Automatic Memory Reductions for RTL Model Verification

Panagiotis Manolios
Sudarshah K. Srinivasan
Daron Vroon
Motivation

- Pipelined machine verification
  - State of the art: term level models
  - *The* major limitation
  - We really want to verify RTL-level models
  - RTL models too hard for state of the art
  - We developed BAT, Bit-level Analysis Tool
2 Stage Pipelined Machine

Verification time (sec)

State-of-the-Art

Number of words in memory
2 Stage Pipelined Machine

Verification time (sec) vs. Number of words in memory.

- State-of-the-Art
- BAT
Contributions

- Automatic and efficient memory abstraction
  - Memories are first class objects
  - Memory comparisons allowed in all contexts
  - Memories can be passed to and returned from functions

- Incorporation of term-rewriting techniques
  - Decrease size of abstract memories
  - Drastic improvements in verification time

- Implemented in BAT
  - Extensive validation of techniques
Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions
Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions
BAT Specification Language

- Strongly typed
- Type inference
- Function definitions allowed
- Powerful Lisp-based language
- Syntax extensions enabled by Lisp
- Parameterized models are easy to define
- Target language for Verilog or VHDL
- Bounded model checker & $k$-induction engine
Memory Usage Example

*Applicative Memory Operations:*

\[(\text{get } m \ a) \quad : \quad \text{Get the value in memory } m \text{ at address } a\]
\[(\text{set } m \ a \ v) \quad : \quad \text{New memory, like } m, \text{ except address } a \text{ has value } v\]

**Vars:**  
\((\text{mem } 8 \ 4)\)  
\((\text{adr } 3)\)  
\((\text{val } 4)\)

**Formula:**  
\(= (\text{get } (\text{set } \text{mem } \text{adr } \text{val}) \text{adr}) \text{val)}\)
Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions
Bit-level Analysis Tool (BAT)

- BAT Specification
- Inline functions, constant prop, simplify
- RBC + if + next memory operations
- Unroll transition relation
- Abstract memories
- Translate to CNF
- RBC + if
- Propositional Formula (CNF)
- Solve with SAT
- Valid/Invalid
Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions
Previous Work

- **Ganai et al. approach**

- **UCLID approach**

- Limited to reads and writes
- Memories are *not* first class objects
  - Cannot be passed to functions
  - Cannot be directly compared
Limitations of Previous Work

In some contexts, memories can be compared for equality using memory reads

\[
( = \ (set \ m_1 \ a_1 \ v_1) \\
( set \ m_2 \ a_1 \ v_2 ) \ )
\]
Limitations of Previous Work

In some contexts, memories can be compared for equality using memory reads

\[
(= \ (get \ (set \ m_1 \ a_1 \ v_1) \ a) \\
(\ get \ (set \ m_2 \ a_1 \ v_2) \ a))
\]
Limitations of Previous Work

In some contexts, memories can be compared for equality using memory reads

\[
\begin{align*}
&= (\text{get} \ (\text{set} \ m_1 \ a_1 \ v_1) \ a) \\
&\text{get} \ (\text{set} \ m_2 \ a_1 \ v_2) \ a))
\end{align*}
\]

Memories cannot be compared in all contexts

\[
\begin{align*}
\text{not} &\ (= \ (\text{get} \ (\text{set} \ m_1 \ a_1 \ v_1) \ a) \\
&\text{get} \ (\text{set} \ m_2 \ a_1 \ v_2) \ a))
\end{align*}
\]
BAT Memory Abstraction

Memories are treated as first class objects

\[(= (\text{set } m_1 a_1 v_1) (\text{set } m_2 a_1 v_2))\]

Memories can be directly compared in all contexts

\[(\text{not } (= (\text{set } m_1 a_1 v_1) (\text{set } m_2 a_1 v_2)))\]
BAT Memory Abstraction

\[(\text{get } (\text{set } (\text{set } m \ a_1 \ v_1) \ a_2 \ v_2) \ a_3)\]

