Proceedings of the World Congress on Engineering and Computer Science 2011 Vol I
WCECS 2011, October 19-21, 2011, San Francisco, USA

Using a Hardware Simulation within a Genetic Algorithm to Evolve Robotic Controllers

M. Beckerleg, J. Collins

Abstract— **This paper uses a novel method of implementing a genetic algorithm (GA) using a hardware simulation to evaluate the fitness of an individual for a robotic controller, rather than the normal practise of a software simulation. A simulation is required within a GA to model the actions of the robot and its environment in order to evaluate how well each individual within the population performs. Typically a simulation is written in software and executed sequentially on a processor. However, this paper implements the simulation as a digital circuit within a FPGA using a hardware description language (HDL). A comparison between identical hardware and software simulations is performed, resulting in the hardware simulation evolving a successful solution over seven hundred times faster than the software simulation. The robot is in the form of a balancing beam, the GA was implemented in hardware and the circuit driving the beam was a virtual FPGA.**

Index Terms—**Evolvable Hardware, Genetic Algorithm, Hardware Simulation, Evolvable Robotics, Virtual FPGA**

I. INTRODUCTION

This paper uses the novel approach of using a hardware robotic simulation within a GA. The basis of a GA is to find a solution to a problem using evolution as a search engine. The GA uses natural selection to evolve a population of individuals where each individual represents a possible solution to a problem. The process is iterative and is comprised of three main sections: reproduction, fitness evaluation and selection. The selection process determines which individuals within the population will survive to the next generation based on their fitness. The reproduction process creates new offspring from the surviving parents using the genetic operators crossover and mutation. The fitness evaluation determines how well each individual within the population performs as a potential solution to the problem with this process being the most time intensive. In order to evaluate the fitness of an individual it must be tested either in real life or in simulation. As it is time consuming and potentially destructive to evolve a real life robot, a robotic simulation is used. Historically a simulation is run in software on a computer. If the simulation could be implemented in hardware on a FPGA, then the mathematical equations describing the simulation could be executed in parallel and there should be a decrease in the time taken for fitness evaluation.

Manuscript received June 03, 2011; revised August 12, 2011.
Mark Beckerleg is with the School of Engineering, AUT University, New Zealand. phone: +64-9-9219999, fax: +64-9-9219973 (e-mail mark.beckerleg@aut.ac.nz)
John Collins is with the School of Engineering, AUT University, New Zealand. (e-mail john.collins@aut.ac.nz)

This paper created two identical simulations for a robotic controller (Fig 1) with the first coded in software and the second implemented in hardware. The GA used to evolve the virtual FPGA which controlled the robot and the virtual FPGA itself were identical and were implemented in hardware. This enabled a valid comparison between hardware and software simulation to be performed.

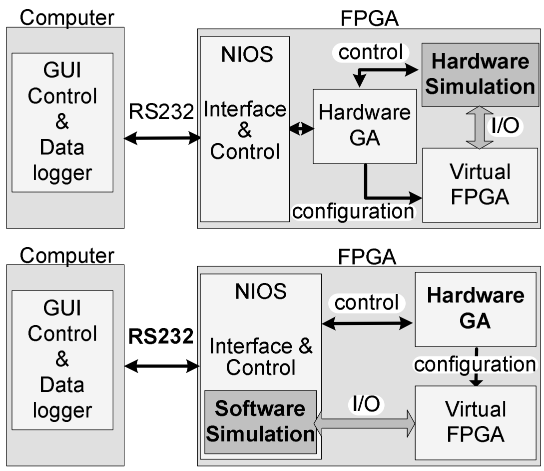

Fig 1. The two systems used to evaluate the software and hardware simulation.

The robotic platform used to test both simulations was a ball-beam system (Fig 2). This used a beam driven by a stepper motor to balance a ball between two end-stops. The beam had 19 sensors to determine ball position and a stepper motor which could move at 27.5 degrees per second to drive the beam. The beam itself was curved to make the system inherently unstable.

