Scheduling for Design Reuse of Datapath Components*

Roger Ang and Nikil Dutt

Technical Report 94-16
revised May 10, 1994

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-8059

rang@ics.uci.edu

Abstract

Traditional High-Level Synthesis (HLS) techniques do not allow reuse of complex, realistic datapath components during the tasks of scheduling and allocation. However, such datapath components are often custom designed and placed in technology libraries and databooks for potential reuse in future designs. We present a novel scheduling approach that, for the first time, permits reuse of such datapath components during HLS. Given a library of user-defined datapath components, an allocation of components from this library, and a limit on the maximum propagation delay through the datapath components, our algorithm generates an effective schedule for a given input behavior. We present experimental results of our approach on some HLS benchmarks using realistic combinatorial datapath components. Since the scheduling technique works on user-defined templates of datapath components and uses the delay information from the library for the components, we believe our approach can significantly impact design productivity through the reuse of complex RT datapath components.

*This work was supported by SRC contract 94-DJ-146 and a UCI GPOP fellowship.

Contents

1 Introduction and Problem Definition

Current approaches to High-Level Synthesis (HLS) typically generate a datapath as a netlist of generic RT components, plus a symbolic Finite-State Machine (FSM) that sequences this datapath (Figure 1a). The symbolic FSM and the generic datapath netlist is then refined through RT- and logic synthesis, technology mapping, and physical design for implementation. Such an approach simplifies HLS algorithms since the generic datapath component models often assume 2-input, single output semantics (e.g., simple adder or subtractor) to match the semantics of popular, imperative languages such as VHDL.

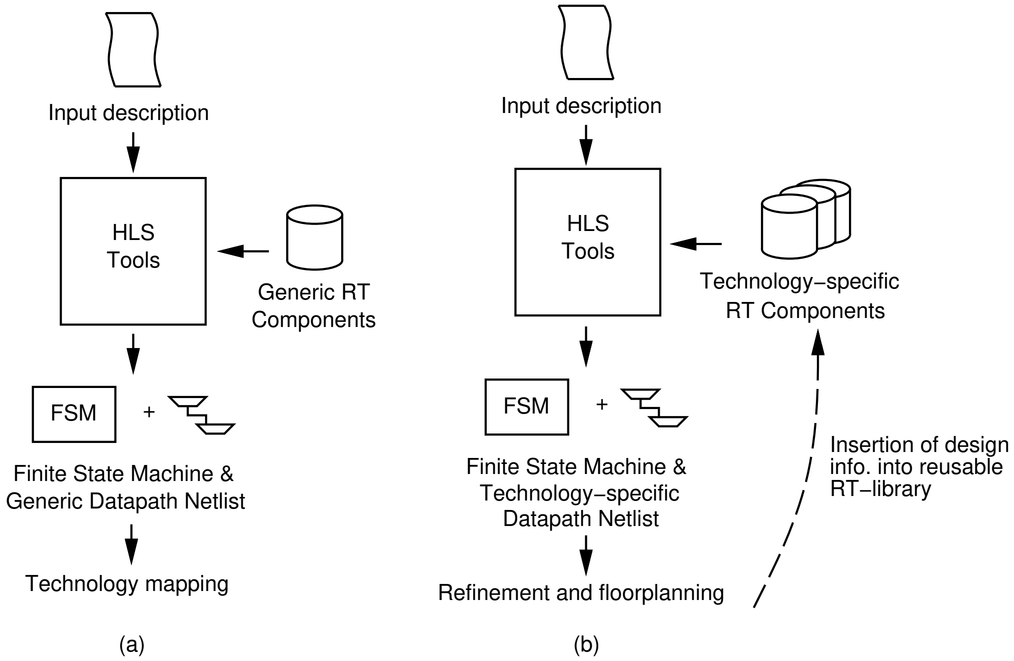

Figure 1: (a) Conventional HLS with generic components, (b) Synthesis for design reuse.

However, datapath components often have more complex semantics and exhibit RT-level parallelism where several outputs (e.g., ALU sum, carry, status bits) are generated *simultaneously* for several inputs (e.g., ALU's 2 data inputs and a carry-in). Furthermore, such datapath components are often hand-optimized for performance, area, power or other design criteria within specific design projects; these designs are then characterized as custom components into a technology library to enable reuse for future design projects.

Existing HLS systems are unable to allow effective reuse of such technology-specific datapath components since the algorithms for scheduling, allocation and binding cannot handle the design model for these reusable, technology-specific custom datapath components. The inability to reuse such datapath components during HLS is a serious problem, since the resulting (simple) generic component models have to be mapped to more complex reusable parts *after* all the HLS decisions have been made. This may not only result in inefficient or infeasible design implementations, but also prohibits the exploration of a larger design space that effectively uses such custom datapath components.

Currently, most approaches towards high-level synthesis (HLS) of synchronous circuits from hardware description languages (HDLs) assume a simple model of register-transfer (RT) components. Synthesis tools often assume an abstract operator directly maps to a single RT operation which in turn can be performed by one or two components. For example, an abstract addition, such as + in an HDL, will map to an *add* operation which can be performed by a RT-component such as an Adder or ALU. However, these assumptions prevent effective usage of databook components, customized cells and modules, and RT module generators.

Since there is a semantic mismatch between the multi-function, multi-output RT components and the single-function-single-output HDL operators, current HLS tools typically ignore some of the functionality of such components. For example, consider the adder shown in Figure 2a. The behavior of the adder is described abstractly in Figure 2b. If this description were given to a typical HLS scheduling tool with an allocation of a single adder, it would likely schedule the two additions in two consecutive states even though the adder could perform both additions in one state. This is why reuse of existing modules is inefficient in current HLS tools.

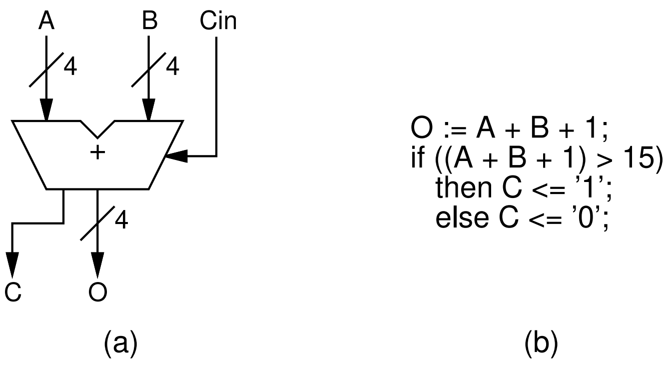

Figure 2: Adder: (a) RT component, (b) behavior in VHDL.

The above problem demonstrates that difficulties with reusing existing RT components in HLS are due to the models and assumptions generally used in traditional HLS approaches. An alternative design flow shown in Figure 1b illustrates how HLS tools can permit reuse of user-defined datapath components from a technology-specific library. In order to enable reuse of components in HLS new models and techniques based on a different set of assumptions must be created.

This paper describes a scheduling technique for design reuse of realistic datapath components that uses an alternate model for RT component functionality, and takes advantage of the delay information available from the existing RT components. Specifically, we describe a scheduling technique for reuse of existing combinatorial RT units. Given a library of components, an allocation of components from that library, and a limit on the permitted propagation delay through the datapath components, our scheduling algorithm quickly determines an effective schedule for a given behavior.

The rest of this paper is organized as follows. Section 2 describes previous and related work. Section 3 presents our design models and representations. Section 4 describes our scheduling algorithm. Section 5 presents experimental results on some standard HLS benchmarks to illustrate the versatility of our approach. Section 6 concludes with a summary.

2 Related Work

Much work has already be done on the task of scheduling in HLS (i.e., mapping abstract operations to time steps). However, traditional scheduling algorithms assume a direct mapping of operators (only 1 operator can be performed on an RT unit at any time) or that all operators execute in a single time step (1 operator to 1 RT unit per state) [Camp91] [PLNG90] [PaKn89] [PaGa87] [PaPM86]. The scheduling problem for realistic datapath components may have multiple operators assigned to the same RT unit during the same time step (state). This fundamentally changes how the scheduling problem must be formulated. As a result, our approach is not comparable to previous work in scheduling for several reasons:

- Determining the mapping of HDL operations to RT units is not trivial. A significant amount of searching can be involved in determining how operations can be "clustered" to be performed on a given set of RT Units.

- No assumptions can be made about the duration of HDL operations. It is possible that the function described by a cluster of HDL operations can be performed by several different types of units or different implementations of the same unit type, each with different delays for the function. For example, the behavior in Figure 2b may be performed by an Adder, Adder/Subtractor, or ALU, and each of these units may be implemented with a ripple-carry or carry-lookahead structure. Consequently, it is possible to get six different delays for the statement **O := A + B + 1;**.

- Since it is possible that a given RT unit can perform multiple functions simultaneously, determining how functions can be mapped to RT units becomes a complex problem itself.

As we have pointed out, by assuming a direct mapping of operators to RT units, the scheduling algorithms above implicitly assume that the RT units used only have a single data output. In contrast, other representations for multi-I/O RT units have been proposed. [KnWi92] describes models for representing behavior, timing, and RT structures, and the links between them. These models were general enough to deal with multi-function, multi-I/O RT units, and was used in an interactive RT-level synthesis system. However, no algorithms or tools for scheduling were ever developed for this system. Also, the input language for this system (similar to Lisp) had fundamentally different operation semantics than languages like VHDL or Verilog®. Translating from another intermediate format (like a Control/Data Flow Graph) to the behavior format of this system is impractical, and it is unclear how models for existing RT components can be used effectively in this system. In [WPAV92], Signal Flow Graphs (SFGs) are used to assign behavior to a single processing unit (PU). The PU may consist of several RT units and the behavior of a single SFG is performed by the PU in a single step. However, this is a different problem than the task of scheduling, since there are no dependency relationships between SFGs, and there are no concepts of time (or states) in this model. SFGs may possibly be used for the scheduling problem but the extensions necessary to make that possible are not described.

At a conceptual level, the problem of mapping an HDL model to a set of RT units has similarities to mapping a programming language to a microprocessor. Therefore, we are attempting an approach analogous to the approach described in [GlGr78]. This approach described a method that used a table to map intermediate code to a specific microcode implementation on a fixed, single processor architecture. This approach has been applied to HLS in [LMPa94], which describes instruction-set matching and selection for retargetable code generation for DSPs and Application Specific Instruction-set Processors. However, the problem we are dealing with is larger in scope. For HLS, the problem would be analogous to performing the same tasks for multiple processors, where each processor may have a different instruction set.

For synthesis, [Kahrs86] expanded the idea of using a table to map instructions to RT structure. Instead of a table, this work proposes a library of "parts." Each part may be able to perform several functions and each function has a subgraph representation. The algorithm proposed in [Kahrs86] used graph matching to map behavior to generic parts to be generated later. Along this same line of thinking, [RuGB90] described a "unit table" which is used to map mutually exclusive operators to multi-function units. [RaKu92] proposed a "template" library and a regularity extraction algorithm that may be used for synthesis. This idea of "template" mapping, usually formulated as graph matching, has also been used in HLS in various ways. For example, [CKPR93] and [LMPa94] used templates for mapping instruction sets to specialized processors, and [GCDM93] and [Mar93] used templates to map algorithms to predefined RT structures. But in all of these methods, the libraries and models are not sufficiently defined to deal with components that generate multiple data outputs. In our work, we use the idea of having a component library with behavior templates for the components, define this library for multi-function, multi-output components, and demonstrate how such a library may be used to schedule a given behavioral description with a given set of component instances.

3 Models and Representations

3.1 RT Data Flow Graph (RTDFG)

In [AnDu93], we presented a representation capable of describing the mapping of behavioral constructs to multi-function, multi-output RT components. This representation encompassed the mappings between three sets of entities:

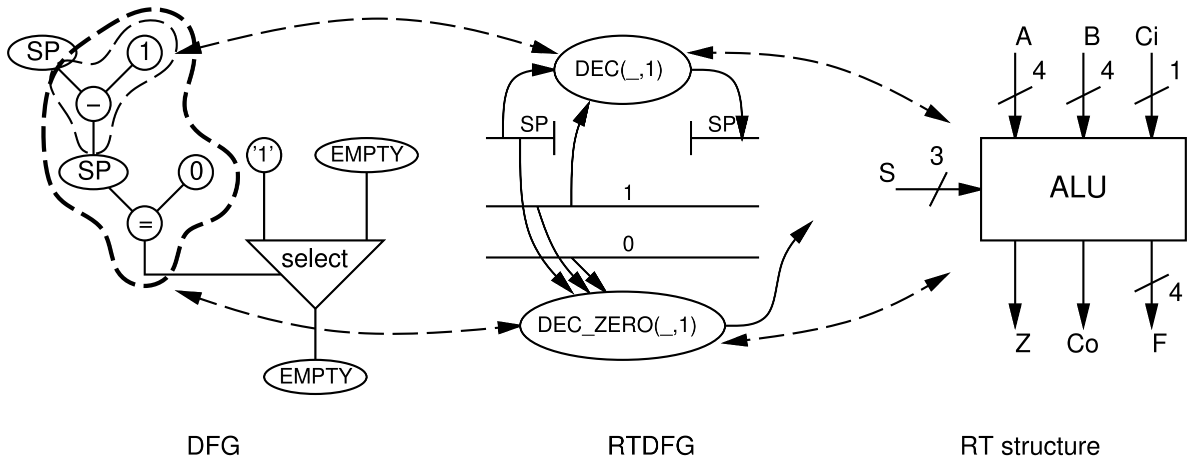

Figure 3: Links between behavior represented as a CDFG, RTDFG transformation nodes, and RT structure.

