A High-Level Synthesis Flow for Custom Instruction Set Extensions for Application-Specific Processors

Nagaraju Pothineni[1], Philip Brisk[2], Paolo Ienne[3], Anshul Kumar[4], Kolin Paul[4]

[1]Google India Pvt Ltd.
Bangalore, India – 560016
nagarajup@google.com

[2]Department of Computer Science and Engineering
Bourns College of Engineering
University of California, Riverside
Riverside, CA 92521
philip@cs.ucr.edu

[3]School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland CH-1015
Paolo.Ienne@epfl.ch

[4]Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
New Delhi, India – 110016
{anshul, kolin}@cse.iitd.ernet.in

***Abstract*—Custom instruction set extensions (ISEs) are added to an extensible base processor to provide application-specific functionality at a low cost. As only one ISE executes at a time, resources can be shared. This paper presents a new high-level synthesis flow targeting ISEs. We emphasize a new technique for resource allocation, binding, and port assignment during synthesis. Our method is derived from prior work on datapath merging, and increases area reduction by accounting for the cost of multiplexors that must be inserted into the resulting datapath to achieve multi-operational functionality.**

I. INTRODUCTION

Extensible processors allow the user to augment a base processor, typically an in-order RISC or VLIW, with application specific custom *instruction set extensions (ISEs)* [4, 13-15, 17]. ISEs may be realized in ASIC technology, along with the processor, or synthesized on a closely-coupled reconfigurable datapath. FPGA-based *soft processors*, e.g., *Xilinx Microblaze* or *Altera Nios II*, are extensible as well.

This paper describes a high-level synthesis flow for ISEs that have been identified by a compiler. Fig. 1 depicts the flow in the context of a compilation and synthesis system for extensible processors. The input to the flow is a set of multi-cycle ISEs, G_1, …, G_n, that have been identified using existing techniques, the number of read and write ports of the base processor's register file, R and W, and a clock period constraint Λ. The output of the flow is a datapath whose critical path delay does not exceed Λ, and executes each ISE with minimal latency, given the I/O constraints of the base processor's register file.

The first step of the flow is to compute a minimum latency schedule for each ISE given the I/O constraints of the processor [15, 17]. The second step of the flow reschedules each ISE to reduce a coarse-grain estimate of the resource requirements; the scheduler ensures that the critical path is not violated and that the latency of the schedule does not increase. The third step decomposes each multi-cycle ISE into a set of contiguously occurring single-cycle ISEs; during each cycle, at most R values are read from, and at most W values are written to, the processor's register file. The fourth step performs resource allocation, binding, and port assignment for the single-cycle ISEs produced by the third step. Our problem formulation is based on *datapath merging (DPM)*: our tool merges the single-cycle ISEs to form a single datapath that can execute each ISE without increasing its latency. Our work extends prior techniques for DPM to account for port assignment, which influences the size and number of multiplexors (muxes) that must be inserted into the datapath to facilitate multi-operational functionality.

Fig. 2 illustrates the binding process. Fig. 2(a) shows two ISEs that compute $A \bullet B$ and $B \bullet C$, where $\bullet$ is a binary commutative operator. One allocation and binding solution is shown in Fig. 2(b), which has one instance each of A, B, C, and $\bullet$. In this datapath, $\bullet$ has three predecessors, which is infeasible as $\bullet$ is a binary commutative operator. One possibility is to allocate a mux on one of the inputs of $\bullet$, as shown in Fig. 2(c); another is to instantiate two instances of $\bullet$, as shown in Fig. 2(d). The solution in Fig. 2(c) is preferable if $\bullet$ is larger than a mux; the solution in Fig. 2(d) is preferable otherwise.

Inserting muxes may cause the critical path delay to exceed the clock period constraint Λ. When this occurs, we re-execute the flow with a constraint of Λ - δ, where δ is a small constant. δ acts a slack factor, leaving space in the clock period for muxes that will be inserted during DPM. The flow repeats until the clock period of Λ is met.

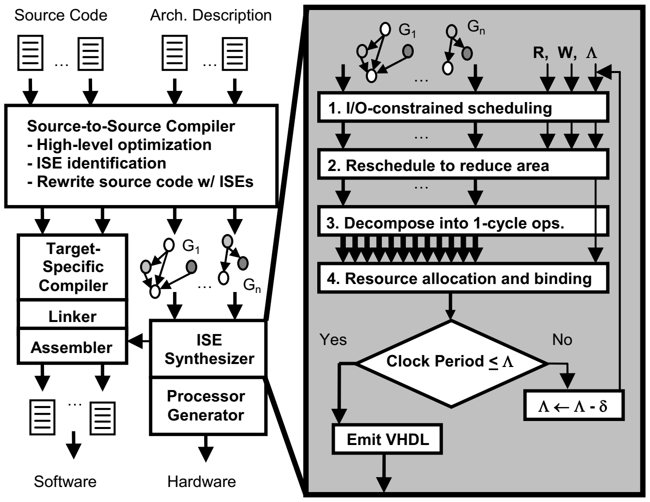
Source Code
Arch. Description
Source-to-Source Compiler
- High-level optimization
- ISE identification
- Rewrite source code w/ ISEs
Target-Specific Compiler
Linker
Assembler
G1
Gn
ISE Synthesizer
Processor Generator
Software
Hardware
R, W, Λ
1. I/O-constrained scheduling
2. Reschedule to reduce area
3. Decompose into 1-cycle ops.
4. Resource allocation and binding
Yes
No
Clock Period ≤ Λ
Λ ← Λ - δ
Emit VHDL

Figure 1. A compilation and synthesis flow for an extensible processor; the ISE synthesis flow, shown on the right, is the focus of this paper.

