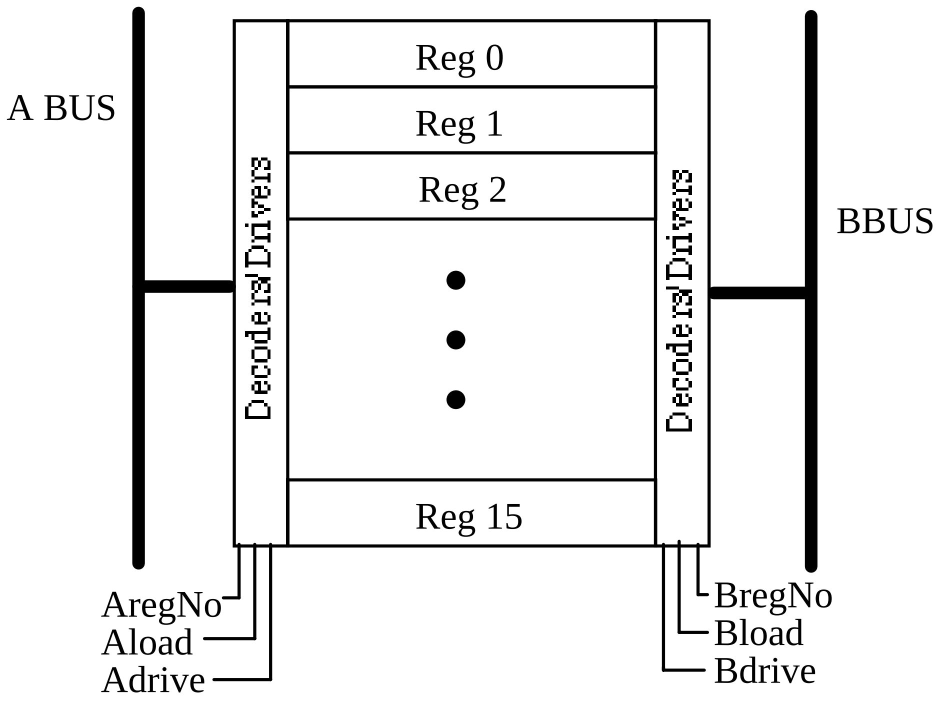
Project Step 7

Behavioral modeling of a dual ported register set.

The register set

- Register set specifications
- 16 dual ported registers each with 16-bit words
- Control signals for each bus
  - Register Number
  - Load Strobe
  - Drive Strobe
- Registers know nothing about time



Note: AregNo,BregNo are type Integer

Dual ported registers

- Very common in today's architectures
- Capable of being written and/or loaded from the busses simulteanously
- Possible Operations in a cycle
  - Load the Abus value into register n, the Bbus value into rejester o, where n does not equal o
  - Load a register from one bus and drive a different register on to the other
  - Drive both busses from different registers

Each bus

- The type of the bus will be std_logic_vector which is a resolved type.
- It will have two drivers
  - The testbench is a driver of the bus.
  - The register set is a driver of the bus.
- Original contents of the registers is “UU…U”
- Must set the bus to “ZZ…ZZ” so that values can be transferred

To get the bus to Z

- To get the bus to high impediance, Z, all the drivers of the bus must go to high impedance.
- Testbench starts out during initialization by assigning values of Z to the bus. These transactions get post to the current value of the testbench driver of the buses at 0 +1 delta.
- The register set code must also set its driver to Z so values can be loaded.

The testbench driver

- At time 0 driver of the bus starts out ‘U’, unresolved, the default initialization value.
- At time 0 the process that runs the bus begins execution.
- First steps in this process are to assign high impedance to the busses and initialize the control signals.
- It then calls a procedure which suspends until 9 ns.
- This will allow update at 0+1delta of the busses

The key to this project

- Must write VHDL code to model the register set. Note that the registers are 16 bits.
- Best done in a process. Array of bit vectors can be a variable array declared in the process.
  - May need a type definition 1st and then declaration of variable.
- Remember that processes with a sensitivity list run once through and then hold for an event on the signals they monitor.
- They also run once through at time 0

