Using makemem template

V1.2 November 18, 2004

Modified by Erik Brunvand for CS/EE 6710 – Nov 2006

This document describes how to set up to use makemem the ROM and SRAM generator. It also gives an example of creating each type of file.

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 (using the library manager **Edit -> Library Path** or by editing your cds.lib file) to include the following new library:

memCells /uusoc/facilities/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

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

5. Now, for an example ROM definition file, copy /uusoc/facilities/cad_common/local/Cadence/lib/mem/examples/ROMfiles/**R64x32.rom** to your IC_CAD/mem 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** ; use the working directory

2. **java –cp /uusoc/facility/cad_common/local/Cadence/lib/mem/j makemem –h**

   The –h gets the help message to check if 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 look to see if 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. Make a new Cadence library to hold your new ROM. I'll call mine **R64x32**
2. **CIW→File→Import→CIF**
3. Run directory: **~/IC_CAD/mem**
4. Input file: **~/IC_CAD/mem/R64x32.cif**
5. Top Cell name: <leave blank> ; this tells it to use the structure cell as the top cell name, leaving it blank will bring in all the cells
6. Library name: your new library to hold these cells. i.e. **R64x32** if that's what you named it
7. Make sure the option: **Do not over write cell views** is <u>not</u> asserted (this is inside the options menu, and should be set correctly unless you change it, so you probably don't have to worry)
8. You should get a notice that PIPO CIFIN has completed with no errors or warnings. Actually, you'll probably get one warning that you didn't assert the Do Not Over Write Cell Views option. But since this is the way you wanted things set, this is ignorable. The **PIPO.LOG** file will be in the directory you chose as your run directory (~/IC_CAD/mem in this case).

Verilog IN:

1. **CIW→File→Import→Verilog**
2. Target library name: **R64x32**
3. Reference libraries: **memCells** (remove sample) **basic**

4. Verilog files to import: **~/IC_CAD/mem/R64x32.v**
5. –v Options: **~/IC_CAD/mem/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. Note that you have to have added the memCells library to your library path for this to work with no errors!

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 **R64x32**. 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 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. You want the cell origin when you use the "q" properties key to be at 0,0. The RR cell overlaps the nor decoder by 3.9 um and the top row of "pup" pullups by 1.2um.
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. Have the **SR64x32** layout open when you do these steps.
2. **CIW_window command field → load "~/IC_CAD/mem/R64x32.il"**
3. This will return a "**t**" (no quotes) if it is successful.
4. **CIW_window→make_ pads("R64x32" "SR64x32")**
   R64x32 is the library name, SR64x32 is the overall ROM layout cellview name
5. This will place all of the IO pads into the layout. It also returns a "**dbxxxxxx**" if successful which is the reference to the database object that was updated.
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 layout, schematic and symbol that can be integrated into your design hierarchy.

Note about the ROM file format – the first line of the file describes the format for the file. The syntax is

: <base> <rows> <columns> <optional – width of tristate bus on output>

In R64x32 that first line is ":16 64 32" which means hex format (base 16), 64 lines, 32 bits per line. In chars10.rom that first line is ":2 32 128 8" which means binary format (base 2), 32 rows, 128 bits per row, grouped into groups of 8 in the output tristate bus. I know that bases 2 and 16 are supported. I don't know about other bases…

The rest of a ROM definition file is the data that should be in the ROM, one ROM-row per line.

This is an exercise in creating an SRAM.

makemem: ; creates SRAM files for Cadence

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

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

3. **java –cp /uusoc/facility/cad_common/local/Cadence/lib/mem/j makemem -s 32 8**

4. makemem create a 32x8 SRAM..

5. makemem will produce three files:

SRAM32x8.cif	; CIF file for layout import
SRAM32x8.v	; verilog file for schematic import
SRAM32x8.il	; Cadence SKILL code file for IO pin attachment

CIF IN: ; creates an SRAM layout (remember to do this in a new library! – I'll use SRAM32x8 as the library name in this example)

1. **CIW→File→Import→CIF**
2. Run directory: **~/IC_CAD/mem**
3. Input file: **~/IC_CAD/mem/SRAM32x8.cif**
4. Top Cell name: **SRAM32x8** ; this tells it to use the structure cell as the top cell name
5. Library name: **SRAM32x8**
6. Make sure the option: **Do not over write cell views** is <u>not</u> asserted

Verilog IN:

1. **CIW→File→Import→Verilog**
2. Target library name: **SRAM32x8**
3. Reference libraries: **memCells** (remove sample) **basic**
4. Verilog files to import: **~/IC_CAD/mem/SRAM32x8.v**
5. –v Options: **~/IC_CAD/mem/memCells.v**
6. Check for: **Overwrite existing views**

You now have the SRAM layouts **SRAM32x8** in the library. There is also a schematic and symbol for the layout. All of these should reside in the new library that you created.

Layout:

1. Open the layout view of SRAM32x8.
2. Place the IO pins into the design as described below

IO Pins:

1. The next step is to put I/O pins into the layout.
2. **CIW_window→load "~/IC_CAD/mem/SRAM32x8.il"**
3. This will return a "**t**" (no quotes) if it is successful.
4. **CIW_window→make_ pads("SRAM32x8" "SRAM32x8")**
   The first **SRAM32x8** is the library name, and the second is the cell name. Adjust as needed…
5. This will place all of the IO pads into the layout. It returns a "**db:xxxxxxxx**" if successful. which is the pointer to the database object that was updated.
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.

The help text for makemem is as follows:

```
103 lab1-12:~/IC_CAD/mem> java -cp /uusoc/facility/cad_common/local/Cadence/lib/mem/j makemem -h
makemem v2.2  Nov 8, 2004
  Allen Tanner University of Utah CS6710

Enter the following:
java makemem choice options
   Where: choice selects the creation of either ROM or SRAM.
    for ROM  enter:-r rname       : rname.rom  is the file name.
                                  :
    for SRAM enter:-s  r c          : Version 1 SRAM single port.
    for SRAM enter:-s1 r c          : Version 2 SRAM single port.
    for SRAM enter:-s2 r c          : Version 2 SRAM dual port.
    for SRAM enter:-s3 r c          : Version 2 SRAM triple port.
                                    : r is the number of rows (decimal).
                                    : c is the number of columns (decimal).
                                    :
              :-h -H                 : help (no processing occurs when help is requested).
              :-f fname              : output file name. Used with .cif, .v & .il files.
              :-n sname rname       : sname for array top cell name.
              :                     : rname for ROM (only) dockable ROM array top cell name
              :-t n                 : use tristate buffers on the outputs of ROM.
              :-q                   : output hello.txt file to find the working file directory.
104 lab1-12:~/IC_CAD/mem>
```

For SRAM there are minor differences in the SRAM v1 and v2 cells that I can't remember what they are. I suspect the v2 cell is better, but it may be that it's just different and able to accept multiple ports. It also looks like the v1 SRAM cell is a little wider and shorter than v2 so the SRAMs using v1 cells will tend to be a little wider and shorter than the same size SRAM using the v2 SRAM cells.