FPGA Ignite Summer School 2024
Heidelberg University, August 5th to 9th

Task 1 – Start FABulous and build a first fabric

Open a shell and change into the directory: ~/summer_school/FABulous

Start FABulous with (the -h switch shows you the help page)

```
python3 FABulous.py -h
```

This will throw an error as you must define the **FAB_ROOT** environment variable:

```
export FAB_ROOT=$PWD
```

Being able to set a different root was considered to be useful in case we play with different versions…

Restart **python3 FABulous.py -h**

We first generate a new template project using:

```
python3 FABulous.py -c Ignite2024_demo
```

This creates a directory structure under ./Ignite2024/. Go into the Ignite2024 root and open fabric.csv in a spreadsheet program and explore the file!
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. Also, when you shift the RAM_IO to the left, don't forget the termination tiles.

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 and then start our project with:

```
python3 FABulous.py Ignite2024
```

Follow the steps displayed inside FABulous (starting with **load_fabric**)

Note to play with <TAB> when entering commands, you can also fetch previous commands using the <UP> key.

When done with run_FABulous_fabric (generates the fabric RTL), gen_geometry (generates the model for the FABulator graphical FPGA Editor), and run_FABulous_bitstream, (compiles and implements a simple counter), you can open the FABulator (start the icon on your desktop). Inside, run File → Open Ignite2024/eFPGA_geometry.csv. After this, you can 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 Ignite2024/user_design.

Congratulation, you just built you first FPGA!
(we do the GDS part later)

This was quite automated and if you want to explore what happened under the hood, follow the steps in: https://fabulous.readthedocs.io/en/latest/Building%20fabric.html
For that check what is added in rt_demo/tiles/LUT4AB *after each step.*

Task 2 – Customizing a tile

Open a file browser and change into the RegFile directory under **Ignite2024**/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.

> *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. 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).*

Add now the possibility that, depending on a third parameter (configuration bit), we will return a 0 if we read from address 0. 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.

> *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. 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 and that will be released soon.*

Homework

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

An example that instantiates the DSP primitive is provided in:

https://github.com/gatecat/fabulous-mpw2-bringup/blob/main/sim/test_design/vga_bram_mul.v

Check lines 120-140
https://github.com/gatecat/fabulous-mpw2-bringup/blob/34bcb9d3bb253eb7651532fac6a44f3a3e31a732/sim/test_design/vga_bram_mul.v#L120-L140
and line 61
https://github.com/gatecat/fabulous-mpw2-bringup/blob/34bcb9d3bb253eb7651532fac6a44f3a3e31a732/sim/test_design/vga_bram_mul.v#L61

https://www.dropbox.com/scl/fi/hteaip8it9s4fitu6jlkd/FABulous_RT_Tutorial.pdf?rlkey=p9ikd13xhflsjw031kfj8hjnq&st=p5kia8zz&dl=0

https://acesse.dev/pJK6U