veriSIMPLER: An Automated Formal Verification Methodology for SIMPLER MAGIC Design Style Based In-Memory Computing

Chandan Kumar Jha

chajha@uni-bremen.de

University of Bremen,

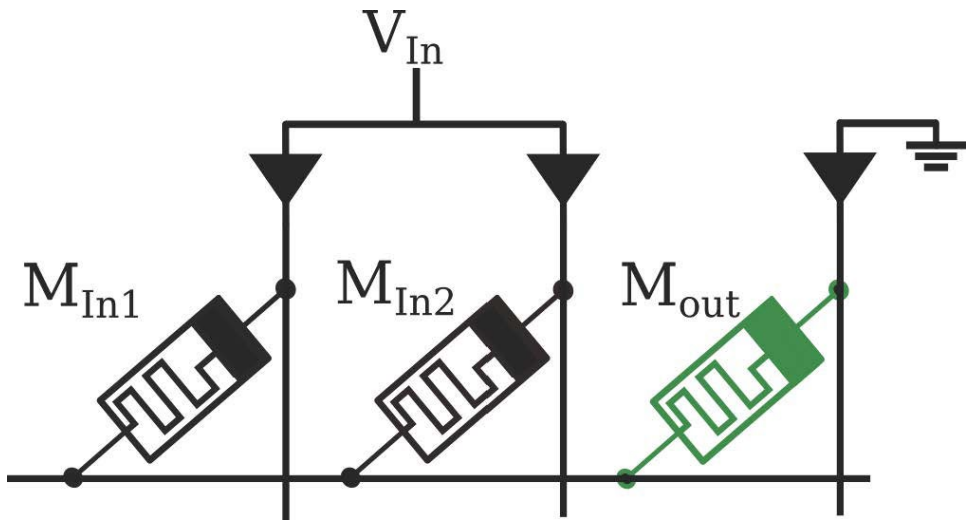
Group of Computer Architecture

IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, VOL. 71, NO. 9, SEPTEMBER 2024

veriSIMPLER: An Automated Formal Verification Methodology for SIMPLER MAGIC Design Style Based In-Memory Computing

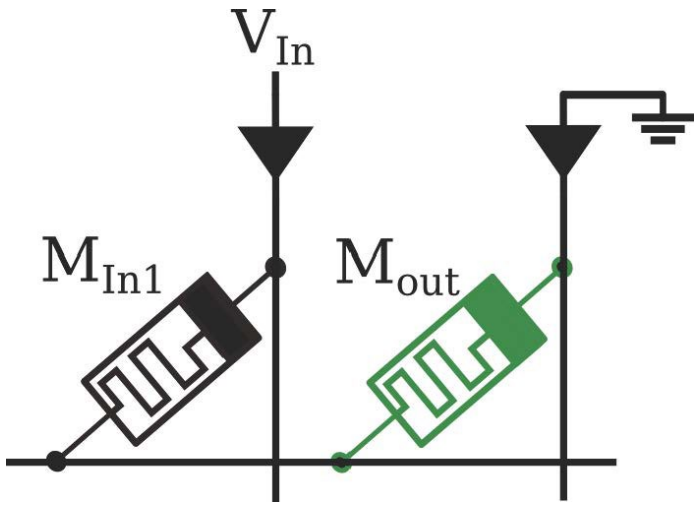
Chandan Kumar Jha, Khushboo Qayyum, Kemal Çağlar Coşkun, Simranjeet Singh, *Graduate Student Member, IEEE*, Muhammad Hassan, Rainer Leupers, Farhad Merchant, and Rolf Drechsler, *Fellow, IEEE*

Background: IMC MAGIC NOR



- The output (M_{out}) memristor is initialized to logic ‘1’ (low resistance state) before the operation
- The input memristors (M_{In1} and M_{In2}) are loaded with the input values
- Execution Voltage (V_{In}) is applied to perform the operation
- The final result is stored in the (M_{out}) memristor

