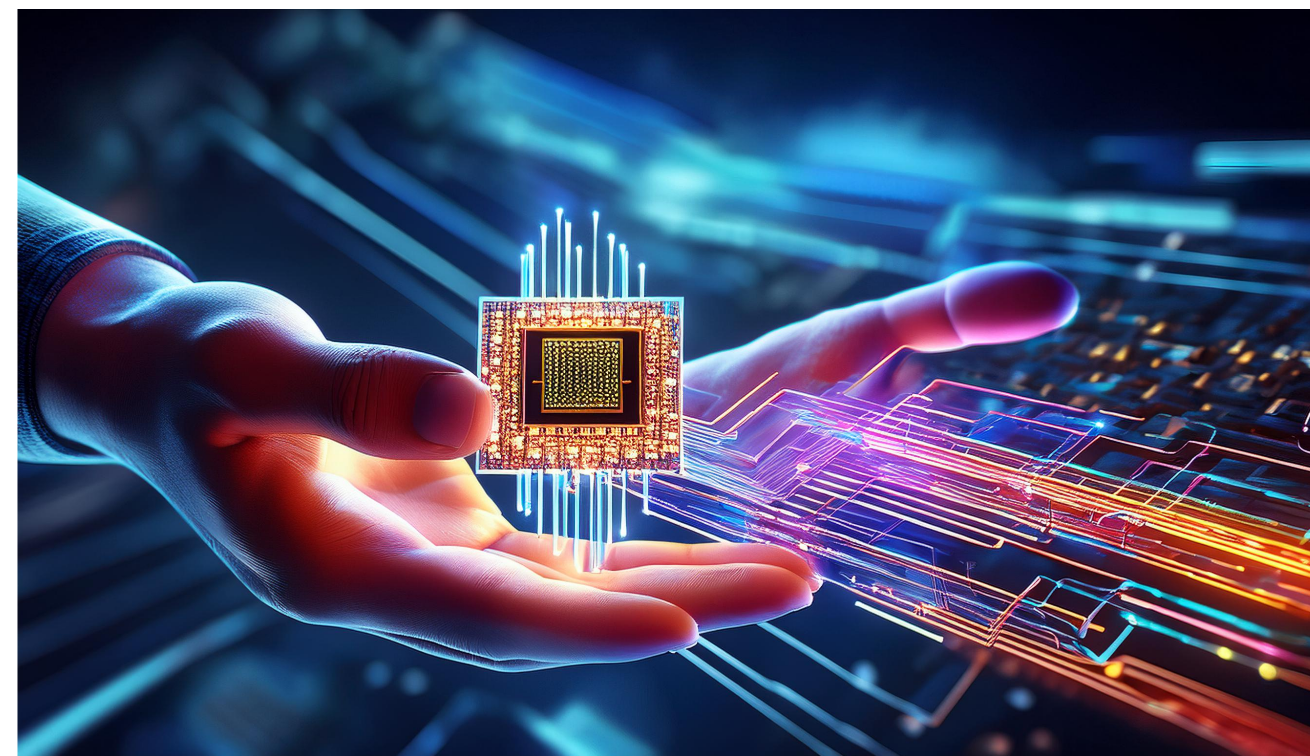
20.09.24 17:37

CSCI 250
Introduction to Computer Organisation
Lecture 1: Beyond Integer Arithmetics IV



Jetic Gū

2024 Fall Semester (S3)

Overview

- Focus: Course Introduction
- Architecture: Logical Circuits
- Textbook: LW Chapter 7
- Core Ideas:
  1. More VHDL: Concurrent Statements

Why FPGA?

