Estimation of HW/SW Cost Parameters in Altera FPGA Design Environment

[1]M.JAGADEESWARI, [2]M.C.BHUVANESWARI
[1]Department of Electronics and Communication Engineering
Sri Ramakrishna Engineering College,
Coimbatore-641022,
INDIA
jagadee_raj@rediffmail.com, http://www.srec.ac.in
[2]Department of Electrical and Electronics Engineering
P.S.G College of Technology,
Coimbatore 641 004,
INDIA

Abstract: - This paper explains the method for obtaining Hardware/Software (HW/SW) cost parameters such as hardware area, hardware time, software area and software time using Altera FPGA design environment. HW/SW partitioning of FFT and JPEG FDCT are derived using multi-objective optimization techniques Weighted Sum Genetic Algorithm (WSGA), Elitist Non-dominated sorting Genetic Algorithm (ENGA) and Multi-Objective Particle Swarm Optimization using Crowding Distance (MOPSO-CD) algorithms. Experimental results show that ENGA is effective in obtaining the HW/SW partition that obtains both minimum area and minimum time for both the applications.

Key-Words: - Evolutionary algorithms, HW/SW partitioning, Multi-objective optimization, Pareto-optimal solutions.

1 Introduction

This paper deals with the HW/SW partitioning of FFT and JPEG-FDCT applications in such a way as to minimize both the area and execution time of the partition. Embedded system involves the design of functions that may be implemented in hardware using hardware blocks or Verilog code or as software by writing program to run on an embedded processor. The decision to partition sections into HW/SW is dependent on the determination of the cost values for the functionality of the embedded systems. The hardware implementation costs consist of hardware resources (also called data-path resources), control logic, registers, and communication structures like bus and multiplexer circuits. The software costs for each task may be estimated from memory latency, channel (bus) speed, total amount of data on the edge and the number of tokens for each firing.

Altera Quartus II FPGA design environment is used for the estimation of cost parameters such as the hardware area, the software area, the hardware time and the software time. Two data dominated applications related to DSP and image processing are partitioned using the multi-objective optimization algorithms WSGA, ENGA and MOPSO-CD.

2 HW/SW Design Flow

Fig.1 shows the HW/SW design flow of the embedded system consisting of implementation of the system functions in the hardware and the software, and the merging of the results to evaluate the performance of the total system. The first step in the design is to decide what parts of the complete system are done in hardware using the hardware packages, HDL code and what parts are done by writing a program to run on a given processor. This is a manual or semi-manual process and is the most difficult system design phase.

The hardware part of Fig.1 as in Zainalabedin Navabi [1] becomes a description of various hardware modules that are described in an HDL or are available as predefined hardware modules. Using tools and design environments, a hardware designer can choose to code parts of the design in Verilog/VHDL or use parts from a library of pre-defined modules. The hardware design environments include configurable parts for commonly used components such as arithmetic

functions, register banks and counters. The software part describes the memory contents of the processor that runs the program. The designer may choose to code the part in a high-level language and compile it or directly code it in assembly or machine language. C/C++ is generally used to describe the part of the design that is to be implemented in software part. The middle of the diagram shows a block that specifies interconnection of the hardware and the software parts. The interconnection may be done using simple shared bus, interconnection wires, or a complex switch structure. Usually, embedded system design environments have their own bus structures. Handshaking, timers, block transfer hardware and other high-level transactions take place in this bus.

