Verification of Retiming Transformations

Jan Vandenbergh, Luc Claesen, Hugo De Man

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium
phone: +32-16-281220, email: vandenbj@imec.be

Abstract

In this paper we present an algorithm to verify the correctness of the results of retiming transformations. Starting from a high level data path description, Chopin-2 [5], the retiming tool used in IMEC, performs retiming, pipelining and hardware selection to construct an area-minimised data-path meeting the timing constraints. Making use of the formal definition of the retiming lemma [3] and knowledge of the constraints imposed by Chopin-2, a fast and efficient verification algorithm has been developed that precisely can diagnose and pinpoint problems in the circuit. The algorithms presented here have been implemented in a prototype tool called Sand.

1 Theoretical Background

1.1 Circuit representation

The circuits considered here are simple synchronous circuits consisting of functional building blocks and registers. The retiming algorithm is unconcerned with the complexity of the building blocks which can range from simple gates to an ALU, or still more complex. Only globally clocked, unconditional registers with one input and one output are allowed.

We represent this physical circuit by a directed graph $G(\mathcal{V}, \mathcal{E}, w)$. $\mathcal{V}$ is the set of vertices, and consists of subsets of different kinds of vertices: $\mathcal{V} = \mathcal{F} \cup \mathcal{I} \cup \mathcal{O}$. Each $f \in \mathcal{F}$ represents a functional element of the physical circuit, $i \in \mathcal{I}$ represents an input of the circuit and $o \in \mathcal{O}$ represents an output.

$\mathcal{E}$ is the set of directed edges. A directed edge $e : u \rightarrow v$, with $e \in \mathcal{E} \wedge u, v \in \mathcal{V}$ represents a connection in the circuit, the direction of the edge following the data-flow. The graph is not a direct representation of the circuit. The registers in the circuit are not represented by vertices. Instead, they are bypassed and the resulting edges get a register count value $w(e)$ equal to the number of registers the edge bypasses.

A path $p(u \Rightarrow v)$ is a sequence of vertices and edges $u = t_0 \stackrel{e_0}{\rightarrow} t_1 \stackrel{e_1}{\rightarrow} t_2 \cdots \stackrel{e_{n-1}}{\rightarrow} t_n = v$ connecting vertex u to vertex v. A path p is a simple cycle when its start and end point are the same vertex and all vertices in between are visited exactly once. This is written down as $p(v \Rightarrow v)$. For a path p, the path weight is then defined as:

$$w(p) = \sum_{e=0}^{n} w(e) \tag{1}$$

For a certain graph G to represent a physical circuit the following restrictions are necessary:

$$\forall e \in \mathcal{E} : w(e) \geq 0 \tag{2}$$

$$\forall v \in \mathcal{V} : \forall p(v \Rightarrow v) : w(p) > 0 \tag{3}$$

The first restriction reflects the fact that negative register counts are physically impossible. The second restriction states the fact that we only consider synchronous systems so cycles with path weight zero are disallowed.

To be able to verify the hardware selection aspects, the graph model has to be extended into a hierarchical model. In that case a functional building block modeled by $v \in \mathcal{F}$ is itself a graph $G'(\mathcal{V}', \mathcal{E}', w')$. This graph represents the internals of the functional building block. Here also, $\mathcal{V}' = \mathcal{F}' \cup \mathcal{I}' \cup \mathcal{O}'$. When $\mathcal{F}' = \emptyset$, we have reached the bottom level of the hierarchy. In that case we call v a *leaf* vertex.

In this model, an edge $e : u \rightarrow v$ with, for instance, $v \in \mathcal{F}$, can be written down more precisely as $e : u \rightarrow v'$ with $v' \in \mathcal{I}', \mathcal{I}' \subset \mathcal{V}'$ and $\mathcal{V}' \in G' = v$. In a hierarchical graph G with u and/or v subgraphs, the following edges are possible:

- $e : u \rightarrow v, u \in \mathcal{I} \wedge v \in \mathcal{O}$: in this case we have a through connection from an input of the circuit to an output, without passing through a functional block.
- $e : u \rightarrow v', u \in \mathcal{I} \wedge v' \in \mathcal{I}'$: this is the connection of an input of the circuit to an input of a functional block of the graph.
- $e : u' \rightarrow v, u' \in \mathcal{O}' \wedge v \in \mathcal{O}$: this is a connection of an output of a functional block to an output of the circuit
- $e : u' \rightarrow v', u' \in \mathcal{I}' \wedge v' \in \mathcal{O}''$: this is a connection from an output of a functional block to an input of another or the same functional block.