1. Intermediate behavioral representation — the internal representation used by a synthesis tool for circuit specifications, e.g., a Control/Data Flow Graph (CDFG). This is typically derived from the parsing of an HDL.

2. RT Data Flow Graph (RTDFG) — a combination of a global data flow graph and a state transition graph. However, data transformations in this graph are based on *RT functions*, which are functional abstractions for each data output of the components.

3. RT structure — instances of components. Each RT component is characterized by sets of RT functions that describe modes of operation for the RT component.

In the RTDFG, each node of the graph has an associated RT function. Each node can be matched to a "cluster" of HDL behavior, as well as linked to an instance of an RT component to associate a component operation to that unit and data values to its pins. Figure 3 illustrates part of a single state in a design. The RT functions DEC and DEC_ZERO have been bound to an ALU indicating both functions can be performed simultaneously on that unit.

An important part of this representation is the *behavior templates* (see Figure 4). These templates are used to specify how the abstract behavior (e.g., data flow graphs) can map to the functionality of the RT units. In contrast to traditional HLS techniques, clusters of HDL behavior can map to a single operation of an RT unit. For this work, we are limiting the RT units we are working with to combinatorial circuits. This simplifies behavior matching but is general enough to demonstrate the effectiveness of our approach.

3.2 Template Parse Tree

A fundamental task in our approach is matching an arbitrary cluster of behavior to a behavior template. To do this efficiently, we build a parse tree of the behavior templates (see Figure 5). The first level of internal nodes of the tree describe the outputs of the templates. The other internal nodes of the tree are operators. Edges from an internal node indicate alternative inputs for the operator. Each alternative is a set of inputs, in this case input pairs. This arrangement maintains the context of the templates, i.e., the particular combinations of inputs and operations. Input pairs that have leaf nodes are labeled with the RT functions that can perform that template. For flexibility, we consider operator commutativity in the matching process. During matching, we also attempt to "jump over" temporary variables, i.e., an assignment to a variable with only a single usage. In this way, it is possible for a template match to "cover" multiple assignment statements of the original behavior description (see Figure 6).

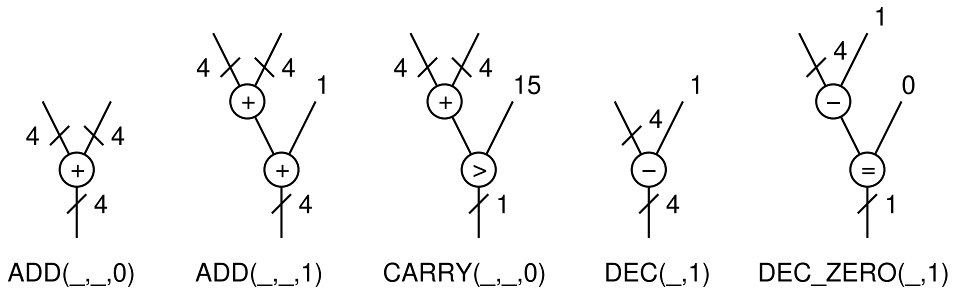

Figure 4: Example behavior templates of RT functions.

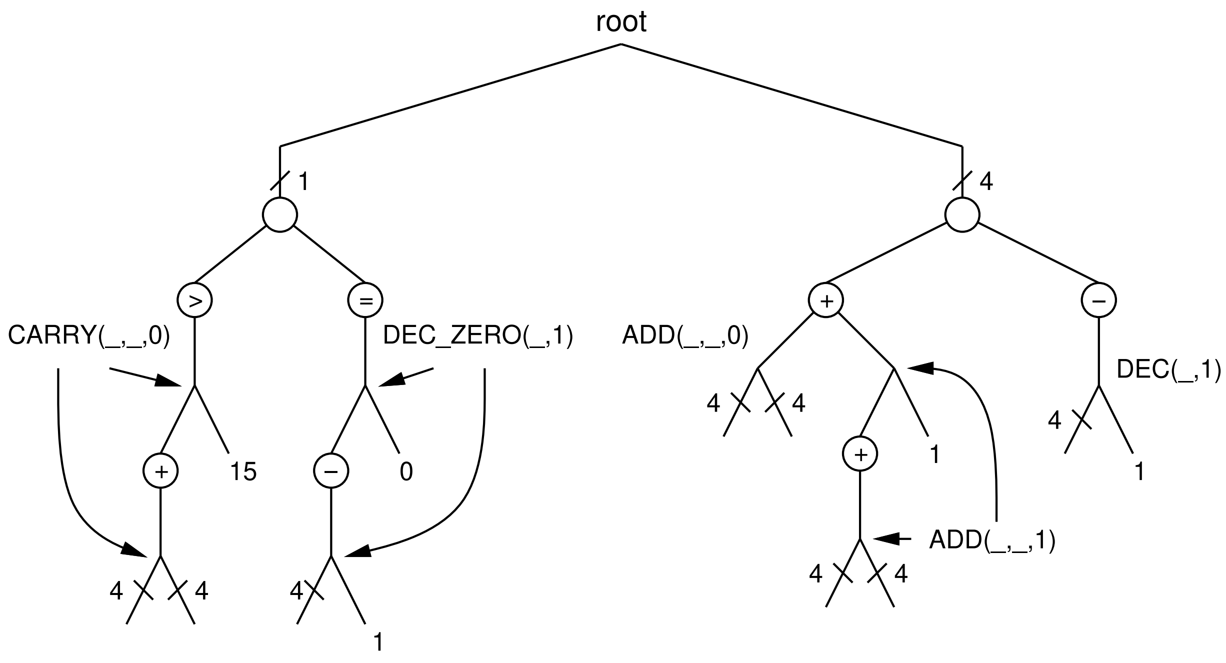

Figure 5: Parse tree for example templates.

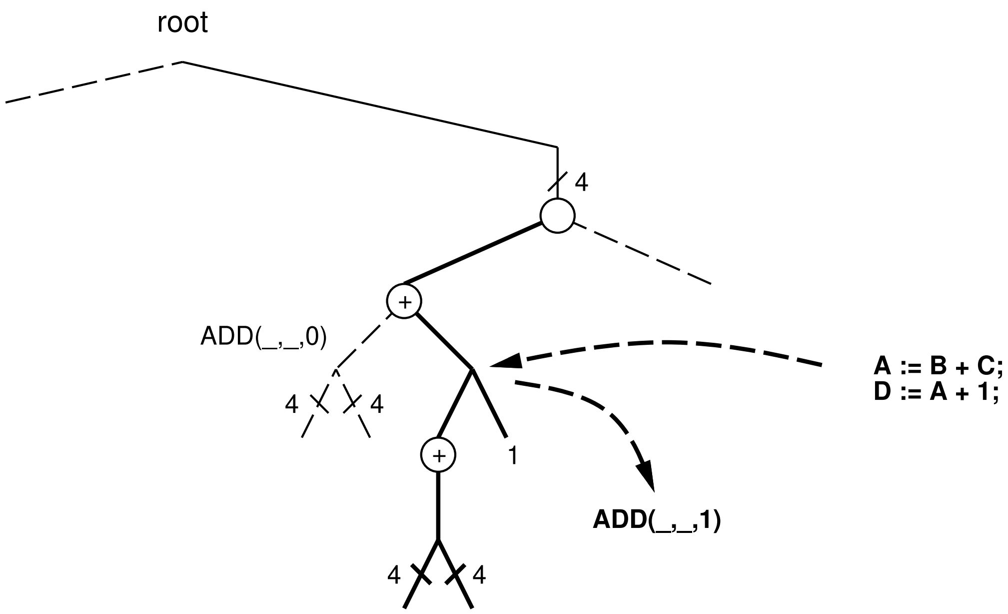

Figure 6: Example parse tree match.

4 Scheduling Algorithm

For our scheduling algorithm, we assume we are given the following:

- A basic block of behavior — a "straight-line" sequence of assignments, which has been parsed into an intermediate representation. Data-dependency analysis has been done on these assignments.
- A library of RT components with
  - RT functions with behavior templates.
  - A mapping of RT functions to RT component types.
  - Input-to-output delays for each RT function of a component.
- An allocation of components from the library — types and maximum number of each type of component.
- The maximum datapath propagation delay for a state. We are only considering the functional units in the datapath. Thus, this limit only refers to the data propagation delay associated with functional units in the datapath, not delays associated with interconnect and storage units.

Our scheduling algorithm is a heuristic-guided, branch-and-bound search. The algorithm has three phases:

1. Identification of the the critical path.
2. Scheduling of operations for the critical path.
3. Scheduling of operations off the critical path.

For identifying the critical path through the given behavior, we calculate the *delay to output* (DTO) for each operator. To do this, we first must find the input-to-output delay of each operator, which involves mapping the individual operators to the RT components allocated from the library. An operator may map to more than one type of component, so we select the fastest, i.e., the component with the minimum input-to-output delay. Since we assume data-dependency analysis has been done, a dependency graph of all the operator is available. We calculate the DTOs by finding the operators at the bottom of the graph (the operators with output data that is not used by other operators) and work up the graph to the operators that provide input data (see Figure 7). For each operator, this formula is used:

$$\text{DTO} = \text{delay of operator} + \text{max. DTO of dependent operators}$$

A dependent operator is an operator that uses the data output of this operator. For example, in Figure 7a, the addition in **D := A + 1;** is a dependent operator of the addition in **A := B + C;**. Once the DTOs have been calculated, we find the critical path in the dependency graph by starting at the node with the highest DTO. We then proceed to the dependent operator with the highest DTO. Once we reach an operator with no dependent operators, we have found the bottom of the critical path. We then trace all data dependencies to this operator and put those operators on a stack according to data dependencies (see Figure 7b).

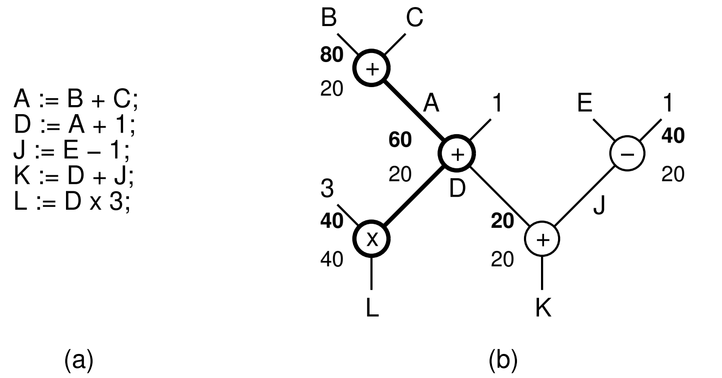

Figure 7: DTO calculation: (a) assignments and (b) data flow graph for assignments. Operator delays are given next to the operators (e.g., 20 ns for an addition) and DTOs for each operator are in bold. The critical path is highlighted.

The scheduling algorithm is described in Figure 8. During scheduling of the critical path, the algorithm does not try to evaluate all possible combinations of template matches for the entire path. Instead, it evaluates only a portion of the critical path at any time. In effect, the algorithm schedules a small "window" of the critical path, sliding the window down the critical path at each step of the scheduling process. All template matches in this window are examined. The operators that can be covered by a template are scheduled to a RT function and RT unit. If the selected function cannot be performed within a time step, it is scheduled to multiple time steps, i.e., a multi-cycle operation. If there is time left in a step after a function is performed, another function will be scheduled to the same step, i.e., functions will be chained. After a critical path operator is scheduled, the scheduling window will slide down so that the next unscheduled operator on the critical path is at the top of the window.

Figure 9 illustrates the scheduling of the critical path. In Figure 9a, the algorithm is attempting to schedule the addition in **A := B + C;** and finds the two template matches indicated. Schedules using each match will be attempted. If the template for the RT function **ADD(_,_,1)** is used, then the next critical path operator that will be scheduled will be the multiplication in **L := D** × **3;**. Figure 9b shows a possible schedule for the operators given a propagation delay limit of 30 nanoseconds. Note, that the RT functions **ADD** and **MULT** have been chained together in State 1 and that **MULT** has been multicycled across States 1 and 2.

