Lab Assignment 2, Stage 6: Support for all DT instructions features

Kushal Kumar Gupta, 2020CS10355

The features supported include byte and half word transfers (signed and unsigned), auto increment/decrement with option of pre/post indexing.

Files, entities and Architectures:

Changes from stage 5-

`I.PMconnect_stage6.vhd`:

```
entity PMconnect is
Port(
     DT_instr : in DT_instr_type;
     state : in integer;
     ADR10 : in std_logic_vector(1 downto 0);
     Rout : in word;
     Mout : in word;
     Rin : out word;
     Min : out word;
     MW : out nibble
);
end PMconnect;
```

Combinational circuit between the processor and memory which does the required transformation of words into half-words / bytes and vice versa.

Implementation considerations -

- It contains a single process `connect` which has all the inputs in its sensitivity list.
- Depending on the type of DT instruction, state of the FSM and the lowest 2 bits of memory address, it decides the `Min`, `Rin` and `MW`. Document for stage 6 on Moodle has been followed appropriately.
- Here `Min` is the data to be written in memory, calculated appropriately from `Rout`, and `Rin` is the data to be written in Register File, calculated from `Mout`.
- `MW` is the byte level memory write enable, which is decided depending on the type of load instruction and `ADR10`.

For more details regarding the implementation of `PMconnect`, see the architecture `PMconnect_beh`.

II.Processor_stage6.vhd:

Included component of PMconnect, and appropriate glue logic followed to incorporate this module.

Implementation considerations -

- Auto-indexing supported by writing back the calculated address stored in register Res in the base register in state 7 for store instructions and state 8 for load instructions
- Pre and post indexing supported by taking the memory address from register A (base register) in case of post-indexing and register Res (address with offset) in case of pre-indexing.
- Half-word and byte transfers supported by using the PMConnect outputs PMConnect_Min as write data for Memory and PMConnect_Rin as write data for Register File.PMConnect has been supplied registers DR and B as inputs for Mout and Rout respectively.
- In case of DT instructions, for Shifter module, the input data, shift type and shift amount has been selected appropriately. For instructions that don't take shift/rotate, LSL #0 has been applied.
- In FSM, whether to transition to state 10 or 11 from state 3 has been decided appropriately.

For more details regarding the implementation of PMconnect, see the architecture PMconnect_beh.

III.mytypes_stage6.vhd:

Changed instr_class_type to differentiate between the two DT instruction formats (depeding on the F field) and included the following types to augment the decoder-

```
type instr_class_type is (DP, DTtype00, DTtype01, MUL, BRN, none);
type DT_instr_type is (STR, STRB, STRH, LDR, LDRB, LDRSB, LDRH, LDRSH, invalidDT);
type pre_post_type is (pre, post);
type write_back_type is (write_back, no_write_back);
```

IV.Decoder_stage6.vhd:

```
entity Decoder is
Port (
    instruction : in word;
    instr_class : out instr_class_type;
    operation : out optype;
    DP_subclass : out DP_subclass_type;
    DT_instr : out DT_instr_type;
    DP_operand_src : out src_type;
    load_store : out load_store_type;
    pre_post : out pre_post_type;
    DT_write_back : out write_back_type;
    DT_offset_sign : out DT_offset_sign_type;
    DT_offset_src : out src_type;
    reg_shift_type : out shift_rotate_op_type;
    reg_shift_src : out src_type
);
end Decoder;
```

Extra output ports
DT_instr: Which DT instruction among STR, STRB, STRH, LDR, LDRB, LDRSB, LDRH, LDRSH or invalidDT
pre_post: Whether pre or post indexing in DT instruction.
DT_write_back:Whether to write back the calculated address or not in the base register.

How to use:

On edaplayground.com, upload testbench.vhd and run.do in the left column, and ALU_stage3.vhd, RegFile_stage1.vhd, Mem_stage3.vhd, mytypes_stage6.vhd , decoder_stage6.vhd, FlagUpdater_stage5.vhd, conditionChecker_stage4.vhd, programCounter_stage3.vhd, processor_stage6.vhd, shifter_stage5.vhd, PMconnect_stage6.vhd. Copy contents in design.vhd section as given in the design.vhd file. Select Testbench + Design as VHDL. Then, for-

1.)Simulation

Type testbench in the Top entity. Select Aldec Riviera Pro 2020.04 simulator to simulate the design. Set the run time accordingly and select the EPWave option. Then Save and Run the simulation to get waves of the signals defined in these modules.

