Formal Verification of High Level Synthesis by means of *SFG-Tracing.*

Mark Genoe, Luc Claesen, Eric Verlind, Hugo De Man

IMEC, Kapeldreef 75, B-3001 Leuven

Abstract.

SFG-Tracing is a multi-level formal verification methodology, allowing the behavioral verification of VLSI hardware implementations with respect to the specifications. Design levels can range from SPICE transistor netlist, gate netlist, structural and behavioral Register Transfer descriptions, and high level algorithmic specifications as they are used in High Level Synthesis. *SFG-Tracing* has been used succesfully for the automatic verification of the high level synthesis results for bit-serial implementations [14] in CATHEDRAL-I and microcoded implementations [13] in CATHEDRAL-II, from the layout-extracted SPICE transistor netlists up to the high level algorithmic specifications. A modem chip consisting of over 31,000 transistors is the largest design that has been fully verified up to now.

Current work is oriented towards automating the verification for high complexity synthesis results. For this a 230,000 transistor vocoder chip as synthesized by CATHEDRAL-II is the current driving application.

SFG-Tracing can be used as a feasible verification methodology for the full behavioral verification of high level synthesis results as well as for the verification of manual, or mixed manual/synthesized designs.

1 Introduction

High level synthesis [1, 2, 3] is a method which was originally aimed to be *correct by construction.* Both design bugs as well as software bugs in tools or their interfaces can result in large costs. This is caused by costs involved both in VLSI processing iterations as well as by profit loss due to a late market introduction. Many CAD-tools are involved in current state-of-the-art high level synthesis systems (scheduling, allocation, bus-merging, memory management, controller synthesis, data path synthesis, technology mapping, module generation, layout generation, standard cell place & route, floorplanning, block place & route, layout assembly, mask generation). There is no doubt that individual high level tools can or will be bug-free. As many high level synthesis tools are complex software systems and are still under construction and further elaboration, and due to the complex interaction with CAD tools, the possibility for errors is still open. *Independent verification* to cross-check the results from synthesis results is highly desirable to increase the confidence in design correctness.

Such cross checking of the results of *correctness-by-construction* CAD tools is not new. Even such established techniques like schematics entry followed by place and route tools to generated

