High-Level Synthesis by Dynamic Ant

Rachaporn Keinprasit[1] and Prabhas Chongstitvatana[2]
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Pathumwan, Bangkok 10330, Thailand
Phone: (66-2) 218-6982 Fax: (66-2) 218-6955
E-mail: p41rkp@cp.eng.chula.ac.th[1], prabhas@chula.ac.th[2]

Abstract: In this paper a new algorithm called Dynamic Ant is introduced. It was a combination of Ant Colony Optimization techniques and Dynamic Niche Sharing scheme. The interesting point of this algorithm is that it is simple to be implemented and could be well matched with existing design algorithms by adding the heuristic weights to speed up the algorithm. The algorithm uses the problem state structure as in reinforcement-learning algorithm, but the storage explosion is prevented by mean of the pheromone trail. This algorithm was investigated by data path design problem of High-Level Synthesis of which has a large number of design steps and design techniques.
Key words: Ant algorithms, High-Level Synthesis, Data Path Design, Genetic Algorithms, Dynamic Niche Sharing

1. Introduction

In hardware synthesis usually the algorithm or behavior of the target system will be described in high-level language as VHDL or Verilog. This description will be transformed to control data flow graph (CDFG). Then a batch of algorithms will transfer this CDFG in to register-transfer level, consisted of data path and control unit. This process, as explain in [4], is called High-Level Synthesis. In this paper we propose an algorithm to synthesis the data path. This problem is known to be an NP-Hard problem as state in [4], [10], [24].

Pierre G. Paulin and John P. Knight published their well-known heuristic algorithm, the Force-Directed Scheduling [5]. This algorithm was refined later by W. F. J. Verhaegh et al. in [6]. C. H. Gebotys et al. studied the optimum solution by using integer linear programming algorithm [10] and these optimum solutions were widely used as a target reference.

There were many works that use Genetic Algorithm to solve the High-Level Synthesis problem. Marc J. M. Heijligers et al. studied how to adapt this technique and reported in [11], [12], [13]. Muhammad Khan Dhodhi et al. in [14], [15], [16], [17], [18] reported the successful of this technique especially in [18]. John P. Knight et al. were interested in this algorithm and implemented a very efficient algorithm [7], [8], [9]. The main contribution of these people, who adopt the Genetic Algorithm to High-Level synthesis, is to find a technique to encode or decode the chromosome with feasible solution, which prevent the GA from wasting the computation capability in evaluating the infeasible solutions.

2. ACO Algorithms

In this section, we give an overview and the development of the Ant Algorithms. Ant Algorithms are recently developed, mostly by Marco Dorigo et al. as in [20], [21], [22] to solved traveling salesman problem and later expand to another problem. The algorithms are based on the natural behavior of ant colony, which use pheromone as their communication media. This pheromone is left along the way as a trail to communicate with each other. While traveling, ants will use the information (pheromone) which left along the way as their guidance. Ant Colony Optimization (ACO) algorithms simulate this behavior by update the pheromone level based on the positive feed back from the quality of the solution, and make decisions based on these pheromone levels. These algorithms usually are the iterative algorithms. In each iteration ants will try search for the solution by state transition rule. After they finished the tours for the solutions, the quality of these solutions will be weight as a pheromone level to be laid in the trials, which called pheromone update rule. These two rules play an important role for ACO algorithms.

Ant System [20] is the first algorithm introduced. The state transition rule of this algorithm is to choose by probabilistic proportional of the pheromone level and the heuristic weight of all the possible choices. All ants are allowed to update

their trails. Ant-Q [21] and Ant Colony System (ACS) [22] are very similar algorithm. Ant-Q algorithm was inspired by Q-learning, a type of Reinforcement Learning algorithm [25]. The ACS is almost the same as Ant-Q but has less computation. Their state transition rules are almost the same as in the Ant System algorithm, except that they are heavily biased by the maximum pheromone level and heuristic weight. For the pheromone update rule only the best solution is allowed to increase the pheromone level, but every ant will decrease the pheromone level as it travels. This is done in order to prevent another ant from taking the same route. The main difference of *MAX-MIN* Ant System [23] from ACS is that the pheromone level is controlled in a limited range. As we can see that the improvement of the algorithm is to prevent ants from following the same route or search stagnation or premature convergence in genetic algorithm.

