IL2234

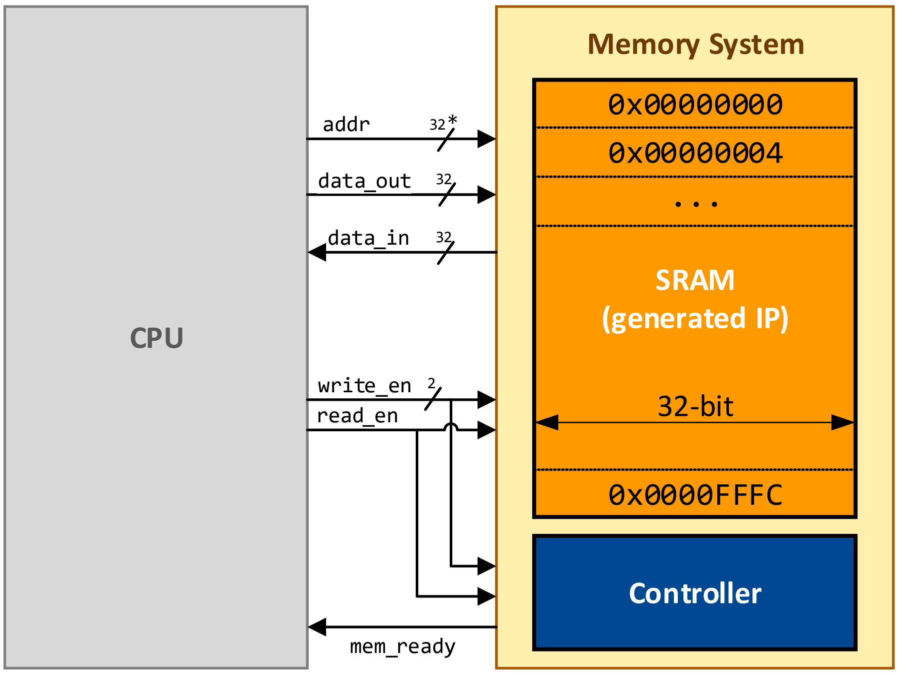
Digital Systems Design and Verification using Hardware Description Languages

Project Milestone 3

Memory Implementation

Introduction

In this milestone, we will generate the SRAM that will be used as the main memory for our CPU design. We will use the Xilinx IP generator to generate a memory. Additionally, we will design a simple control logic to generate the flags which indicate to the controller that the memory operation is finished.

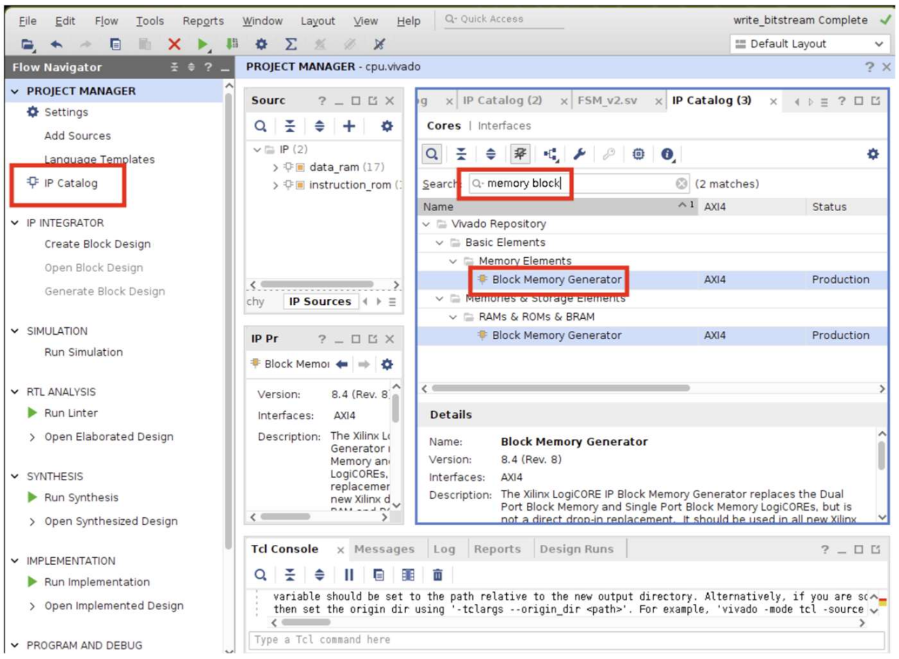


* The CPU address bus is 32 bits, but since we are using a 32-bit byte-addressable memory, the lower two bits of the address are used to generate the `write_en` signal (byte select), resulting in the “effective” address width being 30 bits .

