Optimizing Control by Delayed Execution of Operations

David C. Ku Dave Filo Giovanni De Micheli

Center for Integrated Systems
Stanford University

1 Introduction and Background

We consider the synthesis of synchronous digital systems from behavioral descriptions that include the specification of timing constraints [1]. Significant progress has been made in the area of data-path and register-transfer level synthesis. We address the problem of finding a *minimal-area control implementation*, such that the overall hardware is a valid implementation of its behavioral model. Control optimization can be performed either at the logic level, by using a *finite-state machine* model [7, 9], or at a higher level, by using a hardware model described in terms of constraints on the sequencing and timing of the operations [10]. The former case includes sequential logic synthesis and microcode compaction techniques where operations are bound to control states [8, 2]. Since the cycle-per-cycle behavior of the control cannot change, these approaches achieve only limited control cost reduction in many designs. In contrast, the latter approach takes advantage of operation mobility in optimizing control, i.e. operations are not bound to control states. The only requirement is to satisfy the sequencing dependencies and the timing constraints in the specification. The wider latitude in choosing among a set of possible implementations can lead to a more efficient control implementation in terms of area. We present in this summary a control optimization approach based on *delayed execution of operations* that supports detailed timing constraints and unbounded delay operations, implemented in the *Hercules/Hebe* high-level synthesis system [5]. A full description of this technique is reported in [3].

Hardware Model. We model hardware timing behavior as a polar directed weighted *constraint graph* $G(V, E)$; the vertices V represent the operations, and the edges E capture the timing relationships (sequencing and min/max constraints) among the operations. The model supports *unbounded delay* operations such as synchronization mechanisms and data-dependent loops, concurrency, hierarchy, and detailed timing constraints. We refer the interested reader to [6] for details of the constraint graph model.

The execution delay of a vertex v is denoted by $\delta(v)$, which can be either fixed or unbounded. A weight w_{ij} associated with each edge $e_{ij} = (v_i, v_j) \in E$ represents the requirement that the start time of v_j (denoted by $T(v_j)$) must occur later than w_{ij} after the start time of v_i, i.e. $T(v_j) \geq T(v_i) + w_{ij}$. The edges are categorized into *forward* (E_f) and *backward edges* (E_b). The forward (backward) edges have positive (negative) weights and represent minimum (maximum) timing requirements among the operations. Both forward and backward edges may have unbounded weights. The graph model captures the essential timing relationships among the operations, and it serves to determine the extent to which operations can be delayed in optimizing the control implementation. We assume that the mapping of operations to resources has been performed, and the resource conflicts resolved prior to control synthesis.

Control Model. To define the objective of the control optimization, we describe the mapping from a constraint graph to a control implementation. The mapping involves two tasks – scheduling and control generation. *Scheduling* finds the start times of the operations satisfying the timing constraints, which are then used by *control generation* to derive an FSM specification of the control.

- *Scheduling the operations* – The presence of *unbounded delay operations* in our hardware model invalidates the traditional scheduling formulation since an absolute schedule satisfying timing constraints no longer exists. We use a formulation called *relative scheduling* to schedule an operation with respect to the completion of a set of unbounded delay operations, called **anchors**; we refer to [6] for further details. The **anchor** set $A(v)$ of a

vertex v is the subset of anchors that are in the transitive fan-in of v. The start time $T(v)$ is defined as offsets $\sigma_a(v)$ from each anchor in the anchor set $a \in A(v)$, i.e. $T(v) = \max_{a \in A(v)}\{T(a) + \delta(a) + \sigma_a(v)\}$. In the presense of unbounded delays, a timing constraint is characterized as well-posed (ill-posed) if its satisfiability does not (does) depend on unbounded delays. For well-posed constraint graphs, a relative schedule exists if and only if there are no positive cycles in the constraint graph. A constraint graph is valid if it is both well-posed and contains no positive cycles.

- *Generating the Control* – Given a schedule, we abstract the task of control generation as generating *enable/done* signals for each vertex v such that its execution is initiated by the assertion of $enable_v$. We model the control in terms of a modular interconnection of synchronous FSMs; the FSM abstraction decouples the control generation from a particular style of logic-level implementation. Note that our control abstraction considers only the synchronization of an operation with respect to the completion of its anchors; the support for conditional branching and looping is described in [4].