3. Dynamic Ant Algorithm

In this research we are interested in finding an algorithm that can be matched well with the human design process. So our experience with the design problem can be used as a heuristic to guide the design algorithm.

Normally when we face a design problem, we will look for some of the possible way and make a decision and then repeat the process until we get a complete design. After that we will make a refinement by changing some of the decision we have made until we get a satisfactory solution. Sometimes with a prior experience we can make a short cut or select a choice that gets to the complete design quicker. From the human design process, we formulate it in to an algorithm. Before we explain the algorithm, let us define some terms first.

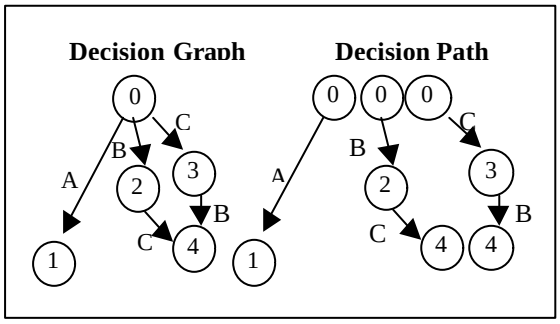

Figure 1: Decision Graph and Decision Path

Decision Path is a record of the decisions, which were made for a design. Each node of a path represents a design problem state. The first node of the decision path is the starting point of the design process. From the first node a designer makes a decision in selecting a choice from all of the possible ways. After that the designer will reach to another state (node) in the design problem. This process is repeated until the designer gets a complete design or gets to the dead end.

Decision Graph is a structure that represents a decision space for a design problem. Each node represents a design problem state. Each arc represents a choice that must have been made to get to the next node or the next state. It is the same as decision tree but in this case the different sequence of decision can get to the same state or the same node. Or each node can have more than one in degree. There is one root node, which represents the initial state of the design problem. A path from the root node to a leaf node is called a decision path.

Construction of a decision path is a process that a sequence of selection is made. This process begins from the first node, which is an initial design state. Then the possible choices are listed and each possible choice is assigned with a heuristic weight and a pheromone weight as in the Ant algorithm. A choice is selected according to the state transition rule, which will be described next, and then the algorithm gets to the new state of the design problem. This process continues until the design is completed. While constructing the decision path the decision graph is also updated.

State transition rule is the rules that will be applied in order to advance from one design step to the next step by selecting a choice from the possible weighted choice list. If all of the choices were selected (by previous ants), then a choice will be randomly selected by the biases from pheromone level. This is diverted from another ACO, in ACO the selection will be based on a linear combination of heuristic weights and pheromone levels but in Dynamic Ant the heuristic weights are used until all choices are selected then the pheromone levels are considered. The advantage of our method is that the determinations of heuristic weights are independent to pheromone levels.

Pheromone updating rule is the rule for calculating the new level of pheromone. In this algorithm we use the technique which was modified from *MAX-MIN* Ant System [23]. The pheromone level is used to prevent the algorithm from storage explosion. When a path is selected the pheromone is set to a fixed value. Each iteration, every arc in the graph is decreased to evaporate the pheromone. Hence, the number of nodes in the decision graph is bounded.

Path exploration is a process to initialize new paths from the interesting path. The process begins by selecting a random exploration point in the path. Then a path from the root to the exploration point is copied for a new ant as an initial path.

Now we introduce the steps of the Dynamic Ant Algorithm. We call this Dynamic Ant because some of techniques were borrowed from Dynamic Niche Sharing as in [19].

