Protokoll zum Praktikum Äquivalenzprüfung von Schaltkreisen

Fakultät Informatik
TU Dresden

Christian Kroh
Matrikelnummer: 3755154
Studiengang: Informatik (Diplom)
Jahrgang: 2010/2011

26. Dezember 2013, Dresden

Inhaltsverzeichnis

1 Programm

1.1 Entwurf

1.1.1 Circuit

Das Grundgerüst des Programms bildet die `Circuit`-Klasse, die einen Schaltkreis und seine Funktionalitäten implementiert. Die Eingangsbelegung des Schaltkreises wird durch eine Zustandsvariable `unsigned int state` repräsentiert, die einen Wert zwischen 0 und 2^N annehmen kann, wobei N die Anzahl der Eingänge ist.

Die Eingänge (`Input`-Objekte), Gatter (`Gate`-Objekte) und Ausgänge (`Output`-Objekte) werden in jeweils eigenen Maps abgelegt, deren Keys vom Typ `std::string` die Bezeichner der jeweiligen Objekte sind. Alle Gatter sind als Klassen implementiert, die von der abstrakten Klasse `Gate` erben. Bei manchen Gattern - wie AND, Or, usw. - können beliebig viele Eingänge definiert werden. Ein Gattereingang ist ein Zeiger, auf entweder den Wert eines `Input`-Objektes oder auf den Ausgang eines anderen `Gate`-Objektes. Die Ein- und Ausgänge der Gatter sind vom Typ `bool`.

Bei der Simulation eines Schaltkreises wird über den `state` der `Circuit`-Klasse iteriert und dessen Wert in Binärdarstellung auf die Eingänge abgebildet. Nach jeder neuen Eingangsbelegung müssen die Gatter durchlaufen werden, bis das neue Signal die Ausgänge erreicht.

1.1.2 Parser

Zum Parsen wird die Klasse `Parser` als Grundgerüst zur Verfügung gestellt, wobei die genauere Implementierung für unterschiedliche Benchmarks in von dieser Klasse erbenden Klassen verschoben wurde. Für das Benchmark BENCH ist eine solche gleichnamige Klasse vorhanden.

Ein Parser kann Schaltkreise aus Dateien lesen und diese anschliesend als `Circuit`-Objekte zurückgeben.

Der Parser für das BENCH-Format geht davon aus, dass in der einzulesenden Datei zuerst alle Eingänge (INPUT), dann die Ausgänge (OUTPUT), danach die ggf. vorhandenen FlipFlops (DFF) und zuletzt die restlichen Gatter definiert werden.

Um einen Schaltkreis aus einer Datei zu Parsen, müssen dem Programm die Argumente
`-p BENCHMARK DATEI` mitgeteilt werden.

1.1.3 Simulator

Die Klasse `Simulator` enthält eine Liste von geparsten `Circuit`-Objekten, die simuliert werden können. Dazu wird wieder über den Zustand der Schaltkreise iteriert.

Um zum Beispiel zwei Schaltkreise aus Dateien zu parsen und zu Simulieren müssen folgende Argumente übergeben werden:
`-p BENCHMARK1 DATEI1 -p BENCHMARK2 DATEI2 -s`

1.2 Äquivalenzprüfung duch Simulation

1.2.1 Voraussetzungen

- Schaltkreise müssen gleichviele Eingänge bzw. Ausgänge besitzen
- falls in einem Schaltkreis ein Eingang bzw. Ausgang vorkommt, muss ein gleichnamiger Eingang bzw. Ausgang auch in den anderen Schaltkreisen vorkommen
- es können beliebig viele Schaltkreise verglichen werden, falls einer nicht äquivalent mit einem anderen ist, wird `false` zurückgegeben
- ist kein Schaltkreis definiert, wird `false` zurückgegeben
- ist nur ein Schaltkreis definiert, wird `true` zurückgegeben

1.2.2 Vorgehen

1. über mögliche Zustände (2^N mit N ... Anzahl der Eingänge) iterieren
   a) durch Schaltkreise iterieren
      i. Zustand auf Eingänge abbilden
      ii. Schaltkreis traversieren und Ausgangsbelegung ermitteln
   b) Belegungen mit denen des vorangegangenen Schaltkreises Vergleichen
   c) Bei unterschiedlicher Belegung wird `false` zurückgegeben
   d) Sonst wird die Iteration über die Zustände fortgesetzt
2. wurde über alle 2^N Zustände iteriert und keine Varianz der Ausgangsbelegung bei den Schaltkreisen festgestellt, wird `true` zurückgegeben

1.2.3 Implementierung

Für alle Circuits werden die möglichen Inputs berechnet und die erhaltenen Outputs miteinander verglichen. Sollte dabei ein Circuit enthalten sein, der abweicht, wird false zurückgegeben (Circuits sind nicht äquivalent).

