A Smart Configuration Mechanism for Ontogenetic Hardware

Nantian Wang, Yue Li, Yanling Qian, Xiubin Liu
Science and Technology on Integrated Logistics Support Laboratory
National University of Defense Technology
Changsha, Hunan, P.R. China
E-mail: wangnant@126.com

Abstract—Ontogenetic hardware, as a branch of bio-inspired hardware, mainly refers that the hardware can develop based on some basic blocks and genetic material similar to the development of cellular organism. An ontogenetic artificial organism is composed of organelles, each of which consists of a cluster of molecules. The construction of organelle from molecules and the replication of organelle are basic problems in the organism's developing. A smart configuration mechanism providing good supporting abilities of organelles' construction, differentiation, replication and etc. is presented. A hardware implementation of the configuration mechanism is also provided.

keywords - Bio-inspired Hardware; POE Model; Self-organization; Gene, Replication; Differentiation

I. INTRODUCTION

Nature has always been amazing, such as the self-repair ability of cellular systems and the learning ability of humans. It is always a dream of engineers to implement bio-inspired systems. Artificial Neural Networks (ANNs) [1] and evolution system [2] are some paradigms of bio-inspired systems, whose architectures or emergent behaviors resemble the structure or behavior of bio-systems [3].

Analogous to nature, bio-inspired hardware systems can be partitioned into three axes, which are phylogeny, ontogeny and epigenesist, as the POE model shown [4] [5]. The phylogenetic (P) axis of hardware systems is inspired by the processes involved in the evolution of a species. The evolvable hardware [6] is considered to be a phylogenetic hardware. The epigenetic (E) axis mainly concerns the learning from environment of an organism. The ANNs and the artificial immune system [7] are the representatives of the epigenetic process. The ontogenetic (O) axis concerns the developing of a signal multi-cellular organism from its own genetic material and basic blocks, and it is the organism's development. Ontogenetic hardware mainly involves hardware implementing self-replicating and self-healing [8] based on cellular division and cellular differentiation [9]. An ontogenetic hardware may also be considered as a POE hardware, if its genetic material is gotten in a phylogenetic process, e.g. evolution arithmetic based, and endowed with learning ability, e.g. an ANNs. POEtic [9] [10] and ubichip [11] are some representatives of POE hardware. In this paper, researchers focus the concern on the ontogenetic axis.

Generally, an ontogenetic hardware is a multi-levels programmable hardware. The terms molecule, organelle, cell and organism in biology has been borrowed to ontogenetic hardware. For clarifying these terms, some analogies between ontogenetic hardware and reconfigurable logic are line-out. Similar to the composing of biologic cellular organism, a four-level ontogenetic organism was presented in [12] as shown in Figure 1. A molecule, which is considered to be equivalent to a logic cell in a programmable logic cell array, such as a slice in a FPGA, is the basic programmable unit whose function is defined by its configuration. An organelle is a Function Unit (FU) composed of a set of molecules nearby where runs a special task in a cell. A cell includes several FUs, while not every FU is an organelle, and it may be specially designed block. A set of cells forms the organism which is equivalent to the whole reconfigurable device as an FPGA.

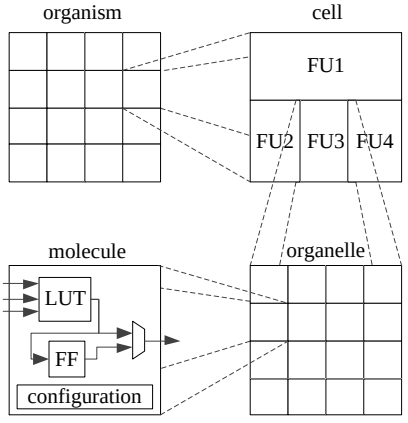

Figure 1. Hierarchical decomposition of an artificial ontogenetic organism

During the development of ontogenetic organism, there are several basic but important problems. The first one is how to organize the molecules to form the organelles on a designed ontogenetic hardware, since the molecule is the basic programmable unit of the organism. In POEtic [4] [13], an organelle has to be preconfigured and cannot be changed during it works. Theseus mechanism [12] used in the ubichip

from PERPLEXUS project [14] provides dynamic building ability during working, while it cannot build organelles concurrently and independently. It also shows incapability in decomposing of a unique formed organelle for molecules' reusing in self-healing.

