Rephasing: A Transformation Technique for the Manipulation of Timing Constraints

Miodrag Potkonjak
C&C Research Laboratories
NEC USA, Inc.
Princeton, NJ 08540

Mani Srivastava
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Abstract - We introduce a transformation, named rephasing, that manipulates the timing parameters in control-dataflow graphs. Traditionally high-level synthesis systems for DSP have either assumed that all the relative times, called *phases*, when corresponding samples are available at input and delay nodes are zero or have automatically assigned values to as part of the scheduling step when software pipelining is simultaneously applied.

Rephasing, however, manipulates the values of these phases as a transformation before the scheduling. The advantage of this approach is that phases can be chosen to optimize the algorithm for metrics such as area and power. Moreover, rephasing can be combined with other transformations. We have developed techniques for using rephasing to optimize several design metrics. The experimental results show significant improvements.

1. Introduction

Algorithm transformations have emerged as important optimization tools in the process of high-level synthesis. For the most part transformations in high level synthesis have relied on the following two techniques: dataflow optimizations based on algebraic identities and redundancy manipulation and reorganization of control flow structures such as loops (e.g. retiming, loop unfolding).

The focus of our work is on an algorithm transformation, named *rephasing*, that belongs to a new category - it manipulates the timing relationships between different parts of a computation. While the precise positioning of operations along the time axis is the job of the scheduler, an algorithm transformation like rephasing manipulate timing relationships without violating specified timing constraints so as to veer the synthesis process towards optimizing various design metrics.

The control-dataflow graphs (CDFGs) that are used during the high-level synthesis of numerically intensive applications have associated timing parameters such as sampling period, the latencies between input-out pairs, the relative times at which corresponding samples become available on different inputs, and the relative times at which the corresponding samples become available at the delay nodes. While some of the timing parameters may be constrained by performance requirements or by the interface to the external world, others remain free to be chosen during high-level synthesis. For example, while the sample period is usually specified, the relative times at which corresponding samples become available at the delay nodes are internal timing parameters that are free to be chosen by an implementation.

Traditionally high-level synthesis systems have either assumed that the relative times, called *phases*, when corresponding samples are available at input and delay nodes are all zero or have automatically assigned values to these phases as part of the datapath allocation / scheduling step in the case of newer schedulers that use techniques like software pipelining and overlapped scheduling.

Rephasing manipulates the values of these phases as an algorithm transformation before the scheduling / allocation stage. The advantage of rephasing is that phase values can be chosen to transform and optimize the algorithm for explicit metrics like area, throughput, and power. Moreover, the rephasing transformation can be combined with other transformations such as algebraic transformations. Compared to retiming and functional pipelining, the rephasing transformation not only preserves their key advantages but also offers improvements such as elimination of granularity bottlenecks and no need for initial state computation.

We have developed techniques for using rephasing to optimize a variety of design metrics, and our results show significant improvements in several design metrics. We have also investigated the relationship and interaction of rephasing with other high level synthesis tasks.

2. Previous Work

Relevant to the manipulation of timing relationships by rephasing is the handling of timing constraints that is done by the schedulers during high level synthesis. While there are several notable exceptions [Ku92, Fil93], most of high level synthesis work has been based on the synchronous dataflow model of computation. As pointed out earlier, most high-level synthesis systems for DSP either assume that all input and delay node samples are available at the same time (all phases are zero), or indirectly assign values to the phases by using schedulers that incorporate techniques such as overlapped scheduling and

32nd ACM/IEEE Design Automation Conference ®

software pipelining to generate complex time shapes [Goo89, Pot94, Lam88].

However only recently has some limited work been done on relaxing the assumption that all phases are zero and explicitly manipulating the phases. Perhaps the first direct effort at directly manipulating the phases as part of an algorithm transformation was described by [Sri94] who applied it to simultaneously optimize throughput and latency for linear DSP systems. Besides targeting a narrower set of goals for the limited domain of linear systems, [Sri94] assumed that the corresponding samples at all the inputs are mutually aligned to each other (all input phases were zero), and the corresponding samples at all the delay nodes in CDFG were mutually aligned to each other (all delay node phases were a constant).