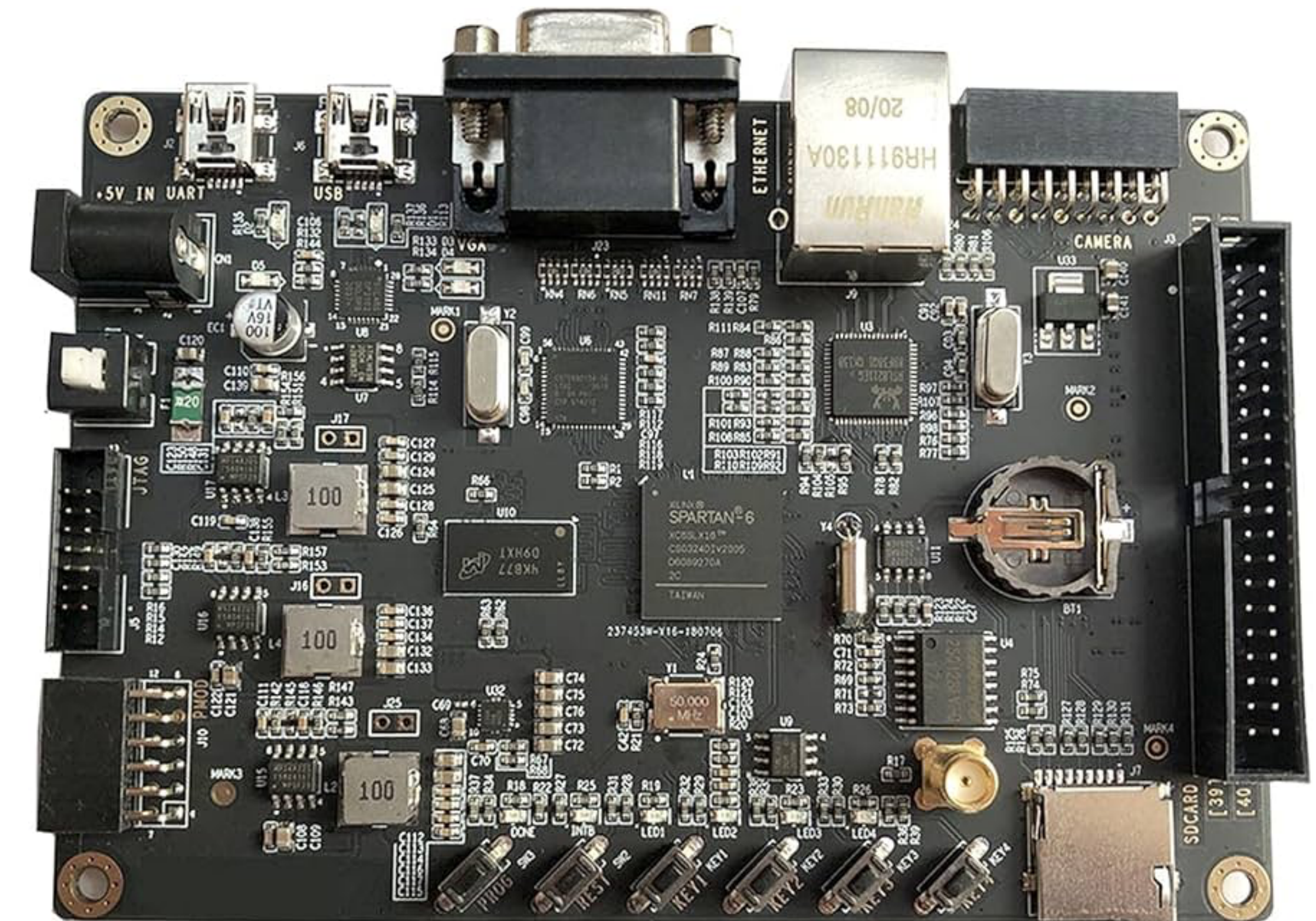
So far, in VHDL

- You’ve used signal to declare new variables
- You’ve use logical/arithmetics similar to Register Transferring Operations
- You’ve built components in VHDL
- Let’s provide you with more detailed explanation of how VHDL works

Review

FPGA

- Field Programmable Gate Arrays
- Embedded system (development)
- Programmable logical circuit
  - Contains Arrays of Logical gates (a lot of LUTs, like truth tables)
  - Logical gates are connected with reconfigurable routing networks
  - Other components: RAM, etc…



Concept

Why FPGA

- Real circuits with physical gates are GOOD
- But you need to make a new one every time you change your design, that's very very very expensive
- Impractical for quick implementation and testing
- FPGAs are more expensive, but reconfigurable in the field (hence the name), with minimal punishment in performance (delays, heat, energy consumption, etc.)