When dealing with algorithms considering only the first level of the hierarchy, the simpler non hierarchical graph model can be used and an edge $e : u \rightarrow v'$ is considered as $e : u \rightarrow v$.

1.2 Retiming

The retiming transformation optimises the clock period of a circuit by adding and/or removing registers. It can be viewed as a vertex labeling $r : \mathcal{V} \rightarrow Z$ that assigns to each vertex an integer valued lag r. A circuit $G(\mathcal{V}, \mathcal{E}, w)$ satisfying (2) and (3) is transformed into a circuit $G_r(\mathcal{V}, \mathcal{E}, w_r)$. Of course, G_r also has to satisfy (2) and (3). The only change in the graph G_r is in the edge weights w_r, since the transformation works by only changing the number and position of registers in the circuit. The edge weights of the retimed graph are given by

$$w_r(e) = w(e) + r(v) - r(u) \qquad (4)$$

It can be proven that this equation can be extended for a path $p(u \Rightarrow v)$:

$$w_r(p) = w(p) + r(v) - r(u) \qquad (5)$$

The retiming lemma proves that the circuit G_r obtained through retiming is functionally equivalent to the original circuit. This proof can be found in [4], and is the foundation of our verification approach.

1.3 Hardware selection

Hardware selection consists of choosing an appropriate functional building block among a set of functional equivalent ones. The elements of this set have the same interface description; they only differ in their internal implementation. Selection is done following criteria of speed and area: we need an implementation as small as possible and only as fast as needed. Alternative implementations can be obtained from a library, through logic synthesis, or by applying redundancy removal...An example would be the selection of the type of adder used in a design. The library can contain a simple ripple adder and a carry look-ahead adder. The carry look-ahead adder can be selected when a faster cycle time is needed.

2 Verification

Making abstraction of the hardware selection, the verification problem is: given a circuit $G'(\mathcal{V}', \mathcal{E}', w')$ that is claimed to be a retiming of $G(\mathcal{V}, \mathcal{E}, w)$, prove that this is indeed the case. The verification algorithm can be broken down in the following steps:

1. verify that G and G' are well-formed graphs representing physically possible circuits.
2. verify that G and G' are structurally the same, i.e. $G(\mathcal{V}, \mathcal{E}) \equiv G'(\mathcal{V}', \mathcal{E}')$
3. prove that there exists a retiming r such that equation (4) holds.

Hardware selection is a separate verification issue. For the retiming verification it becomes important when determining the structural equivalence between the two graphs. This will be discussed further on in the section on structural equivalence.

2.1 G and G' are valid circuits

The first step in the verification algorithm consists of building the hierarchical graphs of the circuits from their description in a hierarchical hardware description language. Both circuits are then checked for correct connectivity. While this is not necessary for verification of the retiming transformation, this step has proven itself useful as circuit problems of this nature are quite common. Given a hierarchical graph, the following connections are allowed:

- except for the top level graph, input vertices should have only one arriving edge. The top level input vertices have no arriving edges.
- except for the top level graph, output vertices can have zero or more departing edges. The top level output vertices have no departing edges.
- except for a leaf vertex, input vertices have zero or more departing edges. Input vertices of a leaf vertex have no departing edges.
- except for a leaf vertex output vertices have exactly one arriving edges. Output vertices of a leaf vertex have no arriving edges.

The next step involves removing the registers from the graphs and replacing them by weights on the resulting edges. This step is only performed at the top level of the hierarchy. Once well-formed graphs are constructed, we can verify that they satisfy (2) and (3). Constraint (2) is met by construction of the graph and assignment of the edge weights.