  The control is divided into *offset control* for each anchor, and *synchronization control* for each vertex. The offset control for an anchor a indicates the time offsets with respect to the completion of a. It is abstracted as an FSM that is activated by the assertion of $done_a$. The FSM generates a set of signals $C_a(i), 1 \leq i \leq \sigma_a^{max}$ where $C_a(i)$ is asserted when *at least* i cycles have elapsed after the completion of a. The synchronization control for a vertex v synchronizes the activation of v, denoted by $enable_v$, to offsets from the completion of its anchors. Specifically, the enable signal is defined as $enable_v = \prod_{\forall a \in A(v)} C_a(\sigma_a(v))$. Figure 1 shows the block diagram of the offset control. Note that the number of states in the offset control for a is equal to the maximum value of offsets w.r.t. a (i.e. σ_a^{max}) in the schedule.

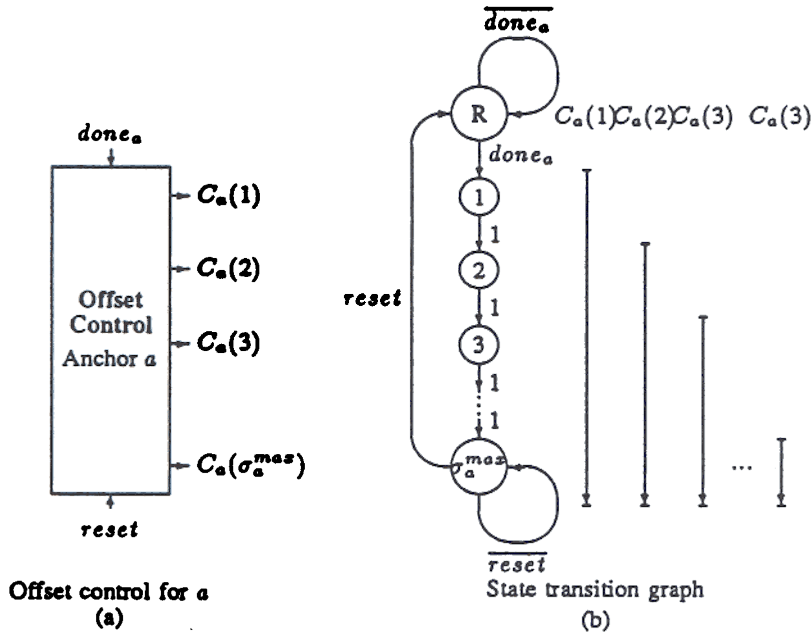


Figure 1: Offset control for an anchor a: (a) block diagram, where $done_a$ denotes completion of the anchor a, (b) FSM model, where the offset signal $C_a(i)$ is asserted when the FSM is in the states covered by the corresponding interval.

Given a specification of control logic in terms of FSMs, well-known logic synthesis techniques can be applied to generate a minimal area sequential logic implementation [7, 9]. However, a direct relationship between the FSM model and the final sequential logic implementation is difficult to accurately obtain because of the complexity of sequential logic optimizations, including the state assignment phase. We therefore estimate the total control cost $COST_{area}$ of the FSM implementation of control as:

$$
\begin{aligned}
COST_{area} &= \sum_{\forall a \in A} COST_{off}(a) + \sum_{\forall v \in V} COST_{sync}(v) \\
&= \alpha \cdot \sum_{\forall a \in A} f_{off}(\sigma_a^{max}) + \beta \cdot \sum_{\forall v \in V} f_{sync}(|A(v)|)
\end{aligned}
$$

The first term $COST_{off}$ is related to the cost due to the length of the schedule, and is a function f_{off} of the maximum offset values that yields the number of registers implementing the offset FSM. The second term $COST_{sync}$ is related to the cost of the synchronization logic, and is a function f_{sync} of the size of the anchor sets. The values α and β represent appropriate weight factors related to the actual cost of the logic implementation.