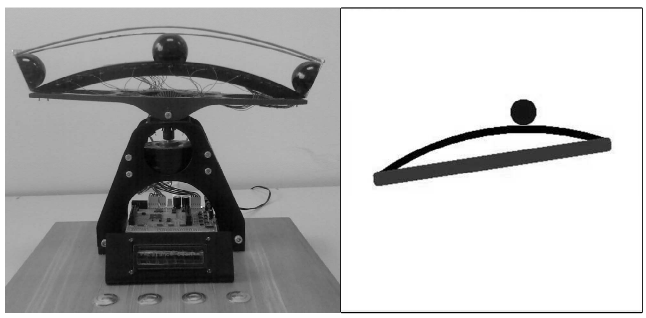

Fig 2. The physical beam with a GUI representation allowing the ball and beam to be dynamically observed during the evolution.

The main difficulty with creating a hardware simulation inside a FPGA is that unlike a computer, there is no arithmetic logic unit. All arithmetic formulae written in HDL will generate individual circuits to implement the arithmetic function. With floating point operations, a large number of the FPGA logic element resources are required for each calculation due to the complexity of dealing with

ISBN: 978-988-18210-9-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

signed, mantissa and exponent parts. As there are typically many floating point calculations in a simulation, it becomes impractical to use this technique.

An alternative to floating point calculations is the use of integer arithmetic, which reduces the logic element resources required to implement the circuit within the FPGA. Trigonometric functions will also need to be implemented as an arithmetic approximation or a look up table, as they are difficult to implement in hardware.

The disadvantage of using integer calculations is the loss of precision compared to floating point calculations. In addition the algorithms must be checked to make sure that no arithmetic overflow occurs as the numbers are confined to 32 bits ($\pm$ 2 x 10^9). It is also important to ensure that the timing between the arithmetic calculations and the timing between the simulation and other systems is correct. Finally the hardware simulation must be integrated to the GA and the virtual FPGA.

To implement the simulation in hardware, the integer arithmetic calculations can be directly coded in Verilog HDL using the standard multiply and divide syntax.

II. Background

There has been a large amount of research in the use of GAs to evolve robotic controllers using both software and hardware GA's. However to the to the authors knowledge the use of a hardware simulation within these systems has not previously been used. Using evolution to create robotic controllers has been widely studied with advances in path planning [1, 2], obstacle avoidance [3, 4], tracking [5, 6] and even evolving the robot form itself [7, 8]. This paper advances the field by the use of a hardware robotic simulation to improve the completion time for the GA process.

A virtual FPGA was used to control the motion of the beam. This is a digital circuit which was evolved by modifying its configuration bit stream (CBS) which determined the circuit parameters within the virtual FPGA. This method, referred to as evolvable hardware, was first implemented by Thompson [9] when he evolved a tone discriminator on a evolutionary tolerant Xilinx FPGA. However this type of FPGA has been discontinued and it has become difficult to directly evolve a commercial FPGA by modifying its CBS.

The main requirements of an evolvable FPGA are a) scalability to enable large systems to be evolved, b) partial reconfigurability, where parts of the FPGA can be reconfigured while other parts are still running and c) non destructive architectures that are resilient to a random CBS. One solution that meets these requirements is the virtual FPGA with functional elements employed in a Cartesian based array that could be downloaded into the FPGA. The operation of the functional elements and their inputs are determined by the CBS, thus evolving this bit stream would change the operation of the virtual FPGA. Virtual FPGAs have been evolved for several applications including an adaptive equalizer with lossy data compression, [10] image processing [11] and character recognition [12].

The hardware GA used in the experiments was a mutation only GA (MOGA) implemented without the use of the crossover operator, which requires less FPGA resources allowing the complete system to be implemented in a relatively small Altera Cyclone EP1C12F324C8 device. This technique has been used previously in both hardware and software GAs with studies that showing that a MOGA can compare favorably against a normal GA [13]. An advantage of this technique is a reduction in chromosome damage caused by the crossover operator [14]. Various mutation only algorithms have been studied such as frame shift and translocation, once again finding a good comparison with a normal GA [15]. Several papers have used this method to evolve digital circuits.[12, 16, 17]. Other hardware GA systems have used crossover templates to minimize the number of two bit multiplexers for the crossover operation [18], while others have used pipelining and parallelism to develop a high speed hardware GA[19].

