DSnP: Final Project

Student ID: R06921095, Name: Jui-Che Wu

Phone number: 0988195199, E-mail: r06921095@ntu.edu.tw

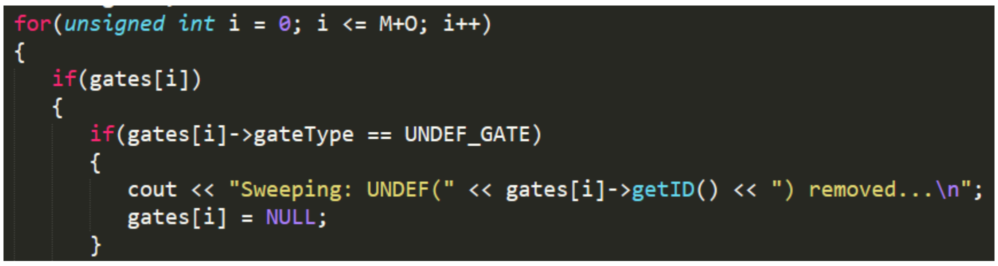
➢ Abstract

As for the time result, I only implement two part of the final project: sweep and optimize respectively. First, the part of parsing related to hw6, I use the code by other student published in github, which is listed in the reference. Then, I will detail about what I've done in the final project.

➢ Implementation

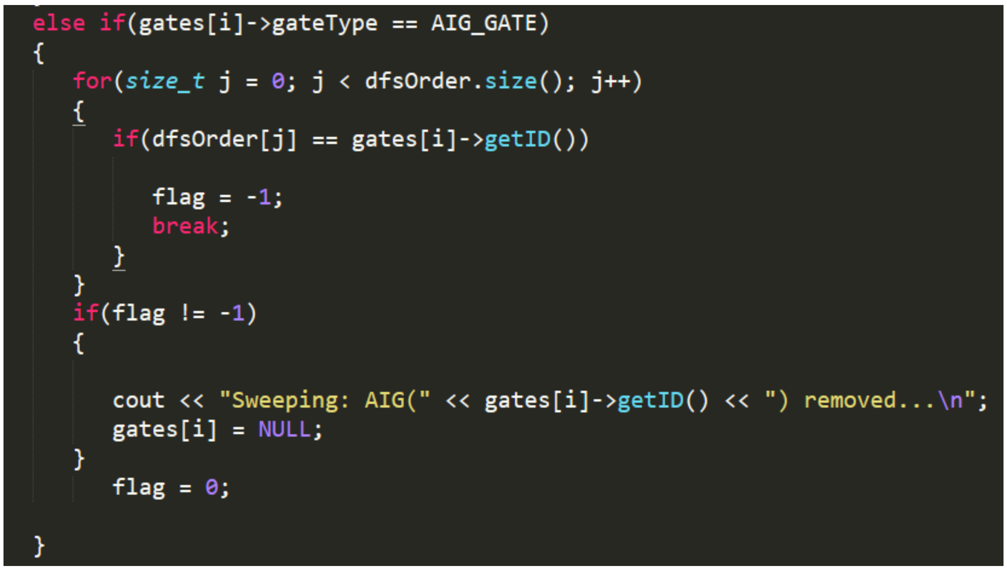
✓ Sweep

In this task, we have to remove the gates that can not be reached from POs. To remove the UNDEF_GATE, we can utilize the recorded while performing DFS. Traverse the whole linked list of the gates, if the gates Type is an UNDEF_GATE, we can set the address of the gate to be NULL. The code to remove UNDEF_GATE is shown below.



```
for(unsigned int i = 0; i <= M+O; i++)
{
    if(gates[i])
    {
        if(gates[i]->gateType == UNDEF_GATE)
        {
            cout << "Sweeping: UNDEF(" << gates[i]->getID() << ") removed...\n";
            gates[i] = NULL;
        }
```

Then, we also have to remove the AND_GATE, which can not reach the POs. To achieve so, we can utilize the dfsList recorded before while performing DFS. For the gates that are not in the dfsList, we can set their addresses to NULL. The code od removing unreached AND_GATE is shown below.



