What I know about...

FPGAs in Telecommunications

Charles Clayton

Last updated: March 1, 2017

I. FPGAs

A. VHDL

VHDL in an IEEE standard that allows us to describe hardware. These descriptions can then be *synthesized* into logic gates, and the gates can be implemented on an FPGA. You can either describe the hardware *structurally*, as in what the hardware looks like, or *behaviourally*, as in what the hardware does. This moves us on level higher than schematics, improving productivity in design and allowing for easier testing.

Combinational logic consists of basic gates, but has no memory or storage, so the outputs only depend on the current inputs.

When writing VHDL combinational logic, you start with the *entity* component, which names the inputs and outputs of the module and describes their logic type. Note that you cannot write to inputs, and you cannot read outputs.

```
entity XOR_GATE is
   port(A, B : in BIT;
           Z : out BIT);
end XOR_GATE;
```

Then you write the *architecture* component, which describes the behaviour or structure of the module.

```
architecture XOR_BEHAVIOURAL of XOR_GATE is
begin
   Z <= A xor B;
end XOR_BEHAVIOURAL;
```

If you want to have internal signals (like variables), you could use them to simplify your description.

```
architecture XOR_BEHAVIOURAL of XOR_GATE is
   signal option1, option2 : bit;
begin
   option1 <= A and not B;
   option2 <= B and not A;
   Z <= option1 or option2;
end XOR_BEHAVIOURAL;
```

1) Vectors and Buses: You can define buses using `bit_vector` or preferably `std_logic_vector`, like so:

```
signal X:bit_vector(7 downto 0); -- 8 bit vector
...
A <= bit_vector(4);              -- index bit 5
B <= bit_vector(3 downto 0);     -- index bits 1-4
C <= A & B;                      -- concat vectors
```

We can also define two-dimensional arrays like so:

```
type REGARRAY is array(7 downto 0)
              of bit_vector(3 downto 0);

signal R: REGARRAY;
```

If you get sick of using hard-coded magic numbers everywhere in your code, you can use predefined *attributes* that allow you to index certain bits, for instance:

```
signal Y: bit_vector(7 downto 0);
...
A <= Y(Y'low);   -- means bit Y(0)
B <= Y(Y'high);  -- means bit Y(7)
C <= Y(Y'range); -- means bit Y(7 downto 0)
```

2) Other Types: This has all been educational, but from now on and until the end of time we will never use `BIT` or `BIT_VECTOR`. In all our VHDL, use `STD_LOGIC` and `STD_LOGIC_VECTOR` by including their libraries at the top of your code like so:

```
library IEEE;
use IEEE.std_logic_1164.all;
```

Standard logic types are like the bit types, but instead of just having values of '0' and '1', standard logic also has types:

- Z – high impedance (floating)
- X – unknown value or contention
- U – uninitialized value

To get integer types `unsigned` and `signed`, you will also want to include:

```
use IEEE.numeric_std.all;
```

3) Hierarchy: It's very important to break down larger more complex structural blocks into compositions of smaller blocks, then define your module as a combination of components and how they are connected.

For instance, consider an OR gate as a combination of a NOR gate and an inverter.

```
entity OR_GATE is
   port(IN1, IN2 : in bit;
        OUT      : out bit);
end OR_GATE;


architecture OR_STRUCTURAL of OR_GATE is

   -- component declarations (defined elsewhere)
   component NOR_GATE is
            port(A, B : in bit;
                     Z : out bit);
   end component;

   component INV_GATE is
            port(A : in bit;
                 Z : out bit);
   end component;

   -- internal signal definition
   signal X : bit;

   -- component instantiations and connections
   begin
      UO: NOR_GATE
         port map(IN1, IN2, X);
      U1: INV_GATE
         port map(X, OUT);

end OR_STRUCTURAL;
```

4) Process: Sometimes it's tiring to write this code in terms of gates because you have to think of it all executing simultaneously, so we can use the *process* to write descriptions for hardware behaviour in sequential statements typical of other programming languages, such as *if*, *else*, *case*. The algorithm you use to write the has no bearing on the hardware implementation, your synthesizer will figure out something to do that provides the same behaviour.