1.3 Äquivalenzprüfung durch SAT-Solver

1.3.1 Voraussetzungen

- Schaltkreise müssen gleichviele Eingänge bzw. Ausgänge besitzen
- falls in einem Schaltkreis ein Eingang bzw. Ausgang vorkommt, muss ein gleichnamiger Eingang bzw. Ausgang auch in den anderen Schaltkreisen vorkommen
- es können beliebig viele Schaltkreise verglichen werden, falls einer nicht äquivalent mit einem anderen ist, wird `false` zurückgegeben
- ist kein Schaltkreis definiert, wird `false` zurückgegeben
- ist nur ein Schaltkreis definiert, wird `true` zurückgegeben

1.3.2 Vorgehen

1. es wird über alle Schaltkreise iteriert
   a) die KNFs aller Gatter werden ermittelt
   b) falls der Schaltkreis nicht der erste ist, werden seine Eingänge mit denen des ersten Schaltkreises verbunden d.h. die Identität wird duch eine KNF dargestellt
   c) falls der Schaltkreis nicht der erste ist, werden seine Ausgänge XOR mit denen des ersten Schaltkreises verbunden d.h. als KNF dargestellt
2. alle Ausgänge der XOR-Verbindungen werden OR verküpft, d.h. es wird eine Klausel gebildet, die alle diese Ausgänge enthält

   lässt sich eine dieser Variablen auf `true` abbilden (die These ist SATISFIABLE), dann sind die definierten Schaltkreise nicht äquivalent
3. gesamte KNF wird in Datei geschrieben und der SATsolver gestartet

1.3.3 Implementierung

Der verwendete SAT-Solver ist miniSAT in der Version 1.14 als Binary. Die generierten KNF-Klauseln werden in eine Datei mit dem Namen cnf geschrieben und anschliesend vom SAT-Solver gelesen. Die jeweiligen Gatter-Klassen sind so implementiert, dass sie ihre Funktion als KNF darstellen können. Die Klasse `Solver` enthält einen Zähler für die Benennung der KNF-Variablen.

2 Versuche

2.1 1. Versuch

Benchmarks	tests/01-test/s344.bench	tests/01-test/s349.bench
Parser	BENCH	BENCH
Inputs	9	9
DFFs	15	15
Outputs	11	11
Gatter	160	161
Zeit in Sekunden	0.105102	0.103989
Äquivalenzcheck durch SAT-Solving	äquivalent	
Zeit in Sekunden	0.196012	
Äquivalenzcheck durch Simmulierung	nicht ausgeführt	
Zeit in Sekunden	ewig	

Zu lange Ausführungszeiten für den Simulator.

2.2 2. Versuch

Benchmarks	tests/02-test/s298.bench	tests/02-test/s298.bench
Parser	BENCH	BENCH
Inputs	3	3
DFFs	14	14
Outputs	6	6
Gatter	119	119
Zeit in Sekunden	0.079251	0.05012
Äquivalenzcheck durch SAT-Solving	äquivalent	
Zeit in Sekunden	0.096006	
Äquivalenzcheck durch Simmulierung	äquivalent	
Zeit in Sekunden	2034.31	

2.3 3. Versuch

Benchmarks	tests/03-test/s298.bench	tests/03-test/s298a.bench
Parser	BENCH	BENCH
Inputs	3	3
DFFs	14	14
Outputs	6	6
Gatter	119	119
Zeit in Sekunden	0.048399	0.047988
Äquivalenzcheck durch SAT-Solving	nicht äquivalent	
Zeit in Sekunden	0.100006	
Äquivalenzcheck durch Simmulierung	nicht äquivalent	
Zeit in Sekunden	0.016245	