Background: IMC MAGIC NOT

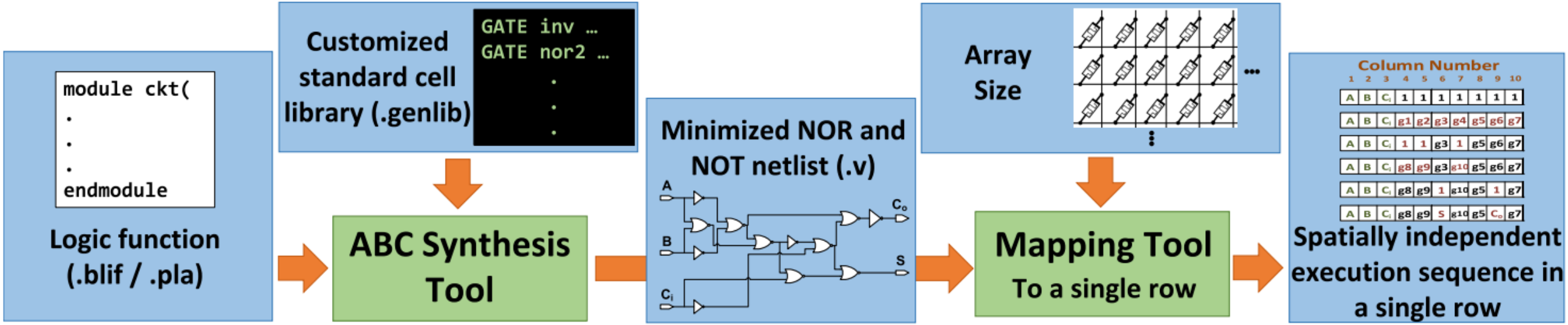


- The output (M_{out}) memristor is initialized to logic ‘1’ (low resistance state) before the operation
- The input memristor (M_{In1}) is loaded with the input value
- Execution Voltage (V_{In}) is applied to perform the operation
- The final result is stored in the (M_{out}) memristor

Formal Verification for IMC

- Automated techniques to generate the mapping using various design styles
- How do we guarantee that the generated mapping is correct?
- Formal Verification can help in guaranteeing the correctness
- We focus on the MAGIC design style

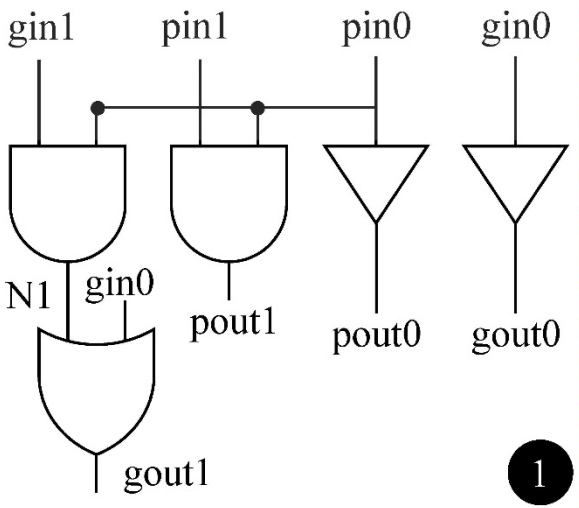
SIMPLER MAGIC Tool



Ben-Hur, Rotem, Ronny Ronen, Ameer Haj-Ali, Debjyoti Bhattacharjee, Adi Eliahu, Natan Peled, and Shahar Kvatinsky. "SIMPLER MAGIC: Synthesis and mapping of in-memory logic executed in a single row to improve throughput." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, no. 10 (2019): 2434-2447.

Motivating Example

Design



1

Verilog

```
module pp (gout1,pout1,gout0,pout0,
           gin1,pin1,gin0,pin0);
input gin1,pin1,gin0,pin0;
output gout1,pout1,gout0,pout0;
wire N1;
buf BUFF1 (gout0,gin0);
buf BUFF1 (pout0,pin0);
and AND1 (pout1,pin0,pin1);
and AND2 (N1, pin0, gin1);
or OR1 (gout1,N1,gin0);
endmodule
```

