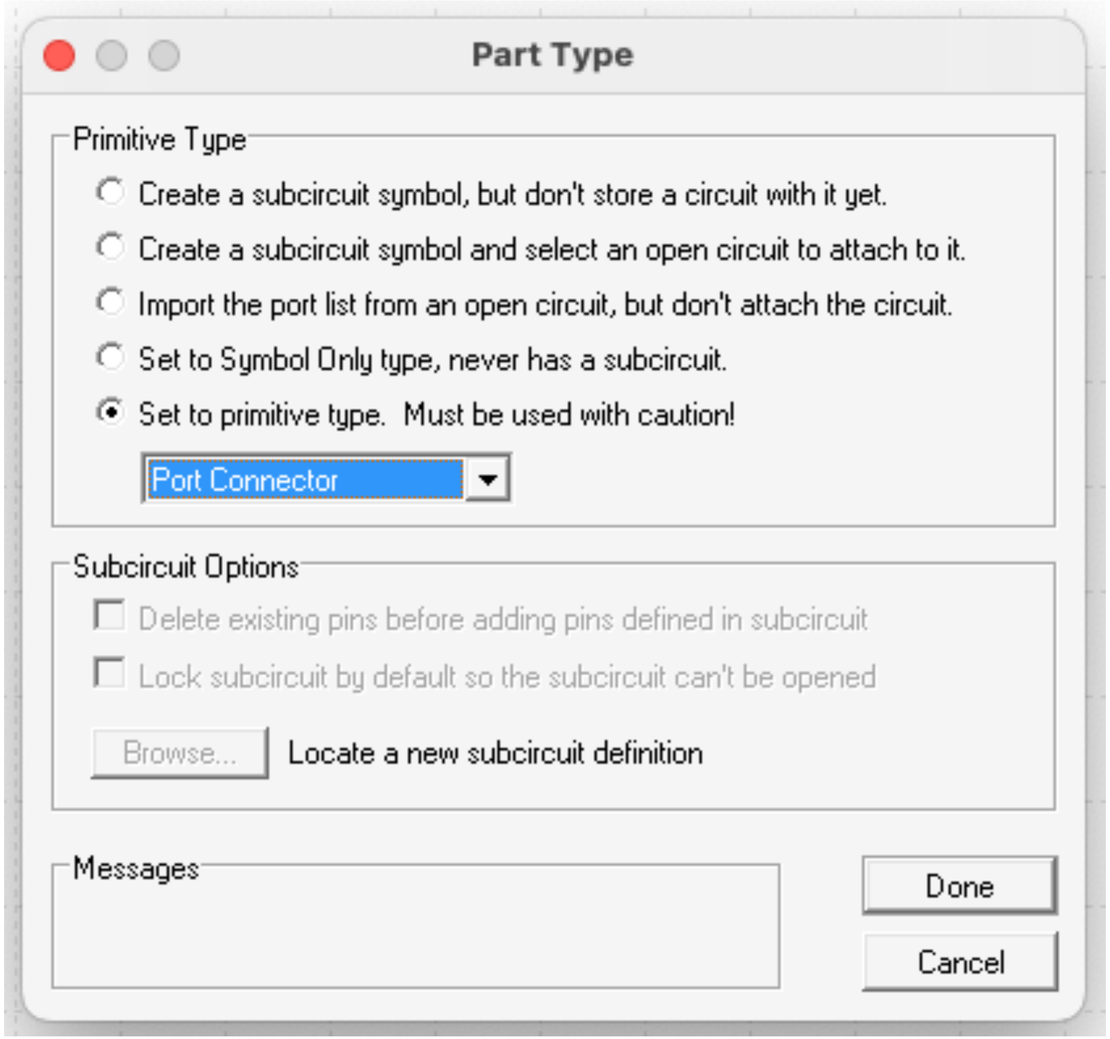
Concept

Implementing a D16 Port



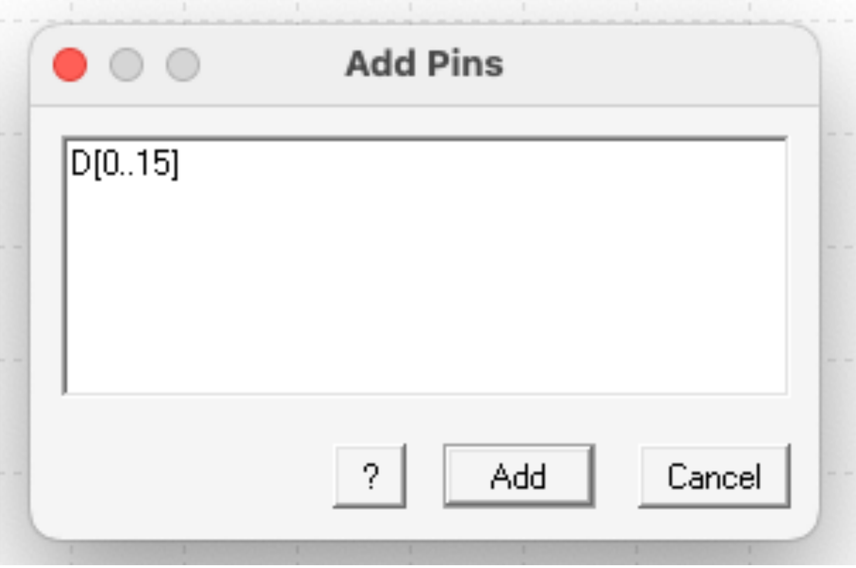
1. **Go straight to** `File-New`**, select** `Device Symbol`