2.4 4. Versuch

Benchmarks	tests/04-test/c17.bench	tests/04-test/c17a.bench
Parser	BENCH	BENCH
Inputs	5	5
DFFs	0	0
Outputs	2	2
Gatter	6	6
Zeit in Sekunden	0.002447	0.00174
Äquivalenzcheck durch SAT-Solving	nicht äquivalent	
Zeit in Sekunden	0.004	
Äquivalenzcheck durch Simmulierung	nicht äquivalent	
Zeit in Sekunden	0.000122	

2.5 5. Versuch

Benchmarks	tests/05-test/c17.bench	tests/05-test/c17a.bench
Parser	BENCH	BENCH
Inputs	5	5
DFFs	0	0
Outputs	2	2
Gatter	6	7
Zeit in Sekunden	0.002533	0.001905
Äquivalenzcheck durch SAT-Solving	äquivalent	
Zeit in Sekunden	0	
Äquivalenzcheck durch Simmulierung	äquivalent	
Zeit in Sekunden	0.001855	

Veränderung: ein NAND-Gatter durch ein AND und ein NOT ersetzt. Ergebnis korrekt.

2.6 6. Versuch

Benchmarks	tests/06-test/c6288.bench	tests/05-test/c6288a.bench
Parser	BENCH	BENCH
Inputs	32	32
DFFs	0	0
Outputs	32	32
Gatter	2416	2416
Zeit in Sekunden	8.64887	8.4408
Äquivalenzcheck durch SAT-Solving	nicht äquivalent	
Zeit in Sekunden	17.3411	
Äquivalenzcheck durch Simmulierung	nicht äquivalent	
Zeit in Sekunden	6.16808	

Veränderung: 2 Gatter geändert.

3 Anhang

3.1 Circuit

Listing 3.1: Circuit.h

```
1  #ifndef circuit_h
2  #define circuit_h
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <cmath>
8  #include <map>
9  #include <vector>
10 #include <string>
11 #include <set>
12
13 class Circuit;
14
15
16 #include "../gates/Gates.h"
17 typedef boost::shared_ptr<Circuit> CircuitPtr;
18
19
20
21 class Circuit{
22
23   private:
24 //  unsigned int gateCount;
25     std::string name;
26     std::string description;
27     gateMap gates;
28     inputMap inputs;
29     outputMap outputs;
30
31     unsigned int state;
32     unsigned int depth;
33
34   public:
35
36     Circuit();
37     void simulateAllInputs();
38     void simulateCircuit();
39
40     void printInputs();
41     void printOutputs();
42
43     void resetGates();
44     void resetOutputs();
45
46     void addInput(std::string name, InputPtr input);
47     void addOutput(std::string name, OutputPtr output);
48     void addGate(std::string name, GatePtr gate);
49
50     bool isInput(std::string name);
51     bool isOutput(std::string name);
52     bool isGate(std::string name);
53
54     GatePtr getGate(std::string name) { return this->gates[name]; };
55     InputPtr getInput(std::string name) { return this->inputs[name]; };
```

```
56      OutputPtr getOutput(std::string name) { return this->outputs[name]; };
57
58      unsigned int getGatesCount() { return this->gates.size(); };
59      unsigned int getInputsCount() { return this->inputs.size(); };
60      unsigned int getFFInputsCount();
61      unsigned int getOutputsCount() { return this->outputs.size(); };
62
63      unsigned int getState() { return this->state; };
64      bool setNextState();
65      void resetState();
66
67      std::set<std::string> getInputKeys();
68      std::set<std::string> getOutputKeys();
69      std::set<std::string> getGateKeys();
70
71      void setName(std::string name) { this->name = name; }
72      void calculateDepth();
73
74
75
76  };
77  #endif
```

3.2 Simulator

Listing 3.2: Simulator.h

```
1   #ifndef simulator_h
2   #define simulator_h
3
4
5   #include "../parser/Parsers.h"
6   #include "Circuit.h"
7
8   typedef std::map<std::string, CircuitPtr> circuitMap;
9
10  class Simulator{
11
12    protected:
13      circuitMap circuits;
14      unsigned int globalGateCount;
15
16
17    public:
18
19      Simulator() { this->globalGateCount = 1; };
20      void parseCircuit(std::string parser, std::string filePath);
21      void simulateCircuits();
22      virtual bool checkEquivalenceOfCircuits();
23      unsigned int getCircuitCount() { return this->circuits.size(); };
24
25      unsigned int getGlobalGateCount() { return this->globalGateCount; };
26      circuitMap getCircuits() { return this->circuits; };
27      void importCircuits(circuitMap circuits, unsigned int globalGateCount) { this->circuits
            = circuits; this->globalGateCount = globalGateCount; };
28  };
29  #endif
```