1. Here's a youtube clip: https://www.youtube.com/watch?v=G2uYU7aD698

Concept

P1
FPGA

Use cases: Parallel Computing

- Computer CPUs are traditionally NOT parallel
  - Solution1: **more cores**
    You can only have so many cores (currently hundreds in a single chip)
    Con: expensive, wasteful, energy hungry, hard to maintain
  - Solution 2: **coprocessors**
    For multi-media, you have decoders; for rendering and matrix arithmetics, you have GPUs (thousands of arithmetic units in a single chip)
    Con: very very application specific
  - Solution 3: **FPGA**
    Programmable, highly parallel, as sophisticated as CPUs yet reconfigurable
    Con: not a lot of people knows how to use it?

P1
FPGA

Use cases: Hardware Emulation

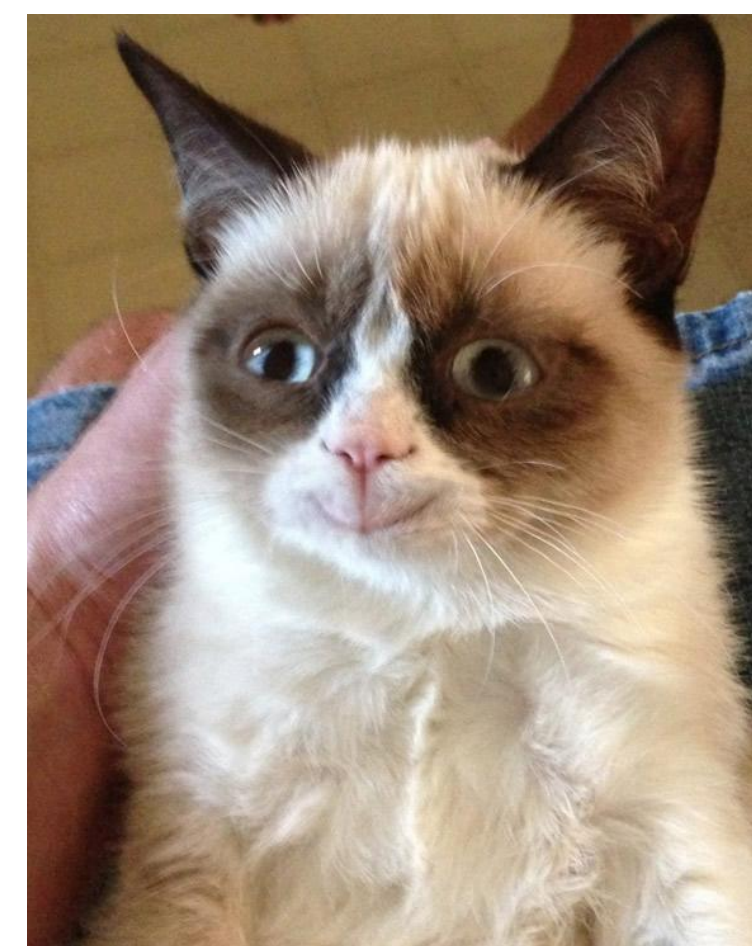
- Everything dies, including microchips
  - Vintage computers and gaming consoles
  - ECUs in cars
  - Controllers for hardware instruments
- Some of these can be **remanufactured**, but that’s very expensive
- Some of these can be **emulated**, but you’ll take a hit on performance and bugs can be unavoidable (accounting for physical gating delays using software is difficult)
- FPGAs are relatively inexpensive in comparison, and can have identical performance (or better) without software emulation issues

Technical

P1
FPGA

In Short

💰(VHDL) > 💰(Python+C+WebDev)



Technical

More about VHDL

Comments

- C/C++
```
// this is a comment
```
- Python
```
# this is a comment
```
- VHDL
```
-- this is a comment
```

Technical

Numbers

- Integer Literals
  - e.g. -1, 2, 42
- Real Literals
  - e.g. 3.1415926535
- Bits and Bit strings
  - Single bit: `'0'`, `'1'`, double quote also works but not recommended
  - Binary starts with `B`, e.g. `B"0100011"`
  - Hexadecimal starts with X, e.g. `X"12CB"`