It is interesting to note that in some sense rephasing corresponds to cycle stealing and wave pipelining in logic synthesis [Lam92, Won89, Ung86, Ish90].

3. Rephasing: Definition and Intuition

The DSP systems that we are interested in have multiple inputs, multiple outputs, and finite state. They accept streams of samples on each of the inputs, and produce streams of samples on each of the output ports. We represent an algorithm for a system by a hierarchical directed control-dataflow graph (CDFG). In a CDFG the nodes represent data operators or sub-graphs, data edges represent the flow of data between nodes, and control edges represent sequencing and timing constraints between nodes.

The system state is represented in a CDFG by special delay operator nodes which are initialized to a user specified value. A delay operator node (often referred to as just *delay* or *state* in this paper) delays by one sample the stream of data on its sole input port. A CDFG corresponds to an algorithm for computing the output samples and the new samples to be stored at the delay nodes (i.e. the new state) given the input samples and the old (current) samples at the delay nodes (i.e. the old state).

Figure 1a shows an example CDFG with two inputs X and Y, and one output Z. The two delay nodes, U and V, are represented by boxes with the letter 'D'.

Associated with a CDFG are also timing constraints specified by the user. These constraints arise from requirements of the interface to the external world and from performance requirements. Consider a CDFG with P inputs, Q outputs, and R delay nodes (state nodes). Let,

$X[n] = (X_1[n]\ X_2[n] \ldots X_P[n])^T$ be the vector of n-th samples at the P input nodes of the CDFG

$Y[n] = (Y_1[n]\ Y_2[n] \ldots Y_Q[n])^T$ be the vector of n-th samples at the Q output nodes of the CDFG

$S[n] = (S_1[n]\ S_2[n] \ldots S_R[n])^T$ be the vector of n-th samples at the R delay nodes of the CDFG

Given initial values S[0] of the samples in the delay nodes, the CDFG repeatedly computes output samples Y[n] and new samples S[n] for delay nodes from input samples X[n] and old samples S[n-1] at the delay nodes for n = 1, 2, 3...

Since we have restricted ourselves to single-rate synchronous CDFGs, the data rates are identical at all nodes so that the inter-sample time interval is identical and constant for all nodes. The maximum rate at which such a CDFG can process samples is called its **Throughput**, and the inverse of this rate, called **Sample Period,** is the minimum required time between successive samples at a node in the CDFG. The sample period, denoted by T_S, is an important timing parameter that is usually constrained not to exceed a maximum value.

The relative arrival times of the n-th sample at each of the P input nodes form the third important set of timing parameters for a CDFG. $\Phi_I(i)$ is the skew in the arrival of $X_i[n]$, the n-th sample at the i-th input, relative to the arrival of $X_1[n]$, the n-th sample at the 1-st input. We call $\Phi_I(i)$, which may be negative, the **Phase** associated with the i-th input. The interface to the external world may require that the input phases be constrained to specific values. However many high-level synthesis systems [Rab91] simplify scheduling by assuming that the n-th sample at each input arrives at the same time, i.e. $T_{IA}(i)=0$ for all i=1..P.

It must be noted that the timing parameters T_S, $T_L(i,j)$, and $\Phi_I(i)$, for i=1..P, j=1..Q, are sufficient to completely characterize the timing associated with the inputs and the outputs. In other words, the external timing behavior of a CDFG is completely characterized by them. The values of some of these externally visible timing parameters may be constrained by design requirements.

The final CDFG timing parameter of interest is $\Phi_D(i)$, the skew in the arrival of $S_i[n]$, the n-th sample at

