BTOR: Bit-Precise Modelling of Word-Level Problems for Model Checking

Robert Brummayer, Armin Biere and Florian Lonsing

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria

BPR 2008
Princeton, New Jersey, USA

July 14th, 2008

Overview of Talk

R. Brummayer, FMV, JKU Linz

- Overview of BTOR
- Bit-vectors
- Overflow detection
- Arrays
- Sequential extension
- Experiments
- Conclusion

Overview of BTOR (1/2)

R. Brummayer, FMV, JKU Linz

- Quantifier-free
  - Bit-vectors
  - One-dimensional arrays
- Easy to parse and strongly typed
  - Visualization
- Clean semantics
  - Division by zero is fully defined
  - Result of shifting is fully defined

Overview of BTOR (2/2)

R. Brummayer, FMV, JKU Linz

- Sequential circuits
  - Support for synchronous registers and memories
- Model checking of safety properties
- Overflow detection
- Multi-rooted
  - Support for multiple outputs
- Support for binary, decimal and hexadecimal constants

Bit-Vector Example (1/2)

R. Brummayer, FMV, JKU Linz

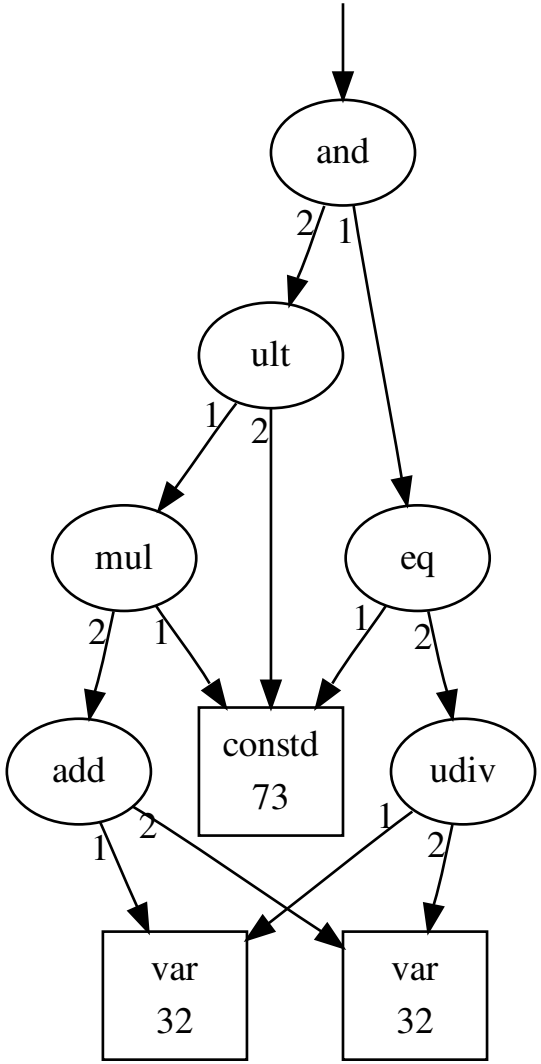
```
1 var 32
2 var 32
3 constd 32 73
4 udiv 32 1 2
5 eq 1 3 4
```

```
 6 add 32 1 2
 7 mul 32 3 6
 8 ult 1 7 3
 9 and 1 5 8
10 root 1 9
```

- First column: ID
- Second column: Operator
- Third column: Bit-width of result
- Other columns: IDs of operands, or immediates

Bit-Vector Example (2/2)

R. Brummayer, FMV, JKU Linz



Bit-Vector Constructors

R. Brummayer, FMV, JKU Linz

- `var`
  - Bit-width
  - Optional string for back annotation
- `const` for binary constants
- `constd` for decimal constants
- `consth` for hexa-decimal constants

Unary Bit-Vector Operators

R. Brummayer, FMV, JKU Linz

class	operators	w_1	w_r
negation	**not**, neg	n	n
reduction	redand, redor, redxor	n	1
arithmetic	inc, dec	n	n

- One's complement `not`
  - Can also be expressed by a minus in front of an operand
- Two's complement `neg`
- Reduction operators from Verilog
- Increment and decrement by one

