High Level Synthesis Using Operation Properties

Jan Langer, Ulrich Heinkel
Chemnitz University of Technology
Chemnitz, Germany
{*laja,heinkel*}*@infotech.tu-chemnitz.de*

***Abstract*—We propose a high level synthesis approach to generate RT level hardware from a specification of operation properties. The property language is called InTerval Language (ITL) and we assume the set of properties is complete, such that the properties alone are strong enough to map every possible sequence of input data to exactly one sequence of output data. A major advantage of using operation properties as a design method is the existence of commercial tools to check the completeness and consistency of the property set. Furthermore, operation properties are well suited for specifications of consecutive operations of finite length. We show the practicality of our method by implementing a particle filter for a localization application.**

I. Introduction

Over the last years, formal verification has become powerful enough to be applicable to large designs. Writing temporal properties to capture the behavior of a system is now common knowledge for both designers and verification engineers. Usually, properties are used as assertions in a traditional simulation-based verification strategy or they are formally proven by a model checker. Recently, they are also used to generate models that can monitor the correctness of a design during simulation or emulation. Alternatively, such models can represent an abstraction of specific aspects of the design to speed up simulation or to enable the simulation of components before they are actually implemented. A new idea is to derive the complete implementation of the design from a specification consisting of properties.

A *Gap Free Verification* methodology based on operation properties has been introduced by the commercial tool 360MV^{TM}by OneSpin Solutions [1]. It provides a special property syntax known as InTerval Language (ITL). A set of additional rules helps to write a complete set of properties, that explicitly covers the design intent for every valid sequence of input values. The tool employs a powerful engine to prove the completeness of the property set as well as the correctness of each individual property with respect to the design. A property set is said to be complete if the conjunction of the properties alone is able to map every valid sequence of input data to exactly one corresponding sequence of output data [2], [3]. The completeness of a property set can be proven without the need of an actual design. It is possible to use PSL or SVA properties in a similar methodology. However, we concentrate on ITL, because, in contrast to PSL or SVA, it is specifically designed for specifying complete sets of properties.

In this work, we argue that using operation properties is a very good description method to design certain types of hardware components. The description of operation properties resembles timing diagrams in traditional specifications. Furthermore, we developed a synthesis tool that can generate very large circuits and is not prone to the state space explosion problem. We achieve this by avoiding the need to handle the circuit at the boolean level. We derive the control graph only from the structure of the properties and the data flow is directly mapped to hardware on RT level. Using the existing methodology and tools during the specification phase to prove consistency and completeness of the property set is a major advantage of using operation properties.

Writing a complete ITL specification based on operation properties is often a grey boxing approach, i.e. a certain level of insight into the inner workings of the desired design is required. This makes the specification less abstract than traditional Linear Time Logic (LTL) formulas. However, for the same reason it appears more appropriate to derive an actual implementation from ITL.

This paper is structured as follows: in Section II we introduce some of the previous work on synthesizing hardware from properties. In Section III we briefly show the structure and verification methodology behind operation properties. Section IV describes the basic algorithm we use for our synthesis approach and in Section V we show two examples we have implemented. Section VI summarizes this paper and gives a short outlook of future work.

II. Previous Work

Much effort has been put into the efficient generation of models of single properties used as monitors, or sets of properties describing the complete behavior of a component. Most of this work has been focused on a specification consisting of properties in LTL, mainly a subset of PSL [4]–[10]. All methods to synthesize LTL formulas are restricted in either the subset of supported constructs or the complexity they can handle. Another problem is the ambiguity of the specification. In most cases, a property or a set of properties

This research work was supported in part by the German Federal Ministry of Education and Research (BMBF) in the project HERKULES under the contract number 01 M 3082.

is satisfied by more than one exact behavior. Thus, the synthesis method can either create a general solution that contains all consistent behavior or an arbitrarily chosen specific solution.

In [11] a specification consisting of properties in the form $A \rightarrow P$ are normalized and transformed into an executable design description.