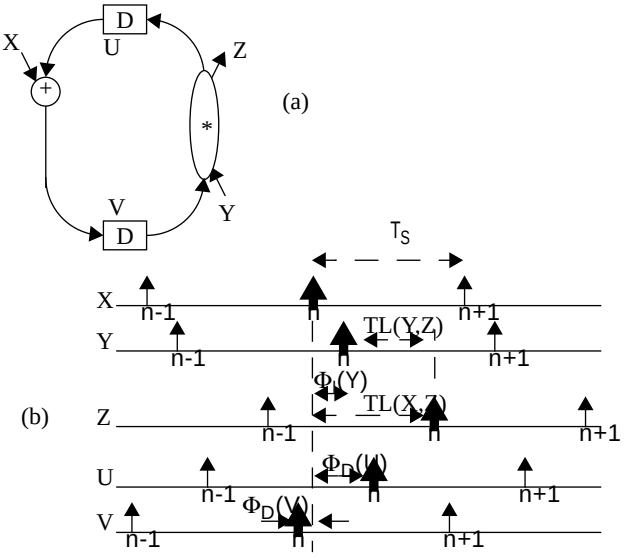

Figure 1: Timing Parameters Associated with a CDFG

the i-th delay node of the CDFG, relative to the arrival of $X_1[n]$, the n-th sample at the 1-st input. We call $\Phi_D(i)$ the **Phase** associated with the i-th delay node. It is clear that $S_i[n]$ is not available earlier than $\Phi_D(i)$ after the arrival of $X_1[n]$, and $S_i[n]$ must be calculated no later than $T_S+\Phi_D(i)$ after the arrival of $X_1[n]$. Note that $\Phi_D(i)$ may be negative. Unlike T_S, $T_L(i,j)$, and $\Phi_I(i)$ which are external timing parameters that may be constrained by the user, $\Phi_D(i)$ is an internal timing parameter that is unconstrained by the user and may be chosen so as to satisfy design constraints while optimizing design cost metrics.

What makes the delay node phases $\Phi_D(i)$ for i=1..R interesting as free timing parameters is that, as shown below, T_S and $T_L(i,j)$ can be expressed in terms of various path lengths in the CDFG, input phases, and the delay node phases. Let:

$P_{ID}(i,j)$ = length of the CDFG path (in control steps) from the i-th input node to the j-th delay node

$P_{IO}(i,j)$ = length of the CDFG path (in control steps) from the i-th input node to the j-th output node

$P_{DD}(i,j)$ = length of the CDFG path (in control steps) from the i-th delay node to the j-th delay node

$P_{DO}(i,j)$ = length of the CDFG path (in control steps) from the i-th delay node to the j-th output node

k = number of pipeline stages that have been added (or removed if k<0) to the initial CDFG that was given by the user as a specification, and from which the current CDFG was obtained after some transformations

It can be shown that:

$$T_S = \max \{P_{DD}(r,s)+\Phi_D(r)-\Phi_D(s)\ \forall r,s\in 1..R\} \cup \{P_{ID}(p,r)+T_{IA}(p)-\Phi_D(r)\ \forall p\in 1..P,\ \forall r\in 1..R\}$$

$$T_L(i,j) = k*T_S + \max \{P_{IO}(p,j)+\Phi_I(p)-\Phi_I(i)\ \forall p\in 1..P\} \cup \{P_{DO}(r,j)+\Phi_D(r)-\Phi_I(i)\ \forall r\in 1..R\}\ \forall i\in 1..P,\ \forall j\in 1..Q$$

From the above expressions it is clear that the values of T_S, T_L, Φ_I, and Φ_D are coupled - choosing some may place constraints on the achievable values of the remainder. Consequently an algorithm transformation can manipulate those timing parameters that are free. In fact, such a transformation may be combined with other algorithm transformations that effect the CDFG path lengths P_{ID}, P_{IO}, P_{DD}, and P_{DO}.

Our transformation *rephasing* is based on the idea of assigning values to the input and delay node phases that are free so as to improve desired design characteristics. Many DSP schedulers assume that all input samples and old delay node samples are available simultaneously at the beginning of a sample period (i.e. all phases are zero), and that the new delay node samples are to be calculated by the end of the sample period. Rephasing plays around with the free phases so as to stagger the computation suitably - hence the name rephasing. Figure 2introduces the basic idea behind our new transformation. The control-data flow graph (CDFG) shown has two operations: multiplication which takes 3 control cycles and addition which takes 1 control cycle. The CDFG also has two delay nodes: U and V. The delay nodes are annotated by their phases. In addition, the pair of numbers above the box corresponding to a delay node denote the earliest time at which the delay node sample is available and the latest time by which the next delay node sample has to be calculated. The goal is to obtain a minimal area implementation under the sampling rate constraint of 2 cycles. The initial best sampling rate is obviously 3 control cycles. Algebraic and redundancy manipulation transformations are not effective since only one operation is available between delay nodes. Retiming and functional pipelining are also not effective - actually they can not be applied at all since no delay nodes can be moved due to the blocking inputs and the blocking output.

