High-Level Optimization by Combining Retiming and Shannon Decomposition

Cristian Soviani Olivier Tardieu Stephen A. Edwards[*]
Department of Computer Science
Columbia University, New York

Abstract

Applying Shannon decomposition can reshape sequential circuits and improve opportunities for retiming. Both Shannon decomposition and retiming only rely on limited information about combinational blocks (timing estimates), so both techniques are suitable for high-level synthesis.

We describe an efficient algorithm to preprocess a circuit using Shannon decomposition to increase retiming efficacy. It assembles complex chains of Shannon decompositions while carefully avoiding parallel ones in order to limit the area overhead due to logic duplication.

We compare a traditional retiming flow with the same flow augmented with our algorithm. Although our algorithm provides no improvement on half of our benchmarks, for the other half we obtain a 25% speed-up on average (7% to 61%), while only increasing area by 5% (3% to 12%).

1 Introduction

IC technology has made it possible to build enormous systems on a chip. High-level synthesis and optimization techniques are needed to handle not only low-level entities such as gates but also more complex structures such as arithmetic units.

There are many techniques for low-level optimizations, including two-level optimization, algebraic methods, and redundancy elimination. But if we increase the level of abstraction, we can no longer use most of them. However, Shannon decomposition and retiming scale very well, as they ignore the complexity of the combinational blocks. In this work, we concentrate on these two techniques.

There are too many ways to combine these transformations, so heuristics are usually applied to choose good ones. Even worse, the circuit is usually optimized after each transformation (critical Boolean opportunities might otherwise be missed), meaning a systematic way of considering combinations of transformations is difficult. Any high-level approach must therefore ignore Boolean properties or account for them simply because of the lack of low-level information.

[*]soviani, tardieu, sedwards@cs.columbia.edu. Edwards and his group are supported by an NSF CAREER award, a grant from Intel corporation, an award from the SRC, and from New York State's NYSTAR program. http://www.cs.columbia.edu/˜sedwards

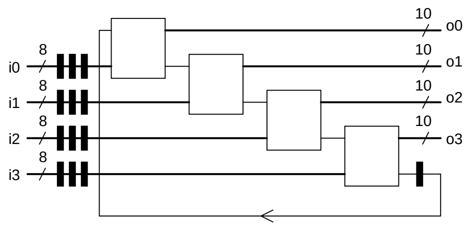


Figure 1: Motivating example: four slow combinational blocks and a tight feedback loop that cannot be retimed.

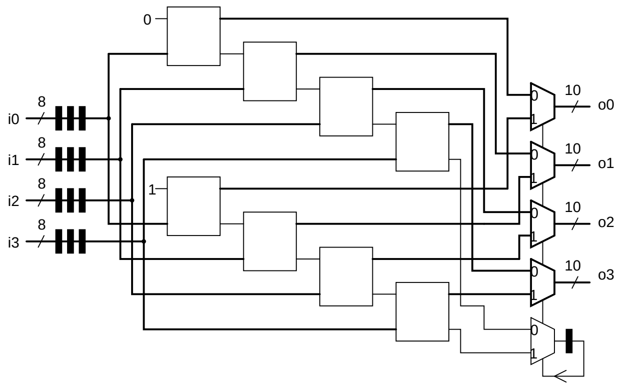


Figure 2: Shannon decomposition reduces the feedback loop to a single multiplexer delay.

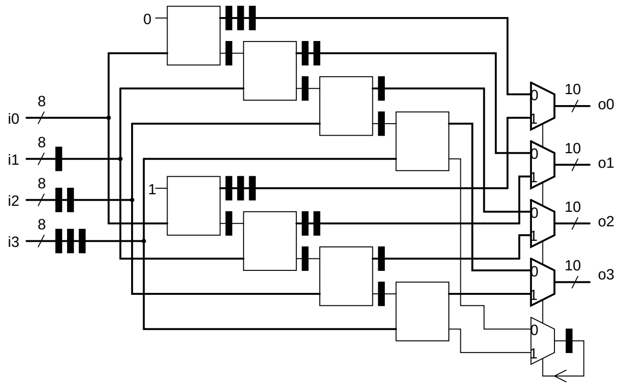


Figure 3: Retiming has reduced cycle time to one block's delay plus a multiplexer.

In this paper, we propose an efficient algorithm to select and apply Shannon decompositions that enable effective retiming while avoiding excessive area increase. In other words, by preprocessing a circuit using our algorithm just before retiming, the speed-up obtained by retiming is significantly improved. The algorithm handles both acyclic and (sequential) cyclic circuits. It is both efficient and exact, provided our timing estimation function and that of the retiming algorithm agree.

1.1 An example

Consider the circuit in Figure 1, which was extracted from a Gigabit Ethernet 8b10b encoder. The four identical combinational nodes are already optimized—they were designed manually for speed. We may even have several variants, exploiting a performance/area trade-off. Therefore, it would be desirable to speed up the circuit without modifying the nodes.

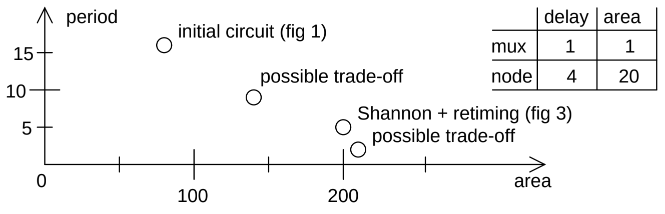
Retiming is a widely-used transformation. In our example, the designer put three registers on each input in the hopes that the increased latency would allow retiming to improve throughput, otherwise known as pipelining. Unfortunately, retiming fails on this circuit because of the tight (single-register) feedback loop. If d_{node} is the delay of the combinational node, the minimum period remains $4d_{\text{node}}$ after retiming.