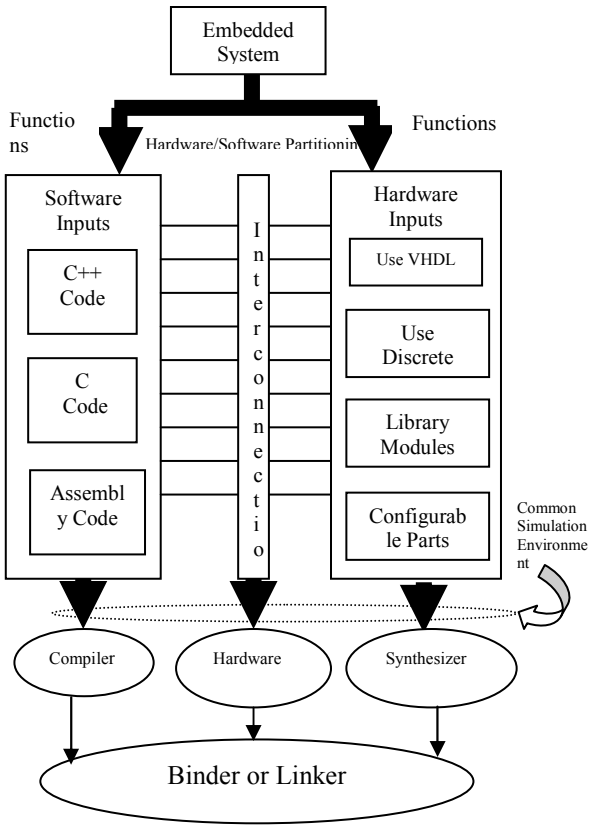

Fig. 1. Hardware/Software System Design Flow

The target architecture assumed in this work is a single hardware module and a single software processor. The Altera Quartus II is used for implementation of the tasks in hardware module and the NIOS II processor is used for implementation of the tasks in software processor. The Altera Quartus II design environment provides a complete, multi-platform design environment that easily adapts to the design needs.

2.1 ALTERA QUARTUS II Design Environment

The Altera Quartus II is a comprehensive environment for System-On-a-Programmable-Chip (SOPC) design. The Quartus II software includes solution for all phases of FPGA and CPLD design. The Quartus II includes the modules such as analysis and synthesis, partition merge, fitter, assembler, timing analyzer, so on and so forth. Quartus II synthesis generates various report and log files showing the details of compiler settings, device settings, elapsed time and the resources used. The hardware information generated by the compiler includes diagram showing FPGA utilized areas, interconnections, and the logic diagrams. An important part of the information provided by the synthesis tool is the timing information. This includes the setup and the hold time of registers, the maximum clock frequency and the worst-case delays. This is obtained in the timing analyzer section of the compilation report (Quartus II handbook version 9.1).

2.2 NIOS II Integrated Development Environment

The Altera Quartus II has an inbuilt soft core called NIOS II processor, which is equivalent to a 32-bit micro controller. The NIOS II processor is a general-purpose RISC processor and a configurable soft-core processor. In this context ‘configurable’ means that features can be added or removed on a system-by-system basis to meet the performance or the price goals. Soft-core comes in ‘soft’ design form (i.e., not in silicon) and can be targeted to any Altera FPGA family. The term “NIOS II processor system” refers to NIOS II processor core, a set of on-chip peripherals, on-chip memory, and interfaces to off-chip memory, all implemented on a single Altera chip. Altera uses Integrated Development Environment (IDE) for the development and testing of the software programs (C/C++ programs) that run on the processor of the embedded system. NIOS II IDE provides several project management tasks that speed up the development of embedded applications. It includes C compiler, necessary program entry and test utilities. This phase compiles the C program and generates memory contents for the program memory of NIOS II. The SOPC builder Graphical User Interface (GUI) enables hardware designers to configure NIOS II processor systems with any number of peripherals and memory interfaces. This generates software files

for the embedded software development such as a memory-map header file and component drivers [2].

3 Determination of Cost Parameters

Most of the tasks in the benchmark DAG can be realized using adders and registers or based on add and shift operations. For example, multiplication and division are estimated to consume two full adders, whereas subtraction consumes one [3]. In this research, the hardware area, software area, hardware time and software time of 8 bit full adder are determined using Altera Quartus II FPGA design environment. The communication delay between the hardware and the software tasks is generated at random between one to twenty micro seconds [4], [5].
The hardware implementation of a full adder is done in Quartus II and programmed into Cyclone II EP2C5T144C6. The size of a hardware implementation is expressed in Look-Up Tables (LUTs), using a specific library of gates required for implementation of the hardware. The software implementation is done in NIOS II and programmed in C. The area of the software implementation is expressed as the memory utilized by the tasks when implemented in software. Table 1 lists the relevant information for full adder that is determined from the synthesis results in Quartus II design environment.

3.1 Hardware Task Area

Hardware area is determined by synthesizing the VHDL/Verilog code of the 8-bit full adder into Quartus II EP2C5T144C6 device. From the synthesis results, the number of LUTs occupied for one 8-bit full adder is found to be 10. The area consumed by any task of the DAG in an embedded system is dependent on the accumulated number of adders. Using the information from Table 6.1, an estimate of the area required for multiplier task is obtained as (2 x 10) = 20 LUTs and that of subtractor task is same as that of the adder task. In Altera FPGA, Cyclone III uses each LUT as a 512 x 1-bit RAM [6].

