Speeding Up Image Computation by using RTL Information

Christoph Meinel
FB Informatik
University of Trier
meinel@uni-trier.de

Christian Stangier
FB Informatik
University of Trier
stangier@uni-trier.de

Speeding Up Image Computation by using RTL Information

Abstract

Image computation is the core operation for optimization and formal verification of sequential systems like controllers or protocols. State exploration techniques based on OBDDs use a partitioned representation of the transition relation to keep the OBDD-sizes manageable. This paper presents a new approach that significantly increases the quality of the partitioning of the transition relation of controllers given in the hardware description language Verilog. The heuristic has been successfully applied to reachability analysis and symbolic model checking of real life designs, resulting in a significant reduction both in CPU time and memory consumption.

1 Introduction

The computation of the reachable states (RS) of a sequential circuit is an important task for synthesis, logic optimization and formal verification. The increasing complexity of sequential systems like controllers or protocols requires efficient RS computation methods. If the RS are computed by using Ordered Binary Decision Diagrams (OBDDs) [2], the system under consideration is represented in terms of a transition relation (TR). Since the monolithic representation of the circuit's TR usually leads to unmanageable large OBDD-sizes, the TR has to be partitioned [3, 6]. The quality of the partitioning is crucial for the efficiency of the RS computation. The computation of transitions will be unnecessarily time consuming, if the TR is divided into too many parts On the other hand a number of partitions that is too small will lead to a blow-up of OBDD-size and hence, memory consumption. Partitioning the TR is usually done without utilizing any external information. The standard method is to to sort the latches according to a benefit heuristic [7] and then apply a clustering algorithm. This clustering algorithm follows a greedy scheme [5] that is guided only by OBDD-size, i.e if the OBDD-size of a partition is exceeding a certain threshold a new partition has to be created.

In this paper we propose a heuristic for partitioning of protocols that uses information given by the register transfer level (RTL) description of controllers written in Verilog [10]. The application of our heuristic to reachability analysis reduced the computation time by 21% and memory consumption by 15% (overall). Additionally, we performed experiments with symbolic model checking [4]. Here, we got a reduction of CPU time by 63% and memory consumption by 55%, when applying our heuristic. Especially the experiments with model checking, where often properties concerning interaction of functional modules are checked, show the potential of our RTL based approach.

The heuristic is based on but not limited to Verilog. It may easily be adapted to other hardware description languages that provide RTL information like e.g. VHDL [8].

2 Preliminaries

2.1 Verilog

For the purpose of logic synthesis, designs are currently written in an hardware description language (HDL) at register transfer level (RTL). The term RTL is used for an HDL description style that utilizes a combination of *data flow* and *behavioral constructs*. Logic synthesis tools take the RTL HDL description to produce an optimized gate level netlist and high level synthesis tools at the behavioral level output RTL HDL descriptions. Verilog and VHDL are the most popular HDLs used for describing the functionality at RTL. Within the design cycle of optimization and verification the RTL level plays an important and frequently used role.

The design methodology in Verilog is a top down hierarchical modeling concept based on modules. A module is the basic building block in Verilog. Since modern complex designs require a structured hierarchical description to be feasible, Verilog is a first choice for an HDL.

2.2 Partitioned Transition Relations

The computation of the RS is a core task for optimization and verification of sequential systems. The essential part of OBDD-based traversal techniques is the transition relation TR:

$$\text{TR}(x, y) = \prod_i \delta_i(x_i) \equiv y_i,$$

which is the conjunction of the transition relations of all latches (δ_i denotes the transition function of the ith latch). This *monolithic* TR is represented as a single OBDD and usually much too large to allow an efficient computation of the RS. Sometimes a monolithic TR is too large to be represented with OBDDs. Therefore, more sophisticated RS computation methods make use of a *partitioned* TR [3], i.e. a cluster of OBDDs each of them representing the TR of a group of latches. A transition relation partitioned over sets of latches $P_1, \ldots, P_j$ can be described as follows:

$$\text{TR}(x, y) = \prod_j \prod_{i \in P_j} \delta_i(x_i) \equiv y_i.$$

The RS computation consists of repeated image computations $Img(\text{TR}, R)$ of a set of already reached states R:

$$Img(\text{TR}, R) = \exists_x(\text{TR}(x, y) \cdot R)$$

With the use of a partitioned TR the image computation can be iterated over P_j and the $\exists$ operation can be applied during the product computation *(early quantification)*. The so called *AndExist* [3] or *AndAbstract* operation may prevent a blow-up of OBDD-size during the image computation. Another important problem is finding an optimal schedule of the partitions for the AndExist operation. Geist and Beer [7] presented a heuristic for the ordering of partitions each representing a single state variable.

