A Synthesizable Hardware Evolutionary Algorithm Design for Unmanned Aerial System Real-Time Path Planning

Jonathan Kok, Luis Felipe Gonzalez, Rodney Walker
Australian Research Centre for Aerospace Automation (ARCAA)
Eagle Farm, Queensland 4009, Australia
j1.kok@qut.edu.au, felipe.gonzalez@qut.edu.au, ra.walker@qut.edu.au
Timothy Gurnett, Neil Kelson,
High Performance Computing & Research Support, Queensland University of Technology (QUT)
Brisbane, Queensland 4001, Australia
timothy.gurnett@qut.edu.au, n.kelson@qut.edu.au

Abstract

The main objective of this paper is to detail the development of a feasible hardware design based on Evolutionary Algorithms (EAs) to determine flight path planning for Unmanned Aerial Vehicles (UAVs) navigating terrain with obstacle boundaries. The design architecture includes the hardware implementation of Light Detection And Ranging (LiDAR) terrain and EA population memories within the hardware, as well as the EA search and evaluation algorithms used in the optimizing stage of path planning. A synthesisable Very-high-speed integrated circuit Hardware Description Language (VHDL) implementation of the design was developed, for realisation on a Field Programmable Gate Array (FPGA) platform. Simulation results show significant speedup compared with an equivalent software implementation written in C++, suggesting that the present approach is well suited for UAV real-time path planning applications.

1 Introduction

Unmanned Aerial Vehicles (UAVs) are widely used in military and civilian contexts. Military missions could involve target and decoy, reconnaissance, combat and logistics operations. In the civilian context, UAVs are being developed for environmental and agricultural purposes such as weather forecasting, storm and bush fire detection, farm field seeding and aerobiological sampling. All these scenarios involve a common task, which currently is determined by a human operator: Path Planning.

One of the main objectives in path planning is to develop optimization techniques which are effective and efficient in terms of computational cost and solution quality [Deb, 2001], [Lee *et al.*, 2008]. Traditionally, optimal path plans are found using deterministic optimizers. However, many approaches have a tendency to become trapped in local minima [Zheng *et al.*, 2003]. Instead, evolutionary algorithms (EAs) are preferable as the most viable search algorithms for a real-time UAV path planner [Allaire *et al.*, 2009]. These are more robust and allow them to find global solutions, but at a large computational expense. Hence of interest is a computationally efficient hardware implementation of a path planning algorithm based on EA running on an FPGA platform. The drawback of having a population based algorithm manipulated sequentially, which is a computationally intensive process, is overcome by exploiting the parallelism processing capability of the FPGA. Earlier work involving partial FPGA implementation of EAs for UAV real-time path planning [Allaire *et al.*, 2009] considered EA modules running on an FPGA platform, while the evaluation and simulation phases were performed on a PC. Results indicated that partial FPGA implementation could provide orders of magnitude speedups over software-only execution.

The main objective of this paper is to detail the development of a feasible hardware EA-based design to determine flight path planning for Unmanned Aerial System (UAS) navigating terrain with obstacle boundaries. The design architecture can provide UAS with autonomous real-time path planning capability using an EA entirely implemented on an FPGA platform, and includes hardware implementations of Light Detection And Ranging (LiDAR) source terrain data and EA population memories, as well as the EA search and evaluation algorithms used in the optimizing stage of path planning. The design will be herein referred to as Hardware-EA. As will be shown subsequently, simulation results for the Hardware-EA show significant speedup compared with an equivalent software implementation written in C++, suggesting that the present approach is well suited for UAS real-time path planning applications.

This paper is organized as follows. Section 2 gives a brief description of related work and the main difference in our approach. Section 3 describes in detail the architecture of the Hardware-EA. Section 4 describes an example of the Hardware-EA design implementation and its details. Section 5 discusses the computation time results of the PC-based EA and the Hardware-EA, and elaborates on the efficiency of the Hardware-EA. Section 6 presents a real world application. Section 7 concludes with a brief summary and possible enhancements for future work.

2 Background and Related Work

Although route planning has been widely researched, less attention has been given to UAV applications [Zheng *et al.*, 2003]. Both [Zheng *et al.*, 2003] and [Allaire *et al.*, 2009] elaborate on the importance of path planning, describe various types of path planning techniques, and argue why Evolutionary Algorithms are preferred as the most viable search algorithms for a real-time UAV path planner. [Lavelle, 2006] also presents a good summary of path planning algorithms, while the theory of circuit design using VHDL and FPGAs is extensive, and can be found in e.g. [Pedroni, 2004].