2.)Synthesis

Copy the VHDL file of the component you want to synthesise into design.vhd. (Only design.vhd entities are synthesised)

Then, select Mentor Precision 2021.1 to synthesise. Select netlist option to view the Verilog description. Then Save and Run to get the synthesis result. We get the report containing the resource table specifying number of IOs, LUTs, CLB slices, ports, nets, etc. used to implement this module in the given FPGA specified by run.do.

Results of EPWave (Simulation):

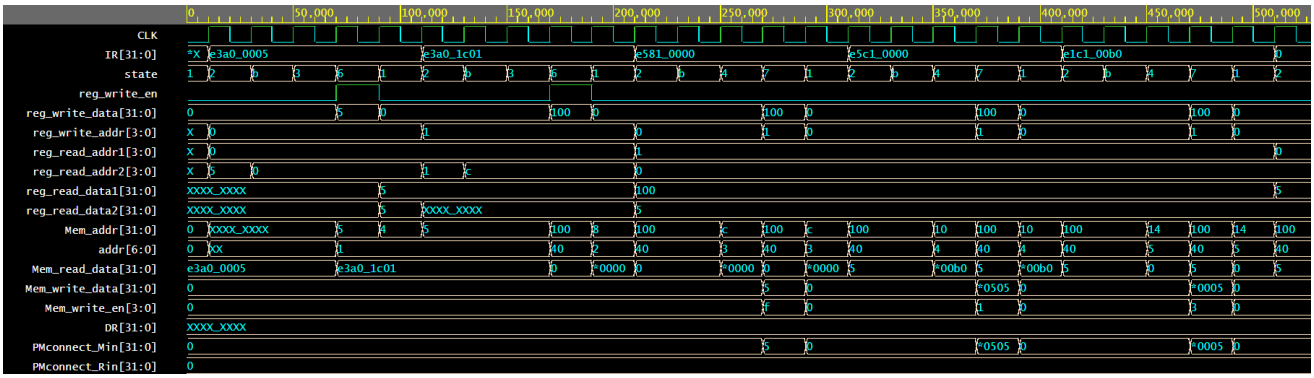
NOTE: Please ensure that Program memory is from address 0 to 63, data memory is from 64 to 127

Can see the input and output signals of processor against the clock.

Memory initialisation with different programs and EPWave-

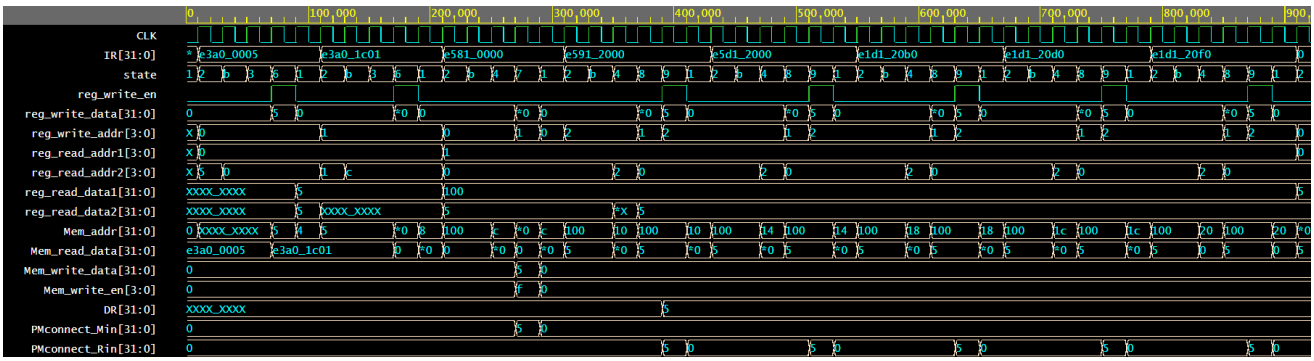
1.

```
    signal Mem_space : type_mem :=
                                ( 0 => X"E3A00005",
                                  1 => X"E3A01C01",
                                  2 => X"E5810000",
                                  3 => X"E5C10000",
                                  4 => X"E1C100B0",
                                  others => X"00000000"
                                  );

--1 mov r0, #5
--2 mov r1, #256
--3 str r0, [r1] --store in word number 64
--4 strb r0, [r1]
--5 strh r0, [r1]
```



