FACTA UNIVERSITATIS
Series: **Electronics and Energetics** Vol. 27, N° 2, June 2014, pp. 235 - 249
DOI: 10.2298/FUEE1402235S

EXECUTION TIME – AREA TRADEOFF IN GAUSING RESIDUAL LOAD DECODER: INTEGRATED EXPLORATION OF CHAINING BASED SCHEDULE AND ALLOCATION IN HLS FOR HARDWARE ACCELERATORS

Anirban Sengupta[1], Reza Sedaghat[2], Vipul Kumar Mishra[1]

[1]Computer Science and Engineering, Indian Institute of Technology, Indore, India
[2]Electrical and Computer Engineering, Ryerson University, Toronto, Canada

Abstract. *Design space exploration is an indispensable segment of High Level Synthesis (HLS) design of hardware accelerators. This paper presents a novel technique for Area-Execution time tradeoff using residual load decoding heuristics in genetic algorithms (GA) for integrated design space exploration (DSE) of scheduling and allocation. This approach is also able to resolve issues encountered during DSE of data paths for hardware accelerators, such as accuracy of the solution found, as well as the total exploration time during the process. The integrated solution found by the proposed approach satisfies the user specified constraints of hardware area and total execution time (not just latency), while at the same time offers a twofold unified solution of chaining based schedule and allocation. The cost function proposed in the genetic algorithm approach takes into account the functional units, multiplexers and demultiplexers needed during implementation. The proposed exploration system (ExpSys) was tested on a large number of benchmarks drawn from the literature for assessment of its efficiency. Results indicate an average improvement in Quality of Results (QoR) greater than 26 % when compared to a recent well known GA based exploration method.*

Key words: *area; high level synthesis; exploration; scheduling; chaining; execution*

1. INTRODUCTION

As the complexity of Very Large Scale Integration (VLSI) designs increases, the design of Application Specific Integrated Circuits (ASIC) should be addressed at higher levels of abstraction in order to meet the growing challenges. Of late there has been a major shift among all well-known Electronic Design Automation (EDA) vendors from traditional Register Transfer Level (RTL) designs to high level synthesis. However, for comprehensive high level system designs, efficient design space exploration techniques are required during HLS that can concurrently meet the user specified constraints of

Received January 27, 2014
Corresponding author: Reza Sedaghat
Electrical and Computer Engineering, Ryerson University, Toronto, Canada
(e-mail: rsedagha@ee.ryerson.ca)

hardware area and execution time. Furthermore, design space exploration should also be able to concurrently resolve the orthogonal issues encountered during DSE, such as minimizing the time of the exploration process and maximizing the precision required. Hence, the tremendous advancement of highly complex digital VLSI circuits in the current generation of portable devices and other electronic products has mainly become possible owing to the efficient design techniques developed so far [1].

The process of HLS can be broadly classified into three phases. The first phase involves the conversion of the algorithm into Data Flow Graph (DFG). The second phase includes scheduling, which assigns operations into the appropriate control steps. Allocation, the third phase in high level synthesis, is the data-path synthesis that allocates hardware resources such as registers and busses, and binds the operations of DFG to functional units [1].

The HLS phase consists of interdependent tasks such as scheduling and allocation. Scheduling is the process of assigning the operations in specific control step while resource allocation refers to the assignment of the functional units to perform the operations, multiplexers and demultiplexers to switch between different inputs and output. However, the problem of solving the integrated scheduling and allocation by exhaustive analysis is strictly prohibited [1].

2. RELATED WORK

The problem of design space exploration was addressed in [2], where the authors have proposed the use of a genetic algorithm in the binding and allocation phase in high level synthesis. This method involves crossover dependence on the force directed data path binding completion algorithm. One of the problems with [2] is that the method accepts a scheduled data flow graph as an input. This clearly signifies the inability of their approach to resolve the scheduling problem. Authors in [3] have also proposed a genetic algorithm for time constrained scheduling. The chromosome is encoded with the permutation of operations, which is decoded by a list decoder, to decode the chromosome into a valid schedule. However, the approach does not handle chaining and execution time optimization. In addition, authors in [4] have proposed a problem space genetic algorithm for design space exploration of data paths. The authors have used the concept of heuristic/problem pair to convert a data flow graph into a valid schedule. The chromosome is encoded based on the 'work remaining' value of each node. One of the problems with approach [4] is that the second special parent chromosome built in correspondence with the minimum functional units (i.e. serial implementation) does not differ in the work remaining field of the first special chromosome. This may not always properly lead to reaching the optimal solution. Further, the cost function considers only latency and not total execution time. The problem of design space exploration was also addressed in [5] by suggesting order of efficiency, which assists in deciding preferences amongst the different Pareto optimal points. Research in [6] suggested that identification of a few superior design points from the Pareto set suffices for an excellent design process. Evolutionary algorithms in [7], such as the Genetic Algorithm (GA), have been suggested to yield better results for the design space exploration process. The use of GA has also been suggested as a framework for DSE of data paths in high level synthesis in [8]. Authors in this approach have proposed a priority order based chromosome for the data schedules and an independent chromosome for the functional units. Their work uses the robust search capabilities of the genetic algorithm for scheduling and