Amongst previously proposed transformations, only unfolding or unfolding combined with pipelining and retiming can resolve the throughput (sampling rate) constraint. However, it is well known that unfolding often result in increased hardware requirements.

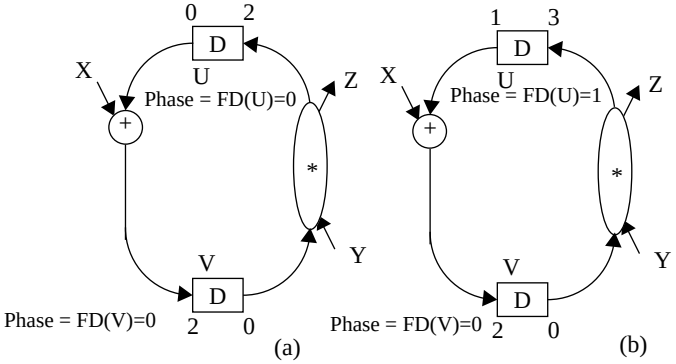

Figure 2: Rephasing: The introductory example. The example is also used to illustrate how rephasing can be used to eliminate granularity bottlenecks.

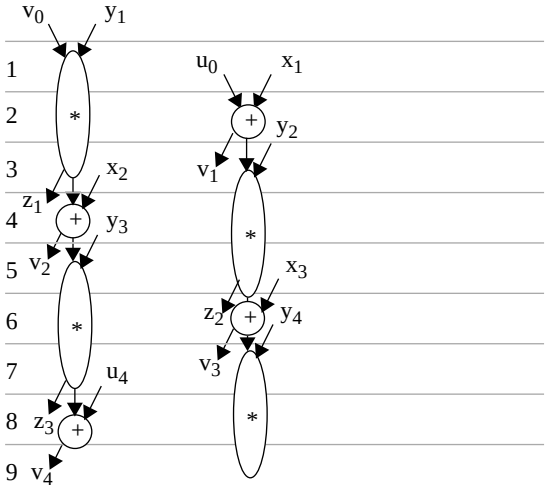

Figure 3: Using rephasing to improve throughput by eliminating granularity bottlenecks. Schedule for the example shown in two different representations in Figures 3 and 4.

Now consider the same CDFG, as shown in Figure 2b. The only difference is that the phase, or timing constraints, associated with the delay node U is moved by one control step relative to phase of the delay node V and inputs. The resulting throughput (critical path) is now only two cycles, since the maximal difference between the availability and the required times for each delay node is still two. That everything is correctly done can be checked by analyzing the corresponding functional dependences and from the graphically represented schedule in Figure 3. In the remainder of the paper we will demonstrate systematic methods for exploring this mechanism of phase alteration to improve a number of design metrics.

4. Properties of Rephasing and Using Rephasing to Optimize Throughput

The first fundamental question about rephasing is the following: Given a CDFG and a set of phase values for each delay node and each input, is the computation well defined i.e., is there a schedule that does not violate timing constrains? Recall that in each cycle of the CDFG sum of differences between the required times and the arrival times of the various delay nodes has to be at least equal to the sum of the computation delays of all the operations.

It is easy to see that this problem is equivalent to the problem of detection of negative cycles. The detection of negative cycles is a well studied problem in the theoretical computer science literature. It can be solved in cubic run time by using the Bellman-Ford or Yen algorithms [see Law76, Section 11, pages 90-91].

