AUTOMATED CONSTRAINT-BASED HARDWARE ARCHITECTURE GENERATION FOR RECONFIGURABLE COMPUTING SYSTEMS

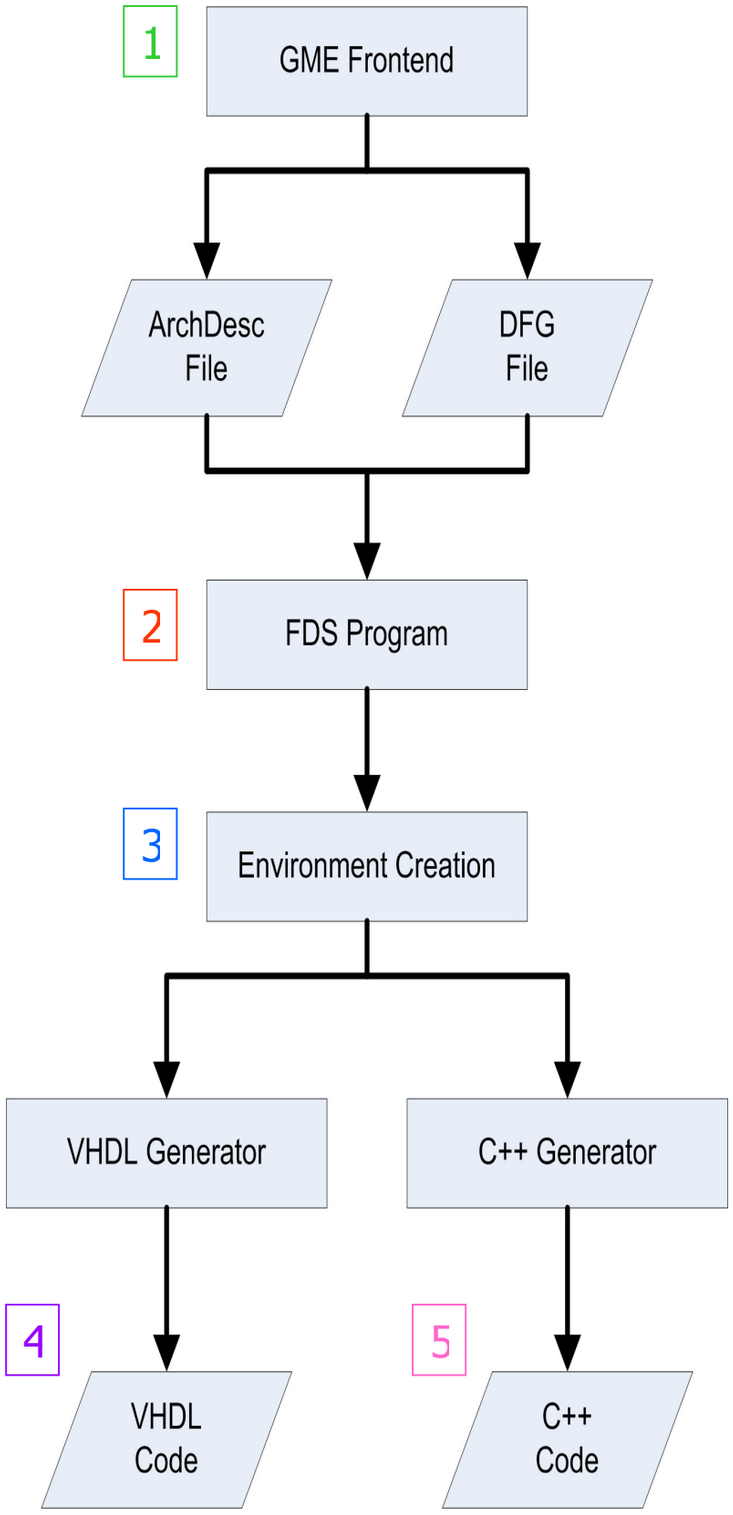
Matthew Areno, Brandon Eames, Aravind Dasu
Utah State University
Logan, UT 84322

Abstract

This paper proposes a novel approach to the automated generation of hardware micro-architectures targeting FPGA devices. The approach offers both high level architecture synthesis and VHDL generation based on dataflow graphs, as well as the generation of an archtiecture-level functional/performance simulator that can be used for quick and accurate testing of the design. We demonstrate the use of this approach through the derivation of architectures to implement computational kernals commonly encountered in Molecular Dynamics simulations.

