A Biological Development model for the Design of Robust Multiplier

Heng Liu*, Julian F. Miller* and Andy M. Tyrrell*

* University of York, Department of Electronics, Bio-inspired Architectures Lab, Heslington, York YO10 5DD, UK
{hl142, jfm, amt}@ohm.york.ac.uk

Abstract. A biologically inspired developmental model targeted at hardware implementation (off-shelf FPGA) is proposed which exhibits extremely robust transient fault-tolerant capability. All cells in this model have identical genotype (physical structures), and only differ in internal states. In a 3x3 cell digital organism, some individuals which implement a 2-bit multiplier were discovered using evolution that have the ability to "recover" themselves from almost any kinds of transient faults. An intrinsic evolvable hardware platform based on FPGA was realized to speed up the evolution process.

1 Introduction

Living multi-cellular biological organisms exhibit several intrinsic characteristics electronic engineers earnestly long for, such as growth and fault-tolerance. Multicellular living organisms achieved these traits through millions of years of evolution by means of cells which have identical genotypes. All of them come from a single special cell (zygote): this process is called development. The entire process of development is controlled by the interaction of cells rather than by a centralized process.

The development of an embryo is determined by genes, which control where, when and how many proteins are synthesized [1]. Complex interactions between various proteins and between proteins and genes within cells and hence interactions between cells are set up by activities of genes. It is these interactions that control how the embryo develops.

Development involves cell division, the emergence of pattern, change in form, cell differentiation and growth. The model proposed in this paper contains only two of these aspects, cell division and differentiation. Since in hardware, no new resources can be created as cells are pre-formatted and their number can not be increased, "growth" is used in this report to refer to "cell division".

Cell differentiation emerges as a result of differences in gene activities which lead to the synthesis of different proteins. As development is progressive, the fate of cells becomes determined at different times. Inductive interactions by means of chemicals or proteins between cells can make cells different from each other and the response to these inductive signals depends on the state of this cell.

Built-in redundancies and error handling capabilities are the most widely used conventional fault-tolerant technologies. Redundancies can be employed either spatially or temporally. Spatial (area) redundancy can be applied using Dual Modular Redundancy or Triple Modular Redundancy, both of which are based on the majority vote of individual modules. In temporal (time) redundancy techniques, after an error output is detected, it is recomputed in an attempt to recover from the transient fault. Although time redundancy in general requires fewer resources than area redundancy, it demands error handling capability which will incontrovertibly increase the complexity of the system and its design cost. What's more, it is difficult to design such an error handling circuit which stores adequate information for recovery so that it can discover most transient faults.

Transient faults account most system failures [11], so at this stage we only concentrate on this kind of faults.

Development has been used as a bio-inspired technique in the past [6, 7, 8]. However, this paper considers a new development-inspired technique that makes use of a chemical signal which gives the system high tolerance to transient faults.

2 Development Cellular Model for Digital System

One of the most fundamental features of the development principle is the universal cell structure: each of the cells in a multi-cellular organism contains the entire genetic material, the genome.

Every cell only has direct access to the information of its four adjacent cells: No direct long term interaction between non-adjoining cells is permitted in this model.

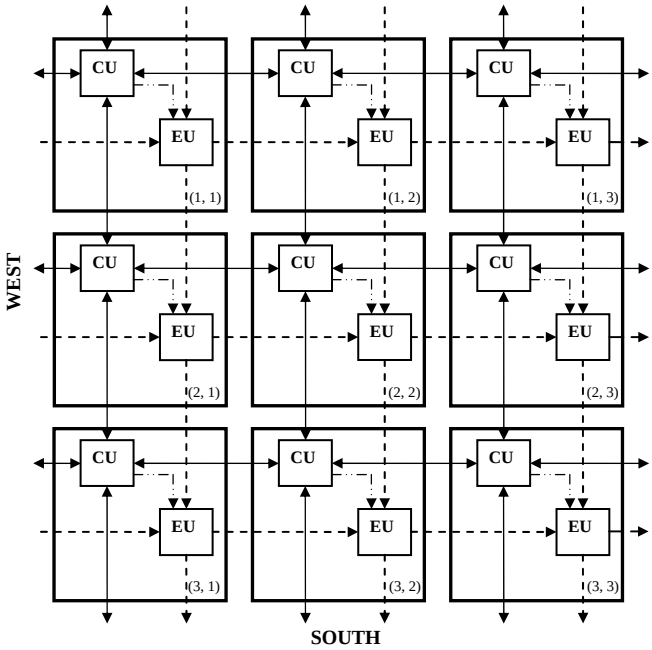
In the digital hardware model proposed here (as shown in Fig. 1), the internal structure of digital cells is shown in Fig. 2. A digital cell is composed of three main components: Control Unit (CU), Execution Unit (EU) and Chemical Diffusion module (CD).

