Using the makemem template

V1.3 November, 2009

Originally by Allen Tanner, 2004, Modified by Erik Brunvand for CS/ECE 5710/6710

This document describes how to set up to use the makemem ROM generator with an example***. Important note:*** The makemem program claims to generate both ROM and SRAM. DO NOT USE THE SRAM! The ROM has been fabricated and works fine. The SRAM generated by this program is not reliable. Use the memCellsF09 SRAM instead if you're looking for SRAM!

Required directories and files:

1. First you need to edit your library path to point to the memCells library that contains all the pieces that makemem will use to create your memory structure. Edit your library path to include the following new library:

memCells //uusoc/facility/cad_common/local/Cadence/lib/mem/memCells

2. For EACH new memory that you want to build, you need to create a new Cadence library to hold that memory. This is because the makemem program generates a LOT of stuff in the library that holds your memory and you don't want it cluttering up your main cell library or your project library. If you make a new library for each memory, then you can include your memory circuit into your project out of that new library and you won't mess up your primary library. For this example of a ROM, I'll create a library named **rom64x32**. Make sure to attach the UofU_TechLib_ami06 technology to your new library.

3. Now you need to make a unix directory in which to run makemem. This is because of the same issue: makemem creates files that you want to keep localized in a directory. For this example I'll create a directory called ~/IC_CAD/mem-f09. You don't need a new dir for every memory, just one new dir from which to run makemem.

4. Before you start, you need to copy some files into your new IC_CAD/mem-f09 directory. From /uusoc/facility/cad_common/local/Cadence/lib/mem/cif copy memCells.cif and memCells.v to your IC_CAD/mem-f09 directory.

5. Now, for an example ROM definition file, copy /uusoc/facility/cad_common/local/Cadence/lib/mem/examples/ROMfiles/R64x32.rom to your IC_CAD/mem-f09 directory.

Now you will use the **makemem** java program to create the files for your ROM.

makemem: ; creates ROM files for Cadence

1. **cd ~/IC_CAD/mem-f09** ; use the working directory

2. **java –cp /uusoc/facility/cad_common/local/Cadence/lib/mem/j makemem –h**
   ; get the help message to check that things are set up properly

3. **java –cp /uusoc/facility/cad_common/local/Cadence/lib/mem/j makemem -r R64x32**

4. makemem will reproduce the binary contents of the ROM file (R64x32.rom) on the console log. You should make sure it is the data from the file.

5. makemem will produce three files:

R64x32.cif	; CIF file for layout import
R64x32.v	; verilog file for schematic import
R64x32.il	; Cadence SKILL code file for IO pin attachment

Your files will need to be brought into the CADENCE database. The following three procedures will read your files in.

CIF IN: **; creates a structure layout**

1. **CIW→File→Import→CIF**
2. Run directory: **~/IC_CAD/mem-f09**
3. Input file: **~/IC_CAD/mem-f09/R64x32.cif**
4. Top Cell name: ; this tells it to use the structure cell as the top cell name, leaving it blank will bring in all the cells
5. Library name: the name of the Cadence library you want to use for your ROM (rom64x32 in this example)
6. Make sure the option: **Do not over write cell views** is not asserted (this is inside the options menu, and should be set correctly unless you change it, so you probably don't have to worry)
7. You should get a notice that PIPO CIFIN has completed the no errors or warnings.

Verilog IN:

1. **CIW→File→Import→Verilog**
2. Target library name: **rom64x32**
3. Reference libraries: **memCells** (remove sample) **basic**
4. Verilog files to import: **~/IC_CAD/mem-f09/R64x32.v**
5. –v Options: **~/IC_CAD/mem-f09/memCells.v**
6. Make sure that the following is enabled: **Overwrite existing views**
7. You should see Verilog import completed using the memCells versions of the cells.

You now have two layouts, SR64x32 & RR64x32 in the library. There is also a schematic and symbol for the combined layout. All of these should reside in the library rom64x32. When you make changes to the ROM contents, and re-run the processing steps you will get new RR64x32 layout and a new SR64x32 schematic and symbol (the symbol will be new, but it won't change unless the structure changes, then of course you will also need a new SR64x32 layout.

The generated ROM is in two parts: SR64x32 which is the decoder and the support structure for the ROM, and RR64x32 which is the ROM contents. This is so that once you put the ROM into a larger layout, you can replace the contents (to fix a bug, for example), just by replacing the "dockable" ROM contents within the ROM structure. But, it also means you need to assemble these pieces before you have a working ROM.

Now use CADENCE (in the rom64x32 library) to assemble your generated layout.

Layout:

1. Open the layout view of SR64x32.
2. Insert an instance of RR64x32 at exactly X: 0.0 and Y: 0.0. Use the controls at the top of Virtuoso to guide you. You will probably want to zoom into the origin for fine control.
3. Save the new combined layout.
4. Use DRC to check to see if RR64x32 is in the right place.
5. Place the IO pins into the design

Add IO pins to your layout.

IO Pins:

1. The next step is to put I/O pins into the layout.
2. **CIW_window→load "~/IC_CAD/mem-f09/R64x32.il"**
3. This will return a "t" (no quotes) if it is successful.
4. **CIW_window→make_ pads("myLib" "SR64x32")**
5. This will place all of the IO pads into the layout. It also returns a "t" if successful.
6. If it fails, you can use undo in the layout view to reverse the action.
7. Use DRC, Extract & LVS tools to check everything.

At this point you will have complete ROM layout, schematic and symbol that can be simulated and integrated into your design hierarchy.