Alternative strategies to describe the control logic exist, e.g. we can implement the offset control as a counter and the synchronization as a set of comparisons between the counter values and appropriate offsets, or we can implement the offset control as a shift register and the synchronization control as logic conjunctions of the appropriate shift register entries. The two alternate implementation styles are shown in Figure 2. Let α be the cost of a register, and let β be the cost of a literal. In the counter-based implementation of Figure 2(a), we see that the complexity of the offset control is a logarithmic function of the maximum offset values, i.e. $f_{off}(n) = \lceil \log_2(n) \rceil$ represents the number of registers. Likewise, the complexity of the synchronization control is a linear function of the number of comparators, i.e. $f_{sync}(n) = n$ represents the number of literals in the required comparators. In the shift-register based implementation of Figure 2(b), f_{off} is now a linear function of the maximum offset values, i.e. $f_{off}(n) = n$. However, since comparisons are no longer needed in the synchronization, $f_{sync}(n)$ represents the number of literals in a n-input AND gate. We see that in all these formulations, the control complexity can be reduced by either minimizing the *maximum offsets* and/or by reducing the size of the *anchor sets*.

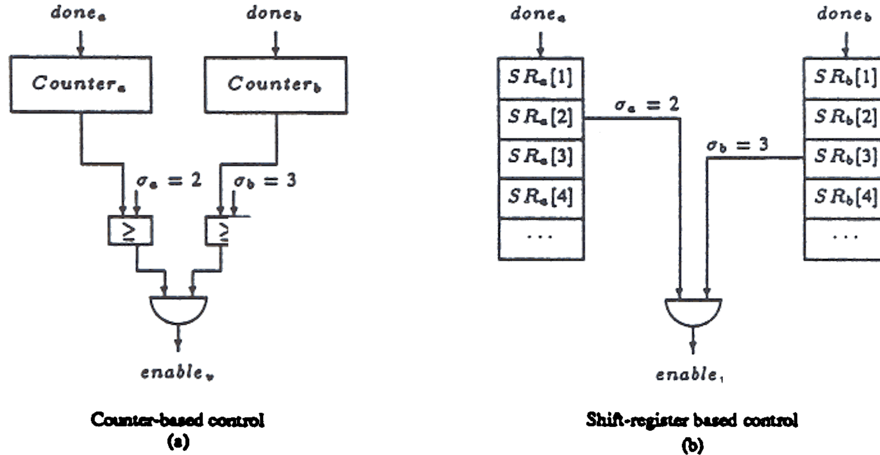


Figure 2: Alternate implementation styles for control generation: (a) counter-based, where offset control is implemented as counters and synchronization control is implemented as comparators, and (b) shift-register based, where offset control is implemented as shift registers and synchronization control is implemented as AND gates.

2 Control Optimization Formulation

The task of control optimization can be formulated as the task of minimizing the control cost $COST_{area}$ by *delaying the execution of operations*, where we consider any modification to be acceptable as long as it satisfies all the constraints of the original specification. In other words, control optimization reduces the control cost by altering the schedule to take advantage of the degrees of freedom in assigning operations to control states. We describe next how operations can be delayed by *lengthening* or *serializing* to reduce the control cost by introducing *redundancy* into the anchor sets.

Redundancy arises due to the cascading effect of anchors. Specifically, it is often the case that some anchors in the anchor set of a vertex v are not needed in the computation of the start time, i.e. $T(v)$ is unchanged if offsets from these anchors are not used in its computation. We call these anchors **redundant**, and the remaining anchors **irredundant**. An example of redundancy is shown in Figure 3. We state without proof that the start time computed with only irredundant anchors is identical to the start time computed with the full anchor set, for well-posed constraints and minimum offsets [6]. By using only irredundant anchors in computing the start time, the control cost can be reduced significantly by (1) reducing the size of the anchor sets, translating to lower synchronization costs, and by (2) reducing the values of maximum offsets, translating to fewer number of states in the corresponding FSM. Returning to the example in Figure 3(b), the sum of maximum offsets is decreased from $10(\sigma_a^{max}) + 5(\sigma_b^{max}) = 15$ to $5(\sigma_a^{max}) + 5(\sigma_b^{max}) = 10$, and the cardinality of the anchor set of v is decreased from 2 to 1. Therefore, it is possible to reduce the cost of the control implementation without affecting the schedule by using the concept of redundancy. We show now how anchors can be made redundant, and we analyze the extent to which redundancy can be introduced to the graph.