Figure 2 shows how the loop can be broken using Shannon decomposition. The combinational nodes are duplicated and some muxes added, resulting in a significant area penalty. This is not so severe in practice, as usually only a small fraction of a circuit will require Shannon decomposition. The longest path is now $4d_{\text{node}} + d_{\text{mux}}$, where d_{mux} is the delay of a mux. But the loop is much shorter—its delay is d_{mux}—enabling retiming to achieve a period of $d_{\text{node}} + d_{\text{mux}}$ (Figure 3).

In fact, we can generate multiple points on the period/area curve (Figure 4). Adding more than three registers at the inputs makes it possible to further reduce the minimum period. Although impractical beyond a certain limit, we can theoretically decrease the minimum period until it reaches the delay of the loop: d_{mux}.

1.2 Related work

Speeding up combinational logic by resynthesizing the minimum slack paths has a long history. Relevant techniques include tree-height reduction (THR, Singh et al. [12]), the generalized select transform (GST, Berman et al. [1]), the generalized bypass transform (GBX, McGeer et al. [8]), and exact sensitization of critical paths (Saldanha et al. [9]). Like ours, GST is based on Shannon decomposition.



	delay	area
mux	1	1
node	4	20

Figure 4: Period/area trade-off.

Speeding up sequential logic has also been the focus of extensive research, such as the work of Singh [11]. Retiming (Leiserson and Saxe [5]), can decrease the minimum period without restructuring the logic. Malik et al. [6] combines retiming and resynthesis. The algorithm is efficient for circuits with little feedback, such as pipelines. In contrast, our combination of retiming and resynthesis focuses on feedback.

Addressing high-level synthesis, Hassoun et al. [4] proposes architectural retiming, which attempts to optimize high-level pipelines by mixing retiming with pre-computation and prediction. Marinescu et al. [7] proposes an algorithm to automatically pipeline a circuit by increasing the pipeline length with the help of stalling and forwarding. But several transformations applied successively may interfere with each other. As a result, most approaches apply various transformations iteratively, in a heuristic manner. By contrast, in this work we take all retiming opportunities into account when choosing Shannon decompositions, so only a single pass of our algorithm followed by a single pass of retiming is needed.

1.3 Paper organization

Section 2 introduces the notation we use and reviews Shannon decomposition, retiming, and retiming efficiency estimation. Section 3 describes serial compositions of Shannon decompositions. Section 4 presents the exhaustive exploration algorithm for combinational circuits. Section 5 sketches the extension of this algorithm to sequential circuits. In Section 6, we describe our implementation and discuss experimental results. We conclude in Section 7.

2 Basics

2.1 Sequential circuits

A sequential circuit is a directed graph $S = (V, E)$ with vertices $V = PI \cup PO \cup N \cup R \cup \{\text{spi}, \text{spo}\}$. PI & PO are the primary inputs & outputs; N are the internal combinational nodes; R are registers; spi and spo are two supernodes connected to/from all PI/PO respectively. The edges $E \subset V \times V$ model the interconnect: $\text{fanin}(n) = \{n' | (n', n) \in E\}$, $\text{fanout}(n) = \{n' | (n, n') \in E\}$.

Each combinational node $n \in N$ computes a Boolean function of its p input wires $f : \mathbb{B}^p \to \mathbb{B}$ which defines the common value of all its output wires. $\forall r \in R : |\text{fanin}(r)| = 1$.

Combinational cycles are not allowed: the subgraph of S excluding registers $D = (V \setminus R, E|_{V \setminus R \times V \setminus R})$ must be acyclic.

2.2 Arrival times

We define weights $d : V \to \mathbb{R}$ as follows:

$$d(n) = \begin{cases} \text{arrival time (from clock)} & n \in PI \\ \text{delay of logic} & n \in N \\ \text{setup time (to next clock)} & n \in PO \\ 0 & n \in R \cup \{\text{spi}, \text{spo}\} \end{cases} \qquad (1)$$

```
procedure ArrivalTimes($S$)
    for each $n \in R \cup \{\text{spi}\}$ do $\{\text{at}(n) \leftarrow 0\}$
    for each $n \in V \setminus (R \cup \{\text{spi}\})$ in topological order in $D$ do
        $\text{at}(n) \leftarrow d(n) + \max_{n' \in \text{fanin}(n)} \text{at}(n')$
```

Figure 5: Algorithm to compute arrival times.

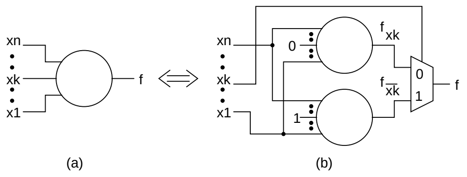


Figure 6: Shannon decomposition of f on input x_k.