The following problem is how the constructed organelles differentiate, in other words, how to define the unique function of each organelle. Since the function of a molecule is defined by its configuration, the problem changes into how to load the configuration of the molecules into the organelle. If researchers view ontogenetic hardware as reconfigurable hardware, the general configuration method of reconfigurable hardware based on BPI, SPI and JTAG are some solutions [15]. Configuration based on these methods must be predefined, and the configuration path is difficult to change after the hardware had been produced which means low smart. It seems they cannot satisfy the configuration requirements of efficiency, dynamic and flexible in ontogenetic hardware.

Another problem is how to implement the cells' replication for development. The essential of this problem is how to copy the configurations of molecules to other molecules. In [12], replication solutions on both cellular level and molecular level are presented. The cellular level solution shows good idea. However, the molecular level replication solution can not realize multi-cell's parallel replication and the functionality of the organelle must be paused during the replication.

In this paper, researchers present a smart configuration mechanism mainly aiming at solving the problems above. The mechanism is introduced in detail in Section 2. A hardware implementation of the mechanism is presented in Section 3. In the final Section 4, conclusions of this paper are made.

II. SMART CONFIGURATION MECHANISM

The smart configuration mechanism focuses on the construction, differentiation and replication of organelles, which are three of the key steps during the developing process of an organism, and works on molecule level and organelle level in Figure 1. For better understanding, researchers first introduce the gene hierarchy and structure of molecule based on the smart configuration mechanism.

A. Gene Hierarchy and Structure of molecule

Each molecule includes a structural gene, several functional genes and an operating gene. The structural gene defines how the molecule is used in an organelle. The functional genes combined define the unique function of the molecule. The operation gene is an operator, which is used for controlling the structural gene and the functional genes. The genes of a molecule M can be denoted as (1).

The $gf(M)$, whose first bit must be 1, refers the structural gene of M. The $g_1(M)$, $g_2(M)$,…, $g_n(M)$ refer the n functional genes. The $fc(M,q)$ refers the operating gene. The variable q is just used for distinguishing different operators.

If an organelle O is composed of p molecules, the genome of the organelle is represented as (2).

$$Gm(M) = \{gf(M), g_1(M), g_2(M), \ldots, g_n(M), gc(M,q)\} \quad (1)$$

$$Go(O) = \{Gm(m_1), Gm(m_2), Gm(m_p)\} \quad (2)$$

In molecule, the structural gene stores in a configuration interface selecting shift register called Path and a flag register called F. The operating gene stores in an instruction shift register called IR. The functional genes store in some registers. Another shift register call DR is also included in the molecule as a data buffer used during the configuration process.

Each molecule includes several Configuration Interfaces (CIFs), which are used for constructing a configuration network. Each CIF includes several inputs and outputs. In the network, a molecule's CIF connects with another molecule's CIF, where input connects to output. A configuration network with typical mesh topology is shown as Figure 2. The network provides the medium for the configurations' transmission, from one molecule to another molecule. In this section, all the examples are based on the structure as shown in Figure 2, if not specially mentioned.

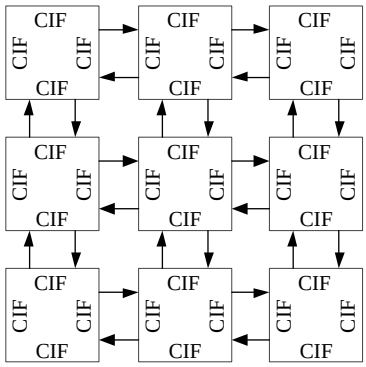

Figure 2. A mesh network of configuration interface

B. Construction of Organelle

The configuration of ontogenetic hardware is different from the whole device configuration of general reconfiguration hardware. The idea behind ontogenetic hardware is to only configure a part of the hardware, letting the organism to grow on the electronic substrate. In order to do this, an organelle is constructed to define the part will be configured. The Theseus mechanism [12] provides a solution for the construction of organelle, and researchers adapt its basic idea of building path to build the organelle. Figure 3 shows a molecules array within three constructed organelles based on the configuration mechanism. The mechanism goes as follows.

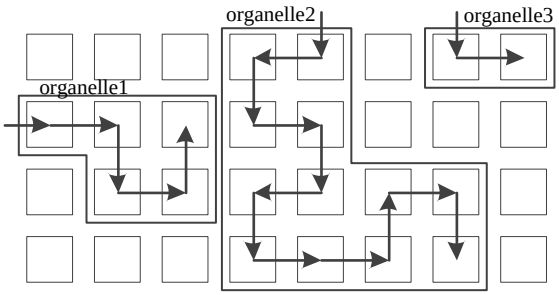

Figure 3. A molecular array includes three constructed organelles.