After the critical path is scheduled, the algorithm will attempt to schedule the remaining operators in an as-soon-as-possible (ASAP) fashion, in descending order of DTOs. First, the algorithm will attempt

```
Schedule():
/* Schedule critical path first */
Pop an unscheduled node from the stack
if there's an unscheduled node on the critical path
   if can ASAP schedule node to existing schedule.
      Schedule() next node on stack
   /* Try other schedules of node */
   for depth = max. downto 0 do
      find covers of node of size depth
      for each cover of node do
         for each RT function cover can bind to do
            for each RT unit RT Function can bind to do
               if RT function needs to be multi-cycled,
                  add states and schedule
               if RT function can be chained, schedule.
               if schedule not shorter than best schedule found,
                  cancel this search.
               if could not schedule RT function,
                  if could not schedule RT function to an empty state,
                     cancel this search.
                  else maybe no RT units were available,
                     add a state to the schedule and retry
               else could schedule RT function
                  Schedule() next node on stack
else critical path is scheduled
   /* Schedule off critical path operators */
   if not shorter than best schedule found,
      cancel this search.
   ASAP_schedule()
   if failed
      while not successful do
         add state to front of schedule
         ASAP_schedule()
         if not shorter than best schedule found,
            cancel this search.
         if successful exit while loop.
         add state to end of schedule
         ASAP_schedule()
         if not shorter than best schedule found,
            cancel this search.
   if a schedule found
      save as best schedule found

ASAP_schedule():
do
   find unscheduled operator with highest DTO.
   ASAP schedule operator to available units
   if fails return failed.
while there is an unscheduled node.
```

Figure 8: Scheduling Algorithm

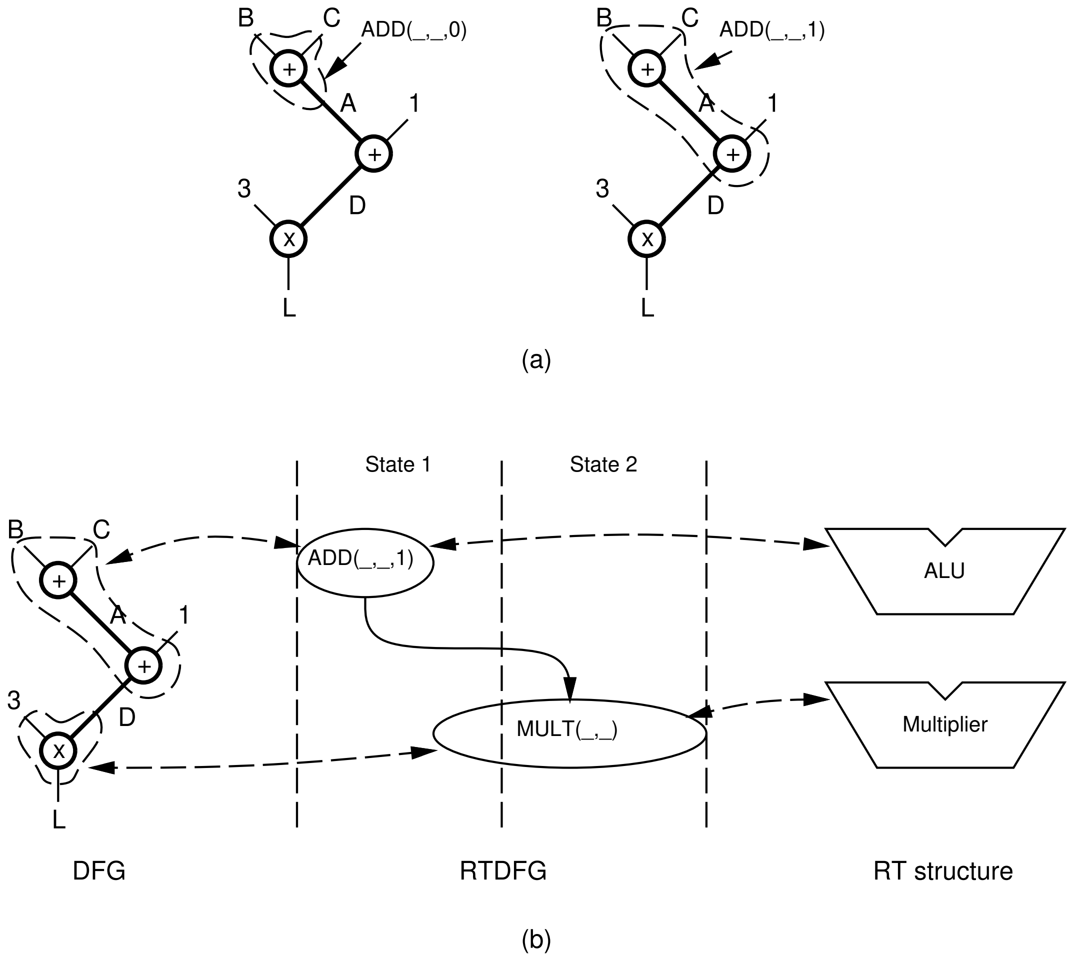

Figure 9: Example of critical path scheduling. (a) Templates that "cover" the addition in **A := B + C;**. (b) Possible covering and schedule.

to schedule the remaining operators ASAP, without additional time steps. If this fails, the algorithm will schedule the remaining operators ASAP with additional time steps added to the schedule. When attempting to schedule an operator ASAP, chaining is not attempted, though multicycling of RT function will be done if necessary.

As options for the scheduling program, RT function multicycling or chaining can be disabled. Also, multicycling and chaining can be done exclusive of each other. That is, no chaining will be permitted with a multicycled function and chained functions can not be multicycled. These options permit searching different styles of schedules. In addition, the search can be limited for time considerations. The program can be specified to stop searching after finding the n^{th} improved scheduled, or after a specified number of search attempts.

5 Experimental Results

Max. delay	Scheduling options	# states in schedule	Calls to sched_unit_to_state	
			to find best	exhaustive search
25 ns	chaining multicycling	19	97	100,000+[1]
50 ns	chaining multicycling	15	139	100,000+[1]
25 ns	no chaining multicycling	19	107	100,000+[1]
50 ns	no chaining multicycling	16	112	100,000+[1]
25 ns	exclusive chaining & multicycling	19	97	100,000+[1]
50 ns	exclusive chaining & multicycling	16	99	100,000+[1]
50 ns	chaining no multicycling	16	99	100,000+[1]
25 ns	no chaining no multicycling	NA[2]	NA[2]	117
50 ns	no chaining no multicycling	16	112	100,000+[1]

Table 1: Results for the Elliptic Filter with an allocation of 2 8-bit Ripple-Carry Adders and 2 8-bit Carry-Save Multipliers. [1] The program did not complete its search in within 100,000 attempts. [2] No schedule possible with the given scheduling options.

The scheduling algorithm has been implemented in C on a Sun SPARC workstation. The input description of the behavior is a temporary format based on BIF [DuHG90]. An input file is used to specify the allocation of units, the propagation delay limit, and options for scheduling (e.g., no function chaining). The allocated units are from a library of components derived using Synopsys® 3.0 design tools targeting LSI 1.0 micron CMOS technology. The components were either generated from modules available with the Synopsys tools or were synthesized from VHDL models [Synop]. The delay estimation features of the Synopsys tools were used to obtain the delay values for the RT functions of the library components. The output of the scheduler is a set of states. Each state has a set of RT functions. Each RT functions is associated with an RT unit instance, and a group of operators and operands from the input description.

We ran experiments on descriptions of the Elliptic Filter, the main computations for the Fast Fourier Transform, and a differential equation solver with overflow checking. We ran our experiments to explore the possible schedules for different allocations and delay limits, and to examine the effects of combining operation multicycling and chaining. We measured the complexity of our experiments by the number of calls to the

Allocation (8-bit components)	Scheduling options	Max. delay	# states in schedule	Calls to sched_unit_to_state	
				to find best	exhaustive search
1 Carry-Lookahead Adder/Subtractor 1 Carry-Save Multiplier	chaining multicycling	25 ns	13	29	117
		50 ns	8	27	75
2 Carry-Lookahead Adder/Subtractors 2 Carry-Save Multipliers	chaining multicycling	25 ns	7	239	285
		50 ns	4	14	103
	no chaining multicycling	50 ns	5	27	64
	chaining no multicycling	50 ns	4	14	29
	exclusive chaining & multicycling	50 ns	4	14	29
	no chaining no multicycling	50 ns	5	27	64
2 Carry-Lookahead Adder/Subtractors 1 Carry-Save Multiplier 1 Shifter	chaining multicycling	25 ns	12	29	144
		50 ns	8	27	302
	no chaining multicycling	50 ns	7	71	143
	chaining no multicycling	50 ns	6	65	130
	exclusive chaining & multicycling	50 ns	6	65	130
	no chaining no multicycling	50 ns	7	71	143

Table 2: Results for Fast Fourier Transform computations.

Allocation (8-bit components)	Scheduling options	Max. delay	# states in schedule	Calls to sched_unit_to_state	
				to find best	exhaustive search
2 ALUs 2 Carry-Save Multipliers 1 1-bit, Or gate	chaining multicycling	55ns	12	2670	100,000+[1]
	exclusive chaining & multicycling	55ns	9	26	100,000+[1]
2 ALUs 1 Carry-Save Multiplier 1 8-bit Shifter 1 1-bit, Or gate	chaining multicycling	55ns	10	28	100,000+[1]
	exclusive chaining & multicycling	55ns	9	27	100,000+[1]

Table 3: Results for Differential Equation Solver. [1] The program did not complete its search in within 100,000 attempts.

procedure *sched_unit_to_state*. This procedure is used whenever an RT function is assigned to an RT unit and state. The Elliptic Filter was found to have 34 RT functions. The Fast Fourier Transform computations had 12 RT functions. The differential equation solver had 24 RT functions. Even with pruning of the search space, the complexity of this search still increases exponentially with the number of RT functions found.

Though we do not claim a formal bound on the performance of this scheduling algorithm, our results substantiate our belief that this algorithm is useful for practical application. While an exhaustive search of all possible schedules this algorithm can generate cannot be completed in polynomial time, we find that the algorithm discovers the best schedule it can generate early in its search. By using the options available in our implementation, our experiments show that good results can be obtained quickly.

Tables 1, 2, 3 show the results for the experiments. For the Elliptic filter, allowing a combination of RT function chaining and multicycling achieved an improvement only in one instance. However, for the Fast Fourier Transform computations, for the allocation of 2 Adder/Subtractors, 1 Multiplier, and 1 Shifter, combining chaining and multicycling produced poorer results. This is because, by allowing combined chaining and multicycling, the RT functions for the critical path can excessively monopolize the available resources. Table 3 shows this was especially a problem for the differential equation solver. An example of this problem is shown in Figure 10. If two ADDs are chained (i.e., the output of one Adder is the input to another Adder) and the execution of the second ADD must be carried over to the next state, the second Adders will be unavailable for two states to perform a single addition, even though that Adder could perform an addition in a single state. This leads to the heuristic that operations on RT units should only be multicycled when there is no way possible to perform them within a single state, i.e., operation multi-cycling and chaining should be done exclusive of each other.

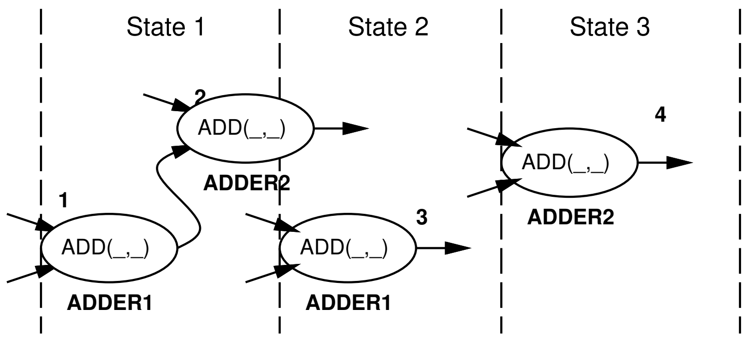

Figure 10: Monopolizing resources by combining chaining and multicycling: the chaining and multicycling of ADD() **1** and ADD() **2** forced ADD() **4** to be scheduled to a third state. But all four additions can be performed in only two states.

You will note that one of the allocations for the Fast Fourier Transform computations used an 8-bit shifter. This allocation demonstrates how the user-defined templates in the component library are exploited by the scheduling algorithm. In the Fast Fourier Transform computations, there is a multiplication by 2, **i = 2 * le;**, done for calculating the memory index of the next sample. In the component library, a template was defined that such a multiplication can be performed by the shifter. The scheduling algorithm examined implementing that multiplication on both the allocated multiplier and shifter, and selected the shifter. This demonstrates how the semantics of the component operations is determined by the the user-defined templates, and how some “hardware” optimizations can be incorporated into scheduling.