For each node $n \in V \setminus (R \cup \{\text{spi}\})$, we have an arrival time:

$$\text{at}(n) = d(n) + \max_{n' \in \text{fanin}(n)} \text{at}(n') \qquad (2)$$

Since D is acyclic, computing arrival times is straightforward (Figure 5). The minimum feasible cycle period for S is

$$\max \left\{ \text{at}(n), n \in \{\text{spo}\} \cup \bigcup_{r \in R} \text{fanin}(r) \right\}.$$

2.3 Shannon decomposition

Let $f : \mathbb{B}^p \to \mathbb{B}$ be the Boolean function of the combinational node n and $1 \le k \le p$. Then

$$f(x_1, x_2, \ldots, x_p) = x_k f_{x_k} + \overline{x_k} f_{\overline{x_k}}$$

$$\text{where} \quad \begin{aligned} f_{x_k} &= f(x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_p) \\ f_{\overline{x_k}} &= f(x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_p) \end{aligned}$$

Such a Shannon decomposition suggests an alternate implementation of the node (Figure 6) with arrival time

$$\text{at}(n) = \max \left\{ \text{at}(f_{\overline{x_k}}) + d_{\text{mux0}}, \text{at}(f_{x_k}) + d_{\text{mux1}}, \text{at}(x_k) + d_{\text{muxs}} \right\}$$

For simplicity, we assume $d_{\text{mux0}} = d_{\text{mux1}} = d_{\text{muxs}} = d_{\text{mux}}$, so

$$\text{at}(n) = \max \left\{ \text{at}(f_{\overline{x_k}}), \text{at}(f_{x_k}), \text{at}(x_k) \right\} + d_{\text{mux}}$$

Such a transformation, therefore, improves $\text{at}(n)$ provided that x_k arrives later than the other inputs x_i $(i \ne k)$. The cost comes in an area increase from node duplication.

2.4 Retiming

Retiming follows from observing that moving registers back or forth in a sequential circuit preserves its functionality (Figure 7). Its goal is to move registers to decrease long (critical) combinational paths at the expense of short (non-critical) ones.

Let $\text{ret}(S)$ be the minimum period achievable by retiming a circuit S. Retiming cannot decrease the period of a cycle. If d_c and r_c are the combinational delay and the number of registers of the cycle c in S, $\text{ret}(S) \ge d_c/r_c$. Similarly, if p is a path from spi to spo having r_p registers and of combinational delay d_p, $ret(S) \ge d_p/(r_p+1)$. Thus, $\text{ret}(S) \ge \text{lb}(S)$ where

$$\text{lb}(S) = \max \left(\max_{c \in \text{cycles}(S)} \frac{d_c}{r_c} \quad , \quad \max_{p \in \text{paths}(S, \text{spi}, \text{spo})} \frac{d_p}{r_p + 1} \right) \qquad (3)$$

For the example in Figure 8(a), $\text{lb}(S) = (3+7+2)/3 = 4$.

In addition to moving registers past nodes (as in classical retiming, Figure 8b), achieving the period $\text{lb}(S)$ requires moving registers inside nodes (Figure 8c). Large combinational blocks built from small gates, such as those in an FPGA [13], can usually be modified this way. We therefore assume $\text{ret}(S) = \text{lb}(S)$. In the sequel, we focus on transforming S to minimize $\text{lb}(S)$.

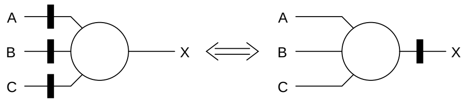


Figure 7: Basic retiming.

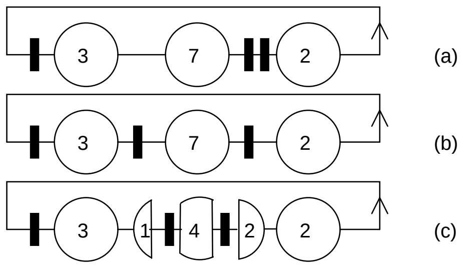


Figure 8: Decreasing cycle period through retiming. (a) initial cycle with period 10, (b) retiming preserving nodes: period 7, (c) retiming splitting nodes: period 4.

2.5 Retiming efficiency estimation

The number of cycles can be exponential in the size of the circuit, so computing $\text{lb}(S)$ directly with (3) is not practical.

Assign weight $-c$ to the registers: $\forall r \in R : d(r) = -c$, where c is the desired period. Every other node keeps its weight d as defined by (1). Then there exists a retiming for period c iff $\text{at}(\text{spo}) \le c$ and the graph S has no positive cycles. In our example, $c = 3$ gives the cycle weight 3; $c = 5$ gives -3. Not surprisingly, for $c = \text{lb}(S) = 4$, the cycle has weight 0.

The Bellman-Ford algorithm [3] (Figure 9) detects positive cycles. The algorithm terminates after at most $|V| - 1$ iterations iff there exists no positive cycle. Therefore, $\text{lb}(S)$ can be approximated by binary search on the period c.

3 Variants

Consider building many variants of a combinational node in parallel (i.e., with identically-connected inputs). The fanouts of a node may then choose to connect to any of these variant's outputs without affecting the circuit's function. However,

```
procedure BellmanFord($S$)
    at(spi) $\leftarrow 0$
    for each $n \in V \setminus \{\text{spi}\}$ do $\{\text{at}(n) \leftarrow -\infty\}$
    repeat
        changes $\leftarrow$ false
        for each $n \in V \setminus \{\text{spi}\}$ do
            if RelaxNode($n$) then {changes $\leftarrow$ true}
    until not changes

procedure RelaxNode($n$)
    at$_{new} \leftarrow d(n) + \max_{n' \in \text{fanin}(n)} \text{at}(n')$
    if at$_{new} \neq$ at$(n)$ then
        at$(n) \leftarrow$ at$_{new}$
        return true
    else
        return false
```

Figure 9: The Bellman-Ford algorithm for calculating positive cycle weights.

if our only goal is the fastest circuit, only the variant with minimum arrival time is interesting; we may ignore the others.