1. Initialize path of each ant by an initial design state.
2. For each ant constructs a decision path by mean of state transition rules.
3. Evaluate the cost function of each path, if the target cost is found then terminate.
4. Decrease the pheromone level of every arc in the graph to simulate the pheromone evaporation.
5. Use Dynamic Niche Sharing process to find the peaks (local minimum cost of the near by solution).
6. For each peak, updates pheromone level according to pheromone updating rule.
7. Remove all the arcs, which has zero level of pheromone.
8. Remove all the nodes, which does not have input arc.
9. Initialize path of each ant in the next iteration by path exploration.
10. Go to step 2.

Adjusting the pheromone level in step 4 and removing some arcs and nodes in step 7 and 8 prevent the algorithm from storage explosion which occurs in reinforcement learning [25]. In step 5 the dynamic niche algorithm as in [19] is used to overcome the premature convergence problem and to explore the search space. To initialize a path for each ant in step 9, each new initial path will be generated by path exploration of a randomly selected peak.

The state transition rule in our algorithm is different from other ACO as it constructs a decision path. This is well matched with the path initialization in step 9. So in this algorithm the Q0 parameter which can be found in normal state transition rule was eliminated.

4. Problem

We select Differential Equation Solver as our problem for testing this algorithm. It was used as an example in various literatures [1], [2], [5], [18]. Input to the algorithm was a CDFG that will numerically solve the equation $y'' + 3xy' + 3y = 0$ In this experiment, we use the target architecture as in [2], which are combinational functional units (except the pipeline functional unit), distributed registers, multiplexers or unidirectional buses. All registers use the same clock edge. We also put the constraint that each register and functional unit has only one bus per input as in [10]. A wire, connecting from an output to input, will be counted as a bus, because it also has to be routed in a channel [1]. For loop registers, we ensure that the input loop registers will be valid until their last use. The output loop registers will valid until the end of the loop. The input loop register is the same register as the output loop register for the same variable.

As in [10], we use the cycle time cost equal to 50 and the cost of each hardware unit as in Table 1.

Table 1: Cost of hardware unit

Hardware Unit	Cost
Single cycle multiplier	250
Two cycle multiplier	250
Pipeline multiplier	250
Single cycle ALU (+, -, <)	250
Single cycle adder	50
Single cycle subtractor	50
Single cycle comparator	50
Register	15
Bus	100

5. Experiment

There are two major parts of the system that we have to implement. First part is the Dynamic Ant algorithm, which make decisions for the second part, the design algorithm.

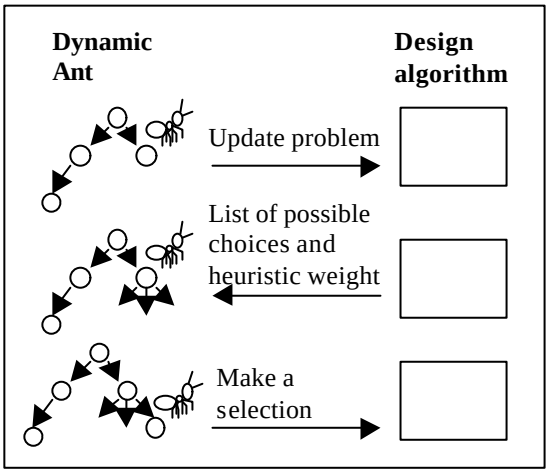

Figure 2: The interaction of Dynamic Ant and design algorithm.

In implementing the Dynamic Ant algorithm we have to design the structure, which will keep a problem state. The structure that we choose is a table that is indexed by operation number to assigned time step, assigned functional unit, assigned register number, assigned input and output buses. The structures of graph and path also have to be implemented to keep track of decision graph and decision paths.

For the design algorithm we follow the following step:

1. Allocation of Functional units (assigned operation to the functional unit in the library).
2. Scheduling (Mobility scheduling, each operation bound by ASAP and ALAP time step).
3. Reallocation (allocate enough function unit for all of the time steps).
4. Functional unit assignment.
5. Register assignment.
6. Bus assignment.