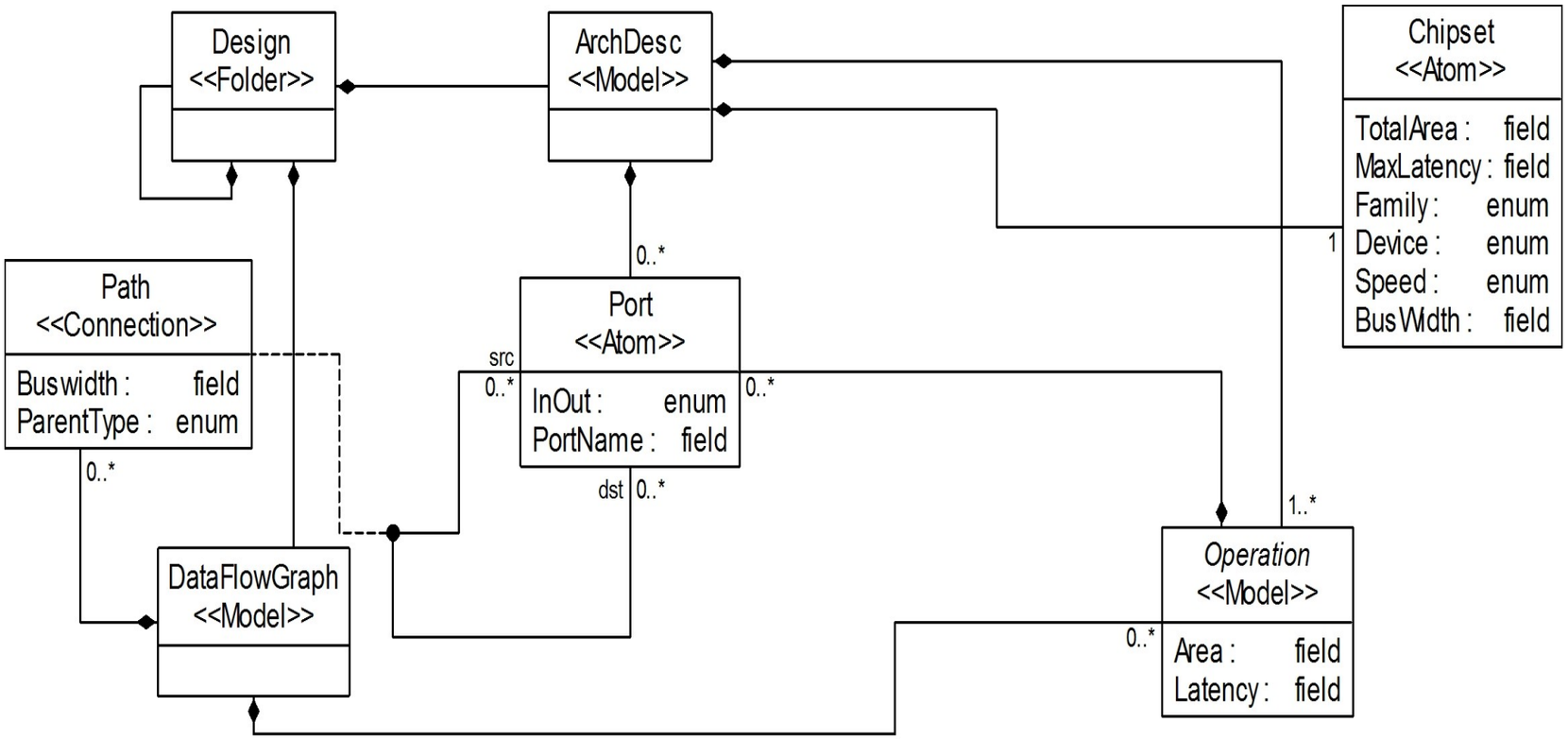
UtahState
UNIVERSITY

TOOL FLOW



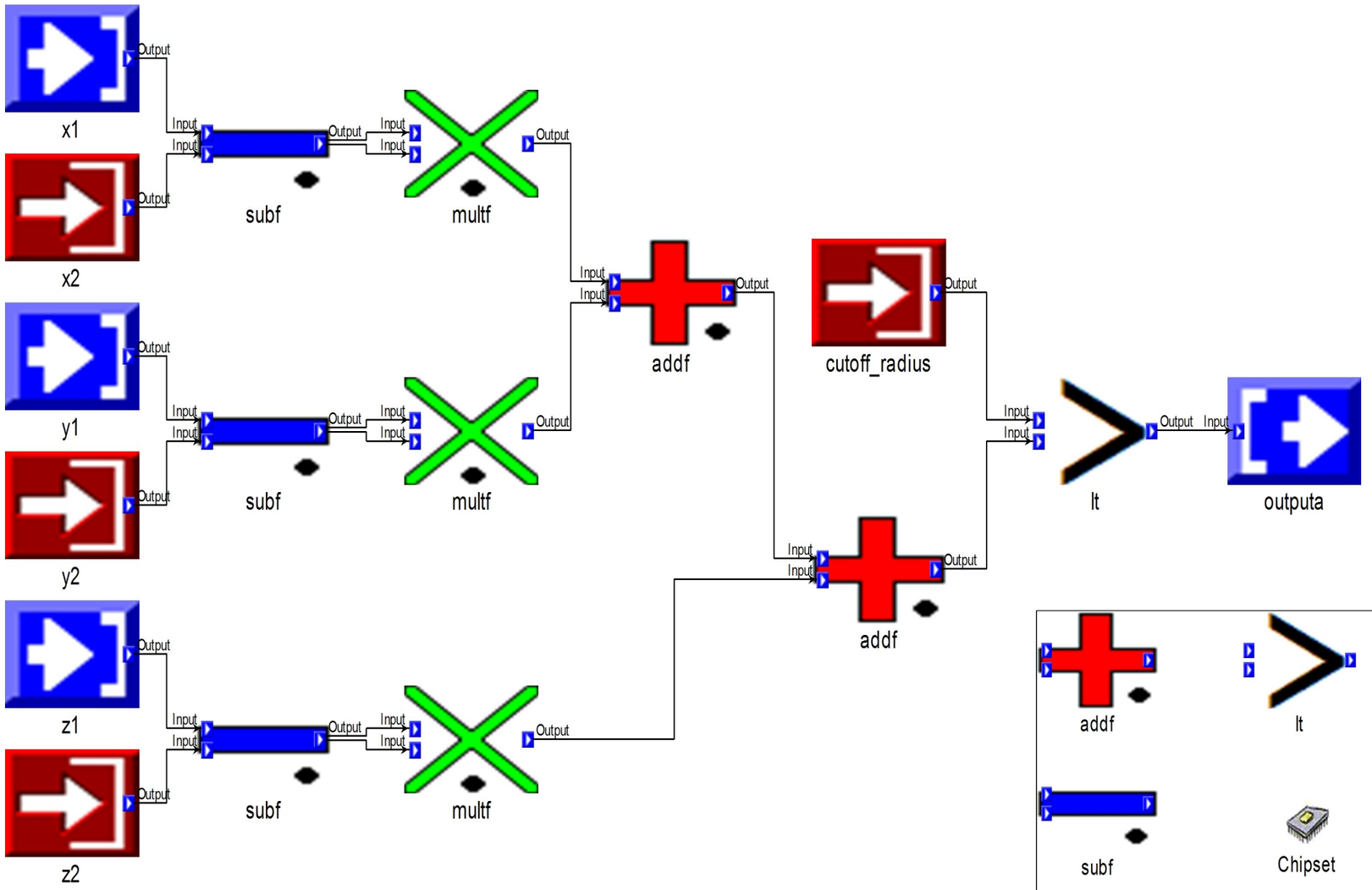
1. Front-end development
   A Graphical Modeling Environment (GME) front-end is used to interpret DFGs developed by the designer, as well as specified architectural constraints, and generate respective file representations of each.
3. Modified FDS
   A modified force-directed scheduling (FDS) implementation receives the DFG and ArchDesc files and derives a schedule for the DFG and determine the resource allocation necessary to facilitate this schedule.
5. Memory Allocation
   Information is then passed to an environment generator which initializes as variable required to properly generate VHDL and C++ implementations, as well as determining memory allocation requirements with the DFG.
7. VHDL Controller Generation
   From the generated schedule and environment, a VHDL representation in developed targeting the Xilinx ISE tool suite.
9. C++-based Simulator
   In addition to a VHDL representation, a C++-based simulator is generated to provide the designer with fast, clock-cycle accurate performance predictions on derived architectures.

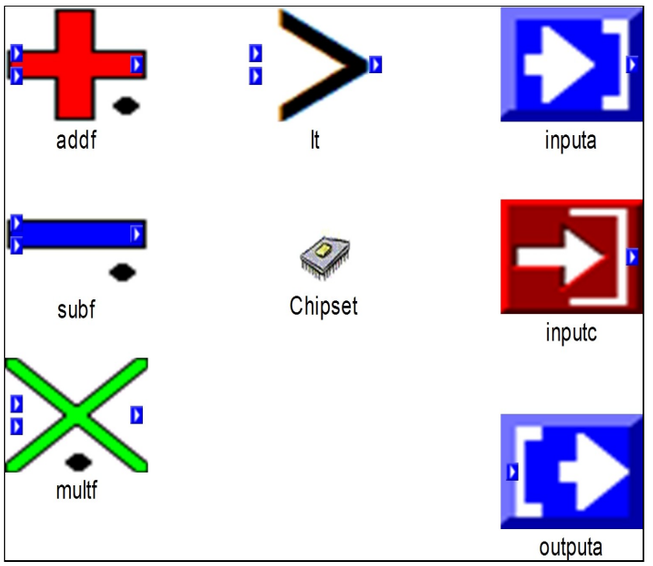
GME FRONT-END



- **The GME meta-model presents a GUI tool to the designer for the generation of DFGs.**
- **The meta-model supports the specification of multiple FPGA chips, as well as a family of IP hardware components, compatible with the Xilinx IP Coregen library.**
- **This tool provides a method for placing constraints on the system in order to guide the architecture generation process.**
- **Currently supported operations include: arithmetic (addition, subtraction, multiplication, division, square root), logic (less than, less than or equal, equal, equal to or greater, greater than, not equal), and IO (input, output, ram).**

GME FRONT-END (cont.)





