NAIST-IS-MT0351085

Master's Thesis

Reconfigurable 1-bit processor array with reduced wiring area

Nobuo Nakai

February 3, 2005

Department of Information Systems
Graduate School of Information Science
Nara Institute of Science and Technology

Master's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
MASTER of ENGINEERING

Nobuo Nakai

Thesis Committee: Katsumasa Watanabe, Professor
Hideo Fujiwara, Professor
Shigeru Yamashita, Associate Professor

Reconfigurable 1-bit processor array with reduced wiring area*

Nobuo Nakai

Abstract

Semiconductor makers have a problem of how to reduce the production cost. The production cost is increasing because the increasing logic gates to implement and shortening production cycle. One of the way to solve this problem is to use of reconfigurable hardwares. Although reconfigurable hardwares seemed to be useful, they have some disadvantages. As as result, a system using software or ASIC costs lower than reconfigurable hardware in many cases. In this paper, we propose the new reconfigurable hardware architecture with low cost. The proposed architecture has the following features; It has high routability but wiring area is reduced, and the number of processor elements can be increase easily. We mapped DCT circuit to the proposed architecture for evaluating the performance and chip area. The proposed architecture can be mapped DCT circuit with lowest area in comparison objects and achieved sufficient performance for real time processing of the MPEG2 encoding.

Keywords:

reconfigurable computing, coarse-grained architecture, bit-serial data path, wiring resource

*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT0351085, February 3, 2005.

配線リソースを考慮した
再構成可能 1bit プロセッサアレイ*

中井 伸郎

内容梗概

半導体メーカは，製品製造にかかるコストの増加を如何にして抑えるかに悩まされている．コストの増加はユーザニーズの多様化や実装する論理回路規模の増大が原因で起こっている．その問題を解決する一手法として再構成可能ハードウェアの利用が挙げられる．しかし再構成可能ハードウェアは様々な問題を抱えているため，ASIC やソフトウェアを用いたシステムの方が結果として低コストである場合が多い．本稿では低コストでシステムに組み込む事が可能な再構成可能ハードウェアのアーキテクチャを提案する．提案するアーキテクチャは，省配線領域であるが柔軟な配線構造，プロセッサエレメント数の拡張が容易であるという特徴を持っている．そして提案するアーキテクチャを評価するために DCT 演算器をマッピングした．その結果，他の比較対象の中で最も低い面積で DCT 演算器をマッピングでき，MPEG2 のリアルタイムエンコーディングを十分行える性能を達成した．

キーワード

リコンフィギャラブル コンピューティング, 粗粒度アーキテクチャ, ビットシリアルデータパス, 配線リソース

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT0351085, 2005 年 2 月 3 日.

Contents

List of Figures

List of Tables

1. Introduction

Semiconductor makers are always struggling to reduce the production cost that increases every year. One of the reasons of the cost growth is that the number of implementation gates on chip is increasing. Another reason is that production cycle becomes shorter and shorter since many new formats and protocols are being proposed, and consumers' tastes are changeable.

One of the way to solve the problem is to use reconfigurable hardware. Reconfigurable hardware brings the following advantages [1].

1. It is not necessary to make the chip for trial purposes. Therefore, the development period will be reduced.

2. Even though a part of a chip has failure, it is possible to use the chip by reconfigure the chip so that a failure part is not used. Therefore, the yield ratio of a chip can be increased.

3. Reconfigurable hardware can be used for many applications. Thus we need to produce only one hardware for many applications. Then production cost will be reduced.

Currently Field Programmable Gate Array (FPGA) are the most commercially successful reconfigurable hardware [2] [3]. They are called *fine-grained* architecture [4] since the function of a logic element is relatively small. Although FPGAs are used widely in industry, this type of architecture essentially suffers from the wiring resource problems, i.e., they need a lot of wiring resources that may degrade the performance, i.e., speed and chip area [5].

Recently *coarse-grained* reconfigurable architectures have been studied[6] [7]. This type of architectures generally need less wire resources than fine-grained type. However, if the width of data in an application does not match with the width of the logic elements, some logic parts of an element are not used.

Thus, both types of reconfigurable hardwares have a problem that the performance per chip area is not good enough to be replaced with ASIC implementation or software implementation in most cased. However, it is needless to say that reconfigurable hardwares have potential, and indeed they are already used

for a practical usage in some cases.[8] [9] Thus, if the performance of reconfigurable hardwares is improved more, they would be more valuable hardwares. To improve the performance of reconfigurable hardwares, we should consider the "wiring resource" problem the most.

To reduce the necessary wiring resources, bit-serial architectures have been proposed [10] [11]. In bit-serial architecture, each element processes only one bit at a time and outputs a one bit of the result to adjacent elements. Therefore, they do not suffer from the problem that some part of an element are not used. Also they do not need complex wiring resources. However, some computations, such as multiplications, are difficult to be mapped on a bit-serial architecture, and some applications with complex data flow may be difficult to be mapped on the bit-serial architecture proposed in [11] because each element can communicate with only adjacent neighbors in their architecture.

In this paper, we propose a bit-serial architecture with the following new features:

1. In our architecture, there is a global bus, and a designer can determine the length of the bus, and the number of buses freely. With global bus and a local communication with adjacent elements, we can map many applications very easily.

2. Each element have some designated data registers and some logics so that multiplications can be done efficiently in a bit-serial architecture.

3. We make the overall architecture simple and scalable, and so it is easy to increase the number of elements in our architecture.

Indeed the comparisons with other methods indicate that our architecture is promising in the sense that it can provide a good performance per chip area.

This paper is organized as follows. Section II explains the features of bit-serial data path since they are important for understanding our architecture. Then we show the overview of the proposed architecture and wiring resource in Section III. The proposed architecture has three types of element. The concept of each element is described in Section IV. We also show how efficient multiplication is done in our architecture. Section V shows the implementation report, and the

performance evaluation is described in Section VI. Finally, we conclude the paper in Section VII.