Binary Bit-Vector Operators

R. Brummayer, FMV, JKU Linz

class	operators	w_1	w_2	w_r
bitwise	**and**, or, xor, nand, nor, xnor	n	n	n
boolean	implies, iff	1	1	1
arithmetic	**add**, sub, **mul**, **urem**, srem **udiv**, sdiv, [us]mod,	n	n	n
relational	**eq**, ne, **ult**, slt, [us]lte, [us]gt, [us]gte	n	n	1
shift	**sll**, **srl**, sra, ror, rol	n	$log_2 n$	n
overflow	[us]addo, [us]subo, [us]mulo, sdivo	n	n	1
concatenation	**concat**	n_1	n_2	$n_1 + n_2$

- Unsigned and signed context

- Right operands of shifts have bit-width $log_2 n$

class	operators	w_1	w_2	w_3	w_r
conditional	**cond**	1	n	n	n

- `cond` is the only ternary operator
  - Functional if-then-else

class	operators	w_1	upper	lower	w_r
extract	**slice**	n	u	l	u - l + 1

- `slice` extracts bits out of a bit-vector
  - Operands are immediates

Overflow Detection

R. Brummayer, FMV, JKU Linz

- BTOR supports set of overflow detection operators
- Typically, overflow detection important in software verification
  - but not in hardware verification
- Operators allow explicit modelling of overflow detection
  - The decision procedure do not have to care about overflows
- Alternative
  - Decision procedure treats overflows internally
  - Marks results as undefined
  - Three-valued logic necessary

Arrays (1/2)

R. Brummayer, FMV, JKU Linz

- BTOR supports one-dimensional bit-vector arrays
- Constructor
  - `array e i`
  - Elements have bit-width `e`
  - Indices have bit-wdith `i`, i.e. size is $2^{\texttt{i}}$
- Array access
  - `read`
  - `write`

Arrays (2/2)

R. Brummayer, FMV, JKU Linz

- If-then-else on arrays
  - `acond`
- Comparing arrays
  - Arrays of the same type can be compared for equality with `eq`
  - Two arrays are equal iff their elements are equal

Array Example (1/2)