- The GME meta-model allows for a single ArchDesc (right), used to specify constraints upon the operational units, and multiple DataFlowGraphs (above).
- Additionally, each design is required to have a Chipset, which may be used to assign area and latency constraints upon the system, as well as specifying the target FPGA platform.
- From these representations, text files are generated which may then be feed to the modified FDS program to determine a feasible schedule, as well as resource allocation requirements.

Modified FDS Algorithm

$let\ pred : N \rightarrow P\{N\}$ be the set of all predecessor nodes
$let\ succ : N \rightarrow P\{N\}$ be the set of all successor nodes
$let\ iter : N \rightarrow \mathbb{R}^+$ be the iteration value of a node
$let\ upper : N \rightarrow \mathbb{R}^+$ be the upper bound of a node
$let\ lower : N \rightarrow \mathbb{R}^+$ be the lower bound of a node

$\forall\ n \in N$:

if pred(n) == NULL

$$lower(n) = 0 + \frac{iter(n)}{\#of iterations} \times (cp_length_with_relaxation - cp_length_w/o_relaxation)$$

else

$$lower(n) = \min_{p \in pred(n)} st(p) + lat(p)$$

if succ(n) == NULL

$$upper(n) = critical_path_length_without_relaxation - lat(n) + \frac{iter(n)}{\#of iterations} \times (cp_length_with_relaxation - cp_length_w/o_relaxation)$$

else

$$upper(n) = \max_{p \in pred(n)} st(p) + lat(p)$$

- **Using the latency constraint imposed by the designer, the modified FDS program performs critical path relaxation in an attempt to reduce the area requirements incurred as a result of loop-unrolling.**

- **In order to derive a near-optimal architecture, a graph decomposition method was devised to balance the amount of additional time given to each node. The graph decomposition is performed based upon the associated iteration value of each node in comparison with the total number of iterations. This proportional value is then multiplied by the additional time allotted through critical-path relaxation, and the result is added to the earliest start time and latest start time of each node.**

MEMORY ALLOCATION

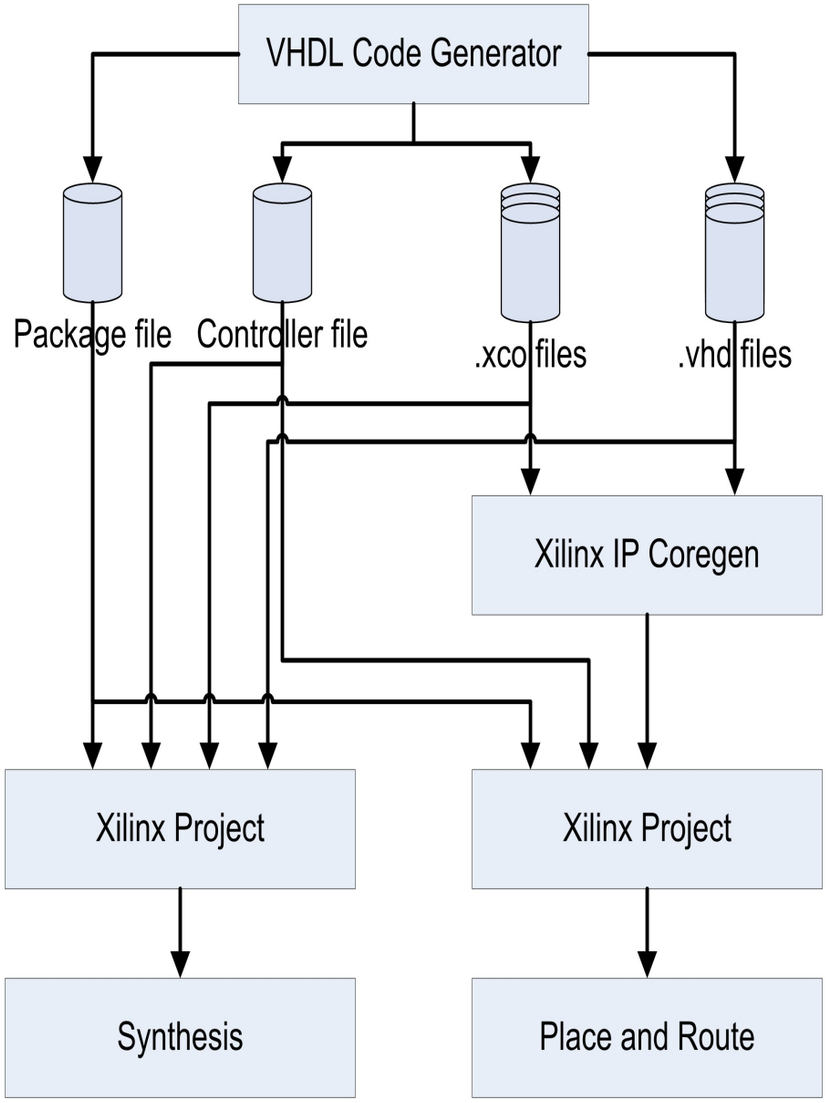
Algorithm 4.4 Algorithm for determining memory needs

$let\ pred : N \rightarrow P\{N\}$ be the set of all predecessor nodes
$let\ ct : N \rightarrow \mathbb{R}^+$ be the completion time of a node
$let\ lat : N \rightarrow \mathbb{R}^+$ be the latency of a node
$let\ st : N \rightarrow \mathbb{R}^+$ be the start time of a node

$\forall n \in N :$
    $\forall p \in pred(n) :$
        if st(n) != ct(p) + lat(p)
            $allocate\ memory\ unit\ of\ size\ st(n) - (ct(p) + lat(p))$