6 Conclusions

In this paper, we presented a scheduling technique that, for the first time, enables reuse of existing datapath components from user-defined libraries. The reusable datapath components can exhibit more parallelism than simple, generic models of RT components used in the past. We believe this is an important issue that will enable the acceptance of HLS techniques for datapath-oriented designs – an area where traditional HLS tools are often discounted as "unrealistic" and "too naive" by real designers.

Our scheduling approach uses a heuristic-guided, branch-and-bound algorithm on our novel design representation for RT-functionality. We presented experimental results on some HLS benchmarks and demonstrated the practical effectiveness of our approach. In our model, we observed that operation chaining and multicycling are most effective when done exclusive of the other.

Our approach is a first step in techniques for design reuse of realistic datapath components in HLS, and thus has several limitations. Currently, we have assumed reuse of non-pipelined, combinatorial datapath components, and have also restricted ourselves to scheduling of basic blocks; effects of wiring and physical design have not been incorporated. Future work will address the extension of our approach to overcome these restrictions.

References

[AnDu93] R. Ang and N. Dutt, "A Representation for the Binding of RT-Component Functionality to HDL Behavior," *Proceedings of the Conference on Hardware Description Languages*, pp. 251–266, April 1993.

[Camp91] R. Camposano, "Path-Based Scheduling for Synthesis," *IEEE Transactions on Computer-Aided Design for Integrated Circuits and Systems*, vol. 10, no. 1, pp. 136–141, January 1991.

[CKPR93] M. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, and J. Rabaey, "Instruction Set Mapping for Performance Optimization," *Proceedings of ICCAD*, pp. 518–521, 1993.

[DuHG90] Dutt, N., Hadley, T. and Gajski, D., "An Intermediate Representation for Behavioral Synthesis," *Proceedings of the* 27^{th} *Design Automation Conference*, pp. 14–19, 1990.

[GlGr78] R. Glanville, S. Graham "A New Method for Compiler Code Generation," *Conference Record of the 5th Annual ACM Symposium on Principles of Programming Languages*, pp. 231–240, 1978.

[GCDM93] W. Guerts, F. Catthoor, and H. De Man, "Heuristic Techniques for the Synthesis of Complex Functional Units," *Proceedings of the European Conference on Design Automation (EDAC)*, pp. 552–556, 1993.

[Kahrs86] M. Kahrs, "Matching a parts library in a silicon compiler," *Proceedings of ICCAD*, pp. 169–172, 1986.

[KnWi92] D. Knapp and M. Winslett, "A Prescriptive Formal Model for Data-Path Hardware," *IEEE Transactions on Computer-Aided Design for Integrated Circuits and Systems*, vol. 11, no. 2, pp. 158–184, February 1992.

[LMPa94] C. Liem, T. May, and P. Paulin, "Instruction-Set Matching and Selection for DSP and ASIP Code Generation" *Proceedings of the European Conference on Design Automation (EDAC)*, pp. 31–37, 1994.

[Mar93] P. Marwedel, "Tree-Based Mapping for Algorithms to Predefined Structures," *Proceedings of ICCAD*, pp. 586–593, 1993.

[PaGa87] B. Pangrle and D. Gajski, "Slicer: A State Synthesizer for Intelligent Silicon Compilation," *Proceedings of the International Conference on Computer Design*, pp. 42–45, 1987.

[PaPM86] A. Parker, J. Pizarro, M. Mlinar, "MAHA: A Program for Datapath Synthesis," *Proceedings of the* 23^{rd} *Design Automation Conference*, pp. 461–466, 1986.

[PLNG90] R. Potasman, J. Lis, A. Nicolau, D. Gajski, "Percolation Based Synthesis," *Proceedings of the* 27^{th} *Design Automation Conference*, pp. 444–449, 1990.

[PaKn89] P. Paulin and J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Transactions on Computer-Aided Design for Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661–679, June 1990.

[RaKu92] D. S. Rao and F. J. Kurdahi, "System Modeling for High-Level Synthesis Using Regularity Extraction," *Proceeding of 6th Workshop on High Level Synthesis*, pp. 267–272, 1992.

[RuGB90] E. Rundensteiner, D. Gajski, and L. Bic, "The Component Synthesis Algorithm: Technology Mapping for Register Transfer Descriptions," *Proceedings of ICCAD*, pp. 208–211, 1990.

[Synop] *Synopsys DesignWare Databook, Version 3.0*, Synopsys, Inc., Dec. 1992.

[WPAV92] A. van der Werf, M. Peek, E. Aarts, J. Van Meerbergen, P. Lippens, W. Verhaegh, "Area Optimization of Multi-Functional Processing Units," *Proceedings of the International Conference on Computer-Aided Design*, pp. 292–299, 1992.

A Limitations

Because of the manner that the critical path is calculated compared to the way the critical path is scheduled, the critical path found may prove to be not the "slowest" path through the behavior. For calculating the DTOs used to find the critical path, only individual operators are considered. But during scheduling of the critical path, the algorithm will try to schedule multiple operators to a single RT Function. Consequently, in the scheduled behavior, the critical path may actually be performed faster than the DTOs indicated, and some other non-critical path may be slower.

When scheduling operations that are not on the critical path, chaining of operations is no considered.

B Examples

The following is the text of the examples used for the experiments. The example below is for the main computations for the Fast Fourier Transform.

```
SYMBOL_TABLE {

   TYPE
      bit8 = {7..0};

   VAR
      le, i, Wptr_Real, Wptr_Imag, xuReal, TempReal, TempImag,
      TmReal, TmImag, xlReal, xuImag, xmImag, lower : bit8;

}

TABLE fft_body {

   OPS_BASED SYNCHRONOUS

   FIRST STATE: State_1
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: lower = i + le;
            NEXT_STATE: State_2;
      }
   },
   STATE: State_2
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: TempReal = xuReal + xlReal;
```

```
         NEXT_STATE: State_3;
   }
},
STATE: State_3
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: TempImag = xuImag + xmImag;
         NEXT_STATE: State_4;
   }
},
STATE: State_4
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: TmReal = xuReal - xlReal;
         NEXT_STATE: State_5;
   }
},
STATE: State_5
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: TmImag = xuImag - xmImag;
         NEXT_STATE: State_6;
   }
},
STATE: State_6
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: TempReal = TmReal * Wptr_Real - TmImag * Wptr_Imag;
         NEXT_STATE: State_7;
   }
},
STATE: State_7
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: TempImag = TmReal * Wptr_Imag - TmImag * Wptr_Real;
         NEXT_STATE: State_8;
   }
},
STATE: State_8
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: i = 2 * le;
         NEXT_STATE: dead;
   }
},
STATE: dead
{
```

```
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: ;
            NEXT_STATE: dead;
      }
   }
}
```

The example below is for the loop body of the Elliptic Filter.

```
SYMBOL_TABLE {

   TYPE
      bit8 = {7..0};

   PORT
      IN_PORT: INPUT OF bit8;
      OUT_PORT: OUTPUT OF bit8;

   VAR
      I,N2,N40,N33,N39,N43,N13,N41,N26,N42,N44,M1,N47,M2,N46,
      N48,N51,N50,N49,M3,N53,N52,M4,N54,N55,N59,N56,N63,N18,N60,
      M5,N57,N64,N67,N38,N65,N58,M7,N61,M6,N66,M8,OTT: bit8;

}

TABLE ellip_filter {

   OPS_BASED SYNCHRONOUS

   FIRST STATE: State_1
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: I=IN_PORT;
            NEXT_STATE: State_1b;
      }
   },
   STATE: State_1b
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: N40=N2+I;
            NEXT_STATE: State_1c;
      }
   },
   STATE: State_1c
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: N43=N33+N39;
            NEXT_STATE: State_2;
      }
   },
   STATE: State_2
   {
```

```
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N41=N40+N13;
         NEXT_STATE: State_3;
   }
},
STATE: State_3
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N42=N41+N26;
         NEXT_STATE: State_4;
   }
},
STATE: State_4
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N44=N42+N43;
         NEXT_STATE: State_5;
   }
},
STATE: State_5
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N47=N44*M1;
         NEXT_STATE: State_7;
   }
},
STATE: State_7
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N46=N44*M2;
         NEXT_STATE: State_7a;
   }
},
STATE: State_7a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N48=N47+N41;
         NEXT_STATE: State_8;
   }
},
STATE: State_8
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N51=N48+N41;
         NEXT_STATE: State_8a;
```

```
   }
},
STATE: State_8a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N50=N48+N44;
         NEXT_STATE: State_9;
   }
},
STATE: State_9
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N49=N46+N43;
         NEXT_STATE: State_9a;
   }
},
STATE: State_9a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N53=N51*M3;
         NEXT_STATE: State_10;
   }
},
STATE: State_10
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N52=N49+N43;
         NEXT_STATE: State_11;
   }
},
STATE: State_11
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N54=N52*M4;
          NEXT_STATE: State_11a;
   }
},
STATE: State_11a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N26=N49+N50;
         NEXT_STATE: State_11b;
   }
},
STATE: State_11b
{
   CONDITION: (true);
```

```
   {
      CONDVALUE: (true);
         ACTIONS: N55=N53+N40;
         NEXT_STATE: State_12;
   }
},
STATE: State_12
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N59=N48+N55;
         NEXT_STATE: State_12a;
   }
},
STATE: State_12a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N56=N55+N40;
         NEXT_STATE: State_13;
   }
},
STATE: State_13
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N63=N54+N39;
         NEXT_STATE: State_13a;
   }
},
STATE: State_13a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N60=N59+N18;
         NEXT_STATE: State_13b;
   }
},
STATE: State_13b
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N57=N56*M5;
         NEXT_STATE: State_14;
   }
},
STATE: State_14
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N64=N63+N49;
         NEXT_STATE: State_14a;
   }
```

```
},
STATE: State_14a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N67=N63+N39;
         NEXT_STATE: State_15;
   }
},
STATE: State_15
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N65=N64+N38;
         NEXT_STATE: State_15a;
   }
},
STATE: State_15a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N58=N57+I;
         NEXT_STATE: State_15c;
   }
},
STATE: State_15c
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N61=N60*M7;
         NEXT_STATE: State_16;
   }
},
STATE: State_16
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS:
                  N2=N58+N55;
         NEXT_STATE: State_17;
   }
},
STATE: State_17
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N66=N65*M6;
         NEXT_STATE: State_17a;
   }
},
STATE: State_17a
{
   CONDITION: (true);
```

```
   {
      CONDVALUE: (true);
         ACTIONS: N18=N61+N18;
         NEXT_STATE: State_18;
   }
},
STATE: State_18
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N13=N18+N60;
         NEXT_STATE: State_19;
   }
},
STATE: State_19
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N38=N66+N38;
         NEXT_STATE: State_19a;
   }
},
STATE: State_19a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: OTT=N67*M8;
         NEXT_STATE: State_20;
   }
},
STATE: State_20
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N33=N38+N65;
         NEXT_STATE: State_21;
   }
},
STATE: State_21
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: N39=OTT+N63;
         NEXT_STATE: State_21a;
         EVENT: ;
   }
},
STATE: State_21a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: OUT_PORT=OTT;
         NEXT_STATE: dead;
```

```
      }
   },
   STATE: dead
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: ;
            NEXT_STATE: dead;
      }
   }
}
```

The example below is for a differential equation solver with overflow checking.

```
SYMBOL_TABLE {

   TYPE
      bit1 = {1};
      bit8 = {7..0};

   VAR
      x_var,y_var,u_var, dx_var,
      y1, t1,t2,t3,t4,t5,t6: bit8;
      t2a,t3a: bit8;
      ovf_err: bit1;
}

TABLE diffeq {

   OPS_BASED SYNCHRONOUS

   FIRST STATE: State_1
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: t1 = u_var * dx_var;
            NEXT_STATE: State_2;
      }
   },
   STATE: State_2
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: t2a = 2 * x_var;
            NEXT_STATE: State_2a;
      }
   },
   STATE: State_2a
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: t2 = t2a + x_var;
            NEXT_STATE: State_3;
      }
```

```
},
STATE: State_3
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: ovf_err = ovf_err ||
                     (((t2a + x_var) > 127) || ((t2a + x_var) < -128));
         NEXT_STATE: State_4;
   }
},
STATE: State_4
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: t3a = 2 * y_var;
         NEXT_STATE: State_4a;
   }
},
STATE: State_4a
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: t3 = t3a + y_var;
         NEXT_STATE: State_5;
   }
},
STATE: State_5
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: ovf_err = ovf_err ||
                     (((t3a + y_var) > 127) || ((t3a + y_var) < -128));
         NEXT_STATE: State_6;
   }
},
STATE: State_6
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: t4 = t1 * t2;
         NEXT_STATE: State_7;
   }
},
STATE: State_7
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: t5 = dx_var * t3;
         NEXT_STATE: State_8;
   }
},
STATE: State_8
{
```

```
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: t6 = u_var - t4;
         NEXT_STATE: State_9;
   }
},
STATE: State_9
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: ovf_err = ovf_err ||
                    (((u_var - t4) > 127) || ((u_var - t4) < -128));
         NEXT_STATE: State_10;
   }
},
STATE: State_10
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: u_var = t6 - t5;
         NEXT_STATE: State_11;
   }
},
STATE: State_11
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: ovf_err = ovf_err ||
                    (((t6 - t5) > 127) || ((t6 - t5) < -128));
         NEXT_STATE: State_12;
   }
},
STATE: State_12
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: y1 = u_var * dx_var;
         NEXT_STATE: State_13;
   }
},
STATE: State_13
{
   CONDITION: (true);
   {
      CONDVALUE: (true);
         ACTIONS: ovf_err = ovf_err ||
                    (((y_var + y1) > 127) || ((y_var + y1) < -128));
         NEXT_STATE: State_14;
   }
},
STATE: State_14
{
   CONDITION: (true);
   {
```

```
      CONDVALUE: (true);
         ACTIONS: y_var = y_var + y1;
         NEXT_STATE: State_15;
      }
   },
   STATE: State_15
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: ovf_err = ovf_err ||
                        (((x_var+dx_var) > 127) || ((x_var + dx_var) < -128));
            NEXT_STATE: State_16;
      }
   },
   STATE: State_16
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: x_var = x_var + dx_var;
            NEXT_STATE: dead;
      }
   },
   STATE: dead
   {
      CONDITION: (true);
      {
         CONDVALUE: (true);
            ACTIONS: ;
            NEXT_STATE: dead;
      }
   }
}
```

C Results

Below is the allocation input and output results for our experiments. For the scheduling program, an input file was used specifying the allocation of units, the clock (maximum propagation delay per state), and options for scheduling. The output is a set of states where each state has a set of RT functions. Each RT function is associated with an RT unit instance, and a group of operators and operands from the input description. Below, each allocation is followed by the best schedule found for that allocation.