III. Mathematical Model

In the model of the beam (Fig 3), the beam position is measured as an angle φ from horizontal, and the ball position is measured as an angle θ from the centre of the beam. The full derivation for the mathematical model has previously been described by the authors [20]. The final equations for the ball acceleration are given in equations (1) and (2).

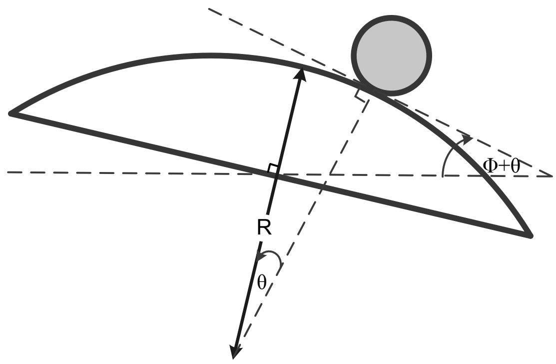

Fig 3. The ball and beam showing the relationships between the angles and motion.

$$\ddot{\theta} = A(\theta + \emptyset) \tag{1}$$
$$A = \frac{g}{R(1+\frac{I}{mr^2})} \tag{2}$$

From physical experimentation on the beam, the value for acceleration (a) of the ball was determined as a factor of the ball position (x) and beam position (b) in equation (3).

Placing this into the mechanical modeling we can determine the new position of the ball, dependant on its current position, velocity (v) and acceleration in equation (4), and the new speed of the ball dependant on its current speed and acceleration in equation (5). The simulation was set to a time period of 1 ms, in equations (6) and (7) and these were modified to give a divisor which was a multiple of two, enabling for efficiencies in the hardware implementation, in equations (8) and (9).

$$a = 12x + 2.8b \tag{3}$$
$$x_{new} = x + vt + \frac{at^2}{2} \tag{4}$$
$$v_{new} = v + at \tag{5}$$

$$x_{new} = x + \frac{v}{10^3} + \frac{12x+2.8b}{2x10^6} \qquad (6)$$

$$v_{new} = v + \frac{12x+2.8b}{10^3} \qquad (7)$$

$$x_{new} = x + \frac{1049v}{2^{20}} + \frac{101x+24b}{2^{24}} \qquad (8)$$

$$v_{new} = v + \frac{786x+184b}{2^{16}} \qquad (9)$$

Where
- g - gravitational acceleration
- I – moment of inertia of the ball
- R - radius of curvature of the beam
- m - mass of the ball
- r - radius of the ball
- θ - ball position (angle from the centre)
- Ø - beam position (angle from horizontal)
- x – ball position
- v – ball velocity
- b – beam position
- a - acceleration of the ball

An investigation of the hardware simulation generated by the HDL compiler showed that no dividers and only four signed multipliers were used in the simulation circuit.

The beam state is defined by ball position, ball speed and beam position which were derived from the ball position sensors and the number and direction of the motor pulses. These parameter values were stored in a 32 bit word with 19 ball positions, 3 ball speeds and 10 beam positions, with only 1 bit active for each parameter at any one time.

Within the hardware simulation these parameters were encoded as integer values scaled up by 10000 to maintain accuracy. The first parameter was the beam position which was an angular parameter determined by the number of beam pulses. The beam had a total movement of $\pm 30^0$ with 270 pulse required to move the beam over the total range giving a movement per pulse of 0.22^0 and a pulse range from ±135. As the motor could only be pulsed every 8ms then the beam movement per 1ms was 0.0275^0.

The location of the ball was determined by 19 sensors. When the ball was midway between two sensors both became active. This enabled the resolution of the sensors to be doubled giving a range from ±19.

The ball velocity was estimated from the time between sensor signal changes, however the velocity estimation was quite poor thus only three ball velocities were used; moving left, moving right and almost stationary.