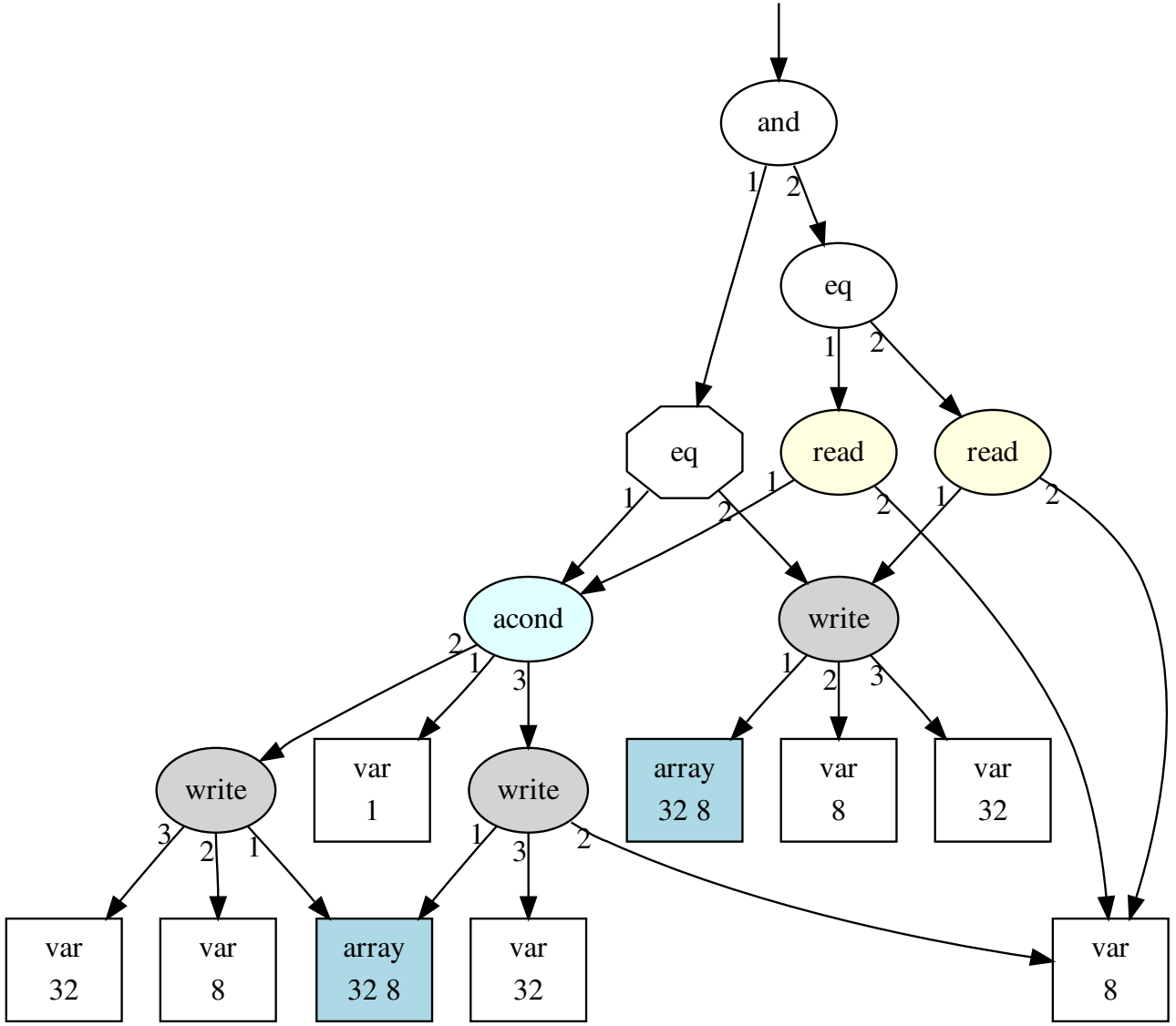
R. Brummayer, FMV, JKU Linz

```
1 array 32 8
2 array 32 8
3 var 8
4 var 32
5 var 8
6 var 32
7 var 8
8 var 32
9 var 1
10 write 32 8 1 3 4
11 write 32 8 1 5 6
12 acond 32 8 9 10 11
13 write 32 8 2 7 8
14 eq 1 12 13
15 read 32 12 5
16 read 32 13 5
17 eq 1 15 16
18 and 1 14 17
19 root 1 18
```

- `write` and `acond` return an array of type `32 8`
- `read` returns a bit-vector of bit-width `32`

Array Example (2/2)

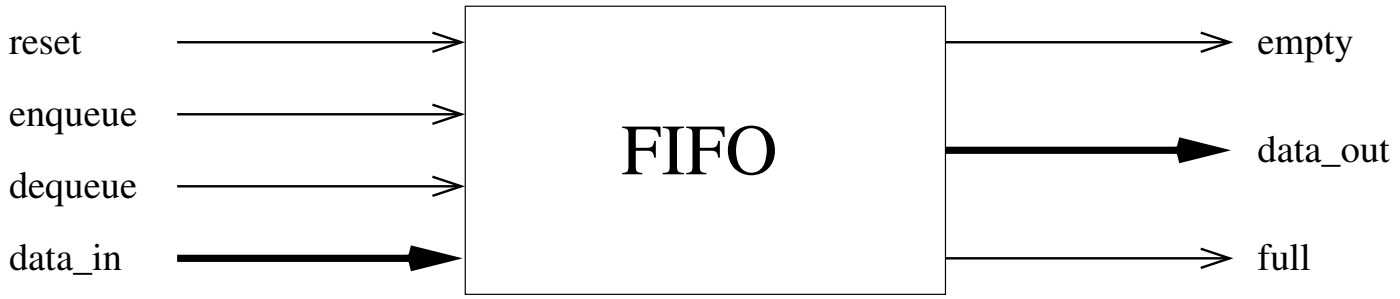
R. Brummayer, FMV, JKU Linz



- `next`
  - Funktion determines next state
  - Variables with next-functions model registers
    - Registers are initialized to zero
  - Arrays with next-functions model memories
    - Memories are uninitialized
- Variables and arrays without next-functions are primary inputs
  - Fresh in every cycle
- Boolean roots model safety property violations