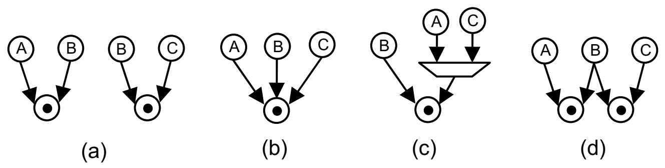

Figure 2. Two ISEs (a); one infeasible allocation and binding solution that does not respect the assumption that • is a binary operator (b); two feasible solutions (c) and (d): the former has lower area if • is larger than the mux.

II. Latency-constrained Rescheduling

The first step of the flow determines the minimum latency of each ISE given the I/O constraints of the processor's register file [15, 17]. The following step reschedules each ISE in order to reduce area, without increasing its latency. Our approach uses *simulated annealing*.

Let G_i be an ISE and $SW(G_i)$ and $HW(G_i)$ be the number of cycles required to execute G_i in software and hardware respectively. The number of cycles saved each time G_i executes, therefore, is $M(G_i) = SW(G_i) - HW(G_i)$. Our flow does not increase $HW(G_i)$ for any ISE, after the initial scheduling step. To reduce the area overhead, we propose a rescheduling mechanism that tries to uniformly distribute operations of each type across the different time steps of the schedule.

Let $RR[r]$ be the number of instances of resource r allocated thus far, and Reg be the number of registers. When rescheduling G_i, let $C_G[r]$ and R_G respectively be the number of resources of type r and the maximum number of variables *live* at any time step of the current schedule; the *left edge algorithm* [9] computes R_G. For example, if $C_G[r] \leq RR[r]$, then the schedule needs no further instances of r; if $C_G[r] > RR[r]$, then this schedule requires $C_G[r] - RR[r]$ additional instances. Let $\Delta[r] = max\{C_G[r] - RR[r], 0\}$, $\Delta_{reg} = max\{R_G - Reg, 0\}$, and $Area[r]$ and $Area[reg]$ be the respective areas of resource r and reg registers. Then the objective function is computed as follows:

$$A = \left(\sum_{r} \Delta[r] \times Area[r]\right) + \left(\Delta_{reg} \times Area[reg]\right), \tag{1}$$

Once the schedule is computed, the resource count is updated. For each resource r, $RR[r] \mathrel{+}= \Delta[r]$; for the register count, $Reg \mathrel{+}= \Delta_{reg}$.

Let $L(v)$ and $U(v)$ be lower and upper bounds on the time step at which vertex v can be scheduled. Without resource constraints, $L(v)$ and $U(v)$ can be determined from *as-soon-as-possible (ASAP)* and *as-late-as-possible (ALAP)* schedules; register file I/O constraints, however, do not permit this. We keep the schedule of register file reads and writes computed by the initial retiming step; given these constraints, the ASAP and ALAP schedules are then computed.

The time steps at which v can be scheduled are $\{L(v), ..., U(v)\}$. Let $S(v)$ be the step at which v is currently scheduled. If $S(v) > L(v)$, then we can reschedule v at time $S(v) - 1$. A predecessor p of v such that $S(p) = S(v) - 1$ may need to be pushed back as well; the same holds true recursively for p's predecessors. Similarly, v can be moved forward. A *move* operation describes a local perturbation to the current solution of the problem. Our *move* operation randomly selects a vertex v and moves it backward or forward one time step. We do permit moves that increase latency of the ISE, because additional moves may find new minimum-latency solutions that have not yet been explored. Minimum-latency solutions that improve on the area estimate for the current ISE are always accepted. Solutions that do not improve the current one may be accepted based on a probabilistic calculation (the *temperature schedule*). Termination occurs when a sufficient number of random moves fail to improve upon the best solution found so far.

III. Resource Allocation, Binding, and Port Assignment

A. Preliminaries

Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* ($G_1 \approx G_2$) if there exists a bijection $f: V_1 \to V_2$ such that $(v_1, v_2) \in E_1$ if and only if $(f(v_1), f(v_2)) \in E_2$. Graph G_3 is a common supergraph of G_1 and G_2 if it has subgraphs G_3' and G_3'' such that $G_1 \approx G_3'$ and $G_2 \approx G_3''$. Given a weight function W on the vertices and/or edges of the graph, the goal of the *weighted minimum cost common supergraph (WMCS)* problem is to find a common supergraph G_3 of G_1 and G_2 such that $W(G_3)$ is minimum. This problem is NP-complete [2].

Past formulations of this problem start by computing the WMCS of a set of input datapaths [6, 10]; for example, the infeasible solution in Fig. 2(b) is the WMCS of the datapaths shown in Fig. 2(a). The second step modifies the WMCS to respect the input connectivity and (non-)commutativity of each operator. This involves the insertion of muxes, e.g., Fig. 2(c), or partial reallocation and rebinding of resources, e.g., Fig. 2(d). Mux insertion, including port reassignment, is NP-complete for *every* commutative operator [12]. We augment the WMCS problem to include account for port assignment by including connectivity and commutativity information.

B. Problem Statement

Let N_O be the set of supported operations. For $o \in N_O$, $comm(o) = 1$ if o is commutative, and 0 if not. Let CL be a component library containing N_R different resources. For each resource $r \in CL$, the known quantities are: $Area(r)$, $Latency(r)$, and the number of input ports of r, $N_{in}(r)$. Let $OtoR$ be an $|N_O| \times |N_R|$ matrix that maps operations onto the resources that can execute them; $OtoR[x][y]$ is *true* if resource y can execute operation x, and *false* otherwise.