3.2 Hardware Execution Time

Hardware execution times of a task are estimated by running Quartus II timing analyzer. The total propagation delay of the gate/task encountered is taken as the hardware execution time. It also refers to the time taken for the output to reach from the source input pin to destination output pin. From the simulation results of the timing analyzer, the hardware execution time of a full adder is found as 18.95 nano seconds. The hardware execution time of the multiplier is calculated as twice the execution time of a single full adder and the hardware execution time of the subtractor is the same as the execution time of a single full adder.

Table 1. Estimated Values for 8-bit Full Adder

S. no	Parameter	Value
QUARTUS II OUTPUT		
1.	Logic element usage by number of LUT inputs	10/4608
2.	Hardware Execution Time (ns)	18.95
3.	Family	Cyclone III
4.	Device	EP2C5T144C6
NIOS II IDE OUTPUT		
5.	Program size (code + initialized data)	44KB
6.	Bytes free for stack + heap	4092 bytes
7.	Software Execution Time (μs)	2.09

3.3 Software Memory Analysis and Execution Time Evaluation

The C program for the full adder is compiled using NIOS II core. After the completion of the project build, a report of the software is created in the environment's console. This report indicates the memory usage of the program (program size) and the free bytes available for the stack and heap. It also indicates whether the allocated memory was sufficient for the given software program. The program memory specifies the software memory required for full adder which provides the software area for the system. The software area is found to be 44 KB. The software time is the time required to compile the C program of full adder in NIOS II environment which is found to be 2.09 micro seconds.

4 WSGA for Hardware/Software Partitioning

WSGA is similar to conventional GA. It uses a classical approach of weighted sum of objectives and is used to solve multi-objective optimization of hardware/software partitioning problem. The steps of optimization using WSGA is given below

Step 1: Generate an initial population randomly containing 'N 'individuals in the population.

Step 2: Calculate the values of the cost and execution time for the generated individuals.

Step 3: Calculate the fitness F of each individual

$$F(x) = \sum_{i=1}^{M} w_i F_i(x)$$

Weight vector 'w' is randomly generated for each solution during selection phase of each generation so that the search space is widened in all direction instead of having a fixed search direction, such that $\sum_{i=1}^{M} w_i = 1$. M represents the number of objectives considered for optimization.

Step 4: Select a pair of individuals using the selection probability given as

$$P = \frac{F_i}{\sum_{j=1}^{i} F_j}$$

Step 5: For each selected pair apply crossover operation to generate two new individuals with a crossover probability.

Step 6: For each bit value of the individuals generated, mutation operation is applied with a mutation probability.

Step 7: Randomly remove old individuals from the population and replace with new individuals having optimal fitness value for the next generation.

This approach aims to stipulate multiple search directions in a single run without using any additional parameters.

5 ENGA for Hardware/Software Partitioning

Genetic algorithms are capable of sampling large and complex search spaces. For a multi-objective optimization problem, a simple GA is not sufficient because these algorithms are well suited only for single-objective optimizations; hence, the multi-objective evolutionary algorithm proposed by Deb (2002) was used for the problem analysis. Elitist Non-dominated GA differs from simple GA mainly by fitness assignment procedure. Fitness assignment in ENGA is done by two methods.

(i) Non-domination sorting of individuals and identify the different non-domination fronts.
(ii) Apply crowding distance strategy to generate next generation population.

The overall ENGA procedure for hardware/software partitioning is outlined in the following steps.

Step 1: Combine parent population (P_t) of size N and offspring population (Q_t) of size N and create a population (R_t) of size 2N.

Step 2: Perform non-dominated sorting to R_t and identify the different fronts F_i, i=1, 2…, etc.

Step3: Set new population (P_{t+1}) empty and initialize counter 'i' to one. Until $|P_{t+1}| + |F_i| < N$, perform $P_{t+1} = P_{t+1} \cup F_i$ and increment the counter.

Step 4: Perform crowding sort procedure and include the most widely spread solutions into P_{t+1}. For each objective m=1, 2...M, sort the individuals in the descending order of fitness values and assign large distance to the boundary solutions, $d_{I_1} = d_{I_l} = \infty$ in each non-domination fronts. For all other individuals, j = 2 to (l-1), assign distances as given below

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{(I_j^m+1)} - f_m^{(I_j^m-1)}}{f_m^{\max} - f_m^{\min}}$$

The index I_j denotes the solution of the j^{th} member in the non-domination front, 'l' represents the number of solutions in the non-domination front.

Step 6: Create offspring population Q_{t+1} from P_{t+1} using crowding tournament selection, crossover and mutation operators.