3 Partitioning of Transition Relations

The quality of the partitioning is crucial for the efficiency of the RS computation. The image computation is iterated over the partitions and includes costly product (i.e. AND) computations. Therefore, maintaining a large number of partitions is time consuming. A small number of partitions may lead to unmanageable large OBDDs. One extremum of this trade-off is the partitioning where each latch forms a partition, which is usually small but requires many iterations. The other extremum is a monolithic TR, that can be computed in one iteration but has large OBDD-size. Furthermore, the schedule of latches and the clusters is crucial for an efficient AndExist operation.

In the following we will describe the standard partitioning strategy and our new approach.

3.1 Common partitioning strategy

A common strategy for partitioning of the TR as it is used e.g. by VIS [5] proceeds in three steps:

1. **Order latches**. First, the latches are ordered by using a benefit heuristic [7] that performs a structural analysis of the latches transition function to address an effective AndExist operation. Hence, the heuristic considers: variables that may be quantified out, highest index in the function, etc.

2. **Cluster latches**. The single latch relations are clustered by following a greedy strategy. Latches are added to a OBDD (i.e. by performing AND) until the size of the OBDD exceeds a certain threshold.

3. **Order clusters**. In the last step the clusters are ordered similarly to the latches by using a benefit heuristic (VIS uses the same heuristic as in Step 1).

Figure 1 a) gives a schematic overview of this process.

3.2 RTL based Partitioning Heuristic

The way to build a complex design is to break it into modules, each with a dedicated functionality and a smaller complexity. For example communication protocols contain transmitter and receiver that represent independent modules. These modules are usually not too complex, thus the complexity of their TRs will be small. If a partition contains state variables of several modules, we need to represent the Cartesian product of these modules leading to a much more complex TR. The main reason for the efficiency of the partitioned TR approach is that state variables not appearing in other partitions are quantified out during the AndExist operation. This leads to much smaller OBDD-sizes and a faster computation. If the state variables of a module are spread over several partitions, the quantification does take effect only lately during the image computation. Therefore, most of the computation has to be done with large OBDDs.

RTL level description languages like Verilog support a hierarchical design methodology by providing module constructs. As it can be seen this modularization has effects on the image computation (see discussion below) that should not be neglected.

Although the standard method optimizes the partitioning twice, its main disadvantage is that it only uses structural information to optimize the partitioning for an efficient schedule for the AndExist operation during the image computation.

Our new heuristic improves this optimization by including additional semantical information about the represented functions. As the analysis and the experimental results show, there is a close conjunction between the RTL description and an efficient image computation.

The RTL heuristic proceeds in three steps:

1. **Group latches**. The latches are grouped according to the modules given in the top module of the RTL description in Verilog. Within the groups the latches are ordered by a lexicographic order that takes into account submodule names and bit numbers (names of latches from submodules are prefixed by the submodule name). Also, the bits of a certain register are named by the register and the bit number. The effect of this sorting is, that latches of a submodule within the group stay adjacent, without being grouped explicitely. The same holds for the bits of a register.

2. **Cluster groups**. The groups represent borders for the clusters. There is no cluster containing latches from different groups. To control the OBDD size of the clusters, the greedy partitioning strategy is applied within the groups. The clustering given by the groups lowers the influence of the arbitrary clustering produced by the OBDD-size threshold. Thus, resulting in a more *natural* partitioning.

3. **Order clusters**. In the last step the clusters are ordered by using the benefit heuristic from the standard method.

Figure 1 b) gives an overview of this strategy.

Modifications of this strategy are possible:

- **Step 1a)** As an additional step the benefit heuristic of the standard method may be applied to order the latches within the single groups. It emerged that the lexicographic order of the latches preserves more of the structure of the design and leads to better results.