allocation of datapath with the aim to find a solution for both the module selection and scheduling. One of the drawbacks of [8] is that the approach does not consider resource binding. Thus, the cost function proposed does not reflect the multiplexer and demultiplexers' resources. Furthermore, like other GA design space exploration approaches, [8] only considers optimization of latency and area. Another approach introduced by researchers in [1] was also based on Pareto optimal analysis. According to their work, the design space was arranged in the form of an architecture vector design space for architecture variant analysis and optimization of performance parameters. Though the results proved promising the approach was unable to handle chaining based scheduling. Furthermore in [9] and [10], authors described another approach to DSE in high level systems based on binary encoding of the chromosomes. Work shown in [11] for DSE suggests that authors used an evolutionary algorithm for successful evaluation of the design for an application specific SoC. Approaches [9]-[11] only considered traditional latency and not the execution time constraint for data pipelining. The work shown in [12] discusses the optimization of area, delay and power in behavioral synthesis, but does not focus on the high level design space exploration using multi chromosomal genetic algorithm nor does it consider execution time during data pipelining. Furthermore, authors in [13] introduce a tool called SystemCoDesigner that offers rapid design space exploration with rapid prototyping of behavioral systemC models. Automated integration was developed by integrating behavioral synthesis into their design flow, while authors in [14] describe current state-of-the-art high-level synthesis techniques for dynamically reconfigurable systems. Additionally, authors in [15]-[17] also used genetic algorithms for scheduling and resource allocation for data path synthesis. Another class of scheduling methods employed previously was probabilistic in nature. For example the simulated annealing (SA) and simulated evolution (SE) based scheduling techniques have been used for the high level synthesis problem. Authors in [18], [19] have proposed simulated annealing scheduling method called 'SALSA' which uses many probabilistic search operators to enhance the performance of SA-based technique for high level synthesis problem. Moreover, authors have also proposed an extended binding model for handling the scheduling problem in high level synthesis. Furthermore, authors in [20] also used SA for scheduling problem with simultaneous minimization of registers and function units. SE has been proposed by authors in [21] for solving the combined problem of scheduling and resource allocation in high level synthesis. All aforementioned approaches [15]-[21], however, do not consider execution time, chaining and data pipelining. In contrast to the proposed approach, [15]-[17] do not incorporate a special seeding process based on serial and parallel implementation in order to efficiently guide the GA to optimal/near-optimal solution. Other previously proposed approaches [22], [23] are based on integer linear programming (ILP). Here, the computational complexity is massive and although able to provide good results, consume enormous time. Furthermore, the concept of data pipelining based on execution time was not shown during system trade-off. Constructive approaches [24]- [27] are very straightforward to implement but suffer from the major drawback of leading to poor quality of solutions owing to their greedy nature.

3. The Proposed Approach for Genetic Algorithm Based Exploration System (ExpSys)

The approach proposed in this paper for finding the optimal integrated scheduling, allocation, binding and module selection, employs a special multi chromosomal compound

chromosome structure that has the efficient ability to search the design space. It provides an integrated solution to the problem of scheduling, allocation and binding by yielding a set of hardware resources that contains the details of functional units (e.g. number and kind). Further, this solution reduces the cost function based on constraints provided for hardware area (consisting of function units, multiplexers, demultiplexers) and execution time (considering latency, cycle time and number of sets of data to be executed). In order to reduce the final cost, the module selection indicates the optimal number of resources needed of each kind, as well as the right version of a specific resource needed from the module library during implementation The ExpSys has been developed by a new chromosome encoding technique that consists of separate chromosome structures for each of the resources, rather than the traditional method consisting of a single chromosome structure to represent all the resources. Moreover the proposed approach also includes an independent chromosome representation of the module allocations fields.

3.1 The ExpSys overview

The input to the GA framework is the behavioral description of the dataflow graph (DFG), or the high level description of the algorithm in C language, that describes the behavior of the application.

In addition to the behavioral description of the application input to the GA framework also includes the set of user specified design constraints for hardware area and execution time (with the user specified weight factors for hardware area-execution time tradeoff), control parameters for the genetic algorithm, and the module library that contains specifically three different information viz. maximum resources available, clock cycles and area. The proposed framework is comprised of two basic units. The first unit is the proposed heuristic that acts as an input to the skeleton for the genetic algorithm. The second unit processes the information provided by the first unit to produce a final integrated scheduling, allocation and module selection solution. The proposed skeleton (algorithm) for the genetic algorithm is shown is Fig.1. It uses a new heuristic based on residual load criterion that assigns a specific priority for each operation in the chromosome structure. The first parent (P1) chromosome of the nodal string (this string is defined later in Section 4.2) is encoded based on the residual load (α) of each resource from the ASAP scheduling graph. On the contrary, each operation of the second parent (P2) nodal string is encoded based on the difference of the latency obtained by using ASAP scheduling with maximum resource (L^{ASAP}) and the residual load (α) for each operation (oi) obtained for P1 chromosome. Hence, the encoded value of each operation (oi) of the second parent chromosome is calculated using Equation (1).

$$\beta = L^{ASAP} - \alpha(o_i) \tag{1}$$

The rest of the parents of the population in the nodal string encoded with the residual load values are obtained by random perturbation. The other parent chromosomes (P3…..Pn) of the population obtained by the perturbation function should be individuals lying between the Parent P1 derived from the schedule based on maximum resource and Parent P2 derived based on minimum resource. This is more logical because the optimal solution to the integrated problem lies somewhere between the maximum and the minimum resource. The developed perturbation function, which yields the residual load values, is given in Equation (2)

$$PF = (\alpha + \beta)/2 \pm \mu \quad (2)$$

where 'μ' is a random value between 'α' and 'β'. The additional random value 'μ' is added to the perturbation function because, in order to have more diversity in the initial population, the residual load value for the rest of the parents (P3…..Pn) should be different (Note: This residual load value determines the priority among nodes during the decoding process. Thus, it is necessary to have different residual load values by adding the random value to the perturbation function). Moreover, having greater diversity results in searching all the corners of the design space, thereby assisting in finding the optimal/near-optimal solution. Ignoring 'μ' in the above function would encode the nodal string part for the rest of the parents (P3…..Pn) with the same residual load values, thereby reducing the diversity of the initial population. The function in Equation (2) is used when encoding the values of the nodal string for the rest of the parents. On the other hand, the perturbation of the resource allocation string (this string is defined later in Section 3.2) for the other parents is obtained by applying the algorithm shown below:

Algorithm

1) Schedule the DFG using ASAP algorithm and calculate the latency (L).
2) Generation G =1.
3) Creation of the initial population by chromosome encoding with priority list of nodes based on 'residual load' which is done as follows:
   a) Encode the first parent (P1) of the nodal string using the residual load (α) based on the ASAP schedule. Encode the first parent (P1) of resource allocation string with maximum resources.
   b) Encode the second parent (P2) of the nodal string using residual load (β) calculated as: $L^{ASAP} - \alpha\ (o_i)$ based on minimum resources. Encode the second parent (P1) of the resource allocation string with minimum resources.
   c) Create the rest of the parent (P3…Pn) of the nodal string with residual load based on the perturbation function = $(\alpha + \beta)/2 \pm \mu$; where 'μ' is a random value between 'α' and 'β'.
4) Perform crossover with very high probability (P_{cross}) among parents to create off-springs.
5) Decode the chromosomes using the proposed 'Residual load Heuristic' to find scheduling solutions by binding DFG operations to FU, allocating MUX's and DEMUX's.
6) Get information about the functional units (FU) such as versions, area occupied, clock cycle etc. from the module library.
7) Calculate the global cost function and determine the fitness of each individual. Global cost function considers A) Total Area which is a combination of: i) Area of FU ii) Area of MUX iii) Area of DEMUX. B) Total execution time which is a combination of, i) Latency ii) Cycle time and iii) Number of sets of data.
8) Perform mutation on the least fit nodal string chromosome and the resource allocation string chromosome with probability, $P_m = 0.25$. Mutation is performed once every generation
9) Decode the mutated chromosomes using the proposed 'Residual load Heuristic' to find scheduling solutions and then calculate the cost of the mutated chromosome again.
10) Select the best population from the set of off-springs and parents from this generation and take it forward to the next generation. Increment G, (G=G+1) until G< Generation $_{Max}$
11) End GA Run.

Fig. 1 The proposed skeleton for the ExpSys

Perturbation rule for the resource allocation chromosome for rest of the parents

1. Randomly pick any two nodes (v1, v2) from the chromosome that represents the resource allocation.
2. Randomly select any integer value (i) ranging between or equal to 'α' and 'β' for that specific operation (node). Hence, $\alpha <= i <= \beta$

Once the parents for the initial population are formed direct crossover is applied. Crossover results in creation of off-spring in that generation. For every mating between two parents, two off-springs can be created. If, for example, size of the parents in the population is 8, then 16 off-spring will be produced. Therefore, the total population of the first generation is 24. The next task is to decode the generated individuals of the first generation by applying a new 'residual load heuristic' that always results in a valid schedule. During the process of formation of the schedule solution, the data dependency is strictly followed before any operation is selected for scheduling. The global cost function is then determined in order to judge the fitness of each individual solution. The least fit individual is mutated in order to hope for a better solution. After mutation, the mutated chromosome is again decoded and its fitness is adjudged. The best fit individuals from this first generation are then forwarded to the next generation. This process continues until the maximum generation G(Max) specified in reached.

3.2 Chromosome representation

Suitable encoding of the problem dictates the capability of the genetic algorithm to find optimal or near–optimal solutions. The proposed approach uses a multi chromosome structure consisting of independent strings to separately represent the priority of the nodes of the DFG for each FU type and the resource allocation information. The approach is called multi chromosomal because each FU (resource) is represented as an independent substring in the nodal string structure. It has two independent strings to separately represent the nodes of the DFG (called 'nodal string') and the resource allocation (called 'resource allocation string'). The 'nodal string' contains the residual load values of each node which will determine the priority of the nodes during scheduling. The 'residual load heuristic' is used when decoding the nodal string in order to obtain a valid scheduling solution. The 'resource allocation string' contains a list of integers, which indicate the maximum number of resources allowed during scheduling. The resource allocation string contains a substring with integers to represent the maximum number of functional units of each type available for scheduling in every time step of the schedule. This encoding scheme for both the resource allocation string and nodal string assures that the genetic algorithm always produces a valid schedule as well as reaching all the corners of the design space to explore the integrated solution of scheduling, allocation and binding. The encoding scheme for the 'nodal string' and the 'resource allocation string' is shown with an example of a benchmark 'Differential Equation Solver'. Small values of delay in cc are used during demonstration. For clarity, during experimentation real values have been used. The schedule of the DFG of the differential equation solver using ASAP is shown in Fig.2. The latency (L) obtained is 12cc (Note: Assumes multipliers and adders/subtractors take 4cc and 2cc respectively). The corresponding chromosome encoding for the first parent (P1) of the nodal string is shown in Fig. 3(a). The total residual load of each operation (node) is obtained by summation of the residual load of the successor operations following that node. E.g. for

node 1, the residual load is (4+4+2+2) cc = 12cc. The second parent (P2) chromosome is encoded based on the residual load values obtained using Equation (1). The second parent (P2) chromosome encoding is shown in Fig. 3(b). The rest of the parents of the initial population is obtained using Equation (2) which is a perturbation function used to encode the residual load values. The residual load values for rest of the parents always lie between the values from the first parent and second parent. This scheme has been developed because the optimal solution to the problem should always lie between the serial and maximally parallel implementation [4]. On the other hand, the first parent (P1) shown in Fig. 3(a) and second parent (P2) of the resource allocation string shown in Fig. 3(b) are based on the user specified maximum and minimum resources respectively. For example, the first parent (P1) of the resource allocation string shown in Fig. 3(a) consists of three multipliers, three adders, two subtractors and one comparator. Additionally, second parent (P2) of the resource allocation string shown in Fig. 3(b) consists of one multiplier, one adder, one subtractor, and one comparator. The rest of the parents (P3…P8) of the 'resource allocation string' are obtained using the algorithm in Fig 3. The 'resource allocation string' for the rest of the parents of the initial population is also encoded with multiplier, adder, subtractor, and comparator option (Note: 'M', 'A', 'S', 'C' refers to multipliers, adders, subtractor, and comparators respectively in the resource allocation string). Thus, the final solution found by the proposed ExpSys is able to indicate the final combination of multipliers, adders, subtractor, and comparators needed to implement the problem based on the user specified hardware area and execution time constraints. The nodal string and the resource allocation string for the rest of the parents are shown in Fig.4(a) and Fig.4(b) respectively. For example, in case of Fig 4(a), the encoding of the third parent for the resource allocation string is obtained by first picking up randomly any two nodes M (multiplier) & A (adder) and then randomly selecting any integer value between '3' and '1' for M and between '3' and '1' for A. The randomly selected value for both M & A is '2'. Similarly, the rest of the parent chromosomes can be built by perturbation. This type of perturbation for the 'resource allocation string' and the perturbation function for the 'nodal string' described before aids in searching all the possible combinations of the design space so that the GA can reach an optimal or near-optimal solution.