2. bit-serial data path

In this section, we describe the feature of the bit-serial data path. The bit-serial data path is a base of the proposed architecture. First, the difference between the bit-serial data path and the bit-parallel data path is explained. Next, how to construct the bit-serial data path is shown. Then feature of the bit-serial pipeline is described.

2.1 bit-serial vs. bit-parallel

Consider about n-bit arithmetic addition. Using a n-bit synchronous full adder, we can get a result in one clock (see Figure 1 (a)). Like this n-bit full adder, an

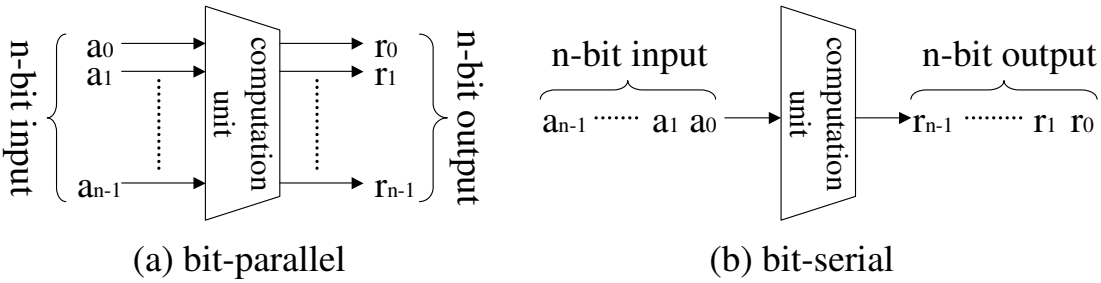

Figure 1. bit-parallel and bit-serial data path

operation that processes each data bit in parallel is called "bit-parallel operation." A data path for a bit-parallel operation is called "bit-parallel data path." On the other hand, an operation that processes each data bit per clock is called "bit-serial operation." Bit order of processed data is start from LSB to MSB, or from MSB to LSB. In a bit-serial operation, it takes n clocks to compute n-bit data operation (see Figure 1 (b)). A data path for a bit-serial operation is called "bit-serial data path."

Compared with a bit-serial data path, a bit-parallel data path has following advantages.

1. We can obtain relatively higher throughput.
2. We need only small controller.

The first point is due to the fact that we need to process only one bit in a bit-serial data path in one clock while a bit-parallel data path needs to process n bits. A bit-serial data path has to recognize the end of data bits and do a special operation (e.g. reset the carry when addition). Thus a bit-serial data path needs a control unit. However in many cases, a bit-parallel data path does not need such a controller.

On the other hand, compared with a bit-parallel data path, a bit-serial data path has the following advantages.

1. The number of the logic gates to be implemented is reduced.
2. It is easy to raise the frequency.
3. It is not need much wiring resource.

In a bit-serial data path, the width of processed data in one clock cycle is only one bit. This is the reason why the first advantage is obtained. A computation with carry propagation does not become a factor of critical path and logic depth is kept low, thus the second advantage is obtained. The third advantage is obtained by the same reason as the first one, and we consider this is the key point that we adopt a bit-serial data path.

In general, many reconfigurable hardware architecture consist of many elements which processes data and programmable wiring resources. It would be desired for an element to having a simple structure and small area. It would be desired for wiring resources to having high routability and reduced area. We consider that a bit-serial data path is suitable for a reconfigurable hardware because it satisfies these two desired features.

2.2 construction of the bit-serial data path

One way to construct a data path is mapping a control/data flow graph (CDFG) to reconfigurable resources. A bit serial data path can be constructed by the same way. A CDFG is a graph in which an operation is corresponding to a node and a data flow is corresponding to a directed edge. Additionally, a CDFG is able to show control flow. Figure 2 is an example of a CDFG. In the example, if variable a's value is greater than variable b's value, then the value that $a - b$

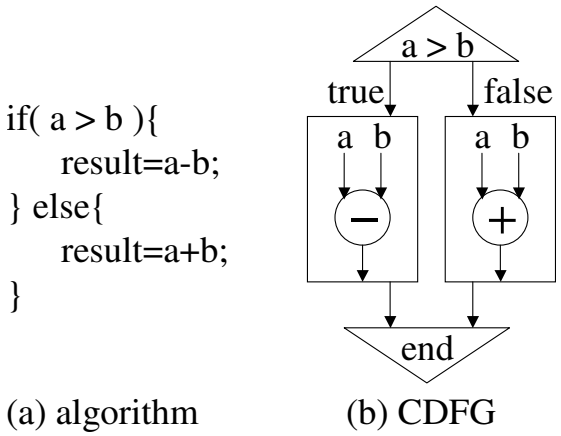

Figure 2. control/data flow graph

becomes a result. Otherwise the value $a+b$ becomes a result. The objective data path is constructed by transforming a CDFG into logic gates.

To construct a bit-serial data path, an additional process is needed; we need to insert proper delays to synchronize data inputs. In a bit-serial operation, some operations generate delay. Therefore we need to synchronize data inputs. The delay occurs because we cannot compute until all data bits arrive, e.g. comparison, right shift, division is such operation. Especially it is necessary to decide the division operation whether to subtract the dividend from the divisor in every stage. Therefore it is relatively difficult to implement a division operation by a bit-serial data path.

We describe how to synchronize the data inputs. Consider about the operation with two inputs and one output as shown in Figure 3. Node A outputs a calculation result with no delay to node C. Node B outputs a calculation result with some delay to node C. Thus, there is a gap between the arrival time of data from node A and node B. Then, node C outputs wrong result. To solve this problem, we insert new node for delay between the source node without delay and the destination node. Figure 4 is the CDFG that new node was inserted between node A and node C of Figure 3.

