# Pushing SRAM Blocks through CS250's Toolflow

CS250 Tutorial 8 (Version 093009a)
September 30, 2009
Yunsup Lee

In this tutorial you will gain experience pushing SRAM blocks through the toolflow. You will use these SRAM blocks to incorporate dense memory structures into your design. Especially in lab 3, you will use these SRAM blocks to implement on-chip memory. You will also learn how to cascade smaller blocks to make a larger block. Finally, you will see how the SRAM blocks are pushed through simulation, synthesis, place and route, and power analysis.

Normally, you will use a memory generator provided by the foundry to generate an arbitrarily sized memory. Unfortunately, we don't have access to the memory generators. For the class, you will use pre-sized memory blocks which are provided by Synopsys. The library contains 16 pre-built SRAM blocks in different width and depth. However, the SRAM blocks are in development and not all of them will work as intended. We have one SRAM block working with the toolflow. Please contact the TA if you need use other SRAM blocks.

The following documentation is located in the course locker (`~cs250/docs/manuals`) and provides additional information about SRAM blocks. OpenSparc megacells include some larger structures, for example, cache blocks, cache tag structures, and register files.

- `synopsys-90nm-databook-memories.pdf` - Memory Databook
- `synopsys-90nm-databook-opensparc.pdf` - OpenSparc Megacell Databook

## Getting started

Before using the CS250 toolflow you must run the course setup script with the following command.

```
% source ~cs250/tools/cs250.bashrc
```

Synopsys 90nm educational library includes the following SRAM blocks. The table explains which SRAM block works with which part of the toolflow. Please consult the TA if you need to use the SRAM blocks which is marked not working.

You can spot the relevant files with the following commands.

```
% ls ~cs250/stdcells/synopsys-90nm/default/verilog/memories/
% ls ~cs250/stdcells/synopsys-90nm/default/db/memories/
% ls ~cs250/stdcells/synopsys-90nm/default/mw/memories/
```

For this tutorial you will be using a 32 wide, 512 deep (32x512) SRAM block as your example RTL design. The block is made out of four 32 wide, 128 deep (32x128) SRAM blocks. You should create a working directory and copy files from the course locker using the following commands.

```
% mkdir tut8
% cd tut8
% TUT8_ROOT=`pwd`
% cp -R ~cs250/examples/v-sram32x512/* $TUT8_ROOT
```

| SRAMnxm | VCS (Simulation) | DC Compiler (Synthesis) | IC Compiler (Place and Route) | PrimeTime PX (Power Analysis) |
|---|---|---|---|---|
| SRAM4x16 | X | X | X | X |
| SRAM4x32 | X | X | X | X |
| SRAM4x64 | X | X | X | X |
| SRAM4x128 | X | X | X | X |
| SRAM8x16 | X | X | X | X |
| SRAM8x32 | X | X | X | X |
| SRAM8x64 | X | X | X | X |
| SRAM8x128 | X | X | X | X |
| SRAM16x16 | X | X | X | X |
| SRAM16x32 | X | X | X | X |
| SRAM16x64 | X | X | X | X |
| SRAM16x128 | X | X | X | X |
| SRAM32x32 | X | X | X | X |
| SRAM32x32 | X | X | X | X |
| SRAM32x64 | X | X | X | X |
| SRAM32x128 | O | O | O | O |

Figure 1 depicts the block diagram of the SRAM32x512 block.