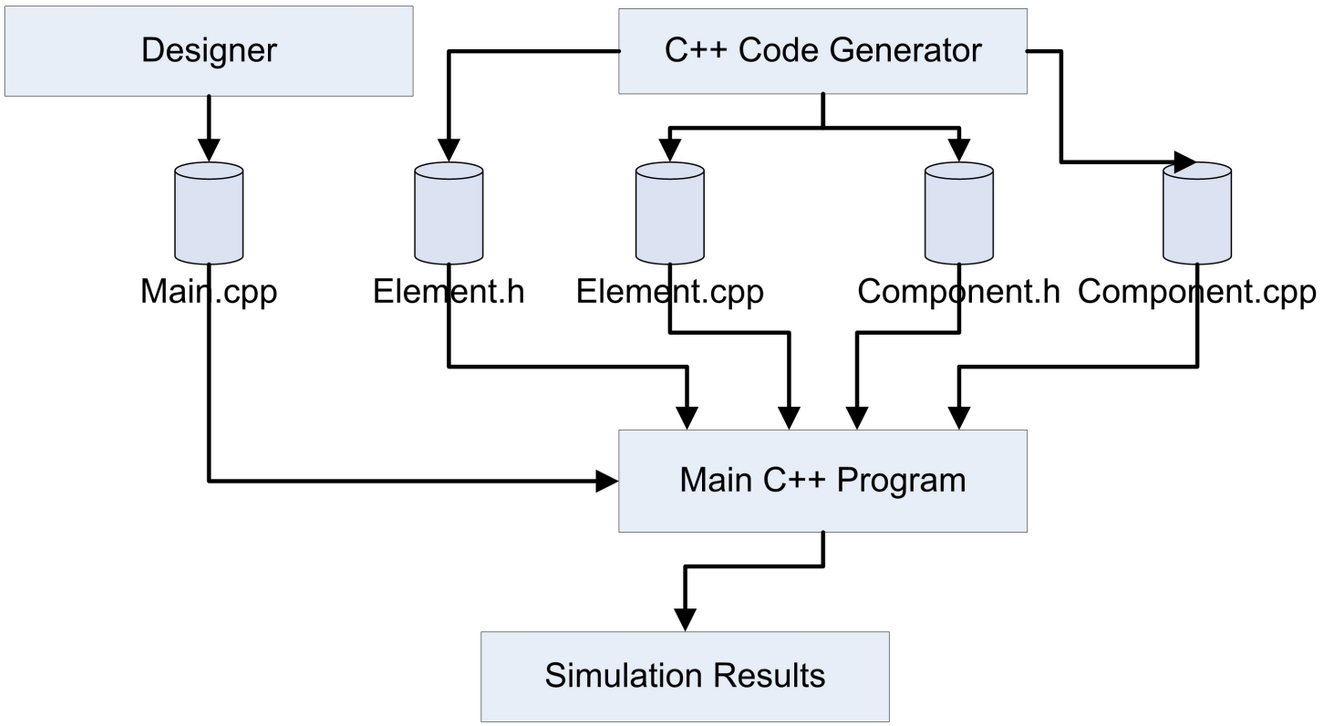
- **To determine when and where memory elements are needed, this algorithm checks the scheduled start time of each node with the scheduled start time, plus latency, of each predecessor. Any time these values are not equal, a memory element is required to maintain the value for the difference in time.**
- **Once all memory needs have been identified, an optimization can be performed to reduce the number of memory elements needed to handle the memory requirements of the architecture.**
- **Using shift registers as the primary data retention mechanism, memory needs of the same length of time may be accommodated using the same shift register when their start times are not equal.**
- **If equal memory needs exist that occur on identical clock cycles, a single memory element may be used by increase the input and output buswidth's, allowing for better utilization of block ram unit on a FPGA.**

VHDL Code Generation



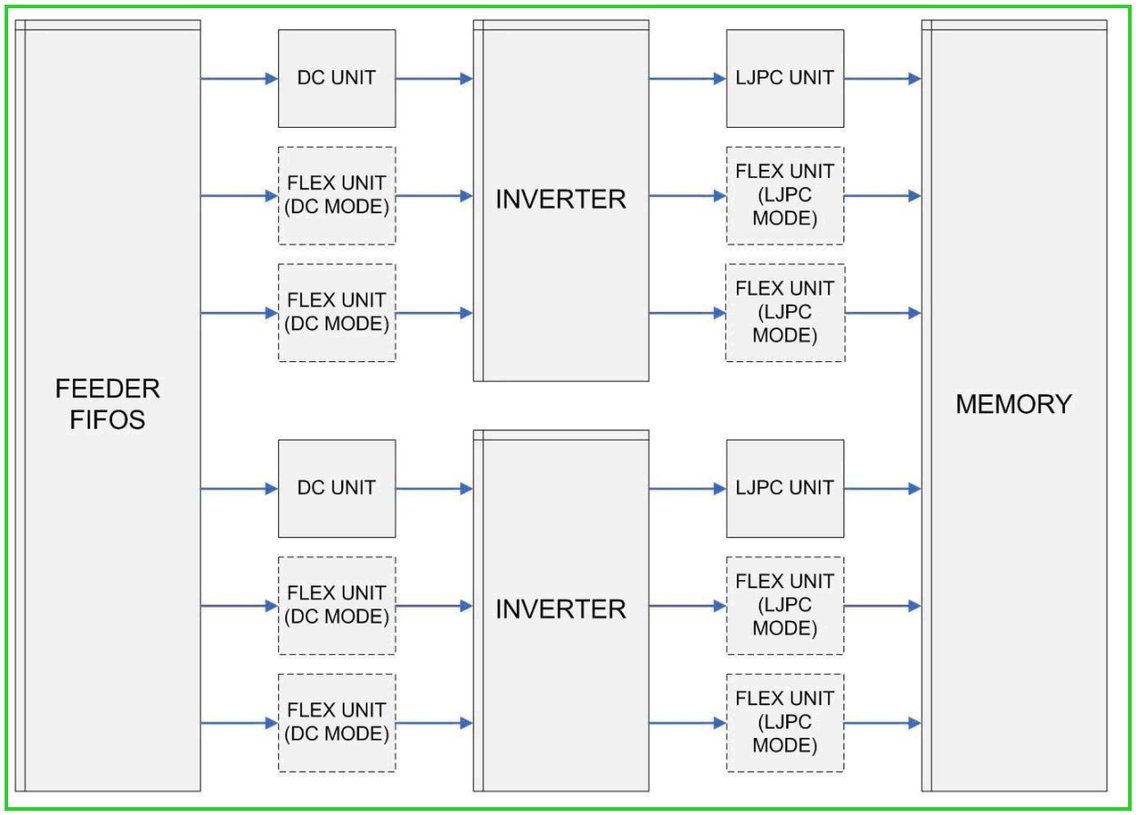
- The VHDL code generator converts the modified FDS output into VHDL by generating a controller and package file, as well as .xco, and .vhdl files.
- This group of file may then be added to a Xilinx project in order to perform synthesis on the design.
- In order to perform place and route of the design, the .xco and .vhd file must be feed through the Xilinx IP Coregen program in order to generate .ngc files required for timing and area calculations.
- Since the modified FDS program may generate multiple architectures, a separate directory is maintained for each implementation.
- The main controller file instantiates all required operational units, creates signals to connect to all necessary inputs and outputs, and handles all conditional execution cases.
- The main controller uses a case statement to handle the transfer of data within the system. Using a DFG node to operational unit mapping and the start time of each node, cases are developed that make the appropriate connections for the output of one operation unit to the input of another operational unit or units.