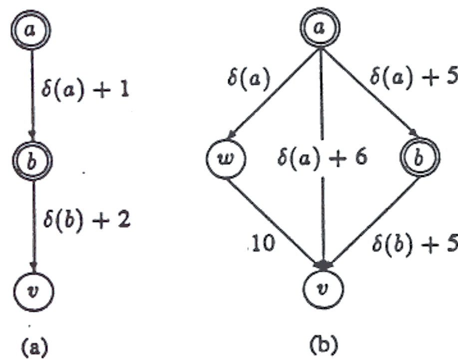


Figure 3: Examples of *redundant* anchor a with respect to the vertex v. In both cases, note that the longest path from a to v passes through an anchor b.

2.1 Making Anchors Redundant

It is possible in some cases to make an otherwise irredundant anchor redundant by either *lengthening* or *serializing* a constraint graph $G(V, E)$. An **anchor lengthening** of G delays the activation of a vertex v with respect to an anchor $a \in A(v)$ by increasing the length of a path from a to v. An **anchor serializing** of G serializes a vertex v with respect to an anchor $a \notin A(v)$ by introducing a sequencing edge from a to v, so that the activation of v now depends on the completion of the anchor. Figure 4 and Figure 5 illustrate how the anchor a can be made redundant with respect to vertex v by the two techniques.

The lengthening and serializing must be carried out with care to avoid violating the constraint graph, i.e. make the resulting graph ill-posed or introduce positive cycles. We state the following theorem that determines the extent to which anchor lengthening and serializing can be applied before constraints are violated. The proofs, which can be found in [3], are not presented here for brevity.

Theorem 2.1 *Consider a valid constraint graph $G(V, E)$ in which a is an anchor and v is a vertex. If a forward edge $e_{av} = (a, v)$ with unbounded weight $w_{av} = \delta(a) + \alpha$, $\alpha \geq 0$ is introduced from a to v such that (1) no cycles are formed by the forward edges of G, and (2) no positive cycles are formed in G, assuming all unbounded delays are set to zero, then the resulting graph $\tilde{G}$ is valid.*

2.2 Prime Versus Non-prime Anchors

In anchor lengthening, the anchor sets of the vertices remain unchanged because no new dependencies are introduced. We now analyze the structure of the graph to identify the subset of anchors which can be made redundant by anchor lengthening alone, i.e. without introducing new serializations. For example, if there exists no other anchors on any

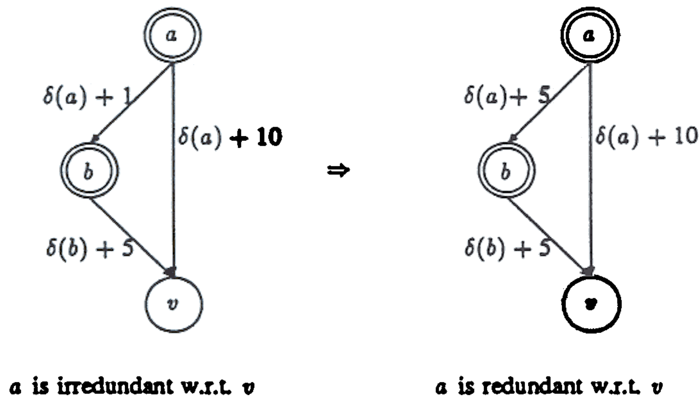


Figure 4: Making the anchor a redundant with respect to v by *anchor lengthening* the edge e_{ab}. Before lengthening, a is not redundant because for the case $\delta(b) < 4$, the offset of 10 from a is necessary to satisfy the minimum constraint.