However, we may also want to consider more complex variants that instead of a single output wire w, (redundantly) encode it as a series of wires $(v_i)_{i \in I}$, for instance as $w = \overline{v_s} v_0 + v_s v_1$. We say that (v_s, v_0, v_1) is a virtual wire of type sh where

$$sh : v_s, v_0, v_1 \mapsto \overline{v_s} v_0 + v_s v_1.$$

3.1 Variant types

In general, a type is a function $t : \mathbb{B}^p \to \mathbb{B}$ that decodes a virtual wire to give its true value. The type of a real wire is $id : w \mapsto w$.

Providing a variant with an output of type t for a node of function f means designing f' such that $t \circ f' = f$. Computing f' instead of f may be seen as a speculative computation that we will later complete using function t.

Such variants are interesting because the arrival times for the several components of a virtual wire may be different, thus giving further opportunities to optimize speed.

To this aim, we need to chain variants. In addition to virtual output wires, we support virtual input wires. We say that f' is a variant of f of type $t_1 \times \cdots \times t_n \to t$ iff

$$t \circ f'(w_1, \ldots, w_n) = f(t_1(w_1), \ldots, t_n(w_n)) \quad (\forall i : w_i \text{ has type } t_i)$$

In other words, f' is such a variant of f iff f', provided with virtual wires of types $t_1, \ldots, t_n$, computes a virtual wire of type t so as to match the computation of f for the corresponding real wires (Figure 10).

3.2 Shannon variants

In principle, we can generate arbitrarily complex variants; we can even consider the collapsed input cone of a node as a variant, deferring all computation to the type function. In this paper, we only consider the variants generated by nested Shannon decompositions, which we now define.

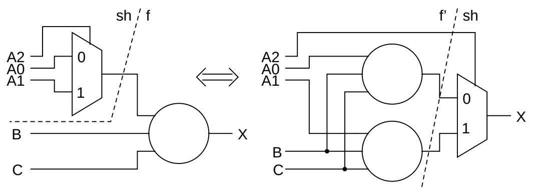


Figure 10: f' is a variant of f of type $sh \times id \times id \to sh$.

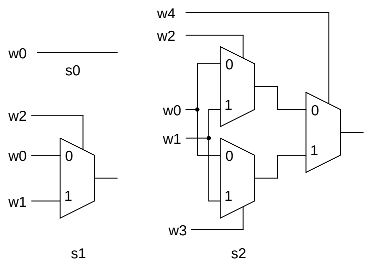


Figure 11: Virtual wire types s_0, s_1, and s_2.

First, we restrict ourselves to the set of wire types $\{s_k\}_{k \geq 0}$ (Figure 11), where $s_k : \mathbb{B}^{2k+1} \to \mathbb{B}$ are recursively defined as

$$s_0 : v_0 \mapsto v_0 \qquad \forall k \geq 0, s_{k+1} = s_k \circ r_k,$$

where $\{r_k\}_{k \geq 0}$ are the functions $r_k : \mathbb{B}^{2k+3} \to \mathbb{B}^{2k+1}$ such that

$$\begin{aligned} r_0 &: v_0, v_1, v_2 \mapsto \overline{v_2} v_0 + v_2 v_1 \\ r_{k+1} &: v_0, \ldots, v_{2k+4} \mapsto (\overline{v_2} v_0 + v_2 v_1, \overline{v_3} v_0 + v_3 v_1, v_4, \ldots, v_{2k+4}). \end{aligned}$$

Intuitively, the types $\{s_k\}_{k \geq 0}$ describe a series of Shannon decompositions, each r_k function specifying one such decomposition: $s_k = s_0 \circ r_0 \circ r_1 \circ \cdots \circ r_{k-1}$ (Figure 12).

To limit the number of node copies, we only allow one non-real input wire on a node. That is, we only consider variant types in the set $\{s_{in} \times s_0 \times \cdots \times s_0 \to s_{out}\}_{in \geq 0, out \geq 0}$ (modulo permutation of the input wire types).

We only use a few variants of such types. Basically, we restrict variants to make at most one Shannon decomposition. As a result, $out \leq in + 1$. These variants can be seen as series combinations. Combining Shannon decompositions in parallel would require a node to be copied four times or more times, which we consider impractically costly.

Formally, consider the function $f : \mathbb{B}^p \to \mathbb{B}$ of node n. In Table 1, we first define the sets of primitive variants $\{start_k^f\}_{k \geq 0}$ and $\{extend_k^f\}_{k \geq 1}$, which start and extend Shannon decompositions (Figure 13).