III. Operation Properties

In order to understand our synthesis approach, it is necessary to understand the underlying verification methodology [12]. It uses a set of properties P, called operation properties. Each property $p \in P$ is a tuple $(A, C, t_{Left}, t_{Right})$ of assumptions A, commitments C and the two time points t_{Left} and t_{Right}, that indicate the start and end time of the property. An assumption is a function of all input values, output values and internal state variables of the component. It can use those values at every time point relative to the respective property and must evaluate to true or false. An assumption could be the function

$$g_{in}(t) + h_{in}(t+3) < 10,$$

that delays input g at time t, adds input h at time $t+3$ and checks if the result is less than 10. The time points are not absolute values. They always represent an offset to an arbitrarily chosen time t. The time point t_a at which an assumption a is completely evaluated is the maximum time point of all variable accesses in a, $t+3$ in the above example.

A commitment $c \in C$ is a function similar to an assumption, that evaluates not to a boolean value, but to a value that is assigned to an internal or output variable at a specific time point. An exemplary function that assigns the sum of two input variables to output o_{out} at time point $t+4$ is given by

$$o_{out}(t+4) \leftarrow g_{in}(t) + h_{in}(t+3).$$

Properties that are not intended to synthesize a design can use commitments of arbitrary form. However, our synthesis approach requires the above restrictions to generate valid design descriptions.

The time point t_{Last} of a property is the maximum time at which either an assumption is completely evaluated or a signal is assigned via a commitment. Furthermore, it must be greater or equal to t_{Right}.

A complete property describing the $SYNC$ operation of a frame synchronization component (BITFRAMER) is listed in Alg. 1. It detects a synchronization word "1010" in a serial stream of input data and generates the correct output signals. This property will contain four assumptions at time points t to $t+3$ and twelve commitments (at time points $t+5$ to $t+16$) for each of the three output signals. The time points t_{Left} and t_{Right} are $t+3$ and $t+15$ respectively. t_{Last} will be $t+16$.

The described component has one input signal din representing the serial input data stream. Additionally there are

Algorithm 1 $SYNC$ property

```
property sync is
  assume:
    at t   : din = "1";
    at t+1 : din = "0";
    at t+2 : din = "1";
    at t+3 : din = "0";
  prove:
    during [t+5,t+16] : inframe = '1';
    during [t+5,t+16] : dout = prev(din);
    during [t+5,t+16] : frmpulse = '0';
  left_hook:  t+3;
  right_hook: t+15;
end property;
```

three output signals. $dout$ is the serial output data stream, whereas $frmpulse$ signals the start of each completely synchronized frame. The $inframe$ signal is valid as long as the output stream is synchronized.

There may exist an operation $prev(d, n)$ representing the value of the variable d exactly n time steps before the time step specified with at or $during$. In contrast to the original verification methodology, we do not need to handle the internal state of the design in the properties. In case we need to prove the correctness of those properties against an implementation, we would have to add the necessary statements to our property before using the 360MV tool.

Additionally, a property graph is required that defines the successors of each property. For two consecutive properties, the time point t_{Right} of the predecessor is equivalent to the time point t_{Left} of the successor property. In case a property has more than one successor, the assumptions of the successor properties decide which property will define the design's behavior. A property graph for our BITFRAMER example is shown in Fig. 1.

After reset, the design always starts at the first time point of the reset property. Every possible sequence of input data must lead to one or more sequence of properties. However, in case there is more than one matching sequence, all sequences must result in the same sequence of output values. The difference to a traditional FSM is the fact that an operation property is usually active for more than one time point, and its duration differs from property to property.

IV. Synthesis Algorithm

In this section we first propose an algorithm to synthesize the control flow of the design. After that we describe how assumptions and commitments form the corresponding data flow. During construction of the control graph, the commitments are not considered and the assumptions are simplified to boolean predicates. Furthermore, commitments are only allowed to assign variables at least one time point after