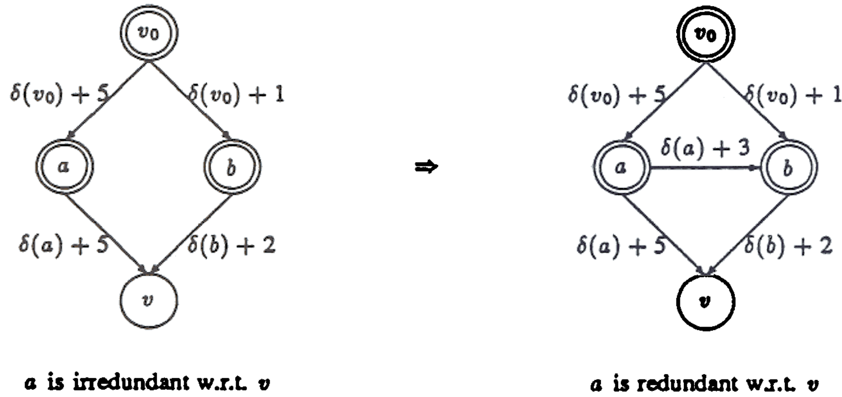


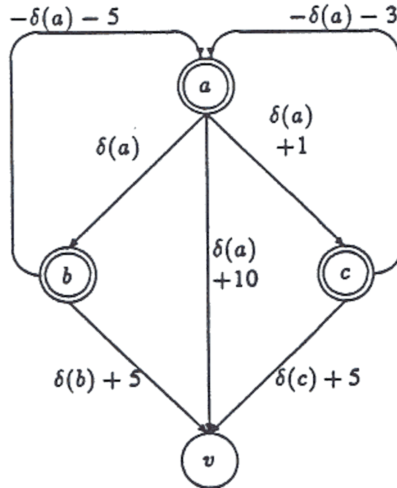
Figure 5: Making the anchor a redundant with respect to v by *anchor serializing* a and b, introducing the edge e_{ab}. Note that before serializing, a and b are prime anchors of v. After serializing, only b is a prime anchor of v.

path from an anchor a to a vertex v, then a can not be made redundant by lengthening any existing edges. We formalize this observation with the following definition.

Definition 2.1 *An anchor $p \in A(v)$ of a vertex v is* **prime** *if and only if no unbounded delays other than $\delta(p)$ are encountered on all paths of forward edges from p to v. Otherwise, the anchor is* **non-prime**. *The set of prime (non-prime) anchors of a vertex v is the* **prime (non-prime) anchor set** *of v.*

Theorem 2.2 *A prime anchor p of a vertex v is always irredundant with respect to v.*

The prime anchors constitute the minimal set of reference points that affect the activation of a given operation. We observe that an irredundant non-prime anchor can be made redundant by lengthing the path from the non-prime anchor to another anchor on the path, provided that no timing constraints are violated. Let us consider the example in Figure 6 with two maximum timing constraints. Irredundant anchor a can be made redundant by either lengthening b-to-v or lengthening c-to-v. However, the path from a to b cannot be lengthened beyond the maximum constraint of 5, and the path from a to c cannot be lengthened beyond the maximum constraint of 3. An important question is whether an anchor lengthening exists to make all non-prime anchors redundant. We state the following key theorem that demonstrates the existence of a solution to reduce to a minimum the irredundant anchor sets.



Making a redundant w.r.t. v

Figure 6: Example to make anchor a redundant with respect to vertex v by anchor lengthening only. Two backward edges with unbounded weight corresponding to maximum timing constraints between a and b, and between a and c. Note that the graph is valid since no unbounded length cycle exists.

Theorem 2.3 *Given a valid constraint graph $G(V, E)$, there always exists an anchor lengthening of G such that all non-prime anchors of every vertex v are redundant with respect to v.*

3 Control Optimization Algorithms

The objective of control optimization is to reduce the cost of control implementation for a given constraint graph. The two factors in the control cost – *synchronization* and *offset* costs – are tightly related. The reduction of one may result in an increase of the other. Simultaneous minimization of both factors requires casting the problem as a combinatorial optimization, which is computationally hard to solve exactly. Our strategy is to instead use a heuristic approach that exploits the structure of the constraint graph in minimizing control cost. The idea is to *partition* the vertices into groups, each of which depends on a minimal set of anchors. It consists of the following steps.

1. *Identify anchor clusters.* The anchors are first partitioned into *anchor clusters* where each cluster consists of a subset of anchors that are strongly connected by a cycle in G. It can be shown that a partial order exists among the clusters and that anchors within a cluster cannot be serialized with respect to each other.

2. *Order clusters to form a chain.* The anchor clusters are ordered in a manner compatible with the partial order, forming a *chain* of clusters. The ordering is obtained by sorting the anchors according to their longest path lengths from the source vertex; this ranking determines the ordering among the clusters. Anchor serialization is then used to impose the complete ordering among the clusters by adding edges between the clusters. This ordering, being only one of many possible orderings, is used because it is easily computed and it leads to a minimum control cost for constraint graphs with no maximal constraints.