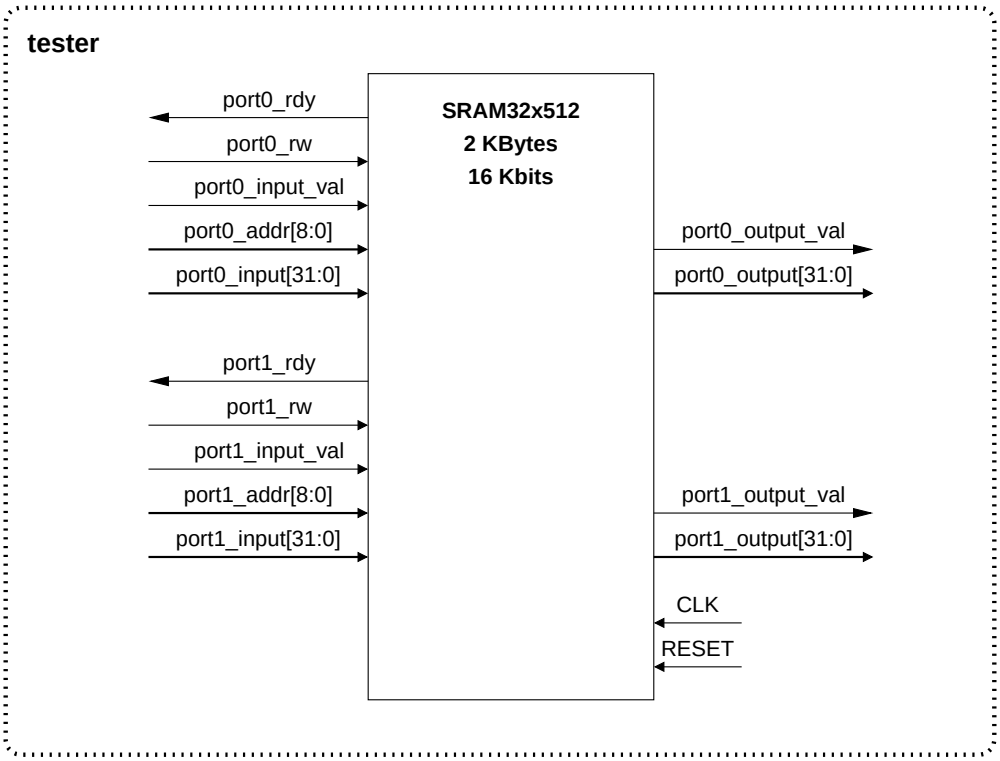


Figure 1: Block diagram for the SRAM32x512 block

This memory is dual-ported - it has two read ports and two write ports. Notice that each port has the same structure. Each port has a `rdy` signal which indicates whether or not the module is ready to accept any requests. The `rw` signal tells you if the request is a read or a write. Notice the `input` and `output` are 32 bit wide. Since the memory is 512 deep, the `addr` signal is 9 bits. Each `input` and `output` has a `valid` signal.

Since you don't have a memory generator, you need to make this block out of small SRAM blocks which are provided. You will use four `SRAM32x128` blocks to make a `SRAM32x512` block. Figure 2 shows the port configuration of a `SRAM32x128` block.

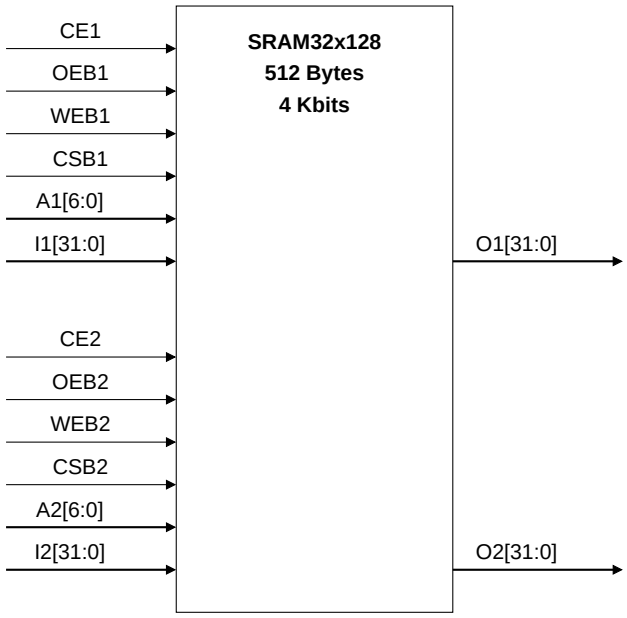


Figure 2: Port configuration of the SRAM 32x128 block

`CE` ports are for the clock. `CSB` ports are for the chip select, `OEB` ports are for the output enable, and `WEB` ports are for the write enable. Notice that the ports with the `B` suffix are active low. `A` ports are for the address, `I` ports are for the input, and `O` ports are for the output. Suffix `1` is for the first port, suffix `2` is for the second port.

Figure 3 describes how to make a 32x512 SRAM block out of four 32x128 SRAM blocks. You can use the higher two bits or the lower two bits of the address to choose banks, however, you will be using the lower two bits for this example. Notice that the clock, higher seven bits of the address, and the 32-bit data inputs are connected to all banks. Since the individual SRAM blocks have output enable signals, you can save muxes on the data output port. The lower two bits of the address picks which bank's chip select signal will be asserted low. The output enable signal and the write enable signal of each bank will depend on the read/write signal and the lower two bits of the address.

Now let's go ahead and try to simulate, synthesize, and place and route the 32x512 SRAM block. You will also analyzer power consumption. You will use the automated makefiles and scripts. If you are not familiar with the automated build process, consult individual tutorials for each tools.