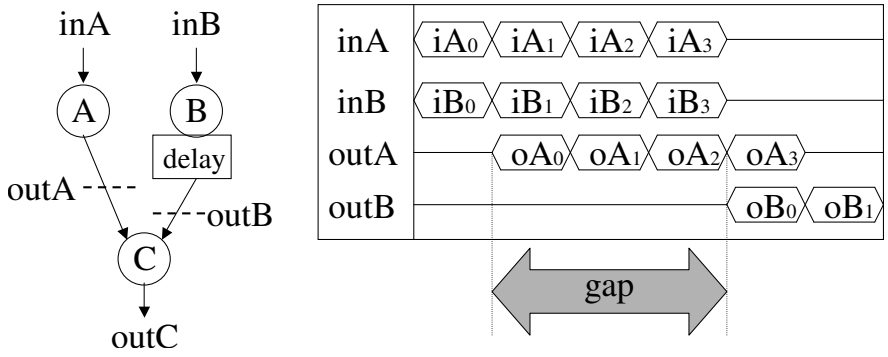

Figure 3. Normal CDFG and its timing chart. The first letter of each waveform means input or output. The second letter means node. Then subscript of the second letter shows bit order of the input/output data.

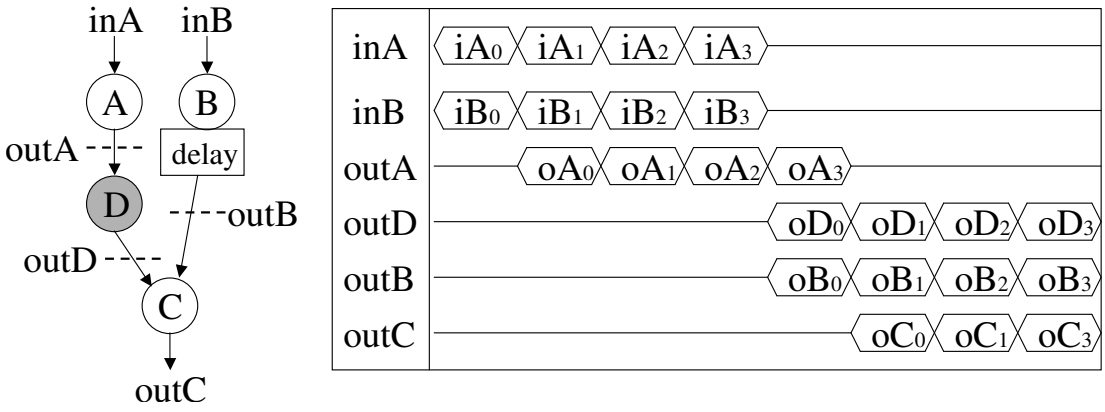

Figure 4. CDFG with insertion delay

2.3 bit-serial pipeline

In the previous subsection, we discussed a case where we perform one calculation. As usual, we can implement an efficient pipeline when we perform multi computational tasks sequentially. A bit-serial data path needs n times clock cycles as a bit-parallel data path, when one element processes n-bit data. However, a bit-serial data path does not need $n \times m$ times clock cycles, when we perform m sequential computations of n-bit data.

Figure 5 is a CDFG which consists of three stage of full adder node, and computes $((a + b) + c) + d$. $X_{nm}(1 \leq n \leq 3, 0 \leq m \leq 3)$ in the Figure shows the m-th output bit of node n. At the time $t = 1$, the first node processes LSB

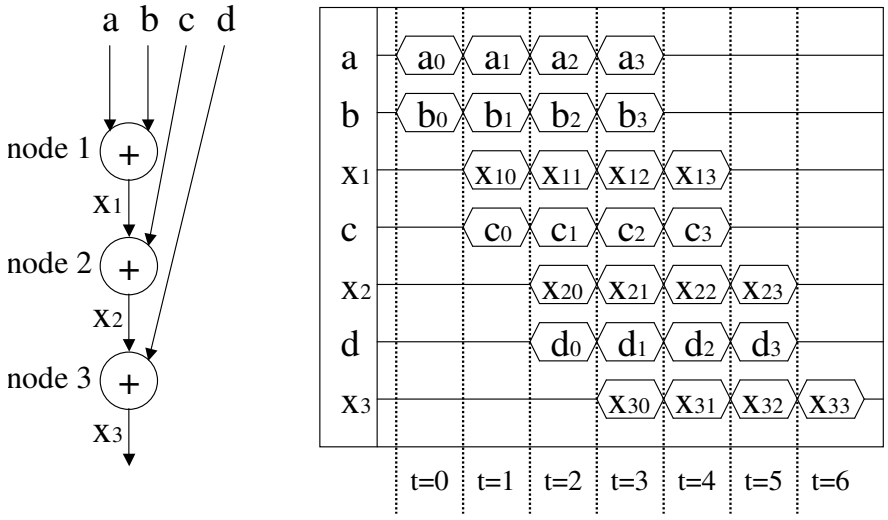

Figure 5. bit-serial pipeline consist of 3 stages

of input a and b, then output result X_{10}. At the time $t = 2$, the first node processes the second bit of inputs. Concurrently, the second node processes X_{10} and LSB of input c. At the time $t = 3$, the all nodes processes input bits for each node concurrently. Then the third node outputs X_{30} as a final result. Each node of the CDFG works as a pipeline stage by inputting data continuously. Accordingly, we can obtain the 4 bit result every 4 clock cycles if input data come continuously. We utilize this pipeline technique in our bit-serial data path to increase the throughput.

