Experiment: Design and synthesize a 32-bit processor using Verilog

Design a 32-bit RISC-like processor in Verilog with the following specifications:

- Sixteen 32-bit general-purpose registers R0..R15, organized as a register bank with two read ports and one write port.
    - R0 is a special read-only register that is assumed to contain the fixed value of 0.
    - Memory is byte addressable with a 32-bit memory address. For simplicity, in this design we shall assume that all operations are on 32-bits data, and all loads and stores occur from memory addresses that are multiples of 4.
- A 32-bit program counter PC
- A 32-bit stack pointer SP

- Addressing modes to be supported:
    a) Register addressing
    b) Immediate addressing
    c) Base addressing for accessing memory (with any of the registers used as base register)
    d) PC relative addressing for branch

- The following instruction set has to be implemented:
    a) Arithmetic and logic instructions: ADD, SUB, AND, OR, XOR, NOT, SLA, SRA, SRL. There are corresponding immediate addressing versions with a suffixing "I" (like ADDI, SUBI, etc.). Assume that all shift instructions can have either 0 (no shift) or 1 (1-bit shift) as operand. Some example uses are as follows:

```
ADDI  R3,#25      // R3 <= R3 + 25
ADDI  R5,#-1      // R5 <= R5 - 1
ADD   R1,R2,R3    // R1 <= R2 + R3
SLA   R5,R7       // R5 <= R5 << R7[0]
SLAI  R5,#1       // R5 <=  R5 << 1
SUBI  SP,#64      // SP <= SP - 64
```

    b) Load and store instructions: LD, ST, LDSP, STSP (all load and stores are 32-bits) and use base addressing (any of the registers R1..R15 or SP can be used). Some example uses are as follows:

```
LD    R2,10(R6)   // R2 <= Mem[R6+10]
ST    R2,-2(R11)  // Mem[R11-2] <= R2
LDSP  SP,0(R2)    // SP <= Mem[R2+0]
STSP  SP,100(R7)  // Mem[R7+100] <= SP
```

    c) Branch instructions: BR, BMI, BPL, BZ. Some example uses are as follows:

```
BR    #10         // PC <= PC + 10
BMI   R5,#-10     // PC <= PC - 10 if (R5 < 0)
```

```
BPL   R5,#30      // PC <= PC + 30 if (R5 > 0)
BZ    R8,#-75     // PC <= PC - 75 if (R8 = 0)
```

d) Stack instructions: PUSH, POP, CALL, RET. Any registers R1..R15 can be used for PUSH and POP. Some example uses are as follows:

```
PUSH  R6          // Push R6 in the stack
POP   R10         // Pop from stack and store in R10
CALL  #1000       // SP <= SP – 4; Mem[SP] <= PC + 4;
                     PC <= PC + 1000
RET               // PC = Mem[PC]; SP <= SP + 4
```

e) Register to register transfer: MOVE. Some example uses are as follows:

```
MOVE  R10,R5      // R10 = R5
MOVE  R2,R0       // R2 = R0
MOVE  R7,SP       // R7 = SP
```

f) Program control: HALT, NOP. The **HALT** instruction waits for an interrupt on an input pin "INT". **NOP** is a dummy instruction that performs no operation.

Steps of implementation:

1. Design the ALU in structured Verilog covering all the functions as required to implement the above instruction set architecture. Write a test bench to verify the functionality through simulation, and also test it by downloading the design on the FPGA kit.
2. Design the instruction format for the processor. Make relevant assumptions where necessary, clearly mentioning the same in the documentation with justifications.

   (Must be submitted to Intinno within 11/09/2014)
3. Implement the register bank and integrate the same with the ALU module as designed earlier. Hence write a top-level module for testing the modules by implementing operations like:

   ```
   Rx = Ry op Rz
   ```

   where Rx, Ry, Rz and op can be specified from outside. Also download the design on FPGA and test for the correct operation.

   (To be demonstrated in the laboratory within 11/09/2014)
4. Implement memory as a one-dimensional register array. Complete the processor design by going through the following steps:
   a) Implement the data path using structural design methodology. **Last Date: 16.10.14**
   b) Prepare a table depicting the (sequence of) RTL micro-operations corresponding to typical instructions like ADD, ADDI, LD, ST, BMI, MOVE, CALL and RET, along with the corresponding control signals required at every step. **Last Date: 28.10.14**
   c) Implement the control path using behavioural design methodology. **Last Date: 28.10.14**
   d) Download the design on FPGA and test the functionality. **Last Date: 30.10.14**

5. Write test benches corresponding to at least one of the following problems:
    a) Find the GCD of two integers
    b) Multiply two integers using Booth's algorithm
    c) Sort a set of integers using bubble sort

Last Date: 13.11.14