The complete system (Fig 4) shows the three blocks of the genetic algorithm; a) the hardware GA, b) the virtual FPGA and c) the hardware simulation, each with its associated control lines. These lines controlled the GA process, and allowed the transfer of fitness, best chromosome and ball states to the computer GUI. The virtual FPGA controlled the motion of the beam dependent on the ball-beam states, and it was this circuit that was evolved via a GA process on its CBS. The hardware simulation modeled the dynamics of the ball-beam so the fittest of the current virtual FPGA circuit could be evaluated. The hardware GA used a mutation only algorithm where the crossover operator was not used. This reduced the number of logic elements required for the hardware GA, enabling the complete system to be placed in a Cyclone device using less than 12000 logic elements.

IV. System Overview

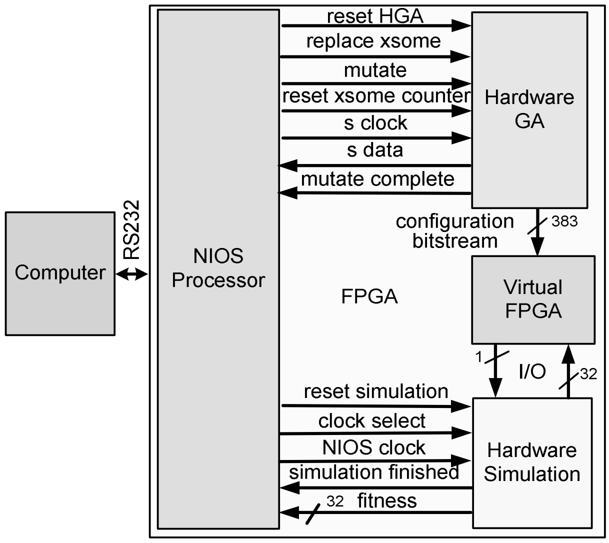

Fig 4. System overview showing connections between the NIOS processor and hardware subsystems

V. Hardware Genetic Algorithm

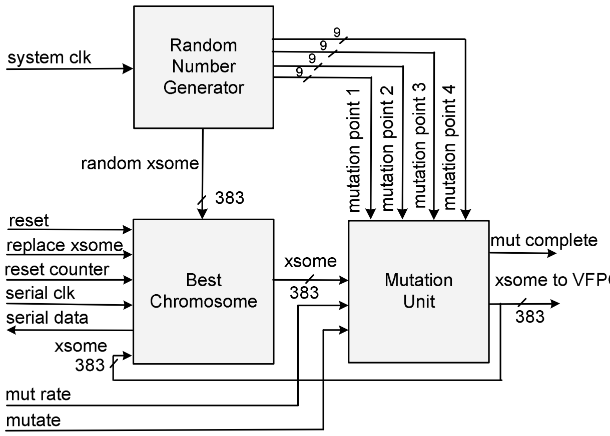

Fig 5. The hardware genetic algorithm showing the random number generator, chromosome storage, mutation unit and control lines.

The hardware genetic algorithm unit (Fig 5), had 3 blocks: random number generator, best chromosome storage and the mutation unit. The mutation random number generator used a linear feedback shift register to produce a 383 bit random number as well as four 9 bit random numbers. The best chromosome stored the current parent which could be sent to the mutation unit and sent to the computer via a serial link. The mutation unit could mutate 1 to 4 bits of the chromosome dependant on the selected mutation rate.

The operation was implemented as follows: on reset, a random CBS was generated, placed into the best chromosome memory, and then passed to the mutation unit. The mutation unit could mutate 1 to 4 bits within the CBS dependant on the fitness. The mutation point was set by a 9 bit random number giving a range of 0 to 511. As the CBS was only 383 bits, there was a possibility that a mutation would not occur (Table I). After mutation the CBS was sent to the virtual FPGA for evaluation. If the CBS had an equal or better fitness, it would be saved back in the best chromosome memory.

TABLE I
MUTATION RATE AND MUTATION PROBABILITY

Mutation Bits	Mutation Probability	Mutation Rate
1	75%	0 - 0.3%
2	94%	0 - 0.5%
3	98%	0 - 0.8%
4	99.6%	0 - 1.0%

VI. VIRTUAL FPGA

The circuit produced in a FPGA is determined by the CBS. Evolvable hardware uses a genetic algorithm to modify the CBS to produce a circuit that can be evolved. However a normal FPGA cannot be used, as a randomly generated CBS can destroy the FPGA. To overcome this problem, a virtual FPGA was created which was tolerant to random CBS. The virtual FPGA (Fig 6) used in this paper was based on the Cartesian based model consisting of functional elements (FE) which were grouped into five layers with the outputs of each layer feeding into the next.