- Determine number of unique gets and sets \((n)\)
- Generate abstract memory consisting of \(n\) words
- Apply abstraction to original addresses
- Note: size of abstract addresses is \(\lg(n)\)
BAT Memory Abstraction

\[
\text{(not } (= \text{(set (set } m_1 \ b \ 0) \ a \ 1) \\
\text{(set (set } m_2 \ b \ 1) \ c \ 0)))
\]

\[\begin{array}{c}
m'_1 \\
a' \\
b' \\
\end{array} \quad = \quad \begin{array}{c}
m'_2 \\
a' \\
b' \\
\end{array}\]

- Cannot abstract memories \( m_1 \) and \( m_2 \) in isolation.
- Memories have to be abstracted together if they are:
  - Compared for equality
  - Appear in the same context
Memory Equivalence Classes

Base memories of a memory expression:

1. $\text{base}((\text{if } e \ m_1 \ m_2)) = \text{base}(m_1) \cup \text{base}(m_2)$
2. $\text{base}((\text{set } m \ a \ v)) = \text{base}(m)$
3. $\text{base}(m) = \{m\}$, where $m$ is a variable
Memory Equivalence Classes

Base memories of a memory expression:

1. \( \text{base}((\text{if } e \ m_1 \ m_2)) = \text{base}(m_1) \cup \text{base}(m_2) \)
2. \( \text{base}((\text{set } m \ a \ v)) = \text{base}(m) \)
3. \( \text{base}(m) = \{m\}, \text{ where } m \text{ is a variable} \)

\[
\text{base}((\text{set } (\text{set } m_1 \ b \ 0) \ a \ 1))) = \{m_1\}
\]
Memory Equivalence Classes

Base memories of a memory expression:

1. \( \text{base}((\text{if } e \ m_1 \ m_2)) = \text{base}(m_1) \cup \text{base}(m_2) \)
2. \( \text{base}((\text{set } m \ a \ v)) = \text{base}(m) \)
3. \( \text{base}(m) = \{m\}, \text{ where } m \text{ is a variable} \)

\[
\text{base}((\text{set } (\text{set } m_1 \ b \ 0) \ a \ 1))) = \{m_1\}
\]

\[
\text{base}((\text{if } (= \ m_1 \ m_2) \ m_3 \ (\text{if } e_2 \ m_4 \ m_5))) = \{m_3, m_4, m_5\}
\]
Memory Equivalence Classes

Base memories of a memory expression:
1. \( \text{base}((\text{if } e \ m_1 \ m_2)) = \text{base}(m_1) \cup \text{base}(m_2) \)
2. \( \text{base}((\text{set } m \ a \ v)) = \text{base}(m) \)
3. \( \text{base}(m) = \{m\} \), where \( m \) is a variable

Equality test relation \( R_f = \{m_1, m_2\} \) such that
1. \( e \ \text{is} \ \text{in} \ f \ \text{and} \ m_1, \ m_2 \ \text{are in} \ \text{base}(e) \), or
2. \( = e_1 \ e_2 \ \text{is in} \ f, \ m_1 \ \text{is in} \ \text{base}(e_1), \ \text{and} \ m_2 \ \text{is in} \ \text{base}(e_2) \)
Memory Equivalence Classes

Base memories of a memory expression:
1. \( \text{base}((\text{if } e \ m_1 \ m_2)) = \text{base}(m_1) \cup \text{base}(m_2) \)
2. \( \text{base}((\text{set } m \ a \ v)) = \text{base}(m) \)
3. \( \text{base}(m) = \{m\}, \) where \( m \) is a variable