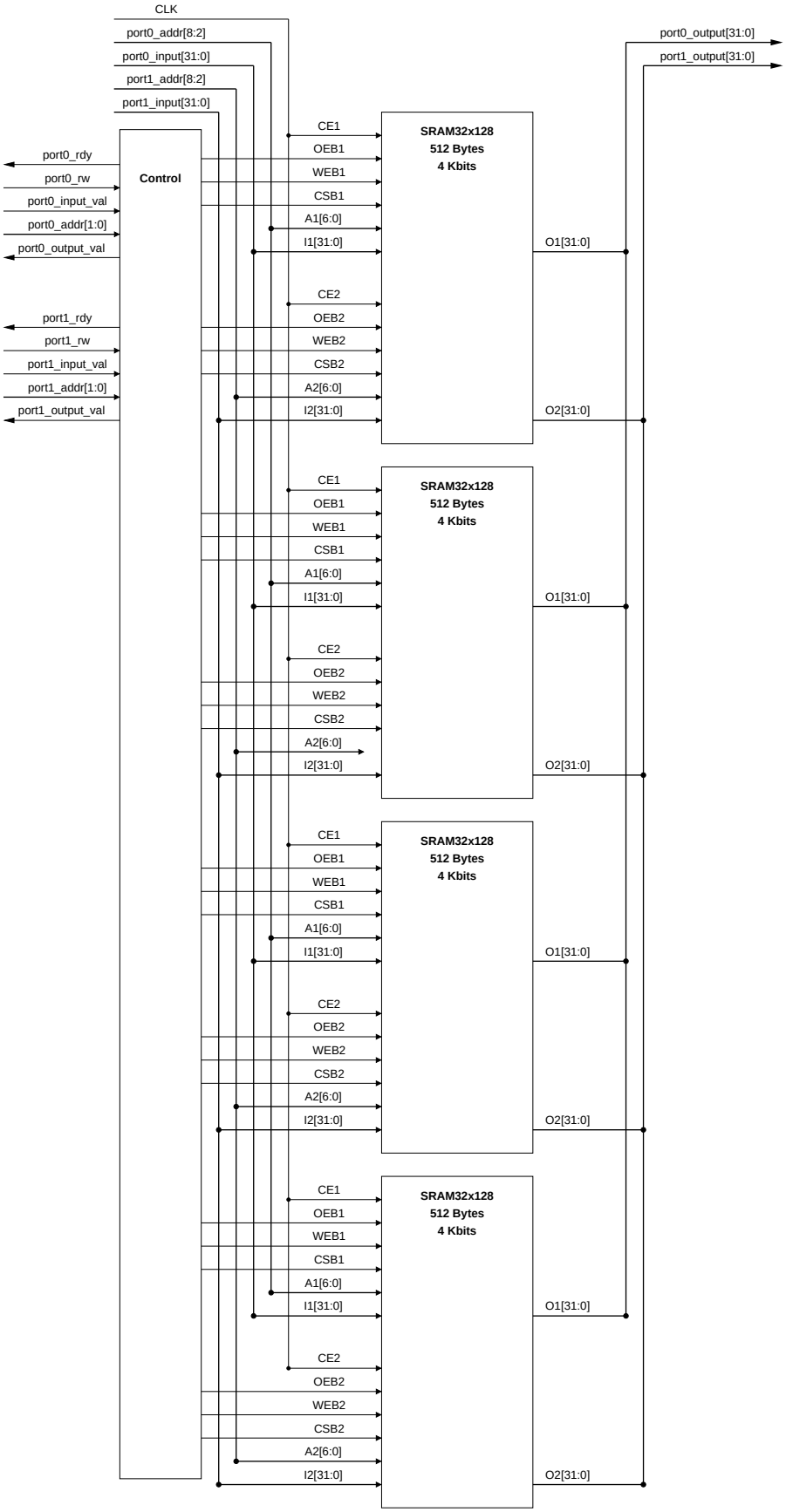


Figure 3: Cascade four SRAM 32x128 blocks to make a SRAM 32x512 block

# Pushing the SRAM Block through the VLSI Toolflow

### Simulate using Synopsys VCS

You will begin by simulating the SRAM block. Use the following commands to simulate. Make sure it passes.

```
% cd $TUT8_ROOT/build/vcs-sim-rtl
% make run
...
  + Running Test Case: SRAM32x512
     [ passed ] Test ( vcTestSink ) succeeded, [ deadbeef == deadbeef ]
     [ passed ] Test ( vcTestSink ) succeeded, [ aaaaaaaa == aaaaaaaa ]
     [ passed ] Test ( vcTestSink ) succeeded, [ 48239582 == 48239582 ]
     [ passed ] Test ( vcTestSink ) succeeded, [ 1042abcc == 1042abcc ]
     [ passed ] Test ( vcTestSink ) succeeded, [ deadbeef == deadbeef ]
     [ passed ] Test ( vcTestSink ) succeeded, [ aaaaaaaa == aaaaaaaa ]
     [ passed ] Test ( vcTestSink ) succeeded, [ cccccccc == cccccccc ]
     [ passed ] Test ( Is sink finished? ) succeeded
...
```