These are the normal techniques as can be found in various literatures as [1], [2], [3]. Most of them are a heuristic algorithm and can not guarantee the optimum solution. Combining many of them together is unlikely to get the optimum solution. We had modified them to match with the Dynamic Ant algorithm. Whenever a decision has to be made and there is not enough information, the design algorithm will list the possible choices with the heuristic weights and let Dynamic Ant select one of them.

The target cost that we use to measure as a success in experiment was taken from [18]. But the solutions in that literature did not report number of bus required. We ran our algorithm and checked for the solution that we think is the optimum or the closest one and use it as a target.

The experiment was carried out with the parameter list in Table 2.

Table 2: Parameter list

Number of ant per iteration (except for 4 cycle 50 ants were used)	100
Pheromone level (if the path was selected)	4
Pheromone evaporation rate (per iteration)	1
Number of run (per experiment)	100

We had run the 4 experiments as follow:
1. 4 time steps with single cycle adder, single cycle subtractor, single cycle comparator and single cycle multiplier.
2. 7 time steps with single cycle adder, single cycle subtractor, single cycle comparator and two cycles multiplier.
3. 7 time steps with single cycle ALU and pipeline multiplier.
4. 6 time steps with single cycle ALU and pipeline multiplier.

To verify the usefulness of adding heuristic, we ran each experiment with and without heuristic to compare the result. The heuristics, which we were added to the algorithm are:

1. Prefer the cycle that has less operation and less data transfer.
2. Prefer the register that connects to the same functional unit output.
3. Prefer the bus that connects to the same functional unit output.

As one may notice that some of these heuristics were inspired by [5]. In the Dynamic Niche selection, if two solutions have the same set of functional units and scheduling or have a few of difference in scheduling, then both of them will be classified in to the same niche.

6. Result

The experiments were carried out on a personal computer with Pentium II processor running at 350 MHz and 128 Mbytes of memory. The execution time is approximately a second per iteration.

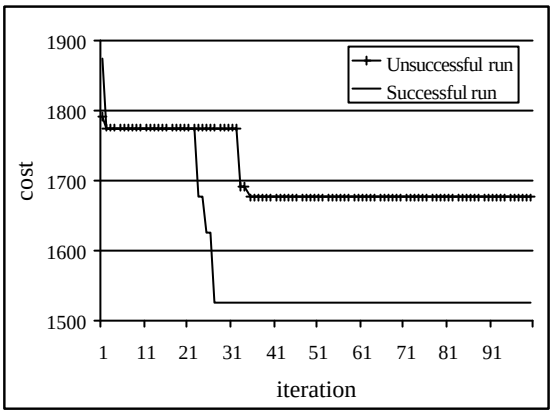

Figure 3: Cost and iteration of a run with heuristic added.

Two of the runs are shown in Figure 3, one for the successful run and another for the unsuccessful run. These two solutions were from the design with 7 cycles using pipeline multiplier. For the successful run, the cost is 1525, which consists of 250(1ALU) + 250(1 pipeline mul) + (5*15)(5 regs) + (6*100)(6 buses) + (7*50)(7 cycles). In case of unsuccessful run, which there is only one for this problem, the cost is 1675, which consists of 250(1ALU) + (2*250)(2 pipeline mul) + (5*15)(5 regs) + (5*100)(5 buses) + (7*50)(7 cycles). The unsuccessful solution has one more multiplier, but one less bus. The data path of the successful run is shown in Figure 5.

Figure 4 shows the solution for the 4 cycles design problem. This is the ASAP scheduling, as one may expect this solution requires more functional unit and more bus. In this data path the cost is 1725, which consists of 50(1 adder) + 50(1 subtractor) + 50 (1 comparator) + (2*250)(2 mul) + (5*15)(5 regs) + (8*100)(8 buses) + (4*50)(4 cycles). From

Table 3 this is the easiest problem. All of the runs obtained the target solution even with only 50 ants.