The Control Unit (CU) has a States Register, which stores the internal states of the cell, including the cell state (type) and chemicals. Each CU connects to its 4 immediate neighbors (shown in Fig. 1) and a Next States & Chemical Generator determines its own next state/chemicals according to the current states and chemicals of the neighbors, its own state and its own chemical (illustrated in Fig. 2). The NSCG contains two components: Next States Generator (NSG) and Next Chemical Generator (NCG), both of which are built from combinational circuits.

The EU Function Selection signal (the state of a cell) is 2-bit wide: 0 means this is a dead cell, and the EU will simply propagate its west (left) inputs to its south and east neighbors, otherwise this is a living cell (it can be in any type among 1, 2 or 3), and the EU will execute and propagate its calculated output to the south and east.

The Execution Unit (EU) is the circuit incorporated to do the real calculation of the target application. The inputs to each EU come from its immediate west and north neighbors, and the state of this cell (refer to Fig. 2). Every EU also propagates its output (Executing Signals) to its immediate north and east neighbors. The Execution Unit Core (EUC) is the evolvable core logic circuit, which determines how to process the input signals in the EU.

At present only combinational applications are considered, hence the EUs are purely combinational circuits. The state and chemical signals are 2-bit and 4-bit wide respectively, while the width of Executing Signal is 3-bit. Both the internal core logical structures of EU (EUC) and CU (NSG and NCG) are determined through evolution. As a result, the genotype encodes the EU and CU internal structures. The representation of the internal structure of EU and CU are based upon Cartesian Genetic Programming [4] (CGP): a program is expressed as an indexed graph which is encoded in a linear string of integers. So the genotype just contains a list of node connections and functions.



LEGENDS:

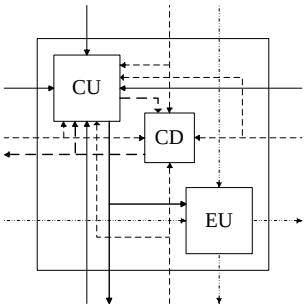
CU	Control Unit	EU	Execution Unit
⇢	EU Function Selection	⟷	States & Chemical Signals
▬	Cell border	⇢	Executing Signals

Fig. 1. Inter-connection of Cells

The Chemical Diffusion module (CD) mimics aspects of the real environment where biological cells live. In principle, CD should not be a component of a digital cell. However, this design decision makes it more convenient practically, so it is merged into the cell internal structure.

The chemical signal is introduced to transmit information between cells. Another function of the chemical is to serve as a resource which is required for a dead cell to transform to a living one.

Previous experiments [3, 5] suggest that chemicals are indispensable in order to achieve a robust solution: without chemicals, evolved individuals have poor stability and much lower fitness. The chemical diffusion regulation is the key mechanism which makes it such a significant aspect of this model: cells have a means to send long-distance messages.



LEGENDS:

CU	Control Unit	CD	Chemical Diffusion Module
EU	Execution Unit	⟶	State Signal
- - →	Chemical Signal	···→	Executing Signal

Fig. 2. Digital Cell Structure

The chemical diffusion rule employed in this work is similar to that in [3], except that there are only 4 immediate neighbors in this case. So the rule is:

$$(C_{ij})_{t+1} = \frac{1}{2}(C_{ij})_t + \frac{1}{8}\sum_{k,l\in N}(C_{kl})_t \qquad (\mathbf{1})$$

Let N denote all the 4 immediate neighbors of a cell at (i, j) with neighboring position (k, l), the chemical at this position at the new time step is given by (1). The meaning of this equation is that each cell retains half of its previous chemical and distributes the other half equally to its four adjacent cells and receives the diffused chemical from them. It is evident the rule makes sure that chemicals are conserved (apart from the unavoidable loss when the level falls below one) in the diffusion procedure.