Notice in the makefile that you included SRAM32x128.v.

```
% cd $TUT8_ROOT/build/vcs-sim-rtl
% cat Makefile
...
srcdir = $(basedir)/src
vsrcs = \
        $(UCB_VLSI_HOME)/stdcells/$(UCB_STDCELLS)/verilog/memories/SRAM32x128.v \
        $(srcdir)/SRAM32x128.wrap.v \
        $(srcdir)/SRAM32x512.v \
        $(srcdir)/SRAM32x512.t.v \
...
```

Take a look at the test harness located in `SRAM32x512.t.v`. Notice that there are two test sources and two test sinks for each memory ports. Each port writes to memory addresses, and reads them out to check if they are the exact bits.

### Synthesis using Synopsys Design Compiler

Go ahead and synthesize the SRAM block using the following commands. After synthesizing, use Formality to formally verify the gate-level netlist. Also simulate the post synthesis gate-level netlist.

```
% cd $TUT8_ROOT/build/dc-syn
% make
% make fm
% cd $TUT8_ROOT/build/vcs-sim-gl-syn
% make run
```

Notice that you included `SRAM32x128.db` in your makefile.

```
% cd $TUT8_ROOT/build/dc-syn
% cat Makefile
...
vars = \
        set ADDITIONAL_SEARCH_PATH      "... $(db_memories_dir) ..."
        set TARGET_LIBRARY_FILES        "cells.db SRAM32x128.db";\n \
...
```

Take a look at the synthesized hierarchy using Design Vision. Figure 4 shows the schematic view. Notice four 32x128 SRAM blocks on the right, and some control logic on the left.

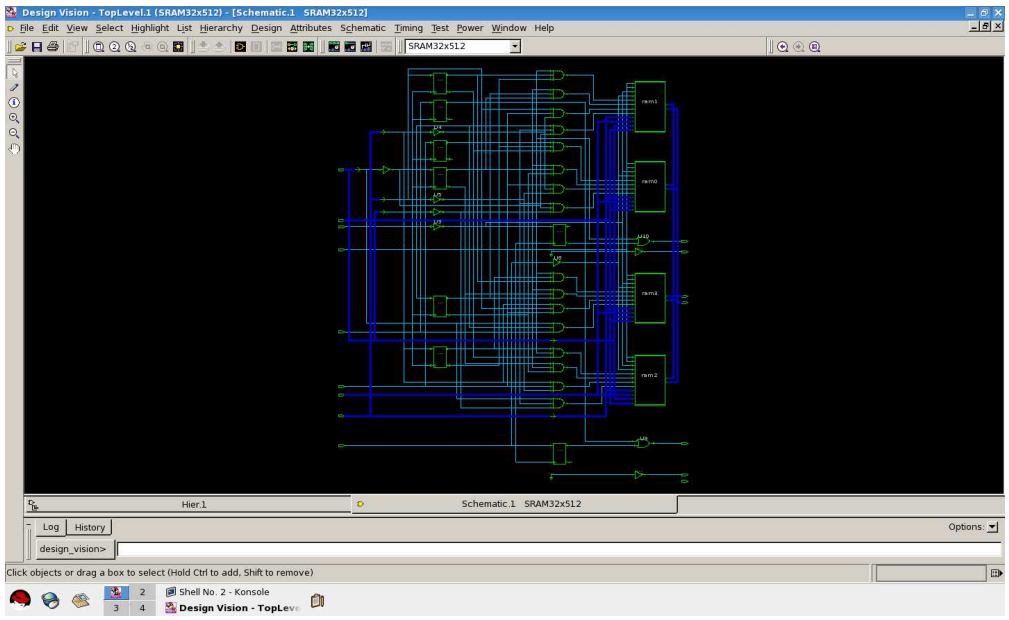


Figure 4: Schematic view of the synthesized 32x512 SRAM block

### Place and Route using Synopsys IC Compiler

Now place and route the SRAM block. After extracting parasitics information, go ahead and do post place and route gate-level netlist simulation to get the activity count. Don't forget to convert the outputs so that you can use those for power analysis.

```
% cd $TUT8_ROOT/build/icc-par
% make
% cd $TUT8_ROOT/build/vcs-sim-gl-par
% make run
% make convert
```

Make sure you included `SRAM32x128.mw` in your makefile.

```
% cd $TUT8_ROOT/build/icc-par
% cat Makefile
...
```

```
vars = \
        set ADDITIONAL_SEARCH_PATH      "$... $(mw_memories_dir) ...";\n \
        set MW_REFERENCE_LIB_DIRS       "$...$(mw_memories_dir)/SRAM32x128.mw";\n \
...
```

Figure 5 shows the place and routed SRAM block. Notice that four 32x128 SRAM blocks are placed in each corner.