2

Mapping Using SIMPLER

Inputs: gin1(M0), pin1(M1), gin0(M2), pin0(M3)
Outputs: gout1(M7), pout1(M5), gout0(M0), pout0(M0)
Execution Sequence:
INIT M4, M5, M6, M7, M8, M9, M10, M11;
T1: M11 = not (M0)
T2: M10 = not (M3)
T3: M9 = nor (M10, M11)
T4: M8 = nor (M9, M2)
T5: M7 = not (M8)
T6: M6 = not (M1)
T7: M5 = nor (M10, M6)

3

Expected Output

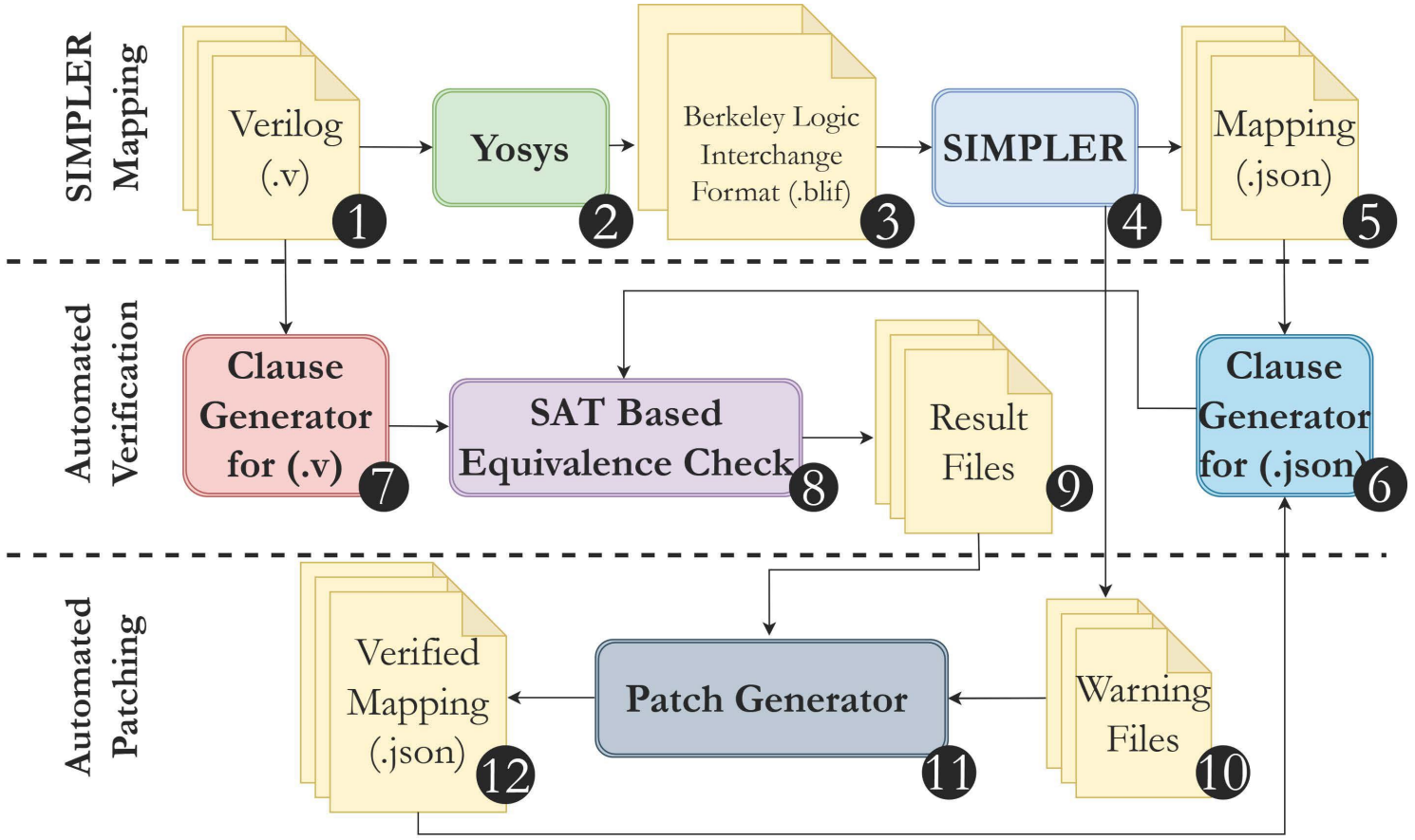
gout1 = M7
pout1 = M5
gout0 = M2
pout0 = M3