Sequential Example: FIFO (1/3)

R. Brummayer, FMV, JKU Linz



- FIFO has internal memory
  - Can be organized as stack or queue
- FIFO needs registers to remember positions
- Safety property
  - `empty` and `full` may not be both 1

Sequential Example: FIFO (2/3)

R. Brummayer, FMV, JKU Linz

```
1 var 1 reset
2 var 1 enqeue
3 var 1 deqeue
4 var 32 data_in
5 xor 1 2 3

6 array 32 2
7 var 1 full_fs
8 var 1 empty_fs
9 var 32 data_out_fs
10 var 2 head_fs
11 var 2 tail_fs

12 constd 2 0
13 constd 2 1
14 constd 2 2
15 constd 2 3

16 zero 1
17 one 1
```

```
;next of head_fs
18 next 2 10 12

;next of tail_fs
19 sub 2 11 13
20 cond 2 8 11 19
21 add 2 11 13
22 cond 2 7 11 21
23 cond 2 2 22 20
24 cond 2 5 23 11
25 cond 2 1 24 12
26 next 2 11 25

;next of full_fs
27 eq 1 11 14
28 cond 1 27 17 7
29 cond 1 3 16 28
30 cond 1 5 29 7
31 cond 1 1 30 16
32 next 1 7 31
```

Sequential Example: FIFO (3/3)

R. Brummayer, FMV, JKU Linz

```
;next of empty_fs
33 eq 1 11 13
34 cond 1 33 17 8
35 cond 1 2 16 34
36 cond 1 5 35 8
37 cond 1 1 36 17
38 next 1 8 37

;next of data_out_fs
39 read 32 6 10
40 and 1 3 -8
41 cond 32 40 39 9
42 cond 32 5 41 9
43 cond 32 1 42 9
44 next 32 9 43
```

```
;next of stack memory
45 write 32 2 6 11 4
46 acond 32 2 7 6 45
47 read 32 6 13
48 write 32 2 6 12 47
49 read 32 6 14
50 write 32 2 48 13 49
51 acond 32 2 2 46 50
52 acond 32 2 5 51 6
53 acond 32 2 1 52 6
54 anext 32 2 6 53

;safety property
55 and 1 7 8
56 root 1 55
```

Experiments

R. Brummayer, FMV, JKU Linz

- Bounded model checker within our SMT solver Boolector
  - Bounded search for witnesses
  - k-induction with All-Different-Constraints
  - k-induction without All-Different-Constraints
- All-different-Constraints on memory states
  - Need extensional theory of arrays
  - Arrays are compared for equality
- Boolector can generate a model in the SAT case

Experiments: Equivalence Checking of two FIFO Implementations

R. Brummayer, FMV, JKU Linz

	boolector				z3	
	inc		non-inc		non-inc	
k	adc	noadc	adc	noadc	adc	noadc
00	0.0	0.0	0.0	0.0	0.0	0.0
01	0.1	0.0	0.1	0.0	0.0	0.0
02	0.8	0.1	0.9	0.1	300.3	0.0
03	15.4	0.3	6.1	0.4	639.9	313.1
04	41.9	0.7	38.0	1.1	*to*	*to*
05	91.2	1.6	96.5	2.7	*to*	*to*
06	419.8	3.7	267.8	5.8	*to*	*to*
07	*to*	6.8	*to*	11.7	*to*	*to*
08	*to*	14.3	*to*	23.9	*to*	*to*
09	*to*	31.1	*to*	47.8	*to*	*to*
10	*to*	73.7	*to*	97.2	*to*	*to*

Conclusion

R. Brummayer, FMV, JKU Linz

- Introduced BTOR
  - Bit-vectors
  - Arrays
  - Sequential extension
- Design decisions
- Overflow detection
- Experiments