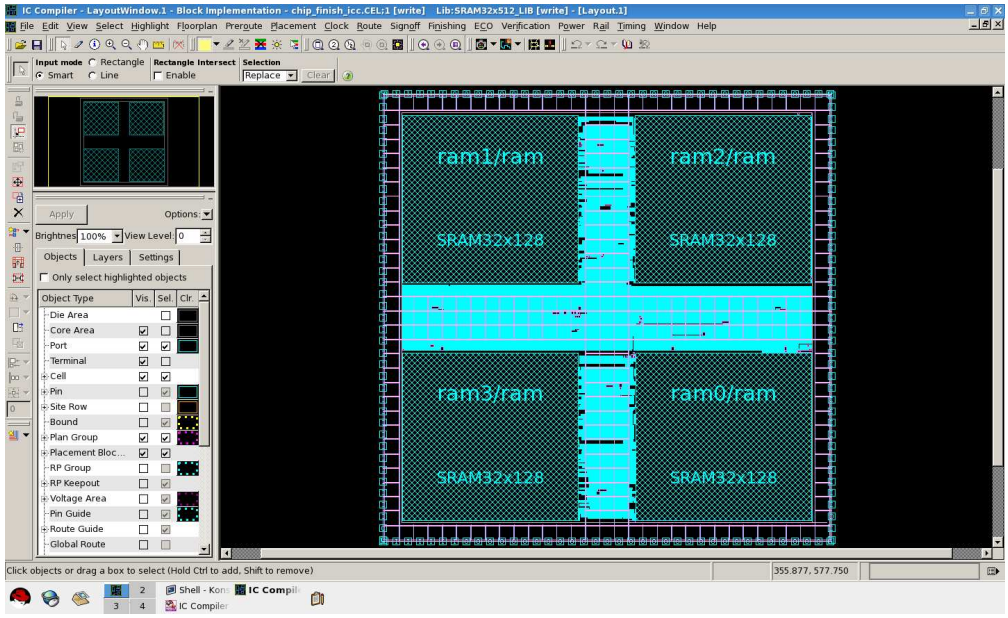


Figure 5: Place and routed 32x512 SRAM block

### Power Analysis using Synopsys PrimeTime PX

Now you can run it through the power analysis tools. Run the following commands.

```
% cd $TUT8_ROOT/build/pt-pwr
% make
```

Open the power analysis reports and see how much power is consumed.

```
% cd $TUT8_ROOT/build/pt-pwr
% cat current-pt/reports/vcdplus.power.avg.max.report
...
                                  Switch   Int      Leak     Total
Hierarchy                         Power    Power    Power    Power    %
--------------------------------------------------------------------------------
SRAM32x512                        3.00e-05 1.16e-03 8.31e-05 1.27e-03 100.0
  ram1 (SRAM32x128_wrap_3)        1.89e-06 3.57e-04 2.00e-05 3.79e-04  29.8
  ram0 (SRAM32x128_wrap_0)        1.89e-06 1.79e-04 2.00e-05 2.00e-04  15.8
  ram3 (SRAM32x128_wrap_1)        1.89e-06 5.36e-04 2.00e-05 5.58e-04  43.9
  ram2 (SRAM32x128_wrap_2)        1.89e-06    0.000 2.00e-05 2.19e-05   1.7
```

## Review

The following sequence of commands will setup the CS250 toolflow, checkout the SRAM32x512 example, synthesize, place and route, and do power analysis.

```
% source ~cs250/tools/cs250.bashrc
% mkdir tut8
% cd tut8
% cp -R ~cs250/examples/v-sram32x512/* .
% cd build
% make pt-pwr
```

## Acknowledgements

Many people have contributed to versions of this tutorial over the years. The tutorial was originally developed for CS250 VLSI Systems Design course at University of California at Berkeley by Yunsup Lee. Contributors include: Krste Asanović, Christopher Batten, John Lazzaro, and John Wawrzynek. Versions of this tutorial have been used in the following courses:

- CS250 VLSI Systems Design (2009) - University of California at Berkeley