Previous work into path planning algorithms on hardware is limited but is now being considered by a number of researchers. Implementation of an EA on FPGAs has been explored in studies including those by [Allaire *et al.*, 2009], [Aporntewan and Chongstitvatana, 2001], and [Fernando *et al.*, 2010], and have shown promising results. [Allaire *et al.*, 2009] concluded that FPGA implementation of EA for UAV real-time path planning inherits the EA's optimal search advantage and overcomes the EA's computational disadvantage. However, they did not implement a fully synthesizable hardware design. [Aporntewan and Chongstitvatana, 2001] showed that their hardware Compact Genetic Algorithm (CGA) implementation implemented on an FPGA ran about 1,000 times faster than the software execution of their original code. [Fernando *et al.*, 2010] illustrated the use of hardware implementation of EA as an efficient optimization engine for evolvable hardware, having speedup of 5 times over an analogous software implementation. Their research focused mainly on the general theory of EAs being implemented on an FPGA. [Bonissone and Subbu, 2007] proposed a multi-objective EA architecture, but due to the simulator constraint, their research was limited to simulation without synthesizing their design. Their simulation results show a speedup of 328 times over its software counterpart. [Gallagher *et al.*, 2004] compares and contrasts a family of CGAs implementations on an FPGA. Their research concluded with recommendations to redesign the dataflow and to introduce systolic array methods to improve efficiency without increasing implementation cost.

The main difference with the approach used here and earlier work is the focus on an EA-based design architecture that is fully synthesizable and implementable on a single FPGA device, with reference to benefits and difficulties of practical application aspects (i.e. UAS path planning). The basic hardware design flow which this research aims to encapsulate is shown in Figure 1. To demonstrate implementation of the design in synthesisable VHDL, a modified version of the [Cocaud, 2006] EA search algorithm for UAV path planning has been used. More complex algorithms such as those given in [Lee *et al.*, 2008] could also be implemented and are being considered. Synthesis is important because it involves producing a netlist from VHDL that can be subsequently mapped onto an FPGA; from low level description to an even lower level description.

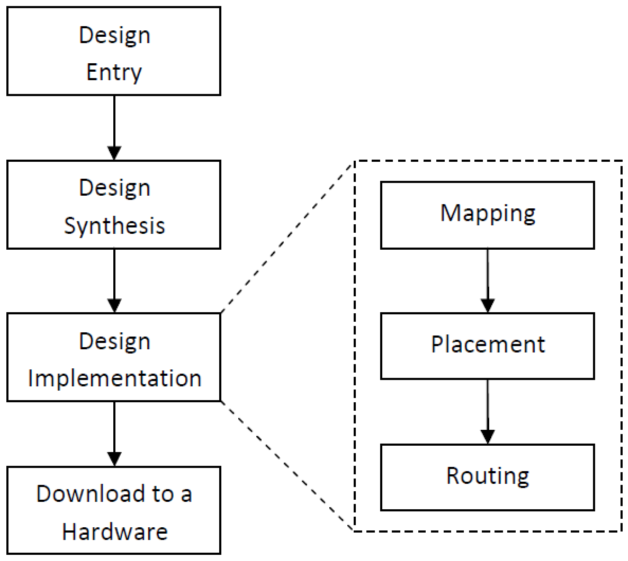

Figure 1: Basic hardware design flow

3 Hardware-Evolutionary Algorithm Design

3.1 The Architecture

The design architecture for the Hardware-EA proposed here is illustrated in Figure 2 and detailed in Figure 3 in the next subsection. The design is intended to fit into a single FPGA module, and includes units corresponding to typical EA tasks (such as selection, crossover, mutation, fitness evaluation, and so forth [Goldberg, 1989; Michalewicz, 1996]) where the functionality of each unit can be set according to the algorithmic requirements of the specific EA under consideration.

The driving component of the Hardware-EA is the Sort/Termination Unit (STU). The STU provides a memory interface, sorting algorithm, population update mechanism, and monitors the termination criterion of the evolutionary algorithm. In doing so, it acts as the main control unit of the Hardware-EA throughout the entire operation and is the Hardware-EA's sole output interface.

Additionally, a Terrain Memory (TM) unit is included where terrain information is stored (e.g. terrain data from LiDAR source stored in a single FPGA Block RAM). The initial and subsequent populations are then evolved based on this data, where each population member represents one possible path-solution for the UAV to traverse the terrain between designated initial and terminal waypoints.