Calculating the diffused chemical in each grow step based on the chemicals from the four immediate neighbors and the cell's own chemical value is the main task of the Chemical Diffusion module (CD) (in Fig. 2). The CD also propagates the calculated value to the four adjacent cells.

Given a genotype, the inner-structure of the cells is determined and a zygote which is located at the centre of an 'artificial environment' with x rows and y columns of

cells can be initiated and ‘duplicate’ itself. The position of the zygote was selected to speed up the growth: it takes least time for the digital organism to “cover” the entire area if the zygote is arranged in the centre. The inputs to the cells on the border of this environment are fixed to 0. Without chemicals no cells can live. This means that initially some chemicals must be injected at the position of the zygote.

Given a genotype, the growth procedure is described as follows:

1. Initialize chemical and the zygote;
2. Chemical diffusion;
3. All cells update their state simultaneously:
4. If no chemical at a position or all the cell’s four neighbors and itself are dead, then this cell’s internal program will not be executed;
5. Otherwise, it executes the program that is encoded by the genotype, to generate its next time chemical and state based on current states and chemicals;
6. If next state generated is alive, then overwrite chemical at this position with its own generated chemical;
7. Otherwise, do not touch the chemical at this position;
8. Unless stopping criterion reached go to 2.

3 2-bit Multiplier: the First Real-World Application

A 2-bit multiplier was selected as an ideal test-case for this model to verify the feasibility and applicability of this model.

The digital organism employed in this application is made up of 3x3 identical digital cells. The maturity of the digital organism means that all the cells are grown to the pattern which implements a 2-bit multiplier.

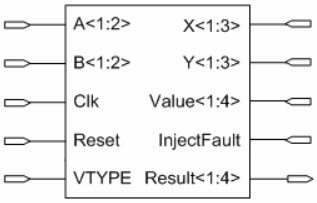


Fig. 3. Digital Organism External Interface

The external interface of the digital organism is shown in Fig. 3: Pin A, B and Result are the inputs and output of the 2-bit multiplier. Clk is the global clock signal; if the Reset pin is high, all the internal registers will be set to their initial values. All the remaining pins are dedicated to injecting transient fault(s) into the digital organism: when InjectFault pin is high, Value will be written into the chemical of cell at coordinate (X, Y) if VTYPE is low, otherwise the lowest 2-bit of Value is written into the state of the cell. Meanwhile the whole organism stops its growth process. Pin A and B

are connected to the cells (1, 1) and (2, 1) in the digital organism, while the Result is driven by the cells (2, 3) and (3, 3).

Every cell has an identical structure: Pin InjectFault, VTYPE and Value are connected to their global counterparts. If this cell is at the coordinate (X, Y) and InjectFault is active (high), the CS pin of this cell will be driven to high and the cell will overwrite its own chemical or state with Value.

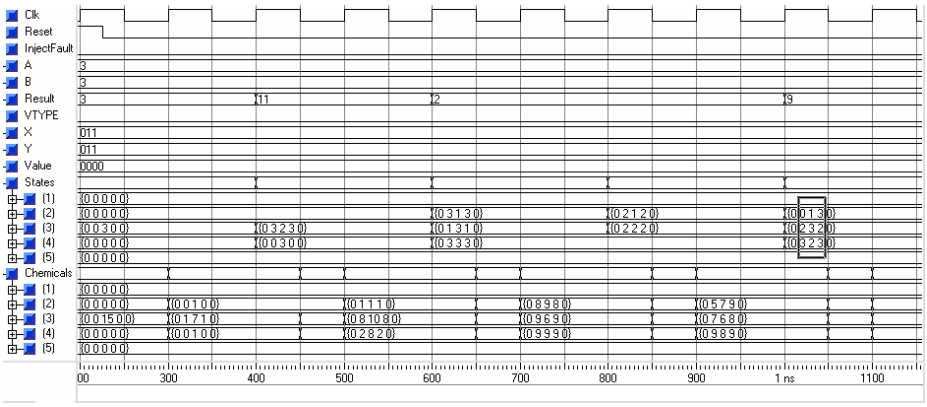


Fig. 4. Developmental Growth Procedure (the white rectangle circle the mature pattern)

A “growth step” lasts two clock cycles: at the falling edge of the first clock cycle a live cell (its state is not 0) will overwrite the existing chemical with its generated one; at the rising edge of the second clock, the chemicals diffuse according to the diffusion rule. At the rising edge of the first clock cycle in the next “growth step”, the state will be updated.

