A note on including memories in your design

This note is for the students who are facing difficulty related to memories in their mini projects. Let me address the following two issues related to memory usage in Xilinx FPGAs.

1. How to create a memory

2. How to put the initial contents in it

There are two ways to include a memory in the design -

a) Declare a signal whose type is an array of std_logic_vector and leave it to the synthesizer to create a memory. The memory created will either be a distributed memory (formed using look up tables (LUTs)) or block memory (formed using BRAMs). The choice made by the synthesizer depends on the size of the array and how the array is accessed. Block memory is preferred for larger sizes and distributed memory for smaller sizes. For each type of memory, there is a size limit defined by the resources available in the FPGA being used. Block memories can have one or two ports, whereas distributed memories can have upto 4 ports. Note, however, that in case of a distributed memory, having 2-ports/4-ports results in using twice/four times the number of LUTs compared to a single port memory of the same size. Port requirement depends on the number of simultaneous accesses to the array. For example, if you write an expression in VHDL involving 9 array elements as operands, it implies a requirement of 9 memory ports. To create a memory with a large number of ports, the synthesizer attempts to create an array of individual registers. This is feasible for very small arrays only.

You can also use the attributes ram_style and rom_style to instruct the synthesis tool on how to infer memory type for your array. This attribute can have the values “block” or “distributed”. Examples are given below.

```
attribute ram_style : string;
attribute ram_style of ram1 : signal is "distributed”;
attribute ram_style of ram2 : signal is “block”;
attribute rom_style : string;
attribute rom_style of rom_small : signal is “distributed”;
attribute rom_style of rom_large : signal is “block”;
```

b) Use IP catalog of Vivado to define a memory of the required type (block or distributed) with required properties (such as size, ports etc.). A new memory entity will be created which you can instantiate in your design. After creating a memory using the IP catalog, you will be able to see a VHDL file containing the new memory entity. From the entity declaration, you can note down the entity name and port names.

At the time of design compilation, it would be expected that the memory entity is compiled before the architecture instantiating that entity is compiled. This can be ensured by arranging the source files in appropriate order. Alternatively, include a

component declaration for this memory in the architecture where the memory is instantiated. Name of this component and its ports can be same as the entity name and port names of the memory created. Component declaration serves the same purpose as function prototype in the programming languages. It allows an architecture to be compiled before the entity instantiated in it is compiled.

For putting the initial contents in a memory there are two ways.

a) If you are defining memory as an array, you can provide initial values of array elements in the declaration itself. This method is ok only for small arrays.

b) For memory generated using IP catalog, define initial memory contents in a file (.coe file). The file name is specified interactively while generating the memory. Format for .coe file is easily available on internet. You may write a program in your favorite programming language (C, Python, Java …) to create the required .coe file.