During the construction of an organelle, the structural gene will be edited. At the beginning of the construction, the molecules will be used to construct are in idle state, and the structural gene can be serially loaded in a molecule through any of its CIFs. After the first bit 1 of structural gene has been loaded, the used CIF, denoted by CIF#, is locked and the genes can only be loaded through CIF#, unless the molecule is released by resetting or organelle's decomposing.

When a molecule is idle, its F equal to 0, the loaded configurations is shifted into Path. After the first bit of structural gene has been shifted into F, F equals to 1, and the CIF indicated by Path, denoted by CIF*, is activated. Then a configuration path from CIF# to CIF* through IR is built. Figure 4 shows the simplified configuration flow in a molecule during the organelle's construction.

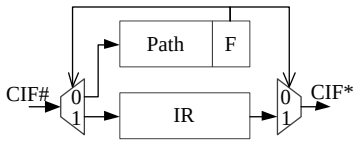

Figure 4. Simplified configuration flow in a molecule during the construction

After a molecule is absorbed in a building organelle, another molecule with a CIF connecting to CIF* will be absorbed in the organelle. Based on this mechanism, organelle will be absorbed into the organelle one by one, until the final molecule is absorbed whose CIF* is CIF#. The first organelle's CIF# is called as Input CIF (ICIF).

If researchers want to build an organelle including p molecules, functional genes $gf(m_1)$, $gf(m_2)$,…, $gf(m_p)$ have to be loaded in sequence via ICIF, and then a following operators $gc(m_p)$, $gc(m_{p-1})$,…, $gc(m_1)$ are loaded to shift the functional genes from IRs into Paths and Fs.

The genes used in sequence during the construction are denoted as (3) and called a gene sequence. In the following, the "[]" will be used to denote a sequence of genes or molecules similar to (3).

$$[gf(m_1), gf(m_2), ..., gf(m_p), \\ gc(m_p, 0), gc(m_{p-1}, 0), ..., gc(m_1, 0)]. \quad (3)$$

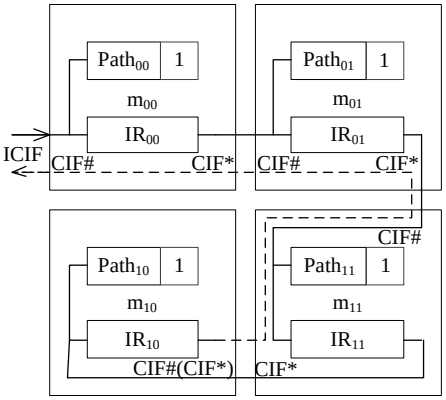

Figure 5. A constructed organelle and its gene flow during construction

The mechanism introduced above is similar with the Theseus mechanism to some extent. However, the mechanism also provides a back flow path during the building process, which will be used for genes' reading back, fault check and etc. Figure 5 shows a constructed organelle consisting of $[m_{00}, m_{01}, m_{11}, m_{10}]$ and the gene flow during construction.

C. Differentiation of Organelle

After an organelle is built, its functional genes have to be loaded into the molecules to define the organelle's function, called differentiation [9]. The basic idea is loading each molecule one functional gene a time, and repeating n times to load all the functional genes.

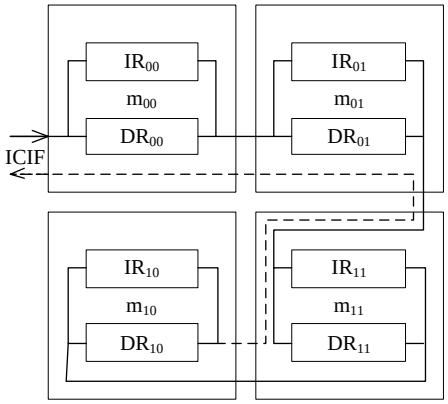

Figure 6. Transmission path in a constructed organelle

After the organelle has been constructed, a transmission path for operating genes has been generated, which is similar to the boundary-scan [16] to some extent. The transmission path in an organelle as Figure 5 is shown as Figure 6. The data from ICIF can move into IR or DR as need. The differentiation process goes as follow.

(A) A sequence of functional genes as (4) of $[m_{00}, m_{01}, m_{11}, m_{10}]$ are shift into the DRs through ICIF.

$$[g_{a1}(m_{10}),\ g_{b1}(m_{11}),\ g_{c1}(m_{01}),\ g_{d1}(m_{00})] \quad (4)$$

$$[gc(m_{10},1),\ gc(m_{11},1),\ gc(m_{01},1),\ gc(m_{00},1)] \quad (5)$$

(B) A group of operating genes followed as (5) are shifted into IRs.