There is a close and easily seen relationship between rephasing and retiming [Lei91] of a CDFG. There are however several important but less obvious differences between rephasing and retiming. Interestingly almost all of the differences result in giving an edge to rephasing for use in optimizing compiler or as a high level synthesis transformation. The advantages include:

1. **There is no need for recomputation of initial states (initial values of delay node samples);**
2. **There is no blocking input/output problem and no operator node granularity bottleneck which prevent achievement of iteration bounds [Lei91];**
3. **All numerical properties.**

Throughput is one of the most important design metrics. The iteration bound imposes a fundamental limit on the achievable throughput when data-flow transformations (e.g. algebraic and redundancy manipulation) are not used. In this section we demonstrate how rephasing can be used as an effective mean to achieve iteration bound. All results can be directly derived using graph theoretic approaches. However, we will take an alternative, more insightful approach. We will establish a relationship between rephasing and a modified version of functional pipelining, and use this relationship to derive the theoretical results and to design efficient algorithms. The second approach not only simplifies the description of our methods, but also provides a valuable insight into the relationship between the two transformations. The following algorithm in polynomial times applies rephasing on an arbitrary CDFG so that a throughput equal to the iteration bound is achieved.

Rephasing for Throughput:

1. Find the iteration bound using Hartman's minimum ratio cycle algorithm [Har90];

2. Assign to each delay node a delay equal to the negative of the iteration bound (for example, if iteration bound is k, assign delay equal to -k);

3. Using the Bellman-Ford algorithm [Law76] find the longest path between one node and all other nodes for each strongly connected component. The node is arbitrarily randomly selected;

4. Assign to the selected node phase 0. Assign to each delay node the phase which equal to distance from the selected node. Assign to each input a phase that is equal to some large negative number M. M has a magnitude larger than the sum of delays of all operations nodes in the CDFG.

For sake of brevity we omit the proof. The run time of the algorithm is $O(n^3 \log n)$.

The relationship between function pipelining and rephasing is established by the following theorem which we state without proof:

Functional Pipelining-Rephasing Relationship Theorem: *Rephasing on a given CDFG G1 is equivalent to functional pipelining on a modified CDFG G2. G2 is obtained from G1 by replacing each node with k multicycle delay by a chain of k nodes with unit cycle delays.*

5. Optimizing Area and Power Using Rephasing

It has recently been demonstrated that retiming at behavioral level can be effectively used to improve resource utilization of the targeted design, and therefore reduce the implementation area [Pot94]. However, the effectiveness of retiming for area optimization is often limited by granularity and input/output bottlenecks. Figure 4a shows a typical example of this situation. It is assumed that the available time is 4 control steps. Retiming can not be applied on the initial CDFG because any movement of the delay nodes is prevented by either the output or the inputs. Since all operations, except addition +1, are on the critical path, implementation requires at least 4 multipliers, 2 adders and 1 subtracter.

The application of rephasing significantly improves the resource utilization. If delay node D1 is shifted in time by 3 cycles, see Figure 4b, it is easy to generate schedule and assignment under significantly lower allocation constraints. Table 1 shows one of the possible resulting schedules - only 1 execution of each type is now required.

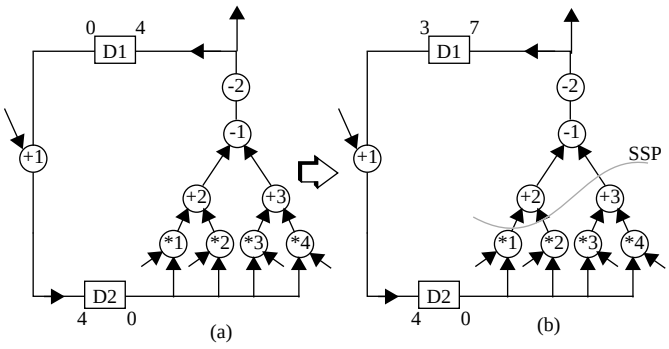

Figure 4: Area minimization using rephasing. While for the initial CDFG 4 multipliers, 2 adders, and 1 subtracter is required, for the transformed CDFG only 1 multiplier, 1 adders and 1 subtracter are sufficient. The corresponding schedules are shown in Table 1.

It has also been recognized that the constraints imposed by inputs and output positioning can be overcome by the use of pipelining. However, rephasing often

	multiplier	adder	subtracter
1. control step	*3	+2	
2. control step	*4	+1	-1
3. control step	*1	+3	-2
4. control step	*2		

Table 1: A Possible Schedule for the Rephased Example in Figure 4b

has significant advantage not just over retiming, but also over functional pipelining due to potential of rephasing to remove granularity bottlenecks.