Step 7: Repeat step 1 until the number of generations are reached.

The parameters f_m^{max} for the first objective (time) corresponds to the value of all-software implementation. For second objective (cost), it refers to the value of all-hardware implementation. The parameter f_m^{min} for the first objective corresponds to the value of all-hardware implementation and for the second objective; it refers to the value of all-software implementation.

6 MOPSO-CD for Hardware/Software Partitioning

MOPSO is based on the concept of PSO with slight variations. In PSO (Kennedy and Eberhart, 1995), a population (swarm) is initialized with random individuals, called "particles". All particles have fitness values that are evaluated by the function to be optimized. Each particle flies through the problem space with a velocity, which is constantly updated by the particle's own experience and the experience of the particle's neighbors, to search for optima iteration by iteration. Compared to genetic algorithms, the advantages of PSO are that it is easy to implement and there are fewer parameters to adjust. The velocity of each particle is updated by two best values. The first one is the best solution it has achieved so far. This value is called *pbest*. Another best value tracked by the optimizer is the best value obtained so far in the neighborhood of each particle. This best value is a local best and is called *lbest*. If the neighborhood is defined as the whole population, each particle will move towards its best previous position and towards the best position it has ever achieved in the whole swarm; this version is called the *gbest* model. In this paper multi-objective optimization, using MOPSO-CD (Tsou *et al*., 2006) is applied for hardware/software partitioning problem. The pseudo code is given below

Step 1: Initialization ( )
Step 2: generation ← 1
Step 3: while generation < MAX GEN do
Step 4: Flight ( )
Step 5: Calculate Objective Vector ( )
Step 6: Update Non-dominated Set ( )
Step 7: generation =generation + 1
Step 8: end while

The steps used for the flight procedure is given below

Step 1: Apply non-domination sorting for the particles in the population and are stored in an archive.

Step 2: Sort gbest particles based on crowding distance

Step 3: If the particles are dominated by top 10% less crowded area randomly select those particles as gbest particles else select from the rest of the archive, to fill up the population.

Step 4: The velocities of all particles at the time k+1 are updated using the particles objective values, which are functions of the particles current positions (x_k^i) in the design space at time k. The objective value of the particle determines which particle has the best global value (p_g^k) in the current swarm, and it determines the best position (p^i) of each particle over time Δt, i.e., in the current and all previous moves. The velocity updating formula uses these two data on the particle in the swarm, along with the effect of current motion (v_k^i), to provide a search direction (v_{k+1}^i) for the next generation. Update the velocity of the particle using

$$v_{k+1}^i = wv_k^i + c_1 rand \frac{(p^i - x_k^i)}{\Delta t} + c_2 rand \frac{(p_k^g - x_k^i)}{\Delta t}$$

where w represents an inertia factor, c_1 a self confidence factor and c_2 a swarm confidence factor, and these values can be adjusted to provide better results. Normally w, c_1 and c_2 can be fixed at 0.5, 1.5 and 1.5, respectively.

Step 5: Update position using $x_{k+1}^i = x_k^i + v_{k+1}^i \Delta t$

7 Applications

7.1 Case Study 1: DSP Applications

FFT algorithm is chosen for implementation in DSP applications. The algorithm is also efficient in the sense of the extendibility of the problem size. It is possible to extend from 8-point to 16-point FFT

algorithm and beyond by increasing the number of butterfly nodes. The 8-point FFT is first designed, the DAG of which is shown in Fig. 2.

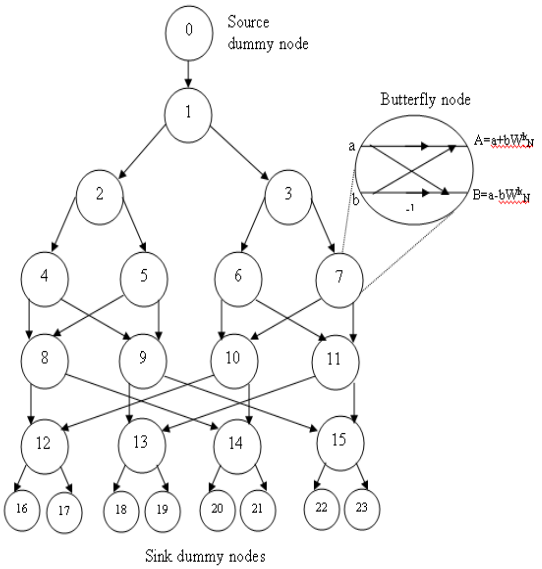

Fig. 2. DAG of 8-point FFT

Nodes 1, 2 and 3 are used for arranging the input data and are implemented as software-only tasks. Tasks 4 to 15 are butterfly computation nodes which are allowed to be implemented in both hardware and software. Butterfly node is assumed as a combination of one adder, one subtractor and one multiplier. The nodes 16 to 23 are dummy nodes for inputting and outputting data and they are implemented in software [7], [8]. The estimation of HW/SW area and time for the tasks in DAG of FFT is obtained using Quartus II development tools and NIOS II core respectively and is summarized in Table 2.