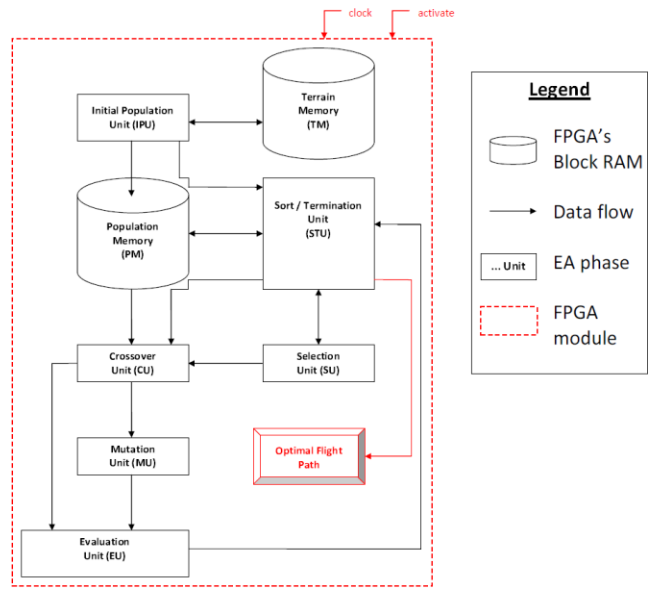

Figure 2: Overview of the Hardware-EA architecture

The design also includes an on-module Population Memory (PM) unit. This initially stores the "parent" population of path-solutions (e.g. stored in another FPGA Block RAM location) generated by the Initial Population Unit (IPU), but is also reused at each step of the EA iterative procedure to store the offspring population. Typically, each path-solution within a population is a bit stream consisting of a binary-code scheme representing the fitness, number of nodes and its transitional waypoints. A simple encoder/decoder is used to convert between this bit stream format and its decimal representation.

The two inputs which drive the Hardware-EA are the clock signal and the activation signal. The clock signal is connected to the embedded global clock of the FPGA, and the activation signal is connected to an input from the outside world.

Although not illustrated in Figure 2, the design includes an input line for initial transfer of terrain data to the TM unit, or subsequent data refresh of the TM unit from an external source to account for changed conditions such as UAV location. There is also an STU signal line where the optimal flight path computed by the module is output to the UAV for subsequent processing.

3.2 Operation

In this section, the flow of execution and communication between the individual units of the Hardware-EA is described and illustrated in Figure 3. Note that during the evolution phases, control is distributed; all units operate autonomously and asynchronously.

To begin the Hardware-EA process, an activation signal is received, initiating transfer and storage of externally generated terrain data into the TM unit. The TM unit interfaces with the IPU via two connections, an incoming address channel and an outgoing data channel. These channels allow the IPU to receive terrain information from the TM unit and generate, influenced by flight parameters such as minimum and maximum UAV elevation, a set of random path-solutions that are to be stored in the PM unit. These solutions are evaluated by the EU, sorted by the STU and then stored into the PM unit. This completes the initial setup of the Hardware-EA.

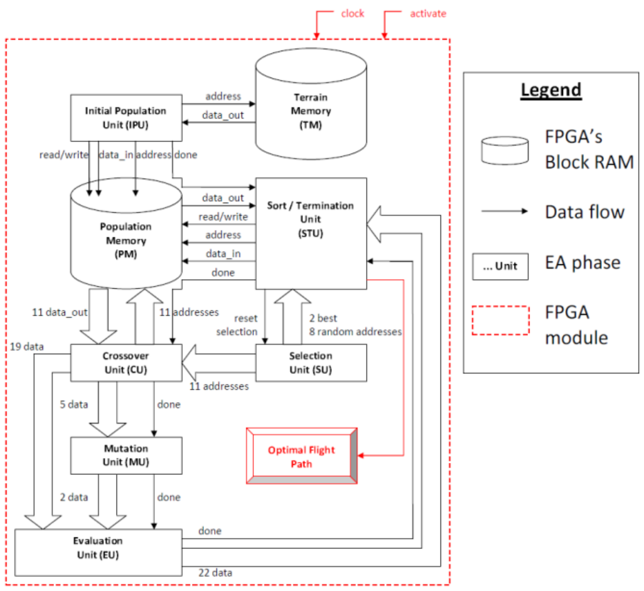

Figure 3: Detailed data path of the Hardware-EA architecture

To commence the first iteration of the Hardware-EA process, the STU notifies the Crossover Unit (CU) and Selection Unit (SU) that the Hardware-EA is ready to begin execution. The task of the SU is to generate random addresses to be used for the CU and population update for the STU. When the CU receives addresses from the SU, it requests these members from the PM unit and starts the crossover operation, creating new members. A selection of the new crossovered members is then passed to the Mutation Unit (MU) to be mutated. Once completed, the mutated members are sent to the Evaluation Unit (EU) for evaluation. The EU determines the fitness of new members generated by the CU and MU. Upon completion, the EU sends the evaluated members to the STU. Finally, the STU writes the new members and their new fitness values into the PM unit.

The above iterative step is repeated until the STU determines that the current Hardware-EA run is finished when the stopping criteria or convergence criterion is satisfied. It then transmits out the optimised flight path-solution, which is the best member decided through the fitness sorting algorithm.

4 Example of Hardware-EA Design Implementation

To explore the feasibility of the proposed design architecture, an implementation of the proposed Hardware-EA in synthesisable VHDL was undertaken. To facilitate comparison with previous work, the EA used by [Cocaud, 2006] for flight path planning was employed. Additionally, a development platform containing a Xilinx Virtex 4 LX200 FPGA processor was available. This platform was used to explore implementation issues such as the population characteristics and extent of parallelism possible within the design, subject to various constraints including the programmable logic resources available on the FPGA. Implementation details of the various elements and the EA population characteristics are briefly described below.