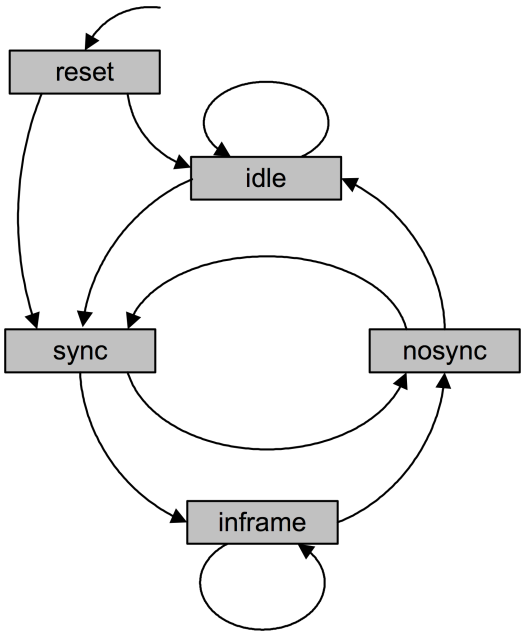

Figure 1. Property graph of the BITFRAMER component.

the last variable access, in order to guarantee synchronous behavior, i.e. a commitment

$$z_{out}(t+2) \leftarrow x_{in}(t) + y_{in}(t+3) \qquad \text{(invalid)}$$

is prohibited because driving the output value of z at time point $t+2$ requires the value of y at time $t+3$.

The property graph of our BITFRAMER example is shown in a different representation in Fig. 2. Now, the time points of each property are unrolled. The question marks indicate the time points of the four assumptions employed in this example. For each property, the light grey and dark grey time points are t_{Left} and t_{Right}, respectively. The transitions between properties do not consume time and can be compared to ϵ transitions in conventional non-deterministic finite automata. However, there is a difference. The time steps after t_{Right} are executed, no matter which property follows. In our example the overlapping between properties is relatively small. In case of a deep pipeline behavior of the design and short properties in relation to the pipeline depth, the overlapping can be significant and more than two properties can overlap. Thus, the property graph in Fig. 2 is not yet ready for synthesis. The following algorithm transforms it to a non-deterministic finite state machine that does not exhibit overlapping behavior. For the rest of the paper, a time point t of a property p is referred to as property position $p{:}\,t$.

The graph that is generated by this algorithm is defined as

$$H(S, E, A', C')$$

where S is the set of states of the new control flow graph

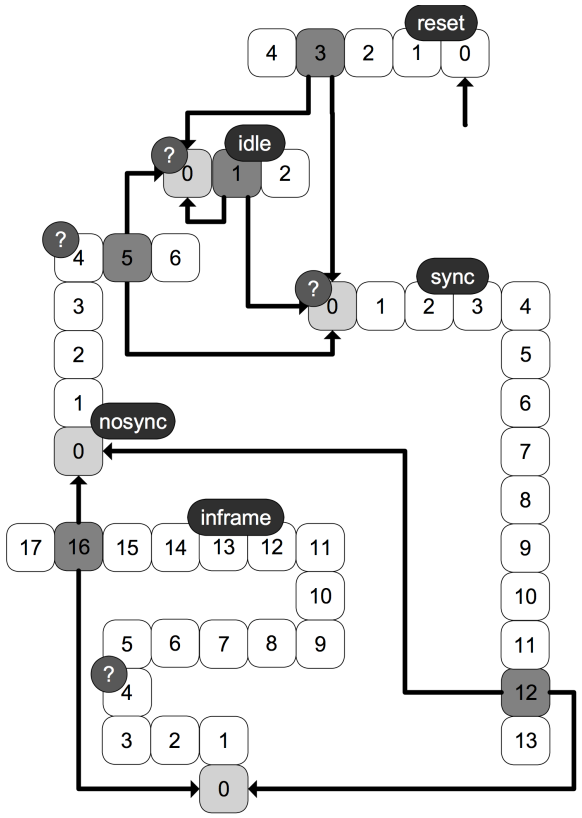

Figure 2. Unrolled property graph of the BITFRAMER component.

and E is the set of transistions of this graph. All transitions are non-deterministic but no ϵ transitions are allowed. The relation $A' : S \mapsto 2^A$ annotates every state with a set of assumptions from the corresponding properties. The relation $C' : S \mapsto 2^C$ is similar for the commitments.

