Contents

1 FABulous Tutorial

1.1 Task 1: Start FABulous and Build a First Fabric

1. **Preparation (Directory and Environment):**
   - Follow the FABulous Quick Start Guide to set up your environment.
   - Additionally, make sure you have the FABulator graphical FPGA Editor installed. You can install with the following command:
     `FABulous install-fabulator`
2. **Generate a Template Project:**
   - We first generate a new template project:
     `FABulous create-project demo`
   - This creates a directory structure under `./demo/`.
3. **Customize the Fabric:**
   - Go into the `demo` root and open `fabric.csv` in a spreadsheet or editor program to explore the file.
   - **Resize the Fabric:** Resize the provided fabric by **removing the last two CLB (Configurable Logic Block) columns**.
     - They are named `LUT4AB`. Be careful not to interfere with other columns in the file.
     - When you shift the `RAM_IO` to the left, **don't forget the termination tiles**.
   - **Modifying Rows:** You could also add or remove rows, but that must be done in **units of two CLB rows** due to the DSP supertile which is two basic tiles (our CLBs) in height.
     - Moreover, that changes the interface of the fabric as we use connections at the left and right border of the fabric in the template.
   - *Note:* It is possible to replace the north and south termination tiles with I/Os.
   - Save the `fabric.csv` file.
4. **Build the Fabric (FABulous Shell):**
   - Start your project with the FABulous shell:
     ```
     cd demo # go to your demo project folder
     FABulous start # start the FABulous shell
     ```
   - Follow the steps displayed inside FABulous (starting with `load_fabric`).

- *Tip:* Play with `<TAB>` when entering commands; you can also fetch previous commands using the `<UP>` key.
- **Run the build sequence:**
  1. `FABulous> run_FABulous_fabric` (generates the fabric RTL, eFPGA_geometry and CAD tool models).
  2. `FABulous> run_FABulous_bitstream ./user_design/sequential_16bit_en.v` (generates the bitstream for a simple 16-bit counter design).
  3. `FABulous> run_simulation fst ./user_design/sequential_16bit_en.bin` (runs a functional simulation of the design with the generated bitstream).

5. **View the Result in FABulator:**
   - When done, open the FABulator from the FABulous shell with:
     - `FABulous> start_FABulator`
   - Inside, run **File → Open** `demo/eFPGA_geometry.csv`.
   - Download the counter test example design with **File → Open FASM**.
     - FASM (FPGA Assembler) is a human-readable textual bitstream format (check it out).
     - The FASM that we generated is in `demo/user_design`.

Congratulation, you just built your first FPGA! (We do the GDS part later).

- **Under the Hood:**
  - This was quite automated. If you want to explore what happened under the hood, follow the steps in:
    - FABulous Building the Fabric.
  - For that, check what is added in `demo/Tiles/LUT4AB` after each step.

1.2 Task 2: Customizing a Tile

1. **Examine the Register File:**
   - Open a file browser and change into the `RegFile` directory under `demo/Tiles/`.
   - Open `RegFile_32x4.v` which contains the BEL (Basic Element) and study how the two configuration bits `ConfigBits` are used.
   - Note the commented lines 17 and 18 that specify symbolic names for the configuration bits.
   - You may also check the `LUT4AB` tile as another example.
2. **Understanding the Architecture:**
   - The tile implements distributed memory in the form of a small register file primitive that is **4-bit wide and 32 entries deep**.
   - The primitive (also called BEL) has **1 write and two read ports**.
   - **Comparison with Xilinx/Altera:** The two read ports are not supported in distributed memory primitives of Xilinx or Altera FPGAs.

  - They would need two distributed memory LUTs per bit of, let's say, a RISC-V Register file, while we would need just 8 of our RegFile primitives for this case.

3. **Customization Task:**
   - **Goal:** Add the possibility that, depending on a **third parameter (configuration bit)**, we will return a 0 if we read from address 0.
   - **Rationale:** This feature would save some LUTs in case we want to implement a RISC-V CPU on our FPGA because register 0 is hardwired to 0 in RISC-V.
   - **Implementation Method:**
     - We could make a copy of the `RegFile` (or `LUT4AB` tile) to derive a new custom tile.
     - This would also require us to derive a new custom tile descriptor in the `fabric.csv` file.
   - **Interfacing Notes:**
     - If we stick to the same BEL interface, that is all it takes to support a custom tile (**for instance something tailored to ML inference**).
     - Otherwise, the interfacing takes a few manual steps that are already automated. A tutorial for that is in the documentation: FABulous Fabric Automation

---

1.3 Homework

1. Write a wrapper for your custom primitive and instantiate that in `sequential_16bit_en.v` (located in the `/user_design` sub folder).
2. You can compile that inside FABulous with `run_FABulous_bitstream`.

1.3.1 Example Reference: DSP Primitive Instantiation

An example that instantiates the DSP primitive is provided in the GitHub repository below.

- **Check lines 120-140**.
- **Check line 61**.
- **Link:** vga_bram_mul.v