Marcko Sicoco

Ted Gervais

**ESE 345 2014 Project:**

**Hierarchical gate-level and dataflow/RTL design of the pipelined multimedia Cell-Lite unit with the VHDL/Verilog hardware description language**

**Project Description**

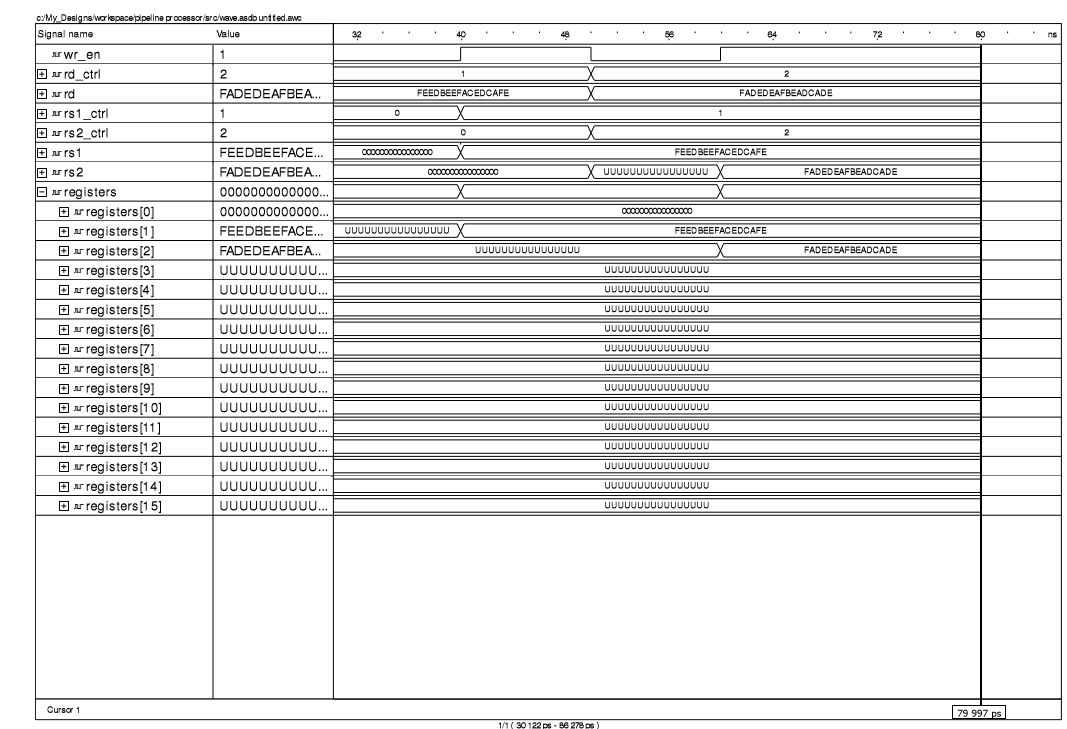
**Purpose:** To learn a use of VHDL/Verilog hardware description language and modern CAD tools for the hierarchical gate-level and dataflow/RTL design of the triple-stage pipelined multimedia Cell-Lite unit with a reduced set of multimedia instructions similar to those in the Sony Cell SPU architecture.

**Goals:** To design a working pipelined Cell-Lite unit with a working ALU, register file and an instruction buffer.

**Description:**

**Register File**

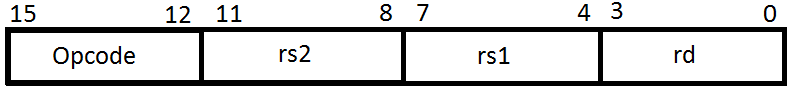
The register file module is used as memory for the rest of the processor. The module uses the dataflow style of architecture, and has 16 64-bit registers, stored in an array within the module. The module has four inputs and three outputs: a write enable signal, used to allow write-back to the register file, and three 4-bit register control vectors, one for the destination register, rd, and two for the two source registers, rs1 and rs2. The register file assigns the 64 bit input rd based on its control signal, and assigns the two 64 bit outputs rs1 and rs2 based on their corresponding control signals. Included in the code is bypass logic, meaning that if rs1 or rs2 are using the same register as rd, the new value in rd will be written to rs1 or rs2 instead of its current value. Below is the waveform verification file of the register file.



In this waveform, two 16-bit hex values are tested and checked to see if they end up in the correct register. At t= 40 ns, the write enable goes up, which allows rd to be written with the hex value x“FEEDBEEFACEDCAFE”. The control signal for rd is set to “0001”, so the value is stored in register 1. Then, rs1 control is set to “0001”, so rs1 outputs the new value in register 1. When write enable goes up again at 60 ns, wd is written with the new hex value x”FADEDEAFBEADCADE”. The control signal is set to “0010”, so the value is saved to register 2. The control signal for rs2 is set to 2, so rs2 outputs the new value in register 2 when it is written by rd.