Starting from these primitive variants, we define the set $Sh(n)$ of all Shannon variants for the function f of node n

$$\left\{ \begin{array}{ll} f & \\ start_k^f & \forall k \geq 0 \\ extend_{k+1}^f & \forall k \geq 0 \\ r_{k-\ell} \circ \cdots \circ r_{k-1} \circ r_k \circ start_k^f & \forall k \geq 0, \forall \ell \geq 0 \text{ s.t. } \ell \leq k \\ r_{k-\ell} \circ \cdots \circ r_{k-1} \circ r_k \circ extend_{k+1}^f & \forall k \geq 0, \forall \ell \geq 0 \text{ s.t. } \ell \leq k \end{array} \right\}$$

$$\begin{array}{rrcl}
\forall k \geq 0, & start_k^f & : & s_k \times s_0^{p-1} \to s_{k+1} \\
 & & & w_1 = (v_0, \ldots, v_{2k}), w_2, \ldots, w_p \mapsto f(0, w_2, \ldots, w_p), f(1, w_2, \ldots, w_p), v_0, \ldots, v_{2k} \\
 & extend_{k+1}^f & : & s_{k+1} \times s_0^{p-1} \to s_{s+1} \\
 & & & w_1 = (v_0, \ldots, v_{2k+2}), w_2, \ldots, w_p \mapsto f(v_0, w_2, \ldots, w_p), f(v_1, w_2, \ldots, w_p), v_2, \ldots, v_{2k+2}
\end{array}$$

Table 1: Primitive Shannon variants for $f : \mathbb{B}^p \to \mathbb{B}$.

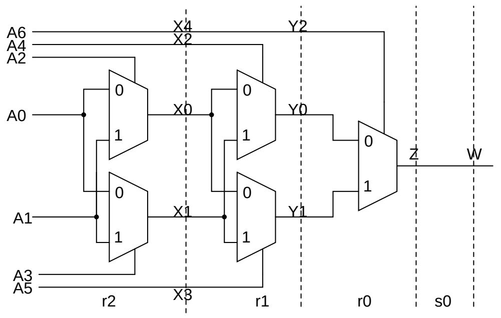


Figure 12: $s_3 = s_0 \circ r_0 \circ r_1 \circ r_2$.

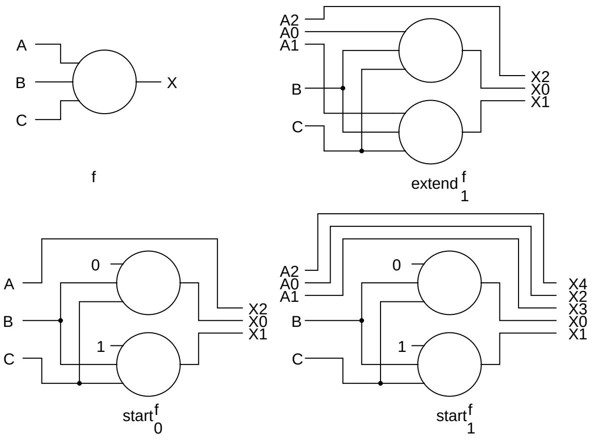


Figure 13: Examples of primitive Shannon variants.

Non-primitive variants are obtained by appending, to primitive variants, chains of multiplexers $r_{k-\ell} \circ \cdots \circ r_k$ that partially recombine the virtual output wires of primitive variants to complete Shannon decompositions started previously.

3.3 Circuit variants and arrival times

A variant S' of the circuit S is obtained by consistently replacing the combinational nodes of S by one or several variants of these nodes and replicating registers accordingly (to latch virtual wires). The obvious constraint is that two nodes connected by a wire in S' must agree on its type.

Circuit variants are still circuits. Node variants contain several atomic combinational nodes in the sense of Section 2.1 (copies of the initial combinational node, multiplexers) and compute $2k+1$ Boolean functions at once, forming a single virtual output wire of type s_k ($k \geq 0$).

Therefore, arrival times in circuit variants can be computed as described in Section 2.2, but since we are now interested in the arrival times of macro nodes rather than atomic nodes, we choose to denote the arrival times of node variants as tuples. For instance, consider a node variant n with output type s_2 and virtual output wire $(v_0, v_1, v_2, v_3, v_4)$. For simplicity, we assume that paired virtual output wires, such as (v_0, v_1) or (v_2, v_3), have the same arrival time[1]. Therefore, if $\text{at}(n_0) = \text{at}(n_1) = 3$, $\text{at}(n_2) = \text{at}(n_3) = 6$, and $\text{at}(n_4) = 10$, where n_i is the atomic node computing v_i, we write the arrival time of this variant as the 3-tuple $\text{at}(n) = (3, 6, 10)$.

4 Combinational circuits

In this section and the next, we describe how to efficiently and systematically build and analyze circuit variants in order to design variants tailored for retiming—variants that maximize retiming efficiency. Ideally, we would like to find a variant S' of the circuit S such that $\text{lb}(S')$ is minimum (cf. Section 2.4). As a secondary goal, we want to select variants that minimize node duplications (area).

To start with, we focus on combinational circuits ($R = \emptyset$), meaning that we are simply looking for a circuit with minimum cycle period (at(spo) minimum).

4.1 Overview

Since S is acyclic, we can build circuit variants by processing combinational nodes in topological order. Assume we have built a circuit variant S' to node $(n-1)$. There are many choices for implementing n. First, we have to decide which variants of $n' \in \text{fanin}(n)$ shall drive n. Second, we have to choose the variant of n itself. By design, the type constraints of Section 3.3 limit the number of choices. We denote the set of possible extensions by S'_n.

How to choose among node variants? It is not clear which variant is better. Each fanout may have some special advantage in using a specific type of virtual output wire that compensates for a late arrival time (a well-understood issue in technology mapping). As a result, we must consider several variants for each node during this construction, and only select optimal variant(s) for each node at the end.