Then a writing operation is executed to load the functional genes in the DRs as (4) into the molecules.

(C) Repeating step (A) and step (B) n times until the genes as (6) are loaded.

During the above steps, the set of subscripts A_i defined as (7) is an any permutation of integers from 1 to n.

$$[g_{an}(m_{10}),\ g_{bn}(m_{11}),\ g_{cn}(m_{01}),\ g_{dn}(m_{00})] \quad (6)$$

$$A_i = \{ai \mid i = 1,2,\ldots,n\} = \{1,2,\ldots,n\} \quad (7)$$

$$a_i = i\,. \quad (8)$$

However, in order to make the process easy and simple, researchers usually define as (8).

The same situations happen on the Bi, Ci and Di in which

$$B_i = \{bi \mid i = 1,2,\ldots,n\} = \{1,2,\ldots,n\}\,, \quad (9)$$

$$C_i = \{ci \mid i = 1,2,\ldots,n\} = \{1,2,\ldots,n\}\,, \quad (10)$$

$$D_i = \{di \mid i = 1,2,\ldots,n\} = \{1,2,\ldots,n\}\,. \quad (11)$$

$$\begin{aligned}[g_1(m_{10}),\ & g_1(m_{11}),\ g_1(m_{01}),\ g_1(m_{00}),\\ & gc(m_{10},1),\ gc(m_{11},1),\ gc(m_{01},1),\ gc(m_{00},1),\\ & g_2(m_{10}),\ g_2(m_{11}),\ g_2(m_{01}),\ g_2(m_{00}),\\ & gc(m_{10},2),\ gc(m_{11},2),\ gc(m_{01},2),\ gc(m_{00},2),\\ & \ldots,\\ & g_n(m_{10}),\ g_n(m_{11}),\ g_n(m_{01}),\ g_n(m_{00}),\\ & gc(m_{10},n),\ gc(m_{11},n),\ gc(m_{01},n),\ gc(m_{00},n)]\end{aligned} \quad (12)$$

If researchers want to finish the differentiation of the organelle as shown in Figure 5, researchers can use the gene sequence as (12).

D. Replication of Organelle

Replication of organelle refers replicating organelle's function, and actually replicating the functional genes. The replication process includes the following steps.

(A) Retrieving the structure. A sequence of operators for getting the structural genes is shifted into the IRs to load the structural genes into DRs. With a number of shifting DR operations, the structural genes are shifted out through the back flow path. If researchers want to retrieve the structure of organelle as shown in Figure 5, operators as (13) are shifted in and then the organelle's structural genes are shifted out along the dashed arrow shown path. Then, the organelle's structure can be retrieved from the shifted out structural genes.

$$[gc(m_{10},n+1),\ gc(m_{11},n+1), gc(m_{01},n+1),\ gc(m_{00},n+1)] \quad (13)$$

(B) Decomposing the mother organelle. Running a decomposing organelle operation to decompose the mother organelle and release the molecules. All the molecules' structural genes are cleared, while the functional genes are still kept in the molecules.

(C) Building a child organelle. The child organelle includes the p molecules belonging to the mother organelle and some other p molecules. A child organelle of Figure 5 showing organelle is shown in Figure 7.

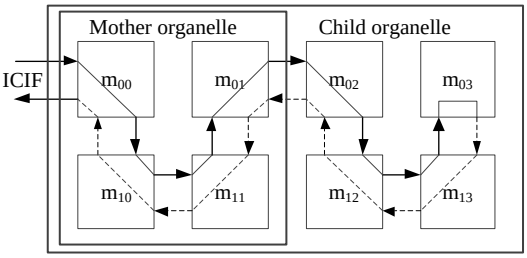

Figure 7. A constructed child organelle from a mother organelle and the child organelle's configuration path

(D) Getting the mother organelle's functional genes. The reading functional gene operators as (14) are shifted into IRs to load the first group of functional genes as (15) in DRs.

$$\begin{aligned}[gc(m_{01},n+2),\ & gc(m_{11},n+2),\\ & gc(m_{10},n+2),\ gc(m_{00},n+2)]\end{aligned} \quad (14)$$

$$\left[g_1(m_{01}),\ g_1(m_{11}),\ g_1(m_{10}),\ g_1(m_{00})\right] \quad (15)$$

(E) Replicating the genes. The gene sequence (15) is shifted into $[m_{03}, m_{13}, m_{12}, m_{02}]$ and then loaded using the method shown in the step (B) in part Differentiation of Organelle.