3. Overview of the architecture and its wiring resources

3.1 Overview of the architecture

Figure 6 shows overview of the proposed architecture. The proposed architecture

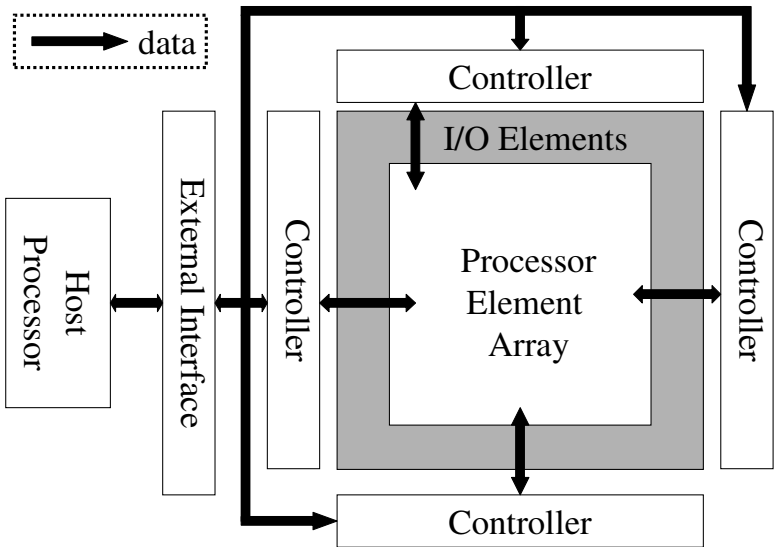

Figure 6. Overview of the architecture

is designed as a peripheral device of a host processor. The external interface mediates the communication between the host processor and the proposed architecture in appropriate way. The host processor is a independently running system which has CPU and memory. The proposed architecture consists of the controllers, the processor elements (PE), and the I/O elements (IOE). The controllers are used to communicate with the host processor. Controlling the PEs and the IOEs is another function of the controller. The PEs process input data and outputs a result. The operation of a PE depends on the configuration data. Giving a appropriate operation to each PE in the PE array, the PE array forms an application specific circuit. The IOEs works as I/O ports which mediates data transfer between the controllers and the PEs.

The PEs and the IOEs are placed two dimensionally. The PE array is sur-

rounded by the IOEs as the Figure 6 shows. Some coarse-grained architectures are not scaled up the number of elements easily because of their ways of I/O interface. For example, we cannot scale up such architectures that all PEs are connected to the controller, and/or global busses exist. This is because the amount of wiring resource is increased, and the wiring delay becomes longer. In our architecture, data transfer between the controllers and the PEs are limited to be performed via the IOEs. Also, as we will mention, there is no global bus. Thus, our architecture can be scaled up easily, and the number of elements is not fixed unlike many other reconfigurable architectures [14] [15]. Therefore, a designer can choose an appropriate number of elements for his/her usage when the chip is implemented.

After the reset process, the proposed architecture processes the following steps.

1. All PEs, IOEs, and controllers accept the configuration data from the host processor via controllers.
2. All PEs, IOEs, and the controllers wait for data from the host processor.
3. The controllers transfer data to the PEs via the IOEs.
4. A PE processes input data and output a result. This result becomes a input of the next PE.
5. The last PE transfer the result to a controller via the IOE.
6. Finally, the controller transfer the result data to the host processor.

3.2 The definition of the data signal

The proposed architecture employs serial method in data transfer between elements. Here, we explain how the data transfer is done. A physical bus that connects elements consists of two lines. One of two lines is called the *master bus*, the other line is called the *inverse bus*. The signal value is defined by combination of the signal level of the *master* and the *inverse bus*. Figure 7 shows a definition of the data signal. If the signal level of the *master bus* and the *inverse bus* is the same, the data bus is high-impedance. If the signal level of the *master bus* and the *inverse bus* is different, the *master bus's* signal level is considered to be

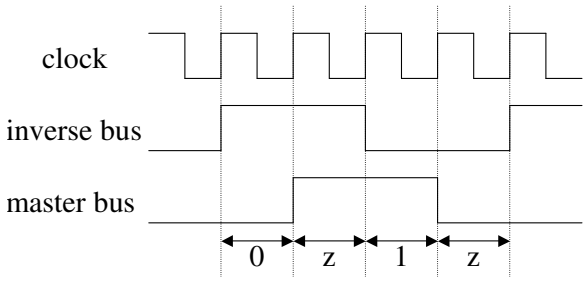

Figure 7. Definition of the data signal

the logical value for the signal. Immediately after reset, all PEs and all IOEs outputs the high-impedance for prevent the PEs and the IOEs from processing data incorrectly.

3.3 The wire networks

The proposed architecture has two types of wires for data transfer. One of them is called the "short wires." The other is called the "long wires." The two types of wires are used for different objects. Figure 8 illustrates how PEs are connected by the short wires and the long wires.

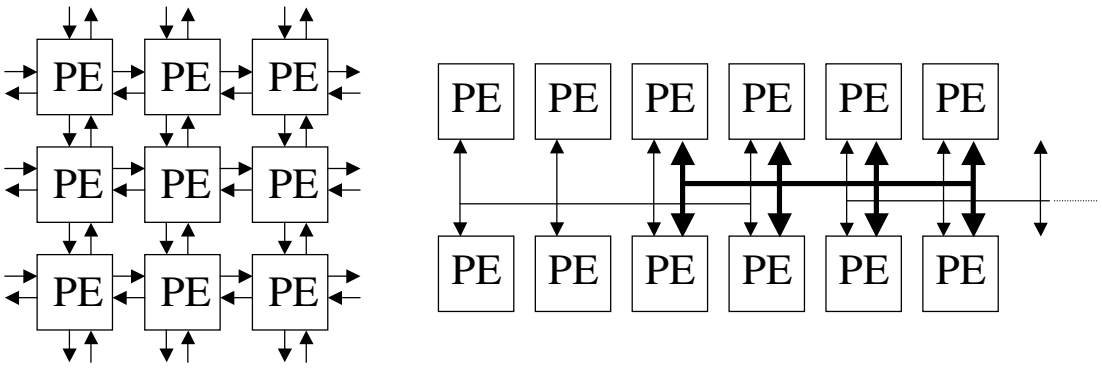

Figure 8. Wire networks

The short wire is a unidirectional bus and used for data transfer between

adjacent elements. Each PE has eight short wires. The half of the eight short wires is directed from a PE to the four adjacent PEs or IOEs. Another half is directed from the four adjacent PEs or IOEs to the PE . There is no short wires between adjacent IOEs since it is not need to transfer data between the IOEs. therefore each IOE only has short wires to/from an adjacent PE. Exceptionally, the IOE placed at a corner of the array does not have short wires and only has long wires.