C.1 Results for Elliptic Filter example

```
----------------
---Allocation---
Adder_RPL8 2
Mult_CSA8 2
clock 25

---Schedule---
State: 1
  Xform: 1, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 39
op: N2, line: 39
op: +, line: 39
  Xform: 4, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 48
op: N33, line: 48
op: +, line: 48

State: 2
  Xform: 2, func: RTAdd8, Unit: Adder_RPL8_2
op: N13, line: 57
op: N40, line: 57
op: +, line: 57

State: 3
  Xform: 3, func: RTAdd8, Unit: Adder_RPL8_2
op: N26, line: 66
op: N41, line: 66
op: +, line: 66

State: 4
  Xform: 5, func: RTAdd8, Unit: Adder_RPL8_2
op: N43, line: 75
op: N42, line: 75
op: +, line: 75
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84

State: 5
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 6
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 7
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 17, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 129
op: N46, line: 129
op: +, line: 129

State: 8
  Xform: 8, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 111
op: N48, line: 111
op: +, line: 111
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 18, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 147
op: N49, line: 147
op: +, line: 147

State: 9
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 28, func: RTAdd8, Unit: Adder_RPL8_2
op: N44, line: 120
op: N48, line: 120
op: +, line: 120

State: 10
```

```
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 31, func: RTAdd8, Unit: Adder_RPL8_1
op: N50, line: 165
op: N49, line: 165
op: +, line: 165

State: 11
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 20, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 201
op: N54, line: 201
op: +, line: 201

State: 12
  Xform: 11, func: RTAdd8, Unit: Adder_RPL8_2
op: N55, line: 183
op: N48, line: 183
op: +, line: 183
  Xform: 21, func: RTAdd8, Unit: Adder_RPL8_1
op: N49, line: 228
op: N63, line: 228
op: +, line: 228

State: 13
  Xform: 12, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 210
op: N59, line: 210
op: +, line: 210
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 22, func: RTAdd8, Unit: Adder_RPL8_1
op: N40, line: 192
op: N55, line: 192
op: +, line: 192

State: 14
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 23, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 246
op: N64, line: 246
op: +, line: 246
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219
  Xform: 25, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 237
op: N63, line: 237
op: +, line: 237

State: 15
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219

State: 16
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 29, func: RTAdd8, Unit: Adder_RPL8_1
op: I, line: 255
op: N57, line: 255
op: +, line: 255

State: 17
  Xform: 15, func: RTAdd8, Unit: Adder_RPL8_2
op: N60, line: 301
op: N18, line: 301
op: +, line: 301
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 32, func: RTAdd8, Unit: Adder_RPL8_1
op: N55, line: 274
op: N58, line: 274
op: +, line: 274
```

```
State: 18
  Xform: 30, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 310
op: N66, line: 310
op: +, line: 310
  Xform: 34, func: RTAdd8, Unit: Adder_RPL8_1
op: N63, line: 337
op: OTT, line: 337
op: +, line: 337

State: 19
  Xform: 33, func: RTAdd8, Unit: Adder_RPL8_2
op: N65, line: 328
op: N38, line: 328
op: +, line: 328

----------------
---Allocation---
Adder_RPL8 2
Mult_CSA8 2
clock 50

---Schedule---
State: 1
  Xform: 1, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 39
op: N2, line: 39
op: +, line: 39
  Xform: 4, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 48
op: N33, line: 48
op: +, line: 48

State: 2
  Xform: 2, func: RTAdd8, Unit: Adder_RPL8_2
op: N13, line: 57
op: N40, line: 57
op: +, line: 57

State: 3
  Xform: 3, func: RTAdd8, Unit: Adder_RPL8_2
op: N26, line: 66
op: N41, line: 66
op: +, line: 66

State: 4
  Xform: 5, func: RTAdd8, Unit: Adder_RPL8_2
op: N43, line: 75
op: N42, line: 75
op: +, line: 75
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84

State: 5
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 6
  Xform: 8, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 111
op: N48, line: 111
op: +, line: 111
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 17, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 129
op: N46, line: 129
op: +, line: 129

State: 7
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 18, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 147
op: N49, line: 147
op: +, line: 147

State: 8
  Xform: 11, func: RTAdd8, Unit: Adder_RPL8_2
op: N55, line: 183
op: N48, line: 183
op: +, line: 183
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_2
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 22, func: RTAdd8, Unit: Adder_RPL8_1
op: N40, line: 192
op: N55, line: 192
op: +, line: 192

State: 9
  Xform: 12, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 210
op: N59, line: 210
op: +, line: 210
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
```

```
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 20, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 201
op: N54, line: 201
op: +, line: 201
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219

State: 10
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 21, func: RTAdd8, Unit: Adder_RPL8_1
op: N49, line: 228
op: N63, line: 228
op: +, line: 228

State: 11
  Xform: 15, func: RTAdd8, Unit: Adder_RPL8_2
op: N60, line: 301
op: N18, line: 301
op: +, line: 301
  Xform: 23, func: RTAdd8, Unit: Adder_RPL8_1
op: N38, line: 246
op: N64, line: 246
op: +, line: 246

State: 12
  Xform: 25, func: RTAdd8, Unit: Adder_RPL8_2
op: N39, line: 237
op: N63, line: 237
op: +, line: 237
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 28, func: RTAdd8, Unit: Adder_RPL8_1
op: N44, line: 120
op: N48, line: 120
op: +, line: 120

State: 13
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_2
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 29, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 255
op: N57, line: 255
op: +, line: 255
  Xform: 30, func: RTAdd8, Unit: Adder_RPL8_1
op: N38, line: 310
op: N66, line: 310
op: +, line: 310

State: 14
  Xform: 31, func: RTAdd8, Unit: Adder_RPL8_2
op: N50, line: 165
op: N49, line: 165
op: +, line: 165
  Xform: 32, func: RTAdd8, Unit: Adder_RPL8_1
op: N55, line: 274
op: N58, line: 274
op: +, line: 274

State: 15
  Xform: 33, func: RTAdd8, Unit: Adder_RPL8_2
op: N65, line: 328
op: N38, line: 328
op: +, line: 328
  Xform: 34, func: RTAdd8, Unit: Adder_RPL8_1
op: N63, line: 337
op: OTT, line: 337
op: +, line: 337

----------------
---Allocation---
Adder_RPL8 2
Mult_CSA8 2
clock 25
nochain

---Schedule---
State: 1
  Xform: 1, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 39
op: N2, line: 39
op: +, line: 39
  Xform: 4, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 48
op: N33, line: 48
op: +, line: 48

State: 2
  Xform: 2, func: RTAdd8, Unit: Adder_RPL8_2
op: N13, line: 57
op: N40, line: 57
op: +, line: 57

State: 3
  Xform: 3, func: RTAdd8, Unit: Adder_RPL8_2
op: N26, line: 66
op: N41, line: 66
op: +, line: 66

State: 4
  Xform: 5, func: RTAdd8, Unit: Adder_RPL8_2
op: N43, line: 75
op: N42, line: 75
```

```
op: +, line: 75

State: 5
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 6
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 7
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 17, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 129
op: N46, line: 129
op: +, line: 129

State: 8
  Xform: 8, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 111
op: N48, line: 111
op: +, line: 111
  Xform: 18, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 147
op: N49, line: 147
op: +, line: 147

State: 9
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 28, func: RTAdd8, Unit: Adder_RPL8_2
op: N44, line: 120
op: N48, line: 120
op: +, line: 120

State: 10
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 31, func: RTAdd8, Unit: Adder_RPL8_2
op: N50, line: 165
op: N49, line: 165
op: +, line: 165

State: 11
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 20, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 201
op: N54, line: 201
op: +, line: 201

State: 12
  Xform: 11, func: RTAdd8, Unit: Adder_RPL8_2
op: N55, line: 183
op: N48, line: 183
op: +, line: 183
  Xform: 21, func: RTAdd8, Unit: Adder_RPL8_1
op: N49, line: 228
op: N63, line: 228
op: +, line: 228

State: 13
  Xform: 12, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 210
op: N59, line: 210
op: +, line: 210
  Xform: 22, func: RTAdd8, Unit: Adder_RPL8_1
op: N40, line: 192
op: N55, line: 192
op: +, line: 192

State: 14
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 23, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 246
op: N64, line: 246
op: +, line: 246
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219
  Xform: 25, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 237
op: N63, line: 237
op: +, line: 237
```

```
State: 15
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219

State: 16
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 29, func: RTAdd8, Unit: Adder_RPL8_1
op: I, line: 255
op: N57, line: 255
op: +, line: 255

State: 17
  Xform: 15, func: RTAdd8, Unit: Adder_RPL8_2
op: N60, line: 301
op: N18, line: 301
op: +, line: 301
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 32, func: RTAdd8, Unit: Adder_RPL8_1
op: N55, line: 274
op: N58, line: 274
op: +, line: 274

State: 18
  Xform: 30, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 310
op: N66, line: 310
op: +, line: 310
  Xform: 34, func: RTAdd8, Unit: Adder_RPL8_1
op: N63, line: 337
op: OTT, line: 337
op: +, line: 337

State: 19
  Xform: 33, func: RTAdd8, Unit: Adder_RPL8_2
op: N65, line: 328
op: N38, line: 328
op: +, line: 328

----------------
---Allocation---
Adder_RPL8 2
Mult_CSA8 2
clock 50
nochain

---Schedule---
State: 1
  Xform: 1, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 39
op: N2, line: 39
op: +, line: 39
  Xform: 4, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 48
op: N33, line: 48
op: +, line: 48

State: 2
  Xform: 2, func: RTAdd8, Unit: Adder_RPL8_2
op: N13, line: 57
op: N40, line: 57
op: +, line: 57

State: 3
  Xform: 3, func: RTAdd8, Unit: Adder_RPL8_2
op: N26, line: 66
op: N41, line: 66
op: +, line: 66

State: 4
  Xform: 5, func: RTAdd8, Unit: Adder_RPL8_2
op: N43, line: 75
op: N42, line: 75
op: +, line: 75

State: 5
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 6
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 17, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 129
op: N46, line: 129
op: +, line: 129
```

```
State: 7
  Xform: 8, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 111
op: N48, line: 111
op: +, line: 111
  Xform: 18, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 147
op: N49, line: 147
op: +, line: 147

State: 8
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 28, func: RTAdd8, Unit: Adder_RPL8_2
op: N44, line: 120
op: N48, line: 120
op: +, line: 120

State: 9
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 20, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 201
op: N54, line: 201
op: +, line: 201

State: 10
  Xform: 11, func: RTAdd8, Unit: Adder_RPL8_2
op: N55, line: 183
op: N48, line: 183
op: +, line: 183
  Xform: 21, func: RTAdd8, Unit: Adder_RPL8_1
op: N49, line: 228
op: N63, line: 228
op: +, line: 228

State: 11
  Xform: 12, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 210
op: N59, line: 210
op: +, line: 210
  Xform: 22, func: RTAdd8, Unit: Adder_RPL8_1
op: N40, line: 192
op: N55, line: 192
op: +, line: 192

State: 12
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 23, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 246
op: N64, line: 246
op: +, line: 246
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219
  Xform: 25, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 237
op: N63, line: 237
op: +, line: 237

State: 13
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 29, func: RTAdd8, Unit: Adder_RPL8_1
op: I, line: 255
op: N57, line: 255
op: +, line: 255

State: 14
  Xform: 15, func: RTAdd8, Unit: Adder_RPL8_2
op: N60, line: 301
op: N18, line: 301
op: +, line: 301
  Xform: 30, func: RTAdd8, Unit: Adder_RPL8_1
op: N38, line: 310
op: N66, line: 310
op: +, line: 310

State: 15
  Xform: 31, func: RTAdd8, Unit: Adder_RPL8_2
op: N50, line: 165
op: N49, line: 165
op: +, line: 165
  Xform: 32, func: RTAdd8, Unit: Adder_RPL8_1
op: N55, line: 274
op: N58, line: 274
op: +, line: 274

State: 16
  Xform: 33, func: RTAdd8, Unit: Adder_RPL8_2
op: N65, line: 328
op: N38, line: 328
op: +, line: 328
  Xform: 34, func: RTAdd8, Unit: Adder_RPL8_1
op: N63, line: 337
op: OTT, line: 337
```

```
op: +, line: 337

----------------
---Allocation---
Adder_RPL8 2
Mult_CSA8 2
clock 25
exclusive

