High-Level Synthesis of Digital Circuits using Genetic Algorithms

Elie Torbey and John Knight
Carleton University, Ottawa, Ontario, Canada

Abstract

This paper describes a High-Level Synthesis system that uses Genetic Algorithms. The use of GAs allows for a synthesis method that is more flexible and more adaptable to new constraints than the traditional heuristic and Integer Linear Programming approaches. The proposed GA tool is suitable for large, realistic problems. It performs simultaneous scheduling, allocation and binding of functional and storage units minimizing multiple related performance constraints such as latency, area, throughput and power. The synthesis tool is capable of performing control/data flow graphs.

1. Introduction

High-level synthesis (HLS) or *architectural synthesis* is the automated design process that converts an algorithm for a digital design, into a dedicated digital circuit. A typical algorithm might be a fast Fourier transform, or a controller to move a robot arm where control of position and velocity are important. The output of such synthesis consists of: (i) functional units (FUs) which implement arithmetic and logic functions, (ii) registers and memory units which store intermediate data, (iii) switchable connections (multiplexers and buses) which interconnect the functional units with storage, and (iv) a controller which mainly controls the switching. Once this design description is obtained, it can be synthesized (logic synthesis), laid out and fabricated to obtain the integrated circuit.

High-level synthesis attempts to optimize some cost function typically of circuit-speed, power-consumption, and production-cost (related to area for integrated circuits). Further there are often fixed constraints. For example, some calculation must take less than 30 ms, or a silicon circuit must fit into a 120 pin standard package.

Current synthesis systems typically use heuristics [1] [6] [9] or integer linear-programming (ILP) [3]. Heuristics are very fast, but lack flexibility. Older heuristics minimized silicon area. These heuristics are very difficult to adapt to new constraints like the recent need to minimize power consumption. They also tended to optimize the four components (FUs, storage, connections and control) independently, even though they interact strongly. The constraining equations for ILP are more easily adapted for the rapidly changing needs of technology. However ILP is basically exponential and one must limit the problem size or partition it in some way.

Genetic algorithms present a solid global optimization methodology that is easily adaptable to the synthesis problem. Some work applying genetic algorithms to scheduling and synthesis has been performed with promising results [4] [8]. These implementations did not take into account conditional execution and register minimization. Different simulated evolution approaches have also been tried [5], but these rely on application-specific heuristics and are not flexible. Problem space GAs have also been used [2] but, again, the use of heuristics in mapping from the problem space to the solutions make the approach less flexible than GAs.

This paper presents a GA-based approach to High-level synthesis. It shows the results achieved using the GA synthesis tool. This method has the advantage of flexibility. It can accommodate real constraints in terms of area and delay and can handle large realistic problems.

2. High-level synthesis

2.1 HLS Flow

A typical HLS flow is shown in Figure 1. The design is specified at an algorithmic level, The tool will then generate a circuit at the Register-Transfer-Level (RTL) using the blocks in the library units. If this circuit is in the form of a hardware description language, the design can be simulated and verified. If the resulting circuit does not meet specification a re-design is required. The HLS portion of the flow includes the three main steps in HLS that are described in Section 3.1.

2.2 HLS Formulation

Let v and w be arithmetic or logical operations and (v, w) be a precedence relation. This means that the output of v must be completed before w can start. Then the problem is

0-7803-4869-9/98 $10.00©1998 IEEE

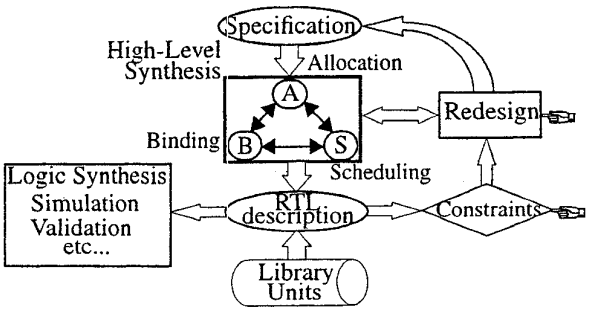

Figure 1. HLS Flow

choosing hardware h_v, h_w to execute v, w and scheduling each into a clock step s so that:

$$(t_w > t_v + l_{v_h} + M) \quad \forall (w, v) \in P$$

while minimizing a cost function $C(a,p,d)$

Here t_x is the initiation time of operation x and l_{vh} is the latency (propagation delay) of operation v, executed on hardware unit h. P is the set of data precedence and M is a timing margin that accounts for interconnect, mux and register delays. In the cost function, a refers to the total area, p to the power consumption and d to the total latency of the algorithm.