SIMPLER Output

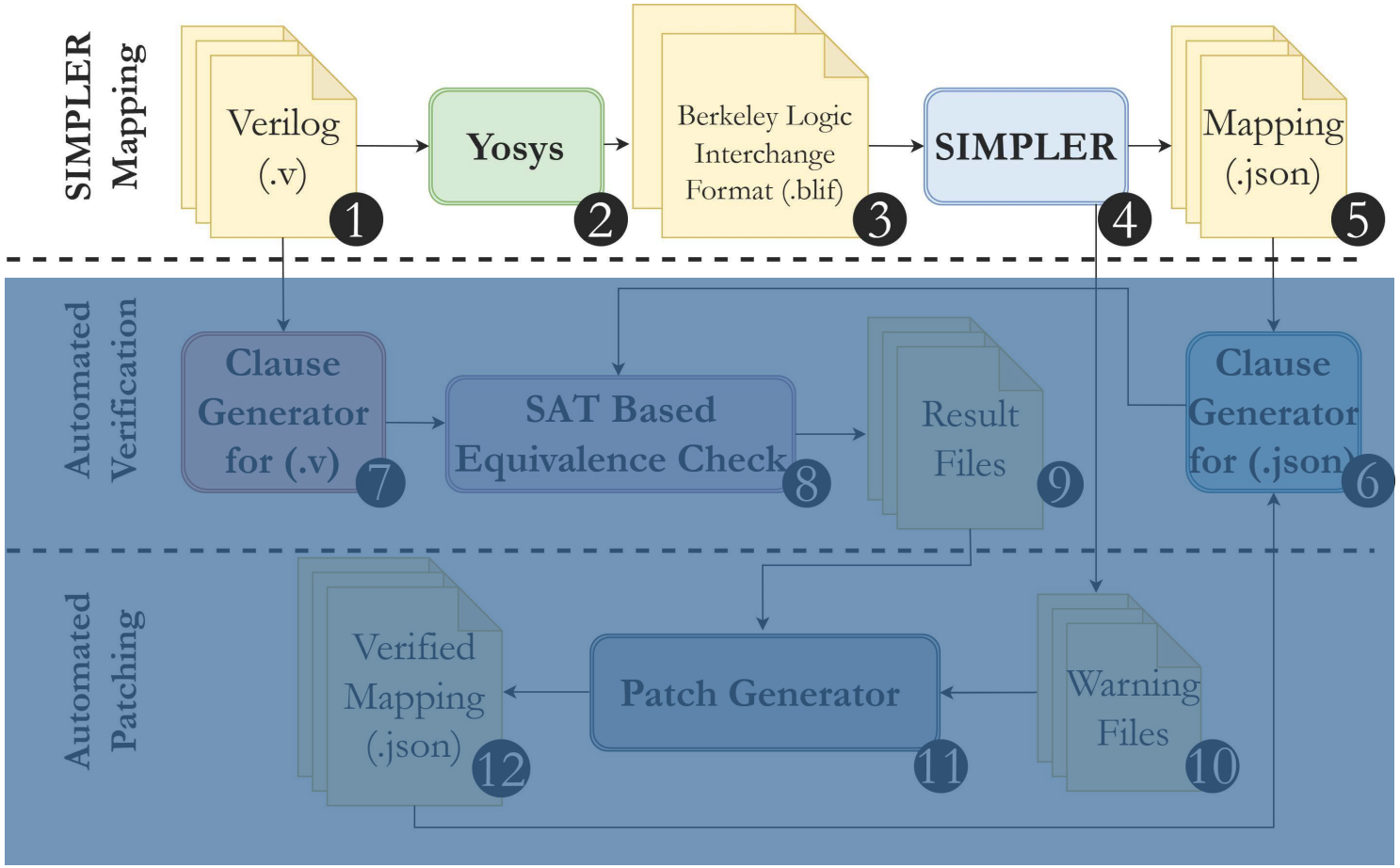
gout1 = M7 ✓
pout1 = M5 ✓
gout0 = M0 ⊗
pout0 = M0 ⊗