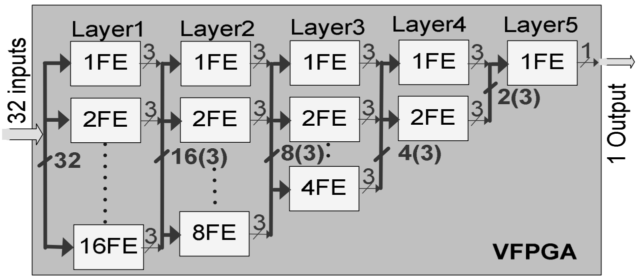

Fig 6. The virtual FPGA showing the hardware in the first two layers and the reducing layer structure within the device.

The configuration for the first and subsequent FEs is different (Fig 7). The first layer does not contain a function LUT. Each FE selects any 3 of the 32 inputs feeding them as a group of 3 bits to the next layer. The FE within the second and subsequent layers can select two groups from the previous layer and pass them to the function LUT which can perform Boolean and arithmetic functions on the two input groups. The operation of the multiplexers and the function LUT are controlled by the CBS which is 383 bits long.

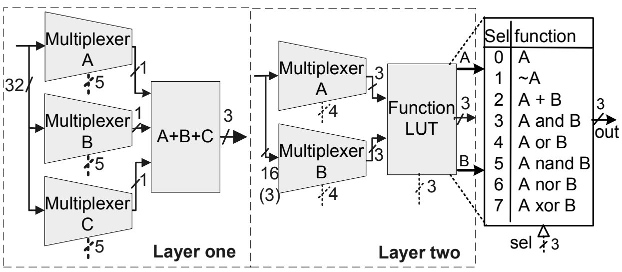

Fig 7. The FE in layer 1 multiplexes the inputs into groups of 3, while the FE in the second and subsequent layers select groups from the previous layer and performs Boolean and arithmetic operations on them.

VII. HARDWARE SIMULATION

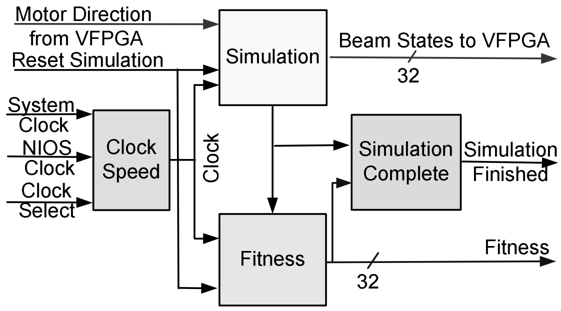

Fig 8. The hardware simulation showing the simulation and fitness calculation blocks with their associated control lines.

The hardware simulation (Fig 8) was comprised of four units: simulation, fitness, simulation complete and clock speed. The simulation unit contained the simulation's mathematical equations implemented in hardware, and the input-output to the virtual FPGA. The fitness unit had a 32 bit register which was incremented on every clock pulse, with each pulse equivalent to 1ms of simulation time. The clock speed unit switched the clock source between a 50MHz clock and a clock driven by the NIOS. Controlling the clock with the NIOS allowed the ball-beam states to be sent to the GUI, allowing the motion of the ball and beam to be displayed as well as enabling the simulation to be paused. The simulation complete unit was used to end the simulation when the fitness counter had reached 300 seconds.

When reset, the fitness counter was cleared and the simulation ball-beam parameters set to a starting position with the ball on the left side of the beam, with the beam set to an angle of 20^0. When not reset, the simulation mathematical equations were calculated on each clock cycle. The mathematical equations for the simulation were designed for a period of 1 ms, i.e. every clock pulse was equivalent to a one millisecond time period within the simulation.