3.3 Solver

Listing 3.3: Solver.h

```
1   #ifndef solver_h
2   #define solver_h
3
4   #include "Simulator.h"
```

```
5  #include "../parser/Parsers.h"
6  #include "Circuit.h"
7
8
9  class Solver : public Simulator{
10
11
12   public:
13
14     Solver() : Simulator() { };
15     bool checkEquivalenceOfCircuits();
16     void writeCNF(std::string cnf);
17     std::string getCNF();
18 };
19 #endif
```

3.4 Parsers

Listing 3.4: Parsers.h

```
1  #ifndef parsers_h
2  #define parsers_h
3
4  class Parser;
5  #include "Parser.h"
6
7
8  #include "BENCH.h"
9  #include "VERILOG.h"
10
11
12 #endif
```

3.5 Parser

Listing 3.5: Parser.h

```
1  #ifndef parser_h
2  #define parser_h
3
4  #include <map>
5  #include <string>
6  #include <boost/regex.hpp>
7  #include "../main/Circuit.h"
8  #include "../gates/Gates.h"
9  class Parser;
10
11 template<typename T> Parser * createParserInstance() { return new T; }
12 typedef std::map<std::string, Parser*(*)()> parserType;
13
14 class Parser
15 {
16
17   protected:
18     std::string parseInput;
19     Circuit *circuit;
20     unsigned int gateCounter;
21
22   public:
23     Parser() { this->gateCounter = 0; };
24
25     Parser* getParser(std::string parser);
26     void parseCircuit(std::string filePath, unsigned int globalGateCount);
27
28     void readFile(const char * path);
29
```

```
30      std::string parseComments() { return ""; };
31      void setParseInput(std::string parseInput);
32      GatePtr getGateFromString(std::string gateName, std::string name);
33
34      virtual void parseInputs();
35      virtual void parseFFs();
36      virtual void parseOutputs();
37      virtual void parseGates() {};
38
39      virtual boost::regex getInputRegex() { return (boost::regex (".")); };
40      virtual boost::regex getFFRegex() { return (boost::regex (".")); };
41      virtual boost::regex getOutputRegex() { return (boost::regex (".")); };
42      virtual boost::regex getGateRegex() { return (boost::regex (".")); };
43      virtual boost::regex getCommentRegex() { return (boost::regex (".")); };
44
45      CircuitPtr getCircuit() { return CircuitPtr(this->circuit); };
46
47      unsigned int getGatesCount() { return this->gateCounter;};
48
49  };
50  #endif
```

3.6 BENCH

Listing 3.6: BENCH.h

```
1   #ifndef bench_h
2   #define bench_h
3
4   #include "Parser.h"
5
6   class BENCH : public Parser
7   {
8     public:
9       BENCH() { this->gateCounter = 0; }
10      boost::regex getInputRegex();
11      boost::regex getFFRegex();
12      boost::regex getOutputRegex();
13      boost::regex getGateRegex();
14      boost::regex getCommentRegex();
15
16      void parseGates();
17
18  };
19  #endif
```

3.7 Gates

Listing 3.7: Gates.h

```
1   #ifndef gates_h
2   #define gates_h
3   #include <boost/shared_ptr.hpp>
4   #include <boost/make_shared.hpp>
5
6   #include "Gate.h"
7   typedef boost::shared_ptr<Gate> GatePtr;
8   typedef std::map<std::string, GatePtr> gateMap;
9
10  #include "Input.h"
11  typedef boost::shared_ptr<Input> InputPtr;
12  typedef std::map<std::string, InputPtr> inputMap;
13
14
15  #include "Output.h"
16  typedef boost::shared_ptr<Output> OutputPtr;
```

```
17 typedef std::map<std::string, OutputPtr> outputMap;
18
19 #include "AND.h"
20 #include "OR.h"
21 #include "NAND.h"
22 #include "NOR.h"
23 #include "NOT.h"
24 #include "BUFF.h"
25 #include "XOR.h"
26 #include "XNOR.h"
27 #include "DFF.h"
28
29
30 template<typename T> Gate * createGateInstance() { return new T; }
31 typedef std::map<std::string, Gate*(*)()> gateType;
32
33 #endif
```