4

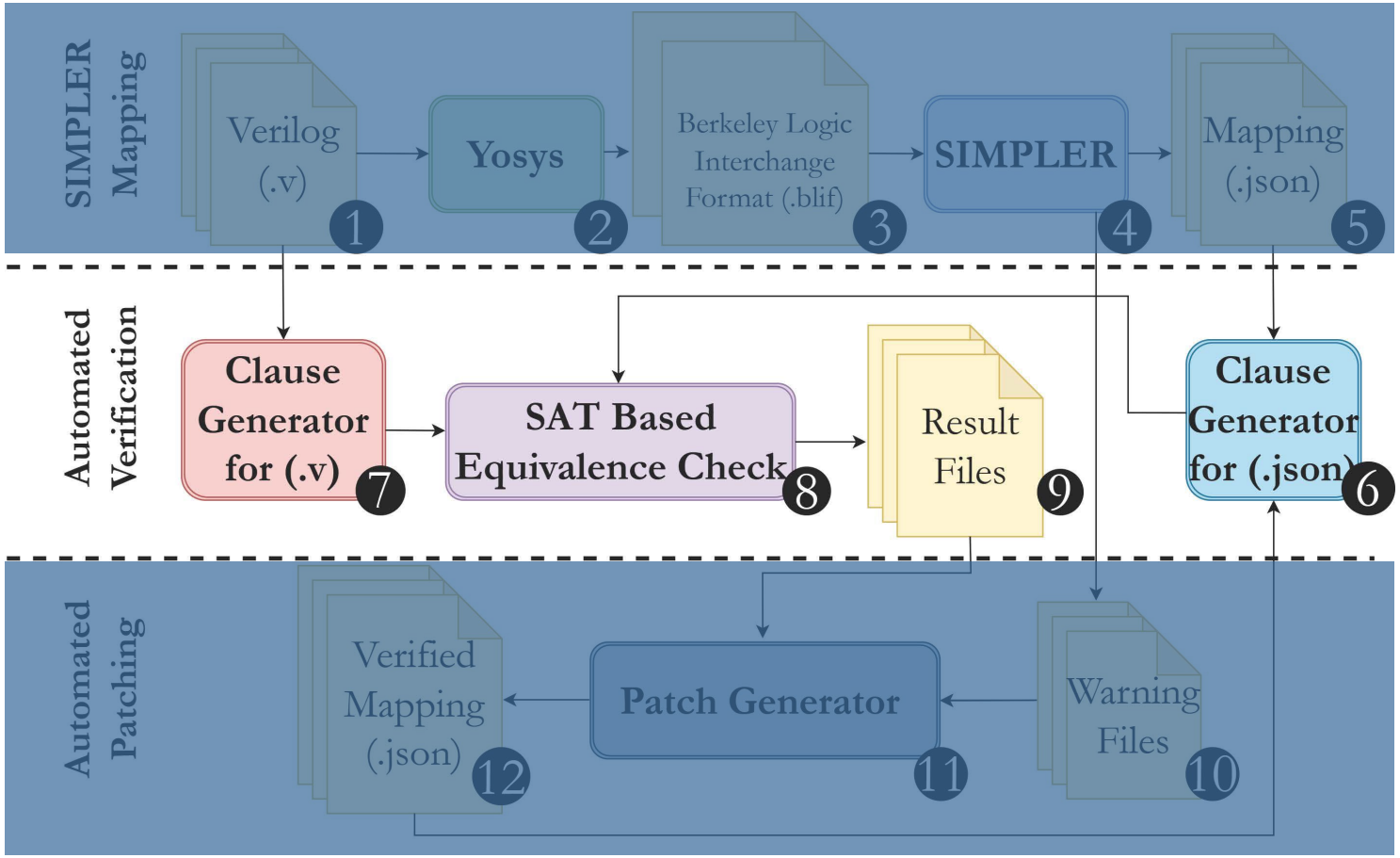
Proposed Verification Framework: veriSIMPLER