---Schedule---
State: 1
  Xform: 1, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 39
op: N2, line: 39
op: +, line: 39
  Xform: 4, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 48
op: N33, line: 48
op: +, line: 48

State: 2
  Xform: 2, func: RTAdd8, Unit: Adder_RPL8_2
op: N13, line: 57
op: N40, line: 57
op: +, line: 57

State: 3
  Xform: 3, func: RTAdd8, Unit: Adder_RPL8_2
op: N26, line: 66
op: N41, line: 66
op: +, line: 66

State: 4
  Xform: 5, func: RTAdd8, Unit: Adder_RPL8_2
op: N43, line: 75
op: N42, line: 75
op: +, line: 75

State: 5
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 6
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 7
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 17, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 129
op: N46, line: 129
op: +, line: 129

State: 8
  Xform: 8, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 111
op: N48, line: 111
op: +, line: 111
  Xform: 18, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 147
op: N49, line: 147
op: +, line: 147

State: 9
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 28, func: RTAdd8, Unit: Adder_RPL8_2
op: N44, line: 120
op: N48, line: 120
op: +, line: 120

State: 10
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 31, func: RTAdd8, Unit: Adder_RPL8_2
op: N50, line: 165
op: N49, line: 165
op: +, line: 165

State: 11
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 20, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 201
op: N54, line: 201
op: +, line: 201

State: 12
  Xform: 11, func: RTAdd8, Unit: Adder_RPL8_2
```

```
op: N55, line: 183
op: N48, line: 183
op: +, line: 183
  Xform: 21, func: RTAdd8, Unit: Adder_RPL8_1
op: N49, line: 228
op: N63, line: 228
op: +, line: 228

State: 13
  Xform: 12, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 210
op: N59, line: 210
op: +, line: 210
  Xform: 22, func: RTAdd8, Unit: Adder_RPL8_1
op: N40, line: 192
op: N55, line: 192
op: +, line: 192

State: 14
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 23, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 246
op: N64, line: 246
op: +, line: 246
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219
  Xform: 25, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 237
op: N63, line: 237
op: +, line: 237

State: 15
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219

State: 16
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 29, func: RTAdd8, Unit: Adder_RPL8_1
op: I, line: 255
op: N57, line: 255
op: +, line: 255

State: 17
  Xform: 15, func: RTAdd8, Unit: Adder_RPL8_2
op: N60, line: 301
op: N18, line: 301
op: +, line: 301
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 32, func: RTAdd8, Unit: Adder_RPL8_1
op: N55, line: 274
op: N58, line: 274
op: +, line: 274

State: 18
  Xform: 30, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 310
op: N66, line: 310
op: +, line: 310
  Xform: 34, func: RTAdd8, Unit: Adder_RPL8_1
op: N63, line: 337
op: OTT, line: 337
op: +, line: 337

State: 19
  Xform: 33, func: RTAdd8, Unit: Adder_RPL8_2
op: N65, line: 328
op: N38, line: 328
op: +, line: 328

----------------
---Allocation---
Adder_RPL8 2
Mult_CSA8 2
clock 50
exclusive

---Schedule---
State: 1
  Xform: 1, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 39
op: N2, line: 39
op: +, line: 39
  Xform: 4, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 48
op: N33, line: 48
op: +, line: 48

State: 2
  Xform: 2, func: RTAdd8, Unit: Adder_RPL8_2
```

```
op: N13, line: 57
op: N40, line: 57
op: +, line: 57

State: 3
  Xform: 3, func: RTAdd8, Unit: Adder_RPL8_2
op: N26, line: 66
op: N41, line: 66
op: +, line: 66

State: 4
  Xform: 5, func: RTAdd8, Unit: Adder_RPL8_2
op: N43, line: 75
op: N42, line: 75
op: +, line: 75

State: 5
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 6
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 17, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 129
op: N46, line: 129
op: +, line: 129

State: 7
  Xform: 8, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 111
op: N48, line: 111
op: +, line: 111
  Xform: 18, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 147
op: N49, line: 147
op: +, line: 147

State: 8
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 28, func: RTAdd8, Unit: Adder_RPL8_2
op: N44, line: 120
op: N48, line: 120
op: +, line: 120

State: 9
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 20, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 201
op: N54, line: 201
op: +, line: 201

State: 10
  Xform: 11, func: RTAdd8, Unit: Adder_RPL8_2
op: N55, line: 183
op: N48, line: 183
op: +, line: 183
  Xform: 21, func: RTAdd8, Unit: Adder_RPL8_1
op: N49, line: 228
op: N63, line: 228
op: +, line: 228

State: 11
  Xform: 12, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 210
op: N59, line: 210
op: +, line: 210
  Xform: 22, func: RTAdd8, Unit: Adder_RPL8_1
op: N40, line: 192
op: N55, line: 192
op: +, line: 192

State: 12
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 23, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 246
op: N64, line: 246
op: +, line: 246
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219
  Xform: 25, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 237
op: N63, line: 237
op: +, line: 237

State: 13
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
```

```
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 29, func: RTAdd8, Unit: Adder_RPL8_1
op: I, line: 255
op: N57, line: 255
op: +, line: 255

State: 14
  Xform: 15, func: RTAdd8, Unit: Adder_RPL8_2
op: N60, line: 301
op: N18, line: 301
op: +, line: 301
  Xform: 30, func: RTAdd8, Unit: Adder_RPL8_1
op: N38, line: 310
op: N66, line: 310
op: +, line: 310

State: 15
  Xform: 31, func: RTAdd8, Unit: Adder_RPL8_2
op: N50, line: 165
op: N49, line: 165
op: +, line: 165
  Xform: 32, func: RTAdd8, Unit: Adder_RPL8_1
op: N55, line: 274
op: N58, line: 274
op: +, line: 274

State: 16
  Xform: 33, func: RTAdd8, Unit: Adder_RPL8_2
op: N65, line: 328
op: N38, line: 328
op: +, line: 328
  Xform: 34, func: RTAdd8, Unit: Adder_RPL8_1
op: N63, line: 337
op: OTT, line: 337
op: +, line: 337

----------------
---Allocation---
Adder_RPL8 2
Mult_CSA8 2
clock 50
nomulticycle

---Schedule---
State: 1
  Xform: 1, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 39
op: N2, line: 39
op: +, line: 39
  Xform: 4, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 48
op: N33, line: 48
op: +, line: 48

State: 2
  Xform: 2, func: RTAdd8, Unit: Adder_RPL8_2
op: N13, line: 57
op: N40, line: 57
op: +, line: 57

State: 3
  Xform: 3, func: RTAdd8, Unit: Adder_RPL8_2
op: N26, line: 66
op: N41, line: 66
op: +, line: 66

State: 4
  Xform: 5, func: RTAdd8, Unit: Adder_RPL8_2
op: N43, line: 75
op: N42, line: 75
op: +, line: 75

State: 5
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 6
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 17, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 129
op: N46, line: 129
op: +, line: 129

State: 7
  Xform: 8, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 111
op: N48, line: 111
op: +, line: 111
  Xform: 18, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 147
op: N49, line: 147
op: +, line: 147

State: 8
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 28, func: RTAdd8, Unit: Adder_RPL8_2
op: N44, line: 120
op: N48, line: 120
op: +, line: 120
```

```
State: 9
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 20, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 201
op: N54, line: 201
op: +, line: 201

State: 10
  Xform: 11, func: RTAdd8, Unit: Adder_RPL8_2
op: N55, line: 183
op: N48, line: 183
op: +, line: 183
  Xform: 21, func: RTAdd8, Unit: Adder_RPL8_1
op: N49, line: 228
op: N63, line: 228
op: +, line: 228

State: 11
  Xform: 12, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 210
op: N59, line: 210
op: +, line: 210
  Xform: 22, func: RTAdd8, Unit: Adder_RPL8_1
op: N40, line: 192
op: N55, line: 192
op: +, line: 192

State: 12
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 23, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 246
op: N64, line: 246
op: +, line: 246
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219
  Xform: 25, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 237
op: N63, line: 237
op: +, line: 237

State: 13
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
op: N67, line: 319
op: *, line: 319
  Xform: 29, func: RTAdd8, Unit: Adder_RPL8_1
op: I, line: 255
op: N57, line: 255
op: +, line: 255

State: 14
  Xform: 15, func: RTAdd8, Unit: Adder_RPL8_2
op: N60, line: 301
op: N18, line: 301
op: +, line: 301
  Xform: 30, func: RTAdd8, Unit: Adder_RPL8_1
op: N38, line: 310
op: N66, line: 310
op: +, line: 310

State: 15
  Xform: 31, func: RTAdd8, Unit: Adder_RPL8_2
op: N50, line: 165
op: N49, line: 165
op: +, line: 165
  Xform: 32, func: RTAdd8, Unit: Adder_RPL8_1
op: N55, line: 274
op: N58, line: 274
op: +, line: 274

State: 16
  Xform: 33, func: RTAdd8, Unit: Adder_RPL8_2
op: N65, line: 328
op: N38, line: 328
op: +, line: 328
  Xform: 34, func: RTAdd8, Unit: Adder_RPL8_1
op: N63, line: 337
op: OTT, line: 337
op: +, line: 337

----------------
---Allocation---
Adder_RPL8 2
Mult_CSA8 2
clock 50
nomulticycle
nochain

---Schedule---
State: 1
  Xform: 1, func: RTAdd8, Unit: Adder_RPL8_2
op: I, line: 39
op: N2, line: 39
op: +, line: 39
  Xform: 4, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 48
op: N33, line: 48
op: +, line: 48

State: 2
  Xform: 2, func: RTAdd8, Unit: Adder_RPL8_2
op: N13, line: 57
```

```
op: N40, line: 57
op: +, line: 57

State: 3
  Xform: 3, func: RTAdd8, Unit: Adder_RPL8_2
op: N26, line: 66
op: N41, line: 66
op: +, line: 66

State: 4
  Xform: 5, func: RTAdd8, Unit: Adder_RPL8_2
op: N43, line: 75
op: N42, line: 75
op: +, line: 75

State: 5
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: M1, line: 84
op: N44, line: 84
op: *, line: 84
  Xform: 16, func: RTMult8, Unit: Mult_CSA8_1
op: M2, line: 93
op: N44, line: 93
op: *, line: 93

State: 6
  Xform: 7, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 102
op: N47, line: 102
op: +, line: 102
  Xform: 17, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 129
op: N46, line: 129
op: +, line: 129

State: 7
  Xform: 8, func: RTAdd8, Unit: Adder_RPL8_2
op: N41, line: 111
op: N48, line: 111
op: +, line: 111
  Xform: 18, func: RTAdd8, Unit: Adder_RPL8_1
op: N43, line: 147
op: N49, line: 147
op: +, line: 147

State: 8
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: M3, line: 138
op: N51, line: 138
op: *, line: 138
  Xform: 19, func: RTMult8, Unit: Mult_CSA8_1
op: M4, line: 156
op: N52, line: 156
op: *, line: 156
  Xform: 28, func: RTAdd8, Unit: Adder_RPL8_2
op: N44, line: 120
op: N48, line: 120
op: +, line: 120

State: 9
  Xform: 10, func: RTAdd8, Unit: Adder_RPL8_2
op: N40, line: 174
op: N53, line: 174
op: +, line: 174
  Xform: 20, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 201
op: N54, line: 201
op: +, line: 201

State: 10
  Xform: 11, func: RTAdd8, Unit: Adder_RPL8_2
op: N55, line: 183
op: N48, line: 183
op: +, line: 183
  Xform: 21, func: RTAdd8, Unit: Adder_RPL8_1
op: N49, line: 228
op: N63, line: 228
op: +, line: 228

State: 11
  Xform: 12, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 210
op: N59, line: 210
op: +, line: 210
  Xform: 22, func: RTAdd8, Unit: Adder_RPL8_1
op: N40, line: 192
op: N55, line: 192
op: +, line: 192

State: 12
  Xform: 13, func: RTMult8, Unit: Mult_CSA8_2
op: M7, line: 264
op: N60, line: 264
op: *, line: 264
  Xform: 23, func: RTAdd8, Unit: Adder_RPL8_2
op: N38, line: 246
op: N64, line: 246
op: +, line: 246
  Xform: 24, func: RTMult8, Unit: Mult_CSA8_1
op: M5, line: 219
op: N56, line: 219
op: *, line: 219
  Xform: 25, func: RTAdd8, Unit: Adder_RPL8_1
op: N39, line: 237
op: N63, line: 237
op: +, line: 237

State: 13
  Xform: 14, func: RTAdd8, Unit: Adder_RPL8_2
op: N18, line: 292
op: N61, line: 292
op: +, line: 292
  Xform: 26, func: RTMult8, Unit: Mult_CSA8_2
op: M6, line: 283
op: N65, line: 283
op: *, line: 283
  Xform: 27, func: RTMult8, Unit: Mult_CSA8_1
op: M8, line: 319
```

```
op: N67, line: 319
op: *, line: 319
  Xform: 29, func: RTAdd8, Unit: Adder_RPL8_1
op: I, line: 255
op: N57, line: 255
op: +, line: 255

State: 14
  Xform: 15, func: RTAdd8, Unit: Adder_RPL8_2
op: N60, line: 301
op: N18, line: 301
op: +, line: 301
  Xform: 30, func: RTAdd8, Unit: Adder_RPL8_1
op: N38, line: 310
op: N66, line: 310
op: +, line: 310

State: 15
  Xform: 31, func: RTAdd8, Unit: Adder_RPL8_2
op: N50, line: 165
op: N49, line: 165
op: +, line: 165
  Xform: 32, func: RTAdd8, Unit: Adder_RPL8_1
op: N55, line: 274
op: N58, line: 274
op: +, line: 274

State: 16
  Xform: 33, func: RTAdd8, Unit: Adder_RPL8_2
op: N65, line: 328
op: N38, line: 328
op: +, line: 328
  Xform: 34, func: RTAdd8, Unit: Adder_RPL8_1
op: N63, line: 337
op: OTT, line: 337
op: +, line: 337
```