- **Step 2a)** One may allow to create clusters that cross a group border. This will lead to a more compact representation of the TR with fewer clusters. Although the representation is more efficient the image computation does not perform as efficient as with the strict group borders. An explanation for this behavior is given below.

3.3 Analysis of the image computation

In the following we will analyze the influence of different partitioning schemes on the image computation of controllers that are inherently modularized. For the ease of understanding we consider a hypothetic protocol consisting of a transmitter (Tx) and a receiver

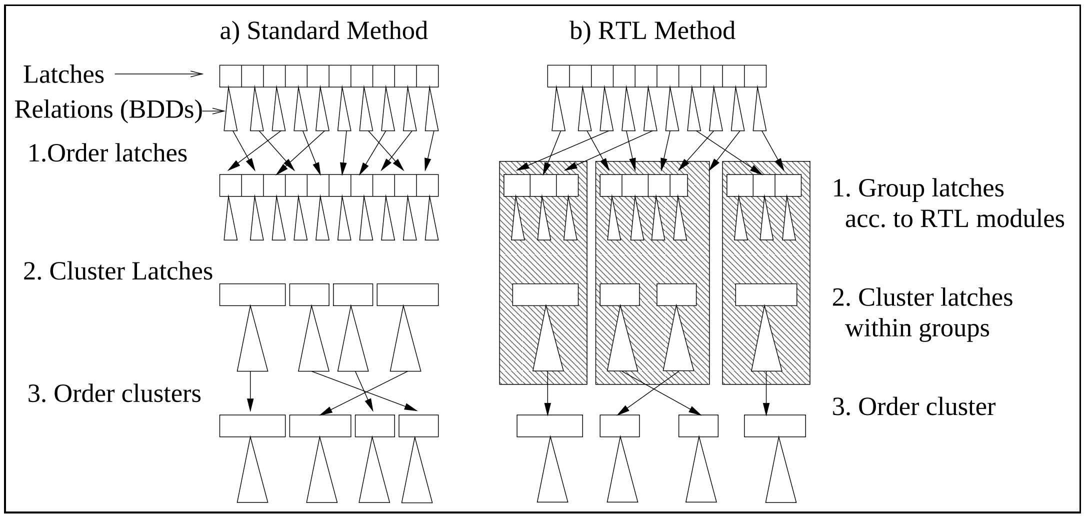

Figure 1: Schematic of Partitioning Strategies.

(Rx). The state variables of Tx are $t_0, \ldots, t_m$ and the corresponding transition functions are $\delta_{t_0} \ldots, \delta_{t_m}$. The state variables of Rx are $r_0, \ldots, r_n$, the corresponding transition functions are $\delta_{r_0} \ldots, \delta_{r_n}$.

A common greedy partitioning strategy merges δ_ts and δ_rs until the threshold for the OBDD-size of the partition is exceeded. We can expect that δ_ts and δ_rs appear in every partition. Hence, every partition depends on all variables $t_0, \ldots, t_m, r_0, \ldots, r_n$. This kind of "mixed" partitioning will not have negative effects on usual monolithic controllers, but it will have negative effects on modularized controllers:

First, the abstraction operation ($\exists$) included in the AndExist operation does take effect very lately, resulting in large OBDDs in the earlier computations.

Secondly, if the partitions depend on all variables, the AND operation within the AndExist is a complete AND operation with a worst case complexity of $O(|I_i| \cdot |P_i|)$, where P_i and I_i are the OBDDs of the ith iteration.

Even if the partitions are almost separated, but e.g. δ_{r_0} is represented in a partition of only δ_ts, the OBDD for δ_{r_0} cannot share any nodes with the OBDDs for the other functions. This results in an unnecessary large partition that is again depending on all variables.

We have a different situation if we use a modularized partitioning. Please notice that the set R of already reached states and the resulting Image (*Img*) of every iteration of the RS computation are independent of the chosen partitioning scheme. Their OBDDs may be different due to variable reordering, but the main reason for the better performance is the different partitioning scheme: Only transition functions of one type (δ_t resp. δ_r) are merged into one partition and iterated consecutively. This scheme performs better for the following reasons:

First, the AND operation is performed on a smaller number of variables. During the ith iteration the partition P_i and the intermediate result I_i share only a fraction of variables. The complexity for the AND operation of two OBDDs A and B with totally disjoint variable sets is $O(|A| + |B|)$. The AND operation in the modularized scheme is much closer to this complexity than the AND operation in the mixed scheme.