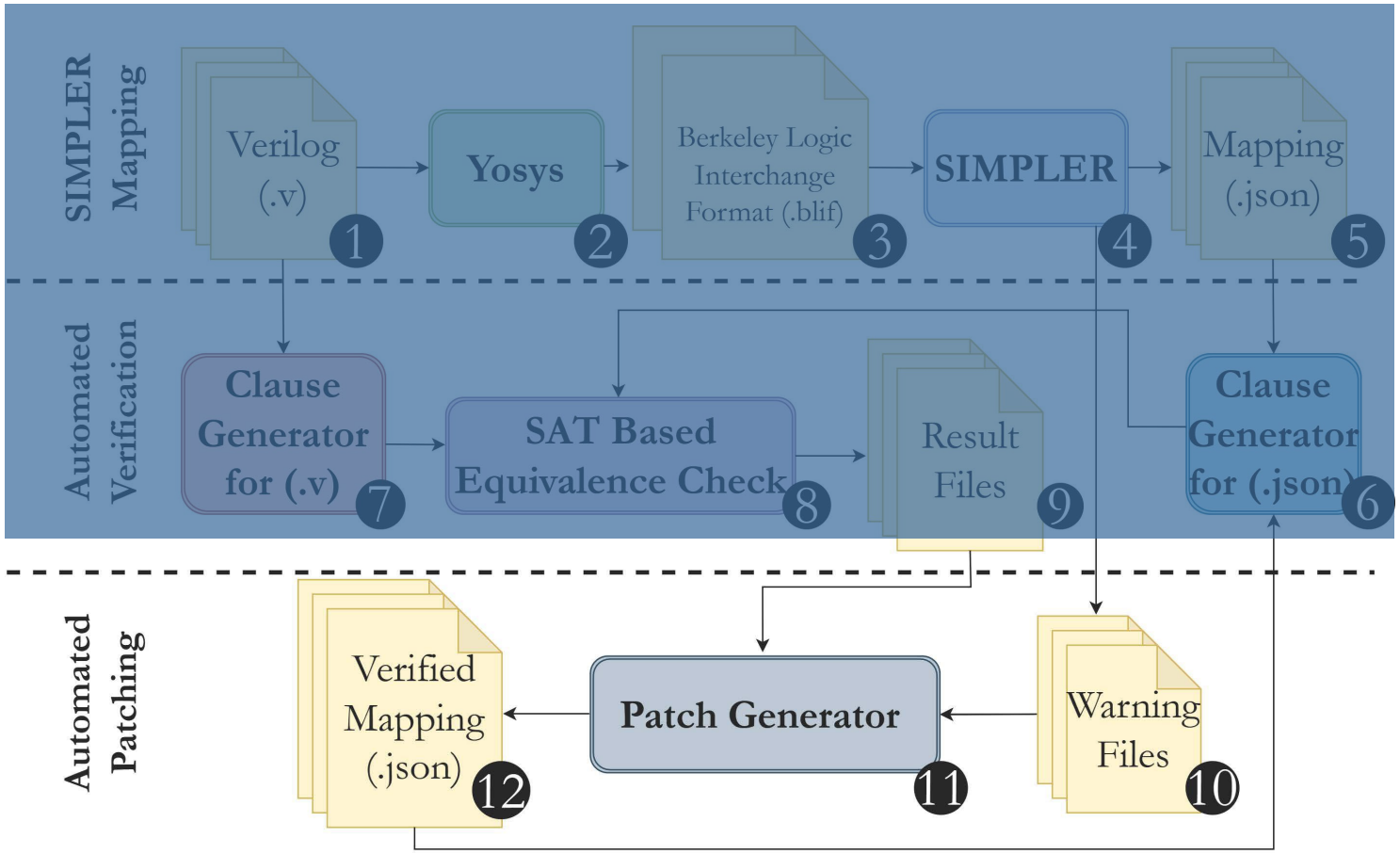
Proposed Verification Framework: veriSIMPLER