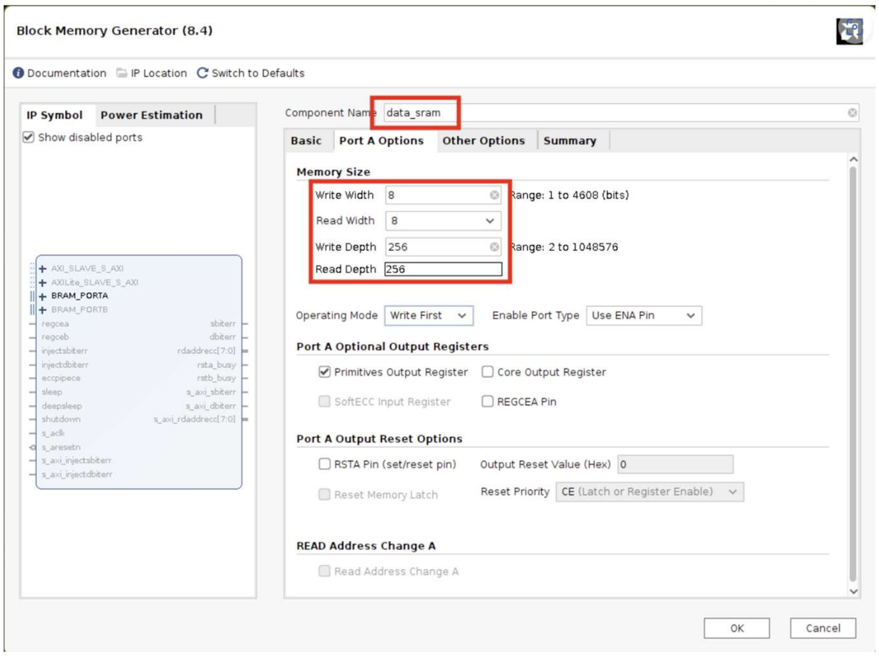
IP generation

Open the IP catalogue and search for Block Memory

Choose the type of memory that you want to create (SRAM, ROM, etc.)



In Port A Options, select the size according to your design. Note that the depth refers to the number of memory rows; Vivado automatically calculates the address width as `clog2(DEPTH)`



Block Memory Generator (8.4)

Component Name: data_sram

Memory Size

Write Width: 8 — Range: 1 to 4608 (bits)
Read Width: 8
Write Depth: 256 — Range: 2 to 1048576
Read Depth: 256

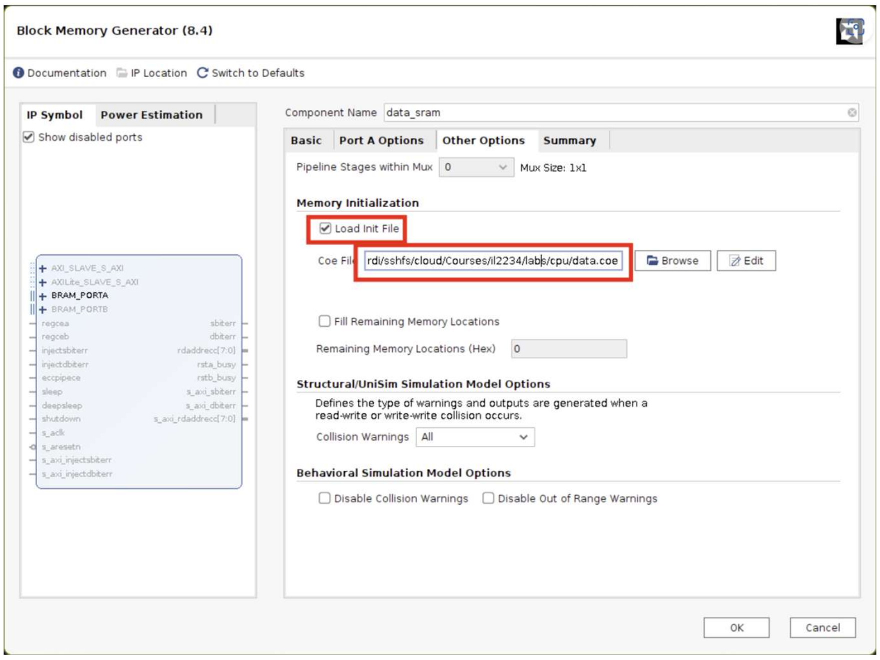
Operating Mode: Write First — Enable Port Type: Use ENA Pin