C++ Code Generation



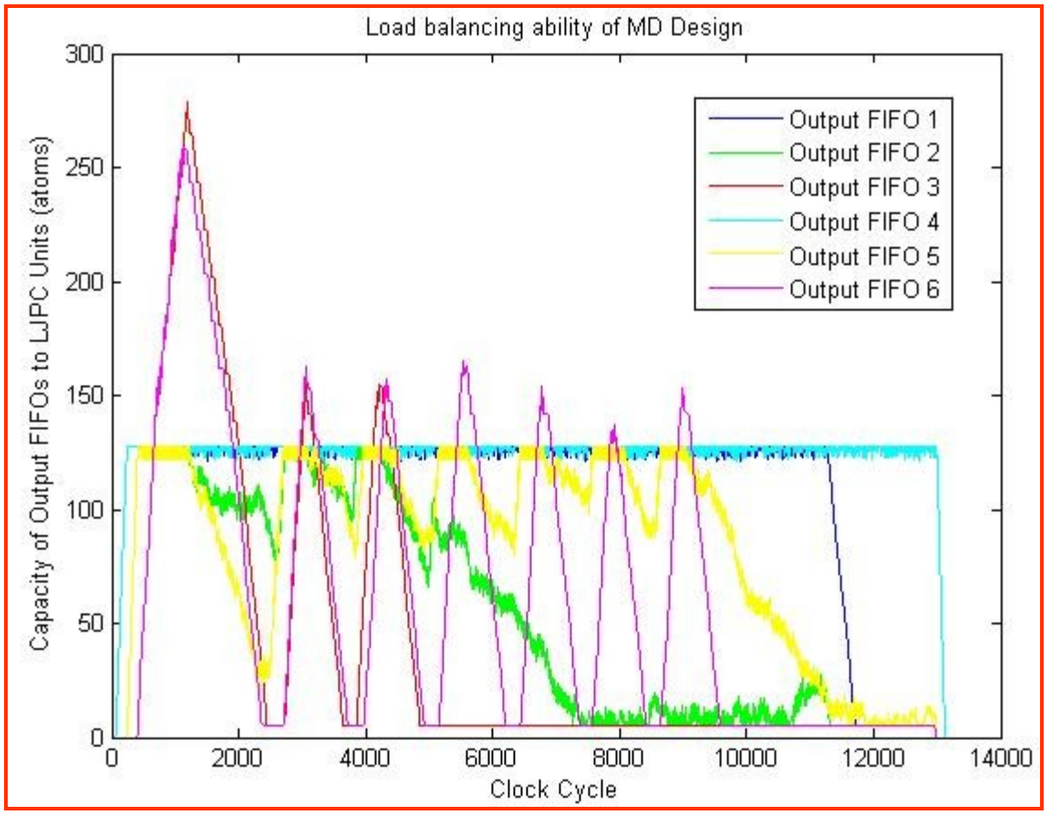
- **The C++ code generator is responsible for creating a design simulator that provides the designer with clock-cycle accurate analysis of the design implementation.**
- **The design is modeled using an Element class and Component class.**
- **The Element class models operational units, considering both the operation that is performed and the latency required.**
- **The Component class automatically instantiates all required elements and connections, and provides the designer with a single Execute function that may be used to run the simulation.**
- **The designer is currently responsible for developing a main function that will call the Component Execute function and provide it data. The development of this function is currently left to the designer, although the modified FDS program presents the designer with a timing for all inputs /outputs to/from the system. Automated generation of this function is a topic of future work.**

Results – MD Architecture

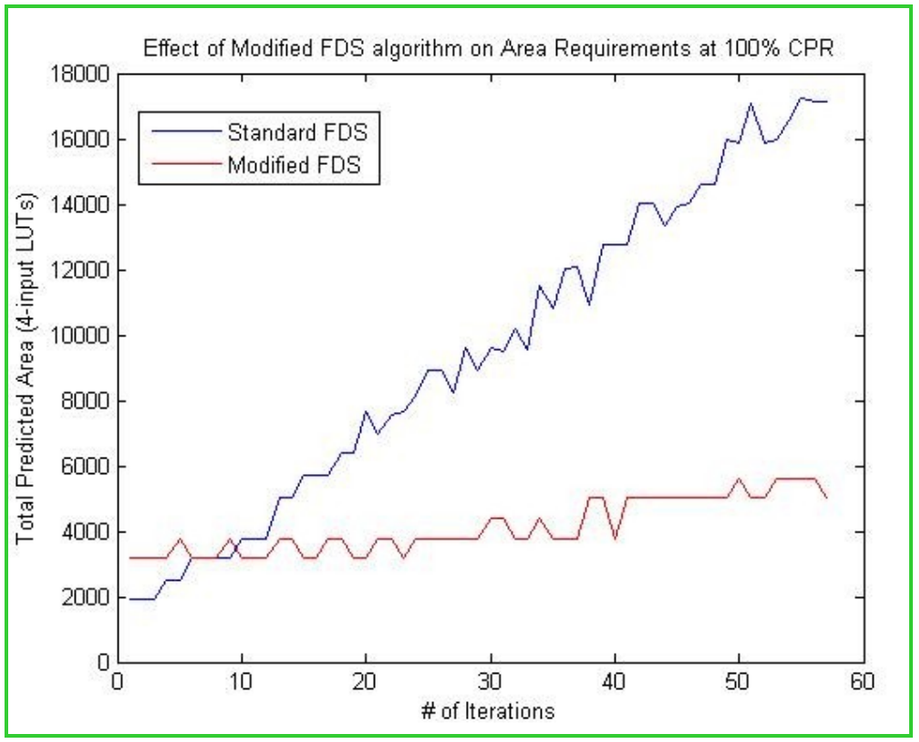


- To test the design tool, it was used on an existing MD architecture to verify the previous design.
- The primary elements to be tested were the distance calculator (DC) unit and Lennard-Jones Potential calculator (LJPC) unit.

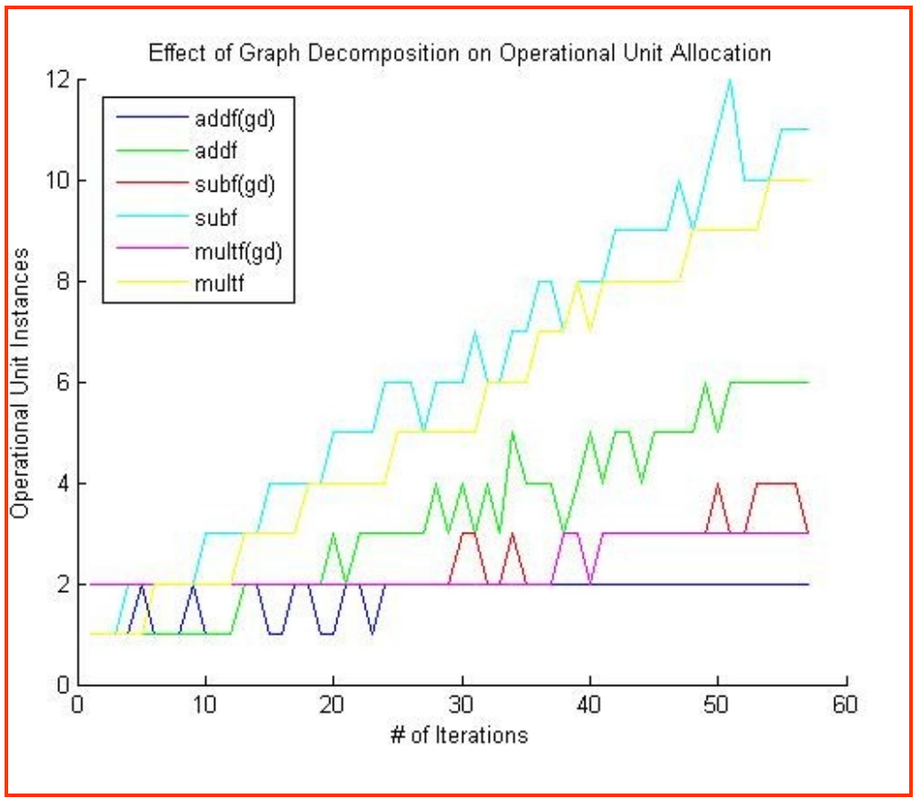
- The difficulty of MD designs is performing load-balancing on an N-body problem.
- The design shown above performs this through the use of FLEX units, as shown in the figure on the right.
- The automated creation of FLEX processors is the next step of development for this tool.