The structures of the evolvable sub-circuits were evolved in software and a robust solution found was transformed into VHDL. The FPGA implementation was synthesized by ISE 6.1i from Xilinx, downloaded into the hardware. The detail of the waveform is demonstrated in Fig. 4. It can be seen that the organism matures at 1ns, when the state pattern is identical to that obtained in the software simulation. The following experiment was carried out: enough time was allocated to let the organism grow and mature (see Fig. 4). Subsequently, two sets of transient faults were injected: the first set composed of 4 transient errors in the chemicals of cell (2, 1), (2, 2), (2, 3) and (3, 3); the other set of faults were injected into the states of cell (2, 1), (2, 3) and (3, 1). Every fault was chosen to make the corresponding value 0. The time between the injections of the two sets of transient faults was more than enough for the organism to recover completely and stabilize itself again in terms of chemicals and states of the cells (see Fig. 5 and Fig. 6).

The recovery from of the first set of transient chemical faults is illustrated in Fig. 5. At the beginning, the chemical of some of the cells are modified, and then the organism resumes growing. It recovers flawlessly at 2.4ns and the result output regains the correct value.

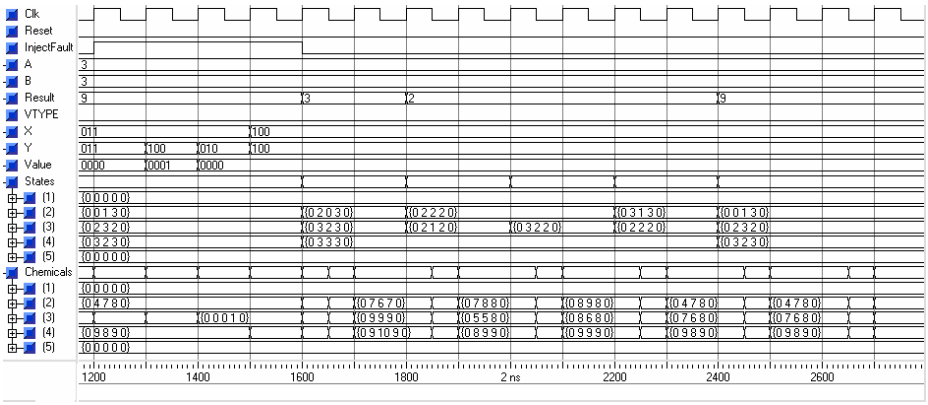


Fig. 5. Injection of the first set of faults and the recovery procedure

Fig. 6 demonstrates the recovery procedure from the second set of transient state faults. The states of the 3 selected "victim" cells are killed (state 0) at the beginning of this period. The organism again recovers completely to the correct pattern at 4ns.

The FPGA on the RC1000 board [9] connects to host PC with very limited data width: only 8-bit read and 8-bit write. So a further FPGA module "IOControler" was implemented to latch all the required inputs and feed them to the digital organism. Another function of IOControler is to cache the result output of the digital organism.

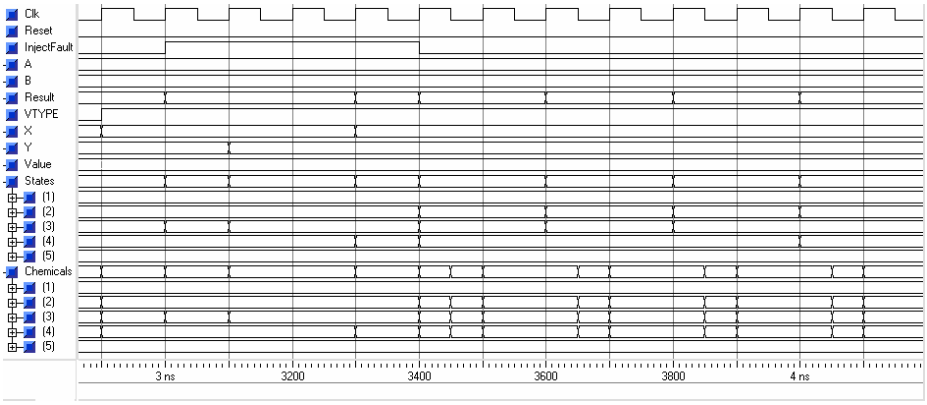


Fig. 6. Injection of the second set of faults and the recovery procedure

4 Intrinsic Evolvable Hardware (IEHW) Employing FPGA

An intrinsic evolvable hardware (IEHW) platform to accelerate the evolution progress was constructed.