Port A Optional Output Registers: Primitives Output Register ✓; Core Output Register; SoftECC Input Register; REGCEA Pin

Port A Output Reset Options: RSTA Pin (set/reset pin); Output Reset Value (Hex): 0; Reset Memory Latch; Reset Priority: CE (Latch or Register Enable)

READ Address Change A: Read Address Change A

In Additional Options, select the COE (Coefficient) and generate an initialization file with dummy data, which you will use to test the SRAM behaviour.



[✓] Show disabled ports

Component Name: data_sram

Pipeline Stages within Mux: 0 — Mux Size: 1x1

Memory Initialization

[✓] Load Init File

Coe File: rdi/sshfs/cloud/Courses/il2234/labs/cpu/data.coe

[ ] Fill Remaining Memory Locations

Remaining Memory Locations (Hex): 0

Structural/UniSim Simulation Model Options

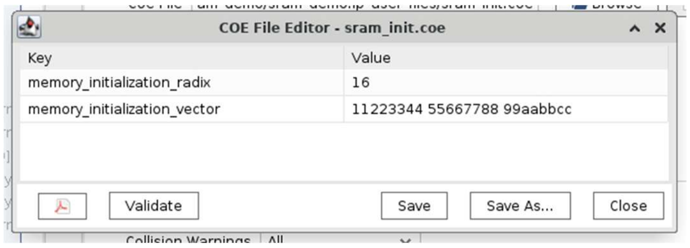
Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

Collision Warnings: All

Behavioral Simulation Model Options

[ ] Disable Collision Warnings [ ] Disable Out of Range Warnings

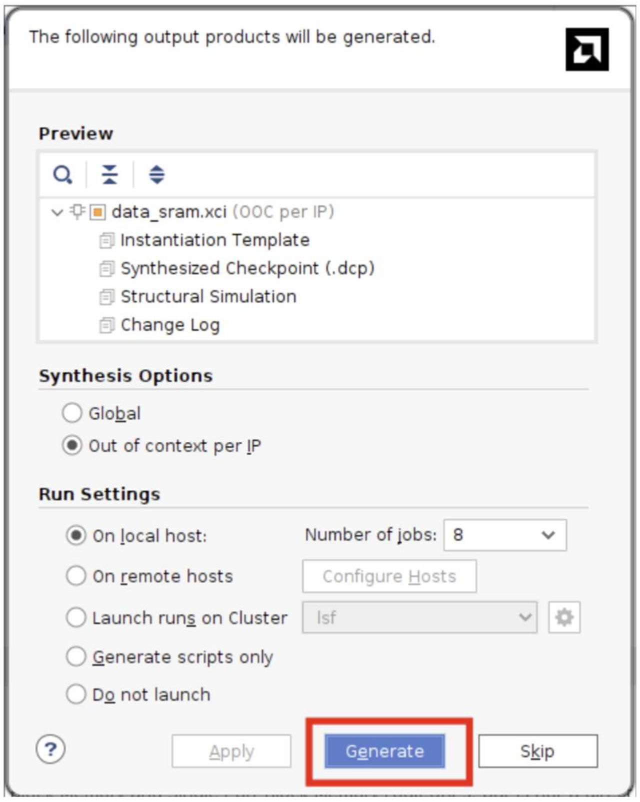
This is an example of dummy data in the first 3 addresses.



