DSnP Final Project
— Fraig Study Report

張顥霆 (@daydreamer)
email: terry160434@gmail.com

January 21, 2016

1 Introduction

In this report, we will discuss about the implementation and design of my final project in the course Data Structure and Programming. The algorithm and also performance of the 5 operations: Sweep, Optimize, Strash, Simulation and Fraig will be discussed.

Special Thanks to 潘廣霖 for its circuit parser. As my HW6 is not really performing well, the parser in this program is in reference to him.

2 Data Structure

The basic gate class (CirGate) is defined for inheritance. In implementation, different type of gates will have its own class and thus have different behaviour.

The basic gate class (CirGate) is storing the gate's ID, input list and also it's output list. The input and output list store a list of size _ t, which is a pointer to CirGate with last bit being a flag of inverse input/ output.

Other class are inherited from the basic class and store their special information such as name for inputs and outputs.

The manager class (CirMgr) is storing four lists of CirGate*: the total gate list, input gate list, output gate list and the DFSList, the list those can be reached from POs in DFS order. Also storing the list of FECGroups.

3 Operations

3.1 Sweep

This operation should remove those gates that cannot be reached by the Output gates. Thus, the operation will go through the DFS List once and mark all those gates. Then go through the gate list and remove the unchecked gates.

3.2 Optimize

This operation is implemented by going through the circuit in DFS order from input gates to output gates.

We should check the following conditions of each AND gate we met:

1. The two input signals include a constant 0 input, then the gate should have a constant 0 output, which can be replaced by a constant 0 gate.

2. The two input signals include a constant 1 input, then the gate should have a output just as same as the other input gate.

3. The two input signals are inverse to each other, then the gate should have constant 0 output.

4. If the two gates are the same, then the gate output should be its input.

Under these conditions, the gate can be removed and replaced by another gate without affecting the final output. Thanks to the DFS order, each gate we met should have input gates which are already checked by the optimize operation. Thus, the result circuit of optimization should not be possible to simplify via optimize operation.

3.3 Strash

The operation check structure equivalent gates and merge them together. This is implemented by hashing the two inputs of a AND gate, check equivalent by collision. The hash function used is simply adding the two modified pointer(a pointer to CirGate with last bit being a flag of inverse input/ output), in order to let the gate collide even the inputs are in different ordering.

3.4 Simulation

Simulation operation includes random simulation and simulation with specified pattern in file.

The simulation also benefits from the DFS order. After assigning the value to the inputs, the value of each gate can be found by doing an binary AND operation of its two inputs.

Here we use a 64-bit integer to pack 64 input patterns and do parallel simulation for better performance.

The number of random simulation is controlled by the count of failing the split a group into 2 or more. If the count goes through a threshold, then just stop the simulation.

The threshold here is simply the logarithm of the number of AND gates. But the simulation gives a satisfying result.

3.5 Fraig

Fraig is implemented by checking if each pair of gates in the same FEC Group is equivalent. The FEC Groups are created by the simulation operation.

By calling miniSAT to solve the SAT clause that is created before, the result of the SAT solver controls the next step.

1. If UNSAT, merge the two gates
2. If SAT, use the input that can satisfy the clause to split the current FEC-Group.

This implementation is simple but extremely ineffective. Some optimization is suggested but not implemented due to lack of time. The performance should be explained more in detail in the performance test section.

Here are some ideas in the list,

1. Pack the inputs given by the SAT solver and do simulation operation to split large groups.
2. Lazy propagation of merging gates.

4 Performance Test

Performance test is done only on simluation and fraig operation. As the other 3 operations is fast enough.

4.1 Testing environment

- OS: OS X El Capitan Version 10.11.2
- Processor: 2.5 GHz Intel Core i7
- Memory: 16 GB 1600 MHz DDR3
- Compiler: Apple LLVM version 7.0.2 (clang-700.1.81)
- Compile arguments: CFLAGS = $-O3 - Wall$

4.2 Simulation

The table shown below gives the comparison of the time needed by simulation of different size. This is tested by the pattern related to the circuit file.

The N here is the number of gates in the circuit.

Input pattern	N	Time	Ref Program
896	753	0	0.01
11936	9642	0.66	0.77
22912	88410	5.03	4.27

Table 1: Comparison between my program and ref program (Time in sec)

4.3 Fraig

The table shown below gives the comparison of the time needed by simulation of different size. This is tested by the pattern related to the circuit file.

The N here is the number of gates in the circuit.

Input pattern	N	Time	Ref Program
896	753	0.03	0.03
11936	9642	9.22	2.57
22912	88410	160.8	72.44

Table 2: Comparison between my program and ref program (Time in sec)

As we can see here, the time that cost by fraiging the largest case of my program is a double of the reference program. It is predicted the difference will increase when the size of input gets larger.