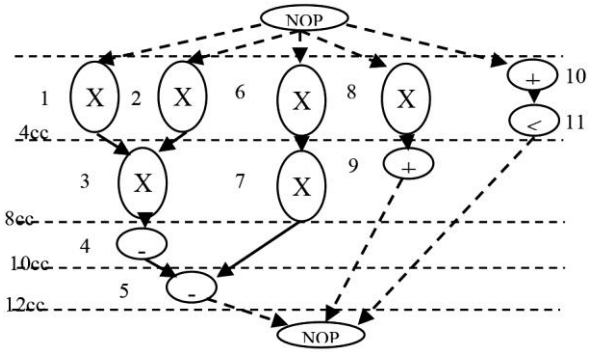

Fig. 2 Scheduling of Differential equation solver using ASAP

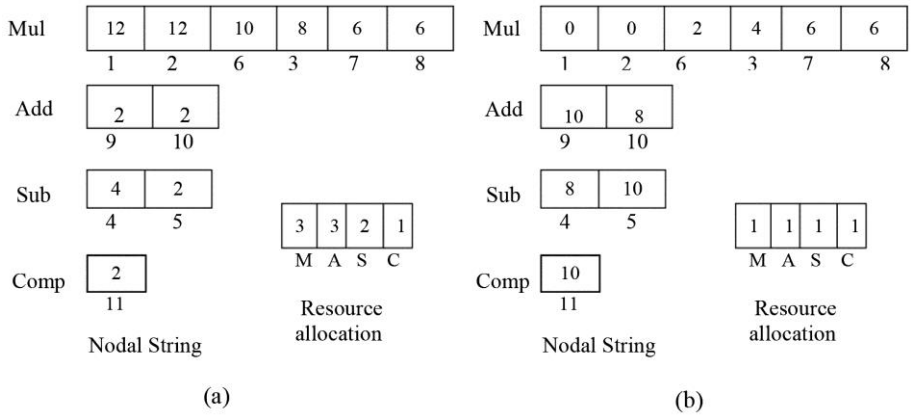

Fig. 3 Chromosome encoding for the first parent (a) and second parents (b)

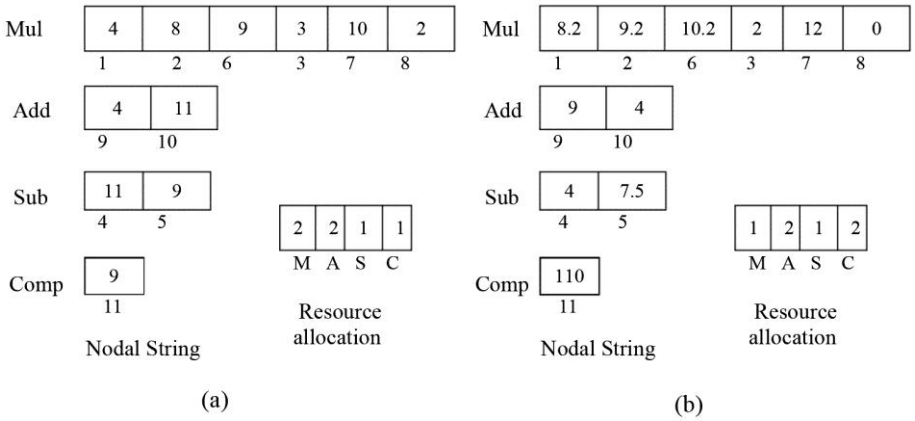

Fig. 4 Chromosome encoding for the third parent (a) and fourth parent (b)

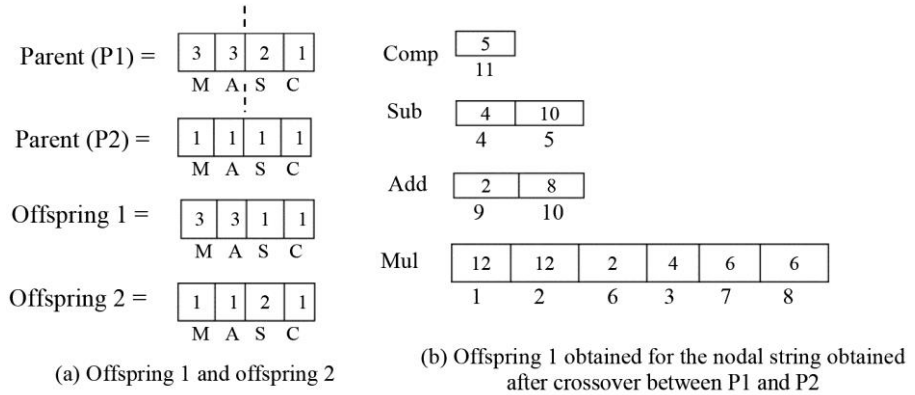

Fig. 5 Crossover between P_1 and P_2

3.3 Crossover technique

Crossover is a technique for producing off-spring when two parents mate. The parents are selected by a binary tournament selection method [28]. In this work, we propose the independent direct crossover of the two independent strings viz. nodal string and resource allocation string to produce separate off-spring for each with a very high crossover probability (Pcross = 1.0). Furthermore, the direct crossover is applied to each sub structure of the nodal string structure. For example, direct crossover is independently applied to adder substring, multiplier substring, subtractor substring, etc. of each nodal string as well as resource allocation string. Since the nodal string encodes the residual load of each operation for a particular FU, the crossover results in crossing only the residual load values. Hence the precedence relationship among the operators is not disobeyed.