Technical

P2
More VHDL

Types of Objects

- `signal`: inputs, outputs, intermediates (temporary holders)
  - These are like individual physical wires in a circuit
  - Use `<=` to perform assignment
- `constant` & `variable`: same as in programming languages
  - Use `:=` to perform assignment
  - You typically cannot use `variable`s a lot of the time, so just avoid it for now

Technical

Types of Objects

- `signal`: inputs, outputs, intermediates (temporary holders)
  - These are like individual physical wires in a circuit
  - Use `<=` to perform assignment
- `constant` & `variable`: same as in programming languages
  - Use `:=` to perform assignment
  - You typically cannot use `variable`s a lot of the time, so just avoid it for now

Technical

`std_logic` and `std_ulogic`

- These are signal types, requires IEEE 1164 package
- Can represent the values ->
- When things are working correctly, you should just see 0 or 1
- `std_ulogic` doesn't resolve driver conflicts, and when it occurs will result in compilation/synthesis error
- `std_ulogic` resolves driver conflicts using a resolution table
- **In short, just use `std_logic`**

Value	Meaning
1	Logic 1
0	Logic 0
Z	High impedance
W	Weak signal, can't tell if 0 or 1
L	Weak 0, pulldown
H	Weak 1, pullup
-	Don't care
U	Uninitialized
X	Unknown, multiple drivers

Technical

std_logic and std_ulogic

```
architecture arch1 of bruh is
    signal x : std_logic;
begin

    -- Driver A
    x <= '0';

    -- Driver B
    x <= '1' after 20 ns;

end architecture;
```

- This is driver conflict: multiple sources providing values to a single `signal`

Technical

References

- https://www2.cs.sfu.ca/~ggbaker/reference
- Greg Baker is a senior lecturer at SFU, he created this page 20 years ago for common `std_logic` libraries
- He also hosts a few `std_logic` library source codes by Synopsys, very readable
- If you ever have error from compilation such as conversion issues etc., you can look at these libraries to find out what's supported and what's not

Concurrent Stuff:
Generate Statements

Concurrent vs Sequential

- Just like in CSCI150, in VHDL you can design combinational circuits, and sequential circuits

- For CSCI250, we'll continue this. When you are using VHDL, we'll have two different kinds of statements:

  - **Concurrent statements**
    Everything is executed concurrently, this implements a combinational circuit

  - **Sequential statements**
    Everything is executed sequentially, like programming language statements

- Unless otherwise specified, we use concurrent statements **ONLY** for now

Concept

P3
Generators

Here's a little something you want

- In Lab 1 part 2, you may or may not need to make a `std_logic` signal into `std_logic_vector`.
- How should you do this?

Technical

Option 1: Brute-force

```
...
signal a: std_logic;
...
signal tmp: std_logic_vector(127 downto 0);
...
begin
    tmp(0) <= a;
    tmp(1) <= a;
    tmp(2) <= a;
    tmp(3) <= a;
...
```

- Type an assignment for every digit

P3
Generators

Option 2: Use Python

```
> script.py
with open("driver.vhdl", "a") as f:
    for i in range(128):
        f.write(f"temp({i}) <= a;\n")
```

- Use a python script to write the code for you

Technical

P3
Generators

Option 3: Use Generate Statements

- Do the same thing as Option 2, but let the VHDL compiler do it for you
- Two types:
    - `for`-generate statement
    - `if`-generate statement
- Big issue: LogicWorks doesn't support this… boooooo

Technical

P3
Generators

`for`-generate Statement

```
label: for VAR in RANGE generate
    --concurrent statements here
end generate;
```

- `label`: could be named anything that's a valid identifier
- `VAR`: variable, like `i`; `RANGE`: e.g. `0 to 10`, `10 downto 0`
- Equivalent to typing things out like in Option 1, or using Python to generate the code in Option 2

Technical

`if`-generate Statement

```
label: if CONDITION generate
    --concurrent statements here
end generate;
```

- `CONDITION`: an expression that gives a boolean value, e.g. `i < 0`
- On the hardware side, you can implement the same idea using a 3-state buffer (beware of driver conflict)

Technical