Table 2. 8-Point FFT Task Profile

Task	Description	HW execution time (ns)	HW area (LUTs)	SW execution time (µs)	SW area (KB)
1,2,3	Input Manipulator	3.00	3	1.00	11
4,5…15	Butterfly Node	75.70	40	8.36	176

7.1.1 Experimental Results

The butterfly node in the DAG of FFT is considered to be constructed using two adders and one multiplier task. The data in Table 1 are used for obtaining the task profile for FFT. Table 2 shows the task profile of 8-point FFT. The task profile of 16-point and 32-point FFT is generated based on 8-point FFT. The HW/SW partitioning is performed using multi-objective evolutionary algorithms namely WSGA [9], [10], ENGA [9], [11], [12] and MOPSO-CD [13] with the parameter settings indicated in Table 3.

Table 3. Parameter Settings Used in Multi-Objective Algorithms for FFT Benchmark

S. No	Parameter	WSGA	NSGA	MOPSO -CD
1.	Population Size	Number of nodes (Rounded)	Number of nodes (Rounded)	-
2.	Crossover Probability	0.9	0.9	-
3.	Mutation Probability	0.02	-	-
4.	Type of Crossover	Two Point	Two Point	-
5.	Number of Generations	100	100	100

The results obtained using WSGA, ENGA and MOPSO-CD algorithm are tabulated and compared in Table 4. From the results obtained it is found that ENGA and MOPSO-CD generates same optimal mean area and mean execution time for 8 point FFT.

Table 4. Simulation Results Obtained using WSGA, ENGA and MOPSO-CD for FFT Benchmark

Benchmark Circuit	WSGA		ENGA		MOPSO-CD	
	Mean Area (KB)	Mean Execution Time (µs)	Mean Area (KB)	Mean Execution Time (µs)	Mean Area (KB)	Mean Execution Time (µs)
8-Point FFT	236	36.83	**80**	**7.23**	**80**	**7.23**
16-Point FFT	2001	86.58	**1001**	**21.71**	1941	73.85
32-Point FFT	5894	257.20	**3721**	**117.63**	5593	217.90

Table 4. Run Time Comparison of Multi-Objective Algorithms for FFT Benchmark

Benchmark Circuit	Run Time (Seconds)		
	WSGA	ENGA	MOPSO-CD
8-Point FFT	1.70	1.06	**0.77**
16-Point FFT	7.02	**3.38**	6.50
32-Point FFT	41.76	**13.16**	35.14

For the other benchmark FFT DAGs, ENGA obtain less mean area and mean execution time with lesser run time compared to WSGA and MOPSO-CD. Fig.3 shows the run time, comparison plot derived on simulating the multi-objective algorithms. It is proved that ENGA generates pareto-optimal solutions faster than WSGA and MOPSO-CD algorithms.

7.2 Case Study 2: Image Applications

JPEG is widely used as a compression standard for both gray scale and color images. The DAG of the JPEG compression algorithm shown in Figure 4 consists of tasks such as Read-BMP, RGB2YCbCr, Encoder and Write-JPEG are forced to be implemented in software for convenience as in Wiangtong [7]. The modules namely Level Shifters, 2D-FDCTs (2 dimensional Fast Discrete Cosine Transform) and Quantizers can be implemented on either software or hardware. Among them FDCTs consume about 81% of the over all timing when implemented in software.