4.1 EA Population Characteristics

The implementation of the Hardware-EA requires specification of the EA population characteristics and operations. The operations of selection, crossover, mutation and evaluation are involved in the iterative EA process, as shown in Figure 4, and require specification. Selection involves identification of those members to undergo crossover from the current "parent" population of path-solutions.

```
Evolutionary Algorithm ();
     Initialise population;
     Evaluate initial population;
     WHILE convergence criteria IS NOT satisfied, DO
          Selection technique;
          Crossover operations;
          Mutation operations;
          Evaluation technique;
          Update population;
     END-WHILE;
```

Figure 4: Pseudo code of EA

For crossover of the selected path-solutions, all transitional waypoints after the middle of the path are truncated and swapped between the two selected parent path-solutions. The resultants are the offspring (see Figure 5).

Two selected offspring are further subjected to an insert mutation and a delete mutation to promote speed-up of convergence (as illustrated in Figure 6 and Figure 7). The insert and delete mutation takes into consideration the minimum and maximum number of transitional waypoints allowed, hence the population member is not corrupted with an invalid number of transitional waypoints. No swap mutation was considered as the algorithm is optimising a single objective function and the swap mechanism recommended by [Cocaud, 2006] is mainly for multi-objective functions.

The population is then updated using a semi-elitist approach, where a selection of the best path-solutions from the old population is retained and the remaining more inferior members are randomly overwritten by the offspring. Finally, the fitness of each member of the updated population is assessed based on feasibility and shortest distance. The iterative steps just described are repeated until a stopping criterion chosen for this implementation of sixty (60) generations has been completed.

Following a slightly modified [Cocaud, 2006] recommendation, the offspring from the new population is set to 70% of the old parent population. All of the offspring are bred from crossover, and 10% of the crossovered offspring are further subjected to mutation. Thereafter, a population update is performed where 5% of the best path-solutions from the old parent population are retained, and each of the offspring has replaced 70% of the remaining 95% of the old population.

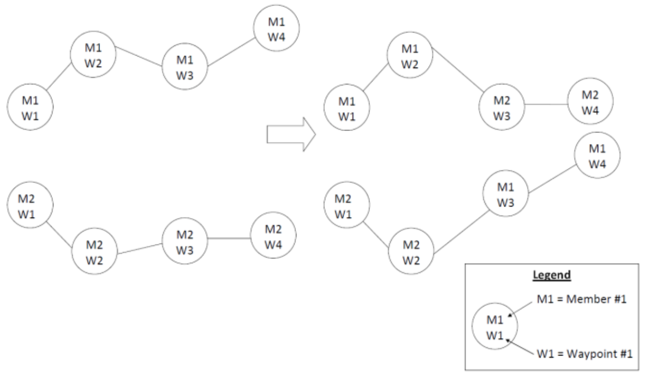

Figure 5: Example of crossover

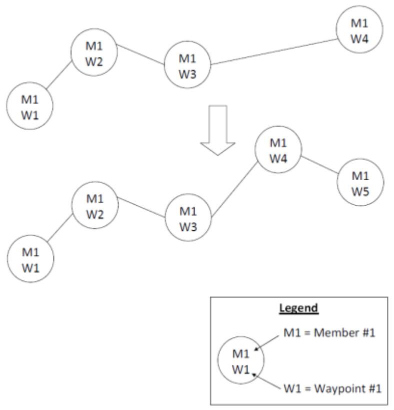

Figure 6: Examples of insert mutation

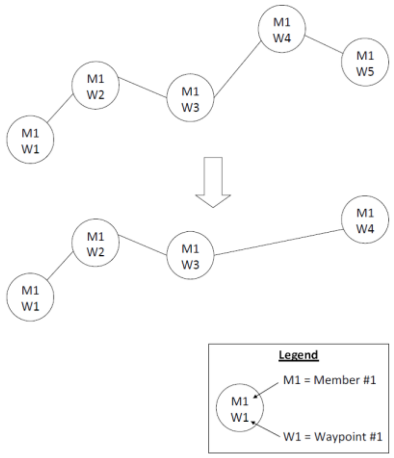

Figure 7: Examples of delete mutation

4.2 Implementation Details

Encoding of EA parameters: One of the first design decisions is determining the encoding of the parameters as this will enhance or hinder the computational time. Here, population members were chosen to correspond to single path-solutions comprising a minimum of two and a maximum of five transitional waypoints in addition to the starting and ending waypoints. Each transitional waypoint is characterized by its three spatial coordinates. The population size and starting/ending waypoints for all path-solutions are not subjected to change during the entire EA process. The bits encoding for the parameters of each member is depicted in Table 1.