This suggests three phases. Through a topological traversal of the circuit, we first compute the "feasible arrival times" for the node n in any variant of the initial circuit: $\text{fat}(n)$. In particular, fat(spo) will contain the minimum feasible cycle period for whole circuit. Then, with a reverse topological traversal, we

[1]We do not apply constant propagation to Shannon variants.

extract from these sets one or several "required arrival times" for each node n: $\mathrm{thin}(n) \subseteq \mathrm{fat}(n)$ that express feasible local requirements (i.e., per-node requirements), which, if locally met by an appropriate choice of node variant(s), guarantee a minimum cycle period for the whole circuit. Finally, we choose and wire node variants accordingly to produce a circuit with minimum cycle period.

4.2 Feasible arrival times

For any circuit S, $\mathrm{fat}(n)$ can be directly computed from the delay $d(n)$ of the node n in S and the feasible arrival times of the nodes in its fanin:

$$\mathrm{fat}(n) = \mathrm{combine}(d(n), \{\mathrm{fat}(n')\}_{n' \in \mathrm{fanin}(n)}) \tag{4}$$

Intuitively, since the arrival time of a node reveals its type (tuple size), the feasible arrival times of the nodes in $\mathrm{fanin}(n)$ carry enough information to decide whether a given Shannon variant of n can be used and how to wire it. Then, for each possible choice of a variant of n (including the choice of its wiring), we can compute its arrival time by applying (2) to each of its inner atomic combinational nodes. For lack of space, we do not formally define the combine function here.

As a result, the algorithm for the computation of feasible arrival times for a combinational circuit (Figure 14) is similar to the algorithm for computing arrival times (Figure 5). The key differences are that feasible arrival times uses (4) instead of (2) and includes a pruning operation, described below.

Although finite, the sets $\mathrm{fat}(n)$ can be very large. Therefore, we prune them (remove irrelevant values) on the fly.

Intuitively, the extension $q \in S'_n$ can be safely discarded iff, regardless of the following circuitry, there exists an extension $p \in S'_n$ that guarantees a better overall cycle period (our main goal). As a result, we can remove certain elements from $\mathrm{fat}(n)$ without fear of producing an inferior circuit. For instance, if p has arrival time (6) and q has arrival time (8) then q can be safely removed from S'_n, thus (8) from $\mathrm{fat}(n)$, without putting our construction at risk.

Let $\preceq$ be a partial order on arrival times such that if $p, q \in S'_n$, $p \neq q$ and $\mathrm{at}(p) \preceq \mathrm{at}(q)$, then q can be safely removed from S'_n. We can exhaustively discard non-minimal feasible arrival times using a pruning algorithm (Figure 14).

A good $\preceq$ relation lets us prune $\mathrm{fat}(n)$ aggressively. Furthermore, a suboptimal relation will not affect the optimality of the final circuit, only the running time of the algorithm.

We choose a simple but effective $\preceq$ relation:

$$(p_0, \ldots, p_i) \preceq (q_0, \ldots, q_j) \text{ iff } (i \leq j) \wedge (\forall k \leq i, p_k \leq q_k)$$

For instance, $(2) \preceq (4) \preceq (4,5) \preceq (4,6) \preceq (4,6,7)$. Because we know exactly how virtual wires can be recombined using chains of multiplexers, we are able to compare variants with different output types as in $(4,6) \preceq (4,6,7)$.

A pruned set always contains exactly one singleton, which corresponds to a node variant of n having a real output wire.

```
procedure FeasibleArrivalTimes($S$)
    $\mathrm{fat}(\mathrm{spi}) \leftarrow \{(0)\}$
    for each $n \in V$ in topological order do
        $\mathrm{fat}(n) \leftarrow \mathrm{Prune}(combine(d(n), \{\mathrm{fat}(n')\}_{n' \in \mathrm{fanin}(n)}))$
procedure Prune($X$)
    while there exist $p, q \in X$ such that $p \neq q$ and $p \preceq q$ do
        $X \leftarrow X \setminus \{q\}$
```

Figure 14: Feasible arrival times and pruning.

We write this arrival time $\mathrm{opt}(n)$. By construction,

$$\mathrm{opt}(\mathrm{spo}) = \min_{S' \text{ variant of } S} \left\{\mathrm{lb}(S')\right\}$$

4.3 Required arrival times

By construction, all arrival times in $\mathrm{fat}(n)$ are feasible through a appropriate choice of variants for the nodes $k \leq n$. Intuitively however, in order to obtain the minimum cycle period, a circuit variant only needs to achieve some of the arrival times in $\mathrm{fat}(n)$ for each node n. First, we rely on partial pruning. Second, circuits fragments we initially considered in the computation of feasible arrival times may end up not being fast enough to be part of an optimal circuit. Third, the minimum cycle period may admit several implementations.

We traverse the circuit again to select required arrival times $\mathrm{thin}(n) \subseteq \mathrm{fat}(n)$ for each node n. Since several variants of the same node may be required to produce alternate encodings of a given node output (needed by subsequent nodes in the circuit), we may end up with more than one required arrival time per node, hence the need for sets. But in our experiments we found a single variant was almost always sufficient.

Required arrival times must comply with two constraints. First, $\mathrm{thin}(\mathrm{spo}) = \{\mathrm{opt}(\mathrm{spo})\}$. Second, if a circuit variant has been built to node $(n-1)$ to achieve the arrival times $\{\mathrm{thin}(k)\}_{k<n}$, then it should be possible to extend it by a proper choice of variants for the node n so as to provide the arrival times $\mathrm{thin}(n)$ for the node n.