C.2 Results for Fast Fourier Transform example

```
----------------
---Allocation---
AddSub_CLA8 1
Mult_CSA8 1
clock 25

---Schedule---
State: 1
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_1
op: le, line: 84
op: 2, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_1
op: le, line: 21
op: i, line: 21
op: +, line: 21

State: 2
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_1
op: le, line: 84
op: 2, line: 84
op: *, line: 84
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 3
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_1
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 4
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_1
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 5
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57

State: 6
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 7
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66

State: 8
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 66

State: 9
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
```

```
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 10
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 11
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 12
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 13
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 1
Mult_CSA8 1
clock 50

---Schedule---
State: 1
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_1
op: le, line: 84
op: 2, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_1
op: le, line: 21
op: i, line: 21
op: +, line: 21

State: 2
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 3
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_1
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 4
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57

State: 5
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 66

State: 6
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 7
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_1
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 8
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 2
clock 25

---Schedule---
State: 1
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_2
op: le, line: 84
op: 2, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
```

```
op: +, line: 30

State: 2
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_2
op: le, line: 84
op: 2, line: 84
op: *, line: 84

State: 3
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: TmReal, line: 66
op: Wptr_Real, line: 66
op: *, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_2
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 4
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: TmReal, line: 66
op: Wptr_Real, line: 66
op: *, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 5
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 6
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 7
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 2
clock 50

---Schedule---
State: 1
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_1
op: le, line: 84
op: 2, line: 84
op: *, line: 84

State: 2
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 3
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
```

```
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_1
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 4
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 2
clock 50
nochain

---Schedule---
State: 1
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_2
op: le, line: 84
op: 2, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 2
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57

State: 3
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_2
op: xmImag, line: 39
op: xuImag, line: 39
```

```
op: +, line: 39

State: 4
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 5
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 2
clock 50
nomulticycle

---Schedule---
State: 1
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_2
op: le, line: 84
op: 2, line: 84
op: *, line: 84

State: 2
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 3
```

```
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_2
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 4
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 2
clock 50
exclusive

---Schedule---
State: 1
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_2
op: le, line: 84
op: 2, line: 84
op: *, line: 84

State: 2
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30
```

```
State: 3
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_2
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 4
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_1
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 2
clock 50
nochain
nomulticycle

---Schedule---
State: 1
  Xform: 8, func: RTMult8, Unit: Mult_CSA8_2
op: le, line: 84
op: 2, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 2
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57

State: 3
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
```

```
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_2
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 4
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_2
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 5
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 1
Shifter8 1
clock 25

---Schedule---
State: 1
  Xform: 8, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 84
op: le, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 2
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_2
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 3
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57

State: 4
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 5
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 6
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66

State: 7
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66

State: 8
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 9
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 10
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 11
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75
```

```
State: 12
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 1
Shifter8 1
clock 50

---Schedule---
State: 1
  Xform: 8, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 84
op: le, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 2
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_2
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 3
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_1
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57

State: 4
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 5
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66

State: 6
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 7
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 8
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 1
Shifter8 1
clock 50
nochain

---Schedule---
State: 1
  Xform: 8, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 84
op: le, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 2
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_1
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 3
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 4
```

```
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_2
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 5
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66

State: 6
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 7
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 1
Shifter8 1
clock 50
nomulticycle

---Schedule---
State: 1
  Xform: 8, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 84
op: le, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 2
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_1
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 3
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 4
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_2
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 5
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 6
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 1
Shifter8 1
clock 50
exclusive

---Schedule---
State: 1
  Xform: 8, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 84
op: le, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 2
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
```

```
op: xuReal, line: 48
op: -, line: 48
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_1
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 3
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 4
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_2
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 5
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 6
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 75

----------------
---Allocation---
AddSub_CLA8 2
Mult_CSA8 1
Shifter8 1
clock 50
nochain
nomulticycle

---Schedule---
State: 1
  Xform: 8, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 84
op: le, line: 84
op: *, line: 84
  Xform: 9, func: RTAdd8, Unit: AddSub_CLA8_2
op: le, line: 21
op: i, line: 21
op: +, line: 21
  Xform: 10, func: RTAdd8, Unit: AddSub_CLA8_1
op: xlReal, line: 30
op: xuReal, line: 30
op: +, line: 30

State: 2
  Xform: 1, func: RTSub8, Unit: AddSub_CLA8_2
op: xlReal, line: 48
op: xuReal, line: 48
op: -, line: 48
  Xform: 11, func: RTAdd8, Unit: AddSub_CLA8_1
op: xmImag, line: 39
op: xuImag, line: 39
op: +, line: 39

State: 3
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 66
op: TmReal, line: 66
op: *, line: 66

State: 4
  Xform: 3, func: RTSub8, Unit: AddSub_CLA8_2
op: xmImag, line: 57
op: xuImag, line: 57
op: -, line: 57
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 75
op: TmReal, line: 75
op: *, line: 75

State: 5
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Imag, line: 66
op: TmImag, line: 66
op: *, line: 66

State: 6
  Xform: 5, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 66
  Xform: 6, func: RTMult8, Unit: Mult_CSA8_1
op: Wptr_Real, line: 75
op: TmImag, line: 75
op: *, line: 75

State: 7
  Xform: 12, func: RTSub8, Unit: AddSub_CLA8_2
op: -, line: 75
```

