EE 231L Lab 5

Putting it All Together: Building the Computer

In this lab you will put the parts of the computer together to build a functional computer. This is just a matter of including the blocks in a `tdf` file, and defining the inputs to and the outputs from the blocks in this file. You will also define the other combinational circuits used by the computer, such as a few multiplexers and a decoder.

You should do the following:

1. Create a file which defines useful constants. For my computer, I defined three sets of constants: the ALU control lines, the memory select control lines, and the computer instructions. Here are a few lines from my file `constants.inc`:

```
%*---------------------------------------------------------------------------
constants.inc - file to hold all the global constants
 *---------------------------------------------------------------------------*%


%* -- ALU constants -- *%
CONSTANT ALU_LOAD     = B"0001";

% Memory Mux constants %
CONSTANT PC_SEL = B"00";

% Instruction constants %
CONSTANT LDAA = H"01";
```

2. Make a file which includes the `Default Include Files` of all your blocks. Here are a few lines from my `computer.inc` file, which contain the prototypes for the ALU and the 8-bit synchronous load register (which is used for, among other blocks, ACCA):

```
%*---------------------------------------------------------------------------
computer.inc - file which contains prototypes of blocks used
               in computer design
 *---------------------------------------------------------------------------*%

FUNCTION alu
    (din[7..0], acca[7..0], x_reg_in[7..0], ctrl[3..0])
    RETURNS
(dout[7..0], cout, zout);

FUNCTION load_reg
(din[7..0], reg_l, clock)
    RETURNS
(dout[7..0]);
```

```
FUNCTION load_inc_reg
(din[7..0], reg_l, reg_inc, clock)
    RETURNS
(dout[7..0]);
```

3. Create the file for the final computer design. Here are some lines from my file `computer.tdf` — I included the parts which define the inputs to the ALU and ACCA:

```
INCLUDE "constants.inc";
INCLUDE "computer.inc";

SUBDESIGN computer
(
prog_addr[7..0] : INPUT;
prog_data[7..0] : INPUT;
port_in[7..0] : INPUT;
reset : INPUT;
clock : INPUT;
prog_r : INPUT;
prog_w : INPUT;

addr_out[7..0] : OUTPUT;
port_out[7..0] : OUTPUT;
mem_r : OUTPUT;
mem_w : OUTPUT;
mem_cs : OUTPUT;

mem_data[7..0] : BIDIR;
)

VARIABLE
my_alu : alu;
acca : load_reg;
x_reg : load_inc_reg;

BEGIN
% Define the inputs to the ALU %
my_alu.din[] = mem_data[];        % din comes from memory bus %
my_alu.x_reg_in[] = x_reg.dout[]; % X input comes from X register %
my_alu.acca[] = acca.dout[];      % acca comes from the outputs of ACCA %
my_alu.ctrl[] = ctrl.alu_ctl[];   % ctl comes from the control machine %

% Define the inputs to the ACCA %
acca.din[7..0] = my_alu.dout[];  % din comes from ALU %
acca.reg_l = ctrl.acca_l;        % load enable comes from control machine %
acca.clock = clock;              % clock is the global clock %
```

You should write this file in pieces – for example, start with the control unit, and make sure that part compiles. (The control unit uses inputs from the instruction register and the carry, which have not yet been defined. Just assign those inputs to a constant value to start, and put in the correct inputs when the appropriate parts are added. For example, you might start with

```
ctrl.inst[] = H"00";
```

After you add the instruction register, you should change this to:

```
ctrl.inst[] = inst_reg.dout[];
```

4. Simulate the function of the computer. In your simulation, include the inputs to and the outputs from the computer, and the values of the internal registers. Attached is a copy of my simulation. You need to supply the values for the data from memory. I did my simulation by having a clock with an input frequency of 1 MHz. I started with the `reset` input low, and made sure the control lines from the control state machine were in the proper states. I then brought `reset` high, and reran the simulation. In order to get data from the memory, both `mem_cs` and `mem_r` need to be low. When either of these lines were high, I made the input on `mem_data[]` to be `ZZ` (high-impedance). When `mem_cs` and `mem_r` were both low, I put the values appropriate for the address on the `mem_data[]` lines. I assumed the following program was in the memory:

```
LDAA #0x2A
ADAA 0xF0
STAA 0xF1
```

This is what I assumed would be in the necessary memory locations:

Address	Data
0x00	0x02
0x01	0x2A
0x02	0x04
0x03	0xF0
0x04	0x03
0x05	0xF1
0xF0	0xEB
0xF1	0x00

At the start of the simulation, when `mem_cs` and `mem_r` were both low, and `addr_out[]` was at `0x00`, I put an `0x02` on the `mem_data[]` input lines. I then reran the simulation. In the next cycle, when the `addr_out[]` was at `0x01`, I put an `0x2A` on the `mem_data[]` input lines. The instruction `LDAA #0x2A` is a two-cycle instruction. At the end to the second cycle, I verified that `0x2A` was loaded into `ACCA`.

I went on to simulate the `ADDA 0xF0` instruction, and verified the `ACCA` contained a `0x15` (`0x2A + 0xEB`), and that the carry was set. I then simulated the `STAA 0xF1` instruction, to verify that the value in `ACCA` was put on the `mem_data[]` output lines, and that `mem_cs` and `mem_w` were both low, so that the value in `ACCA` would be written into memory address `0xF1`.

I replaced the `ADDA 0xF0` instruction with `SUBA 0xF0` to verify that the `SUBA` instruction worked correctly. I continued until I verified that all the instructions worked properly.

5. I then tested the input and output ports. I simulated the instructions `LDAA 0xFF` to verify that the system would load data from the input port, and simulated the `STAA 0xFF` instruction to verify that the system would write data to the output port.

After the simulation works, it is time to program the computer into an Altera chip. I will update this lab with more details on how to do that next week.