3. *Assign vertices to clusters.* The non-anchor vertices are then grouped and distributed among the segments of the cluster chain. Vertices are assigned to the latest possible cluster such that maximal offsets are not increased. A vertex v assigned to a cluster a is serialized with respect to a to minimize the size of its prime anchor set. Every prime anchor set is a subset of an anchor cluster because of the complete ordering imposed in the previous step.

4. *Lengthen graph.* From Theorem 2.3, lengthening insures that all non-prime anchors for each vertex are made redundant. This reduces synchronization costs because redundant anchors are not needed to compute the start

times of the vertices. Lengthening is also used to delay operations as long as possible under the restriction that the overall control cost does not increase. This has the effect of reducing anchor offsets.

The resulting graph consists of an alternating sequence of anchor clusters and groups of vertices, where a group of vertices depends on the anchor cluster that precedes it. The control cost is reduced because the synchronization cost of a vertex is at most the size of the anchor cluster to which it is assigned to. The strategy guarantees that given a constraint graph with an associated control cost $COST_{old}$, the control optimization can be performed such that the new cost $COST_{new}$ is *always* lower than or equal to the old cost, i.e. $COST_{new} \leq COST_{old}$. For the case of no maximal timing constraints in the graph, the resulting graph achieves globally minimum control cost. In particular, both the sum of maximum offsets and the sizes of irredundant anchor sets are reduced to a minimum.

We illustrate the application of the algorithm in Figure 7. The graph contains four anchors $\{A, B, C, H\}$, with A and I as the source and sink vertices, respectively. Each anchor is an anchor cluster because no two anchors are connected on a cycle. The algorithm first orders the anchors according to the longest path from the source; this results in the order $A \rightarrow C \rightarrow B \rightarrow H \rightarrow I$. The vertex D is assigned to the cluster with anchor A. For the graph to remain well-posed, the vertex F must also be assigned to the same cluster as D. Since the maximum offset of anchor A must be at least 10 due to the presence of F, the anchor C can be delayed to $\delta(A) + 10$ without increasing the maximal offset of A. The rest of the vertices are similarly placed so that the resulting graph consists of an alternating sequence of anchors and vertices. The offset cost is reduced from 49 to 23, and the synchronization cost is reduced from 16 to 8.

4 Implementation and Results

The control optimization approach has been implemented in the framework of the Hercules/Hebe High-level synthesis system. The constraint graph model is derived from a high-level specification of hardware behavior, which is then used as the basis for scheduling and control synthesis. We present in Figure 8 the results of applying the technique on some benchmark examples, including diffeq, elliptic digital filter, error-correcting encoder and decoder, and the greatest common divisor. For each example, the table gives the number of anchors $|A|$, the number of vertices $|V|$, the sum of the maximum offsets and the sum of the anchor sets for (1) the full anchor set, (2) the irredundant anchor set, and (3) after control optimization has been performed. The mapped control logic cost in terms of Actel cells is also given. For control-dominated designs, such as the ECC encoder and decoder, the reduction in control is significant in terms of the overall reduction in area. Note that in these examples some of the anchors are introduced by arithmetic operations, such as multiply and divide, that are implemented using data dependent algorithms.

References

[1] R. Camposano and A. Kunzmann. Considering timing constraints in synthesis from a behavioral description. In *Proceedings of the International Conference on Computer Design*, pages 6–9, 1986.

[2] G. Goossens, Jan Rabaey, Joos Vanderwalle, and Hugo DeMan. An efficient microcode compiler for custom DSP-processors. In *Proceedings of the International Conference on Computer-Aided Design*, pages 24–27, Santa Clara, November 1987.

[3] D. Ku, D. Filo, and G. De Micheli. Control optimization based on resynchronization of operations. CSL Technical Report CSL-TR-91, Stanford, March 1991.

[4] D. Ku and G. De Micheli. Optimal synthesis of control logic from behavioral specifications. CSL Technical Report CSL-TR-89-401, Stanford, November 1989. and VLSI Integration Journal, to appear.

[5] D. Ku and G. De Micheli. High-level synthesis and optimization strategies in Hercules and Hebe. In *Proceedings of the European ASIC Conference*, Paris, France, May 1990.