An efficient and effective approach for area optimization requires two components: an objective function to estimate (predict) the area without the time consuming application of scheduling, assignment, and physical design tools; and an optimization algorithm for minimizing the objective function.

Our objective function for area optimization is calculated using the following approach. For each operation *i* with as-soon-as-possible time $ASAP_i$, as late as possible time $ALAP_i$, and the length Duration_i we define the function

$$u_i(k) = \frac{1}{ALAP_i - ASAP_i + 1} \quad \text{for each time slot} \quad t \le k < t + \text{Duration}_i$$

. Next, we calculate the values *U(t)* of the function that describes the likelihood that some operation of the specified type is scheduled in a given time step. This is done by accumulating the contributions of all operations of the specified hardware type:

$$U(t) = \sum_{i=1}^{\text{NumOperations}} u_i(t) \qquad \textbf{(EQ 1)}$$

As an estimation of the requirements for execution units of a specific type, we take the maximum of the function U(t) over all control steps for the corresponding type of operations. Similarly, by considering each type of transfer of data from a particular type of execution unit *i* to a particular execution unit of type *j*, we calculate a function indicating the need for interconnect of type *ij*. We estimate the lifetime of each variable by assuming that it is alive from the most likely control step that its producing operation is scheduled to the expected moment when the latest operation which consume the variable is scheduled. Expected times for both operations are calculated as the middle of the ASAP-ALAP interval. The objective function (OF) is the weighted sum of three components: execution units, interconnect, and registers. The weights are proportional to the cost of each component.

The new optimization algorithm is based on the force-directed paradigm [Pau89] where at each step the most critical part of the predicted cost is targeted. The following pseudo-code describes the algorithm:

while (it is 1st pass, or there was improvement in the previous iteration) {

find objective function (OF) for each one clock step change of the phase for each delay node and input;

find the best OF, and the corresponding one clock steps change of phase;

apply the best selected one clock step rephasing;

}

An important addition that we made to the force-directed optimization is the use of min-bounds [Rab91]. As soon as the requirement for some type of unit reaches the min-bound, further improvements in value of this component are not considered as the improvements of the objective function OF, since the minimum along this dimension has been reached.

Recently, power optimization has become important [Cha92] due to the increasing desire for portability. We now present a rephasing algorithm for power optimization under throughput constraint.

Power minimization at the behavioral level can to the first order of approximation be treated as a simultaneous optimization of throughput, area and number of operations [Cha92]. The experimental studies confirm the validity of this approximation [Cha92]. The objective function for power is more involved than the one for area due to a need for taking into account the energy consumption not only in the data path components but also in the control logic. We used the behavioral level power prediction tools [Cha92] in Hyper for this task.

Rephasing does not alter the number of operations, with the only exception being that it can reduce the number of transfer operations that are needed. This change is usually relatively small. Extensive experimental studies as well as theoretical analysis indicate that power vs.

throughput function has unimodal shape, i.e. there exist a single minimum [Cha92].

Based on this observation we developed an algorithm for power minimization using rephasing. The algorithm conducts a search along the throughput axis. At each step the area, and therefore the effective capacitance, is minimized by applying the algorithm for area minimization using rephasing.

6. Experimental Results and Conclusion

Throughput optimization using rephasing is a problem of polynomial complexity (see Section 3.0). Therefore, the key question is not how well the algorithm performs, but how much improvement can be achieved using rephasing. To analyze the improvements we studied 55 DSP examples. The average critical path was 67 control cycles, while the median was 14. After rephasing the average and median critical path were long 3.6 and 2 control cycles respectively. So the average and median improvements were was by factors 20 and 7 respectively.

Table 2 shows the performance of rephasing for area optimization on the set of 10 examples. The average reduction of area was by 31.5%, the median reduction of area was by 35%.