Algorithm 2 Function *create*

```
s_0 ← {reset: 0}
S ← {s_0}
Q ← {s_0}
while Q ≠ ∅ do
  s ← choose from Q
  Q ← Q \ {s}
  for all s' ∈ successor(s) do
    if s' ∉ S then
      Q ← Q ∪ s'
      S ← S ∪ s'
    end if
    E ← E ∪ {s, s'}
  end for
end while
```

A state $s \in S$ is an ordered non-empty set of property positions. Every element $p{:}\,t \in s$ of this set defines that the state s has to exhibit the behavior described in property p at

time point t. Especially all corresponding assumptions and commitments must be fulfilled. The order of the elements depends on the number of time steps since t_{Left} of the property. The *minimum* element of the state is defined as the property position with the shortest time passed since t_{Left}. The construction algorithm guarantees that exactly one property position is *active*, meaning that its time point is less than t_{Right}. Naturally, this element is also the minimum element in s.

Algorithm 2 starts constructing the graph H by adding the initial state $\{reset\colon 0\}$ to a temporary set Q and iteratively searching for states following the already visited ones and adding the required transitions to E accordingly. New states that are found are stored in Q until they are visited. The algorithm terminates when no new states need to be added to S, i.e. the set Q runs empty.

Algorithm 3 Function *successor(s)*

```
s' ← ∅
for all p: t_p ∈ s do
  if t_p + 1 ≤ t_Last(p) then
    s' ← s' ∪ p: (t_p + 1)
  end if
end for
p: t_p ← minimum element from s'
if t_Right(p) = t_p then
  succ ← ∅
  for all q ∈ P | {p, q} ∈ U do
    s'' ← s' ∪ {q: t_Left(q)}
    succ ← succ ∪ s''
  end for
  return succ
else
  return {s'}
end if
```

In order to find the states following a given state s, Alg. 3 is used. Basically, it increments the time points of every property position in s and drops the ones that exceed the corresponding t_{Last}. Afterwards, the minimum element is selected and in case it is the t_{Right} of its property, the state is split into different states for each successor of the property. The position t_{Left} of the corresponding property is added to each state.

In a next step the assumptions and commitments are annotated on the control graph. Fig. 3 shows the result of our algorithm for the BITFRAMER example. All transitions are non-deterministic and in general without conditions. The question marks on certain states indicate that the state has annotated assumptions. Commitments are not shown. Usually, non-deterministic state machines are first transformed to deterministic ones and then mapped to hardware. However, to avoid the state space explosion problem of this transformation, we directly implement the non-deterministic