The long wire is a bidirectional and shared bus. The long wire is used for data transfer between elements that are far apart. The long wire is lined out between rows/columns of the element array. From the viewpoint of the long wire, the PEs which is placed both side of long wires share the long wire. If an element outputs a data to a long wire, the data is broadcasted to the elements that shares the same long wire. Which element outputs data to the long wire, and which element accepts the data from the long wire is decided when the elements are configured. Once the element that outputs data to the long wire is decided, other elements that share the same long wire can not output data to the same long wire.

Unlike the short wire, the long wire's network is not fixed. When the proposed architecture is implemented on a chip, the long wire's network is determined by the two parameters; the distance and the step. The distance means how far elements of a row/column of the array shares the long wire. As the value of the distance becomes larger, elements can communicate with father elements. The step means interval of the elements a long wire is lined. As the value of the step becomes smaller, but not zero, lager number of the routing channel is obtained. Of course, there is a trade-off; if we set the values of the distance and the step be larger and smaller, respectively, we need more wire resources. A designer can decide these values as considering this trade-off.

3.4 The advantages of the wire networks

As in Figure 9, the maximum number of long wires that exist between rows/columns is to the value of the distance. Moreover a short and a long wire's bus width is two. Thus, if we chose an appropriate value for the distance and the step, the proposed architecture has relatively smaller wiring area than other reconfigurable architectures. Also we can reduce wiring delays since there is no switch blocks

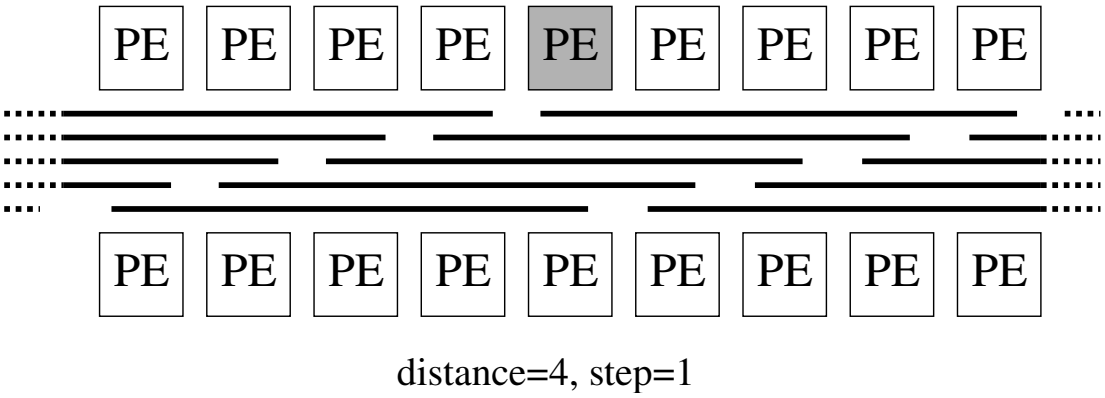

Figure 9. long wires exist between rows

unlike conventional FPGA [16] [17]. An appropriate choice of the value of parameters makes the long wires shorter and makes the wiring delay smaller than architecture with global buses. Of coerce, it becomes difficult to map applications without global buses. However, as we explain the following, our architecture can provides flexible wiring resources. Indeed, we can easily map a DCT circuit without global buses. Consider the case that the start point of the long wire is the same. It needs long wire of distance=8 for the colored PE in Figure 9 to communicate with all the other PEs in the figure. On the other hand, the proposed wiring resource architecture needs only long wires of distance=4. Moreover, the different start point allow the proposed architecture to increase the number of the elements easily.

Because there are long wires between rows/columns of elements, each element can transfer data to the elements in neighbor rows/columns. The base PE can communicate with the dark colored boxes in Figure 10. This property makes the mapper map an application easily. An element can communicate with another elements that does not share the same long wire by using an intermediate element as routing resource. The base PE can communicate with the light colored boxes in Figure 10 by one intermediate element. However mapper should notice that one clock cycle delay is occurred when using an intermediate element.