(F) Complicating the Replication. Repeating step (D) and step (E) n times until all the functional genes are copied.

E. More supporting abilities

The smart configuration mechanism not only supports the construction, the differentiation and the replication of organelle, but also has some further abilities.

(A) Molecule backup support. After an organelle is constructed, not all the molecules have to be differentiated. Some molecules can keep idle state if the functional genes are not loaded. These molecules can be used for self-healing when fault happens.

(B) Part replication support. Replication based on the mechanism can only replicate some of the functional genes quickly just by setting some DRs bypassed. This character is useful in self-healing.

(C) Fault isolating and self-healing support. If the fault not happens on the genes, the fault molecules can be functional isolated and replaced based on the redundant backup molecule(s) and part replication ability.

(D) Parallel configuration support. The constructions of organelles are parallel, as long as a molecule is not wanted to be constructed into two organelles. If this happen, an organelle will fault to be built, and the fault can be checked based on the genes' reading back. After organelles are constructed, the operations in different organelles will be completely in parallel.

III. HARDWARE IMPLEMENTATION

Based on the smart configuration mechanism has been introduced above, a hardware implementation of the mechanism is presented as follows.

We divide a molecule into two parts. The first part includes the reconfigurable logics, which implements the differentiated unique function, and their configuration or the functional genes. Researchers call the first part as Reconfigurable Block (RCB). The second part includes the others except RCB, including structural gene, operating gene and some others, and researchers call it the Configuration Block (CB).

As the concern is not function implementation but configuration process, researchers put out focus on the CB, where the smart configuration mechanism will be executed.

A. Structure of CB

In this implementation, each CB including a Configuration Interface (CFIF) and seven Communication Interface (CIF), which are CIF1, CIF2, CIF3, CIF4, CIF5, CIF6 and CIF7. CFIF is used for configuring the RCB. CIF is used for building the communication network or used as ICIF. The unused CIF(s) can be deleted or be assigned with zero inputs.

Each CIF includes 6 lines, they are 2-bit mode input Mi used for control the working mode of itself, 2-bit mode output Mo connected to the Mi of next CB in the built organelle, 1-bit data input Di and 1-bit data output Do. Signal belonging to CIFi is indexed by a subscript i, such as Mi4, Mo5, Di6 or Do7. The CFIF including a y-bit data bus DB, an x-bit address and control bus AB and an interrupt request signal Int. Figure 8 shows the structure of a CB.

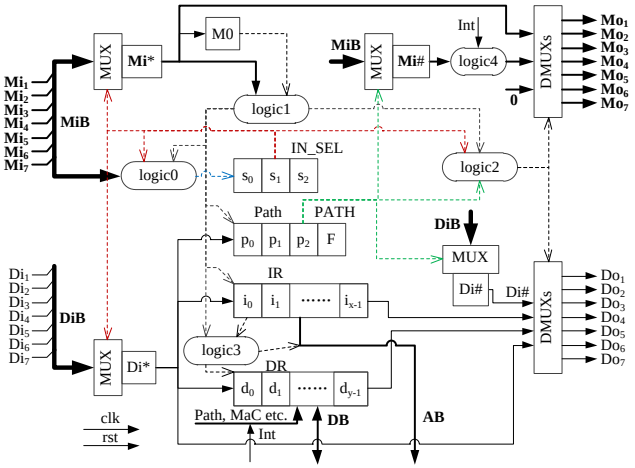

Figure 8. Structure of a configuration block

In the CB, there are a 2-bit mode input buffer Mi*, a 1-bit data input buffer Di*, a 2-bit reserve mode input buffer Mi#, a 1-bit reserve data input buffer Di#, a 1-bit mode buffer M0, a 3-bit input selecting CIF buffer IN_SEL, a 1-bif F, a 3-bit Path, a x-bit IR, a y-bit DR and some combine logic circuit. A physical address MAC is also included in a CB. The working principle will be introduced in the following Simulation part.

B. Simulation

In order to verify the configuration mechanism, a molecule array including two ICIFs and nine molecules as shown in Figure 9 was built by Verilog HDL and simulated in the ISIM embedded in Xilinx ISE13.2.