COE File Editor - sram_init.coe

Key	Value
memory_initialization_radix	16
memory_initialization_vector	11223344 55667788 99aabbcc

Click OK, and in the next window, click Generate.



The following output products will be generated.

Preview

- data_sram.xci (OOC per IP)
  - Instantiation Template
  - Synthesized Checkpoint (.dcp)
  - Structural Simulation
  - Change Log

Synthesis Options

- Global
- [✓ Out of context per IP]

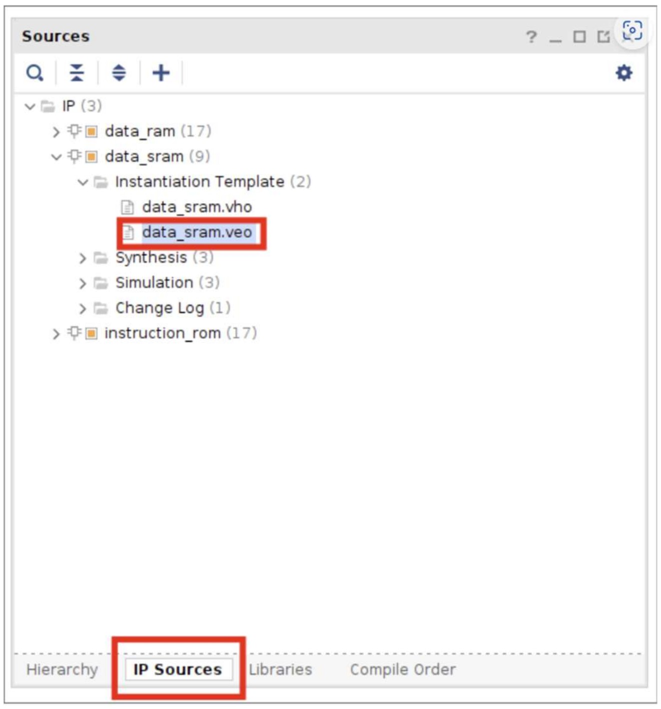
Run Settings

- [✓ On local host:] Number of jobs: 8
- On remote hosts — Configure Hosts
- Launch runs on Cluster — lsf
- Generate scripts only
- Do not launch

You should now see your IP under Sources->IP Sources.

If you expand the generated block, you should see several files, among them instantiation templates.

Open the `.veo` and you will see an instantiation template for Verilog/SystemVerilog.



Sources

- IP (3)
  - data_ram (17)
  - data_sram (9)
    - Instantiation Template (2)
      - data_sram.vho
      - data_sram.veo
    - Synthesis (3)
    - Simulation (3)
    - Change Log (1)
  - instruction_rom (17)

Tasks

Generate a SRAM block with the following parameters:

- Data width: 32 bit
- Depth: 16384 (corresponding to 64 KiB)
- Output register
- Byte-wise write enabled (byte size = 8 bit)

Instantiate the SRAM macro in your memory system module and design a simple state machine that generates the `mem_ready` flag. Note that this flag should be 1 when the output data is valid (for loads) and when the memory has been successfully written (for stores).

Create a testbench to test the different functionalities of the memory (read, write, byte-wise writes). Use this testbench to verify the timing of the SRAM regarding the `enable` and `mem_ready` signals.

Deliverables

Use GitHub Classroom to submit your RTL and testbench files.

The GitHub Classroom repositories contain a skeleton of the RTL and testbench files for the memory that you may use for your development.

Hints

The processor must assert and hold the enable signal (`read_en` or `write_en`) until it receives the corresponding response from the memory. Otherwise, you might see glitches or temporary data for the first clock cycle after the enable signal is asserted.

Bonus tasks

(2 points) Implement multiple smaller memories and add a different number of registers at the output. This simulates different latencies. Generate the corresponding `mem_ready` logic according to the corresponding latency.