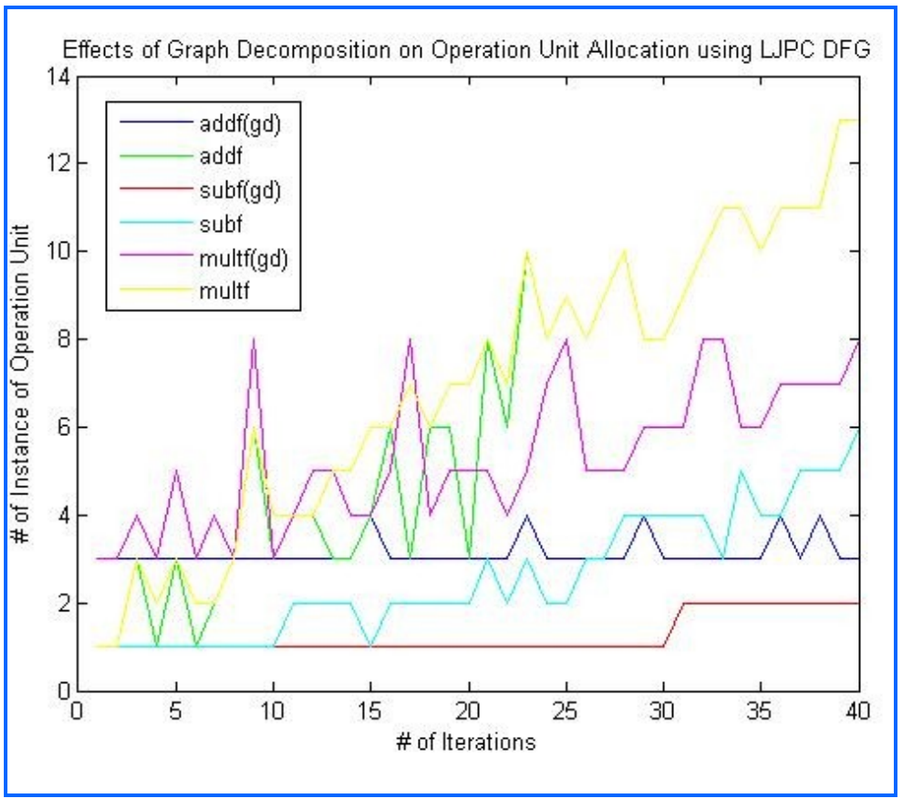


Results – Modified FDS Algorithm (Area)

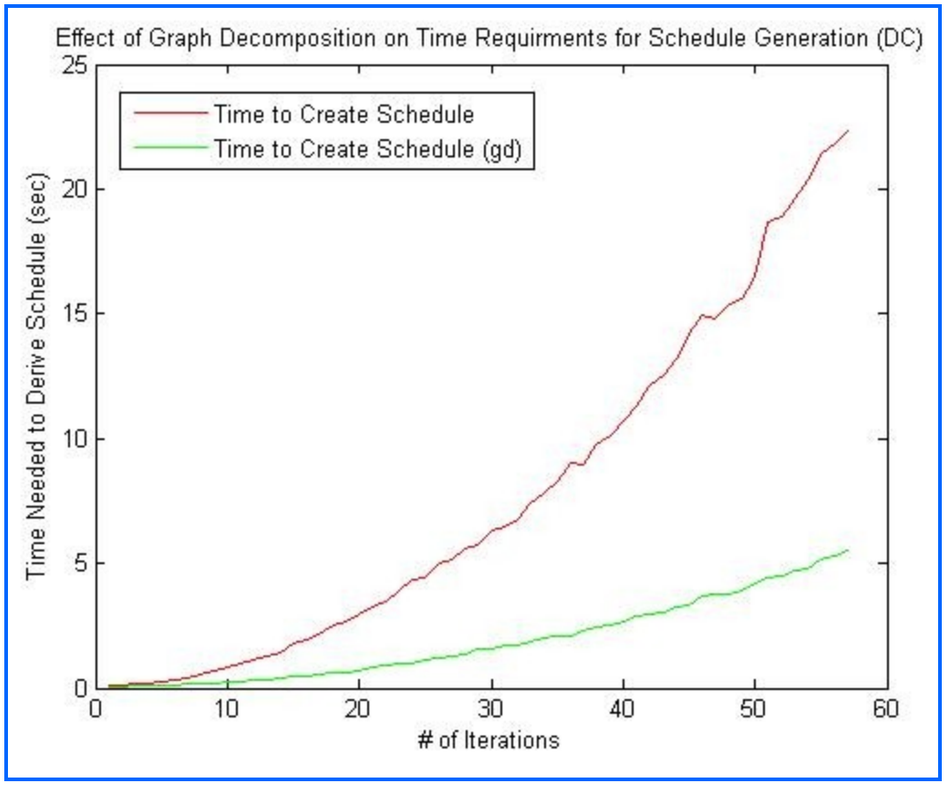


These graphs show the effect of the modified FDS algorithm vs. the standard DFG algorithm on area consumption. These graphs show the total area consumption for the DC unit (upper-left), as well as the number of adders, subtractors, and multipliers required for the DC unit (bottom-left) and LJPC unit (bottom-right) using 100% critical path relaxation (CPR).



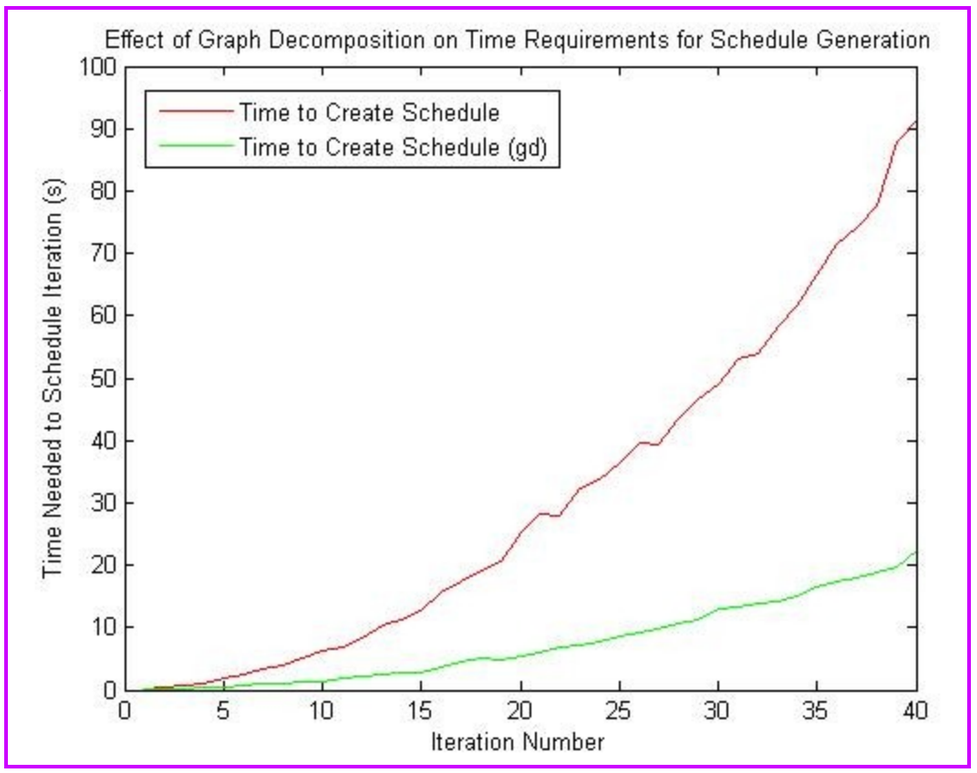


Results – Modified FDS Algorithm (time)