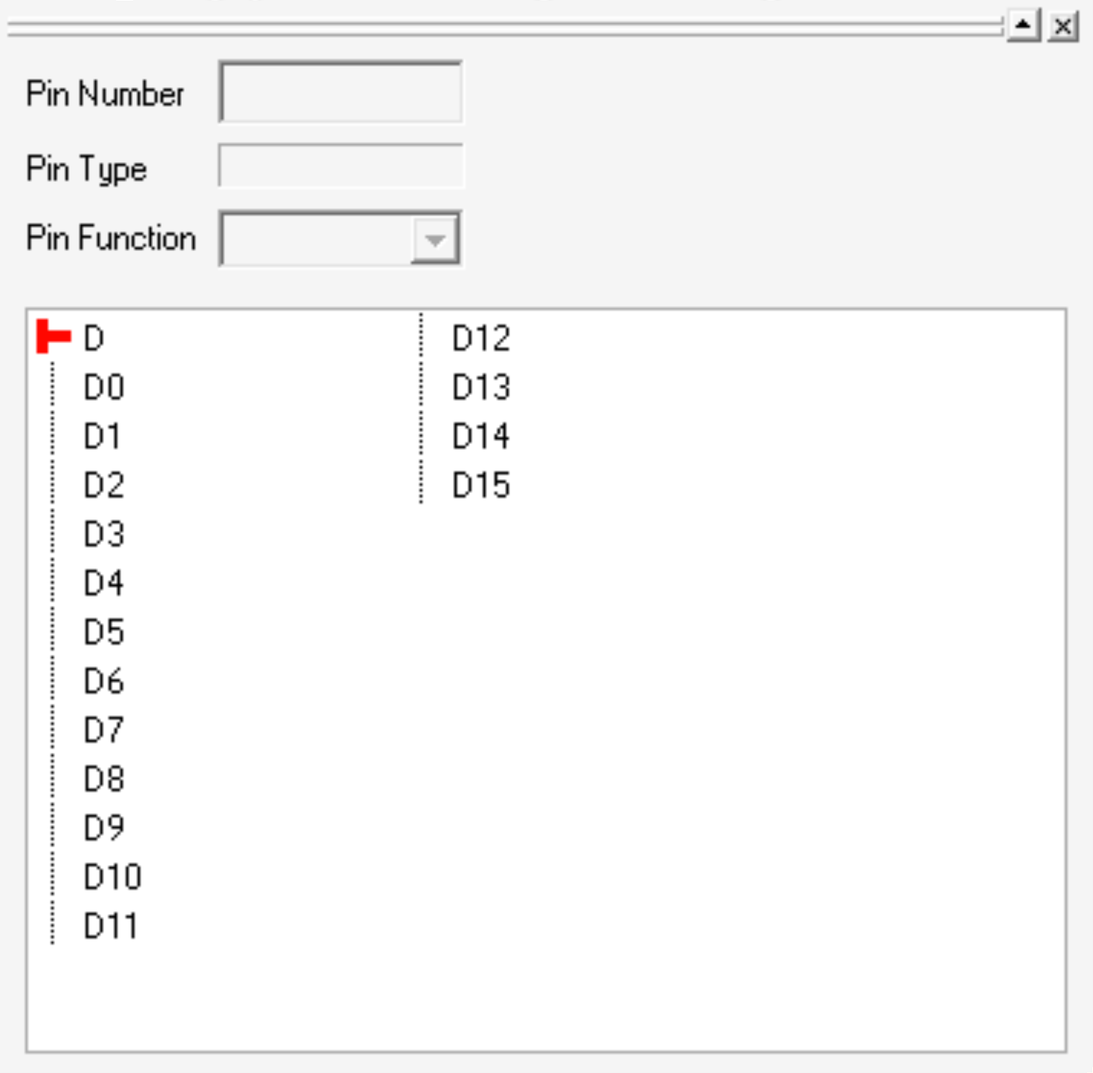
Implementing a D16 Port



2. **Go to** `Options - Subcircuit and Part Type`**, select** `Set to primitive type`**, under the scroll field, select** `Port Connector`