3.3.1 Multi-point crossover of the nodal string

Before the crossover scheme can be applied to the nodal strings, the two parents are randomly divided into two halves at point *n*. The crossover point selected during crossing is absolutely random. This is because the nodal string is encoded with residual load values of the nodes and crossover operation only crosses the residual load values, hence choosing a random cut point for crossover does not disturb the precedence relationship among the nodes. Only random cut point has been used in the proposed work as this technique has been widely used by other approaches and provided efficient results. The proposed crossover is called multi-point because each substring of the nodal string representing independent FUs is divided at a different point. For example, applying the direct crossover operator to the nodal string between the first parent (Fig. 3(a)) and second parent (Fig.3(b)) at point 2 for multiplier and point 1 for adder and subtractor, yields offspring 1 and offspring 2 respectively. Offspring 1 inherits all the properties of the first half from the first parent, while the second half of the offspring is inherited from the second parent. The properties that are inherited from the parents are the residual load values and its corresponding node numbers (operations). The offspring 1 obtained after crossover between P1 and P2 is shown in Fig 5(a), while offspring 2 obtained after crossover between P2 and P1 is shown in Fig. 5(b). Similarly the other offspring are obtained by crossing between the rest of the parents. For the sake of brevity, the rest of the offspring obtained have been omitted in this paper.

3.3.2 Crossover of the resource allocation string

The resource allocation string is responsible for encoding the number of hardware functional units of each type available for scheduling operations in each time step. Since the number of allocated functional units of each type is totally independent of each other, the 1-point crossover can be easily applied. For instance, in the case of the DFG for differential equation solver benchmark, the two parents (P1 and P2) for the resource allocation string are shown in Fig. 3(a) and 3(b) respectively. P1 represents a solution with three multipliers, three adders, two subtractors and one comparator while P2 represents a solution with one multiplier, one adder, one subtractor and one comparator.
Application of the direct crossover at a random cut point between P1 and P2 yields offspring 1 while crossing between P2 and P1 yields offspring2 as shown in Fig 5(b).

3.4 Mutation operation

3.4.1 Mutation operator of the nodal string

The mutation algorithm for resource allocation string is adopted from [8] based on random increment or decrement while mutation for nodal string is shown below:

Algorithm

1. Randomly pick any two nodes (vi, vj) from the nodal string [k].
2. Swap the residual load values of the two selected nodes.

If, vi = Li and vj = Lj, then,
vi = Lj and vj = Li.

According to the algorithm, any two nodes (vi, vj) in the string (k) are randomly selected for mutation. Next, the residual load values of the two selected nodes are swapped. For example, let the residual load value for the two nodes (vi) and node (v2) selected be 'L1' and 'L2' respectively. Therefore, after mutation the new residual load values for node (vi) is 'L2' and node (vj) is 'L1'. This mutation technique drastically alters the residual load values, which act as the priority to select the operations for scheduling. As a result of this drastic alteration, the new operation to be scheduled can vastly affect the scheduling cost.

3.5 Decoding process (determination of a valid schedule)

The decoding of chromosomes always results in a valid scheduling solution, which strictly obeys the data dependency present between the operations. For the decoding process, a 'residual load heuristic' is proposed. The residual load heuristic is shown in Fig. 6. For example, in the case of offspring 1, the resource allocation string and the nodal string are shown in Fig.5(a) and Fig.5(b) respectively. The resource allocation string of offspring1 represents an allocation solution containing three multipliers, three adders, one subtractor, and one comparator. On the other hand, the priority of each operation for a particular type of FU is indicated by the residual load values in the nodal string (Fig.5(b)). Therefore, for the dataflow graph shown in Fig.3, the scheduling solution of offspring 1 is shown in Fig. 7. The resulting solution is a valid schedule, allocation and binding obtained for offspring 1. The solution provides an integrated solution to the concurrent problem of scheduling, allocation and binding.

3.6 Global cost function and fitness evaluation methodology

The proposed approach objective is to simultaneously reduce the execution time required for a specific set of data as well as the total hardware area occupied. Most of the previous approaches [2], [4], [7], [8] have only considered latency as a design constraint and not total execution time, which considers the latency, cycle time and also the number of sets of data to be executed. In the presented approach, a comprehensive cost function has been developed that considers the total execution delay, taking data pipelining as well as the total hardware area into account. The decoding process strictly follows the 'residual load heuristic' and hence always results in a feasible solution. The cost function (CG) developed considers total execution time and area is shown in Eq. (3).