Table 1
Encoding of EA parameters

Parameter	Number of bits	Integer range
Fitness	7 bits	[0,127]
Number of Transitional Waypoints (Excluding starting and ending waypoints)	2 bits	[0,3] ; where 00_2 = 2 waypoints 01_2 = 3 waypoints 10_2 = 4 waypoints 11_2 = 5 waypoints
Transitional Waypoint 1	27 bits = (X+Y+Z) ; where X = Y = Z = 9 bits	X = Y = Z = [0,511] ; where X = longitude Y = latitude Z = altitude
Transitional Waypoint 2	Same as Transitional Waypoint 1	Same as Transitional Waypoint 1
Transitional Waypoint 3	Same as Transitional Waypoint 1	Same as Transitional Waypoint 1
Transitional Waypoint 4	Same as Transitional Waypoint 1	Same as Transitional Waypoint 1
Transitional Waypoint 5	Same as Transitional Waypoint 1	Same as Transitional Waypoint 1
Total Number of bits	144 bits	

Terrain Memory unit (TM): For the example Hardware-EA implementation, sample terrain data to be used for path planning was gathered from a Light Detection And Ranging (LiDAR) source (see Figure 8) and converted, by a in house C++ application, into a bit stream represented by an array of latitude, longitude and altitude coordinates. Subsequently, this bit stream is stored in one of the FPGA's Block RAM modules. The testing environment map is large and consists of 55,556 points. Information for a point includes latitude, longitude, altitude, intensity and classification. The entire terrain is equivalent to 600 KB of data.

Selection Unit (SU): Within one clock cycle, the SU generates 19 different random addresses (11 for the CU and 8 for population update), which change only when instructed by the STU through the incoming control signal connection.

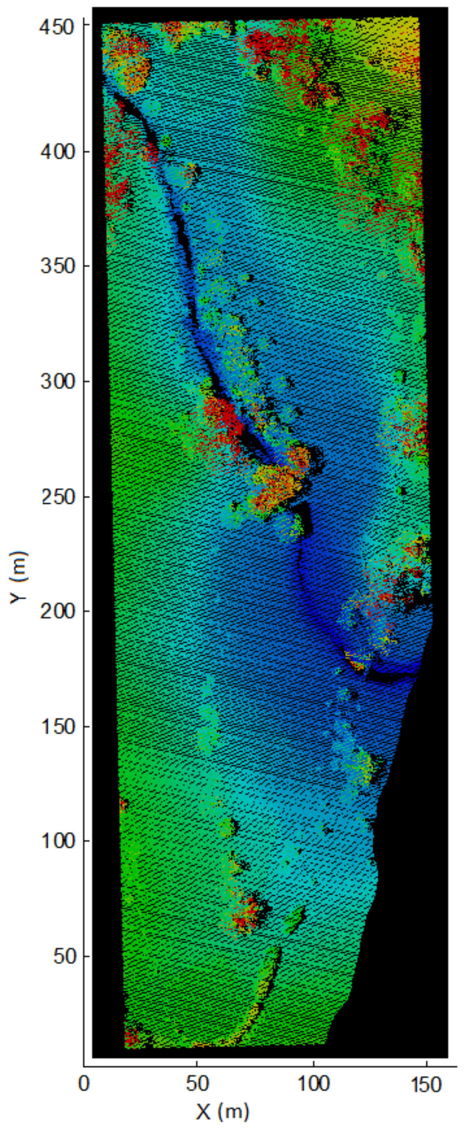

Figure 8: Two-dimensional view of testing environment

Population Memory (PM) & Initial Population Unit (IPU): The PM unit, like TM unit, is implemented using available FPGA Block RAM. As such, there is no shared memory external to the Hardware-EA system. Due to size limitations, the PM unit was fixed to store 32 population members (i.e. path-solutions). Initially, the IPU generates 32 random path-solutions, where each member includes 2 random transitional waypoints, the number of nodes, and the fitness.

Sort / Termination Unit (STU): The STU interface to the PM unit allows for the STU to read, in a single cycle, the entire list of path-solutions, sort them, and overwrite the original contents of the underlying Block RAM with the list of path-solutions now sorted by their individual fitness values.

The STU also interfaces with the EU to receive 22 offspring path-solutions (i.e. ~70% of the population size of 32) from the EU, and is responsible for storing them back in the PM unit using the previously mentioned semi-elitist approach. To these ends, the implementation of the STU interface to the PM unit includes both a data read and data write operation, however all control of this data transfer is initiated from the STU via a read/write control signal. The STU has two outgoing control signals (to the CU and SU) as well as two incoming control signals (from the IPU and EU). These signals determine the current operational state of the evolutionary algorithm. For a given iterative step of the EA process, the internal generation-counter within the STU is incremented if the termination criterion of 60 generations is not met, the SU is notified to generate a new set of random addresses, and the CU is notified to continue the evolution. Otherwise, the path-solution at the top of the PM unit is delivered externally from the FPGA module as the most optimised path-solution.