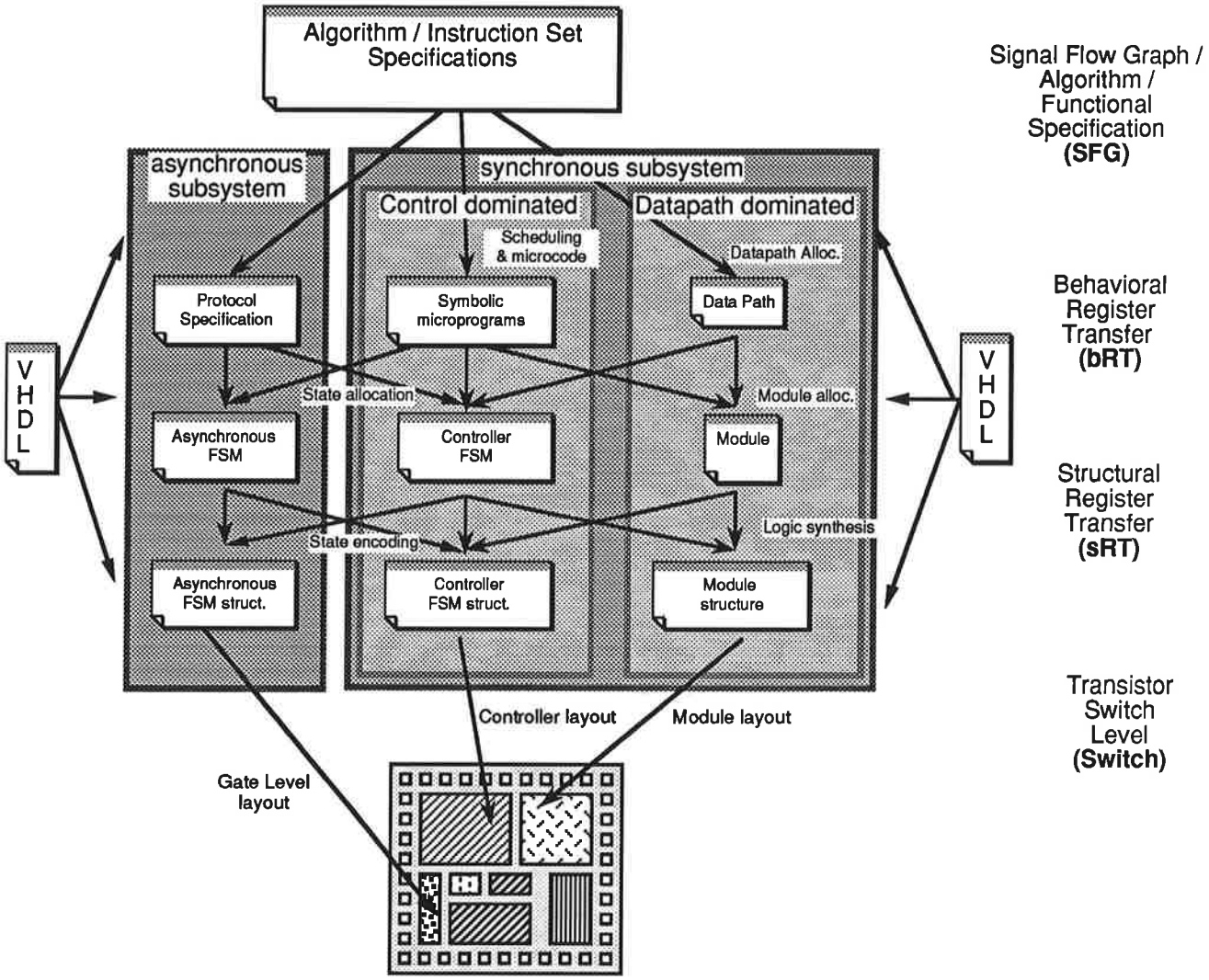

Figure 1: Aspects and Levels of Abstraction in Hardware Design.

the layouts, are in industrial environments cross checked for potential bugs, by netlist extraction from layout and netlist comparison with the original netlists as entered initially.

The need for the independent verification of synthesis results is also recognized by IBM [11] in the HIS high level synthesis system and by Siemens [15] in the CALLAS system.

2 The *SFG-Tracing* methodology.

SFG-Tracing [12] is a verification methodology which allows to verify lower level implementations with respect to higher level implementations. To understand the impact of this, the so-called *formal verification map* [16] is used.

This map is shown in fig. 1. Horizontally the design aspects are indicated (asynchrounous versus synchronous aspects). In the synchronous aspects a differentiation is made in the control dominated and the data path dominated aspects.

Vertically the levels of design abstraction are represented. From the bottom up to the higher levels these are: transistor switch level, structural Register Transfer level (sRT), behavioral Register Transfer level (bRT) and algorithmic or also called signal flow graph (SFG) level. At the algorithmic level no specific binding of the operations is made in terms of hardware, neither there has been done an allocation of specific control steps when operations are executed on the hardware. It is also well known that the algorithmic specfications can be either implemented in bit-serial as well as micro-coded as bit-parallel implementations. It is typically from the algorithmic level (SFG) down to the bRT level that the high level synthesis tasks of allocation and scheduling are situated. Currently most of the verification methods are located either at the sRT level, usually directly comparing the Boolean functionality of the combinatorial logic by means of BDD's [8]. The current wave of FSM verification based on the symbolic traversal (by means of BDD's) is situated at the sRT-bRT levels.

The comparison of complete behaviors from low level implementations up to high level specifications is too complex to be tackled as black boxes for realistically sized problems.