The first thing we consider with the process is its *sensitivity list*, whereby the statements are executed whenever any signal in the sensitivity list changes (be it a clock or an input).

```
entity DFF is
   port(D, clk : in std_logic;
             Q : out std_logic);
end DFF;

architecture DFF_BEHAVIOURAL of DFF is
begin
   process(clk) -- executes whenever clk changes
   begin
      if(rising_edge(clk)) then -- only from 0 to 1
         Q <= D;
      end if;
   end process

end DFF_BEHAVIOURAL;
```

You can also define combinational logic such as a mux using the process.

```
entity MUX is
   port(        Sel: in bit_vector(1 downto 0);
         A, B, C, D : in bit;
                  Y : out bit);
end MUX;

architecture MUX_BEHAVIOURAL of MUX is
begin
   process(Sel, A, B, C, D) -- triggers on input change
      case Sel is
         when "00" =>    Y <= A;
         when "01" =>    Y <= B;
         when "10" =>    Y <= C;
         when others =>  Y <= D;
      end case;
   end process;
end MUX_BEHAVIOURAL;
```

5) Finite State Machines: And now using process, we can implement an FSM.

II. IP Development

III. FPGAs

So you have an array of configurable logic blocks, with a whole bunch of interconnects connected by a switching fabric.

These logic blocks are basically just look-up tables, so you can enter the values of the LUT with whatever you want to emulate other logic gates. Each LUT also has a flip-flop associated with that's muxed at the output so if you want you can also implement sequential logic. And if you don't have enough inputs to your LUT, then you can just cascade more of these logic blocks together.

2 to a number is a bitwise left shift of that number.

A. Verilog

Wire assumes a value, a register holds a value.

Wire needs drivers to get output values. Register does not need drivers.

Wire elements can only be used to model combinational logic. Registers can be used for sequential logic.

Wire can be used as the left-hand side of an assignment. Register cannot be used on the left-hand side of the assignment.

Full case, all possible cases are matched.

== can be 0, 1, or undefined

=== can always be 0 or 1

Setup time is the amount of time the input data must be stable before the clock rising edge, hold time is how long the input data must be stable after the clock rising edge.

Blocking assignments, <=, execute sequentially, non-blocking, =, execute concurrently so order doesn't matter.

Assignments are continuous assignments. There's no clock trigger or anything like that.

Tristate means that the node can be pulled up to 1, or down to zero, but can also take a high impedance state – so it's floating.

B. Levels

Behaviour level RTL Logic level Layout

C. VHDL

Inferred latch warning when memory is assumed.

Entitiy – describes inputs and outputs of the module.

Architecture – describes behaviour/structure of the module.

Process – allows us to describe sequential/software-esque code. Based on the sensitivity list. Can only use conditional statements in process.

Do. Not. assign different values to the same signal in more than one process.

D. SystemVerilog

E. Xilinx/Vivado

F. Altera

IV. UVM Test Methodology

Universal Verification Methodology. I've written testbenches in SystemVerilog, but these were all directed tests and test vectors I manually created in Excel instead of using UVM.

In preparation for this interview though, I watched a webinar to learn more about UVM and I think it's really neat and powerful, and would save a lot of time in the long run.

And I do have experience working with OOP languages, notably C#, so for my own projects I'm definitely going spend some time with it to integrate it into my workflow.

V. Networking

A. Ethernet PCS and MAC

1) Physical Coding Sublayer: Responsible for

- Determining if function link established
- Encoding/decoding
- Scrambling/descrambling
- Alignment marker insertion/removal
- Block and symbol redistribution
- Lane block synchronization
- Deskew

Uses 8B/10B coding, so groups of 8-bits are represented by 10-bits.

2) Media Access Control sublayer: Ethernet has a minimum 64-byte frame size (recall packets in physical layer are called frames). No handshaking, connectionless. It's unreliable, doesn't have acknowledgments. Starts with peamble of seven bytes of alternating 1s and 0s to sync clocks.

Uses CRC error checking.

Ethernet's MAC protocol is CSMA/CD (carrier sense multiple access with collision detection) – waits until the channel is idle, called "carrier sensing". If a collision is detected it stops immediately, so efficiency depends on how fast nodes can detect collisions. Delays are then controlled with binary exponential backoff – meaning the first collision will create a situation where the retransmit delay will be one of two slots, then if there's another collision, it'll be one of 4 slots, then 8, and so on.

B. OTL/OTU Transport Protocols

Optical Transport Network (OTN), like 100G Ethernet, serializes frames into stream of blocks, which are then distributed over 20 lanes. Unlike Ethernet, OTN doesn't add markers to to the lanes to identify them, instead the Logical Lane Marker (LLM) identifies them [1]. This is embedded in the frame alignment signal (FAS) and multiframe alignment signal (MFAS) of the OTN frame. This means you need to test the skew at the OTL layer as well as other layers [2].

It supports optical networking by using wavelength-division multiplexing (WDM), to combine signals of different wavelengths onto one fiber, using the different wavelengths as the lanes.