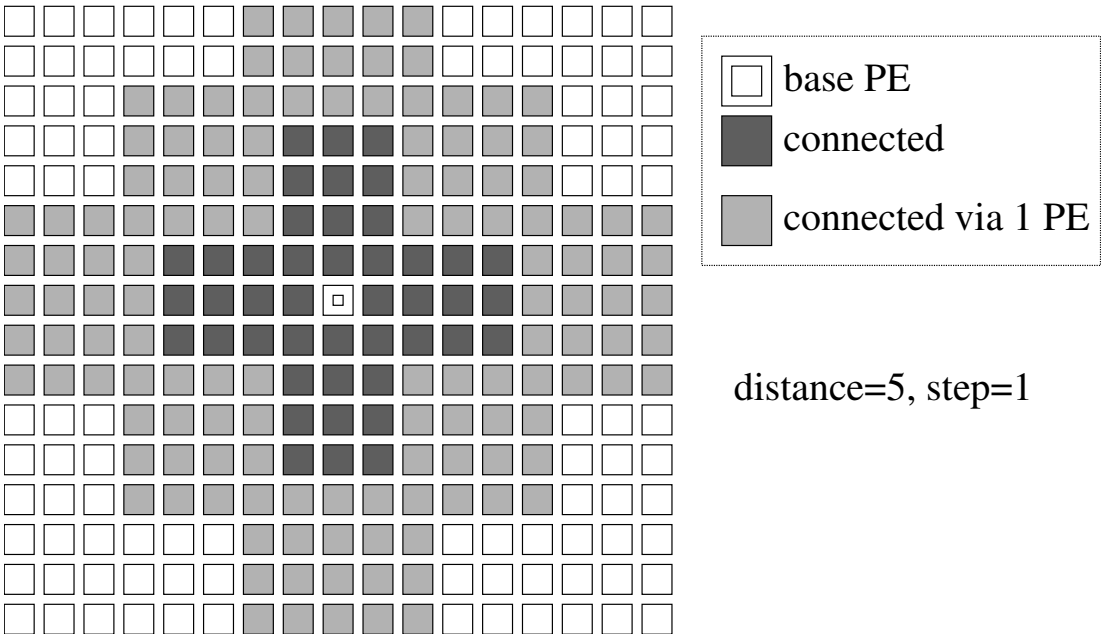

Figure 10. Communicable elements

3.5 Relation between the parameters and the wiring resources

As the value of the distance becomes larger, elements can communicate with father elements by a long wire. However the larger value of the distance yields the following disadvantages.

1. Wiring delay becomes larger. It prevent the proposed architecture from working at higher frequency clock cycle.

2. The number of the logic gates for the selector grows larger. Each element has four selectors. So small selector is desired.

The physical length of the long wire becomes longer when the distance is large. Furthermore, the large number of the elements that are connected to a long wire needs the high drive strength wire. These are the reasons why the first disadvantage is caused. As the value of the distance become larger, the number of the long wires connected to a PE increases. Thus, selectors in a PE become larger.

As the value of the step becomes smaller, the number of the routing channels that an element has become larger. Figure 11 shows the changes of the the number of the routing channels when the distance=5. All elements have symmetric wiring

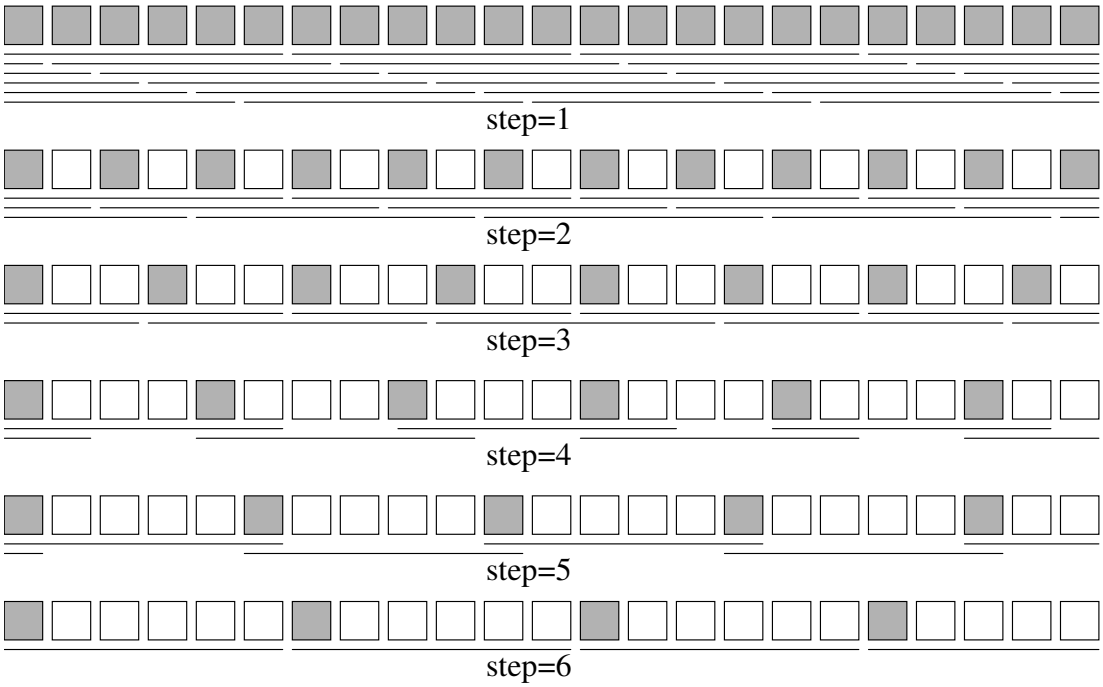

Figure 11. The change of long wire's channels (distance=5)
The box shows element and colored box is start point of long wire.

resources if the remainder when $(distance + 1)$ is divided by $(step)$ becomes zero. As the case of step=1, 2, 3, and 6 in Figure 11. When the remainder is not zero, wiring resources that each element has are asymmetric. It is obvious that asymmetric wiring resources make the mapping difficult, and thus we consider only the case of symmetric wiring resources.

We now focus on the maximum number of the long wires between rows/columns. The maximum number L_{max} is expressed by the following equation.

$$L_{max} = \left\lceil \frac{distance + 1}{step} \right\rceil$$

Although L_{max} of step=3 and L_{max} of step=4 in Figure 11 is the same, routability is different. A designer should take care of this fact when choosing the these

parameters.