C.3 Results for Differential Equation Solver example

```
----------------
---Allocation---
ALU8 2
Mult_CSA8 2
OrGate1_2i 1
clock 55
max 100000
```

```
---Schedule---
State: 1
  Xform: 1, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 173
op: -, line: 173
op: dx_var, line: 173
op: x_var, line: 173
op: +, line: 173
op: <, line: 173
op: 127, line: 173
op: dx_var, line: 173
op: x_var, line: 173
op: +, line: 173
op: >, line: 173
op: ||, line: 173
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: x_var, line: 32
op: 2, line: 32
op: *, line: 32
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: dx_var, line: 23
op: u_var, line: 23
op: *, line: 23
  Xform: 24, func: RTAdd8, Unit: ALU8_1
op: dx_var, line: 182
op: x_var, line: 182
op: +, line: 182

State: 2
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: x_var, line: 32
op: 2, line: 32
op: *, line: 32
  Xform: 3, func: RTAdd8, Unit: ALU8_2
op: x_var, line: 41
op: t2a, line: 41
op: +, line: 41
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: y_var, line: 60
op: 2, line: 60
op: *, line: 60

State: 3
  Xform: 3, func: RTAdd8, Unit: ALU8_2
op: x_var, line: 41
op: t2a, line: 41
op: +, line: 41
  Xform: 5, func: RTMult8, Unit: Mult_CSA8_2
op: t2, line: 88
op: t1, line: 88
op: *, line: 88
  Xform: 8, func: RTAdd8, Unit: ALU8_1
op: y_var, line: 69
op: t3a, line: 69
op: +, line: 69

State: 4
  Xform: 5, func: RTMult8, Unit: Mult_CSA8_2
op: t2, line: 88
op: t1, line: 88
op: *, line: 88
  Xform: 6, func: RTSub8, Unit: ALU8_2
op: t4, line: 106
op: u_var, line: 106
op: -, line: 106
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_1
op: t3, line: 97
op: dx_var, line: 97
op: *, line: 97
  Xform: 15, func: RTOverFlow8, Unit: ALU8_1
op: 128, line: 79
op: -, line: 79
op: y_var, line: 79
op: t3a, line: 79
op: +, line: 79
op: <, line: 79
op: 127, line: 79
op: y_var, line: 79
op: t3a, line: 79
op: +, line: 79
op: >, line: 79
op: ||, line: 79

State: 5
  Xform: 6, func: RTSub8, Unit: ALU8_2
op: t4, line: 106
op: u_var, line: 106
op: -, line: 106
  Xform: 16, func: RTOverFlow8, Unit: ALU8_1
op: 128, line: 51
op: -, line: 51
op: x_var, line: 51
op: t2a, line: 51
op: +, line: 51
op: <, line: 51
op: 127, line: 51
op: x_var, line: 51
op: t2a, line: 51
op: +, line: 51
op: >, line: 51
op: ||, line: 51

State: 6
  Xform: 10, func: RTSub8, Unit: ALU8_2
op: t5, line: 125
op: t6, line: 125
op: -, line: 125
  Xform: 11, func: RTMult8, Unit: Mult_CSA8_2
op: dx_var, line: 144
op: u_var, line: 144
op: *, line: 144
  Xform: 13, func: RTSubOver8, Unit: ALU8_1
op: 128, line: 135
op: -, line: 135
op: t5, line: 135
op: t6, line: 135
op: -, line: 135
```

```
op: <, line: 135
op: 127, line: 135
op: t5, line: 135
op: t6, line: 135
op: -, line: 135
op: >, line: 135
op: ||, line: 135
  Xform: 17, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 50
op: ||, line: 51

State: 7
  Xform: 11, func: RTMult8, Unit: Mult_CSA8_2
op: dx_var, line: 144
op: u_var, line: 144
op: *, line: 144
  Xform: 12, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 154
op: -, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: <, line: 154
op: 127, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: >, line: 154
op: ||, line: 154
  Xform: 18, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 78
op: ||, line: 79

State: 8
  Xform: 12, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 154
op: -, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: <, line: 154
op: 127, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: >, line: 154
op: ||, line: 154
  Xform: 23, func: RTAdd8, Unit: ALU8_1
op: y1, line: 163
op: y_var, line: 163
op: +, line: 163

State: 9
  Xform: 14, func: RTSubOver8, Unit: ALU8_2
op: 128, line: 116
op: -, line: 116
op: t4, line: 116
op: u_var, line: 116
op: -, line: 116
op: <, line: 116
op: 127, line: 116
op: t4, line: 116
op: u_var, line: 116
op: -, line: 116
op: >, line: 116
op: ||, line: 116
  Xform: 19, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 115
op: ||, line: 116

State: 10
  Xform: 20, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 134
op: ||, line: 135

State: 11
  Xform: 21, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 153
op: ||, line: 154

State: 12
  Xform: 22, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 172
op: ||, line: 173

----------------
---Allocation---
ALU8 2
Mult_CSA8 2
OrGate1_2i 1
clock 55
exclusive
max 100000

---Schedule---
State: 1
  Xform: 1, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 173
op: -, line: 173
op: dx_var, line: 173
op: x_var, line: 173
op: +, line: 173
op: <, line: 173
op: 127, line: 173
op: dx_var, line: 173
op: x_var, line: 173
op: +, line: 173
op: >, line: 173
op: ||, line: 173
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_2
op: dx_var, line: 23
op: u_var, line: 23
op: *, line: 23
  Xform: 7, func: RTMult8, Unit: Mult_CSA8_1
op: y_var, line: 60
op: 2, line: 60
op: *, line: 60
  Xform: 24, func: RTAdd8, Unit: ALU8_1
```

```
op: dx_var, line: 182
op: x_var, line: 182
op: +, line: 182

State: 2
  Xform: 2, func: RTMult8, Unit: Mult_CSA8_2
op: x_var, line: 32
op: 2, line: 32
op: *, line: 32
  Xform: 8, func: RTAdd8, Unit: ALU8_2
op: y_var, line: 69
op: t3a, line: 69
op: +, line: 69
  Xform: 15, func: RTOverFlow8, Unit: ALU8_1
op: 128, line: 79
op: -, line: 79
op: y_var, line: 79
op: t3a, line: 79
op: +, line: 79
op: <, line: 79
op: 127, line: 79
op: y_var, line: 79
op: t3a, line: 79
op: +, line: 79
op: >, line: 79
op: ||, line: 79

State: 3
  Xform: 3, func: RTAdd8, Unit: ALU8_2
op: x_var, line: 41
op: t2a, line: 41
op: +, line: 41
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_2
op: t3, line: 97
op: dx_var, line: 97
op: *, line: 97
  Xform: 16, func: RTOverFlow8, Unit: ALU8_1
op: 128, line: 51
op: -, line: 51
op: x_var, line: 51
op: t2a, line: 51
op: +, line: 51
op: <, line: 51
op: 127, line: 51
op: x_var, line: 51
op: t2a, line: 51
op: +, line: 51
op: >, line: 51
op: ||, line: 51

State: 4
  Xform: 5, func: RTMult8, Unit: Mult_CSA8_2
op: t2, line: 88
op: t1, line: 88
op: *, line: 88
  Xform: 17, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 50
op: ||, line: 51

State: 5
  Xform: 6, func: RTSub8, Unit: ALU8_2
op: t4, line: 106
op: u_var, line: 106
op: -, line: 106
  Xform: 14, func: RTSubOver8, Unit: ALU8_1
op: 128, line: 116
op: -, line: 116
op: t4, line: 116
op: u_var, line: 116
op: -, line: 116
op: <, line: 116
op: 127, line: 116
op: t4, line: 116
op: u_var, line: 116
op: -, line: 116
op: >, line: 116
op: ||, line: 116
  Xform: 18, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 78
op: ||, line: 79

State: 6
  Xform: 10, func: RTSub8, Unit: ALU8_2
op: t5, line: 125
op: t6, line: 125
op: -, line: 125
  Xform: 13, func: RTSubOver8, Unit: ALU8_1
op: 128, line: 135
op: -, line: 135
op: t5, line: 135
op: t6, line: 135
op: -, line: 135
op: <, line: 135
op: 127, line: 135
op: t5, line: 135
op: t6, line: 135
op: -, line: 135
op: >, line: 135
op: ||, line: 135
  Xform: 19, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 115
op: ||, line: 116

State: 7
  Xform: 11, func: RTMult8, Unit: Mult_CSA8_2
op: dx_var, line: 144
op: u_var, line: 144
op: *, line: 144
  Xform: 20, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 134
op: ||, line: 135

State: 8
  Xform: 12, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 154
op: -, line: 154
op: y1, line: 154
op: y_var, line: 154
```

```
op: +, line: 154
op: <, line: 154
op: 127, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: >, line: 154
op: ||, line: 154
  Xform: 21, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 153
op: ||, line: 154
  Xform: 23, func: RTAdd8, Unit: ALU8_1
op: y1, line: 163
op: y_var, line: 163
op: +, line: 163

State: 9
  Xform: 22, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 172
op: ||, line: 173

----------------
---Allocation---
ALU8 2
Mult_CSA8 1
Shifter8 1
OrGate1_2i 1
clock 55
max 100000

---Schedule---
State: 1
  Xform: 1, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 173
op: -, line: 173
op: dx_var, line: 173
op: x_var, line: 173
op: +, line: 173
op: <, line: 173
op: 127, line: 173
op: dx_var, line: 173
op: x_var, line: 173
op: +, line: 173
op: >, line: 173
op: ||, line: 173
  Xform: 2, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 32
op: x_var, line: 32
op: *, line: 32
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: dx_var, line: 23
op: u_var, line: 23
op: *, line: 23
  Xform: 24, func: RTAdd8, Unit: ALU8_1
op: dx_var, line: 182
op: x_var, line: 182
op: +, line: 182

State: 2
  Xform: 3, func: RTAdd8, Unit: ALU8_2
op: x_var, line: 41
op: t2a, line: 41
op: +, line: 41
  Xform: 5, func: RTMult8, Unit: Mult_CSA8_1
op: t2, line: 88
op: t1, line: 88
op: *, line: 88
  Xform: 7, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 60
op: y_var, line: 60
op: *, line: 60
  Xform: 16, func: RTOverFlow8, Unit: ALU8_1
op: 128, line: 51
op: -, line: 51
op: x_var, line: 51
op: t2a, line: 51
op: +, line: 51
op: <, line: 51
op: 127, line: 51
op: x_var, line: 51
op: t2a, line: 51
op: +, line: 51
op: >, line: 51
op: ||, line: 51

State: 3
  Xform: 5, func: RTMult8, Unit: Mult_CSA8_1
op: t2, line: 88
op: t1, line: 88
op: *, line: 88
  Xform: 6, func: RTSub8, Unit: ALU8_2
op: t4, line: 106
op: u_var, line: 106
op: -, line: 106
  Xform: 8, func: RTAdd8, Unit: ALU8_1
op: y_var, line: 69
op: t3a, line: 69
op: +, line: 69
  Xform: 17, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 50
op: ||, line: 51

State: 4
  Xform: 6, func: RTSub8, Unit: ALU8_2
op: t4, line: 106
op: u_var, line: 106
op: -, line: 106
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_1
op: t3, line: 97
op: dx_var, line: 97
op: *, line: 97
  Xform: 14, func: RTSubOver8, Unit: ALU8_1
op: 128, line: 116
op: -, line: 116
op: t4, line: 116
op: u_var, line: 116
op: -, line: 116
op: <, line: 116
```

```
op: 127, line: 116
op: t4, line: 116
op: u_var, line: 116
op: -, line: 116
op: >, line: 116
op: ||, line: 116

State: 5
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_1
op: t3, line: 97
op: dx_var, line: 97
op: *, line: 97
  Xform: 10, func: RTSub8, Unit: ALU8_2
op: t5, line: 125
op: t6, line: 125
op: -, line: 125
  Xform: 15, func: RTOverFlow8, Unit: ALU8_1
op: 128, line: 79
op: -, line: 79
op: y_var, line: 79
op: t3a, line: 79
op: +, line: 79
op: <, line: 79
op: 127, line: 79
op: y_var, line: 79
op: t3a, line: 79
op: +, line: 79
op: >, line: 79
op: ||, line: 79

State: 6
  Xform: 10, func: RTSub8, Unit: ALU8_2
op: t5, line: 125
op: t6, line: 125
op: -, line: 125
  Xform: 11, func: RTMult8, Unit: Mult_CSA8_1
op: dx_var, line: 144
op: u_var, line: 144
op: *, line: 144
  Xform: 13, func: RTSubOver8, Unit: ALU8_1
op: 128, line: 135
op: -, line: 135
op: t5, line: 135
op: t6, line: 135
op: -, line: 135
op: <, line: 135
op: 127, line: 135
op: t5, line: 135
op: t6, line: 135
op: -, line: 135
op: >, line: 135
op: ||, line: 135
  Xform: 18, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 78
op: ||, line: 79

State: 7
  Xform: 12, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 154
op: -, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: <, line: 154
op: 127, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: >, line: 154
op: ||, line: 154
  Xform: 19, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 115
op: ||, line: 116
  Xform: 23, func: RTAdd8, Unit: ALU8_1
op: y1, line: 163
op: y_var, line: 163
op: +, line: 163

State: 8
  Xform: 20, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 134
op: ||, line: 135

State: 9
  Xform: 21, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 153
op: ||, line: 154

State: 10
  Xform: 22, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 172
op: ||, line: 173

----------------
---Allocation---
ALU8 2
Mult_CSA8 1
Shifter8 1
OrGate1_2i 1
clock 55
exclusive
max 100000

---Schedule---
State: 1
  Xform: 1, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 173
op: -, line: 173
op: dx_var, line: 173
op: x_var, line: 173
op: +, line: 173
op: <, line: 173
op: 127, line: 173
op: dx_var, line: 173
op: x_var, line: 173
op: +, line: 173
op: >, line: 173
op: ||, line: 173
```

```
  Xform: 2, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 32
op: x_var, line: 32
op: *, line: 32
  Xform: 4, func: RTMult8, Unit: Mult_CSA8_1
op: dx_var, line: 23
op: u_var, line: 23
op: *, line: 23
  Xform: 24, func: RTAdd8, Unit: ALU8_1
op: dx_var, line: 182
op: x_var, line: 182
op: +, line: 182

State: 2
  Xform: 3, func: RTAdd8, Unit: ALU8_2
op: x_var, line: 41
op: t2a, line: 41
op: +, line: 41
  Xform: 7, func: RTShiftL8, Unit: Shifter8_1
op: 2, line: 60
op: y_var, line: 60
op: *, line: 60
  Xform: 16, func: RTOverFlow8, Unit: ALU8_1
op: 128, line: 51
op: -, line: 51
op: x_var, line: 51
op: t2a, line: 51
op: +, line: 51
op: <, line: 51
op: 127, line: 51
op: x_var, line: 51
op: t2a, line: 51
op: +, line: 51
op: >, line: 51
op: ||, line: 51

State: 3
  Xform: 5, func: RTMult8, Unit: Mult_CSA8_1
op: t2, line: 88
op: t1, line: 88
op: *, line: 88
  Xform: 8, func: RTAdd8, Unit: ALU8_2
op: y_var, line: 69
op: t3a, line: 69
op: +, line: 69
  Xform: 15, func: RTOverFlow8, Unit: ALU8_1
op: 128, line: 79
op: -, line: 79
op: y_var, line: 79
op: t3a, line: 79
op: +, line: 79
op: <, line: 79
op: 127, line: 79
op: y_var, line: 79
op: t3a, line: 79
op: +, line: 79
op: >, line: 79
op: ||, line: 79
  Xform: 17, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 50
op: ||, line: 51

State: 4
  Xform: 6, func: RTSub8, Unit: ALU8_2
op: t4, line: 106
op: u_var, line: 106
op: -, line: 106
  Xform: 14, func: RTSubOver8, Unit: ALU8_1
op: 128, line: 116
op: -, line: 116
op: t4, line: 116
op: u_var, line: 116
op: -, line: 116
op: <, line: 116
op: 127, line: 116
op: t4, line: 116
op: u_var, line: 116
op: -, line: 116
op: >, line: 116
op: ||, line: 116
  Xform: 18, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 78
op: ||, line: 79

State: 5
  Xform: 9, func: RTMult8, Unit: Mult_CSA8_1
op: t3, line: 97
op: dx_var, line: 97
op: *, line: 97
  Xform: 19, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 115
op: ||, line: 116

State: 6
  Xform: 10, func: RTSub8, Unit: ALU8_2
op: t5, line: 125
op: t6, line: 125
op: -, line: 125
  Xform: 13, func: RTSubOver8, Unit: ALU8_1
op: 128, line: 135
op: -, line: 135
op: t5, line: 135
op: t6, line: 135
op: -, line: 135
op: <, line: 135
op: 127, line: 135
op: t5, line: 135
op: t6, line: 135
op: -, line: 135
op: >, line: 135
op: ||, line: 135

State: 7
  Xform: 11, func: RTMult8, Unit: Mult_CSA8_1
op: dx_var, line: 144
op: u_var, line: 144
op: *, line: 144
  Xform: 20, func: RTOr1_2i, Unit: OrGate1_2i_1
```

```
op: ovf_err, line: 134
op: ||, line: 135

State: 8
  Xform: 12, func: RTOverFlow8, Unit: ALU8_2
op: 128, line: 154
op: -, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: <, line: 154
op: 127, line: 154
op: y1, line: 154
op: y_var, line: 154
op: +, line: 154
op: >, line: 154
op: ||, line: 154
  Xform: 21, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 153
op: ||, line: 154
  Xform: 23, func: RTAdd8, Unit: ALU8_1
op: y1, line: 163
op: y_var, line: 163
op: +, line: 163

State: 9
  Xform: 22, func: RTOr1_2i, Unit: OrGate1_2i_1
op: ovf_err, line: 172
op: ||, line: 173
```