SFG-Tracing is a methodology [12] that uses the high level specification as a point of reference to verify the lower level implementations against. This has the advantage that the high level specification are most of the time much more abstract than the detailed low level implementations (that can contain thousands of transistors). In *SFG-Tracing* the high level specification is partitioned in a systematic way and all of the partitions in the specification are traced and their correct implementation is fully verified by symbolic analysis [5, 6, 7] and making use of BDD's for their correspondance with the specified behavior. In order to make this feasible a number of reference signals and their mapping functions from the implementation in space and time and under certain conditions have to be used [12]. The amount of signals at the borders of the partitions in the high level specification is much lower than the amount of signals in the implementation. E.g. the circuitry of the controller generated in the CATHEDRAL-II environment is not visible in the high level SILAGE specifications and do not have to be taken as reference signals. The effect of the controller is implicitly verified by the symbolic analysis and verification of the data operations on the implementation.

Some definitions:

- Reference signals : the input and output signals of each partitioned signal flow graph.
- Mapping functions : the correspondance of the reference signals between implementation and specification in space and in time.

3 Partitioning and verification.

In most industrial applications the complexity of the algorithmic specification (called SFG) is too high to verify only input-output behavior. A partitioning of the complete SFG into manageable pieces can be derived from intermediate signals out of the specification or given as references by the synthesis system. The correctness of each partitioned SFG (called pSFG) can be verified by symbolic analysis and boolean comparison. In practice this means that we have to verify the correct I/O timing for read and write operations, the correct functionality of each pSFG, and the non-corruption of stored signals between the pSFG's. Special constructs are built to verify conditional operations and iterations in the specification. All these aspects will be explained in the next paragraphs.

Verification of pSFG's : The verification of each pSFG can be done by comparing the bit-true functionality of both the specification and the implementation. The functionality of the specification can be derived from the semantics of the specification language operators, the implementation functionality from symbolic simulation with the signals in the datapath of the implementation. The representation of both formulas within the Cathedral Verification Environment is done by BDD's. Verifying the correctness of the pSFG behavior is so reduced to tautology checking of both derived BDD's. This can be illustrated by the following example, which is also the driving example of the other paragraphs.

```
#define  word   fix<4,0>
#define  N      100

function main(in[N]: word) out:word =
  {
  temp[0] = word(0);
  for (i:0 .. N-1):
    temp[i+1] = if (in[i] < temp[i]) -> temp[i]+in[i];
                else                 -> temp[i];
  out = temp[N];
  }
```

The partitioning is based on the availability of intermediate signals after high level synthesis. Let's suppose that in this case we can distinguish the signals *temp[0], i, (in[i] < temp[i]), temp[i+1]* and *out*, and take the pSFG with *(in[i] < temp[i])* as output under consideration. The non-corruption of stored values property (see later) claims that the inputs of this pSFG, which are *in[i]* and *temp[i]*, are available somewhere in the circuit. By symbolic analysis we can compute the circuit behaviour for this pSFG by making use of the mapping-information. This means that we know when and where we can find the computed output of this pSFG. The specification function of this pSFG is as follows: (derived from the semantics of the specification language)

```
small_0 = (!in[i]_0 && temp[i]_0)
small_1 = (!in[i]_1 && temp[i]_1) || (small_0 && !(in[i]_1 ^ temp[i]_1))
small_2 = (!in[i]_2 && temp[i]_2) || (small_1 && !(in[i]_2 ^ temp[i]_2))
small_3 = (in[i]_3 && !temp[i]_3) || (small_2 && !(in[i]_3 ^ temp[i]_3))
smaller_than = small_3
```

Once we have verified the correctness of this pSFG, we reduce this symbolic output expression to a new unique symbolic variable, which will be an input of one of the following pSFG's under examination, or which can also be an external output signal.