Provides some forward error correction (FEC).

1) OTL: Optical transport lane

2) OTU: Optical transport unit

VI. Their Questions

Why Precise-ITC – One of the reasons was that it's an IP development job, so I think there will be potential to be working with and get involved in emerging technologies. I know its a verification position, but still I would be able to learn from and see the implementations of your more senior engineers. I also think it's a real draw and would be satisfying to be able to be able to sell to and work with larger companies while still working with a tighter group of people, Peter described it as punching above your weight class.

I mean, that's my impression but maybe I'm misunderstanding the job posting, would you say that's accurate description of the role? Could you tell me a little bit more about my position?

—

Adoption of 100G ethernet and OTU4 are dependent on rigorous testing to ensure reliability and robustness to develop confidence, 2011.

Communications skills – documentation, technical writing, as well as interpersonal skills – good for listening and speaking with clients and helping out with consultation.

FPGA programming – for my capstone project, it's basically an array of ultrasonic sensors that we can send time pulses to in order to create patterns of constructive interference (I can tell you more about that if you're interested), I'm setting up our controller with the MicroZed.

So what I've done is written most of the algorithm to develop the timings in C in the PS side using SDK. So it generates the timings, and then loads these delays into eight BRAM registers on the PL side over AXI.

Then there's one more register that's a control register for the eight channels of this array, so then I enable whatever bits correspond to the channels I want to pulse, and so then the VHDL picks it up from there. For each of the channels it starts a count down based on the delay I loaded into the register, then reaches the end and begins a pulses sequence that triggers our high-voltage pulser to send however many specified pulses to our transmitters. So I have the whole transmitter side working.

I've also implemented an SPI protocol in VHDL to specify the settings of our analog front end's registers through this custom 24-bit SPI protocol.

What I was working on on Saturday was trying to store data on an SD card from the MicroZed using, xilisf, for some reason I can read but I'm failing to write from this card.

Discipline – I'm fairly self-motivated because I'll like to take ownership of a part of the project.

Weaknesses –

Strengths – I think this is a strength anyway, but I'll admit when I don't know know something. When people are talking about stuff or using acronyms I'm unfamiliar with, I'm not too proud to say that I don't know what that means or get some clarity. I've been in a few scenarios with my capstone team or other project teams in which someone's been talking to us and everyone's nodding along and I feel like I'm the only one who's missing something, then I'll try to interject and clarify their meaning, and it turns out afterwards that no one was really following.

Networking – Yeah I just sort of happened to pick my upper-level technical courses based on my own interests then I was pleasantly surprised that they complemented each other pretty well in industry, so I chose computer communications which was networking, computer architecture, and VLSI systems which was largely a FPGA programming course.

I've already told Peter about this, but for that course we had to implement some sort of FSM, and just because I enjoyed my computer communications course I decided to write an implementation of the TCP protocol, although admittedly it was dummed down (no flow control, no congestion control, there were acknowledgements but no other error detection or anything), and although not a physical layer protocol, I do have some experience thinking about that sort of thing.

Also with programming I've worked with some networking on the application layer with HTTP requests, and the transport layer with UDP, but I'm not going to lie and say that working with frames on the physical layer has come up in my personal projects.

VII. My Questions

- Will I have a P.Eng as my immediate supervisor?
- Can you tell me about the workplace culture?
- Can you tell me about the team I would be working with?
- Are there any additional technologies you suggest I look into?
- Why are you interested in moving away from consulting toward IP development?
- Would there be opportunities for me to learn and grow at Precise ITC, like maybe working with different groups?
- What's your idea of a good fit with the company, what sort of candidate are you looking for?
- So if I started tomorrow, what would my first assignments be?
- And how could I be successful? What could I do to really impress you with my work?
- And if you're kind enough to offer me the job, when would I start?
- Out of curiosity, I'm just trying to get a grasp of what's being used in industry, I'm wondering if you use HLS?
- It's not a problem for me, but I'm curious about your linux usage requirement. In what way does that apply to your business?
- I was reading about some of these protocols, and I'm curious to what extent you deal with electromagnetism in addition to digital design
- What tools do you use to integrate the digital sphere with the optics sphere

References

[1] Understanding 100g otn - 100g testing - exploring test methodologies for 100g ethernet and 100g otn high speed optical networks. [Online]. Available: http://testing100g.net/understanding-100g-otn/

[2] 100g network: Evolution and challenges — blog. [Online]. Available: http://www.exfo.com/corporate/blog/2011/100g-network-evolution-challenges