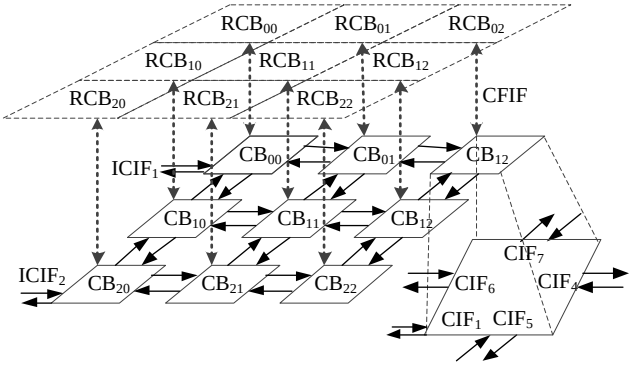

Figure 9. A framework of molecule array with the configuration blocks.

The mesh network is built using CIF4, CIF5, CIF6 and CIF7. The ICIFs use the ICF1s. As the concern is the CB, researchers just define the RCB as two 8-bit registers, called reg0 and reg1, to store the functional genes. In order to configure the functional genes, researchers define y equaling to 8 and x equaling to 3, which means DR is 8-bit width and IR is 3-bit width.

In the simulation, the operations are shown as TABLE I. Step 2 is running in parallel with step 1.

Before showing the simulation result, researchers give a general introduction of the working principle of the CB by taking the CB00 as example. (1) Once a shifting IR mode

(mode is defined by Mi* and M0) is presented in Mi of ICIF1, serial number of the CIF1 (ICIF using CIF1) 1 is locked into IN_SEL, making Mi*/Di* can only receive data form Mi1/Di1 belonging to CIF1 (CIF#). Functional gene is serially shifted into PATH (A combination of Path and F), and then IR after the gene's first bit 1 has been shifted into F. (2) After structural gene has been loaded, Path equals to 5, so CIF5 is selected as CIF*, paths (Mi1 to Mo5, Di1 to Do5) and reserve paths (Mi5 to Mo1, Di5 to Do1) is built, and other five Mos are all sent to 0. (3) The functional gene is serial shifted into DR as a shift DR mode is presented at Mi1 (Mi*) and pass to Mo5. (4) When a writing sub-mode (sub-mode is defined by Mi* and lower two bits of IR) is set, writing enable signal in AB is activated to write data in DR into RCB through CFIF. If a reading sub-mode is set, the function gene, the structural gene or the MAC and Int, selected by AB, can be loaded into the DR. These are used during the construction checking and/or the replication of organelle. (5) A bypass DR sub-mode can be set to bypass the DR. The Int signal is melted into the Mo1 as an interrupt request and indexed by a reading sub-mode. (6) A release mode can be set to release the molecule from an organelle (decomposing the organelle), which will clear the IN_SEL, PATH, IR and M0.

TABLE I. OPERATIONS DURING SIMULATION

Steps		Operations
1	1.1	Building an organelle including m_{00}, m_{10}, m_{11}, m_{01}, m_{20}.
	1.2	Loading A0, B0, C0, D0, E0 into reg_0s of m_{00}, m_{10}, m_{11}, m_{01}, m_{20}.
	1.3	Loading A1, B1, C1, D1, E1 into reg_1s of m_{00}, m_{10}, m_{11}, m_{01}, m_{20}.
2	2.1	Building an organelle only including m_{20}.
	2.2	Loading 50 into reg_0 of m_{20}.
	2.3	Loading 51 into reg_1 of m_{20}.
3	3.1	Decomposing the organelles built in step 1.1 and step 2.1.
	3.2	Building a new organelle including m_{20}, m_{21}, m_{11}, m_{12}.
	3.3	Replicating the functional genes of m_{11} to m_{12}, and replicating the functional genes of m_{20} to m_{11} to replace its old genes.
	3.4	Decomposing the organelle built in step 3.2.

The simulation result is shown as Figure 10. The values of F and Path are shown in binary and the values of functional gene reg0 and reg1 are shown in hexadecimal. From Figure 10, researchers can conclude that all the genes are successful loaded and replicated.

At the first line showing 0.465us, the first organelle was constructed because each F of molecule belonging to the organelle equaled to 1 and the Path not equaled to 000. It can be found that the organelle is built in the sequence of m00, m10, m11, m01 and m20 from the structural genes' change sequence over time. At the second and third lines showing times, the reg0s and reg1s had been updated in step 1.2 and step 1.3.

Step 2 finished at soon after the time shown by line 1. It shows the configuration was running in parallel with step 1.