Proposed Verification Framework: veriSIMPLER



Proposed Verification Framework: veriSIMPLER



veriSIMPLER: Clause Generation

```
def NAND3(a, b, c):
    return ~(a & b & c)
def NOR3(a, b, c):
    return ~(a | b | c)
```

```
def NOT(a):
    return ~(a)
def parsed_json(...):
    ...
    M11 = 1
    M11 = NOT (M0) & M11 # AND with M11 checks that M11 has
      been initialized before operation
    ...
```

veriSIMPLER: Python File for Verification

```python
# CLAUSE generation from Verilog
def from_verilog (gin1,pin1,gin0,pin0):
    gout0= BUFF1(gin0)
    pout0= BUFF1(pin0)
    pout1= AND2(pin0,pin1)
    N1= AND2(pin0,gin1)
    gout1= OR2(N1,gin0)
    return(gout1,pout1,gout0,pout0)
```

```python
# CLAUSE generation from Mapping
def from_json(in0,in1,in2,in3):
    M0 = in0 # initialization clause
    M1 = in1 # initialization clause
    M2 = in2 # initialization clause
    M3 = in3 # initialization clause
    M4 = 1 # initialization clause
    M5 = 1 # initialization clause
    M6 = 1 # initialization clause
    M7 = 1 # initialization clause
    M8 = 1 # initialization clause
    M9 = 1 # initialization clause
    M10 = 1 # initialization clause
    M11 = 1 # initialization clause
    M11 = NOT(M0) & M11 #execution clause
    M10 = NOT(M3) & M10 #execution clause
    M9 = NOR(M10,M11) & M9 #execution clause
    M8 = NOR(M9,M2) & M8 #execution clause
    M7 = NOT(M8) & M7 #execution clause
    M6 = NOT(M1) & M6 #execution clause
    M5 = NOR(M10,M6) & M5 #execution clause
    return (M7,M5,M2,M3) #PATCHED M2 <- M0; M3 <-M0
```

veriSIMPLER: Python File for Verification

```
from z3 import *
gin1,pin1,gin0,pin0 = BitVecs ('gin1 pin1 gin0 pin0', 1)
s= Solver()
json_res = from_json(gin1,pin1,gin0,pin0) #gets clauses for
      all outputs from JSON
verilog_res = from_verilog(gin1,pin1,gin0,pin0) #gets
     clauses for all outputs from Verilog
for i in range (0,len(json_res)):
    #Equivalence checking code
    Theorem = (json_res[i] == verilog_res[i])
    s.add(Theorem)
    prove (Theorem)
```

veriSIMPLER: Clauses Internal Representation

Output Mapping/Verilog	Clauses from Verilog (Reference)	Clauses from Mapping (SIMPLER)
gout1/M7	pin0 & gin1 | gin0	~(~(~(~pin0 & 1 | ~gin1 & 1) & 1 | gin0) & 1) & 1
pout1/M5	pin0 & pin1	~(~pin0 & 1 | ~pin1 & 1) & 1
gout0/M2	gin0	gin0
pout0/M3	pin0	pin0