Figure 6 shows the solution for 7 cycles with multi-cycle multiplier the cost is 1775, which consists of 50(1 adder) + 50(1 subtractor) + 50 (1 comparator) + (2*250)(2 mul) + (5*15)(5 regs) + (7*100)(7 buses) + (7*50)(7 cycles). From the result in Table 3 this is the hardest problem, since in this problem the algorithm has the least percentage of success.

Figure 7 shows the solution for 7 cycles with pipeline multiplier the cost is 1725, which consists of 250(1 ALU) + (2*250)(2 mul) + (5*15)(5 regs) + (6*100)(6 buses) + (7*50)(7 cycles).

From the result in Table 3, the Dynamic Ant can find the target solution in most of the runs. These are the same solutions as found in [18] except that this algorithm also assigned buses for the data path. If we compare the results in the column that used heuristic and the column that did not use heuristic, we can conclude that the algorithm can find the solution faster with heuristic. In all problems, the algorithm with heuristic obtains the target solution more than the algorithm without heuristic. In case of the number of iterations the algorithm also performs better than the one without. Usually the run that can not find the target solution reported the solution that is closed to the target solution.

7. Conclusion

It is important to find a data structure to represent state of the design problem. Then the structure of graph and path have to be built, someone may found it a complicate structure. In Genetic Algorithm we have to find good encoding of the solutions and the proper genetic operators. These steps sometimes are not trivial, because improper encoding method may lead to infeasible solution (which we have to recognize them) and to inefficient mapping (many to one). Many people who adopted the Genetic Algorithm have to find the new encoding and new genetic operators as [9], [13], [18].

By comparing the implementation of Dynamic Ant Algorithm and Genetic Algorithm we conclude that Dynamic Ant is easier if you can not find the suitable encoding and genetic operators. The interesting point is that it is easy to add heuristics (may be from old design algorithm). One even can combine more than one approach in this algorithm. As John P. Knight stated in his paper [9] that we need a reasonable solution in a reasonable time. This may be a good solution even though it is a stochastic algorithm, which can not guarantee the time and optimality of the solution.

8. References

[1] Gajski, Daniel., Dutt, Nikil., Wu, Allen. and Lin, Steve. High-Level Synthesis: Introduction to Chip and System Design, *Kluwer Academic Publishers*, 1992.

[2] Michel, Petra., Lauther, Ulrich. and Duzy, Peter. The Synthesis Approach to Digital System Design, *Kluwer Academic Publishers*, 1992.

[3] Micheli, Giovanni. Synthesis and Optimization of Digital Circuits, *Mcgraw-Hill*, 1994.

[4] McFarland, Michael., Parker, Alice. and Camposano, Raul. The High-Level Synthesis of Digital Systems, *Proceedings of the IEEE*, Vol. 78, No. 2, Page(s). 301-318, Feb, 1990.

[5] Paulin, Pierre. and Knight, John. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's, *IEEE Transaction on Computer-Aided Design*, Vol. 8, No. 6, page (s). 661-679, 1989.

[6] Verhaegh, Wim., Lippens, Paul., Aarts, Emile., Korst, Jan., Meerbergen, Jef. and Werf, Albert. Improved Force-Directed Scheduling in high-throughput digital signal processing, *IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems*, page(s). 945-960, 1995.

[7] Martin, R. and Knight, John. PASSOS: A Different Approach for Assignment and Scheduling for Power, Area and Speed Optimization in High-Level Synthesis, *Proceedings of IEEE 37th Midwest Symposium on Circuits and Systems*, Vol. 1, page(s). 339-342, 1994.

[8] Torbey, Elie. and Knight, John. High-Level Synthesis of Digital Circuits using Genetic Algorithms, *Proceedings of IEEE International Conference on Evolutionary Computation*, page(s). 224-229, 1998.