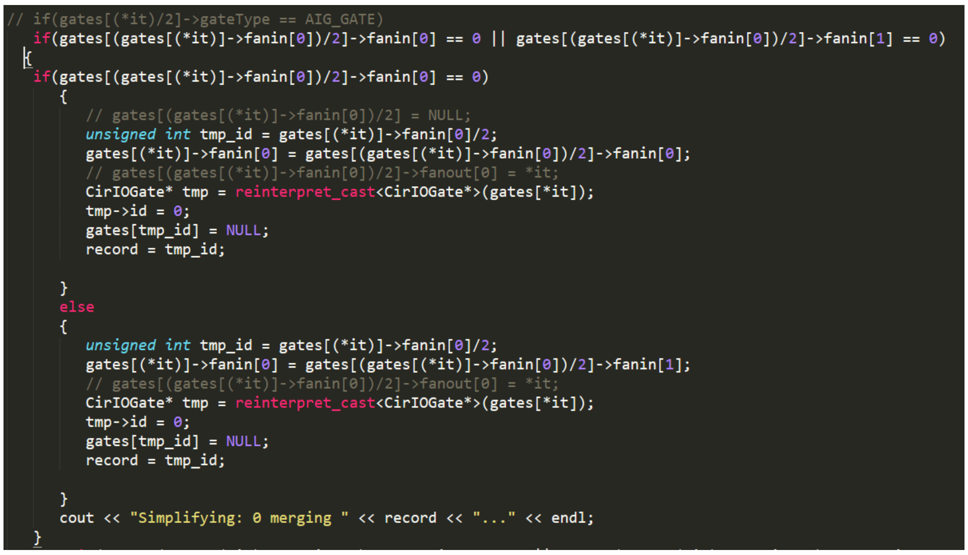
```cpp
else if(gates[i]->gateType == AIG_GATE)
{
    for(size_t j = 0; j < dfsOrder.size(); j++)
    {
        if(dfsOrder[j] == gates[i]->getID())
        
            flag = -1;
            break;
        }
    }
    if(flag != -1)
    {
    
        cout << "Sweeping: AIG(" << gates[i]->getID() << ") removed...\n";
        gates[i] = NULL;
    }
        flag = 0;

}
```

Last but not least, update the floatingList recorded before, so the command can execute correctly.

✓ Optimization

In this task, is to perform trivial circuit optimization, the implementation is more difficult compare to the previous task, which can be divided into four cases.
Case1: one of the input of AND_GATE is 0
In this case, we can simply replace the whole gate to a 0 input, which can save one gate. The implementation method is to set the "before gate" fanout to 0, and set the "after gate" fanin to 0 as well, and solve it recursively. The code of case 1 is shown as follow:



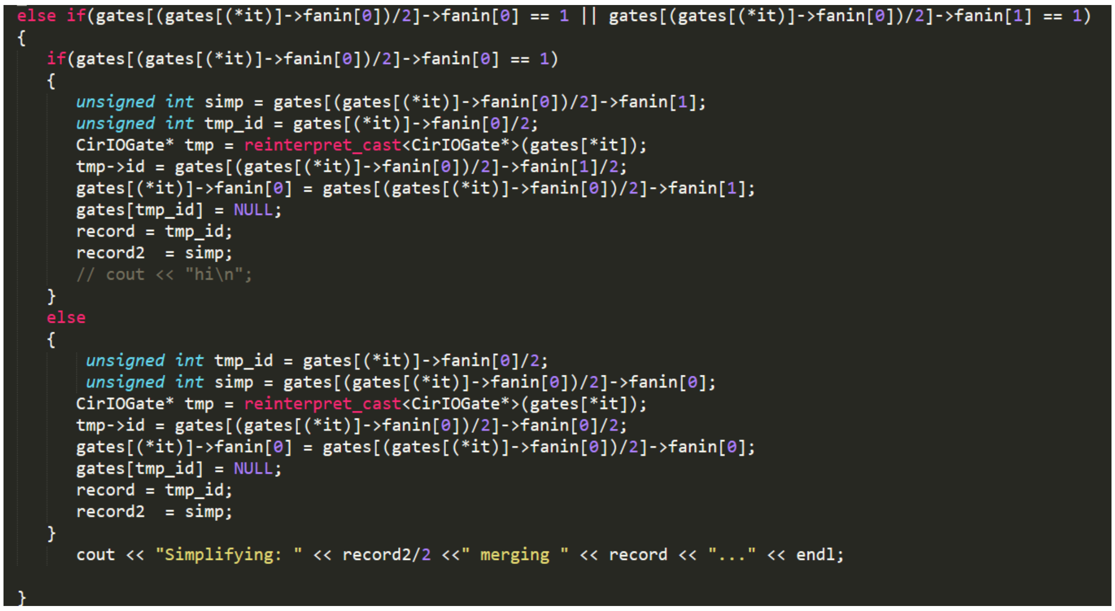
```cpp
// if(gates[(*it)/2]->gateType == AIG_GATE)
   if(gates[(gates[(*it)]->fanin[0])/2]->fanin[0] == 0 || gates[(gates[(*it)]->fanin[0])/2]->fanin[1] == 0)
  {
   if(gates[(gates[(*it)]->fanin[0])/2]->fanin[0] == 0)
      {
        // gates[(gates[(*it)]->fanin[0])/2] = NULL;
        unsigned int tmp_id = gates[(*it)]->fanin[0]/2;
        gates[(*it)]->fanin[0] = gates[(gates[(*it)]->fanin[0])/2]->fanin[0];
        // gates[(gates[(*it)]->fanin[0])/2]->fanout[0] = *it;
        CirIOGate* tmp = reinterpret_cast<CirIOGate*>(gates[*it]);
        tmp->id = 0;
        gates[tmp_id] = NULL;
        record = tmp_id;

      }
      else
      {
        unsigned int tmp_id = gates[(*it)]->fanin[0]/2;
        gates[(*it)]->fanin[0] = gates[(gates[(*it)]->fanin[0])/2]->fanin[1];
        // gates[(gates[(*it)]->fanin[0])/2]->fanout[0] = *it;
        CirIOGate* tmp = reinterpret_cast<CirIOGate*>(gates[*it]);
        tmp->id = 0;
        gates[tmp_id] = NULL;
        record = tmp_id;

      }
      cout << "Simplifying: 0 merging " << record << "..." << endl;
   }
```

Case2: one of the input of AND_GATE is 1

In this case, if one of the input is 1, we can also simplify the AND_GATE as one input (ex: if input1 is 1, the whole input of the current gate will be input2), which can also save one gate. The implementation method is to set the fanout of the “before gate” to input2, and the fanin of the “after gate” to input2 as well, then solve ie recursively. The code of case 2 is shown below.

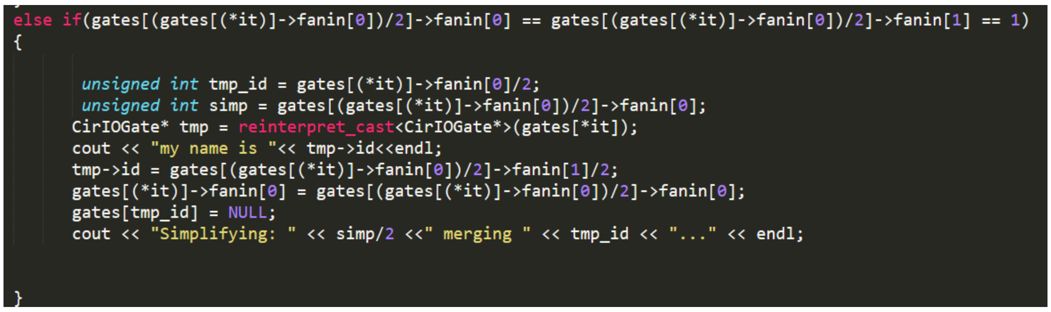


```cpp
else if(gates[(gates[(*it)]->fanin[0])/2]->fanin[0] == 1 || gates[(gates[(*it)]->fanin[0])/2]->fanin[1] == 1)
{
    if(gates[(gates[(*it)]->fanin[0])/2]->fanin[0] == 1)
    {
        unsigned int simp = gates[(gates[(*it)]->fanin[0])/2]->fanin[1];
        unsigned int tmp_id = gates[(*it)]->fanin[0]/2;
        CirIOGate* tmp = reinterpret_cast<CirIOGate*>(gates[*it]);
        tmp->id = gates[(gates[(*it)]->fanin[0])/2]->fanin[1]/2;
        gates[(*it)]->fanin[0] = gates[(gates[(*it)]->fanin[0])/2]->fanin[1];
        gates[tmp_id] = NULL;
        record = tmp_id;
        record2  = simp;
        // cout << "hi\n";
    }
    else
    {
         unsigned int tmp_id = gates[(*it)]->fanin[0]/2;
         unsigned int simp = gates[(gates[(*it)]->fanin[0])/2]->fanin[0];
        CirIOGate* tmp = reinterpret_cast<CirIOGate*>(gates[*it]);
        tmp->id = gates[(gates[(*it)]->fanin[0])/2]->fanin[0]/2;
        gates[(*it)]->fanin[0] = gates[(gates[(*it)]->fanin[0])/2]->fanin[0];
        gates[tmp_id] = NULL;
        record = tmp_id;
        record2  = simp;
    }
        cout << "Simplifying: " << record2/2 <<" merging " << record << "..." << endl;

}
```