The compiler generates a set of multi-cycle ISEs which are decomposed into a set of single-cycle ISEs $G_1, \ldots, G_N$, in Step 3 of Fig. 1. Each ISE is a directed acyclic graph $G_i = (V_i, E_i, T_i)$: V_i is a set of vertices, E_i is a set of edges, and $T_i: V_i \to N_O$ associates an operation with each vertex. Each edge is a triple (v_1, v_2, p), indicating that the output of v_1 connects to the p^{th} input port of v_2; one incoming edge per vertex connects to each input port.

The output is a datapath $G = (V, E, R)$, where V and E are vertices and edges and $R: V \to \{1, 2, ..., |N_R|\}$ maps vertices onto resource instances. G must be a common supergraph of $G_1, \ldots, G_N$; muxes are derived from the port assignment. For each ISE G_i, $f_i: V_i \to V$ defines an isomorphism from G_i onto a subgraph of G; $f_i(v_j)$ is the vertex in G onto which v_j is mapped. For each vertex $v_j \in V_i$, $OtoR[T_i(v_j)][R(f_i(v_j))]$ is *true*, ensuring that $f(v_j)$ can implement v_j's operation.

Consider vertex $v_j \in V_i$. If $comm(v_k) = false$, then the mapping must preserve port assignment, i.e., for edge $(v_j, v_k, p) \in E_i$, an edge $(f_i(v_j), f_i(v_k), p)$ must exist in E. If $comm(v_k) = true$, then the incoming edges (v_j, v_k, p) and $(v_j', v_k, p') \in E_i$ must map onto distinct input ports of $f(v_k)$. If the edges map onto $(f_i(v_j), f_i(v_k), q)$ and $(f_i(v_j'), f_i(v_k), q') \in E$, then we must enforce the constraint that $q \neq q'$. Vertex $v \in V$ has $N_{in}(R(v))$ input ports. Let $in_p(v)$, $p = 1, 2, \ldots, N_{in}(R(v))$, be the set of incoming edges incident on v that connect to port p. A mux m is needed if $|in_p(v)| > 1$.

The area of an operation $v \in V$ is denoted by $Area[R(v)]$. van der Werf et al. [16] estimated the area of a k-input mux, $A_{mux}(k)$, to be kM, where M is an appropriately chosen constant, if $k > 1$, and 0 otherwise. The area of the muxes on the input ports of vertex v is

$$MA(v) = \sum_{p=1}^{N_{in}[R(v)]} A_{mux}(k), \text{ where } k = |in_p(v)|. \tag{2}$$

The area of the WMCS, including muxes and port assignment, is

$$W(G) = \sum_{v \in V} Area[R(v)] + MA(v). \quad (3)$$

The goal is to find a legal solution that minimizes *W(G)*.

C. Optimal Integer Linear Program (ILP) Formulation

We introduce an optimal algorithm for the problem outlined in the preceding section. Let UB_r be an upper bound on the number of instances of each resource *r* in *G*, i.e., we count the number of vertices across all input ISEs that are compatible with *r*:

$$UB_r = \sum_{i=1}^{N} \sum_{v \in V_i} OtoR[T_i(v)][r]. \quad (4)$$

Let $v_{i,j}$ be the jth vertex of the ith ISE and $s_{r,n}$ be the nth instance of resource *r* in *G*. We introduce a Boolean variable $x_{i,j,r,n}$, which is *true* if vertex $f_i(v_{i,j}) = s_{r,n}$, and is *false* otherwise. Constraints (5) and (6) ensure that each vertex is mapped onto exactly one resource, and that all vertices in each ISE G_i map onto distinct resources in *G*.

$$\sum_{r=1}^{N_R} \sum_{n=1}^{UB_r} x_{i,j,r,n} = 1 \quad \begin{array}{l} \forall i, 1 \le i \le N \\ \forall j, 1 \le j \le |V_i| \end{array}. \quad (5)$$

$$\sum_{j=1}^{|V_i|} x_{i,j,r,n} \le 1 \quad \begin{array}{l} \forall i, 1 \le i \le N \\ \forall r, 1 \le r \le N_r \\ \forall n, 1 \le n \le UB_r \end{array}. \quad (6)$$

Let $v_{i,j} \in V_i$ have type $T_i(v_{i,j}) = o$, where $comm(o) = true$; $v_{i,j}$ has $N_{in}(o)$ incoming edges, one per input port. The mapping of incoming edges to input ports is represented as an $|N_{in}(o)| \times |N_{in}(o)|$ Boolean *permutation matrix B*, where $B[k][p] = 1$ if the kth incoming edge to the pth input port of $v_{i,j}$. In a permutation matrix, each row and column has exactly one entry set to *1*; all other entries are *0*. To enforce this property, we introduce a Boolean variable $e_{i,j,p,q}$ which is *true* if the pth incoming edge to $v_{i,j}$ is connected to the qth input port of $f_i(v_{i,j})$ in *G*.