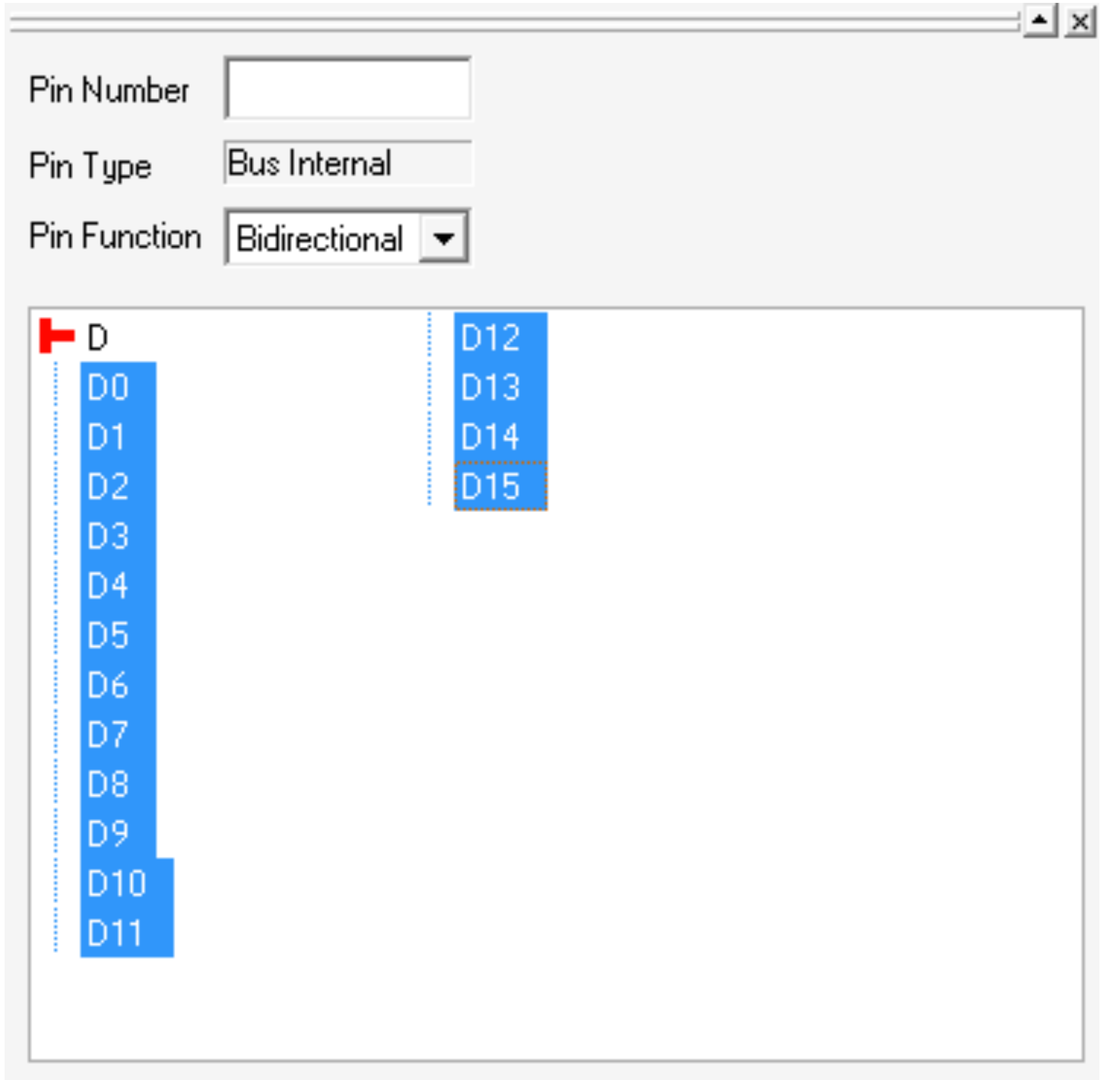
Implementing a D16 Port





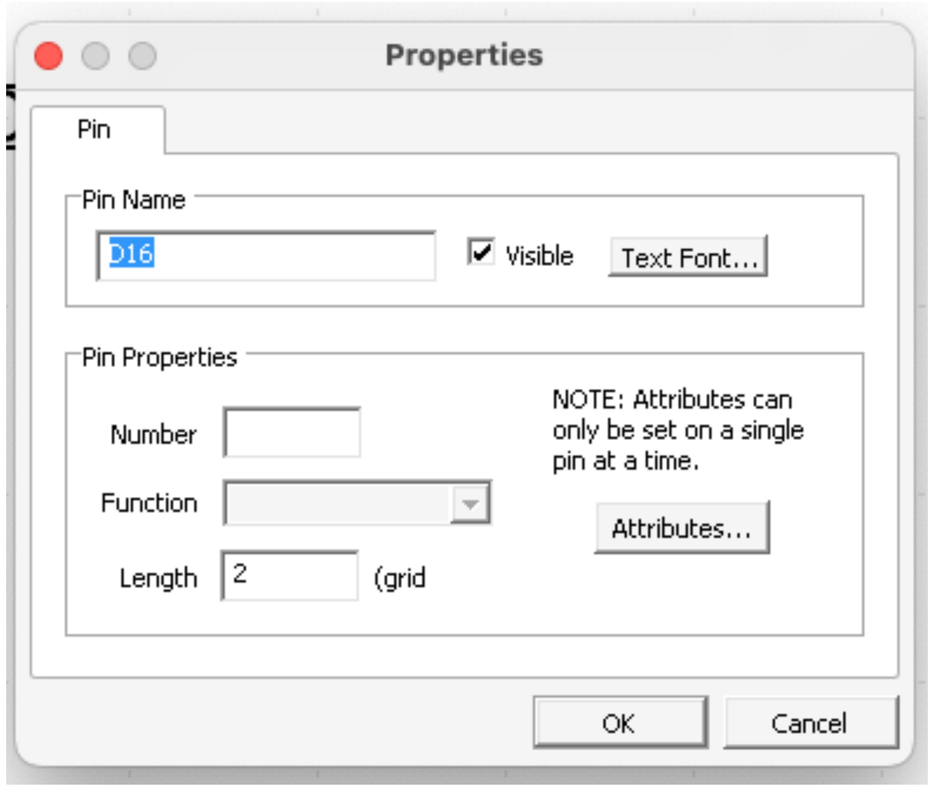
4. **Go to** `Options - Add Pins`**, type** `D[0..15]`**, press** `Add`. You should the have a pin list like this.