The molecules (nodes in the CGP) are the most fundamental elements of the evolvable components of the model. Each evolvable sub-circuit is composed of several molecules.

The top level modules are illustrated in Fig. 7. There are 5 functional independent top-level modules which implement the IEHW as a whole.

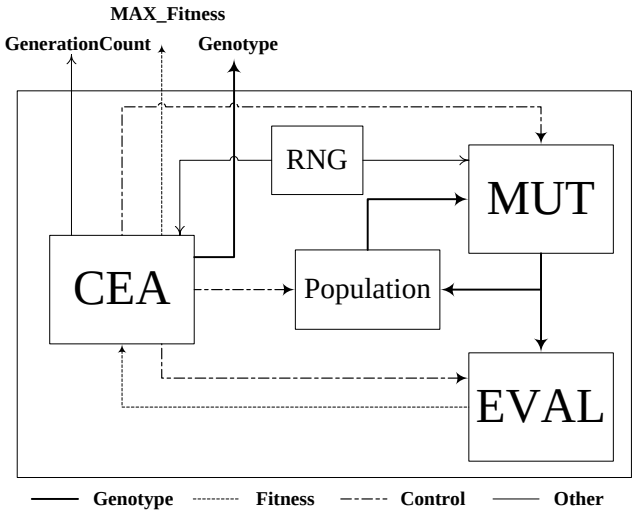


Fig. 7. Top-level Overview of the Intrinsic Evolvable Hardware Platform

The IEHW platform implementation includes 3 main outputs: MAX_Fitness, GenerationCount and Genotype. The first and the second will be updated every generation to reflect latest values, while the last output always propagates the best individual that is evolved so far. Only two inputs are required for this IEHW to function as expected: the global clock signal and reset signal. Other inputs are optional parameters, such as the seed for RNG and stop fitness.

All the genotypes of each individual are stored in the *Population* module. This is implemented in the FPGA as distributed RAM, for only one individual is manipulated at any given time.

The Controller of EA (*CEA*) supervises the entire evolution process and all the other modules. The fitness for all the individuals in the population is also stored in this module. The EA employed is an extended version of D. Levi's HereBoy Algorithm [10]: several parallel HereBoy are executed simultaneously. The CEA module is realized as a finite state machine (FSM).

RNG is a 64 bits Linear Feed-back Shift Register (LFSR), which is employed as a pseudo-Random Number Generator. If supplied, the seed of RNG will also be saved in this module.

The main function of Mutation Module (*MUT*) is to mutate a given genotype and latch the mutated genotype to be used by the *EVAL*. This module reads in the mutation rate and mutates molecules one by one until the specified mutation number is met. This module is also implemented as an FSM.

The core component of this IEHW is the Evaluation Module (*EVAL*), where the Digital Organism resides. Its main function is to evaluate the fitness of every individual. This module feeds every possible input to the 2-bit multiplier implemented by the evolved digital organism and sums up the total correct bits. Finally, the result of the subtraction of the total correct bits from the maximum possible correct bits (which is 64 in this case) is the fitness of this individual. The *EVAL* module is made up from 2 FSM: one is used to manage the digital organism and the other is in charge of feeding inputs, calculating correct output bits, the summary and the final subtraction to generate the fitness output of this module.

After reset signal is pulsed (low) for one clock cycle, all the modules, including all FSM and internal registers, are all cleared to their initial states. In this state, the IEHW will receive and latch any input parameters if provided, otherwise the default parameters are used. When the start signal is activated by the host PC, the *CEA* module will take all the responsibilities of the IEHW.

First, the population are initiated one by one, evaluated and saved into *Population*: *CEA* signals the *MUT* to mutate at the highest possible rate so all the molecules in the genotype are randomly generated, then *EVAL* evaluates it and propagates the fitness to *CEA*, finally the *CEA* saves the fitness and signal the *Population* module to store the new generated individual. These individuals make up the 0 generation.