Constraints (7) and (8) ensure that *G* is a supergraph of each ISE G_i, while ensuring a legal port assignment for commutative operations. Constraint (7) ensures that each incoming edge to a commutative operation connects to one input port; constraint (8) ensures that each input port has one incoming edge connecting to it. To simplify notation, let $N^* = N_{in}(T_i(v_{i,j}))$.

$$\sum_{q=1}^{N^*} e_{i,j,p,q} = 1 \quad \begin{array}{l} \forall i, 1 \le i \le N \\ \forall j, 1 \le j \le |V_i| \\ \forall p, 1 \le p \le N^* \\ comm(v_{i,j}) = 1 \end{array}. \quad (7)$$

$$\sum_{p=1}^{N^*} e_{i,j,p,q} = 1 \quad \begin{array}{l} \forall i, 1 \le i \le N \\ \forall j, 1 \le j \le |V_i| \\ \forall q, 1 \le q \le N^* \\ comm(v_{i,j}) = 1 \end{array}. \quad (8)$$

If $comm(v_{i,j}) = 0$, then an incoming edge connected to the pth input port of $v_{i,j}$ must also connect to the pth input port of $f_i(v_{i,j})$; then we set $e_{i,j,p,q}$ to *1* if $p = q$ and *0* if $p \ne q$; thus, we fix the permutation matrix, rather than permitting the ILP to adjust the port assignment, which is appropriate for non-commutative operations. Let $c_{r,n} = 1$ if at least one operation in one of the ISEs is mapped onto resource $s_{r,n}$, and *0* otherwise; let *V'* be the subset of vertices $s_{r,n} \in V$ having $c_{r,n} = 1$; then

$$c_{r,n} = \left(\sum_{i=1}^{N} \sum_{j=1}^{|V_i|} x_{i,j,r,n} > 0 \right) ? 1 : 0. \quad (9)$$

Let $w_{v,p}$ denote the number of incoming edges at the pth input port of resource $v \in V'$; the equation to compute $w_{v,p}$ is omitted to conserve space. The resource area, *RA*, and mux area, *MA*, are

$$RA = \sum_{r=1}^{N_R} \sum_{n=1}^{UB_r} c_{r,n} \times Area(r), \text{ and} \quad (10)$$

$$MA = \sum_{v \in V'} \sum_{p=1}^{N_{in}(v)} A_{mux}(w_{v,p}). \quad (11)$$

The objective is to minimize $W(G) = RA + MA$.

D. Heuristic

This section presents a heuristic alternative to the ILP formulation described in the preceding section. Like the heuristic of Moreano et al. [10], we build a compatibility graph such that each maximal clique corresponds to a legal binding solution. Additionally, we account for port assignment and the cost of inserting multiplexors during the binding process; and we identify isomorphic subgraphs from the set of ISEs, and operate on the granularity of subgraphs rather than vertices.

Fig. 3 shows pseudocode for the heuristic. In step 1, the WMCS, *G*, is initialized to be empty. Step 2 enumerates the *convex* subgraphs of each input ISE and partitions them into *isomorphic equivalence classes (IECs)* as described by Pothineni et al. [13]. For every pair of vertex *u* and *v* in convex subgraph *S*, every path from *u* to *v* or from *v* to *u* contains only vertices in *S*. A *mergeable class (MC)* is an IEC that contains at most one subgraph per ISE. When an IEC has multiple subgraphs per ISE, we select for inclusion the subgraph that has the largest number of neighboring vertices that have already been mapped onto a resource in *G*. All other subgraphs in the IEC are removed.

The main loop, in lines 3-9, repeatedly selects an MC (step 4) and merges it into *G*, similar to Moreano et al.'s heuristic: a *consistency graph (CG)* is built (step 5), such that each maximal clique corresponds to a legal merging of the selected MC into *G*; the *minimum weighted maximal clique (MWMC)* is the optimal solution at this step. Step 6 computes the MWMC of the CG; any clique-finding method—optimal or heuristic—suffices. Step 7 merges the operations corresponding to the MWMC into *G*. In Step 8, any subgraph containing a vertex that was merged into *G* is removed from its IEC, which ensures that each ISE vertex is bound to one resource; empty IECs are discarded. The process repeats until all vertices in each ISE are merged into *G*. The last step is to generate the datapath from *G*, including the muxes, which are derived from the port assignment.

Moreano et al. [10] also build a consistency graph and compute its MWMC; however, they do not perform port assignment. Port assignment increases the size of the CG, as different vertices must be inserted corresponding to different port assignments. We mitigate this through the use of IECs, so that we perform resource allocation and binding on the granularity of IECs rather than individual operations.

Let $G = (V, E)$ be the partially allocated WMCS. Let us denote each MC with a graph $G_M = (V_M, E_M)$. Let $S_1, \ldots, S_n$, $S_i = (V_i, E_i)$ denote the subgraphs in the MC, and let $f_i : V_i \to V_M$ define an isomorphic mapping of S_i onto G_M. To simplify notation, we introduce a naming convention to make the mapping explicit. We let $K = |V_M|$, $V_M = \{v_1^{(M)}, \ldots, v_K^{(M)}\}$, and, for $i = 1$ to n, $V_i = \{v_1^{(i)}, \ldots v_K^{(i)}\}$, where $f_i(v_j^{(i)}) = v_j^{(M)}$ for each vertex $v_j^{(i)} \in V_i$. This notation explicitly defines the mapping through the subscripts of the vertices.

```
1.  Initialize the WMCS, G, as an empty graph
2.  Compute Isomorphic Equivalence Classes (IECs)
3.  Do {
4.      Select the best mergeable class (MC)
5.      Construct the consistency graph (CG) for the MC.
6.      Find a Max. weighted Max. clique (MWMC) in the CG
7.      Merge the operations in the clique into G
8.      Update the set of IECs
9.  } While there are unmapped operations
10. Generate muxes/control signals and convert G to VHDL
```

Figure 3. High-level pseudocode for our resource allocation, binding, and port assignment heuristic. The critical step is to construct the consistency graph and the bulk of the runtime is spent on clique identification.