Correct I/O timing : The mapping functions do not contain only information about the reference signals of each pSFG, they also contain the necessary information about the external read and write operations. Indeed, external read or write signals are at least input or output of one pSFG. This means, applied to our driving example, we need to know exactly when the input signals *in[0..N-1]* are available at the input ports, and when the correct output signal *out* can be found at the output ports. When there are no read operations specified, "don't care"-values can be put on the external input ports to verify that these ports have no influence to the circuit behaviour when it is not expected.

Corruption of stored signals : A very important point is that we have to be sure that the resulting data from a pSFG will be stored correctly during the whole trajectory between the time that the signals are generated and the time that they are used as input for one of the following pSFG's. In *SFG-Tracing* this is done correctly by using only unique symbolic variables each time we have verified a pSFG. When we verify a pSFG-function with these variables as inputs, there is no way that they can come from somewhere else.

Conditional operations : Conditional operations can be verified in one single pSFG by labeling the results of each branch to the specific condition of that branch, and taking all results together by a multiplexor function. This is illustrated below for the if-then-else construct of our example above.

```
temp[i+1]_0 = (small_than && add_temp[i]_in[i]_0) || (!small_than && temp[i]_0)
temp[i+1]_1 = (small_than && add_temp[i]_in[i]_1) || (!small_than && temp[i]_1)
temp[i+1]_2 = (small_than && add_temp[i]_in[i]_2) || (!small_than && temp[i]_2)
temp[i+1]_3 = (small_than && add_temp[i]_in[i]_3) || (!small_than && temp[i]_3)
```

If the operations are not specified for each possible branch (e.g. if-then without else), the default case is nothing else than the previous value.

Loop constructs : By using symbolic boolean variables, we can reduce the number of times that we have to pass through the loop body to a feasible one. This is a serious advantage in case of microcoded controller architectures like Cathedral-II. Indeed, in this case we can verify loop constructs by applying the principles for symbolic induction. Such an induction proof is based on the correctness of two cases, the *base-case* and the *induction-case.*

The syntax of a loop construct can in general defined as follows:

```
for (Iterator : LowerBoundExpression .. UpperBoundExpression):
  {
  LoopBodyStatement definitions;
  }
```

The verification of the *base-case* consists of the correctness of the LowerBoundExpression for the loop-iterator, and the correctness of the LoopBodyStatements. When all this is proven correctly, we can replace all results -as it is defined by the *SFG-Tracing* methodology- by new symbolic variables expressing that the previous loop was checked correctly.

The UpperBoundExpression of the loop-iterator can be verified symbolically, as well as the increment operation of the loop-iterator. The UpperBoundExpression has to be replaced by a '0' in order to handle the induction-case after the base-case.

This *induction-case* is proven correctly by verifying all the pSFG's by using the symbolic values assigned after the correct verification of the base-case. Starting with these symbolic values of the previous iteration (i.e. n) we can verify the correctness of the next iteration (i.e. $n+1$), as it is aimed by the induction principle. The proof that we return each time to the same identical controller state can be done by comparing symbolically the controller bits after the base and induction case. Finally the UpperBoundExpression has to be replaced by a '1' expressing that we leave the loop construct, and go further on to another pSFG.

When loops are folded, we have to pass an additional number of times through the loop, convenient with the depth of the folded loop. So we are sure that we take the right inputs which can be generated a number of iterations before.