Design	Initial Area [mm^2]	Final Area [mm^2]	Final/Initial [%]
8X8 DCT	40.46	22.19	54.8
9FWT	51.72	51.72	65.0
11WFT	54.08	25.94	66.5
11FIR	7.67	4.92	64.1
7IIR	18.27	16.25	88.9
Volterra	34.42	20.48	59.5
Lin5	37.71	30.45	80.7

Table 2: Area Optimization Using Rephasing

Table 3shows the performance of rephasing for power optimization on the set of 10 examples. The average reduction of power was by 3.76 times, the median reduction of power was by 4.06 times.

Design	Initial Power [nJ/sample]	Final Power [nJ/ sample]	Initial/ Final/
8X8 DCT	284.42	49.27	5.77
9FWT	76.52	18.32	4.18
11WFT	80.02	20.00	4.00
11FIR	21.17	5.21	4.06
7IIR	66.20	34.27	1.93
Volterra	81.77	40.06	2.04
Lin5	39.91	9.21	4.33

Table 3: Power Optimization Using Rephasing

We introduced a new type of transformation - *rephasing*. Rephasing changes the timing constraints associated with delay nodes, inputs, and outputs, and can be used to address a variety of different design goals. We demonstrated the effectiveness of rephasing in optimizing a number of design metrics.

7. References

[Cha92] A.P. Chandrakasan, et al., "Hyper-LP: A Design System for Power Minimization using Architectural Transformations", ICCAD, 300-303, 1992

[Fil93] D. Filo, D.C. Ku, C.N. Coehlo, G. De Micheli: "Interface Optimization for Concurrent Systems Under Timing Constraints", IEEE Trans. on VLSI Systems, Vol. 1, No. 3, pp. 268-281, 1993.

[Goo89] G. Goossens, J. Wandewalle, H. DeMan: "Loop Optimization in register-transfer scheduling for DSP-systems", DAC-89, pp. 826-831, 1989.

[Har90] M. Hartman: "On cycle means and cycle staffing", Technical Report UNC/OR/TR-90/14, University of North Carolina, June 1990.

[Ish90] A.T. Ishii, C.E. Leiserson: "A Timing Analysis of Level-Clocked Circuitry", Advance Research in VLSI Conference, pp. 113-130, 1990.

[Ku92] D. Ku, G. De Michelli: "High Level Synthesis of ASICs Under Timing and Synchronization Constraints", Kluwer, Norwell, MA, 1992.

[Lam88] M. Lam: "Software Pipelining: An Effective Scheduling Technique for VLIW Machines", SIGPLAN'88 Conf. on Programming Language Design and Implementation, pp. 318-328, 1988.

[Lam92] W.C.K. Lam, R.K. Brayton, A.L. Sangiovanni-Vincentelli: "Valid Clocking in Wavepipelined Circuits", ICCAD, pp. 518-525, 1992.

[Law76] E.L. Lawler: "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, New York, NY, 1976.

[Lei91] C.E. Leiserson, J.B. Saxe: "Retiming Synchronous Circuitry", Algorithmica, Vol. 6, No. 1, pp. 3-36, 1991.

[Pau89] P.G. Paulin, J.P. Knight: "Scheduling and Binding Algorithms for High-Level Synthesis", DAC-89, pp. 1-6, 1989.

[Pot94] M. Potkonjak, J. Rabaey: "Optimizing Resource Utilization Using Transformations" IEEE Transactions on CAD, Vol. 13, No. 3, pp. 277-292, March 1994.

[Rab91] J. Rabaey, et al., "Fast Prototyping of Datapath-Intensive Architectures", IEEE Design and Test of Computers, Vol. 8, No. 2, pp. 40-51, June 1991.

[Sri94] M.B. Srivastava, M. Potkonjak: "Transforming Linear Systems for Joint Latency and Throughput Optimization", EDAC-94, pp. 267-271, 1994.

[Ung86] S.H. Unger, C-J. Tan: "Optimal Clocking Schemes for High Speed Digital Systems", IEEE Transactions on Computers, Vol. 35, No. 10, pp. 880-895, 1986.

[Won89] D. Wong, G. De Micheli, M. Flynn: "Inserting Active Delay Elements to achieve wave pipelining", ICCAD, pp. 270-273, 1989.