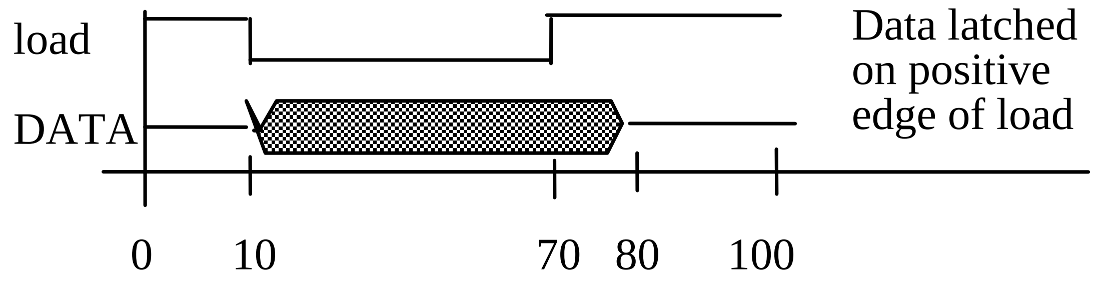
Structure of processes

- What goes in the sensitivity list
  - The control signals?
  - The bus signals?
- The process needs to have code that sets its bus outputs at time 0 to ‘Z’ (even if it happens a delta later)
- How to do this?
- Value to drive bus to at all times
  - Either drive a value from a register
  - Or drive high impedance
- At 0 + 1 delta there will be an event on the control signals
  - They start out as ‘U’ as they are type std_logic
  - Have an event to ‘1’

Bus timing

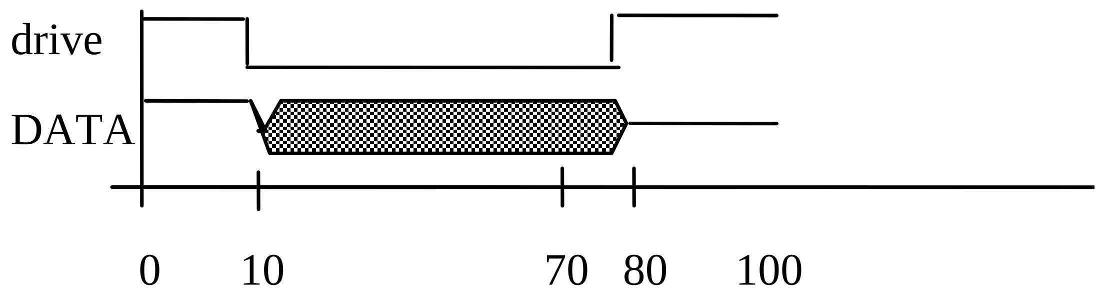
- Note timing of control signals

LOAD - Transfer Data from Bus to Register



Data latched on positive edge of load

Drive - Transfer contents of Register to Bus



Some useful info

Remember that the following BOOLEAN conditions check for:

Level Sensitive:

```
Signal Low         IF (the_sig = ‘0’)  THEN   -- valid for BIT or STD_LOGIC
Signal High        IF (the_sig = ‘1’)  THEN
Signal Low         IF (the_sig = ‘0’   OR   the_sig = ‘L’)  -- for STD_LOGIC
Signal High        IF (the_sig = ‘1’   OR   the_sig = ‘H’)
```

Edge Sensitive Tests (for TYPE BIT or STD_LOGIC)

```
High to Low        IF (the_sig = ‘0’  AND  the_sig’event)  THEN
Low to High        IF (the_sig = ‘1’  AND  the_sig’event)  THEN
```

Edge Sensitive Test for STD_LOGIC that is complete

```
High to Low
IF ((the_sig’LAST_VALUE = ‘1’ OR the_sig’LAST_VALUE = ‘H’)  AND
            (the_sig = ‘0’  OR  the_sig = ‘l’)  AND the_sig’EVENT) THEN
Low to High
IF ((the_sig’LAST_VALUE = ‘0’ OR the_sig’LAST_VALUE = ‘L’)  AND
            (the_sig = ‘1’  OR  the_sig = ‘H’)  AND the_sig’EVENT) THEN
```

Other useful information

- If you don’t make sure the register driver gets set to ‘Z’ then you will just load ‘U’s to start and probably stay there.
- The registers themselves know nothing about time. All they understand of the world is that they have control signals that tell them when to load a value or when to drive a value.
- The registers themselves do nothing to check about a conflict in register number access. They just know the control signals.
- The unit that generates the control signals is the responsible party.