HW06 [ECE 720]

Digvijay Anand
200478940

Q1. Total leaf cells in LMS_pipe.hier:

Solution:

Note:
All my modifications are in hw06/p1/instcount.cpp
Run using: source setup.sh; make; make sim in hw06/p1/

Step - 1:

For this, I have created two struct to store defined and instantiated modules.

- Defined Module: All module definitions with a vector containing all the instances of sub-modules and leaf cells within it. It is instantiated later in the *sc_main()* as *dModules*.
- Instantiated Module: All modules with name and num_leaf_cells contained within it. This count of num_leaf_cells = 1 for leaf_cells or, num_leaf_cells_for_sub_module, otherwise. It is instantiated later in the *sc_main()* as *iModules*.

```
...
#include <vector>

using namespace std;

struct DefinedModule {
    string module_name;
    vector<pair<string, int>> instances;
};

struct InstantiatedModule {
    string instance_name;
    int num_leaf_cells;
};
...
```

Step - 2:

I created a function *countLeafCellsRecursively*, to count the number of total_leaf_cells in that module. In case, it is an instance, total_leaf_cells = 1 is returned. Otherwise, the total number of

leaf cells in the module including its sub-modules and leaf cells is returned. Both *DefinedModule* and *InstantiatedModule* hold this value along with the name.

Note: I have currently disabled a print statement in the defined function. This statement was helpful for verifying the number of leaf cells in each module and checking the call hierarchy for accuracy.

```
...
int countLeafCellsRecursively(const string& mod_name, const map<string,
module*>& mods) {
    if (mods.find(mod_name) == mods.end()) {
        return 1;
    }

    module* current_module = mods.at(mod_name);
    int total_leaf_cells = 0;

    for (const string& instance : current_module->instances) {
        if (mods.find(instance) != mods.end()) {
            total_leaf_cells += countLeafCellsRecursively(instance, mods);
        } else {
            total_leaf_cells++;
        }
    }
    //cout << "mod_name = " << mod_name << "\t --  " << total_leaf_cells <<
endl;

    return total_leaf_cells;
}

int sc_main(int argc, char* argv[])
{

    map<string,module*> mods;
    string line,first,second,current_module;
    size_t pos;
    ifstream f("LMS_pipe.hier");
    vector<DefinedModule> dModules;
    vector<InstantiatedModule> iModules;

    string last_module;
...
```

Note: In the above code, I included a string to track the last_module name, ensuring that the final count for the last module is easily obtained for the final output. Additionally, I added comments that print the dModules and iModules, which proved to be very helpful.

Step - 3:

Next, I modified the given while loop to initialize the last_module name in the *if part* and to add *dModules* and *iModules* in the *else if* part.

The *LSM_pipe.hier* file abstracts the underlying RTL in a bottom-up approach with the root module at the end. However, for accurately calculating the total number of leaf cells, a top-down approach is more effective to understand dependencies and accumulate the leaf cell count as we move up the hierarchy.

I incorporated this logic into the *else if* condition as shown below:

```
...
      //cout << "\"" << second << "\"" << endl;
      if (first == "module") {
        current_module = second;
        last_module = current_module;
        mods[current_module]=new module(current_module);
        //cout << "module " << second << endl;
      }
      else if (second != "") {
        mods[current_module]->addInstance(first);
        DefinedModule dModule;
        dModule.module_name = current_module;
        module* mod_obj = mods[current_module];

            for (const string& instance : mod_obj->instances) {
                if (mods.find(instance) != mods.end()) {
                    int leaf_cells_in_submodule =
countLeafCellsRecursively(instance, mods);
                    dModule.instances.push_back({instance,
leaf_cells_in_submodule});
                    iModules.push_back({instance, leaf_cells_in_submodule});
                } else {
                    dModule.instances.push_back({instance, 1});
                    iModules.push_back({instance, 1});
                }
            }

            dModules.push_back(dModule);
      }
...
```

Step-4:

Finally, I pass the last_module to the *countLeafCellsRecursively* function to get the total number of leaf_cells in the module, as shown below:

```
...
    int total_leaf_cells_in_final_module =
countLeafCellsRecursively(last_module, mods);
    cout << "\n#######################################" << endl;
    cout << "p1:\n" << endl;
    cout << "\nTotal leaf cells in the final module '" << last_module << "': "
<< total_leaf_cells_in_final_module << endl;
    cout << "\n#######################################" << endl;
...
```

Answer:

```
#######################################
p1:


Total leaf cells in the final module 'LMS_pipe': 10581

#######################################
```

Q2. Total leaf cells in LMS_pipe.hier:

Solution:

Note:
I have not changed any other file. *reader.h* is an addition to git.
Run using: source setup.sh; make; make sim in hw06/p2/

- For this, I wrote a class called *reader* such that it reads words from a file and stores them in a STL container: *vector* of strings.
- I have used *#pragma once* as a pre-processor directive to ensure the header file is included only once.
- I have used the *sstream* library to split a line into words.
- The constructor then takes a *filename* and checks if it exists. If it does, it reads line-by-line and the function *void reversePoint()* prints words in the reverse order. This can be easily done by reverse iterators *.rbegin()* and *.rend()*

Code:

```
#pragma once

#include <string>
#include <vector>
```

```
#include <sstream>
using namespace std;

class reader{
    private:
        vector <string> words;
    public:
        reader(const string& filename){
            ifstream file(filename);
            string line, word;

            if(!file){
                cerr << " Error: loading file " << filename << endl;
                return;
            }

            while(getline(file, line)){
                istringstream iss(line);
                while(iss >> word){
                    words.push_back(word);
                }
            }
        }
    void reversePrint() const {
        cout << "####################################" << endl;
        cout << "p2:" << endl;
        cout << endl;
        for(auto it =words.rbegin(); it !=words.rend(); ++it){
            cout << *it << endl;
        }
        cout << "####################################" << endl;
    }

};
```

Answer:

```
####################################
p2:

back
dog's
lazy
the
over
jumpped
fox
brown
quick
The
####################################
```