[9] Torbey, Elie. and Knight, John. Performing Scheduling and Storage Optimization Simultaneously Using Genetic Algorithms, *Proceedings of IEEE Midwest Symposium on Circuits and Systems*, page(s). 282-287, 1999.

[10] Gebotys, C. and Elmasry, M. Optimum Synthesis of High Performance Architectures, *IEEE Journal on Solid State Circuits*, Vol. 27, No. 3, page(s). 389-397, 1992.

[11] Timmer, Adwin., Heijligers, Marc., Stok, Leon. and Jess, Jochen. Module Selection and Scheduling using Unrestricted Libraries, *Proceedings of IEEE fourth European Conference on Design Automation*, page(s). 547-551, 1993.

[12] Heijligers, Marc., Cluitmans, L. and Jess, Jochen. High-Level Synthesis Scheduling and Allocation using Genetic Algorithms, *Proceedings of IEEE International Conference on Hardware Description Languages*, page(s). 61-66, 1995.

[13] Heijligers, Marc., and Jess, Jochen. High-Level Synthesis Scheduling and Allocation using Genetic Algorithms based on Constructive Topological Scheduling Techniques, *Proceedings of IEEE International Conference on Evolutionary Computation*, page(s). 56-61, 1995.

[14] Ahmad, Imtiaz., Dhodhi, Muhammad. and Hielscher, Frank. Design-Space Exploration for High-Level Synthesis, *Proceedings of IEEE 13th International Phoenix Conference on Computer and Communications*, page(s). 491-496, 1994.

[15] Dhodhi, Muhammad., Ahmad, Imtiaz. and Ismaeel, Asad. Data Path Synthesis for Easy Testability, *Proceedings of IEEE third Asian Test Symposium*,page(s).317-322,1994.

[16] Ahmad, Imtiaz., Dhodhi, Muhammad. and Chen, C. Integrated scheduling, allocation and module selection for design-space exploration in high-level synthesis, *IEE Proceedings of Computers and Digital Techniques*, Vol.142, No.1, page(s). 65-71, 1995.

[17] Dhodhi, Muhammad., Ahmad, Imtiaz. and Ismaeel, Asad. High-level synthesis of data paths for easy testability, *IEE Proceedings of Circuits, Devices and Systems*, Vol. 142, No. 4 page(s). 209-216, 1995.

[18] Dhodhi, Muhammad., Hielscher, Frank., Storer, Robert. and Bhasker, Jayaram. Datapath Synthesis Using a Problem-Space Genetic Algorithm, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 8, page(s). 934-944, 1995.

[19] Miller, Brad. and Goldberg, David. Genetic Algorithms with Dynamic Niche Sharing for Multi-modal Function Optimization, *Proceeding of IEEE International Conference on Evolutionary Computation* 1996.

[20] Dorigo, Marco. and Colorni, Alberto. The Ant System: Optimization by a colony of cooperating agents, *IEEE Transaction on Systems, Man and Cybernetics-Part B*, Vol. 26, No. 1, pp.1-13, 1996.

[21] Gambardella, Luca. and Dorigo, Marco. Ant-Q: A Reinforcement Learning approach to the traveling salesman problem, *Proceedings of the 12th. International Machine Learning Conference (ML-95)*, pp. 252-260, 1995.

[22] Dorigo, Marco. and Gambardella, Luca. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transaction on Evolutionary Computation*, Vol. 1, No. 1, 1997.

[23] Stützle, Thomas. and Hoos, H. The *MAX-MIN* Ant System and Local Sear for the Traveling Salesman Problem, *Proceedings of IEEE international Conference on Evolutionary Computation (ICEC'97)*, page(s). 309-314, 1997.

[24] Gary, M. and Johnson, D. Computers and Intractability: A Guide to the Theory of NP-Completeness, *Freeman*, 1979.

[25] Kaelbling, Leslie., Littman, Michael. and Moore, Andrew. Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research 4*, page(s). 237-285, 1996.