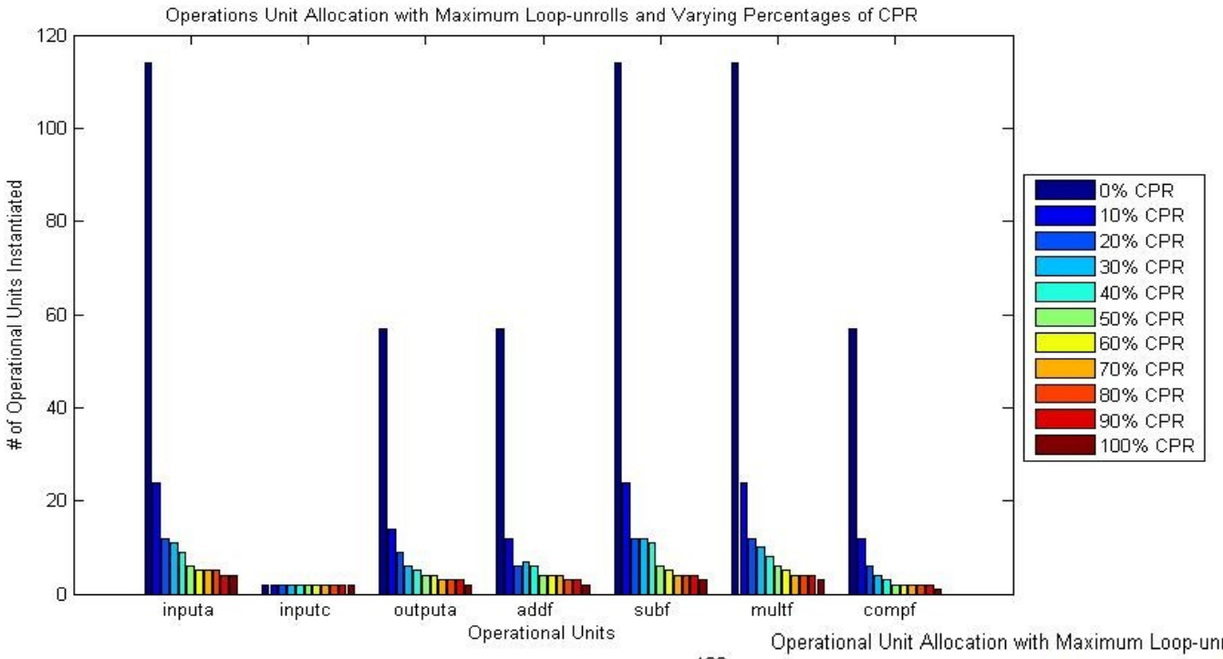


These graphs show the difference in the amount of time required to derive a schedule for 1 to 57 iterations of the DC unit (upper-left) and LJPC unit (lower-right) using 100% CPR. As can be seen from these graphs, the amount of time needed using the modified FDS, or graph decomposition (gd) method, is significantly smaller.

The reasoning behind this change is because of a reduction in the number of comparisons that must be performed by FDS when determining the best time to schedule a node within the DFG. By forcing subsequent iterations to start later than previous iterations, a reduction typically occurs in the number of nodes that can be scheduled at any one time.

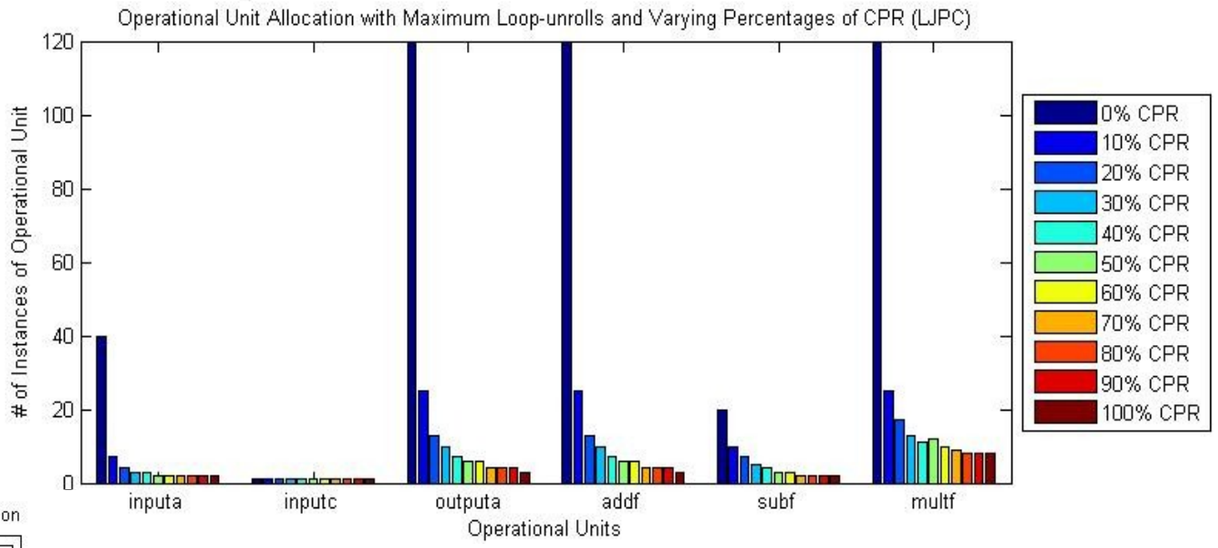


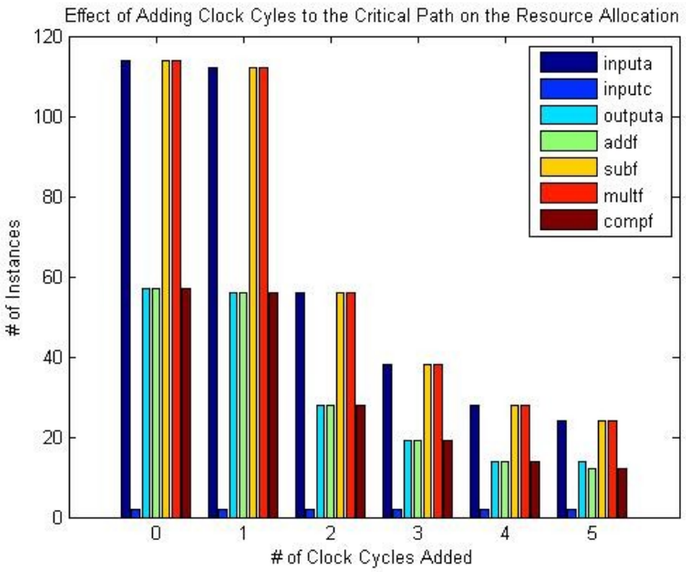
Results – Critical Path Relaxation



These graphs show the results of critical path relaxation (CPR) or resource allocation requirements. As seen, on a change from 0% to 10% reduces allocation requirements by nearly 80%.