Equation (3) is verified using a slightly modified version of Johnson's algorithm [1] to generate all simple cycles of a graph. The algorithm works iteratively on a vertex ordered subgraph that starts out as the complete graph. Of this subgraph the strongly connected components (SCC's) are constructed, and the cycles of the SCC containing the first vertex of the subgraph are enumerated. For the next iteration, a new subgraph is constructed

by removing the first vertex, until there are no vertices left. Though this algorithm uses some clever tricks to minimise the work to be done, the worst case complexity is still very high due to the nature of the problem. However, since we are only interested in finding asynchronous loops, the edges with a strictly positive weight can be made invisible for the cycle-searching algorithm. This means that each cycle we find will be an asynchronous one, and in that case the circuit will be incorrect. This trick drastically improves run times in case the circuit is correct, since no cycles have to be enumerated. If an asynchronous cycle is found we can stop with the verification algorithm.

2.2 Structural equality

G' can only be a valid retiming of G if both graphs are structurally the same. This means that $\forall\ v \in \mathcal{V} : \exists!\ v' \in \mathcal{V}' : v \Leftrightarrow v'$ and $\forall\ e \in \mathcal{E} : \exists!\ e' \in \mathcal{E}' : e \equiv e'$. The sets of edges have to be the identical, the vertices only have to be functionally equivalent because of the possible hardware selection. The equivalence verification is done on the first level of hierarchy only. Given only the structure of G and G' and no additional information, in the general case more than one mapping can be found. This happens if the retiming and hardware selection tools are allowed to change names of the inputs, outputs and functional building blocks of the circuits, so we cannot rely on the name of a vertex $v \in \mathcal{V}$ to find the corresponding vertex in V'. We assume that the names of the vertices are available and not changed by the retiming tools. This is a reasonable assumption, and makes the equivalence checking algorithm very straightforward.

The hardware selection mechanism makes the equivalence check more complicated. We still assume that the name of the vertices have been kept, or that at least the replaced vertices are identifyable as such. We still have to verify then that the vertices, which are not identical, are functionally equivalent. This is the hardware verification problem in general: given two circuits $v = G_a(\mathcal{V}_a, \mathcal{E}_a)$ and $v' = G_b(\mathcal{V}_b, \mathcal{E}_b)$, prove that they are performing the same function. Various approaches and solutions to this problem exist. The candidates for hardware selection can be selected from a set of parametrised circuits which are formally proved to be functionally the same [8]. It is of course also possible to use the general purpose verification techniques already mentioned in the introduction.

2.3 Construction of retiming function r

The last step in the verification algorithm is to prove that there exists a retiming r such that equation (4) is met. Chopin-2 assumes all inputs of the circuit have a lag of zero, and all outputs have the same lag. Given these constraints, it's easy to see that the only possible r can be calculated using (5). For each $v \in \mathcal{F} \cup \mathcal{O}$ there exists an input $u \in \mathcal{I}$ such that there is a path $p(u \Rightarrow v)$. Using (5) and the knowledge that $r(u) = 0$ we can calculate

$$r(v) = w_r(p) - w(p). \tag{6}$$

A depth first search spanning tree [7] starting from the inputs of the circuit is constructed. This gives us a path to an input for each of the vertices, and (6) is easily evaluated. We can then verify that all the outputs have the same lag. The last thing to check is that (4) holds for all $e \in \mathcal{E}$. Note that for the edges in the spanning tree, (4) is met by construction.

If the constraints on lags of inputs and outputs are more relaxed, the verification gets more complicated. [2] shows a solution for that case.

3 Implementation and results

Chopin-2 uses the HILARICS [6] hardware description language as input and output format. We deal with the HILARICS descriptions through the procedural interface SPI [9], which gives us a language independent interface to structural hardware descriptions. Registers are recognised through a special attribute. This attribute is also used by Chopin-2 to recognise the registers.

Chopin-2 can do both word-level and bit-level retiming, which makes implementation of the algorithms more complicated, since the connections between the functional blocks can be bus connections. This results in multiple edges between vertices representing the same logical data flow. Coping with this obfuscates the simplicity of the algorithms and makes implementation tricky.

The program, Sand, is implemented in C++. Most attention has gone to a correct implementation with run time efficiency of only secondary importance, as no major efficiency problems were expected and experienced. General verification of the hardware selection has not been implemented yet. Chopin-2 chooses among a set of alternative implementations, coming from a library of parametrised hardware modules. As mentioned earlier, this library can and has been formally verified.