The distance and the step have a great influence on the properties of a designed architecture (e.g., chip area, clock frequency, what kinds of application can be mapped, etc.). Therefore, to decide the these parameters is important for implementing the proposed architecture. A large distance and a small step is appropriate for mapping the applications which needs complex wire networks like the FFT computation. Contrary to this, a chip having a small distance and large step can accommodate only applications that have simple wire networks. However, this chip needs only small selectors, and thus the chip area is reduced. A large distance and a large step is appropriate for mapping the applications which sometimes needs the communication with father element like the FIR filter.

4. The Processor element and the I/O element

In this section, the architecture of each component is described. Firstly, the architecture of the PE is described. A PE is a basic element which takes inputs then outputs a result. Overview of a PE, format of the configuration data for a PE, how to control a PE is explained. Subsequently the architecture of an IOE is described. An IOE is a special element and used as an external port of the PEs.

4.1 The processor element

A PE has three input ports and one output port. It processes input data bit-serially. A bit-serial processor has to recognize where is the end of data, so the proposed PE has a counter and it can process up to 32 bit data. Inside a PE, there is a bit-serial multiplier that can compute up to 16 bit by 16 bit multiplication.

4.1.1 Overview of the processor element

Figure 12 shows an overview of the processor element. According to the config-

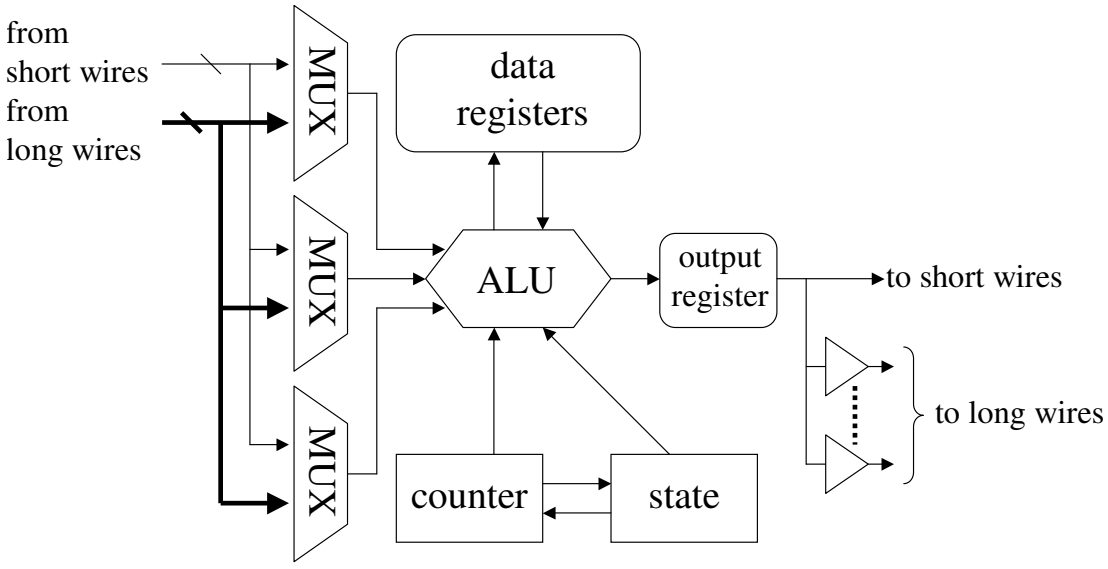

Figure 12. overview of the processor element

uration data, each MUX in Figure 12 selects inputs from the short and the long

wires. Then a MUX outputs the selected data to the ALU as in Figure 12. The ALU processes the three inputs from MUXs. Then outputs a result to the output register. The configuration data specifies the operation that the ALU performs. The output register outputs its data to the short and long wires. Each PE has dedicated short wires to the four adjacent PEs and IOEs. Therefor the data that the output registers keep is transfered to the short wires with no reservation. On the other hand, the data of output register is propagated to one of the long wires through a tristate buffer (triangular shapes in Figure 12). According to the configuration data, the assignment of each tristate buffer is determined. There are three entries of 16 bit data registers. The first data register stores constant value. The second data register is used for shift operations. When the multiplication operation, all data registers are used to store intermediate result and two inputs.

4.1.2 Configuration data format

Figure 13 shows the format of the configuration data for the PE. The configuration

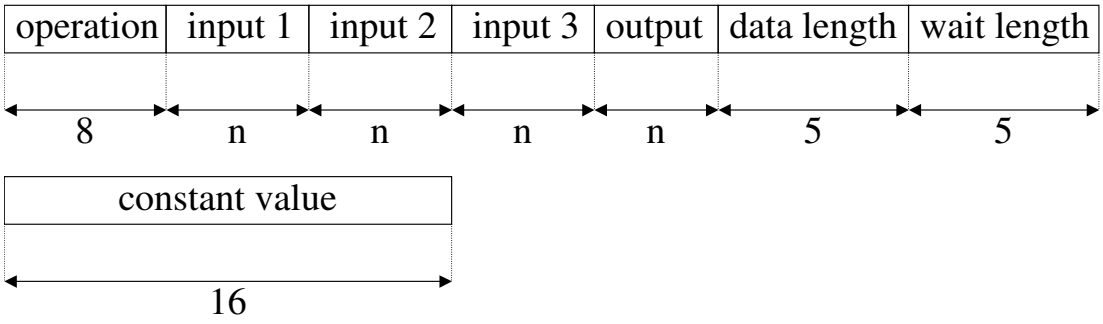

Figure 13. configure data format

data consists of eight fields: the operation field, three input fields, the output field, the data length field, the wait length field, and the constant value field. The operation field contains 8 bit length data. The operation of the ALU is specified by the value of the operation field. The three input fields are assigned to the three MUXs. The output field is decoded to a set of control signals, each of which is assigned to control port of a tristate buffer. Data length of the three input fields and output field depend on the distance and the step of the implemented chip. The data length field is 5 bit data that shows how many bits the PE processes.