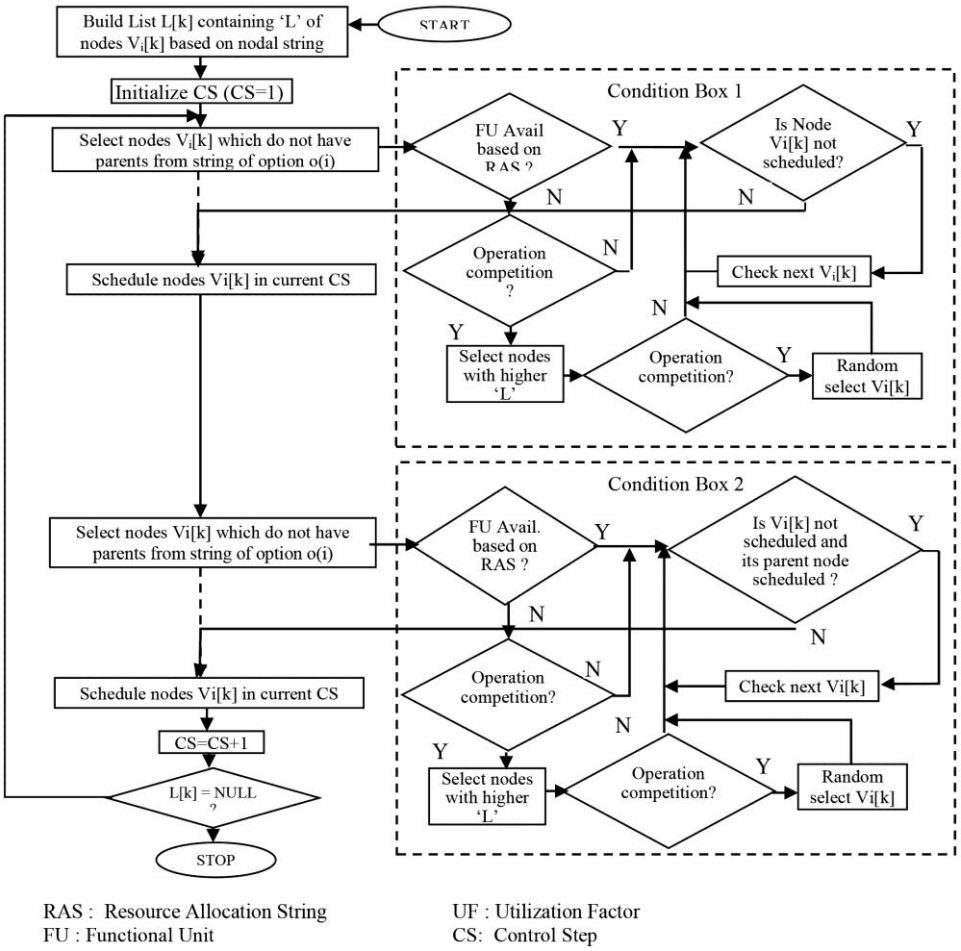

Fig. 6 Flow chart for residual load heuristic

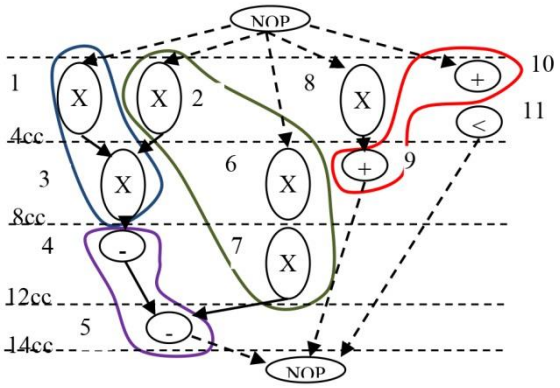

Fig. 7 Chaining schedule and allocation to offspring 1 (Decoded)

$$C_G = W1 \cdot \frac{T_{EXE} - T_{CONS}}{T_{MAX}} + W2 \cdot \frac{[A_{FU} + (A_{MUX} + A_{DEMUX})] - A_{CONS}}{A_{MAX}} \quad (3)$$

T_{EXE} = Total execution time taken for execution of the given sets of data; where T_{EXE} is calculated using the function from [1] given in equation (4):

$$T_{EXE} = \{L + (N-1) \cdot T_C\} \quad (4)$$

L= Latency of the scheduling solution. T_C = Cycle time of the scheduling solution. (Note: The cycle time is the difference in clock cycles between any consecutive outputs of pipelined data instances. The cycle time information is therefore not extracted from the module library since it is not readily available, i.e. the cycle time calculation for the integrated solution (Fig. 7). The output for first set of data is arriving after 14cc while the output for second instance of data is arriving after 26cc. Thus, due to pipelining there is a cycle time difference of 12 cc resulting from considering the initiation interval. Therefore the option of cycle time during pipelining which is the resulting effect of considering initiation interval during data pipelining has been also taken into account during the exploration process.
A_T= Total area calculated using Eq. 5.

$$A_T = A_{FU} + (A_{MUX} + A_{DEMUX}) \quad (5)$$

N = Number of sets of data to be executed.
C_G = Global Cost of the integrated solution
T_{CONS} = Execution time specified by the user.
T_{MAX} = Max execution time taken by a solution during the specific generation (G).
A_{FU} = Total area of the functional units.
A_{MUX} = Total area of the multiplexer used during implementation.
A_{DEMUX} = Total area of the demultiplexers used during implementation.
A_{CONS} = Area constraint specified by the user.
A_{MAX} = Max hardware area of a solution during the specific generation (G).
W_1 and W_2 = User specified preference of the constraints.

The cost function requires input from various sources to evaluate the fitness of each solution found. For the calculation of the execution time, the sources consist of: a) module library information, b) data extracted for the hardware implementation, c) data flow graph and d) scheduling solution found after decoding the chromosome (latency), number of sets of data, cycle time together.

3.7 Termination criterion for the genetic algorithm

The maximum generation has been kept constant for each benchmark run. Although making the number of generations proportional to the problem size is more logical, settling on an average number of maximum generations for both small and large size benchmarks is a good compromise. Therefore, experiments dictated that retaining the maximum generation G(Max) at 100 is an optimal compromise.