2.3 HLS complexity

Considering the number of variables and the complexity of the circuitry in modern designs, a large variety of implementations is possible resulting in a large design space necessitating efficient search and optimization techniques. GA synthesis is an effective way to search the design space for these problems.

Table 3 shows two synthesis benchmarks and their solution spaces. EWF is an elliptic wave filter for DSP applications and ROTOR [7] is a coordinate translator for robotic applications. *Op* refers to the number of operations, *Prec* is the number of precedence relations.

Table 1. Solution space for benchmarks

Design	Op.	Prec.	Paths	Space
EWF	34	47	57	10^{70}-10^{90}
ROTOR	28	37	28	10^{60}-10^{80}

2.4 Library units

In HLS, the library units are hardware units that can perform operations like multiplications and additions. Traditional synthesis tools use one type of hardware units for each type of operation mainly for simplicity of synthesis and execution speed. The synthesis tool presented in this paper allows a variety of different units to be used with minimal additional complexity. Also, traditional approaches modeled these units using relative areas and clock cycle latencies. This tool allows a more realistic description of these units. They are described in terms of their sizes (in equivalent gates) and their latencies in *ns*.

2.5 Design specification

The design specification is usually provided using a high-level description language such as VHDL [9]. This description can then be modeled in a common intermediate form using Control/Data Flow Graphs (CDFGs). A CDFG describes the algorithm in a graph form where the nodes represent the operations of the algorithm and the arcs represent the data precedences. Figure 2 shows an example algorithm. Solid lines show data precedences and dotted lines show control precedences.

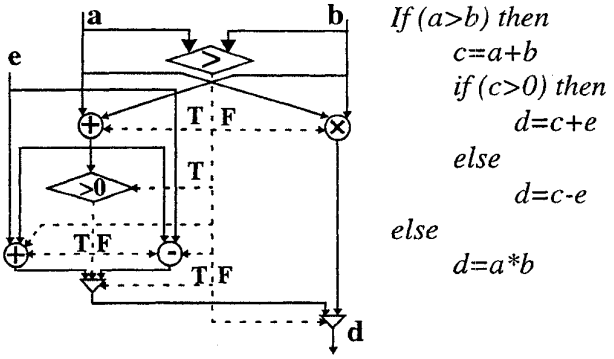

Figure 2. Example CDFG.

2.6 Architectural description

The output of HLS systems is typically described in terms of a data-path that implements the desired functionality and a controller that controls the behavior of the data-path. Figure 3 shows an example architecture corresponding to the algorithm given in Figure 2.

The data-path is composed of:

Hardware units, or functional units (FUs), that implement logic or arithmetic functions. They could include adders, multipliers, ALUs or Multiply-accumulate units.

Storage units (SUs) such as registers, register files and RAMs used for intermediate storage.

Interconnections (buses) and multiplexers, or topology units (TUs). These are shared buses between multiple units and/or dedicated connections between units. They include tri-state drivers and multiplexers.

Input and Output logic or interface units (IUs). These are usually handled like FUs. They may, however, have fixed initiation times corresponding to external interfaces.

The controller is usually implemented as a micro-code ROM or a Finite State Machine (FSM).

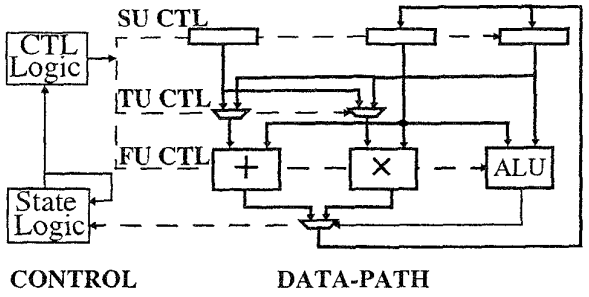

Figure 3. Example synthesized architecture

3. High-level synthesis methods

3.1 HLS steps

High-level synthesis is composed of three related problems: allocation, scheduling and binding [1].