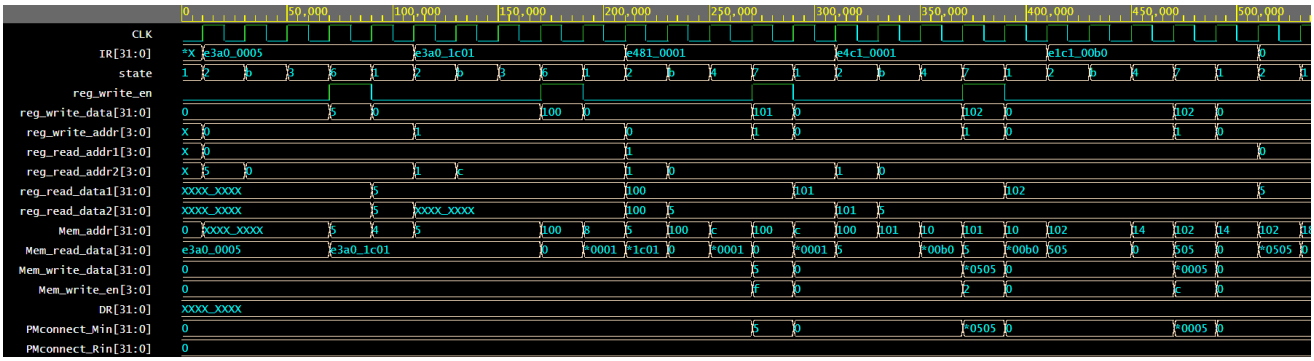
2.

```
    signal Mem_space : type_mem :=
                                ( 0 => X"E3A00005",
                                  1 => X"E3A01C01",
                                  2 => X"E5810000",
                                  3 => X"E5912000",
                                  4 => X"E5D12000",
                                  5 => X"E1D120B0",
                                  6 => X"E1D120D0",
                                  7 => X"E1D120F0",
                                  others => X"00000000"
                                  );

--0 mov r0, #5
--1 mov r1, #256
--2 str r0, [r1] --store in word number 64
--3 ldr r2, [r1]
--4 ldrb r2, [r1]
--5 ldrh r2, [r1]
--6 ldrsb r2, [r1]
--7 ldrsh r2, [r1]
```



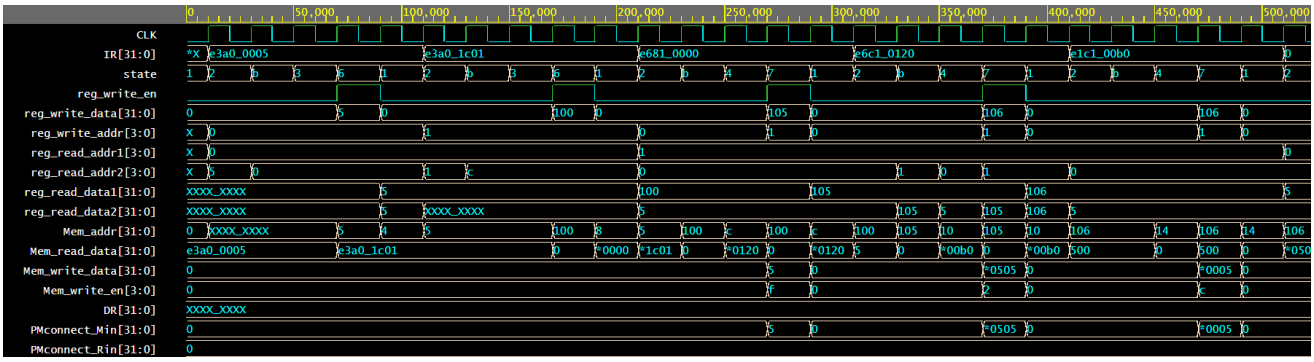
3.

```
    signal Mem_space : type_mem :=
                                ( 0 => X"E3A00005",
                                  1 => X"E3A01C01",
                                  2 => X"E4810001",
                                  3 => X"E4C10001",
                                  4 => X"E1C100B0",
                                  others => X"00000000"
                                  );

--0 mov r0, #5
--1 mov r1, #256
--2 str r0, [r1], #1 @store in word number 64
--3 strb r0, [r1], #1
--4 strh r0, [r1]
```