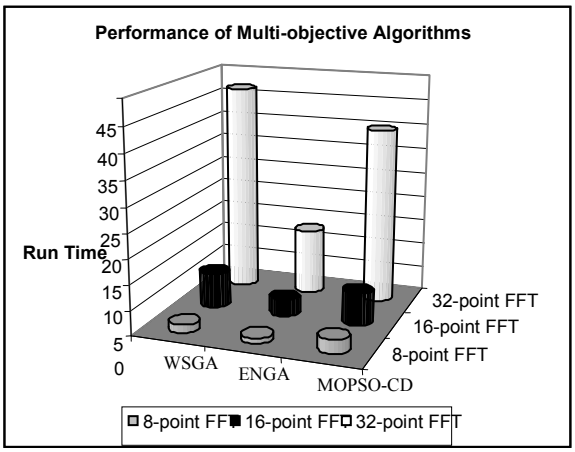

Fig. 3. Run Time Comparison Plot for FFT Benchmark

HW/SW implementation for JPEG-FDCT is carried out in this case study. Fig.5 shows the hardware structure of FDCT. DAG of JPEG-FDCT contains nodes with adder, subtractor, multipliers and I/O registers and the number of nodes and edges present in the DAG are 134 and 169 respectively.

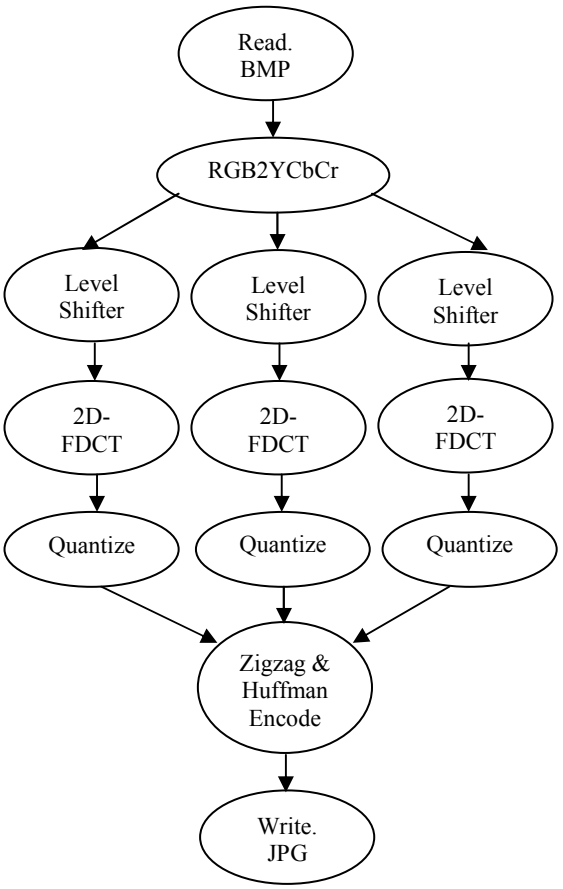

Fig. 4. DAG of JPEG Compression

7.2.1 Experimental results

The values from Table 1 are used for HW/SW partitioning of JPEG-FDCT benchmark. The parameter settings used for the simulation of the multi-objective algorithms for JPEG-FDCT benchmark is shown in Table 6. Fig.6, Fig.7 and Fig.8 shows the optimal solutions obtained by simulating WSGA, ENGA and MOPSO-CD algorithms respectively for 30 and 100 generations. The graph was plotted against area in kilo bytes and execution time in micro seconds. Fig.9 shows the relative comparison of the multi-objective algorithms for the JPEG-FDCT benchmark DAG. It is found that ENGA algorithm finds better optimal solutions than WSGA and MOPSO-CD algorithms.

The multi-objective optimization algorithms WSGA, ENGA MOPSO-CD are simulated with a uniform population size and are run for 30 and 100 generations. It is noted that ENGA searches pareto-optimal solutions faster than WSGA and MOPSO-CD and is shown in Table 7. This proves that ENGA outperforms the other two algorithms in terms of search time and in generating better pareto-optimal solutions.