Step 3.1 referring decomposition starts at the fourth line showing time. Then the new organelle was built in the sequence of m20, m21, m11 and m12. At the fifth line showing time, the replication complicated. Value of m11's reg1 was replaced by 51 coming from m20's reg1, and the older value C1 in m11's reg1 was moved into m12's reg1. At the sixth line showing time, the new organelle decomposed, and all the Fs and Paths were cleared.

We also implemented the design on a Spartan 3 FPGA with a speed of class 4, and the timing report shows the clock frequency can easily arrive 80MHz.

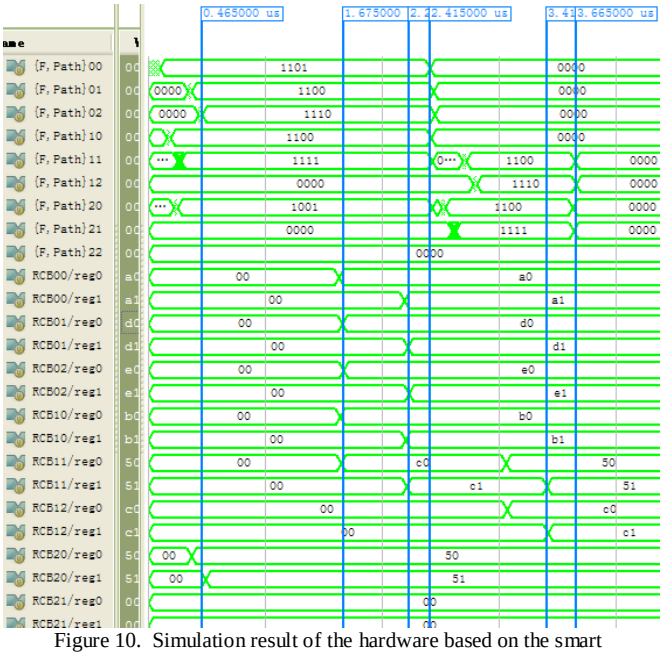

Figure 10. Simulation result of the hardware based on the smart configuration mechanism.

C. Discussion

The register consumption in a CB, not including the functional genes, is

$$Rc = 13 + x + y. \tag{16}$$

If a molecule includes n functional genes with total N bits, and each functional gene is not more than y bits, the register overhead is

$$Ro = (Rc / N) * 100\%. \tag{17}$$

And the restrictions are

$$\begin{cases} n \le 2^{x-2} \\ n * y \le N \end{cases} \tag{18}$$

Based on the configuration mechanism, if researchers want to construct an organelle including p molecules, the minimal clock consumption of the process is

$$cc_c = 6 * p + (x * p + y * p + 1) * n. \tag{19}$$

If researchers want to replicate the organelle built above, the minimal clock consumption of the replication is

$$cc_r = 1+12*p+2*x*p+(2+y*p+2*x*p)*n. \quad (20)$$

Once the reconfigurable logics (RCB) is defined, the N will not change. If researchers want to build an organelle to run a given function, the molecule need is almost fixed. On this situation, the best wish is getting fastest configuration speed with lest resource consumption. If researchers define p equal to 5, TABLE II shows the results.

From TABLE II, researchers can find out that the wish is not easy to satisfy. Researchers must select a suitable x to get a balance. The underline marked $x=5$ when $N=80$ and $x=6$ when $N=240$ may be good selections. Of course, different x may be selected in different situation. When $N=240$, the overhead is about 12% that is a little more than the 10% [12] in the ubichip based on Theseus mechanism.

TABLE II. TIME AND RESOURCE CONSUMPTION

N	x	Rc	Ro	cc_c	cc_r
80	2	95	1.18	441	508
	3	56	0.70	462	560
	4	37	0.46	514	674
	5	28	0.35	638	932
	6	24	0.30	926	1518
	7	23	0.29	1662	2920
	8	23	0.29	3294	6034
240	2	335	1.04	1644	1708
	3	176	0.55	1662	1760
	4	97	0.30	1714	1874
	5	58	0.18	1838	2132
	6	39	0.12	2126	2718
	7	30	0.09	2782	4040
	8	26	0.08	4254	6994
	9	25	0.08	7838	13852
	10	25	0.08	15646	28838

Comparing the two selected result, the register overhead of the later is much less. The time consumption of configuration is about 27us with a clock frequency at 80MHz when $N=240$. If the time consumption is acceptable, a larger molecule can be designed to reduce hardware overhead. What's more, researchers may use one CB to configure several RCBs to reduce hardware overhead.