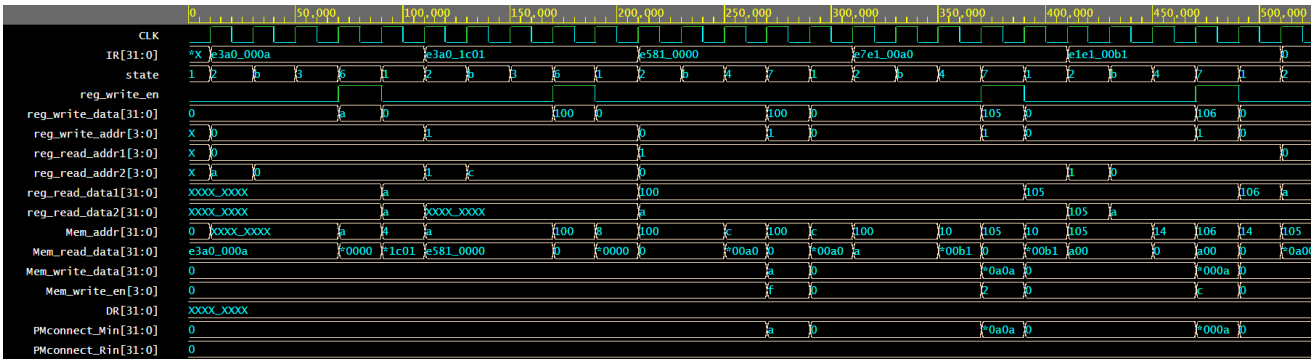
4.

```
    signal Mem_space : type_mem :=
                                ( 0 => X"E3A00005",
                                  1 => X"E3A01C01",
                                  2 => X"E6810000",
                                  3 => X"E6C10120",
                                  4 => X"E1C100B0",
                                  others => X"00000000"
                                  );

--0 mov r0, #5
--1 mov r1, #256
--2 str r0, [r1], r0 @store in word number 64
--3 strb r0, [r1], r0, LSR #2 @store in 2nd byte of word 65
--4 strh r0, [r1] @store in second half of word 65
```



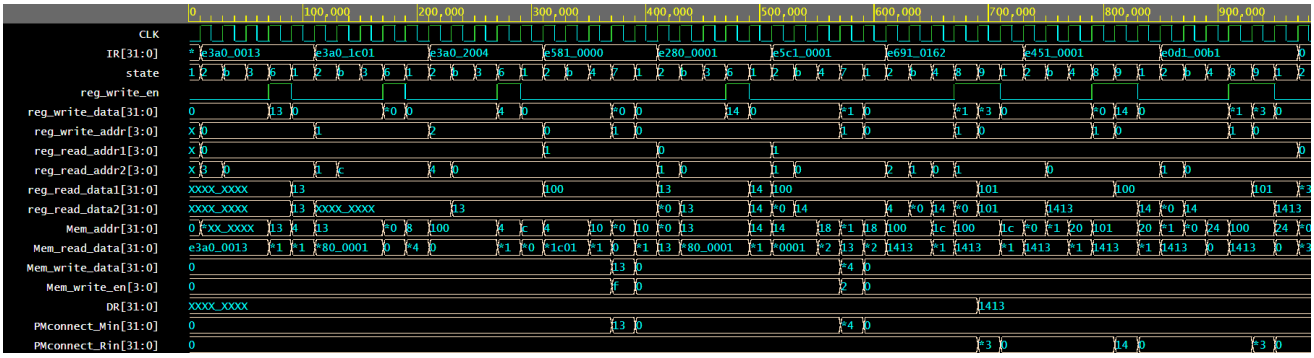
5.

```
    signal Mem_space : type_mem :=
                                ( 0 => X"E3A0000A",
                                  1 => X"E3A01C01",
                                  2 => X"E5810000",
                                  3 => X"E7E100A0",
                                  4 => X"E1E100B1",
                                  others => X"00000000"
                                  );

--0 mov r0, #10
--1 mov r1, #256
--2 str r0, [r1] @store in word number 64
--3 strb r0, [r1, r0, LSR #1]! @store in 2nd byte of word 65
--4 strh r0, [r1, #1]! @store in second half of word 65
```