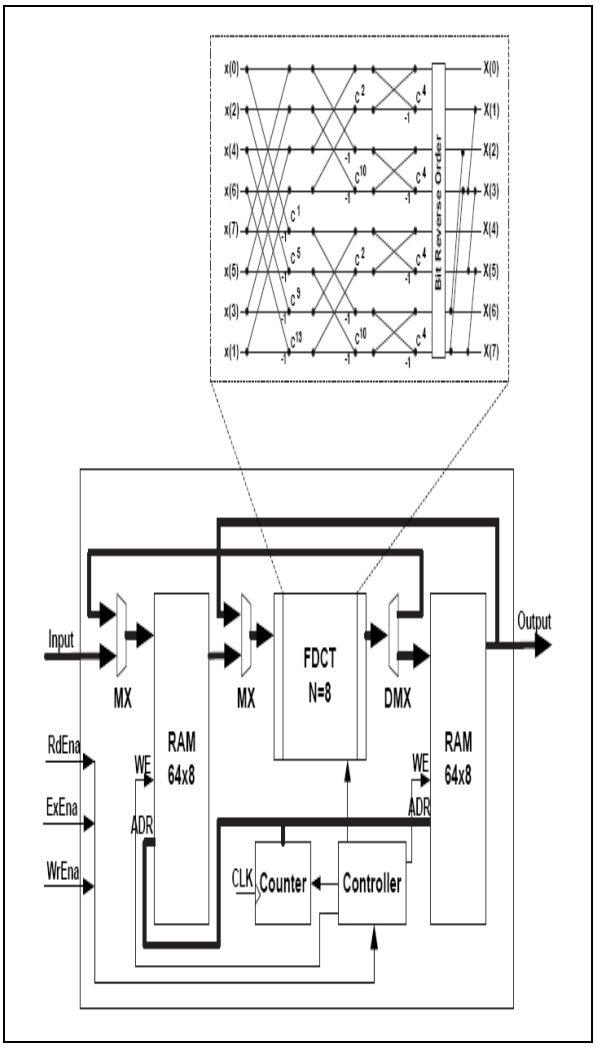

Fig. 5. Hardware Structure of JPEG-FDCT

Table. 6. Parameter Settings used for JPEG-FDCT Benchmark

S.No	Parameter	WSGA	NSGA	MOPSO-CD
1.	Population Size	Number of nodes (Rounded)	Number of nodes (Rounded)	-
2.	Crossover Probability	1	1	-
3.	Mutation Probability	0.02	-	-
4.	Type of Crossover	Two Point	Two Point	-
5.	Number of Generations	100	100	100