Case3: 2 input are the same

In this case, we can simplify the gate as input1 (or input2). The implementation method is similar to the previous case, replace the fanout of “before gate” to input1 (or input2) and replace the fanin of the “after gate” to input1 (or input2). Then solve it recursively. The code is shown below.



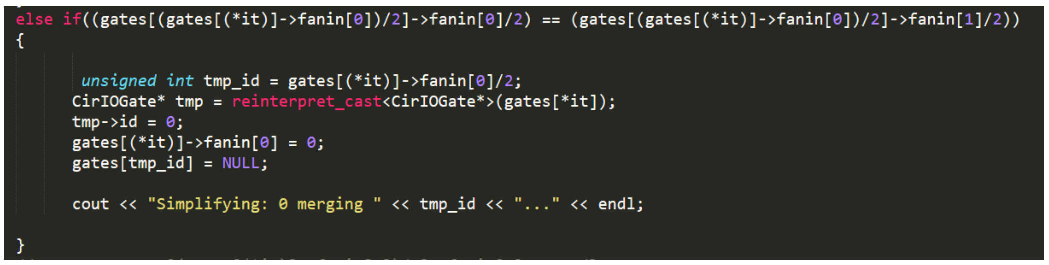
```cpp
else if(gates[(gates[(*it)]->fanin[0])/2]->fanin[0] == gates[(gates[(*it)]->fanin[0])/2]->fanin[1] == 1)
{

         unsigned int tmp_id = gates[(*it)]->fanin[0]/2;
         unsigned int simp = gates[(gates[(*it)]->fanin[0])/2]->fanin[0];
        CirIOGate* tmp = reinterpret_cast<CirIOGate*>(gates[*it]);
        cout << "my name is "<< tmp->id<<endl;
        tmp->id = gates[(gates[(*it)]->fanin[0])/2]->fanin[1]/2;
        gates[(*it)]->fanin[0] = gates[(gates[(*it)]->fanin[0])/2]->fanin[0];
        gates[tmp_id] = NULL;
        cout << "Simplifying: " << simp/2 <<" merging " << tmp_id << "..." << endl;


}
```

Case4: 2 input are inverse of each other

In this case, we can simplify the gate as 0 input due to the fact of “AND” of this case is 0. Thus, the implementation method of case 4 is similar to case 1, replace the fanout of “before gate” to 0, and replace the fanin of the “after gate” to 0 as well. The code of case 4 is shown below.



```
else if((gates[(gates[(*it)]->fanin[0])/2]->fanin[0]/2) == (gates[(gates[(*it)]->fanin[0])/2]->fanin[1]/2))
{

        unsigned int tmp_id = gates[(*it)]->fanin[0]/2;
        CirIOGate* tmp = reinterpret_cast<CirIOGate*>(gates[*it]);
        tmp->id = 0;
        gates[(*it)]->fanin[0] = 0;
        gates[tmp_id] = NULL;

        cout << "Simplifying: 0 merging " << tmp_id << "..." << endl;

}
```

➢ Reference

https://github.com/yan12125/DSnP