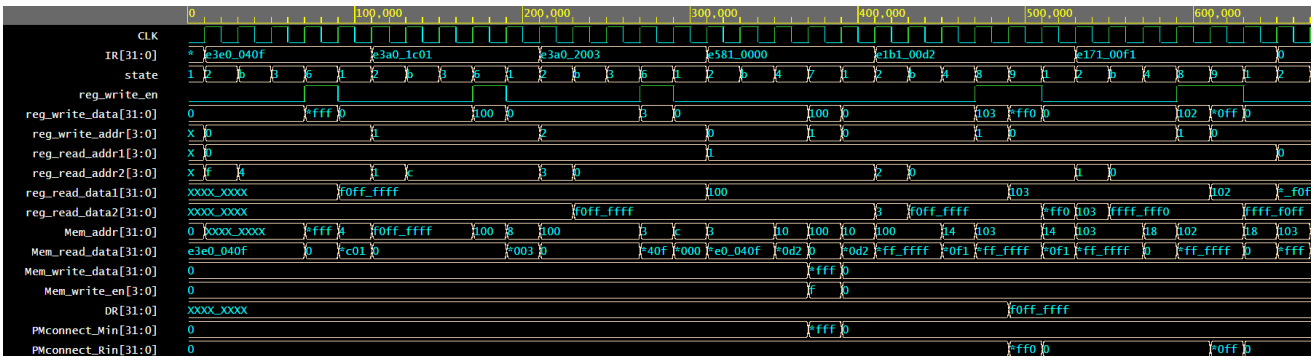
6.

```
    signal Mem_space : type_mem :=
                                ( 0 => X"E3A00013",
                                  1 => X"E3A01C01",
                                  2 => X"E3A02004",
                                  3 => X"E5810000",
                                  4 => X"E2800001",
                                  5 => X"E5C10001",
                                  6 => X"E6910162",
                                  7 => X"E4510001",
                                  8 => X"E0D100B1",
                                  others => X"00000000"
                                  );

--  mov r0, #0x13
--  mov r1, #256
--  mov r2, #4
--  str r0, [r1] @store in word number 64
--  add r0, r0, #1 @r0 now has 0x14
--  strb r0, [r1, #1]
--  ldr r0, [r1], r2, ROR #2
--  ldrb r0, [r1], #-1 @load 2nd byte
--  ldrh r0, [r1], #1 @load first half
```



7.

```
    signal Mem_space : type_mem :=
                                ( 0 => X"E3E0040F",
                                  1 => X"E3A01C01",
                                  2 => X"E3A02003",
                                  3 => X"E5810000",
                                  4 => X"E1B100D2",
                                  5 => X"E17100F1",
                                  others => X"00000000"
                                  );

--  mvn r0, #0x0F000000
--  mov r1, #256
--  mov r2, #3
--  str r0, [r1] @store in word number 64
--  ldrsb r0, [r1, r2]! @load 4th byte with sign
--  ldrsh r0, [r1, #-1]! @load 2nd half with sign
```



2.) Resource Table (Synthesis)

Can see the resources used by the different module implementations on given FPGA.

For example, for `PCconnect` -

```
# Info: Device Utilization for 7A100TCSG324
# Info: ***************************************************************
# Info: Resource                          Used    Avail   Utilization
# Info: ---------------------------------------------------------------
# Info: IOs                               175     210      83.33%
# Info: Global Buffers                    0       32        0.00%
# Info: LUTs                              129     63400     0.20%
# Info: CLB Slices                        28      15850     0.18%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: DSP48E1s                          0       240       0.00%
# Info: ---------------------------------------------------------------
# Info: ***************************************************************
# Info: Library: work    Cell: PMconnect    View: PMconnect_beh
# Info: ***************************************************************
# Info:  Number of ports :                          175
# Info:  Number of nets :                           411
# Info:  Number of instances :                      307
# Info:  Number of references to this view :          0
# Info: Total accumulated area :
# Info:  Number of LUTs :                           129
# Info:  Number of Primitive LUTs :                 135
# Info:  Number of LUTs with LUTNM/HLUTNM :          12
# Info:  Number of accumulated instances :          307
```