Table 3: Comparison between the design with and without heuristic guide

Design	4 cycles single		7cycles multi		7 cycles pipe		6 cycles pipe	
Heuristic	With	W/o	With	W/o	With	W/o	With	W/o
%success	100.00	100.00	84.00	76.00	99.00	88.00	92.00	86.00
Mean*	4.05	7.23	27.92	30.62	14.46	18.93	24.96	31.14
Standard deviation*	3.31	7.55	25.51	25.26	18.24	20.78	23.04	23.55

* Iteration mean and standard deviation of success runs only.

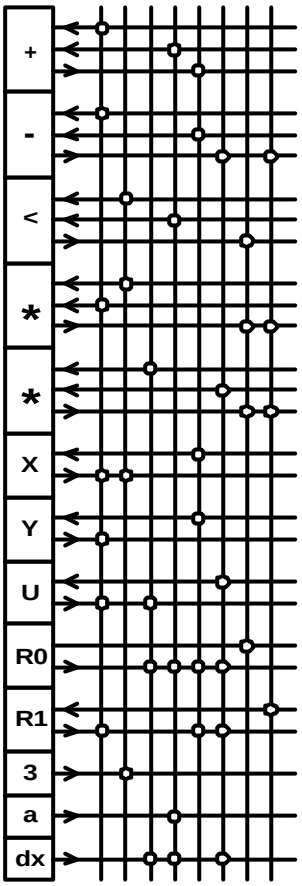

Figure 4: The solution of 4 cycles (single cycle multiplier)

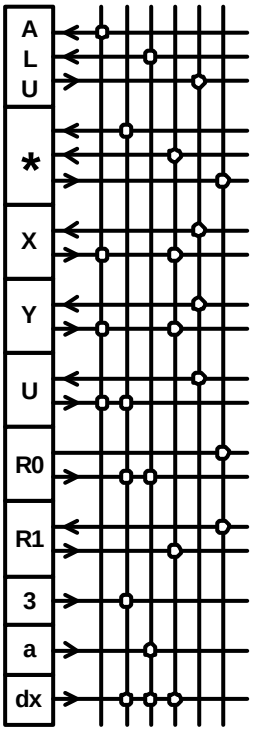

Figure 5: The solution of 7cycles (pipeline multiplier)

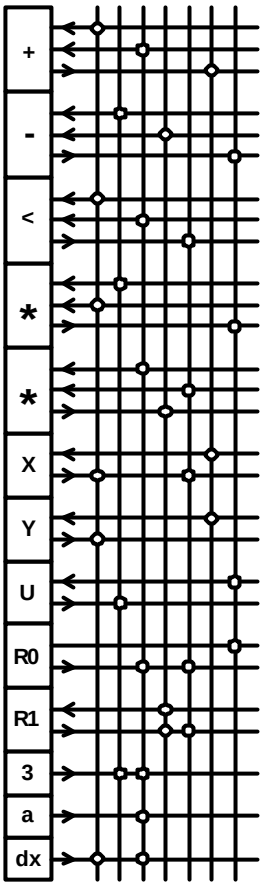

Figure 6: The solution of 7 cycles (two cycles multiplier)

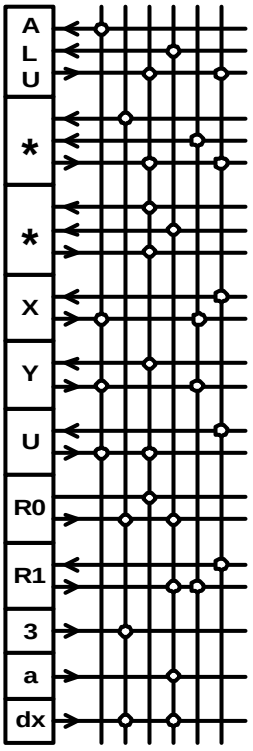

Figure 7: The solution of 6 cycles (pipeline multiplier)