Equality test relation \( R_f = \{m_1, m_2\} \) such that
1. \( e \) is in \( f \) and \( m_1, m_2 \) are in \( \text{base}(e) \), or
2. \( (= e_1 e_2) \) is in \( f \), \( m_1 \) is in \( \text{base}(e_1) \), and \( m_2 \) is in \( \text{base}(e_2) \)

\[
(\text{and } (= m_1 \ m_2) (= m_2 \ m_3))
\]
Memory Equivalence Classes

Base memories of a memory expression:
1. \( \text{base}((\text{if } e \ m_1 \ m_2)) = \text{base}(m_1) \cup \text{base}(m_2) \)
2. \( \text{base}((\text{set } m \ a \ v)) = \text{base}(m) \)
3. \( \text{base}(m) = \{m\}, \text{ where } m \text{ is a variable} \)

Equality test relation \( R_f = \{m_1, m_2\} \) such that
1. \( e \text{ is in } f \text{ and } m_1, m_2 \text{ are in } \text{base}(e), \) or
2. \( (= e_1 e_2) \text{ is in } f, m_1 \text{ is in } \text{base}(e_1), \) and \( m_2 \text{ is in } \text{base}(e_2) \)

\( E_f \) is the transitive closure of \( R_f \)
\( E_f \) is an equivalence relation
Memory variables partitioned into \( \equiv \)-classes induced by \( E_f \)
Memory Equivalence Classes

Abstract memory \( m \) with memory \( m' \) that has \( n \) words:

- \( n \): total number of accesses to all memory variables in the equivalence class \((C)\) of \( m \).

\[
\begin{align*}
1. & (= m_1 \ (\text{if } e \ m_2 \ m_3)) \\
2. & (\text{and } (= m_1 \ m_2)(= m_2 \ m_3))
\end{align*}
\]

Example:

- \( m_1 \): 10 accesses;
- \( m_2 \): 10 accesses;
- \( m_3 \): 5 accesses

\( m_1, m_2, m_3 \) are abstracted using memories with 25 words.
BAT Memory Abstraction

\[ (= \ (\text{set} \ (\text{set} \ m_1 \ a_1 \ v_1) \ a_2 \ v_2) \ \\
(\text{set} \ (\text{set} \ m_2 \ a_1 \ v_1) \ a_2 \ v_2)) \) \]
BAT Memory Abstraction

\[(= (\text{set} (\text{set} m_1 a_1 v_1) a_2 v_2) \ (\text{set} (\text{set} m_2 a_1 v_1) a_2 v_2))\]
BAT Memory Abstraction

\[(= (set (set m_1 a_1 v_1) a_2 v_2) (set (set m_2 a_1 v_1) a_2 v_2))\]

Problem: Original formula not equisatisfiable with memory abstracted formula
Abstract memories are comprised of two components

- $m'$: memory with $n$ words
- $b$: bit-vector with $\lg(C_n)$ bits

Bit-vector $b$ is used to represent the unconstrained words of $m$

Algorithm for abstracting $f$ with $f'$ appears in paper

Theorem: $f$ is satisfiable iff $f'$ is satisfiable

Size of formula generated depends on sets and gets
Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions
Memory Rewriting: Example

\[(\text{set } m \ a_0 \\
\quad (\text{get} \ (\text{set} \ (\text{set} \ m \ a_1 \ v_1) \ a_2 \ v_2) \ a_0))\]

\[= \ [\text{RW1}] \\
\quad (\text{set} \ m \ a_0 \\
\quad \ (\text{get} \ (\text{set} \ m \ a_1 \ v_1) \ a_0))\]

\[= \ [\text{RW1}] \\
\quad (\text{set} \ m \ a_0 \\
\quad \ (\text{get} \ m \ a_0))\]

\[= \ [\text{RW2}] \\
\quad m\]

\[a_0 \neq a_1; \ a_0 \neq a_2; \ a_1 \neq a_2\]
Memory Rewriting

Rewrite Rule 1

\[(\text{get} \ (\text{set} \ m \ a_1 \ v) \ a_2) = \]
Memory Rewriting

Rewrite Rule 1

\[(get (set m a_1 v) a_2) = a_1 = a_2 : v\]
Memory Rewriting

Rewrite Rule 1

\[(\text{get } (\text{set } m \; a_1 \; v) \; a_2) \; = \]

\[a_1 = a_2 : v\]

\[a_1 \neq a_2 : (\text{get } m \; a_2)\]
Memory Rewriting

Rewrite Rule 1

\[(\text{get } (\text{set } m \ a_1 \ v) \ a_2) = \]
\[
a_1 = a_2 : v \]
\[
a_1 \neq a_2 : (\text{get } m \ a_2) \]
\[
(\text{if } (= a_1 a_2) \ v \ (\text{get } m a_2))
\]
Memory Rewriting