4. EXPERIMENTAL RESULTS

Various DSP benchmarks [29], [30] such as digital filter, Auto Regressive Filter (ARF), Discrete Wavelet Transformation (DWT), Digital Butterworth filter, Band Pass Filter (BPF) and Elliptic Wave Filter (EWF), MPEG Motion Vectors, MESA: Matrix Multiplication and JPEG: Down sample were tested and verified. The proposed approach has been implemented in Java and run on Intel core i5-2450M processor, 2.5 GHz with 3MB L3 cache memory and 4GB DDR3 RAM. ExpSys finds optimal/near-optimal results for all the benchmark applications. Moreover, the proposed ExpSys was also compared to [8] with respect to the mentioned benchmarks under the same constraints to make a qualitative assessment and strength of the proposed approach. The proposed achieved better quality of result (determined by Eq.6) as shown in Table I. Furthermore, ExpSysalso considers cycle time resulting from initiation interval and latency to create a genuinely pipelined functional data-path during performance calculation. [8], on the other hand, is not able to optimize the execution time considerably due to its inability to create a genuinely pipelined functional data-path. Thus, for determining of execution time in [8], "*N*" set of processing data is multiplied directly with the latency as per: $\mathbf{T}_{\mathbf{EXE}}^{[8]} = \mathbf{N} * \mathbf{L}.$ Where the QoR is determined as:

$$QoR = \frac{1}{2}\left(\frac{A_T}{A_{\max}} + \frac{T_{EXE}}{T_{\max}} \right) \tag{6}$$

With respect to achieved QoR, ExpSys produces better solutions compared to [8] for all the benchmarks as evident in Table 1. For example, in the case of ARF benchmark, the optimal resource configuration found 3 (*) and 1(+), the area of solution is 10934au, the execution time is 54281μs and the QoR is 0.35. On the other hand [8], based on same constraints, yields an optimal resource configuration which is 4(*), 1(+) with 13776au area, 45630 μs execution time and 0.36 QoR. ExpSys achieves an average improvement in QoR greater than 26% (Table 1).

5. CONCLUSION

This paper proposed a novel technique for Area-Execution time tradeoff using residual load decoding heuristics in genetic algorithm (GA) for integrated design space exploration (DSE). To the best of the authors' knowledge, this approach is the first GAbased DSE method for Area-Execution time tradeoff in HLS. Based on the results obtained from the experiment, the proposed ExpSys is able to provide not only competitive but also superior results for almost all tested DSP benchmarks.

Acknowledgement: *This work is supported by the Optimization and Algorithm Research Lab (OPRAL), Ryerson University, Canadian Microelectronics Corporation (CMC), Motorola, NSERC CRSNG, Ontario Innovation Trust and Sun Microsystems. Additionally, This work acknowledges the assistance provided by Science and Engineering Research Board (SERB), Department of Science and Technology, Govt. of India.*

Table 1 Experimental results of comparison with [8] for the DSP benchmarks

DSP Benchmarks	Parameters of Comparison (Note: us = micro seconds and au = area unit; au = 1Transistor, G(Max)=100 and W1=W2=0.5)								
	Optimal Resource combination			Execution Time N=1000 (us)		Area (au)		QoR	
		ExpSys	[8]	ExpSys	[8]	ExpSys	[8]	ExpSys	[8]
Auto Regressive Filter (ARF)	FU	3(*),1(+)	4(*),1(+)	54281us	45630us	10934au	13776au	0.35	0.36
	Mux	8	10	Constraint 70000us		Constraint 15000au			
	Demux	4	5						
Discrete Wavelet Transformation (DWT)	FU	4(*),1(+)	2(*),1(+)	10844us	66420us	13776au	8092au	0.38	0.56
	Mux	10	6	Constraint 30000us		Constraint 10000au			
	Demux	5	3						
Digital Butterworth Filter	FU	2(*),1(+)	3(*),1(+)	22880us	22410us	8092au	10934au	0.42	0.49
	Mux	6	8	Constraint 30000us		Constraint 9000au			
	Demux	3	2						
Band Pass Filter (BPF)	FU	4(*),1(+)	2(*),1(+)	11642us	68310us	13776au	8092au	0.42	0.52
	Mux	10	6	Constraint 30000us		Constraint 15000au			
	Demux	5	3						
Elliptic Wave Filter (EWF)	FU	3(*),1(+)	2(*),2(+)	21085us	46440us	10934au	10500au	0.45	0.57
	Mux	8	8	Constraint 50000us		Constraint 8000au			
	Demux	4	4						
JPEG Downsample	FU	2(*),1(+)	1(*),1(+)	10818us	29700us	8092au	5250au	0.31	0.59
	Mux	6	4	Constraint 15000us		Constraint 15000au			
	Demux	3	2						
MPEG Motion Vector	FU	4(*),1(+)	5(*),1(+)	32680us	35640us	13776au	16618au	0.24	0.27
	Mux	10	12	Constraint 40000us		Constraint 25000au			
	Demux	5	6						
Discrete Cosine Transformation (DCT)	FU	4(*),1(+)	2(*),2(+)	31467us	88290us	13776au	10500au	0.33	0.47
	Mux	10	8	Constraint 50000us		Constraint 15000au			
	Demux	5	4						
MESA Horner	FU	3(*),1(+)	2(*),1(+)	10843us	65070us	10934au	8092au	0.35	0.59
	Mux	8	6	Constraint 25000us		Constraint 12000au			
	Demux	4	3						
MESA Matrix Multiplication	FU	7(*),1(+)	4(*),2(+)	53628us	132570us	32570au	16184au	0.19	0.24
	Mux	16	12	Constraint 200000us		Constraint 40000au			
	Demux	8	6						

REFERENCES

[1] AnirbanSengupta, Reza Sedaghat, ZhipengZeng, "A High Level Synthesis design flow with a novel approach for Efficient Design Space Exploration in case of multi parametric optimization objective", Microelectronics Reliability, Elsevier, Volume 50, Issue 3, March 2010, Pages 424-437.

[2] C. Mandal, P. P. Chakrabarti, and S. Ghose, "GABIND: A GA approach to allocation and binding for the high-level synthesis of data paths," IEEE Transaction on VLSI, vol. 8, no. 5, pp.747–750, Oct. 2000.

