Instantiating Memory: HDL vs FPGA vs ASIC

Author: Ryan Cramer

Contents:

1. Background

Instantiating memory can prove to be tricky depending on your design target. At the RTL level, memory is often written in behavioral HDL form (arrays). However, in FPGA or ASIC flows, synthesis tools map these behavioral constructs to hardware primitives or dedicated memory macros.

The designer must consider:

- HDL syntax (reg vs logic, unpacked vs packed arrays)
- Technology mapping (does the FPGA vendor or ASIC pdk this construct?)
- Read/write semantics (synchronous vs asynchronous)
- Port configurations (single, dual, or multi-port)

2. Memory using Verilog and SystemVerilog

At the RTL level, memory is often written in behavioral HDL form (arrays). Instantiating memory in Verilog and SystemVerilog is typically done like so:

```
Verilog:        reg logic [NBIT-1:0] memory [0:MAX_ADDR-1];
SystemVerilog:  logic [NBIT-1:0] memory [0:MAX_ADDR-1];
```

- NBIT defines the width (# bits per word)
- MAX_ADDR defines the depth (# of words)

Memory access may be done like so:

```
      always_ff @(posedge clk) begin
  if (write_en)
    memory[addr] <= write_data;
  read_data <= memory[addr];
end
```

This describes a synchronous write and synchronous read memory. Asynchronous reads may be performed by replacing the always_ff with always_comb and replacing the non-blocking <= operator with the blocking = operator. However, be careful, while this may work in simulation, it is never a guarantee that your FPGA or process node supports asynchronous reads.

3. Memory with FPGA

FPGA vendors (Xilinx, Altera, Lattice, etc.) provide block RAM (BRAM) primitives. HDL arrays can often infer BRAM automatically, but exact inference will depend on:

- Read/write style
- reset behavior (not all FPGAs allow resetting every cell)
- dual-port support

For precise control, you can directly instantiate vendor primitives. For example, Xilinx' RAMB36E1 primitive is a 36kB BRAM. To instantiate it, you would simply use:

(* ram_style = "block" *) reg [NBIT-1:0] mem [0:MAX_ADDR-1];

The synthesis attribute ***ram_style*** is set to block, telling the tool to use BRAM rather than LUT memory. If you desire LUT memory, you can say:

(* ram_style = "distributed" *) …

If you desire either/or, maybe with a small design such as a 32x32 register file, you would use:

(* ram_style = "{distributed | block}" *) …

LUT memory is generally faster but can store less memory than BRAM. If you do have the ability to perform asynchronous reads, your memory is likely LUT RAM.

4. Memory with an ASIC

An experienced designer knows that their RTL is simply a template. As your code is processed by the various tools, it will be parsed and converted into a device-level hierarchy, and intermediate forms of the design will be generated. One such intermediate form is the gate-level netlist, which replaces bitwise and arithmetic operators with logic gates from the SCL (Standard Cell Library) of the PDK (Process Design Kit) you are working with.

In ASIC flows:

- Behavioral HDL memory arrays are usually not synthesized directly
    - they'll turn into a massive bank of flip-flops
        - large area and poor timing
        - costly

- Designers use **memory compiler macros** from the foundry/PDK
    - Instantiated like black boxes
        - models provided for simulation (.v)
        - models provided for timing (.lib)
    - Instantiate the same way you would any other module (for simulation)

- Memory is expensive, regardless of whether you do SRAM or DFFRAM, etc.

- Macros are often not available in a limited number of sizes and configurations, which is why you often need to use special interleaving tools or RAM compilers to generate the memory files you need.