Each set of vertices $C_j = \{v_j^{(1)}, \ldots, v_j^{(n)}\}$ forms a *compatible group*: we commit, up front, to bind all of these vertices to some vertex v in G. This decision reduces the space of all possible binding solutions, but is intuitively justifiable due to the subgraph isomorphism relation between the subgraphs in the MC. We impose the condition that $OtoR[v_j^{(i)}][R(v_j^{(M)})] = 1$ for $i = 1$ to n, to ensure that the resource type of $v^{(M)}$ is compatible with each operation $v_j^{(i)} \in C_j$. To reduce notational clutter, we drop the superscript $^{(M)}$ in the following discussion.

During the first iteration, G is empty, so G_M is copied into G. During subsequent iterations, there are two possibilities for each vertex $v_i \in V_M$: either v_i is bound onto a vertex already allocated to G, or onto new resource r that is allocated to G. Mapping onto previously allocated vertices is preferable, as each new resource that is allocated increases the area of the resulting datapath. Figs. 4 and 5 show a running example that illustrates the merging process. In Fig. 4, G_M is the graph representing the current MC that we intend to merge into a partially allocated datapath G. We also show three potential new resources, each of which *may* be added to G. If a vertex in G_M is bound to one of the new resources, then the resource is added to G; otherwise, it is discarded.

To merge G_M with G, we construct a weighted *consistency graph (CG)*, $G_C = (V_C, E_C, w)$. V_C contains vertices that correspond to binding solutions for vertices and edges, including port assignment; edges in E_C are placed between partial binding solutions that are *consistent* with one another; and w assigns weights to vertices in V_C in accordance with the area overhead resulting from the binding.

During binding, we construct a subgraph isomorphism mapping from G_M onto G. Let $f_V: V_M \rightarrow V$ denote the vertex mapping, and $f_E: E_M \rightarrow E$ denote the edge mapping. Let us consider a vertex $v_i \in V_M$. For each vertex $v \in V$ such that $OtoR[v_i][R(v)] = 1$, we add a vertex v_C to the consistency graph to represent the possibility of binding v_i onto v; $w(v_C)$ is set to 0 because binding v_i to v does not increase the area. The CG in Fig. 5 contains two such vertices labeled with mappings $f_V(v_1) = v_4$, $f_V(v_3) = v_6$. We also consider the possibility of binding v_i to a new resource that we must allocate to G. We add a vertex r_i to V_C to model this possibility: $w(r_i) = Area(R(v_i))$, indicating the area overhead of instantiating a new resource. The CG in Fig. 5 contains three such vertices labeled with mappings $f_V(v_i) = r_i$, $1 \leq i \leq 3$.

Moreano et al.'s [10] CG contains all the vertices described above. Our CG includes additional vertices to facilitate different possible port mappings. Consider edge $e = (v_i, v_j, p) \in E_C$. Let $CR(v_i)$ and $CR(v_j)$ be the respective sets of vertices in V onto which v_i and v_j may map. In Fig. 4, $CR(v_1) = \{v_4, r_1\}$, $CR(v_2) = \{r_2\}$, and $CR(v_3) = \{v_6, r_3\}$. We also need to consider the possible port mappings for this edge. If v_j is commutative, then $CP(p) = \{1, 2, \ldots p\}$; otherwise, $CP(p) = \{p\}$. The space of potential mappings for e is $CR(e) = CR(v_i) \times CR(v_j) \times CP(p)$.

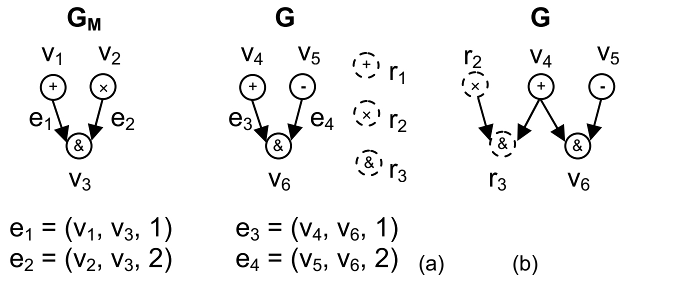

Figure 4. A representing a mergeable class, G_M, a datapath, G, along with a set of potentially new resources to add to G; the goal is to find an allocation and binding solution by merging G_M into G (a). The resulting datapath after merging (corresponding to the MWMC of the consistency graph shown in Fig. 4 (b).

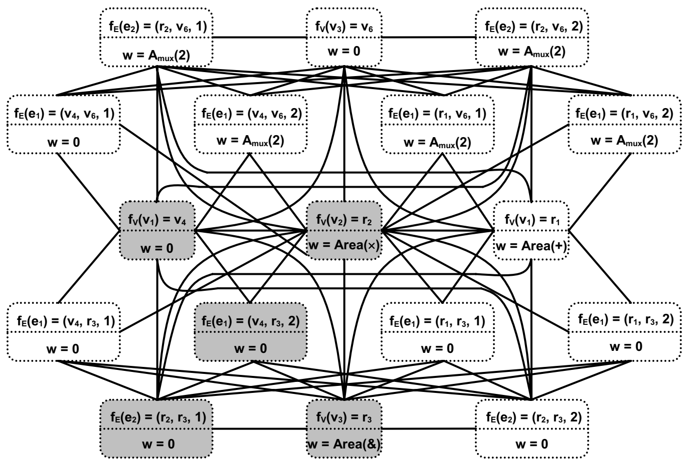

Figure 5. A consistency graph (CG) corresponding to the problem instance shown in Fig. 4(a). The MWMC, shown in gray, corresponds to the solution shown in Fig. 4(b).