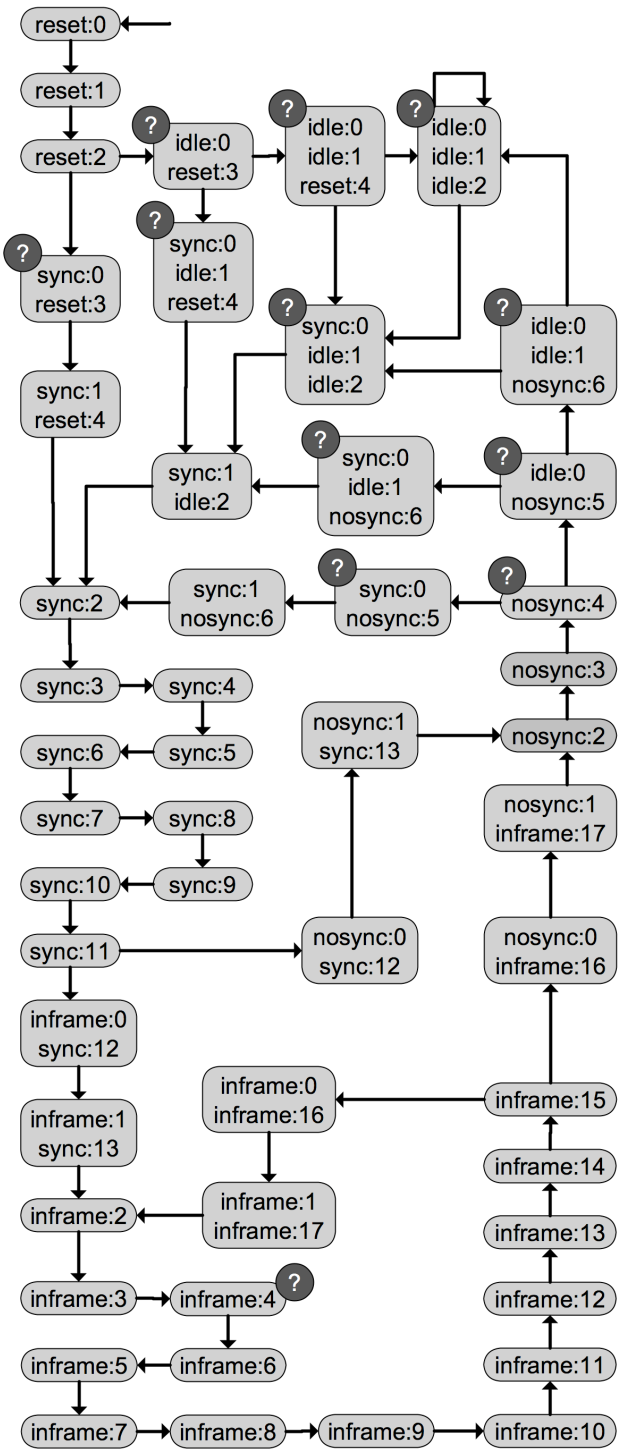

Figure 3. Generated control graph H of the BITFRAMER component.

graph in RT level hardware by means of the algorithm presented in [13] and [14]. It encodes each state as a single bit variable. The variable is set to $'1'$ if any of its predecessor states is active and all of its annotated assumptions are fulfilled. The result is very efficiently generated but does not scale well in terms of flip-flop usage in case very long properties are needed.

The correctness of the result of the algorithm depends on the completeness checks by 360MV to guarantee a correct synthesis. This is because we implicitly assume those restrictions. As an example, the synthesized design will be wrong in case the assumptions of the $IDLE$ and $SYNC$ properties are not mutually exclusive.

The data flow components (assumptions and commitments) are mapped directly to RT hardware. ITL expressions and calls to macros are unrolled and flattened resulting in a plain netlist of ITL operators. We use a library of synthetic operators implemented in VHDL for all possible ITL constructs.

V. Results

In order to demonstrate the conciseness of the proposed design method, we specified and synthesized one of our current design projects using operation properties. The component implements a particle filter used for localization purposes [15]. A particle filter is an estimation technique for Bayesian models. Each of the 8192 single particles of the filter represents a point in the state space that carries a weight. The weight is a simple estimate of the probability density function at that state point.

We split the specification into two partitions and connect them by FIFOs. The first step sequentially updates each particle and computes its weight. When all particles are updated and the sum of the weights is determined, a resampling step drops particles with low weight and clones particles with high weight. Fig. 4 shows the general structure and the data flow of the particles.

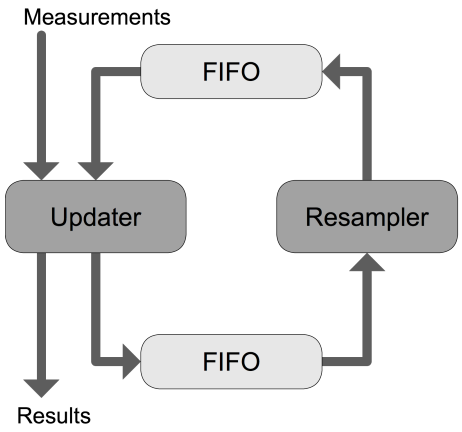

Figure 4. Structure of the particle filter.

The Updater component requires four properties to describe its behavior, whereas the Resampler block needs five. All properties have been written in just under three days and the synthesis to VHDL takes 1 second. The filter sequentially processes all 8192 particles, a number that is appropriate for our 3D localization application. We have been able to implement the complete particle filter in a Xilinx Virtex-II Pro FPGA and it uses roughly half the available slices. Table I shows detailed information of the synthesis results. The maximum frequency is 4.3 MHz, which is quite low but still meets our design goals by a wide margin. One major problem of the generation has been the design of the weight computation. We had to carefully revise our initial properties to use efficient data path algorithms. The initial square root algorithm consumed about four times the available FPGA resources, but after identifying the problem, it was possible to dramatically reduce the resources. Once algorithms like square root or division are implemented as ITL macros, it will be easy to reuse them in other applications.