Allocation is the process of assigning enough appropriate, hardware units to implement all operations.

Scheduling is the process of assigning time steps to each operation in the given behavioral description while ensuring no data precedences are violated.

Binding is the process of assigning operations to hardware units at specific times. This is the stage where all the interconnections in the architecture are defined.

The GA synthesis presented performs all these tasks simultaneously. Systems that perform these functions in sequence may have to iterate between the steps.

3.2 HLS constraints

The main optimization constraints used in high-level synthesis are area and delay. Other constraints can include power (peak and average), throughput (and/or latency) as well as clock speed, testability, interconnect (usually mux and bus area), routability. Some of these constraints vary with the choice of technology, technology scaling and design improvements. A synthesis tool should be flexible enough to accommodate such changes.

3.3 HLS scheduling methods

Following are different scheduling methods in HLS.

ASAP/ALAP scheduling. ASAP refers to a schedule that assigns operations to be executed as soon as all the data necessary for their execution is available. ALAP refers to assigning operations to be executed right before the values they produce are needed. Many methods use these schedules as a starting point.

Heuristic scheduling, such as list scheduling [9], which orders operations based on priorities and force-directed scheduling [6], which assigns operations based on distributed graphs derived from the operation mobilities. Heuristic schedulers are usually fast but not very flexible.

Neural Networks scheduling have been implemented using self-organizing algorithms. The disadvantage of this approach is its speed.

Simulated annealing scheduling generates random modifications in the schedule and accepts or rejects them according to a random rule and a parameter that gradually decreases with time. The disadvantage is the speed.

Integer Linear Programming scheduling (ILP) is an exact scheduling technique. ILP scheduling is formulated as an optimization problem using certain constraints [3]. This method suffers from increasing time-complexity since it is exponential in the worst case. This puts larger, more realistic problems out of its scope.

3.4 Scheduling and synthesis issues

Pipelining in high-level synthesis refers to the use of pipelined units. A synthesis tool can have the ability to understand pipelined modules in its library and assign operations to them. Figure 4 shows an example of a schedule that includes a pipelined unit and an implementation of a pipelined adder.

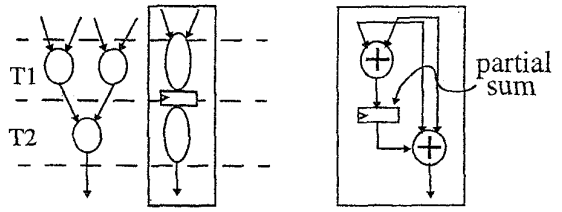

Figure 4. Pipelining example

Chaining is the process of performing more than one operation in one clock cycle. Figure 5 (a) shows an example of operation chaining where operation 1 and 2 are executed in the same cycle as operation 3 which is dependent on their execution.

Multi-cycling of an operation implies that it can be scheduled over multiple time steps when the latency of the FU used exceeds the clock period of the system. Figure 5 (b) shows an example where operation 4 takes 2 cycles.

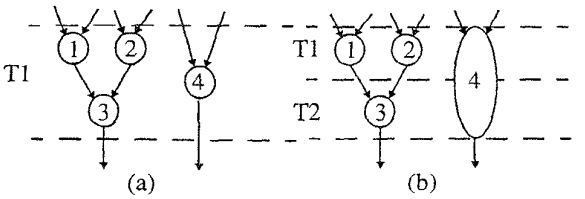

Figure 5. Chaining and multi-cycling

4. GA synthesis formulation

The GA approach used is chosen for its simplicity and its independence from the problem specifics. The heuristics employed in this system are built around the GA optimization for implementing CDFGs and binding.

4.1 GA setup

The GA parameters used are as follows:

The population size *z*: A chromosome represents a complete circuit. A typical population size for synthesis problems range from a hundred to several thousand.

The maximum number of generations *g*: Typical problems of this magnitude are run up to tens of thousands of generations. Often, however, an industrially good solution is found within a hundred.

The crossover probability which typically ranges from 20 to 60%. Solutions shown in this paper use a crossover probability of 50% unless otherwise noted.

The probability of mutation typically ranges from 0.1 to 1%. For examples shown in this paper, it was set to 0.5%.