For each triplet (u_i, u_j, q) such that $u_i \in CR(v_i)$, $u_j \in CR(v_j)$, and $q \in CP(p)$, we instantiate a vertex v_C in V_C. If an edge (u_i, u_j, q) already exists in E, then $w(c) = 0$, as e would be bound to a pre-existing connection; otherwise, we must connect an additional wire to the q^{th} input port of u_j, increasing the size of the multiplexor on the port. To reflect this cost, we set $w(v_C) = A_{mux}(|in_p(u_j)| + 1) - A_{mux}(|in_p(u_j)|)$. These vertices are *not* present in Moreano et al.'s CG. The CG in Fig. 5 contains twelve vertices corresponding to edge/port mappings.

The last step instantiates a set of CG edges, E_C, between vertices that correspond to consistent binding solutions. An edge (u_C, v_C) is added to E_C if the binding implied by u_C and v_C satisfy four criteria.

First, each vertex(edge) in G_M can map onto one vertex/edge in G. For example, in Fig. 5, the four CG vertices labeled $f_E(e_1) = \ldots$ are inconsistent, as e_1 can only be mapped onto one potential edge in G.

Second, the vertex, edge, and port mappings must be consistent. For example, consider vertex v_1 and edge $e_1 = (v_1, v_3, 1)$. The CG vertices in Fig. 5 labeled $f_V(v_1) = v_4$ and $f_E(e_1) = (r_1, v_6, 1)$ are inconsistent, as the latter implies that v_1 is bound to r_1, rather than v_4.

Third, every pair of incoming edges to distinct input ports of vertex v_k must map onto distinct input ports of $f_V(v_k)$. For example, consider edges $e = (v_i, v_k, p)$ and $e' = (v_j, v_k, p')$ in G_M and potential bindings $f_E(e) = (f_V(v_i), f_V(v_k), q)$ and $f_E(e') = (f_V(v_j), f_V(v_k), q')$; this binding is consistent if and only if $q \neq q'$; otherwise, $f_V(v_i)$ and $f_V(v_j)$ would connect to the same input port, which would be inconsistent.

Fourth, if $comm(v_j) = false$, then for each incoming edge $e = (v_i, v_j, p), f_E(f_V(v_i), f_V(v_j), q)$ must satisfy the property that $q = p$. Otherwise, the binding could, for example, transform $A - B$ into $B - A$.

Each legal binding solution corresponds to a maximal clique in G_C. The goal is to find the mapping that minimally increases the area of G, i.e., the maximum clique of minimal weight. We solve this problem with *Cliquer* [11], an open source tool that solves the problem optimally in exponential worst-case time. During our experiments, we found that the runtime of our heuristic approach was reasonable in comparison to other techniques.

IV. EXPERIMENTAL RESULTS

A. Experimental Platform

We implemented our design flow in an internal research compiler for extensible processors. Our target processor is a RISC with $R = 2$ read ports and $W = 1$ write ports. We identify ISEs having up to *5* inputs and *2* outputs [15]. Our benchmarks are cryptography, signal processing, and multimedia applications taken from established suites.

We selected six embedded benchmarks for use in our study; these benchmarks are known to benefit significantly from acceleration through ISEs, making them reasonable candidates for our study. We include two signal processing applications, *adpcm* and *gsm*, *jpeg* encoding, and three cryptography benchmarks: *des, sha*, and *aes*. In later experiments, we also include *H.263* videoconferencing and the *idct* kernel of jpeg decompression.

Resources, input datapaths, and MDs were all synthesized using *Synopsis Design Compiler* with an *Artisan* 0.18μm CMOS standard cell library. The area and delay of each resource were obtained from VHDL model. Our results include placement and routing.

B. Single-Cycle ISEs

Our first experiment treats each ISE as a single-cycle operation, ignoring the I/O constraints of the register file; we compare our ILP and heuristic methods for resource allocation and binding with prior DPM-based methods by Moreano et al. [10] and Brisk et al. [1]. The purpose of this experiment was to select the best approach to include in Step 4 of the synthesis flow shown in Fig. 1. Fig. 6 presents the area savings, relative to the baseline, of the four techniques evaluated here. Area savings is represented as percentages, so a larger area savings implies a smaller datapath.

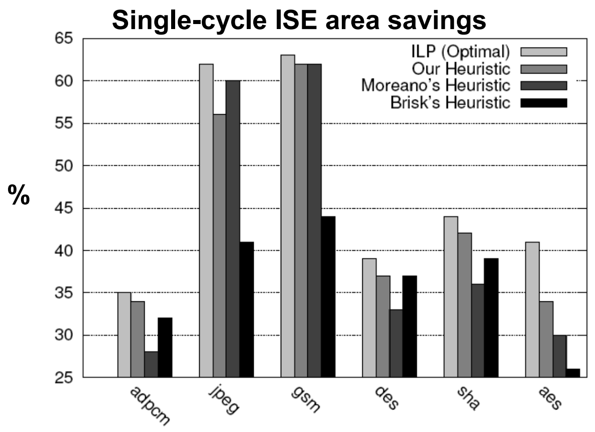

Figure 6. Area savings achieved by four resource allocation, binding methods for single-cycle ISEs.

In all cases, the ILP terminated and found the optimal solution; the heuristics found sub-optimal solutions in all cases. Our heuristic achieved equal or greater area reductions than Moreano et al.'s for all benchmarks other than *jpeg*. For *jpeg* the difference in solution quality was due to ordering affects, i.e., different selection criteria for choosing a mergeable class at each step (line 4 of Fig. 3) could have improved the solution. Except for *des* and *sha*, Brisk et al.'s heuristic was not competitive in terms of area savings. Its goal is to prevent false loops from forming; as a consequence, it is more conservative in the binding decisions it makes, and tends to allocate more resources.