Crossover Unit (CU): The CU is composed of 11 identical processing modules. Notably, all crossover operations for a single generation are done in parallel. In one clock cycle, the CU utilizes the 11 addresses provided by the SU twice, generating a variety of parental combinations to produce 22 offspring path-solutions (i.e. ~70% of the population size of 32).

Mutation Unit (MU): The MU selects 2 of the offspring path-solutions (i.e. ~10% of the offspring size) from the crossover unit for mutation. One for insert mutation and the other for delete mutation. All of the mutation modules operate in parallel.

Evaluation Unit (EU): The EU evaluates the feasibility of the 22 new path-solutions (19 offspring and 2 mutated offspring) and generates a new fitness value for each. Equation (1) is used to calculate the fitness value, based on the feasible distance travelled;

$$\sum_{i=0}^{N-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 + (z_i - z_{i+1})^2} \quad (1)$$

where N is the total number of transitional waypoints and x, y and z are the latitude, longitude and altitude, respectively. Once again, these operations are all done in parallel.

4.3 Synthesize Details

The setup of the design synthesis is as follows. The design was synthesized with the Xilinx Virtex 4 XC4VLX200 FPGA as the target device, and the design goal was set to “balanced”. A “balanced” design implies that no optimization for speed or utilization of resources was considered. Once the design was synthesized successfully, it was the compiled and built for implementation, as shown earlier in Figure 1. This process consists of translating, mapping, placing and routing of the signals. For the design implementation process, no partition was specified and the design was translated and mapped successfully. All signals were placed and routed successfully as well, and all timing constraints were met. Table 2 shows the device utilization summary.

Table 2
Device utilization summary

Logic Utilization	Used	Available	Utilization
Number of Slices	18,415	89,088	20%
Number of Slice Flip Flops	9,051	178,176	5%
Number of 4 input LUTs	32,831	178,176	18%
Number of 18 Kb Block RAM	45	336	13%
Number of Bonded IOBs	146	960	15%
Number of GCLKs	1	32	3%

5 Preliminary Experiments and Results

5.1 Computation Time Comparison

Five experiments were conducted with the C++ executable of the EA and the average timing was recorded. The experiments were run on an Intel® Core™ 2 Duo Core CPU 2.66GHz. The Hardware-EA experiments were run on a FPGA simulator via Xilinx ISE 12.3. The entire terrain consisting 600 KB of data was used. Results for each EA phase are exhibited in Table 3.

Table 3
Timing results per EA cycle

EA phase	PC-based EA	Hardware -EA	Speed improvement
Crossover	4 ms	75 ns	53,000
Mutation	1 ms	250 ns	4,000
Evaluation	1,370 ms	200 ns	6,850,000
Selection + Population Update	0.501 ms	4750 ns	105
Total[a]	1,376 ms	5,275 ns	260,850

The tabulated time values are averages of the multiple runs.
[a]The total time is in reference to one complete EA generation.

5.2 Discussion

From Table 3, it can be seen that the hardware implementation provides significant speed improvement for all of the EA phases. The EA with a convergence criterion of sixty (60) generations results in a computational time improvement from 82 seconds to 0.3 milliseconds. This is mainly due to the fact that the PC-based EA had to run sequentially and the processing speed is CPU dependent. In the Hardware-EA design, parallelism incorporated by duplicating modules, and the Selection Unit generating a set of random numbers for the crossover in a single clock cycle, resulted in the significant speed improvement.

6 A Real World Application

6.1 Overview

The concept of having significantly fast algorithms is to realise real-time application. In a potential practical application, the Unmanned Aerial Vehicle (UAV) is flying at low level avoiding terrain while conducting an air sampling mission (see Figure 9). The UAV has to be able to fly the shortest path between certain points, while avoiding obstacles that are present at low altitudes. An external processor or device could be used to generate the TM unit directly from a LiDAR sensor.

Queensland University of Technology (QUT), in conjunction with the Cooperative Research Centre (CRC) for Plant Bio-security, Department of Agriculture and Food, Western Australia's Murdoch University, Queensland Department of Primary Industries and Fisheries, is currently involved in a project to develop an UAV to monitor inaccessible cultivation areas and sample air and look for either unwanted spores or other plant pathogens. An UAV fitted with such data collection system and air sampling device flying an optimal path can monitor and reduce the risk of pest introduction from international trade and, at the same time, will capture a wide range of plant health information in a cost-effective way so as to cover international and domestic market demands.