[3] M. J. M. Heijlingers, L. J. M. Cluitmans, and J. A. G. Jess, "High-level synthesis scheduling and allocation using genetic algorithms," in Proc. ASP-DAC., pp. 61–66, 1995.
[4] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker, "Datapath synthesis using a problem-space genetic algorithm," in IEEE Trans.Comput.-Aided Des., vol. 14, pp. 934–944,1995.
[5] I. Das. A preference ordering among various Pareto optimal alternatives. Structural and Multidisciplinary Optimization, 18(1):30–35, Aug. 1999.
[6] Alessandro G. Di Nuovo, Maurizio Palesi, Davide Patti, Fuzzy Decision Making in Embedded System Design," Proc. of 4th Intl Conference on Hardware/Software Codesign and System synthesis, pp: 223-228, October 2006.
[7] J. C. Gallagher, S. Vigraham, and G. Kramer,"A family of compact genetic algorithms for intrinsic evolvable hardware," IEEE Trans. Evolutionary Computation., vol. 8, no. 2 , pp. 1–126, Apr. 2004.
[8] Vyas Krishnan and SrinivasKatkoori, "A Genetic Algorithm for the Design Space Exploration of DatapathsDuring High-Level Synthesis, IEEE Tran.on Evolutionary Computation, vol.10, no.3, 2006.
[9] E. Torbey and J. Knight, "High-level synthesis of digital circuits using genetic algorithms," in Proc. Int. Conf. Evol. Comput., pp.224–229, May 1998.
[10] E. Torbey and J. Knight, "Performing scheduling and storage optimization simultaneously using genetic algorithms," in Proc. IEEE Midwest Symp. Circuits Systems, pp. 284–287, 1998.
[11] Giuseppe Ascia, Vincenzo Catania, Alessandro G. Di Nuovo, Maurizio Palesi, Davide Patti, "Efficient design space exploration for application specific systems-on-a-chip" Jrnl of Systems Architecture 53, pp:733–750, 2007.
[12] A.C.Williams, A.D.Brown and M.Zwolinski,"Simultaneous optimisation of dynamic power, area and delay in behavioural synthesis", IEE Proc.-Comput. Digit. Tech, Vol. 147, No. 6, pp: 383-390, 2000.
[13] Christian Haubelt, Thomas Schlichter, Joachim Keinert, Mike Meredith, "SystemCoDesigner: automatic design space exploration and rapid prototyping from behavioral models", Proceedings of the 45th annual ACM IEEE Design Automation Conference, Pages 580-585, 2008.
[14] Xuejie Zhang and Kam W. Ng, "A review of high-level synthesis for dynamically reconfigurable FPGAs", Microprocessors and Microsystems, Elsevier, Volume 24, Issue 4, Pages 199-211,1 2000.
[15] N. Wehn et al., "A novel scheduling and allocation approach to datapath synthesis based on genetic paradigms," in Proc. IFIPWorking Conf. Logic Architecture Synthesis, pp. 47–56, 1991.
[16] R. M. San and J. P. Knoght, "Genetic algorithms for optimization of integrated circuit synthesis," in Proc. 5th Int. Conf. Genetic Algorithms, San Mateo, CA, pp. 432–438, 1993.
[17] R. J. Cloutier and D. E. Thomas, "The combination of scheduling, allocation and mapping in a single algorithm," in Proc. 27th Design Automation Conf., pp. 71–76, Jun. 1990.
[18] J. A. Nestor and G. Krishnamoorthy, "SALSA: A new approach to scheduling with timing constraints," IEEE Trans. Comput.-Aided Des., vol. 12, pp. 1107–1122, 1993.
[19] G. Krishnamoorthy and J. A. Nestor, "Data path allocation using extended binding model," in Proc. 32nd ACM/IEEE Design Automation Conf., pp. 279–284, 1992.
[20] S. Devadas and A. R. Newton, "Algorithms for hardware allocation in data path synthesis," IEEE Trans. Comput.-Aided Des., vol. 8, pp.768–781, 1989.
[21] T. A. Ly and J. T. Mowchenko, "Applying simulated evolution to high level synthesis," IEEE Trans. Comput.-Aided Des., vol. 12, no. 2, pp.389–409, Feb. 1993.
[22] C. H. Gebotys and M. I. Elmasry, "Global optimization approach for architectural synthesis," IEEE Trans. Comput.-Aided Des., vol. 12, pp. 1266–1278, 1993.
[23] C. T. Hwang, J. H. Lee, Y. C. Hsu, and Y. L. Lin, "A formal approach to the scheduling problem in high-level synthesis," IEEE Trans. Comput.- Aided Des., vol. 10, no. 2, pp. 464–475, Feb. 1991.
[24] G. De Micheli, Synthesis and Optimization of Digital Circuits. New York: McGraw-Hill, 1994.
[25] R. Camposano, "Path-based scheduling for synthesis," IEEE Trans.CAD., vol. 10, pp. 85–93, 1991.
[26] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," IEEE Trans. Comput.-Aided Des., vol. 8, no.6, pp. 661–679, 1989.
[27] A. C. Parker, J. T. Pizarro, and M. Mlinar, "Maha: A program for datapath synthesis," in Proc. 23rd ACM/IEEE Design Automation Conf., 1986, pp. 461–466.
[28] T. Blickle and L. Thiele, "A mathematical analysis of tournament selection," in Proc. 6th Int. Conf. Genetic Algorithms, pp. 9–16, 1995.
[29] http://www.cbl.ncsu.edu/benchmarks/.
[30] Saraju P. Mohanty, NagarajanRanganathan, Elias Kougianos and PriyadarsanPatra, "Low-Power High-Level Synthesis for Nanoscale CMOS Circuits" Chapter- High-Level Synthesis Fundamentals, Springer US, 2008.