IRAM Cache & Address Translation – Mini-Report

Group 327 – Omri Dassa, Ori Yeffet

Our progress:

- ASIC product flow – Amir, our advisor lectured us about the ASIC product flow – the methodologic flow of planning and developing a network product.
- Networking and Ethernet – we read several materials about Networking and Ethernet:
    - CompTIA Network+ Certification 1'st Chapter of Network fundamentals.
    - Fujitsu's TCP/IP protocols tutorial
- Logic design fundamentals
    - Frank Vahid's digital design – chapters 1-6
- Intro to Verilog – we participated in an Intel course for introduction to Verilog. The course covered:
    - Module structure
    - Lexical rules
    - Data types
    - Structural & behavioral modeling
    - Generates
    - Hardware elements
    - TB
    - Design guidelines
- Full adder – our 'Hello World' assignment. As an exercise we implemented Full Adder of several variations.
    - The exercise and our solution is at the end of this report.
    - By having this exercise not only we made first contact with Verilog but also:
        - Learned on compiling and elaboration a project.
        - Experienced the Unix environment and Intel Git system.

What's next?

- Advanced logic design methodologies.
- Networking PCIe
- Introduction to Verification and Validation
- Another exercise – FIFO
- Introduction to simulation and implementation tools

The Adder exercise:

1. Design an adder of 1 bit:
   inputs - 2 signals of 1 bit, output – signal of their sum. Use blocks of full-adders. Link for help: http://en.wikipedia.org/wiki/Adder_(electronics)
2. Design an adder of n bit:
   inputs - 2 signals of n bits, output – signal of their sum.
3. Synthesize your design

Our solution:

1 bit Full Adder

```
module full_adder1(
        input           x,
        input           y,
        input           ci,
        output reg      s,
        output reg      co
        );

        assign s = x^y^ci ;
        assign ci = (x&&y) | (x&&ci) | (y&&ci) ;

endmodule // full_adder1
```

n bit Full Adder

```
module full_adder_n #(
        parameter               N=2)
        (
        input [N-1:0]           x,
        input [N-1:0]           y,
        output reg [N:0]        s
        );

        wire [N-2:0]            co;

        full_adder1 fa_b (
                .x      (x[0]),
                .y      (y[0]),
                .ci     (1'b0),
                .s      (s[0]),
                .co     (co[0])
        );

        generate
                genvar  i;
                for (i=1; i<(N-1); i=i+1) begin
                        full_adder1 fa_n (
                                .x      (x[i]),
                                .y      (y[i]),
                                .ci     (co[i-1]),
                                .s      (s[i]),
                                .co     (co[i])
                        );
                end
        endgenerate

        full_adder1 fa_l (
                .x      (x[N-1]),
                .y      (y[N-1]),
                .ci     (co[N-2]),
                .s      (s[N-1]),
                .co     (s[N])
        );

endmodule // full_adder_n
```