After each clock pulse, the beam would be shifted left or right one motor step, dependant on the motor direction input from the virtual FPGA. The new integer ball speed, ball position and beam position would then be calculated, with these values then being converted into a thirty two bit binary format representing the new beam and ball state to be feed to the virtual FPGA. The simulation had an output control line to show when the simulation had finished. This was set whenever the ball position reached either of the two beam end-stops.

The fitness counter could be read by the NIOS processor at any time, with the value of the fitness counter being the time in milliseconds that the ball had remained balanced. The simulation finished line was also connected to the NIOS processor so that the fitness counter could be read at the end of a simulation

VIII. RESULTS

The first requirement was to ensure the software and hardware simulations operated in the same manner. To test this, a recording was taken of the ball position as it moved down the beam which was fixed at a 20^0 angle.

ISBN: 978-988-18210-9-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE II THE MOTION OF THE BALL FALLING ON BOTH THE HARDWARE AND SOFTWARE SIMULATION.

Software Simulation				Hardware Simulation 5MHz				Time
Ball positn	ball speed	beam positn	Time (ms)	Ball positn	ball speed	beam positn	Time (ms)	Between Sensors
0	1	9	0	0	1	9	0	
0	2	9	184	0	2	9	184	
1	2	9	229	1	2	9	229	229
2	2	9	445	2	2	9	445	216
3	2	9	581	3	2	9	581	136
4	2	9	656	4	2	9	656	75
5	2	9	715	5	2	9	715	59
6	2	9	775	6	2	9	775	60
7	2	9	815	7	2	9	815	40
8	2	9	850	8	2	9	850	35
9	2	9	881	9	2	9	881	31
10	2	9	923	10	2	9	923	42
11	2	9	947	11	2	9	947	24
12	2	9	970	12	2	9	970	23
13	2	9	991	13	2	9	991	21
14	2	9	1015	14	2	9	1015	24
15	2	9	1033	15	2	9	1033	18
16	2	9	1050	16	2	9	1050	17
17	2	9	1070	17	2	9	1070	20
18	2	9	1085	18	2	9	1085	15
18	2	9	1096	18	2	9	1096	11

The table (Table II) shows the ball position starting on the left, with the ball speed initially stopped, then moving down the beam to the right with the beam in a fixed position. It can be seen from the table that both simulations are identical. On analysis, the ball slowly moved to the right, increasing in speed as the ball progressed along the beam. This is indicated by the decreasing time as the ball passed the sensors, starting off slowly and then increasing in speed due to the increased slope of the beam and the pull of gravity. Note the ball position sensors used for the GA were not evenly spaced and thus the time taken for the ball to pass between the sensors was not uniform either.

The second test was to compare the evolutionary progress of the two systems. The graphs of the fitness level relative to the current generation (Fig 9 and Fig 10) show that the evolution occurs in stages. A close investigation of the fitness and the movement of the ball showed that there were five stages to the evolutionary process. In all these stages it should be remembered that the beam has only two speeds, left and right. These stages are: Stage I balancing less than one second, the ball simply moved down the beam with no response to the motor. Stage II balancing less than two seconds, there was a jittering of the beam around a static position not dependent on the ball position. Stage III balancing less then ten seconds, there are several jitter points that are linked to the ball position. Stage IV balancing less than 300 seconds, the beam moved in such a fashion as to slow the ball down and kept it relative stationary for long periods of time. When the ball did move the beam would track it and bring it back to a relatively stationary period again. Stage V was a successful solution. The behaviour of the ball-beam was such that the ball would be semi stationary and move around a jitter point on the beam. Eventually the ball would break from this spot and move to the opposite side of the beam. The beam would then move to correct, and bring the ball back to the original location where the pattern would repeat indefinitely.

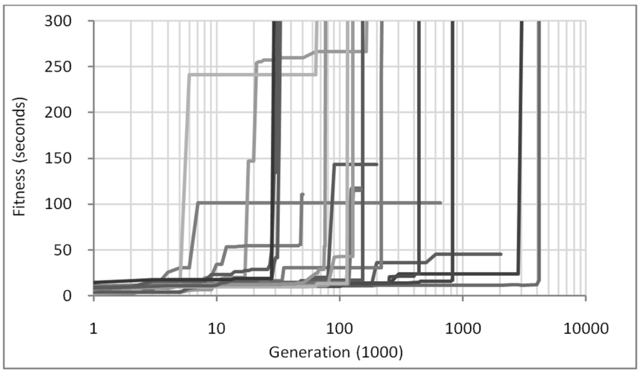

Fig 9. The fitness relative to the number of generations for the software simulation.