Rewrite Rule 1

\[(\text{get (set } m \ a_1 \ v) \ a_2) =\]

\[\begin{align*}
  a_1 = a_2 & : v \\
  a_1 \neq a_2 & : (\text{get } m \ a_2)
\end{align*}\]

\[(\text{if } (= a_1 a_2) \ v \ (\text{get } m \ a_2))\]

Rewrite Rule 2

\[(\text{set } m \ a_1 \ (\text{get } m \ a_2)) =\]
Memory Rewriting

Rewrite Rule 1

\[(\text{get } (\text{set } m \ a_1 \ v) \ a_2) = \]
\[a_1 = a_2 : v \]
\[a_1 \neq a_2 : (\text{get } m \ a_2) \]
\[(\text{if } (= a_1 a_2) \ v (\text{get } m \ a_2)) \]

Rewrite Rule 2

\[(\text{set } m \ a_1 (\text{get } m \ a_2)) = \]
\[a_1 = a_2 : m \]
Memory Rewriting

Rewrite Rule 3

\[(\text{get} (\text{if } e_1 \ m_1 \ m_2) \ a_2) = \]
Memory Rewriting

Rewrite Rule 3

\[(\text{get } (\text{if } e_1 \ m_1 \ m_2) \ a_2) = (\text{if } e_1 \ (\text{get } m_1 \ a_2) \ (\text{get } m_2 \ a_2))\]
Rewrite Rule 3

\[
\text{(get (if } e_1 \text{ m}_1 \text{ m}_2 \text{) a}_2 \text{) } = \\
\text{(if } e_1 \text{ (get } m_1 \text{ a}_2 \text{) (get } m_2 \text{ a}_2 \text{))}
\]

\[m_1: 10 \text{ accesses}\]
\[m_2: 10 \text{ accesses}\]
Abstraction of \(m_1\) and \(m_2\) reduced from 20 to 10
Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting

- Results
- Conclusions
ICRAM Benchmarks

![Graph showing verification time vs ICRAM Address Size](image)

- **Verification time (sec)**: The x-axis represents the ICRAM Address Size, ranging from 4 to 13.
- **ICRAM Address Size**: The y-axis represents verification time in seconds, with a log scale.
- **Graph**: The graph compares verification times for two benchmarks, BAT and VCEGAR, with BAT shown in green diamonds and VCEGAR in blue squares.

- **Key Observation**: Verification time increases significantly with the ICRAM Address Size, with BAT consistently lower than VCEGAR across all sizes.
Out-of-Order Benchmarks

Size of abstracted memory vs. Number of writes to memory

- BAT with Rewriting
- BAT without Rewriting
Out-of-Order Benchmarks

The graph shows the verification time (sec) for BAT with and without rewriting, as the number of writes to memory increases. The verification time increases linearly with the number of writes.
2 Stage Pipelined Machine
Pipelined Machine Benchmarks

Verification time (sec)

Number of words in memory

Verification time (sec)
Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions
Conclusions and Future Work

Conclusions

- BAT: Tool for bit-level verification
- New automatic memory abstraction algorithm
- Memories are first class objects
- Introduced effective term-rewriting techniques
- Can verify 32-bit, 5 stage pipelines automatically

Future Work

- Automatically translate Verilog/VHDL to BAT
- Develop similar abstractions to reduce data path
- Integrate additional term-rewriting techniques
Automatic Memory Reductions for RTL Model Verification

Panagiotis Manolios
Sudarshan K. Srinivasan
Daron Vroon