Secondly, if the end of a module is reached, the abstract operation will quantify out all variables of this module, resulting in smaller OBDDs.

This analysis is absolutely correct only for fully separated modules, but modularized controllers communicate only over very few signals and this analysis is valid although.

4 Experiments

4.1 Implementation

We implemented our strategy in the VIS-package [5] (version 1.3) using the underlying CUDD-package [9] (version 2.3.0). VIS is a popular verification and synthesis packages in academic research. It inherits state of the art techniques for OBDD manipulation, image and reachable states computation as well as formal verification techniques. Together with the vl2mv translator VIS provides a Verilog front-end needed for our heuristic.

4.2 Benchmarks

For our experiments we used Verilog designs from the Texas97 benchmark suite [1]. This publicly available benchmark suite contains real life designs including:

- MSI Cache Coherence Protocol
- PCI Local BUS
- PI BUS Protocol
- MESI Cache Coherence Protocol
- MPEG System Decoder
- DLX
- PowerPC 60x Bus Interface

The benchmark suite also contains properties given in CTL formulas for verification.

We chose those designs that represent RTL (i.e. including more than one module) rather than gate level descriptions. Considered were those designs that could be read in and whose transition relation could be build respecting our system limitations. Too small examples (CPU time $<$ 1s) were not considered.

4.3 Experimental Setup

We left all parameters of VIS and CUDD unchanged. The most important default values are:

- Partition cluster size = 5000
- Partition method for MDDs = inout
- OBDD variable reordering method = sifting
- First reordering threshold = 4004 nodes

The reachable states computation or the model checking was preceeded by a forced variable reordering. The CPU time was limited to 2 CPU hours and memory usage was limited to 200MB. All experiments were performed on Linux PentiumIII 500Mhz workstations.

4.4 Results of Reachability Experiments

As a first experiment we performed reachability analysis on the given benchmarks. For results see Table 1 and Table 2.

RS describes the number of reachable states of the design, Depth its sequential depth. Part gives the number of partitions of the transition relation. The OBDD-size of the transition relation cluster and the peak number of live nodes is given by TRn resp. Peakn. The CPU time is measured in seconds and given as Time. The columns denoted with % describe the improvement in percent [1].

At the bottom of Table 1 you can find the sum of all numbers of partitions, BDD-sizes and CPU-times. Also, the *average of the relative improvement* is given as well as the *total improvement*. In Table 2 results of experiments are given, where the standard method did not finish the computation.

The main result of these experiments is that using the RTL heuristic the reachable states are being computed faster and the OBDD sizes are smaller.

Although the OBDD sizes of the TR are comparable for both methods (the RTL method is 5% smaller), the OBDD peak sizes of the RTL method are 15% smaller than the peak sizes of the standard method. The computation time improves on 21% overall and 19% on average. The heuristic performs worse on 8 benchmarks, but these benchmarks represent only 8% of the original computation time and the time losses for these benchmarks add up to 98 seconds. In most cases the time losses result from extra reordering calls, that are triggered because the OBDDs are smaller (!) in comparison to the standard method.

As a conclusion one may say that the RTL heuristics performs stable and is especially useful for larger designs.

4.5 Results of Model Checking Experiments

In the second series we performed model checking experiments on the basis of the texas97 benchmarks.

In model checking the number of image computations usually is larger than for reachable states computation. On the other hand model checking is not dominated by the size of the reachable states set and the resulting variable reordering effort.

For results see Table 3. The notation in the table is the same as before. Reachable states and the depth of the circuit are not computed. Since various TRs for forward and backward image computation are built for one circuit, the size of the TR is not given (for comparison see Table 1). Additionally the number of image computations (img.comp) is given.

[1] $0 <$ improvement < 100; $-100 <$ impairment < 0

The experiments show significant improvements in time and space: The overall CPU time decreased by 63% overall and 54% on average. The RTL method outperforms the standard method for every benchmark. The decrease in computation time ranges from 10% up to 85%. The OBDD peak sizes could be lowered by 55% overall and 67% on average. A reason for this remarkable improvement may be the fact that often properties checked by model checking concern the interaction of modules in a design. The RTL heuristic targets exactly this behavior and produces a kind of a *smart* partitioning of the TR with respect to this verification task.