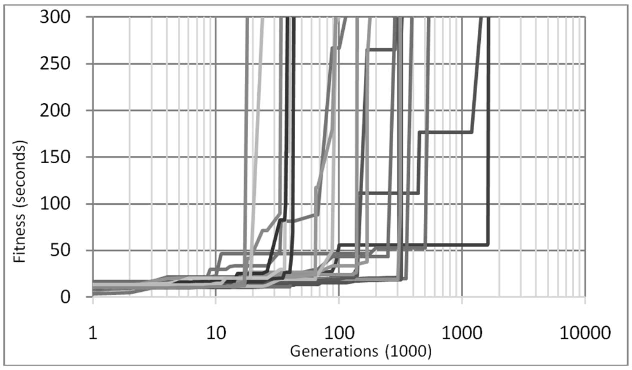

Fig 10. The fitness relative to the number of generations for the hardware simulation.

The final test was to compare the time taken for a successful evolution. These were plotted (Fig 11 and Fig 12), and the times compared. The average time for a successful evolution using a software simulation was 80,000 seconds, whereas the hardware simulation for this time was reduced to 110 seconds. Thus the hardware simulation could evolve a successful circuit on average 700 times faster than an identical software simulation.

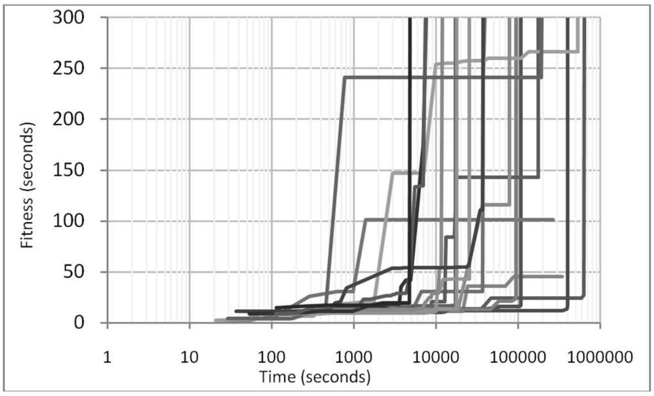

Fig 11. The software simulation with fitness and v time showing an average time of 50 thousand seconds to a successful evolution.

ISBN: 978-988-18210-9-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

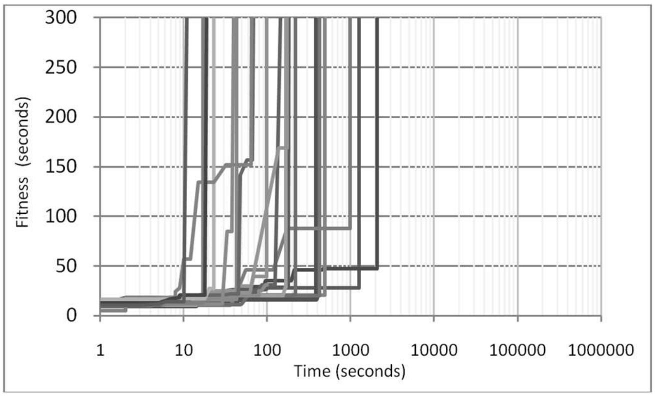

Fig 12. The hardware simulation with fitness v time showing an average time of eighty seconds to a successful evolution.

IX. Conclusion

A hardware simulation replicating a balancing beam has been successfully implemented. This simulation has been used in a hardware GA to evolve a virtual FPGA that was capable of balancing the ball on the beam for more than five minutes. A comparison between identical software and hardware simulations was performed with both systems behaving in an identical manner. It was found that the hardware simulation could evolve successful circuits over 700 times faster than the software simulation.

References

[1] D. A. Ashlock, T. W. Manikas, and K. Ashenayi, "Evolving A Diverse Collection of Robot Path Planning Problems," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2006, pp. 1837-1844.