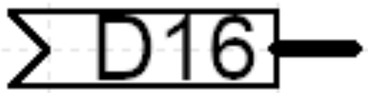
Implementing a D16 Port



5. **Select pins** `D0-D15`**, change the pin function to** `Bidirectional`

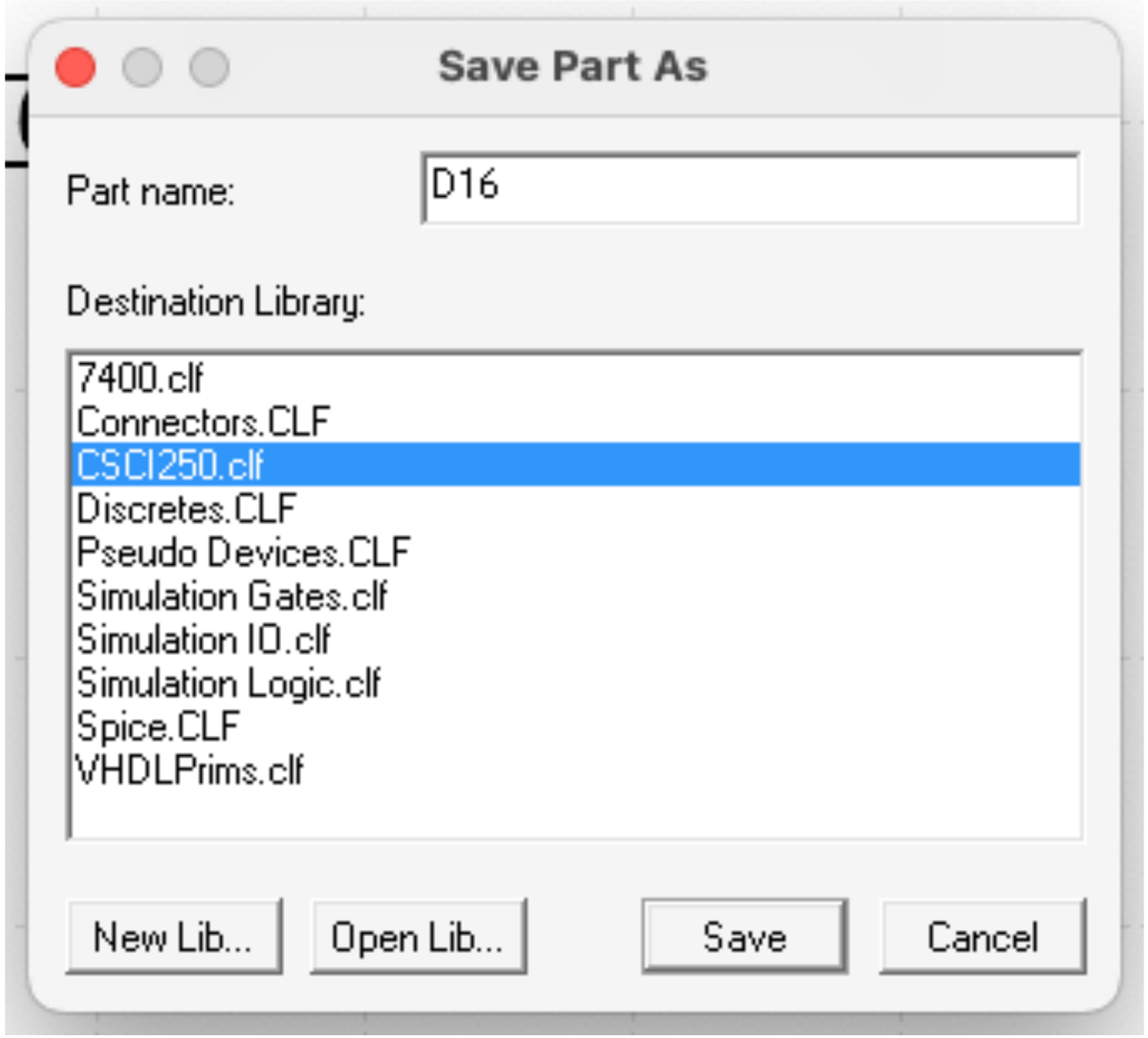