In this experiment, we did not attempt to prevent the critical path delay of the resulting datapath from increasing. For our heuristic, the increases in critical path delay, for each benchmark were: *1.08* (*jpeg*), *1.11* (*gsm encode*), *1.16* (*adpcm*), *1.18* (*sha*), *1.35* (*aes*), and *2.09* (*des*). These estimates include false paths, which were not removed during synthesis. Aside from false paths, the other contributing factor to the critical path delays is the muxes that must be inserted.

The runtimes of the different approaches are as follows: the ILP ranged from *3* to *8* hours; our heuristic ranged from *2* to *10* minutes, with approximately *1* minute dedicated to subgraph enumeration and IEC generation; Moreano et al.'s heuristic ran in approximately *1* minute; and Brisk et al.'s heuristic ran in *5-10* seconds or less. Altogether, we conclude that our heuristic offers the best quality solution with a reasonable runtime, i.e., on the order of minutes.

C. Multi-Cycle ISE Synthesis Flow

Next, we examine the efficacy of the proposed synthesis flow for multi-cycle ISEs shown in Fig. 1. We assume that the target processor has a frequency of $1/\Lambda = 200$ *MHz*. Using the method of Pozzi and Ienne [15], we searched for ISEs having up to *5* inputs and *2* outputs; as the processor has $R = 2$ read ports and $W = 1$ write ports, the largest ISEs found had to be serialized to form multi-cycle ISEs. We considered four different synthesis flows:

Flow 1 (Baseline): The baseline flow performs retiming and I/O-constrained scheduling, but does not perform resource allocation and binding. This flow is equivalent to Fig. 1 with steps 2-4 disabled.

Flow 2: The second flow assumes that each ISE takes one cycle, effectively setting the clock period constraint Λ to infinity. This flow demonstrates that multi-cycle ISEs can save area over single-cycle ISEs when operations in different cycles share resources.

Flow 3: The third flow is the complete synthesis flow shown in Fig. 1 using our heuristic for Step 4.

Flow 4: The same flow as F3, but without the rescheduling step.

Flows 1, 3 and 4 achieve comparable performance in terms of latency and critical path delay; because the resulting datapaths yield the same performance, we evaluate only the area of the resulting datapath. As Flow 2 is unrealistic, it is used solely for illustrative purposes.

Fig. 7 shows the area reduction achieved by Flows 2 and 3 in comparison to the baseline (*0%*). For six of the eight benchmarks, Flow 3 achieved a greater area reduction than Flow 2. For *adpcm* and *des*, the overhead of the registers introduced by Flow 3 was greater than the area savings achieved through binding operations in different cycles of the same ISE to the same resource. This experiment demonstrates that the use of multi-cycle ISEs can reduce the area of the resulting datapath compared to large single-cycle ISEs.

Fig. 8 decomposes the area of the datapaths generated by Flows 2-4 into computational resources, registers, and muxes. The inclusion of Flow 4 lets us evaluate the efficacy of the rescheduling step in Step 2 of Fig. 1. Results are normalized to Flow 2 at 100%.

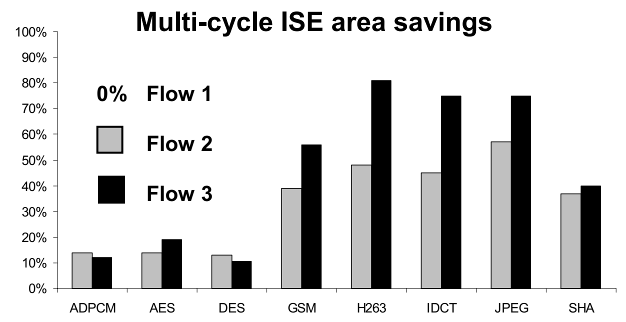

Figure 7. Area savings achieved by ISE synthesis Flows 2, and 3; Flow 1, the baseline, achieves no reduction in area.

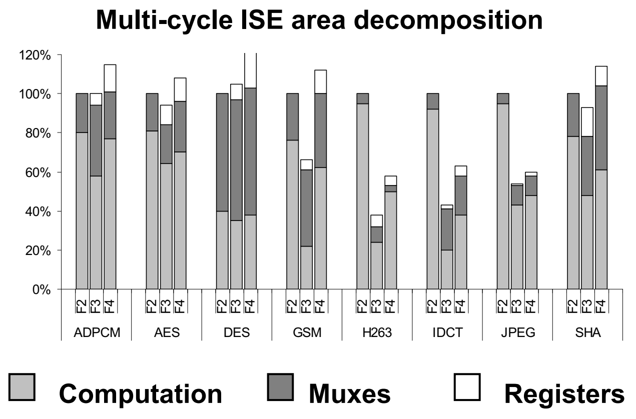

Figure 8. Decomposition of the area of the datapaths generated by Flows 2, 3, and 4.

On average, including the rescheduling step (Flow 3) yields a *14%* area reduction compared to excluding it (Flow 4). A reduction in computation resources is observed across all benchmarks; in the case of H.263, Flow 4 uses less multiplexor area than Flow 3. In this case, replicating resources reduced the amount of multiplexing needed. Additionally, Flow 4 often required more register area than Flow 3. Intuitively, internal steps in a multi-cycle ISE will store some data locally in datapath registers, rather than writing them back to the processor's register file. Increasing the number of computational resources available, and using them at every step, will increase the amount of data stored internally after each cycle, so more registers are required. As Flow 2 assumes a single cycle, it requires no registers.

A*es*, *des*, *sha*, and *adpcm* are dominated by logical operations whose areas are small; they do not benefit much from aggressive resource allocation binding, so Flow 3 offers a marginal advantage over Flow 4. The other benchmarks contain more arithmetic operations and, therefore benefit more significantly from our flow.

