Xilinx Embedded Memory

• At the circuit level Xilinx supports two types of embedded memory
  – Distributed Memory (small, local)
  – Block Memory (large, fixed location)

• HDL Implementation Methods
  – “Inferred” by synthesis tool 😞
  – “Instantiated” from Xilinx library 😞
  – Using the magic Core Wizard! 😊
Why the Wizard?

• Best of both worlds
• Optimized for your target technology
  – speed
  – resource usage
• Portable via a generic “wrapper” that encapsulates Xilinx specific memory code
Distributed Memory

• Small, local memory with standard interface
• Built with unused LUTs (anywhere)
• Synchronous writes, asynchronous reads

Key Features:

Generates Read Only Memories (ROMs), Single, Simple Dual and Dual-port Random Access Memories (RAMs), and SRL16-based RAMs
Supports data depths ranging from 16 to 65,536 words
Supports data widths ranging from 1 to 1024 bits
Optional registered inputs and outputs
Optional sync and async resets for output registers
Slices and CLB
Logic or Memory?

Left-Hand SLICEM
(Logic or Distributed RAM
or Shift Register)

Right-Hand SLICEL
(Logic Only)

Switch Matrix

CLB

Figure 16: Arrangement of Slices within the CLB
Slice M versus Slice L

SLICEM

SLICEL
Dual-Port Example

Data written to both, but independent read address and data lines

Figure 26: RAM16X1D Dual-Port Usage
Using an IP Core

Seems pretty consistent between ISE 9.2 and 14.4 ...
Save the Datasheet before Generating the Memory!!!
The IP Catalog has been reloaded.
Loaded all available family support information.
INFO:sim:172 - Generating IP...
   [...]
Running synthesis for 'my_mem'
Running ngcbuild...
Writing VEO instantiation template for 'my_mem'...
Writing Verilog behavioral simulation model for 'my_mem'...
Delivered 2 files into directory
   [...]
Top level has been set to "/my_mem"
Generating README file...
Generating FLIST file...
INFO:sim:948 - Finished FLIST file generation.
Launching README viewer...
Moving files to output directory...
Finished moving files to output directory
Wrote CGP file for project 'my_mem'.
Core Generator create command completed successfully.
Instantiation Template (my_mem.veo)

// The following must be inserted into your Verilog file for this
// core to be instantiated.

//--------------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
my_mem your_instance_name (  
  .a(a), // input [5 : 0] a  
  .d(d), // input [15 : 0] d  
  .clk(clk), // input clk  
  .we(we), // input we  
  .spo(spo) // output [15 : 0] spo
);
// INST_TAG_END ------ End INSTANTIATION Template ---------

// You must compile the wrapper file my_mem.v when simulating
// the core, my_mem. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".
"Wrapper" File (my_mem.v)

// The synthesis directives "translate_off/translate_on" specified below are
// supported by Xilinx, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

`timescale 1ns/1ps

module my_mem(   
a,  
d,  
clk,  
we,  
spo  
);  

input [5 : 0] a;  
input [15 : 0] d;  
input clk;  
input we;  
output [15 : 0] spo;
Xilinx Specific (my_mem.v)

// synthesis translate_off

DIST_MEM_GEN_V7_2 #(  
  .C_ADDR_WIDTH(6), 
  .C_DEFAULT_DATA("0"), 
  .C_DEPTH(64), 
  .C_FAMILY("spartan3"), 
  .C_HAS_CLK(1), 
  .C_HAS_D(1), 
  
  [...] 
  .C_HAS_SPO(1), 
  .C_HAS_SPRA(0), 
  .C_HAS_WE(1), 
  .C_MEM_INIT_FILE("no_coe_file_loaded"), 
  .C_MEM_TYPE(1), 
  .C_PARSER_TYPE(1), 
  .C_PIPELINE_STAGES(0), 
  
  [...] 
)
Figure 4: Distributed Single-Port RAM Module Schematic
Figure 2: Write Timing Diagram