The initial circuits were randomly generated. A single-point crossover was used for most examples shown in this paper. The selection method was the Roulette Wheel using steady state and generational replacement. The stopping criteria was the number of generations.

4.2 Chromosome encoding

The chromosome is composed of two fields per operation, (*hw, cc*) where *hw* is a field designating the *id* of the FU used and *cc* is the relative clock cycle for the initiation of the operation which is based on the *mobility*.

The mobility is the time interval between the earliest time slot in which the operation can be executed (ASAP) to the latest time slot in which an operation can be executed (ALAP). ALAP does not allow times longer than the critical path. Minimal hardware solutions can often be found for small additional increases in time. For this reason, we define and use an ARAP schedule (As Resource-constrained As Possible) as a schedule on the minimum possible hardware. Obtaining such a schedule has the same complexity as scheduling in general. However, for the purpose of encoding, this ARAP schedule can be relaxed and can use upper bounds rather than optimal assignments allowing all good solutions in the design space to be represented. The number of bits needed to represent a CDFG is:

$$n = \sum_{v=1}^{V} \log_2 K_v + \log_2 mob_v$$

where K is the number of available units that can perform operation v. mob is the mobility and V is the total number of operations.

The ordering of the operations in the chromosome is chosen such that all operations are preceded by their precedents in the graph. This reduces the complexity the objective function calculation.

4.3 Relative clock cycle encoding

An important aspect of the encoding is the clock cycle (clock step) encoding. Absolute encoding where the number in the chromosome corresponds to a clock step invariably leads to many infeasible solutions as there is no precedence constraint in the encoding. Special crossover operators have been designed to combat this problem. Some researchers have opted for problem-space GAs to avoid the problem [2].

Another solution involves a relative encoding where the *cc* value is relative to its latest scheduled precedence. In the presented GA formulation, the operations are ordered ensuring precedences and therefore guaranteeing feasibility. The advantages of this encoding method are:

Minimize infeasible solutions; If absolute timing intervals are used, the GA will produce many chromosomes which violate data precedences.

Reduce the number of bits per chromosome; Using a relative encoding minimizes the number of bits used to encode the clock cycle symbol. In large algorithms, some operations may have to be scheduled at absolute cycles in the hundreds, this would require a large number of bits. This number impacts the convergence time of the algorithm. The larger the number of bits in a symbol, the longer the GA will need to run to obtain good solutions.

The GA synthesis tool presented uses a circular relative encoding where the mobility of operations gets reduced after their predecessors have been scheduled. This ensures operations are not scheduled beyond their upper bounds.

4.4 Objective function calculation

As an example of calculations performed to determine the fitness of a particular architecture, consider the total functional unit area:

$$A_{FU} = \sum_{i=1}^{CC} \sum_{j=1}^{N} (n_{ij} - m_{ij}) \cdot a_j s_{ij}$$

$$\text{where } \begin{cases} s_{ij} = 0 \text{ if } \exists i = l, j = m, l \in [0, i-1], m \in [0, j-1] \\ s_{ij} = 1 \text{ otherwise} \end{cases}$$

N denotes the number of different FUs. *CC is* the largest scheduling clock cycle. n is the number of identical FUs

used in a cycle. m is the number of FUs in a cycle with mutually exclusive data. a is the area of the FU in gates. s is a time-sharing variable between clock cycles.

The SU, TU and IU areas are calculated similarly. The objective function also includes the latency and power of the circuit. Many other constraints can be incorporated.

The fitness is a function of these costs such as:

$$Fitness = \frac{1}{Cost} = \frac{C}{\alpha A_{total} + \beta P_{avg} + \gamma d_{total}}$$

C is a normalizing constant. α,β, and γ are weights. A_{total} is the area cost, P_{avg} is the power cost and d_{total} is the delay, all of which have to be minimized.