Table 1 shows some results. All examples were run on a DECstation 5000/120 with 16Mb memory. The size of the examples is expressed in the number of standard cells, before and after retiming and pipelining. The number of one bit registers in the original circuit is also indicated. The standard cells, from the MIETEC standard cell library, range in complexity from a simple AND cell to full-adder cells. The last two columns indicate the run time when full hierarchy is used, and when only the first level is considered.

circuit	#cells before	#cells after	#regs	full (s)	1st (s)
ex1	74	151	11	30	4
ex1flat	74	157	11	24	9
ex2	294	428	64	99	18
ex2flat	294	524	64	116	59
dirdet	502	1262	48	318	179
section	924	1104	224	369	306
sub_p_uv	914	1616	288	521	326

Table 1: Sand Results

ex1 is a word-level description of a simple circuit on which word-level retiming is used. The functional elements in this circuit are high level functional building blocks like 8-bit adders, multiplexers and comparators. The figures show that quite some time is spent in building the hierarchy and verifying the connections. *ex1flat* is the same circuit after flattening of the hierarchy and bus expansion. This circuit was retimed on the bit level, with the same cycle time as result. In theory the time needed for verifying the full hierarchy and only the first level should the same for this case, since *ex1flat* is fully flattened and contains no hierarchical information. The registers inserted by Chopin-2 however, are hierarchically modeled. Depending on how many registers are added by the retiming tool the difference in the run time figures for full or first level hierarchical verification will be more or less important (remember that the hierarchical connectivity checks are run before register removal). From the figures for *ex2*, a circuit processing 32 bit words, and *ex2flat*, the flattened version of *ex2*, similar conclusions can be drawn. The bit-level retiming of *ex2* resulted in a circuit twice as fast as the word-level retiming, but at the cost of more registers. The figures also show that the verification of a word-level retiming of a hierarchical circuit is about as fast to a bit slower than the verification of the bit level retiming of the flattened circuit. In Sand, the hierarchy is treated in a simple and straightforward way, and a more sophisticated approach would improve run times. The effect of the number of registers inserted by Chopin-2 would be much smaller then, and the verification of full hierarchical descriptions would become much faster. The other circuits, i.e. *dirdet*, *section* and *sub_p_uv* are described on the bit level and hierarchically flattened. The retiming of *sub_p_uv* was found to be faulty: on one of the outputs a register was missing, causing the outputs to have differential lags. This error has been corrected.

4 Conclusions

In this paper we showed how knowledge of the synthesis transformations, in this case retiming, can be used to develop efficient verification algorithms. They allow for fast and precise diagnosis of circuit problems, a feature most general purpose verification techniques lack. Possible improvements are extensions to allow for general verification of the hardware selection actions, and the capability to do buffering verification, as done in [2]. The way hierarchy is handled now is also open for considerable improvement in run time and memory requirements.

References

[1] D. B. Johnson. "Finding all the elementary circuits of a directed graph". *SIAM J. Comput.*, 4(1):77–84, Mar. 1975.

[2] A. Kostelijk and A. van der Werf. "Functional verification for retiming and rebuffering optimization". In *Proceedings of The European Conference on Design Automation*, pages 99–104. IEEE Computer Society Press, 1993.

[3] C. E. Leiserson, F. M. Rose, and J. B. Saxe. "Optimizing synchronous circuitry by retiming". In R. Bryant, editor, *3rd Caltech Conference on Very Large Scale Integration*, pages 87–116, 1983.

[4] C. E. Leiserson and J. B. Saxe. "Optimizing Synchronous systems". *Journal of VLSI and Computer Systems*, 1(1):41–67, Spring 1983.

[5] S. Note. *"Mapping high throughput signal processing algorithms into dedicated data-path architectures"*. PhD thesis, K.U. Leuven - Imec, Mar. 1991.

[6] R. Severyns and E. Willems. "HILARICS-2: The Language". IMEC Internal Report, Aug. 1990.

[7] R. Tarjan. "Depth-first search and linear graph algorithms". *SIAM J. Comput.*, 1(2):146–160, June 1972.

[8] D. Verkest, L. Claesen, and H. De Man. "On the use of the Boyer-Moore theorem prover for correctness proofs of parameterized hardware modules". In L. Claesen, editor, *Formal VLSI Specification And Synthesis*, pages 99–116. North-Holland Elsevier Science Publishers, 1990.

[9] P. D. Worm. "SPI version 2.51". IMEC Internal Report, Mar. 1992.

TAU'93

ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems

September 14-16, 1993
Intermar Hotel Malente
Germany