V. Related Work

We formulated resource allocation, binding, and port assignment for ISE synthesis as a DPM problem. In high-level synthesis, different formulations are often used. Allocation and scheduling are often performed first, and operations are bound onto a set of pre-allocated resources [5, 8]. The port assignment can then be optimized after the fact [3]. In our context, resource allocation and binding are performed concurrently, after scheduling.

Most DPM methods begin by computing the WMCS and insert muxes afterwards [1, 6, 10, 18]. Minimizing the area of the muxes inserted for each binary commutative operator is NP-complete, per operator [12]. Our resource approach extends these techniques to accurately account for the cost of mux insertion.

Our approach is an extension of Moreano et al.'s [10] DPM heuristic. Our experiments show that the inclusion of port assignment can reduce area. A similar approach was taken by de Souza et al. [6], who used DPM to reduce reconfiguration time; their priority focused on interconnect sharing in FPGAs. We believe that our approach could also improve de Souza et al.'s design flow as well.

Geurts [7] presented a DPM heuristic based on maximum bipartite matching; van der Werf et al. [16] built a similar heuristic, but used simulated annealing instead. These approaches estimate an upper bound on the cost of muxes, but do not model the cost exactly.

Brisk et al. [1] and Zuluaga and Topham [18] developed DPM heuristics for DAGs. They decompose each DAG into paths and compute the WMCS via string matching. Their methods do not share resources as aggressively as do ours, because their concern is to prevent false loops from occurring in the resulting datapath.

VI. Conclusion

We have introduced a synthesis flow for ISEs for extensible processors. We have shown that two scheduling steps should be used: one to determine the minimum latency schedule, given I/O constraints on the processor's register file, and a second to reduce the number of resources required to achieve a minimum latency schedule. We also investigated new techniques for resource allocation, binding, and port assignment in this context. We have determined that these problems are best formulated as instances of DPM, and that including port assignment in the formulation significantly reduces area.

References

[1] Brisk, P., Kaplan, A., and Sarrafzadeh, M. Area-efficient instruction sets synthesis for reconfigurable system-on-chip designs. In proc. DAC (San Diego, CA, USA, June 7-11, 2004) 395-400.

[2] Bunke, H., Guidobaldi, G., and Vento, M. Weighted minimum common supergraph for cluster representation. In proc. ICIP, v. 2 (Sept. 14-17, 2003) 25-28.

[3] Chen, D., and Cong, J. Register binding and port assignment for multiplexer optimization. In proc. ASPDAC (Yokohama, Japan, Jan. 27-30, 2004), 68-73.

[4] Clark, N., Zhong, H., and Mahlke, S. A. Processor acceleration through automated instruction set customization. In proc. MICRO (San Diego, CA, USA, Dec. 3-5, 2003), 129-140.

[5] Cong, J., and Xu, J. Simultaneous FU and register binding based on network flow method. In proc. DATE (Munich, Germany, March 10-14, 2008) 1057-1062.

[6] de Souza, C. C., Lima, A. M., Araujo, G., and Moreano, N. The datapath merging problem in reconfigurable systems: complexity, dual bounds, and heuristic evaluation. ACM Journal of Experimental Algorithms 10 (2005).

[7] Geurts, W. Accelerator Data-Path Synthesis for High-Throughput Signal Processing Applications. Kluwer Academic Publishers, 2007.

[8] Huang, C-Y., Chen, Y-S., Lin, Y-L., and Hsu, Y-C. Data path allocation based on bipartite weighted matching. In proc. DAC (Orlando, FL, USA, June 24-27, 1990) 499-504.

[9] Kuradhi, F. J., and Parker, A. C. REAL: a program for REgister ALlocation. In Proc. DAC (Miami Beach, FL, USA, June 28 - July 1, 1987), 210-215.

[10] Moreano, N., Borin, E., de Souza, C. C., and Araujo, G. Efficient datapath merging for partially reconfigurable architectures. IEEE Trans. CAD 24, 7 (July, 2005), 969-980.

[11] Niskanen, S., and Ostergard, P. R. J. Cliquer user's guide, version 1.0. Technical Report T48, Communications Laboratory, Helsinki University of technology, Espoo, Finland, 2003.

[12] Pangrle, B. M., On the complexity of connectivity binding. IEEE Trans. CAD 10, 11 (Nov. 1991), 1460-1465.

[13] Pothineni, N., Kumar, A., and Paul, K. Exhaustive enumeration of legal custom instructions for extensible processors. In proc. VLSI Design (Hyderabad, India, January 4-8, 2008), 348-353.

[14] Pozzi, L., Atasu, K., and Ienne, P. Exact and approximate algorithms for the extension of embedded processor instruction sets. IEEE Trans. CAD 25, 7 (July, 2006), 1209-1229.

[15] Pozzi, L., and Ienne, P. Exploiting pipelining to relax register-file port constraints of instruction-set extensions. In proc. CASES (San Francisco, CA, USA, Sept. 24-27, 2005) 2-10.

[16] van der Werf, A., et al. Area optimization of multi-functional processsing units. In proc. ICCAD (San Jose, CA, USA, Nov. 10-14, 2002) 292-299.

[17] Verma, A. K., Brisk, P., and Ienne, P. Rethinking custom IE identification: a new processor-agnostic method. In proc. CASES (Salzburg, Austria, Sept. 30 – Oct. 3, 2007) 125-134.

[18] Zuluaga, M., and Topham, N. Resource sharing in custom instruction set extensions. In proc. SASP (Anaheim, CA, USA, June 8-9, 2008).