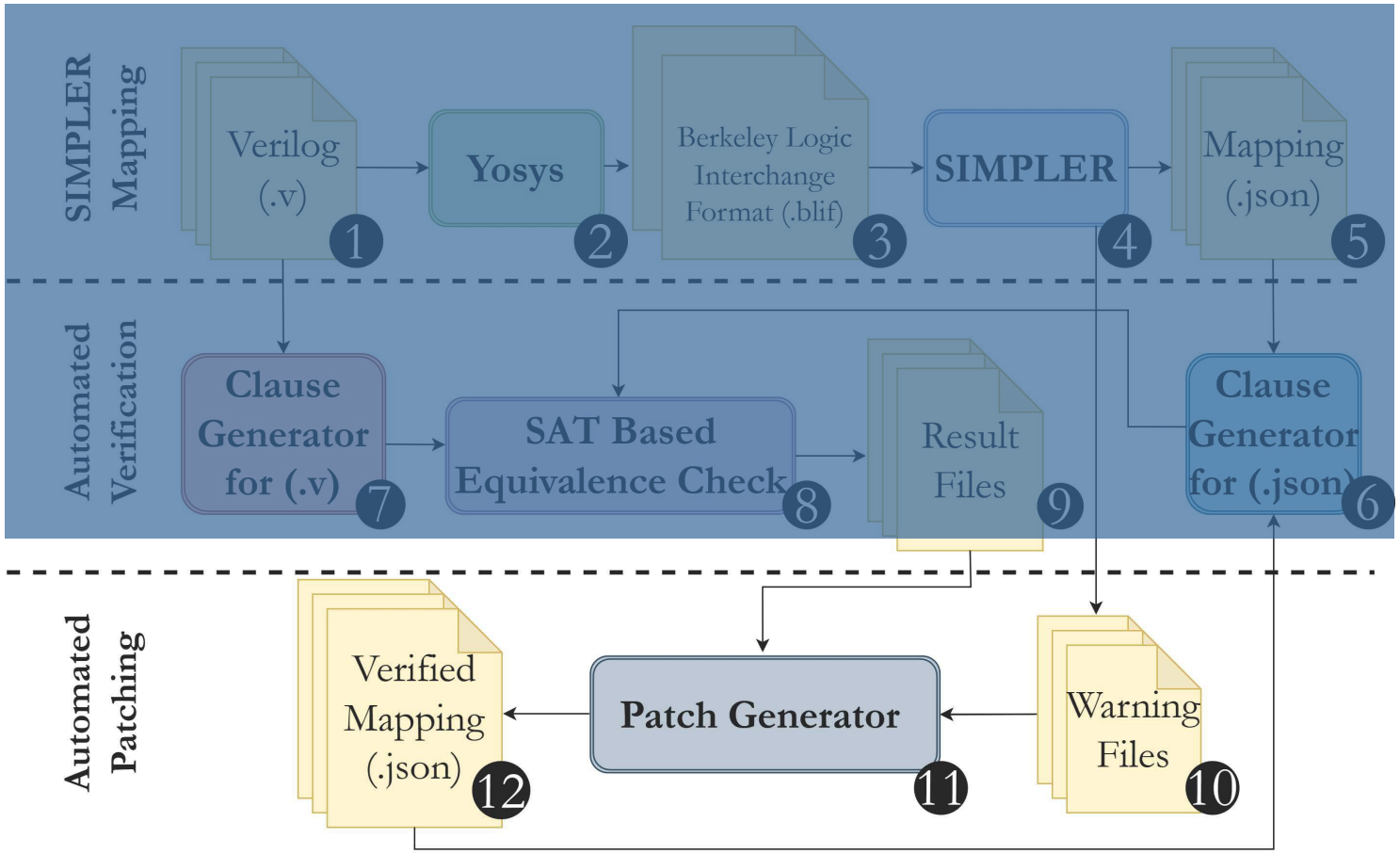
veriSIMPLER: Results

Circuit	Verilog Clauses	Mapping Clauses (NOR+ NOT)	Runtime	Number of Bugs	Bugs after Patching
c17	6	13	0.25	0	0
c1908	880	569	1.141	0	0
c5315	2307	1970	3.063	13	0

veriSIMPLER: Warning Files

```
** Warning ** unsupported operation: '  buf  g7(.a(gin0), .
     O(gout0));'
** Warning ** unsupported operation: '  buf  g8(.a(pin0), .
     O(pout0));'
```

Proposed Verification Framework: veriSIMPLER



Other Works on Formal Verification of IMC

- Bhunia, K., Deb, A., Datta, K., Hassan, M., Shirinzadeh, S., & Drechsler, R. (2024). *ReSG: A Data Structure for Verification of Majority-based In-memory Computing on ReRAM Crossbars.* ACM Transactions on Embedded Computing Systems, 23(6), 1-24.
- Qayyum, K., Kole, A., Datta, K., Hassan, M., & Drechsler, R. (2024, June). *Exploring the Potential of Decision Diagrams for Efficient In-Memory Design Verification.* In Proceedings of the Great Lakes Symposium on VLSI 2024 (pp. 502-506).

More Information

https://agra.informatik.uni-bremen.de/

veriSIMPLER: An Automated Formal Verification Methodology for SIMPLER MAGIC Design Style Based In-Memory Computing

Chandan Kumar Jha

chajha@uni-bremen.de

University of Bremen,

Group of Computer Architecture