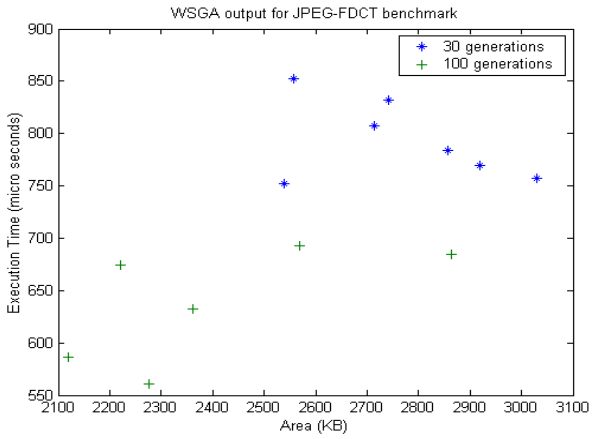

Fig. 6. Simulation Results Obtained using WSGA

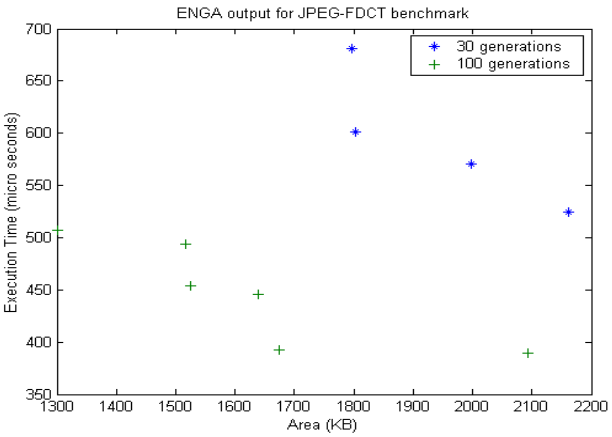

Fig. 7. Simulation Results Obtained using ENGA

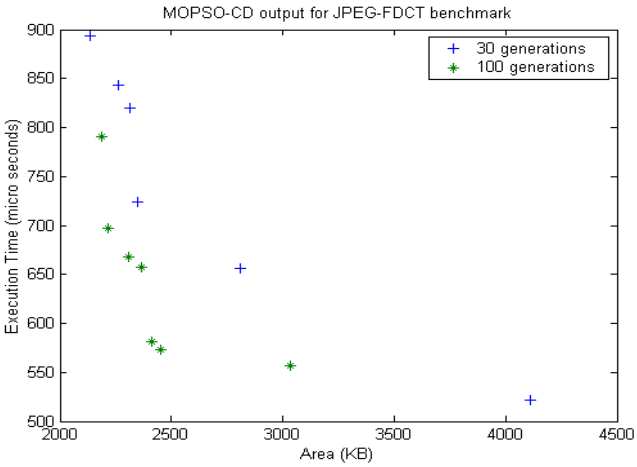

Fig. 8. Simulation Results Obtained using MOPSO-CD

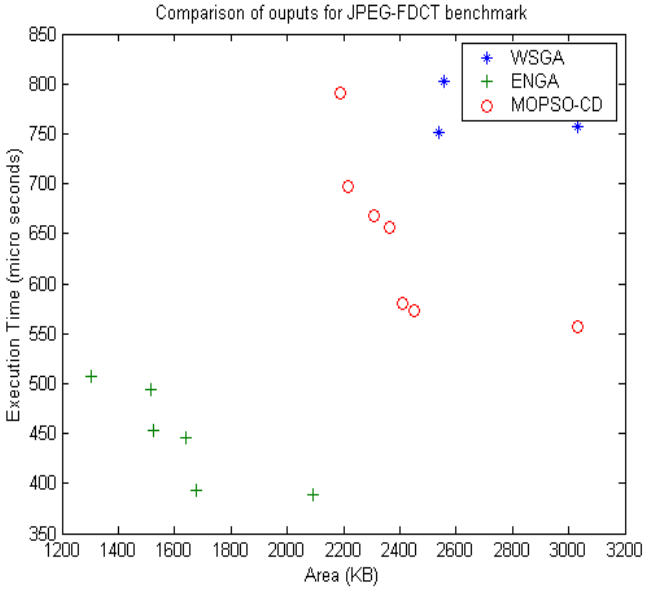

Fig. 9. Comparison plot of multi-objective Optimization Techniques for JPEG-FDCT Benchmark

Table 7. Comparison of Run Times of Multi-Objective algorithms for JPEG-FDCT Benchmarks

Algorithm	Run Time (Sec)
WSGA	66.48
ENGA	**12.39**
MOPSO-CD	16.89

8. Conclusion

This paper explains how Altera Quartus II design environment is used for obtaining HW/SW cost parameters such as the hardware area, the hardware time, the software area and the software time. HW/SW partitioning of FFT and JPEG FDCT are derived using the multi-objective optimization techniques WSGA, ENGA and MOPSO-CD algorithms. In summary, ENGA is found to perform better and generate pareto-optimal solutions faster than WSGA and MOPSO-CD for both the applications.

REFERENCES

[1] Zainalabedin Navabi, Embedded Core Design with FPGAs, Tata McGraw Hill, 2008.

[2] Quartus II Handbook Version 9.1 : http:www.altera.com

[3] Harkin, J., McGinnity, T. M. and Maguire, L. P. "Partitioning methodology for dynamically reconfigurable embedded systems" IEE Proc. Computer Digital Tech., Vol. 147, No. 6, pp. 391-396, 2000.

[4] Wu Jigang and Thambipillai Srikanthan, "Algorithmic aspects of hardware/software partitioning: 1D search algorithms", IEEE Transactions on Computers, Vol. 59, No. 4, pp. 532-544, 2010.

[5] Wu Jigang, Qiqiang Sun and Thambipillai Srikanthan, "Multiple-choice hardware/software partitioning: computing models and algorithms", in Proceedings of IEEE, pp. V2-61 – V2-65, 2010.

[6] Brain Baldwin and William P. Marnane, "An FPGA technologies area examination of the SHA-3 hash candidate implementations", Cryptology ePrint Archive: Report 2009/603.

[7] Theerayod Wiangtong, "Hardware/software partitioning and scheduling for reconfigurable systems", Ph.D thesis, Imperial College, London, 2004.

[8] Theerayod Wiangtong, Peter Y.K. Cheung and Wayne Luk, "Tabu search with intensification strategy for functional partitioning in hardware-software co-design", in Proc. of the 10th annual IEEE

symposium on Field-Programmable Custom ComputingMachines, pp. 297-298, 2002.

[9] Deb, K. "Multi-objective Optimization using Evolutionary Algorithms", John Wiley, and Sons Ltd, 2002.

[10] Deb, K. and Goldberg, D. E. "An investigation of the nichie and species formation in genetic function optimization", in Proc. of Third International Conference on Genetic Algorithms, pp. 42-50, 1989.

[11] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. "A Fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, Vol. 6, pp. 182-197, 2002.

[12] Blickle, T. "Theory of evolutionary algorithms and applications to system Synthesis", Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 1996.

[13] Tsou, C. S., Fang, H. H., Chang, H. H. and Kao, C. H. "An improved particle swarm pareto optimizer with local search and clustering", Lecture Notes in Computer Science, 4247, pp. 400-406, 2006.