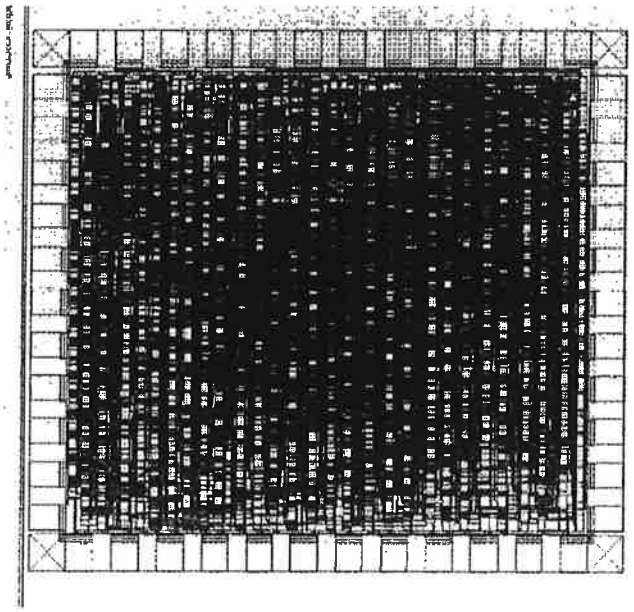

Figure 2: Chip layout of a 32,000 transistor modem receiver pulse shaper and equalizer chip as synthesized by CATHEDRAL-2. This chip has 3 ALU's of 14 bits, a multi-branch controller, micro sequencer, and testability circuitry. This chip has been fully verified w.r.t. high level specification using the *SFG-Tracing* methodology.

Parallel verification : The properties of *correctness, I/O timing* and *non-corruption of stored data* for the many pSFG's can in principle be verified in parallel on different machines. Thanks to the super-symbolic "don't care"-value the influence of possible bugs coming from other places than the one under consideration can be detected.

4 Results

SFG-Tracing has been used to succesfully verify results from the bit-serial CATHEDRAL-I silicon compiler [14]. Results obtained include a fifth order filter, and a 12,000 transistor 9th order wave digital filter implementation. No bugs have been reported up to now.

Table 1 summarises the results on the practical verification of CATHEDRAL-II designs as published in [13]. This includes a 31,614 modem chip as shown in fig 2 which consists of 3 cooperating datapaths of 3 ALU's, a micro-coded controller and testability circuits.

The cpu times (DEC 3100) include symbolic simulation and the checking by means of OBDD's [10] of the proof obligations per member of each partition.

Three bugs in the high level synthesis have been found, which were previously uncovered by normal simulations of the synthesis results. This motivates the need for the independent verification of high level synthesis results...

The current research concentrates on the automation of highly complex synthesis results from CATHEDRAL-II. In this respect a 230,000 transistor vocoder design is being used as a benchmark. This vocoder is used to convert a speech signal of 8kHz into an 800 bit/sec encoded signal for applications in satelite communication. In the verification it is the goal to start from

design	aplusb -m	aplusb -s	count -s	rec3 -s	vocoder -s
# MOS trans.	1,935	3,592	7,108	31,614	230,000
# latches	112	112	230	852	6,526
# states	10^{34}	10^{34}	10^{69}	10^{256}	10^{1960}
# subnetworks	362	912	1,719	7,698	12,067
# uniq. subn.	38	21	32	36	38
# mach. cycles	3	3	503	19	114,686
# sim. cycles	10	10	20	29	*1,400*
# cpu time	11.5 s	12 s	22 s	567 s	*8,000 m*
# script lines	105	105	640	4,782	*680,000*
# partitions	1	1	14	68	*2,200*

Table 1: Results of verification of transistor implementation with respect to high level specification for a number of designs synthesized by CATHEDRAL-II on a DEC 3100

as low a level as possible, thus encluding an as large as possible range of the design trajectory. Therefor the current efforts are mainly concerned with the management of the symbolic analysis data. The specification of the vocoder consists of 15 pages of SILAGE. It is partitioned in about 2200 partitions. The figures in *italic* mode of Table 1 are extrapolations from current results. The implementation of automatic partitioning and nested loop constructs is not yet finished for designs with complexities such as the vocoder-design.

5 Conclusions.

An automatic method for the formal verification of the results from high level synthesis is presented. This method starts from the full and flat transistor netlist (or alternatively the gate netlist) and compares if the high level specification is correctly implemented. To achieve this some information -the reference signals and mapping functions- is required as hints from the high level synthesis system. In case the high level synthesis system would give wrong "hints", *false negatives* can occur, but no *false possitives*. *SFG-Tracing* is being automated for the verification of the results from CATHEDRAL-II. The method is however also useful for the verification of manual, or mixed manual/synthesized designs. For these designs the essential hints have to be supported by the designers.

