COMPSYS 701 – Additional notes for using ReCOP assembler and initialising program memory

Introduction

The aim is to learn how to initialise memory modules (parts 4-7).

Prerequisites:

- Cygwin (with Python) installed, use cygwin1.dll from the provided package

Part One – Memory Initialisation

If you use an Altera Megafunction to instantiate on-chip memory, it allows you to use a .mif file to initialise it. This will be necessary because you will want to initialise your ReCOP Program Memory at startup. The .mif file itself is pretty simple:

```
WIDTH = 16;  ← The memory bus width
DEPTH = 32768;  ← The number of rows of memory (total size = width x depth)

ADDRESS_RADIX = HEX;  ←This usually makes life a little easier
DATA_RADIX = HEX;

CONTENT
    BEGIN
        [00..3FF]: FFFF;  ←Initialises everything to 1s
        0     :3400;  ←Memory address 0, Data is 3400 (everything is in hex)
        1     :4000;
        2     :0000;
        3     :FB00;
        4     :4010;
        5     :0000;
        …
    END;
```

When you write programs for ReCOP in Assembly, this file can be generated by mrasm.exe (see the last section of this lab) – but you can also just write it yourself by hand if you want to. If you want to initialize memory for simulation purposes in ModelSim, you will need to convert it to a .hex file, which is very similar. The easy way is to open the .mif file in Quartus II and then save as a .hex, but you should also have a look at the actual structure of a HEX file.

For the ModelSim testbench, you will need a MIF file. We will just create an arbitrary file – normally you would write the program in Assembly and assemble it. Use a python script to generate a file with

a width of 16-bits and a depth of 256 words. Then, make it procedurally fill in the memory with some algorithm of your choice – for example, you could set each word to be one more than the previous word, or shift a bit to the left, or any other logical progression. Run the script to create the file, and open it up in Notepad++ or similar to check the contents.

Part Two – Using the ReCOP Assembler

Find a mrasm.zip in project resources. In here, there is an application that will convert Assembly (based on the ReCOP ISA) into HEX and MIF files for you! You should be familiar with Assembly from COMPSYS304, but here is an example .asm file:

```
start NOOP ;starting the program
           LDR R1 #1
           LDR R4 #16 ;16 bits before looping
count      SUBV R4 R4 #1
           PRESENT R4 start ;if R4=0 go to start
           ADD R1 R1 R1 ;"double" the number
           LDR R0 #0
           SSOP R0
           LDR R2 #65535 ;max register size, 16 bits
time       SUBV R2 R2 #1
           PRESENT R2 count ;if R2=0 go to count
           JMP time
ENDPROG
END
```

In Cygwin, you can run `./mrasm.exe [asmfile]`, and it will create a rawOutput.hex and rawOutput.mif. Be aware that this will overwrite any existing rawOutput files. You can use the -o flag to set a filename (without an extension) if necessary, e.g. to convert simple.asm into test.hex and test.mif:

```
./mrasm.exe test.asm –o test
```

The application essentially runs off the provided newr.ini, which has a list of all the ReCOP instructions and the relevant addressing modes and opcodes. If necessary, you can add your own opcodes/instructions – open it in Notepad++ and follow the same format as the other instructions.

NOTE: When you use the assembler, it assumes a data depth of 1024. This is likely to be incorrect for your implementation as your program memory should have a much larger depth than that. When you use it for a real ReCOP, you will need to edit the file to change the data depth to the appropriate value or the assembler may ignore your file.

Part Three – Extra instructions

To build and run software on the given ReCOP the following process should be followed:

1) Write the ReCOP code in an ***.asm*** file.
2) Your ***.asm*** file needs to be assembled (mrasm.exe <filename>.asm) which generates two memory files: *rawOutput.hex* and *rawOutput.mif*
3) The .mif file then needs to be reloaded via the Quartus interface (processing > update Memory Initialization File)
4) The Quartus assembler needs to be run again (processing > start > start Assembler). At this stage you **must not recompile** the whole design as the prepared memory contents in the previous step will be lost.
5) The FPGA board can be configured as usual (through the Programmer interface).

Steps 3 and 4 can be performed from the command line by using the following instructions (if it is preferred):

quartus_cdb --update_mif <project name>
quartus_asm <project name>