Implementing a D16 Port





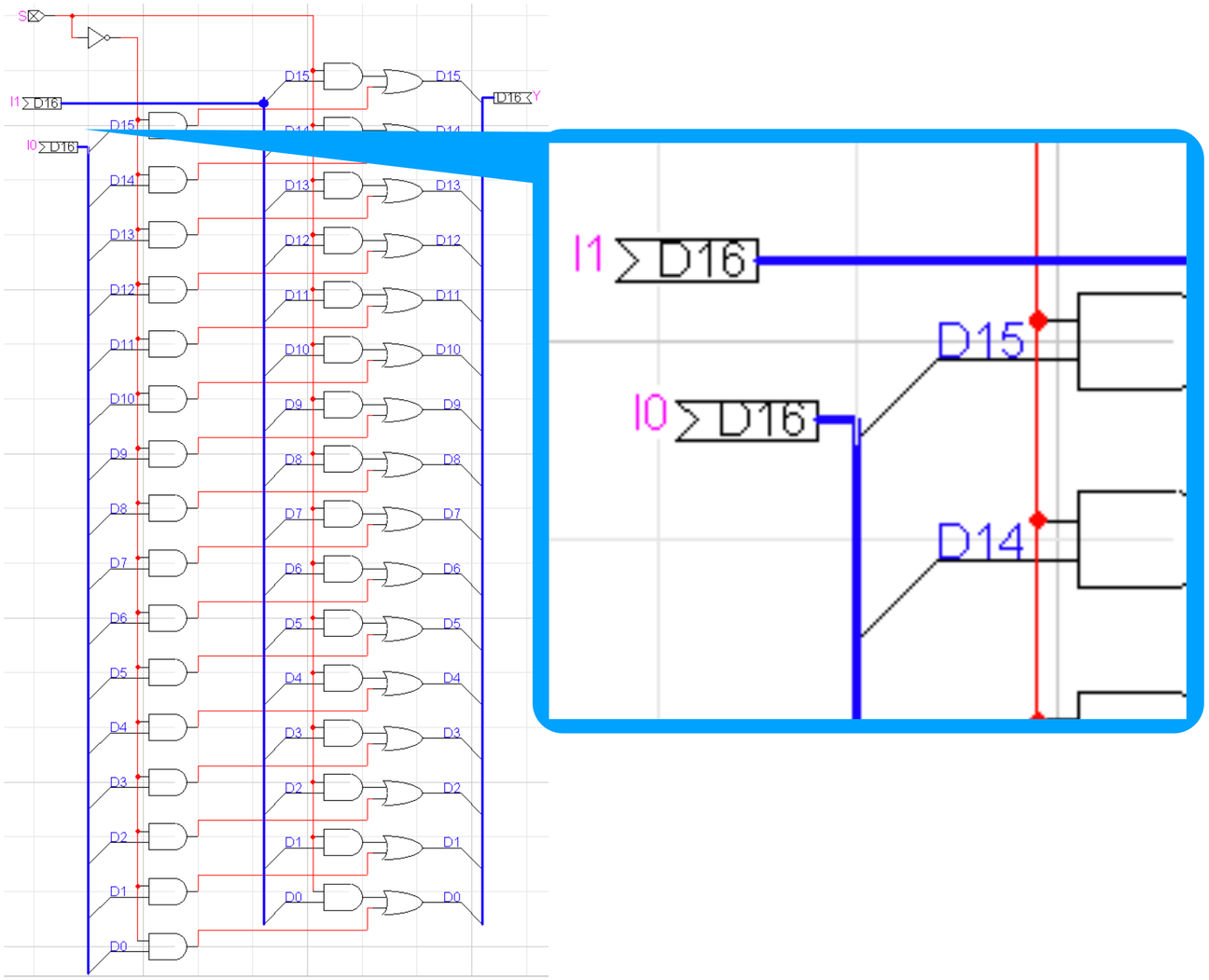
6. Draw a symbol like on the left, add a `Bus pin`. Double click the letter `D`, change the name to `D16`