[2] K. Daehee, H. Hashimoto, and F. Harashima, "Path generation for mobile robot navigation using genetic algorithm," in *Industrial Electronics, Control, and Instrumentation, 1995., Proceedings of the 1995 IEEE IECON 21st International Conference on*, 1995, pp. 167-172 vol.1.

[3] Y. Z. Renato A. Krohling, and Andy M. Tyrrell, "Evolving FPGA-based robot controllers using an evolutionary algorithm," 2002.

[4] A. M. Tyrrell, R. A. Krohling, and Y. Zhou, "Evolutionary algorithm for the promotion of evolvable hardware," *Computers and Digital Techniques, IEE Proceedings-,* vol. 151, pp. 267-275, 2004.

[5] K. C. Tan, C. M. Chew, K. K. Tan, L. F. Wang, and Y. J. Chen, "Autonomous robot navigation via intrinsic evolution," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, 2002, pp. 1272-1277.

[6] R. R. Cazangi, C. Feied, M. Gillam, J. Handler, M. Smith, and F. J. Von Zuben, "An evolutionary approach for autonomous robotic tracking of dynamic targets in healthcare environments," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, 2007, pp. 3654-3661.

[7] T. Koyasu and K. Ito, "Acquisition of the body image in evolution -Role of actuators in realizing intelligent behavior," in *Modelling, Identification and Control (ICMIC), The 2010 International Conference on*, pp. 859-864.

[8] M. Mazzapioda, A. Cangelosi, and S. Nolfi, "Evolving morphology and control: A distributed approach," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, 2009, pp. 2217-2224.

[9] A. Thompson, "An evolved circuit, intrinsic in silicon, entwined with physics.," *Proc. 1st Int. Conf. on Evolvable Systems (ICES'96),* pp. 390-405, 1997.

[10] M. Iwata, I. Kajitani, H. Yamada, H. Iba, and T. Higuchi, "A Pattern Recognition System Using Evolvable Hardware," *Lecture Notes In Computer Science,* vol. 1141, pp. 761-770, 1996.

[11] L. Sekanina, *Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware*, 2606 ed., 2003.

[12] J. P. Wang, C.H. Lee, C. H., "FPGA Implementation of Evolvable Characters Recognizer with Self-adaptive Mutation Rates," in *International Conference on Adaptive and Natural Computing Algorithms ICANNGA'07*, Warsaw, Poland, 2007, pp. 286-295.

[13] Z. Zhu, D. J. Mulvaney, and V. A. Chouliaras, "Hardware implementation of a novel genetic algorithm," *Neurocomput.,* vol. 71, pp. 95-106, 2007.

[14] T. L. Lau and E. P. K. Tsang, "Applying a mutation-based genetic algorithm to processor configuration problems," in *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*, 1996, pp. 17-24.

[15] I. De Falco, A. Della Cioppa, and E. Tarantino, "Mutation-based genetic algorithm: performance evaluation," *Applied Soft Computing,* vol. 1, pp. 285-299, 2002.

[16] L. Sekanina and S. Friedl, "An Evolvable Combinational Unit for FPGAS," *Computing and Informatic,* vol. 23, pp. 461-486, 2004.

[17] L. Sekanina, T. Martinek, and Z. Gajda, "Extrinsic and Intrinsic Evolution of Multifunctional Combinational Modules," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2006, pp. 2771-2778.

[18] S. G. Shackleford B., Carter, R.J., Okushi E., Yasuda M., Seo K. Yasuura H. , "A High-Performance, Pipelined, FPGA-Based Genetic Algorithm Machine " *Genetic Programming and Evolvable Machines,* vol. 2, pp. 33-60, 2004.

[19] T. Maruyama, T. Funatsu, and T. Hoshino, *A Field-Programmable Gate-Array System for Evolutionary Computation*, 1482 ed., 1998.

[20] M. Beckerleg and J. Collins, "Evolving Electronic Circuits for Robotic Control.," in *15th International Conference on Mechatronics and Machine Vision in Practice* Auckland, New Zealand, 2008.