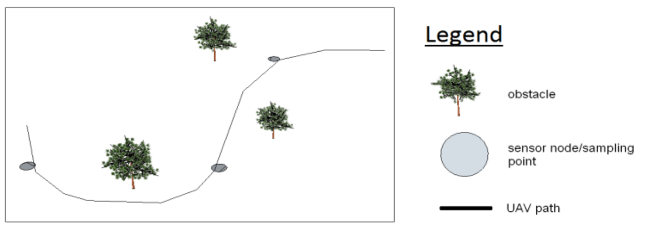

Figure 9: Practical operation scenario: low level flight air sampling mission

6.2 Implementation

The implementation process is best described as a flowchart depicted in Figure 10. The first step is to convert the LAS file format, generated by the LiDAR, to a text file. The second step is to process the points in the text file so that the EA can be applied. Next, the start and end waypoint positions are defined (step 3) and the optimal path is obtained using the EA (step 4). In Step 5, the LIDAR map will be loaded as an image or Digital Elevation Model (DEM) in the UAV ground station software. Step 6 will load the optimal waypoints to the autopilot ground station and simulate the mission. Finally, in Step 7, the mission is executed.

The LiDAR map was converted to a text file, an optimal path was found and subsequently the DEM was loaded to the UAV ground station. Figure 11 shows the three dimensional view of the optimal flight path mission mapped out in MATLAB.

Figure 12 shows the simulation environment of the mission running on Horizon™ MicroPilot flight simulator. Figure 12 also shows that a UAV helicopter is capable of navigating through the terrain via the shortest route and avoiding obstacles (large trees, vegetation, buildings) along the way.

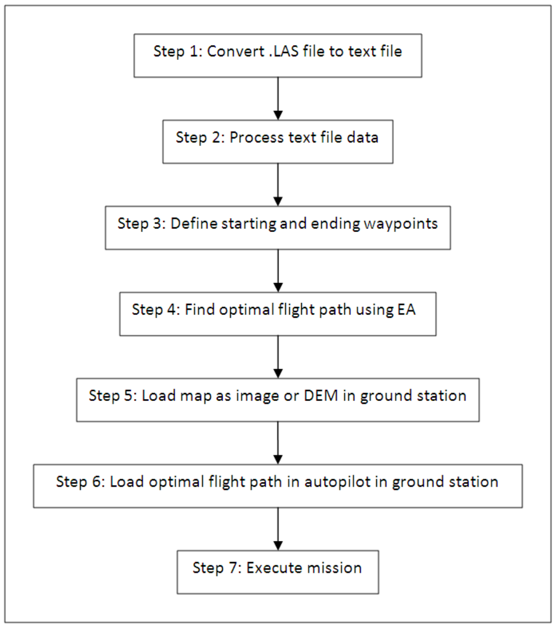
Step 1: Convert .LAS file to text file
Step 2: Process text file data
Step 3: Define starting and ending waypoints
Step 4: Find optimal flight path using EA
Step 5: Load map as image or DEM in ground station
Step 6: Load optimal flight path in autopilot in ground station
Step 7: Execute mission

Figure 10: Flowchart of the implementation process

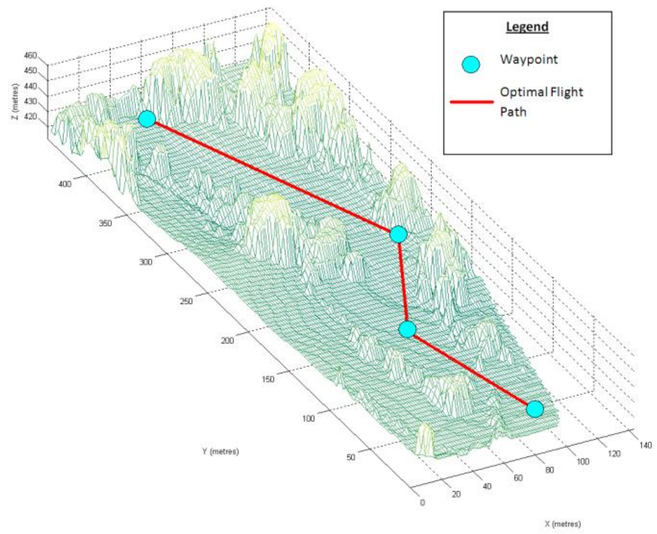

Figure 11: Three dimensional view from MATLAB optimal flight path simulation

7 Conclusion

This research has shown an enhancement in the computation speed of an EA hardware-based UAS path-planner. With this enhanced computation time, the system could be implemented and be used as an on-board path planner, re-computing flight plans in real-time. This implies the possibility of using it in a dynamic environment.

One development difficulty encountered was that the development time for the Hardware-EA using a low level hardware description language (HDL) such as VHDL is very time consuming. Future work will explore utilizing a high level HDL such as Mitrion-C, Handel-C or Impulse C. Additionally, designing a framework or interface between the TM unit and the outside world would bring the platform one step closer to hardware application implementation.