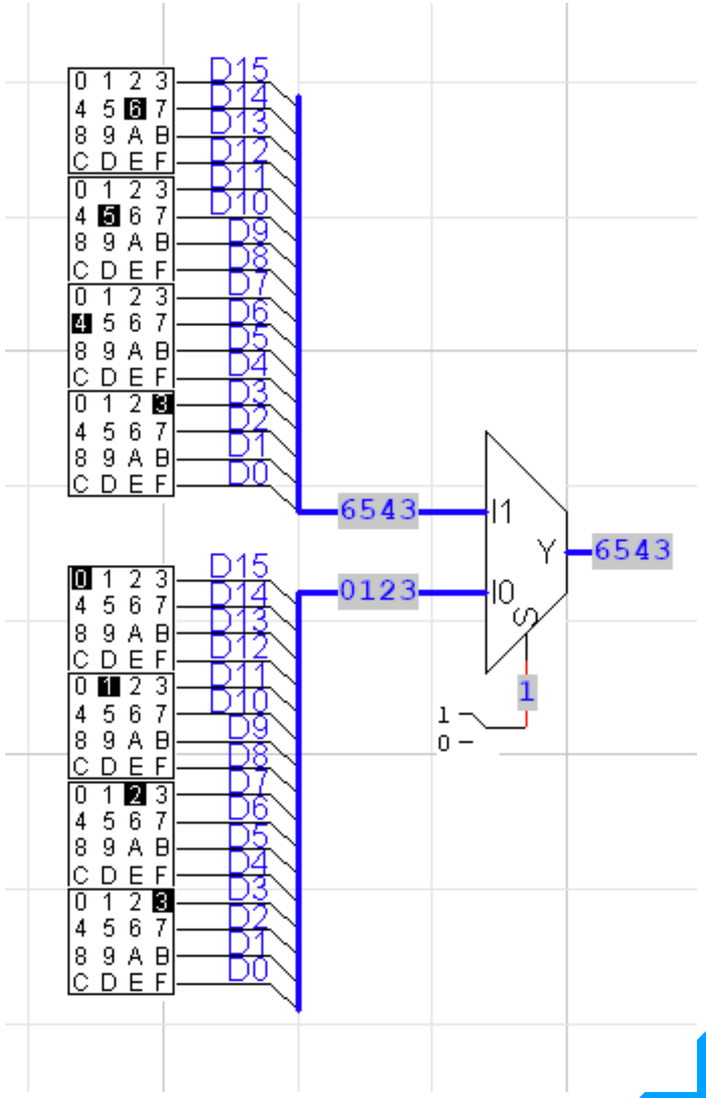
Implementing a D16 Port



7. Save the component as `D16` in your library.

Implementing a D16 Port





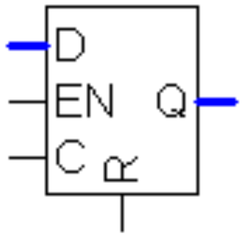
7. Now you can use it just like any other ports (you will need to name them), but it's just like a normal `D0..15` bus.