**Instruction Buffer**

The instruction buffer is a memory storage unit that stores up to 16 16-bit instructions. It reads these instructions from a text-file, then stores them in its internal buffer array. This also includes a Program Counter (PC) that increments when driven by the clock input. When the Program Counter is incremented, the module fetches an instruction on the corresponding value of PC in the internal buffer array. In this way, the PC is used as the address of the instructions. Once it fetches the instruction, the instruction is outputted and sent to the control unit. Instructions are mapped out in the following format:



In order to test the instruction buffer, we use a text file that sends random instructions to the buffer. The buffer then outputs the value corresponding to the buffer address.

**IF/ID Pipeline**

The IF/ID Pipeline module is used to pipe the instructions to the processor control. It does so based on positive clock edges in order that the processor can execute instruction transfers in one cycle. The module has an internal buffer that stores the 16-bit instruction code until it receives a clock cycle. It is one of three clocked modules in the processor.

**ID/EXE Pipeline**

The ID/EXE Pipeline module is responsible for multiple outputs. One of its main concerns is passing the values of rs1 and rs2 to the ALU. It is also responsible for passing along the control signal for Rd, so the ALU will know where to write back to the register file. It also controls the write back control signal, and will set it to 1 whenever it receives an instruction other than the no-op instruction “1111”. It then stores all of these values within its internal buffers, and outputs those values when it receives a clock signal.

**Processor Control**

The processor control module accepts a 16 bit instruction code from the IF/ID Pipeline, then splits the control signals. Instruction values (15 down to 12) go to the operation code input of the ID/EXE Pipeline, while Rs2 (11 down to 8), Rs1 (7 down to 4), and Rd (3 down to 0).

**ALU**

The ALU contains both arithmetic and logical blocks with which instructions are executed. The ALU takes in inputs rs1 and rs2 as 64-bit values used to perform operations. It also takes in a 4-bit opcode value to determine what operation needs to be performed. This opcode is sent to a control logic block within the ALU which determines whether or not the instruction is a logical or arithmetic operation. Additional control signals are then generated that determine whether a value is loaded from the testbench, bypassing the ALU; what logical or arithmetic operation should be done; or in the case of an arithmetic instruction: whether the operation is done with 32-bit or 16-bit operands and whether the result should be saturated or not. The result of the operation is then output as a 64-bit signal rd out of the ALU, or in the case of the nop operation, the previous result is output again and no new operation result is outputted.

The logical blocks of the ALU are implemented in VHDL under a behavioral model. Their code blocks are relatively short as a result and a short, accurate description of what they do. The multiplication and absolute difference arithmetic instructions are implemented in the same manner.

The adder is implemented under a structural model at its lowest level: the half adder and the carry-lookahead logic. The half-bit adders are used to define full adders with propagate-generate signals. The full adders and carry-lookahead logic blocks are then used to define 4-bit adders, which again are used in turn with another level of carry-lookahead logic to define 16-bit adders. Normally, these are then used again with another level of carry-lookahead logic for a plain 64-bit adder, but this application primarily uses packed 32-bit and packed 16-bit unsigned values so multiplexors are placed in between the different carry signals from that third level of carry-lookahead logic to separate the interactions between the 16-bit values or the 32-bit values in the actual 64-bit result. These multiplexors are controlled by signals generated by the ALU’s control logic. Similarly, the sums from each 16-bit adder are sent to saturation blocks, a series of multiplexors controlled again by the ALU’s control logic to determine whether the results need to be saturated from either overflow or underflow or if the result can be passed straight through without any modification.

Using the adder, subtraction is performed in a similar way, with the value of rs2 getting switch with its ones complement and a carry in being sent into the adder to get the proper two’s complement.

**Conclusion:**

The instruction buffer works as per requirements. It accepts the test instructions from the test\_file.txt, and stores those instructions in its internal buffer (up to 16). It then outputs instructions based on the internal Program Counter.

The register file, in testing, works properly. The write back logic passes the new values of Rd into either Rs1 or Rs2 whenever they both access the same file. The Rd input into the buffer is stored in the array until it is overwritten, thus creating a memory location for the rest of the processor. It was written in dataflow style.

The IF/ID register works properly as well. It receives a 16 bit instruction value, saves it in its internal buffer, and passes it to the processor control when it receives a clock edge.

The ALU in testing, works as per requirements, all of the logical operations and the absdb and mpyu instructions work properly. The arithmetic instructions writing in a structural architecture also work properly with words and half-words being separated from and not affecting each other when necessary and saturating with the proper op.