Table 2. EWF and ROTOR Synthesis Results

EWF				ROTOR			
CC	+	*	OF	CC	A	T	CPU
17	3	3	L	13	1	1	42.1
18	2	2	L	7	2	1	17.3
21	2	1	L	7	3	1	11.7
28	1	1	L/A	6	4	1	2.1

5. Results and benchmarks

5.1 Benchmark results

If the available hardware for the implementation of the Elliptic Wave Filter consisted of a 2-cycle multiplier and a single-cycle adder, then the results of the synthesis would be as shown in Table 2. *CC* refers to the number of clock cycles required to complete the algorithm. *+/*/A/T* refer to the number of multipliers, adders, ALUs and table-lookups required for each case. *OF* denotes the objective function. In the above examples, this is a linear combination of the latency and area. The objective function used to obtain the first three results for the EWF was strictly based on latency. All the shown results are known to be optimal [3] [7].

Table 3. Practical FUs

Adder			Multiplier	
# bits	Area (g)	Speed (ns)	Area (g)	Speed (ns)
16	151	19.15	1376	33.05
16	189	9.96	1534	26.44
32	312	39.68	6053	69.50
32	461	9.98	6502	59.15

For the Rotor example, if a single cycle Table lookup is used to calculate the trigonometric functions and a single-cycle ALU is used for all other operations, then the results in Table 2 are obtained. The CPU times are similar to those reported in [7], using symbolic methods for synthesis.

Table 4. Multiple FUs and real areas/delays

Example	Delay (ns)	Period (ns)	FU Area (gates)	OF
EWF	210	10	1874	ad^2
	220	10	1874	ad^2
	220	10	1874	ad
ROTOR	80	10	1098	ad^2
	90	10	1269	ad
	110	10	720	ad^2
	130	10	682	ad

5.2 Multiple hardware and real areas and delays

Section 5.1 showed scheduling examples where only one type of FU per operation was available. One of the main advantages of the GA approach is the ability to use multiple FUs with real area and latency constraints with minimal additional complexity. Several adders and multipliers were synthesized in a typical sub-micron technology and included in a library of functional units. A sample is shown in Table 3. The area is in equivalent NAND gates and the speed is in *ns*.

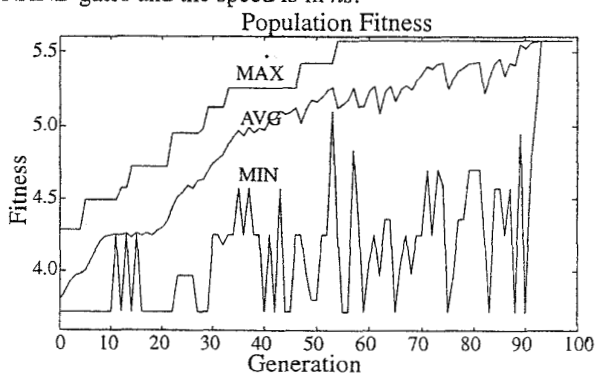

Figure 6. EWF Fitness (steady state).

Using the FUs of Table 3, a 16 bit EWF circuit can be optimized as shown in Table 4 based on *ad* and ad^2. *ad* refers to an objective function proportional to the product of the area and the delay of the resulting architecture. ad^2 puts more weight on the delay. Table 4 shows a sample of good solutions for the optimization criteria given. The fastest solution was not obtained with the *ad* criterion and the GA setup used, but was obtained using ad^2 as an objective function. For the Rotor example, with multiple adders, multipliers and separate compare units, the results shown in Table 4 were obtained. Solutions picked include the best-fit for each criterion as well as some intermediate results illustrating area-delay trade-offs. The best solutions

correspond to the lowest period. Adding the power as a constraint would benefit solutions with longer periods.

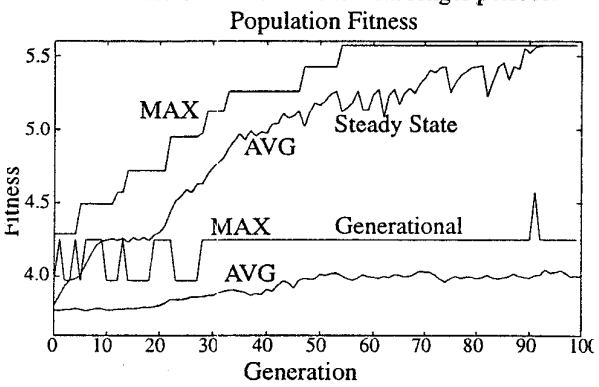

Figure 7. Steady-state vs. generational

5.3 GA parameters

This section shows the results of varying some of the GA parameters. Figure 6 shows the EWF example with a crossover rate of 0.5 and a mutation rate of 0.005. This is a steady-state run where all solutions below the average fitness are replaced.