Resource	Used	Utilization
Flip-Flops	2835	9.76%
CLB Slices	7741	56.52%
Block RAMs	83	61.03%
Block Multipliers	113	83.09%

Table I
Results of the synthesis algorithm for the Particle design using operation properties on a Xilinx 2VP30ff896 device.

The BITFRAMER component is a very simple abstraction of another motivating example for the property based design. The full framer component uses 16 Bit wide inputs and outputs and the frame length is about 20 cycles. The specification consists of five parameterized properties and the generation runs about half a second.

VI. Conclusion

In general, the proposed method offers a very comfortable way to specify complicated timing diagrams for output signals. Often, the specification engineer thinks in operations and its very straightforward to implement those ideas in operation properties. Usually, it is simpler and less error prone to change the time points of signal assignments using properties than changing it in manually coded VHDL.

Future work will concentrate on finding and implementing a control flow synthesis algorithm that constructs a deterministic state machine instead of a non-deterministic one. This will open opportunities to implement an important ITL construct known as freeze variables more efficiently. This improvement will also enable our synthesis algorithm to handle very long properties, such as a synchronization framer of full functionality that motivated our initial effort for this tool.

The other weakness of our current tool is the method of generating VHDL. Some basic optimizations like sharing

common logic or inserting RAM instances when necessary are not performed. For this step, we rely on the RTL synthesis tool, resulting in a high run time and memory usage.

The maximum frequency of circuits generated by our approach is often not yet satifactory. The implementation of the control flow in a deterministic state machine will enable further circuit optimization techniques that could increase performance.

REFERENCES

[1] Onespin solutions. [Online]. Available: http://www.onespin-solutions.com

[2] J. Bormann and H. Busch, "Verfahren zur Bestimmung der Güte einer Menge von Eigenschaften, verwendbar zur Verifikation and zur Spezifikation von Schaltungen (Method for determining the quality of a set of properties, applicable for the verification and specification of circuits)," European Patent Application issued 2005. Publication number EP1764715.

[3] W. Buettner and M. Siegel. Achieving completeness in IP functional verification. [Online]. Available: http://www.eetimes.com/showArticle.jhtml?articleID=197005268

[4] S. Ruah, D. Fisman, and S. Ben-David, "Automata construction for on-the-fly model checking PSL safety simple subset," IBM, Tech. Rep. H-0234, 2005.

[5] N. Piterman and A. Pnueli, "Synthesis of reactive(1) designs," in *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2006, pp. 364–380.

[6] M. Boule and Z. Zilic, "Efficient automata-based assertion-checker synthesis of SEREs for hardware emulation," in *Asia South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2007, pp. 324–329.

[7] Y. Abarbanel, I. Beer, L. Gluhovsky, S. Keidar, and Y. Wolfsthal, *FoCs - Automatic Generation of Simulation Checkers from Formal Specifications*. Springer, 2000, pp. 538–542.

[8] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer, "Specify, compile, run: Hardware from PSL," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 190, no. 4, pp. 3–16, November 2007.

[9] ——, "Automatic hardware synthesis from specifications: A case study," in *Design, Automation and Test in Europe (DATE)*, 2007.

[10] K. Morin-Allory and D. Borrione, "Proven correct monitors from PSL specifications," in *Design, Automation, and Test in Europe (DATE)*, 2006, pp. 1246–1251.

[11] M. Schickel, V. Nimbler, M. Braun, and H. Eveking, *An Efficient Synthesis Method for Property-Based Design in Formal Verification: On Consistency and Completeness of Property-Sets*. Springer, 2007, ch. 11, pp. 179–196.

[12] M. D. Nguyen, M. Thalmaier, M. Wedler, J. Bormann, D. Stoffel, and W. Kunz, "Unbounded protocol compliance verification using interval property checking with invariants," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 11, pp. 2068–2082, November 2008.

[13] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using FPGAs," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2001.

[14] R. Siegmund, "Ein Verfahren zur Spezifikation von Protokollen für die Verifikation und Synthese protokollorientierter digitaler Systeme," Ph.D. dissertation, Technische Universität Chemnitz, July 2005.

[15] S. Thrun, W. Burgard, and D. Fox, *The Particle Filter*. MIT Press, 2005, ch. 4.3, pp. 96–113.