	Mole.	NCG	NSG	EUC	EVAL	RNG	CEA	MUT	All
LUTs	14	320	196	84	5507	64	164	106	7833
FFs					360	3	115	86	926

Table. 1. Synthesis Report

Second, after the initial population is ready, the main loop of evolution process begins: in each generation, the *CEA* selects each of the individuals in the *Population* and feeds it to the MUT. The mutated genotype is then evaluated by the *EVAl* module, and the fitness is again propagated back to *CEA*. If the mutated one (offspring) is better than the original one (parent), or with a probability p_r it substitutes the parent, which means the *CEA* asks the *Population* to store the mutated genotype; otherwise the content of *Population* module is untouched. After all the individuals have undertaken this procedure, a new generation is created. The evolution will continue to process the next generation unless the stop criterion, the specified fitness has been reached, is fulfilled. So no elitism is deployed in the IEHW.

When the main loop of the evolution process terminates, the best individual evolved is presented through the *Genotype* pin, while its fitness and the generation where this evolution stops are propagated out via MAX_Fitness and GenerationCount respectively.

Table. 1 demonstrates how much hardware resources the various modules consume after the synthesis. The entire design occupies 31.9% LUTs, 3.77% Flip Flops and 58.6% IOBs totally.

5 Conclusion and Future Work

It was demonstrated that the biological development model proposed in this work can be applied to real world application and the solution discovered through evolution exhibits the intrinsic highly fault-tolerance feature similar to its living organism counterpart: the best solution found can virtually tolerate any transient damages.

Although this model may consume more resources for the 2-bit multiplier if compared with conventional majority voting systems, it does not require any voter systems and is not dependent on any single resource, so no single point fault. In the meantime, as this is a development model, we can apply it to more complex systems without fundamental modification.

In future, the module will be extended to explore more possibilities, such as making full use of chemical signals when dealing with state signals and unconstrained growth world. In addition, the hardware implementation will receive more improvement, including incorporating adaptive mutation. With the IEHW, we can also carry out more researches about the impacts of different parameters to the evolution outcome.

References

1. Lewis Wolpert: Principles of Development 2nd (2002), Chapter 1, Oxford University Press, c2002.
2. H. Ball and F. Hardy, "Effects and detection of intermittent failures in digital systems" 1969 FJCC, AFIPS Conf. Proc., vol. 35, pp.329-335
3. Julian F. Miller, "Evolving developmental programs for adaptation, morphogenesis, and self-repair" (2003), Seventh European Conference on Artificial Life, Lecture Notes in Artificial Life, Vol. 2801, pp. 256-265
4. J. Miller and P. Thomson: Cartesian genetic programming. Lecture Notes in Computer Science, Vol. 1802, pp. 121-132, Poli, R., Banzhaf, W., Langdon, W.B., Miller, J. F., Nordin, P., Fogarty, T.C., (Eds.)
5. Julian F. Miller, "Evolving a self-repairing, self-regulating, French flag organism", GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004. Proceedings, Part I
6. Ortega, C., Mange, D., Smith, S.L. and Tyrrell, A.M. 'Embryonics: A Bio-Inspired Cellular Architecture with Fault-Tolerant Properties' Journal of Genetic Programming and Evolvable Machines, Vol 1, No 3, pp 187-215, July 2000.
7. Jackson, A.H. and Tyrrell, A.M. 'Implementing Asynchronous Embryonic Circuits using AARDVArc', 4th NASA Workshop on Evolvable Hardware, Washington, pp 231-240, July 2002.
8. Canham, R. and Tyrrell, A.M. 'An Embryonic Array with Improved Efficiency and Fault Tolerance', 5th NASA Conference on Evolvable Hardware, Chicago, pp 265-272, July 2003.
9. http://www.celoxica.com/
10. D. Levi, "HereBoy: A Fast Evolutionary Algorithm", Proceedings of the 2nd NASA/DoD Evolvable Hardware Workshop, IEEE Computer Society, Los Alamitos, Ca, July 2000
11. H. Ball and F. Hardy, "Effects and detection of intermittent failures in digital systems" 1969 FJCC, AFIPS Conf. Proc., vol. 35, pp.329-335