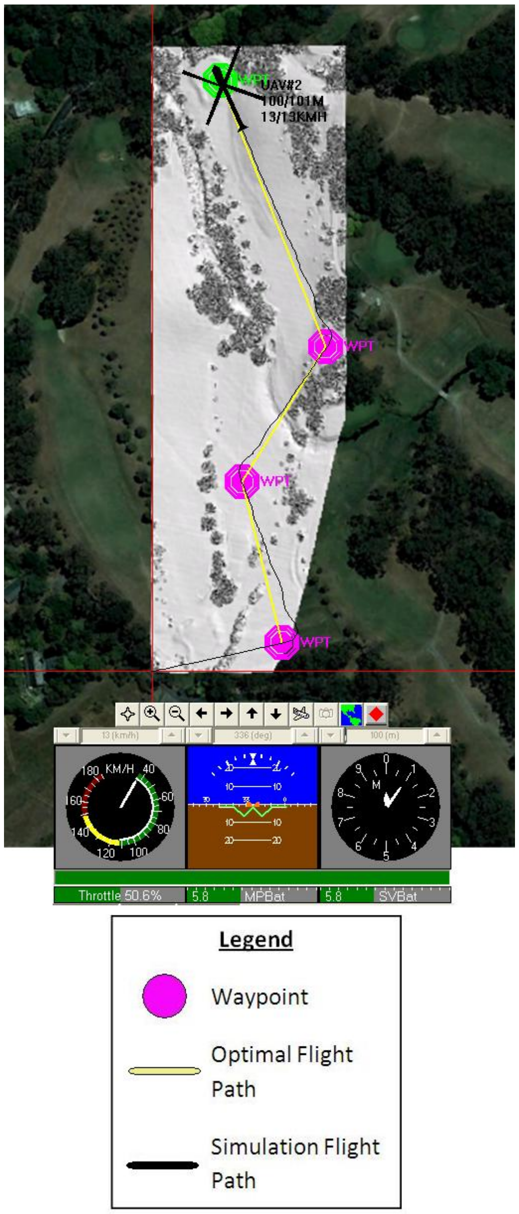

Figure 12: Top-down view from Horizon™ MicroPilot flight simulation of a flight path-solution

Acknowledgement

Computational resources and services used in this work were provided by the High Performance Computing and Research Support Group, QUT. The first three authors would like to acknowledge the Cooperative Research Centre for Spatial Information (CRCSI) for their help with the LiDAR files used in this work and the support of the CRC for National Plant Biosecurity.

References

[Allaire *et al.*, 2009] F. Allaire, M. Tarbouchi, G. Labonte and G. Fusina. FPGA implementation of genetic algorithm for UAV real-time path planning. *Journal of Intelligent & Robotic Systems*, 54(1-3): 495-510, 2009.

[Aporntewan and Chongstitvatana, 2001] C. Aporntewan, and P. Chongstitvatana. A hardware implementation of the Compact Genetic Algorithm. In *Proceedings of the 2001 Congress on Evolutionary Computation*, pp. 624-629, 2001.

[Bonissone and Subbu, 2007] S. Bonissone, and R. Subbu. Evolutionary Multiobjective Optimization on a Chip. *IEEE Workshop on Evolvable and Adaptive Hardware*, pp. 61-66, 2007.

[Cocaud, 2006] C. Cocaud. Autonomous tasks allocation and path generation of UAV's. Masters Thesis, Department of Mechanical Engineering, University of Ottawa, Ontario, 2006.

[Deb, 2001] Deb, K. Multi-objective optimization using evolutionary algorithms. Chichester ; New York, John Wiley & Sons, 2001.

[Fernando *et al.*, 2010] P. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum and A. Stoica. Customizable FPGA IP Core Implementation of a General-Purpose Genetic Algorithm Engine. *IEEE Transactions on Evolutionary Computation*, 14(1): 133-149, 2010.

[Gallagher *et al.*, 2004] J. Gallagher, S. Vigraham and G. Kramer. A family of compact genetic algorithms for intrinsic evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 8(2): 111-126, 2004.

[Goldberg, 1989] D. Goldberg. *Genetic algorithms in search, in: optimization and machine learning*. Boston, Kluwer Academic Publishers, 1989.

[Lavelle, 2006] S. M. Lavelle. *Planning Algorithms*. Cambridge, U.K., Cambridge University Press, 2006.

[Lee *et al.*, 2008] D. S. Lee, L. F. Gonzalez, K. Srinivas and J. Periaux. Robust evolutionary algorithms for UAV/UCAV aerodynamic and RCS design optimisation. *Computers & Fluids* 37(5): 547-564, Jun 2008.

[Michalewicz, 1996] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs.* 3rd rev. and extended ed. Berlin; New York, Springer-Verlag, 1996.

[Pedroni, 2004] V. A. Pedroni. *Circuit design with VHDL*. Cambridge, Mass.; London, MIT Press, 2004.

[Zheng *et al.*, 2003] C. W. Zheng, M. Y. Ding and C. P. Zhou. Real-time route planning for unmanned air vehicle with an evolutionary algorithm. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(1): 63-81, Feb 2003.