[6] D. Ku and G. De Micheli. Relative scheduling under timing constraints. In *Proceedings of the 27 th Design Automation Conference*, Orlando, June 1990.

[7] S. Malik, E. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. In *Proceedings of the 23 rd Hawaii International Conference on System Sciences*, pages 397–406, Hawaii, 1990.

[8] A. W. Nagle. Automatic synthesis of micro controllers. *Annual Workshop on Microprogramming, ACM SIGMICRO Newsletter*, 9:112–117, 1979.

[9] G. Saucier, C. Duff, and F. Poirot. State assignment of controllers for optimal area implementation. In *Proceedings of the European Design Automation Conference*, pages 547–551, Glasgow, Scotland, March 1990.

[10] W. Wolf. Rescheduling for cycle time by reverse engineering. In *Proceedings of the ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Univ. of British Columbia, August 1990.

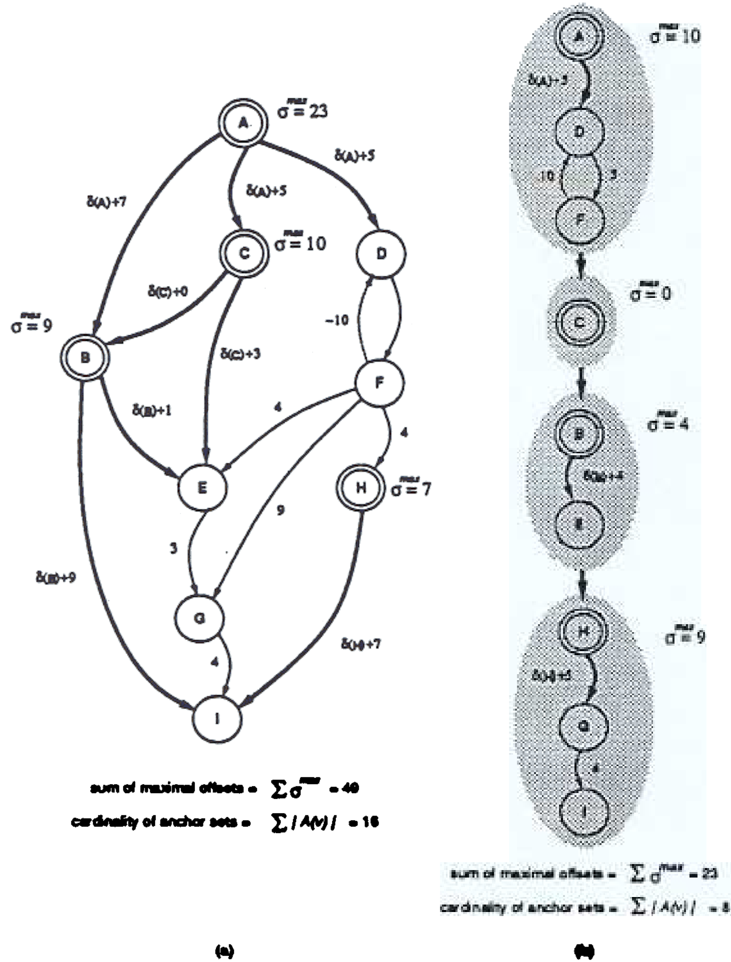


Figure 7: Constraint graph example to illustrate the algorithm. There are four anchors A, B, C, H, denoted by double-circled vertices. The solid (dotted) arcs represent forward (backward) edges.

Example graph	$\lvert A \rvert / \lvert V \rvert$	*Offset* $\sum \sigma_a^{max}$			*Sync* $\sum \lvert A(v) \rvert$			*Mapped logic*	
		Full	Irred.	Opt.	Full	Irred.	Opt.	*Before*	*After*
Diffeq	9/41	27	12	8	106	52	36	248/2848	135/2735
Elliptic	9/66	370	141	95	247	73	63	647/2147	525/2025
ECC encoder	2/47	62	62	42	80	63	43	246/256	203/213
ECC decoder	2/53	59	59	32	92	76	49	316/366	251/301
Gcd	6/38	15	7	7	45	28	28	320/434	320/434

Figure 8: Summary of results. The control costs are given for the *full*, *irredundant*, and *optimized* anchor sets. The logic cost is the # of Actel cells for the *control* portion and the *total* area.