We perform the construction—essentially a pruning operation—through a reverse traversal of the circuit, starting from node spo. In general, there are several choices for the $\mathrm{thin}(n)$ sets corresponding to several implementations of the minimum cycle period. We have a crude heuristic (a partial order on the pruning) that attempts to minimize node duplications. We omit its description for lack of space.

4.4 Circuit construction

We can now build a circuit variant with minimum cycle period. Starting from node spi, we select and wire one or several variants for each node n so as to achieve the arrival times in $\mathrm{thin}(n)$. By definition of required arrival times, the resulting circuit achieves the minimum cycle period.

```
procedure FATBellmanFord($S$)
    fat(spi) $\leftarrow \{(0)\}$
    for each $n \in V \setminus \{\text{spi}\}$ do $\{\text{fat}(n) \leftarrow \{(-\infty)\}\}$
    repeat
        changes $\leftarrow$ false
        for each $n \in V \setminus \{\text{spi}\}$ do
            if FATRelaxNode($n$) then $\{\text{changes} \leftarrow \text{true}\}$
    until not changes

procedure FATRelaxNode($n$)
    $\text{fat}_{new} \leftarrow \text{Prune}(\text{combine}(d(n), \{\text{fat}(n')\}_{n' \in \text{fanin}(n)}))$
    if $\text{fat}_{new} \neq \text{fat}(n)$ then
        $\text{fat}(n) \leftarrow \text{fat}_{new}$
        return true
    else
        return false
```

Figure 15: Bellman-Ford for feasible arrival times.

5 Sequential circuits

Let S be a sequential circuit. While we could restrict the input and output types of register nodes in variants of S to be s_0 (real wires), we see no reason to impose such a limitation. As a result, a register node in a variant consists in general of several atomic registers, each of them latching a single element of a virtual wire. Experimental results show that the number of atomic registers grows typically as fast as the area.

Since S may contain cycles, we can no longer obtain feasible arrival times using a one-pass transversal of the circuit (Figure 14); we have to iterate. As in Section 2.5, we assign delay $-c$ to registers and use a modified Bellman-Ford algorithm (Figure 15) to compute the fat sets. We say the computation succeeds iff it terminates and $\text{opt}(\text{spo}) \leq c$.

We rely on the following result: the computation succeeds for period c iff there exists a variant S' of S such that $\text{lb}(S') \leq c$.

Indeed, by definition of feasible arrival times, if the computation terminates then there exists a variant S' of S such that $\text{lb}(S') \leq \max\{c, \text{opt}(\text{spo})\}$. The converse is less obvious. A circuit always admits slow variants ($\text{lb}(S') > c$). As a result, pruning becomes mandatory to guarantee termination if a fast variant exists ($\text{lb}(S') \leq c$). In other words, the choice of the pruning relation is no longer a matter of optimization, but the correctness of the whole procedure depends on the pruning. We believe we have designed an appropriate pruning relation, but we do not have a formal proof of this.

Although we have not yet obtained a theoretical bound on the size of the fat sets or on the number of iterations required to converge, the numbers remain small even on large examples.

As in the combinational case, we extract required arrival times from feasible arrival times to choose wire and node variants that give a circuit variant S' such that $\text{lb}(S') \leq c$. To achieve this period, we apply retiming to S', which may require pipelining combinational nodes (Section 2.4).

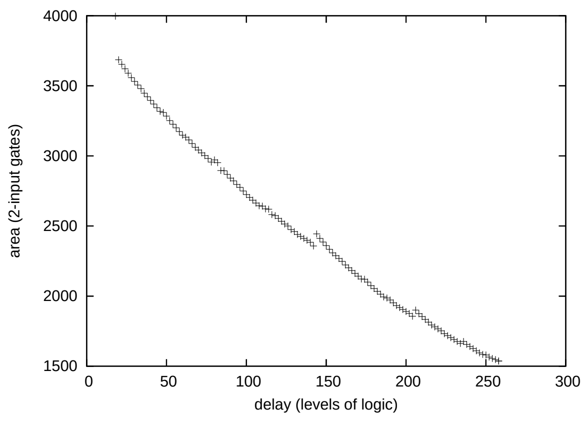


Figure 16: Delay/area trade-off for a 128-bit adder.

In summary, given a candidate period for retiming c, we can decide whether c can be achieved by a retimed Shannon variant of the initial circuit and, if so, produce such a circuit. In addition, we can approximate the minimum period achievable by a retimed variant of S by a binary search.

6 Experiments

We implemented our algorithm for acyclic and cyclic circuits in C++, using SIS libraries [10] to handle BLIF files. Our testing platform is a 2.5 GHz P4 with 512MB running Linux.

We estimate delays by simply counting levels of logic. Though imprecise, this is a widely-used estimate.

6.1 Combinational case: adders

As a quick correctness and performance check, we ran the algorithm on a variety of ripple-carry adders ranging from four to 1024 bits. The algorithm successfully transforms each sample into a $O(\log(n))$-delay carry-select adder.

We have further extended the algorithm for the combinational case to enable a trade-off between delay and area. For the 128-bit adder, we vary the required delay so as to measure the efficiency of our area minimization heuristics (Section 4.3). The values in Figure 16 were measured after a final two-input decomposition in SIS. All 120 points were computed in 88 s, of which our algorithm takes only 24 s.