5 Conclusion

We presented a heuristic for optimizing the partitioning of the transition relation for reachable states computation and model checking of sequential systems written in the hardware description language Verilog. The heuristic significantly decreases computation time and memory consumption during reachable states computation and model checking and thus allows more efficient optimization and verification. The heuristic is general enough to be applied to other hardware description languages that provide RTL information.

References

[1] A. Aziz et. al., *Texas-97 benchmarks*, www-cad.EECS.Berkeley.EDU/Respep/Research/Vis/texas-97 .

[2] R. E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers, C-35, 1986, pp. 677-691.

[3] J. R. Burch, E. M. Clarke, D. E. Long, *Symbolic Model Checking with partitioned transition relations*, Proc. of Int. Conf. on VLSI, 1991.

[4] J. R. Burch, E. M. Clarke, D. L. Dill, L. J. Hwang and K. L. McMillan, *Symbolic model checking: 10^{20} states and beyond*, Proc. of LICS, 1990, pp. 428-439.

[5] R. K. Brayton, G. D. Hachtel, A. L. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. A. Edwards, S. P. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, T. Villa, *VIS: A System for Verification and Synthesis*, Proc. of Computer Aided Verification CAV'96, 1996, pp. 428-432.

[6] O. Coudert, C. Berthet and J. C. Madre, *Verification of Synchronous Machines using Symbolic Execution*, Proc. of Workshop on Automatic Verification Methods for Finite State Machines, LNCS 407, Springer, 1989, pp. 365-373.

[7] D. Geist and I. Beer, *Efficient Model Checking by Automated Ordering of Transition Relation Partitions*, Proc. of Computer Aided Verification CAV'94, 1994, pp. 294-310.

[8] R. D. M. Hunter and T. T. Johnson, *Introduction to VHDL*, Chapman & Hall, 1996.

[9] F. Somenzi, *CUDD: CU Decision Diagram Package*, ftp://vlsi.colorado.edu/pub/ .

[10] D.E. Thomas and P. Moorby, *The Verilog Hardware Description Language*, Kluwer, 1991.