Technical

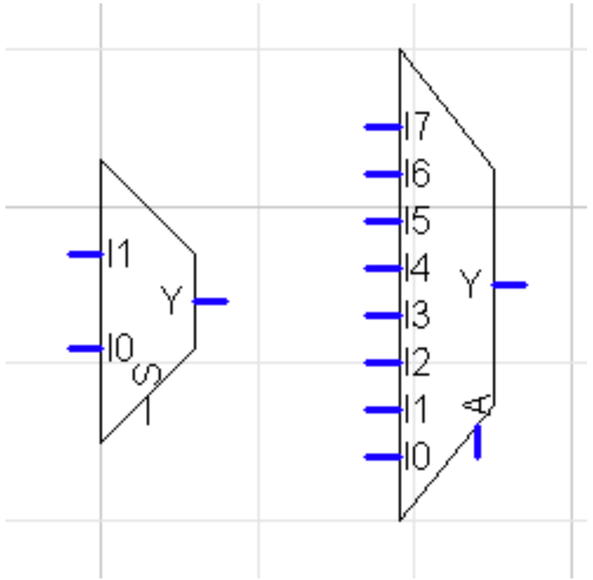
Lab 3 Stuff

P2
Lab 3 Stuff

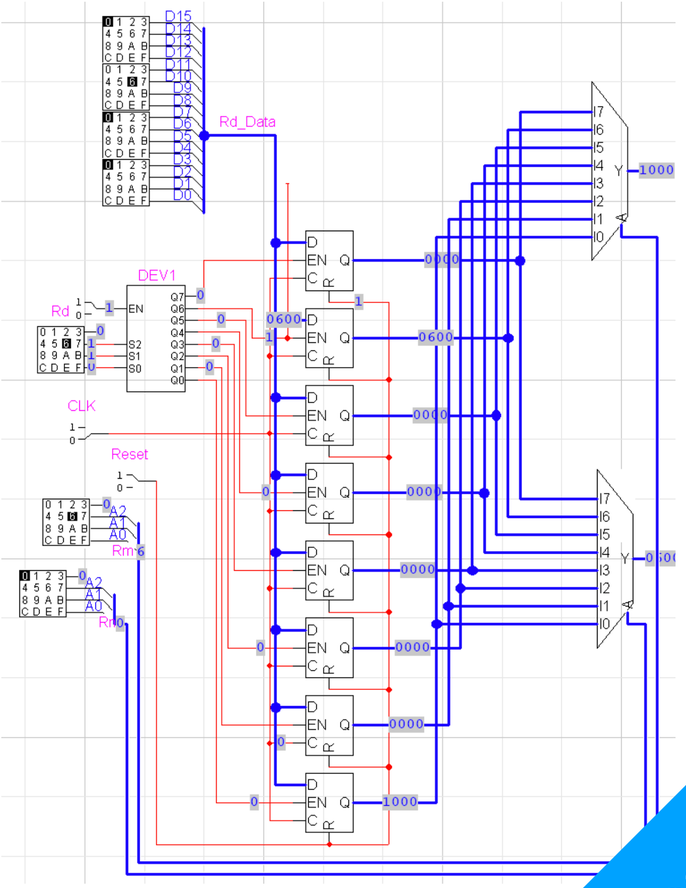
Register Array



16bit Reg



Multiplexers



Register Array

Technical

P2
Lab 3 Stuff

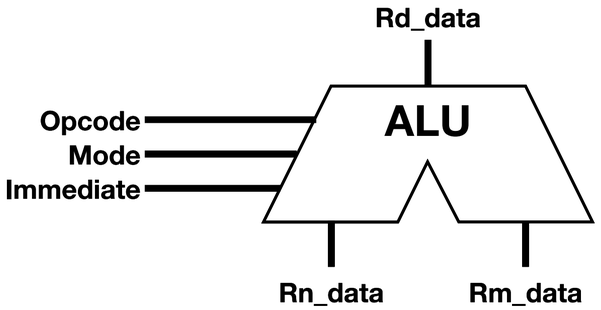
ALU

- ARM Architecture has different opcodes for Logical Unit operations and Arithmetic Unit operations
- The logical unit operations are called *Data-processing* instructions
- The arithmetic operations are called *Shift (immediate), add, subtract, move, and compare* instructions