IV. CONCLUSIONS

In this paper, researchers present a smart configuration mechanism and its hardware implication for the design systems. The mechanism shows similar overhead to Theseus mechanism [12], but better configuration supporting abilities than it. Based on the proposed mechanism, the configuration process shows the following abilities. Any molecules nearby can be constructed serially as an organelle and be configured easily. Part or whole of an organelle can be easily replicated based on this mechanism. The operations between different organelles are independent and run in parallel. The destruction of organelle is supported for molecules' replication and reusing. The functional part does not have to stop when configuring. Fault-check and interrupt are also supported because of the existence of back flow path.

However, there are still some issues to develop. For example, this configuration mechanism supports organelle replication and differentiation, but cannot finish them without higher level replication and differentiation mechanisms, which are needed to extend the works.

REFERENCES

[1] N. Gupta, "Artificial Neural Network," Network and Complex Systems, vol. 3, pp. 24-28, 2013.

[2] H. Mo and L. Meng, "Research on evolution hardware design based on memetic algorithm," in 2013 IEEE Workshop on Memetic Computing, 2013, pp. 32-36.

[3] C. G. Langton, "Artificial life: an overview " Complex adaptive systems. MIT Press, 1995.

[4] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," IEEE Transactions on Evolutionary Computation, vol. 1, pp. 83-97, 1997.

[5] Y. Thoma, G. Tempesti, and E. Sanchez, "POEtic: An Electronic Tissue for Bio-inspired Cellular Applications," Biosystem, vol. 76, pp. 191-200, 2004.

[6] K. Raman and A. Wagner, "The evolvability of programmable hardware," Journal of the Royal Society Interface, vol. 8, pp. 269-281, 2011.

[7] Z. Xuegong, G. Dragffy, A. G. Pipe, and Q. M. Zhu, "Artificial Innate Immune System:an Instant Defence Layer of Embryonics," in the 3rd International Conference on Artificial Immune Systems (ICARIS 2004), Catania, Italy, 2004, pp. 302-315.

[8] J. Xu, Y. Dou, and Q. Lv, "A Bio-Inspired Fault-tolerant Hardware System Supporting Hierarchical Self-healing " ELEKTRONIKA IR ELEKTROTECHNIKA, vol. 4, pp. 103-106, 2012.

[9] A. M. Tyrrell, E. Sanchez, D. Floreano, G. Tempesti, D. Mange, J.-M. Moreno, J. Rosenberg, and A. E. Villa, "Poetic tissue: An integrated architecture for bio-inspired hardware," in Evolvable Systems: From Biology to Hardware, ed: Springer, 2003, pp. 129-140.

[10] J. M. Moreno, Y. Thoma, and E. Sanchez, "POEtic: A hardware prototyping platform with bio-inspired capabilities," in Proceedings of the International Conference Mixed Design of Integrated Circuits and Systems, Lodz, 2006, pp. 363-368.

[11] A. Upegui, Y. Thoma, H. F. Satizabal, F. Mondada, P. Retornaz, Y. Graf, A. Perez-Uribe, and E. Sanchez, "Ubichip, Ubidule, and MarXbot: A Hardware Platform for the Simulation of Complex Systems," in Evolvable Systems: From Biology to Hardware. vol. 6274, G. Tempesti, A. M. Tyrrell, and J. F. Miller, Eds., ed Berlin: Springer-Verlag Berlin, 2010, pp. 286-298.

[12] Y. Thoma, A. Upegui, A. Perez-Uribe, and E. Sanchez, "Self-replication mechanism by means of self-reconfiguration," ARCS 2007, 2008.

[13] W. Barker, D. M. Halliday, Y. Thoma, E. Sanchez, G. Tempesti, and A. M. Tyrrell, "Fault tolerance using dynamic reconfiguration on the POEtic tissue," Ieee Transactions on Evolutionary Computation, vol. 11, pp. 666-684, Oct 2007.

[14] E. Sanchez, A. Perez-Uribe, A. Upegui, and Y. Thoma, "PERPLEXUS: Pervasive Computing Framework for Modeling Complex Virtually-Unbounded Systems," Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems, pp. 587-591, 2007.

[15] Xilinx. Virtex-6 FPGA Configuration User Guide (v3.4) [Online].

[16] T. T. S. C. o. t. I. C. Society, "IEEE Standard Test Access Port and Boundary-Scan Architecture," vol. 1149.1-2001, ed, 2001.