References

[1] D. Gajski (Ed.), "Silicon Compilation", *Addison Wesley*, Reading, Mass., 1988.

[2] M.C. McFarland, A.C. Parker, R. Camposano, "The High-Level Synthesis of Digital Systems", *Proceedings of the IEEE*, Vol. 78, No. 2, February 1990, pp.301-318.

[3] R. Camposano, W. Wolf (Eds.), "Trends in High-Level Synthesis", Kluwer Academic Publishers, Norwell, MA, 1991.

[4] H. De Man, J. Rabaey, P. Six, L. Claesen, "Cathedral-II: A silicon compiler for digital signal processing", *IEEE Design & Test of Computers*, December 1986, Vol. 3, No. 6, pp.73-85.

[5] R.E. Bryant, "Algorithmic aspects of symbolic switch network analysis", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 4, July 1987, pp. 618-633.

[6] R.E. Bryant, "Boolean Analysis of MOS Circuits", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 4, July 1987, pp. 634-649.

[7] R.E. Bryant, D. Beatty, K. Brace, K. Cho, T. Sheffer, "COSMOS: A Compiled Simulator for MOS Circuits", 24th Design Automation Conference, pp. 9-16, 1987.

[8] R.E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, Vol. C-35 No. 8, August 1986, pp. 667-691.

[9] S. Bose, A. Fisher, "Automatic Verification of Synchronous Circuits using Symbolic Logic Simulation and Temporal Logic", *Formal VLSI Correctness Verification*, ed. L.Claesen, ISBN 0 444 88688 5, North-Holland Elsevier Science Publishers, 1990, p.151-158.

[10] K.S. Brace, R.L. Rudell, R.E. Bryant, "Efficient Implementation of a BDD Package", *Proc. 27th ACM/IEEE DAC*, 1990, pp. 40-45.

[11] F. Corella, R. Camposano, R. Bergamashi, M. Payer, "Verification of Synchronous Sequential Circuits Obtained from Algorithmic Specifications", *Proc. International Conference on Computer Hardware Description Languages and their Applications*, ed. D. Borrione, R. Waxman, Elsevier Science Publishers, (North-Holland), IFIP 10.2, Marseille, France, 22-24 April 1991, pp. 229-247.

[12] L.Claesen, F.Proesmans, E.Verlind, H.De Man, "SFG-Tracing: a Methodology for the Automatic Verification of MOS Transistor Level Implementations from High Level Behavioral Specifications", *Proceedings ACM-SIGDA International Workshop on Formal Methods in VLSI Design*, ed. P.A. Subrahmanyam, January 9-11, 1991.

[13] M. Genoe, L. Claesen, E. Proesmans, E. Verlind, H. De Man, "Illustration of the *SFG-Tracing* Multi-Level Behavioral Verification Methodology, by the Correctness Proof of a High to Low Level Synthesis Application in CATHEDRAL-II", *Proc. IEEE ICCD-91*, Conference, Cambridge MA, October 14-16, 1991.

[14] F. Proesmans, L. Claesen, E. Verlind, M. Genoe, H. De Man, "Verification Strategy of the CATHEDRAL-1 silicon compiler based on the SFG-Tracing methodology", *Proc. IEEE CompEuro-92 conference*, The Hague, The Netherlands, 4-8 May, 1992.

[15] M. Payer, T. Filkorn, "Symbolic Verification of Sequential Circuits Synthesized with CALLAS", IFIP Workshop on Application Oriented Synthesis, Dresden, March 23-25, 1991.

[16] L. Claesen, D. Borrione, H. Eveking, G. Milne, J.L. Paillet, P. Prinetto, "CHARME: Towards Formal Design and Verification for Provably Correct VLSI Hardware", *in Correct Hardware Design Methodologies*, ed. P. Prinetto, P. Camurati, 1992 Elsevier Science Publishers (North-Holland), pp. 3-25.