1. https://developer.arm.com/documentation/ddi0406/c/Application-Level-Architecture/Thumb-Instruction-Set-Encoding/16-bit-Thumb-instruction-encoding/Data-processing?lang=en
2. https://developer.arm.com/documentation/ddi0406/c/Application-Level-Architecture/Thumb-Instruction-Set-Encoding/16-bit-Thumb-instruction-encoding/Shift--immediate---add--subtract--move--and-compare?lang=en

P2
Lab 3 Stuff

ALU



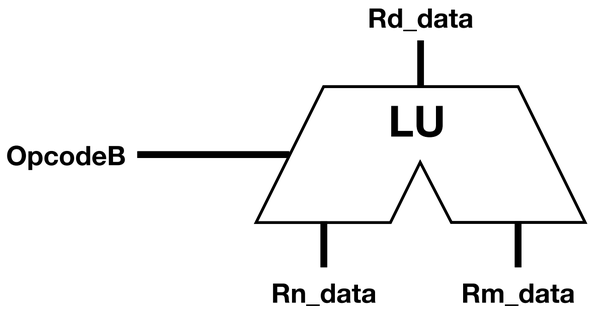
- Some CPU architecture uses just one Arithmetic Logical Unit design. For ARM, I find it easier to separate the AU and LU since their opcodes are formatted quite differently

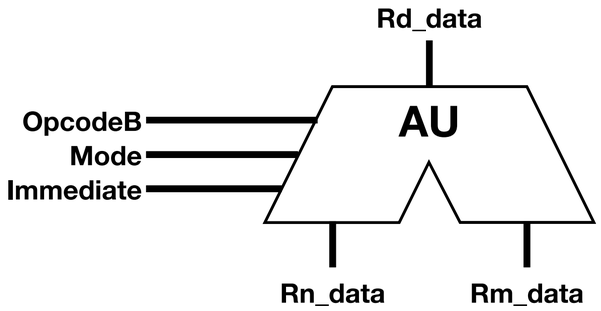
1. https://developer.arm.com/documentation/ddi0406/c/Application-Level-Architecture/Thumb-Instruction-Set-Encoding/16-bit-Thumb-instruction-encoding/Data-processing?lang=en
2. https://developer.arm.com/documentation/ddi0406/c/Application-Level-Architecture/Thumb-Instruction-Set-Encoding/16-bit-Thumb-instruction-encoding/Shift--immediate---add--subtract--move--and-compare?lang=en

Technical

P2
Lab 3 Stuff

ALU





- `OpcodeB` is taking different portions of the instruction for the AU and LU
  For the **AU**, it is the 13-11 digits of the instruction.
  For the **LU**, it is the 9-6 digits of the instruction.
  We'll discuss instruction interpretation implementation in Lecture 4

- `Mode` is specific to **AU**'s adder-subtractor
  It is the 10-9 digits of the instruction. In the ARM specification manual, it is combined with 13-11 to makeup `Opcode`, but to avoid confusion here we name them separately.

- `Immediate` is a value that's embedded into the instruction, you need to pad zeroes in order to use it

Technical

ALU

- Because of the restrictions found in LogicWorks' VHDL implementation, it will be difficult and buggy to use AU and LU designed using VHDL. However, for Lab 3, I'll let you decide what to do.

- Option 1: Use VHDL

  - You won't be able to easily integrate it with the rest of your register array and memory module

  - You can use `when` expressions to make things simple

P2
Lab 3 Stuff

VHDL When Expressions

```
Z <= I0 after 5ns when S0 = '0' and S1 = '0' else
     I1 after 5ns when S0 = '0' and S1 = '1' else
     I2 after 5ns when S0 = '1' and S1 = '0' else
     I3 after 5ns when S0 = '1' and S1 = '1' else
     "00000000" after 5ns;
```

- For example, this is the code for an 8bit 4-to-1 MUX. It is fully concurrent.

Technical

P2
Lab 3 Stuff

VHDL When Expressions

```
Z <= I0 after 5ns when S = "00" else
     I1 after 5ns when S = "01" else
     I2 after 5ns when S = "10" else
     I3 after 5ns when S = "11" else
     "00000000" after 5ns;
```

- You can also use `when` on `std_logic_vector`, just need to use double quote for the values instead of single quotes.

Technical

ALU

- Because of the restrictions found in LogicWorks' VHDL implementation, it will be difficult and buggy to use AU and LU designed using VHDL. However, for Lab 3, I'll let you decide what to do.

- Option 2: Use Circuit Diagrams

  - You will need to use hierarchical design and buses quite a lot to avoid massive circuit diagrams. If you do things correctly, it's not going to take much more time than VHDL.

  - Design a 16bit adder subtractor, start from there. Use everything you learned in CSCI150, this will feel good.