Figure 7 shows the difference between a generational and a steady-state run for the EWF. The steady-state run outperforms the generational run for the number of generations shown. This is typical for these problems.

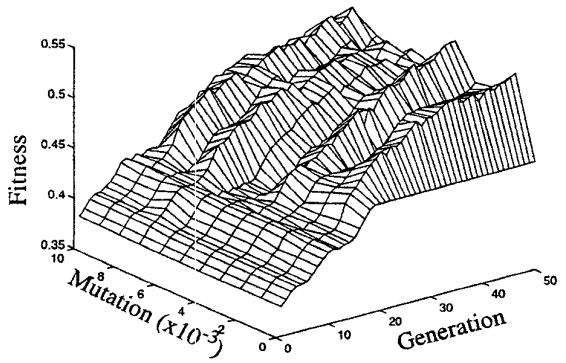

Figure 8. Mutation rate effect

Figure 8 shows the effect of the mutation rate on the average fitness for the EWF problem for an arbitrary 50% crossover rate. The mutation rate is varied from 0 to 1%. The trend of improving average fitness is clear. It can also be seen that for the initial 20 generations, the mutation rate does not make much difference. The mutation rate has more impact where the rate of improvement is highest.

Figure 9 shows the effect of the crossover rate on the mutation for an arbitrary 0.5% mutation rate. The crossover rate is varied from 10 to 100%. The same trends that are seen when varying the mobility can be seen here as well. The difference seems to be a more random distribution of average fitness in the latter generations.

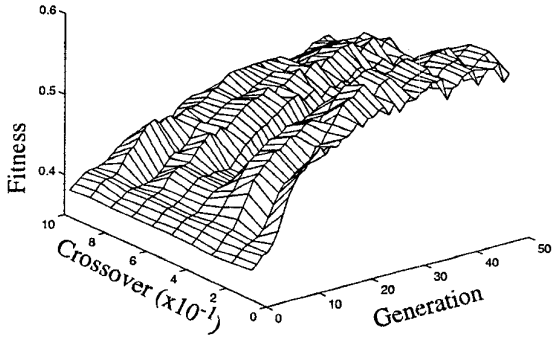

Figure 9. Crossover rate effect

6. Conclusion

This paper presented a Genetic Algorithm approach to the synthesis of digital circuits specified as high-level algorithms. This method is flexible, allowing more realistic constraints to be used than other heuristic approaches. It is also easily adaptable to new constraints. It further supports many architectural optimization techniques including pipelining, chaining and multi-cycling. The experiments show best known results for the two benchmarks used as well as some results indicating the use of real area and latency constraints.

7. References

[1]Giovanni De Michelli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.

[2]M.K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker, "Datapath Synthesis Using a Problem-Space Genetic Algorithm," IEEE Transactions on CAD, Vol. 14, No. 8, August 1995, pp. 934-944.

[3]C.H. Gebotys, "Optimal Scheduling and Allocation of Embedded VLSI Chips," Proc. of the 29th Design Automation Conference, pp. 116-119, 1992.

[4]Marcus Heijligers, *The Applications of Genetic Algorithms to High-Level Synthesis.* Technishe Universiteit Eindhoven, Eindhoven 1996.

[5]T. A. Ly and J. T. Mowchenko, "Applying Simulated Evolution to Scheduling in High Level Synthesis," IEEE Transactions on CAD, Vol. 12, No. 3, March 1993, pp. 389-409.

[6]P. G. Paulin and J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs," IEEE Transactions on CAD, pp. 671-679, June 1989.

[7]I. Radivojevic and F. Brewer, "A New Symbolic Technique for Control-Dependent Scheduling," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 45-57, vol. 15, No. 1, January 1996.

[8]R. San Martin and J. P. Knight, "Genetic Algorithms for Optimization of Integrated Circuit Synthesis," in Proceedings of the Fifth International Conference on Genetic Algorithms and their Applications, 432-438, San Mateo, Ca, 1993.

[9]Robert A. Walker and Raul Camposano, eds., *A survey of High-Level Synthesis Systems.* Kluwer, Boston, 1991.