3.8 Gate

Listing 3.8: Gate.h

```
1  #ifndef gate_h
2  #define gate_h
3  class Gate;
4
5  #include <deque>
6  #include <string>
7  #include <sstream>
8
9
10 class Gate
11 {
12
13   protected:
14     std::deque<bool*> inputs;
15     std::vector<unsigned int> inputKeys;
16     unsigned int outputKey;
17
18     bool temp_output;
19
20     std::string name;
21
22   public:
23     bool output;
24
25     Gate() : inputs(2), inputKeys(2) { };
26     Gate(unsigned int in) : inputs(in), inputKeys(in) {};
27     Gate(std::string name) : inputs(2), inputKeys(2) { this->name = name; };
28
29     void setInput(unsigned int input, bool* outputsRef, unsigned int inputKey)
30     {
31       if(this->inputs.size()-1<input){
32         this->inputs.resize(input + 1);
33         this->inputKeys.resize(input + 1);
34       }
35       this->inputs[input] = outputsRef;
36       this->inputKeys[input] = inputKey;
37     };
38
39     bool getInput(unsigned int input) { return *this->inputs[input]; };
40
41     void calculateOutput() { this->temp_output = this->gateOutput(); };
42     void setOutput() { this->output = this->temp_output; };
43     bool getOutput() { return this->output; };
44     void resetOutput() { this->output = false; this->temp_output = false;};
45
```

```
46      bool* getOutputRef() { return &this->output; };
47
48      virtual bool gateOutput() { return false; };
49
50      void setOutputKey(unsigned int outputKey) { this->outputKey = outputKey; }
51      unsigned int getOutputKey() { return this->outputKey; }
52
53      virtual std::string getCNF() { return ""; }
54
55      virtual unsigned int getNumberOfCNFClauses() { return 0; }
56
57      virtual std::string getGateType() { return ""; };
58  };
59  #endif
```

3.9 Input

Listing 3.9: Input.h

```
1   #ifndef input_h
2   #define input_h
3
4
5   #include "Gate.h"
6
7
8   class Input
9   {
10
11    protected:
12      std::string name;
13      bool *value;
14      bool innerValue;
15      unsigned int outputKey;
16
17    public:
18      Input() { this->innerValue = false; this->value = &this->innerValue; }
19      Input(std::string name) {this->name = name; this->innerValue = false; this->value =
            &this->innerValue; }
20      void setInput(bool* outputsRef) { this->value = outputsRef; };
21      void setInput(bool input) {this->innerValue=input; this->value = &this->innerValue; };
22      bool* getOutputRef() { return this->value; };
23      bool getOutput() { return *this->value; };
24
25      void setOutputKey(unsigned int outputKey) { this->outputKey = outputKey; }
26      unsigned int getOutputKey() { return this->outputKey; }
27
28      virtual std::string getGateType() { return "Input"; };
29  };
30  #endif
```

3.10 DFF

Listing 3.10: DFF.h

```
1   #ifndef dff_h
2   #define dff_h
3
4
5   #include "Gate.h"
6
7
8   class DFF : public Input
9   {
10    public:
11      DFF() : Input() {};
```

```
12      DFF(std::string name) : Input(name) {};
13
14      std::string getGateType() { return "DFF"; };
15  };
16  #endif
```

3.11 Output

Listing 3.11: Output.h

```
1   #ifndef output_h
2   #define output_h
3
4
5
6
7   class Output
8   {
9
10    private:
11      std::string name;
12      bool *value;
13      unsigned int outputKey;
14
15    public:
16      Output() {}
17      Output(std::string name) {this->name = name; }
18      void setInput(bool* outputsRef, unsigned int outputKey) { this->value = outputsRef;
            this->outputKey = outputKey; };
19      bool* getOutputRef() { return this->value; };
20      bool getOutput() { return *this->value; };
21
22      void setOutputKey(unsigned int outputKey) { this->outputKey = outputKey; }
23      unsigned int getOutputKey() { return this->outputKey; }
24  };
25  #endif
```