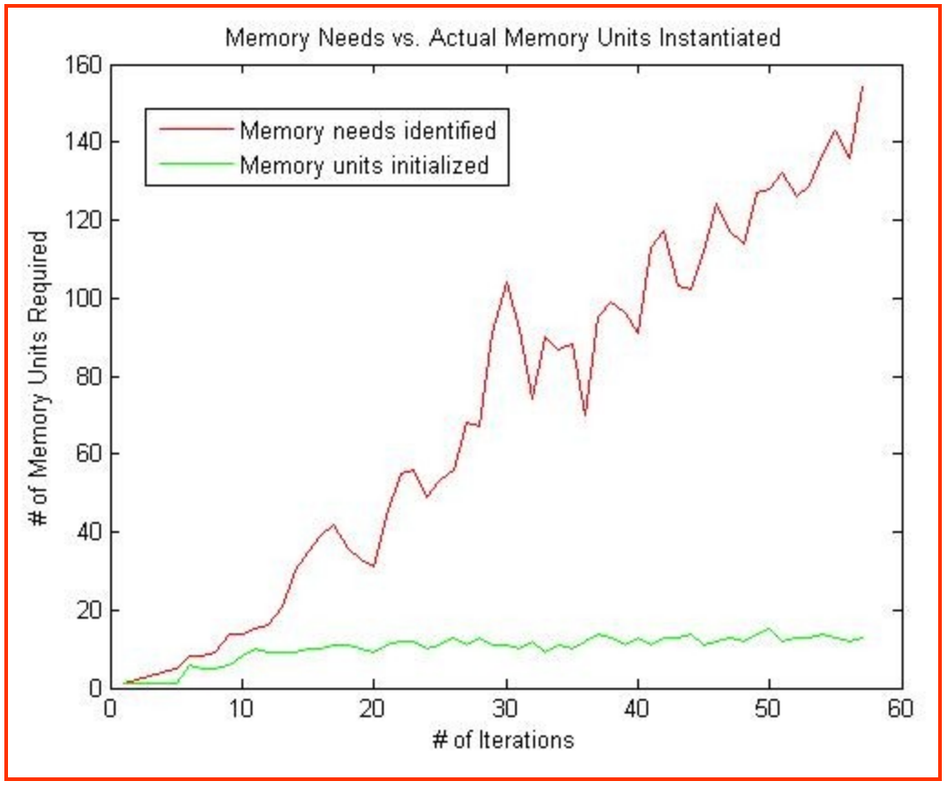
These results are evident in the resource allocation needs for the DC unit (top) and LJPC unit (middle). A close up from 0% to 5% is for the DC unit is shown below.





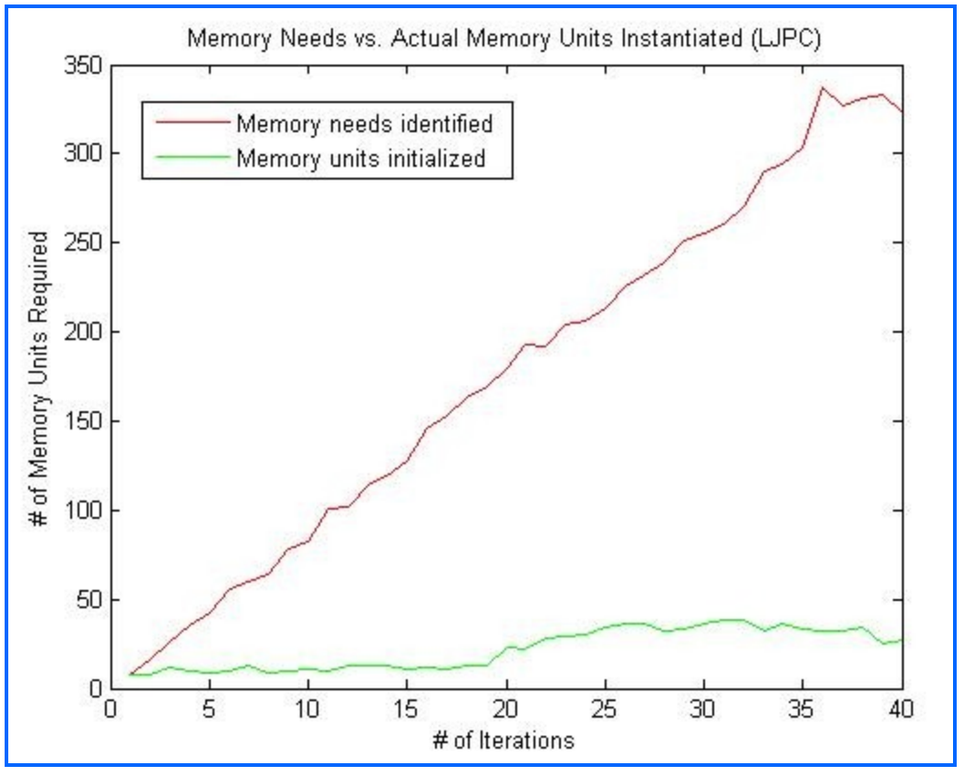
By analyzing the effect of CPR on system resource requirements, we can determine where designers get the most benefit from this type of area optimization. Using CPR, in conjunction with loop unrolling, performance metric on throughput can also be developed in an attempt to derive a trade-off analysis that better customizes a design based upon the constraints imposed by the designer.

Results – Memory Allocation



These graphs show the impact of the memory allocation algorithm used during the environment creation. The number of memory needs are shown in comparison to the number of units actually allocated to handle these needs. This is shown for varying numbers of iterations for both the DC unit (upper-left) and the LJPC unit (lower-right). As can be seen, the algorithm performs very well, providing a decrease of up to 90%.

The data shown here does not represent additional optimization that will be performed in the future, such as the combination of similar memory needs on the same clock cycle. Given the two additional optimizations planned, and the current results, it is believed at near-optimal memory allocation can be achieved.



Conclusions

- The tool presented has shown positive results through the use of the modified FDS algorithm and memory allocation method. The results have proven to be better that through the use of the standard FDS algorithm.
- The previously designed MD micro-architecture has been shown to be efficient, while only taking a fraction of the time to generate with this tool.
- VHDL and C++ implementation have been tested and shown to be both syntactically and functionally correct.
- Future work still remains to further expand the abilities of the tools, as well as to add better functionality, analysis, and constraint satisfaction. Further architecture analysis will be added to compare multiple architectures in order to better determine a near-optimal solution.
- The results shown have proven that this tool can strongly benefit hardware designers in properly analyzing strengths and weakness of different architectures, while only taking a fraction of the time.