6.2 ISCAS89 sequential benchmarks

For the sequential case, we first select an approximately optimal period by binary search. We start from a candidate period equal to half the delay of the critical path and stop when we have obtained an unfeasible period c_u and a feasible period c_f such that $c_f < c_u + 1/2$. We then build a circuit variant for period c_f.

Because they are widely available, we considered mid-sized ISCAS89 sequential benchmarks. We target an FPGA-like, three-input lookup-table architecture. Hence, we report delay and area as levels and numbers of lookup tables.

	Reference		Retimed		Shannon-R		Time	Speed-	Area
	period	area	period	area	period	area	(s)	up	penalty
s510	8	184	8	184	8	184	0.5		
s641	11	115	11	115	9	122	1.1	22%	6%
s713	11	118	11	118	10	121	0.9	10%	3%
s820	7	206	7	206	7	206	0.5		
s832	7	217	7	217	7	217	0.4		
s838	10	154	10	154	8	162	2.6	25%	5%
s1196	9	365	9	365	9	365	0.6		
s1423	24	408	21	408	13	460	3.8	61%	12%
s1488	6	453	6	453	6	453	0.7		
s1494	6	456	6	456	6	456	0.8		
s9234	11	662	8	656	8	684	6.7		
s13207	14	1382	11	1356	9	1416	18.0	22%	4%
s38417	14	7706	14	7652	13	7871	113.0	7%	3%

Table 2: Results on ISCAS89 sequential benchmarks.

Following Saldanha et al. [9], for each sample, we first run *script.rugged* and perform a speed-oriented decomposition *decomp -g; eliminate -1; sweep; speed_up -i*. We then reduce the depth of the circuit while keeping the nodes three-feasible using *reduce_depth -f 3* [14]. We consider the above flow a classical FPGA delay-oriented one. The results are reported in Table 2 under "Reference."

Starting from these optimized circuits, we either directly execute retiming (*retime -n -i*, modified to use the unit delay model) as reported in column "Retimed," or run our algorithm followed by retiming ("Shannon-R"). We verified our algorithm produced functionally-correct circuits by comparing them with the originals using VIS [2].

Although we produced no improvement on half of the samples, we realize a significant speed-up for the other half with only a 5% area increase on average. The algorithm is very fast. In particular, if no improvement can be made, its running time is negligible. Otherwise, it appears linear in the circuit size. The memory requirement is low (e.g., 70 MB for our largest circuit, s38417). Hence, our technique seems to scale well.

7 Conclusions

In this paper, we propose an algorithm that applies Shannon decomposition to enhance retiming opportunities on circuits with tight sequential feedback loops. Provided with an initial circuit and a desired period for this circuit, it tries to identify a series of Shannon decompositions that would make the period achievable through retiming. We approximate the best feasible period with a binary search.

A carefully-designed set of Shannon variants bounds the area penalty. We further reduce area with additional heuristics.

Our technique is sound. If the algorithm produces a circuit, then the target period can be achieved by retiming, provided the combinational nodes can be arbitrarily pipelined.

While we have not yet proved completeness (i.e., if the period is feasible then the algorithm achieves it), experimental results show significant improvements.

References

[1] C. L. Berman, D. J. Hathaway, A. S. LaPaugh, and L. Trevillyan. Efficient techniques for timing correction. In *Proc. ISCAS*, pages 415–419, 1990.

[2] R. K. Brayton, G. D. Hachtel, A. L. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa. VIS: a system for verification and synthesis. In *Proc. CAV*, pages 428–432, 1996.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. The Bellman-Ford algorithm. In *Introduction to Algorithms*, pages 588–591. Prentice Hall, 2002.

[4] Soha Hassoun and Carl Ebeling. Architectural retiming: pipelining latency-constrained circuits. In *Proc. DAC*, pages 708–713, 1996.

[5] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[6] S. Malik, E. M. Sentovich, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. *IEEE Transactions on CAD*, 10(1):74–84, 1991.

[7] Maria-Cristina V. Marinescu and Martin Rinard. High-level automatic pipelining for sequential circuits. In *Proc. ISSS*, pages 215–220, 2001.

[8] Patrick C. McGeer, Robert K. Brayton, Alberto L. Sangiovanni-Vincentelli, and Sartaj K. Sahni. Performance enhancement through the generalized bypass transform. In *Proc. ICCAD*, pages 184–187, 1991.

[9] Alexander Saldanha, Heather Harkness, Patrick C. McGeer, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Performance optimization using exact sensitization. In *Proc. DAC*, pages 425–429, 1994.

[10] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, UCB/ERL M92/41, 1992.

[11] K. J. Singh. *Performance optimization of digital circuits*. PhD thesis, UCB, 1992.

[12] Kanwar J. Singh, Albert R. Wang, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Timing optimization of combinational logic. In *Proc. ICCAD*, pages 282–285, 1988.

[13] H. Touati, N. Shenoy, and A. L. Sangiovanni-Vincentelli. Retiming for table-lookup field-programmable gate arrays. In *Proc. ACM/SIGDA international Workshop on Field Programmable Gate Arrays*, pages 89–93, 1992.

[14] Hervé Touati, Hamid Savoj, and Robert K. Brayton. Delay optimization of combinational logic circuits by clustering and partial collapsing. In *Proc. ICCAD*, pages 188–191, 1991.