Design			Standard VIS				RTL Method						
	Minterms	Depth	Parts	TRn	Peakn	Time	Parts	TRn	%	Peakn	%	Time	%
ONE	2.91e8	16	3	2175	20284	5.32	5	1987	9	19068	6	4.99	6
PCIabnorm	2631170	35	23	47472	81543	123.99	25	34646	27	87882	-7	132.75	-7
PCInorm	86528	30	20	31048	69291	71.93	21	39687	-22	69291	0	81.18	-11
SDLX_sdlx	12	11	17	26943	38106	75.12	25	12946	52	26324	31	35.23	53
TEST_PDLX_sdlx	155	154	25	66680	92726	264.74	29	22142	67	51894	44	137.6	48
TEST_SDLX_sdlx	155	154	25	66680	92726	267.72	29	22142	67	51894	44	137.69	49
mpeg	2081	17	36	39614	56082	293.66	36	18014	54	41684	26	257.36	12
multi_main	1144830	41	18	35804	70727	165.1	16	28368	21	49332	30	97.08	41
p62_LS_LS_V01	2823	61	36	54572	192422	238.25	41	67411	-19	171498	11	245.85	-3
p62_LS_LS_V02	1045	58	36	54544	135371	112.24	41	67349	-19	117970	13	131	-14
p62_LS_L_V01	2743	61	35	78281	193959	294.64	41	72312	8	180578	7	244.58	17
p62_LS_L_V02	1124	66	35	78336	122943	129.2	41	71405	9	119839	2	122.11	5
p62_LS_S_V01	2743	61	35	78281	193959	295.4	41	72312	8	180578	7	247.98	16
p62_LS_S_V02	1124	66	35	78336	122943	127.86	41	71405	9	119839	2	122.86	4
p62_L_L_V01	2445	48	36	54572	141902	202.28	41	67399	-19	150146	-5	137.79	32
p62_L_L_V02	2398	71	36	54544	141716	220.7	41	67349	-19	150533	-6	154.52	30
p62_L_S_V01	3637	85	37	73008	149810	234.7	41	73755	-1	158476	-5	231.23	1
p62_L_S_V02	1327	73	37	73104	108726	121.9	41	67873	7	132183	-18	128.96	-5
p62_ND_LS_V02	1152820	129	37	60005	577741	2537.9	42	73479	-18	493942	14	2018.5	20
p62_ND_L_V02	1570620	156	37	51639	646172	2062.4	42	70036	-26	481212	25	1945.8	6
p62_ND_S_V02	143788	106	37	51639	215531	488.15	42	70076	-26	224360	-4	512.54	-5
p62_S_S_V01	437	45	36	54907	108931	100.5	41	69298	-21	115093	-5	115.68	-13
p62_S_S_V02	317	42	36	54862	96168	94.35	42	69453	-21	113226	-15	102.38	-8
p62_V_LS_V01	595210	144	37	68540	803754	3908.8	41	69578	-1	777437	3	2976.4	24
p62_V_LS_V02	93185	125	37	69098	309808	852.2	41	68658	1	265985	14	600.64	29
p62_V_S_V01	59667	121	37	50733	323051	866.58	41	66873	-24	254568	21	572.3	34
p62_V_S_V02	22309	106	37	51562	197710	435.57	41	66235	-22	155482	21	267.49	39
single_main	1888	14	9	12841	26914	42.36	6	5868	54	17352	35	29.39	31
3-proc	3.65e8	32	8	18322	504293	375.03	5	3568	80	217770	57	101.61	73
3-proc_bin	3.65e8	32	7	19194	192013	96.39	4	5432	72	170849	11	76.23	21
2-proc	1137600	28	4	12792	43540	10.43	3	2593	80	23296	46	6.07	42
2-proc_bin	665518	28	4	10397	34516	6.69	3	3857	63	14885	57	4.02	40
Sum			888	1580525	6105378	15122.09	990	1523506		5204466		11979.83	
Average of rel. improvements:									13%		14%		19%
Total improvement:							-10%	5%		15%		21%	

Table 1: Comparison of Original VIS Partitioning and RTL Heuristic for RS Computation

Design			Standard VIS					RTL Method				
	Minterms	Depth	Parts	TRn	Peakn	Time	Depth	Parts	TRn	Peakn	Time	Depth
p62_ND_LS_V01	–	–	37	60066	Mem. out	–	37	42	73427	1183946	$>$7200	82
p62_ND_L_V01	3724760	134	37	51372	1408735	$>$7200	65	42	70072	1342148	5406.4	134
p62_ND_ND_V02	–	–	37	72712	Mem. out	–	33	41	71709	1289123	$>$7200	45

Table 2: Comparison of Original VIS Partitioning and RTL Heuristic for RS Computation for Unfinished Examples

Design		Standard VIS		RTL Method			
	Img.Comp	Peakn	Time	Peakn	%	Time	%
2-proc	264	871274	744.6	148936	83	112.6	85
PCIabnorm	303	176276	258.7	133146	24	159	38
PCInorm	206	81123	56.8	69291	15	51.1	10
multi_main	93	33796	36.4	33423	1	18.6	49
multi_main1	193	54022	48.8	33423	38	18.9	61
p62_ND_LS_V02-live	192	1564426	3693.7	461650	70	889.5	76
p62_ND_L_V02-live	200	2467117	$>$7200	1097001	55	2663.3	$\geq$ 63
p62_ND_S_V02-live	176	645918	1244.1	233639	64	272.2	78
p62_V_LS_V02-ccp	90	165200	227.8	158604	4	150.4	34
p62_V_LS_V02-live	178	1059895	1957.7	554997	48	1094.8	44
p62_V_S_V02-ccp	84	163439	193.6	132590	19	130.3	33
p62_V_S_V02-ioq	84	163439	179.6	132590	19	127.6	29
p62_V_S_V02-live	177	351553	500.7	381982	-8	405.3	19
Sum		7797478	$>$16343.5	3571272		6093.6	
Average of rel. improvements:					67%		52%
Total improvement:				55%		63%	

Table 3: Comparison of Original VIS Partitioning and RTL Heuristic for Model Checking