The wait length field is 5 bit data and represents the number of clock cycles of interval between operations. A PE suspends the computation to synchronize the data flow by using above value. The constant value field is a 16 bit constant value data that is stored in a data register.

4.1.3 The state machine and the counter

A PE has the state that indicates what the PE does. Roughly classifying the states, the PE has the "running" and the "stop" states. When a PE is in a running state, the PE processes inputs and then outputs a result. When a PE is in a stop sate, the PE outputs high impedance signal defined in 3.2.

The detailed state is shown in Figure 14. Each state is classified more in

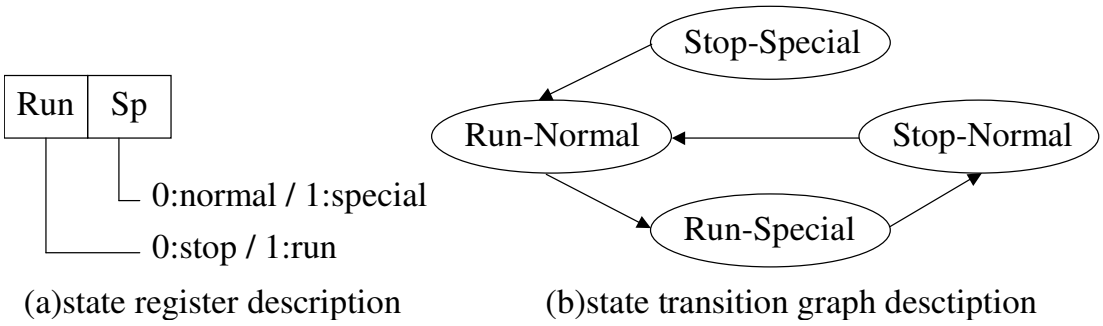

Figure 14. definition of the state. (b) is a example.

detail: normal and special states. Therefor a PE has 2 bit register to show the state; one is "run" bit and another is "special" bit. When the run bit is 1, the PE is in a running state. The PE is in a stop state when the run bit is 0. The stop-special state is the state immediately after reset. The PE keeps the stop-special state until input data from other PEs or/and IOEs arrive. The stop-normal state is the state a PE is in a stop state between operations. Note that as mentioned in Section 2.2, some PEs need to wait to synchronize with other PEs. Like multiplication and shift operations, there are operations in which the method of processing is changed in the first half and the latter half. The run-normal state indicates the first half of processing and the run-special indicates the latter half. Therefor the state transition of the PE depends on the operation field.

The counter is used to detect the end of the data or wait. The state machine changes its own state depending on the current state, the value in the operation field, and the counter's value. The counter begins count up from 0. When counter's value reaches the value in the data length field or the value in the wait length field, the counter resets the own value. Concurrently the state transition occurs.

4.1.4 Bit-serial Multiplier

A PE has the dedicated bit-serial multiplier. Then each PE can compute multiplication up to 16 bit by 16 bit data. Ienne et al. [12] proposed a bit-serial multiplier. The Ienne's multiplier has the following features.

- use n modules to compute n bit by n bit multiplication
- latency of the multiplier is only one clock cycle

Although Ienne's multiplier is useful, we did not employ it. The reasons are as follows.

- Each module outputs intermediate result, carry, and so on. However, our PE has only one output, and thus their multiplier is not suitable for our architecture.
- Since their multiplier needs n elements to implement, mapping process sometimes becomes complex.

Therefor, we designed a bit-serial multiplier dedicated for our architecture; it can perform multiplication by one element with one clock latency.

A 4 bit by 4 bit multiplication is shown in Figure 15. a_i and b_j means multiplicand and multiplier respectively. p_{ij} is a partial product of $(a_i \ \& \ b_j)$. c_{ij} is a carry that is propagated from the $(i-1)$-th stage; c_{0j} is 0. Then r_i is the i-th bit of the result. r_i is obtained by computing exclusive-OR of all partial products of the i-th column and all carry propagations of the i-th column. The next c_{ij} are obtained by counting carry when adding all partial products and carry propagations of i-th column. Therefor the 4-bit-serial multiplier is realized by Figure 16. p_{ij} and c_{ij} of i-th column are input to ports p_i and c_i in the figure respectively.

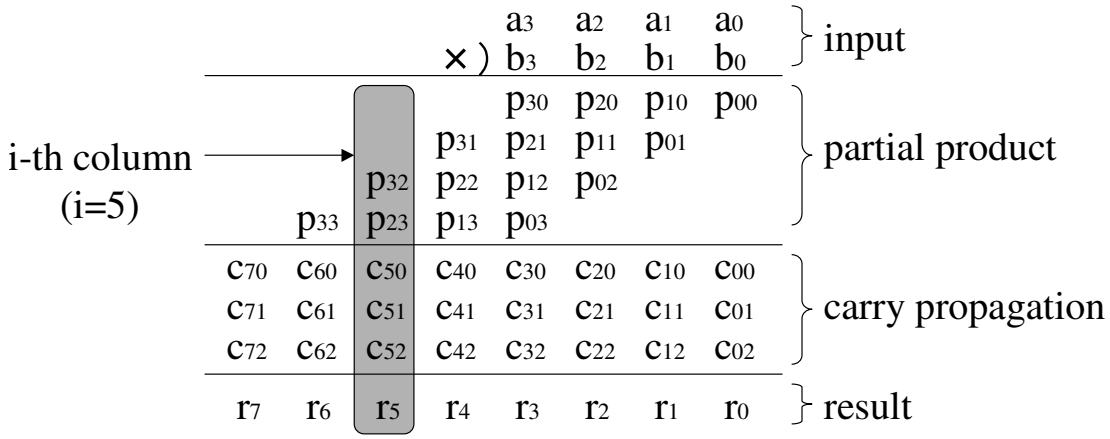

Figure 15. 4 bit multiplication

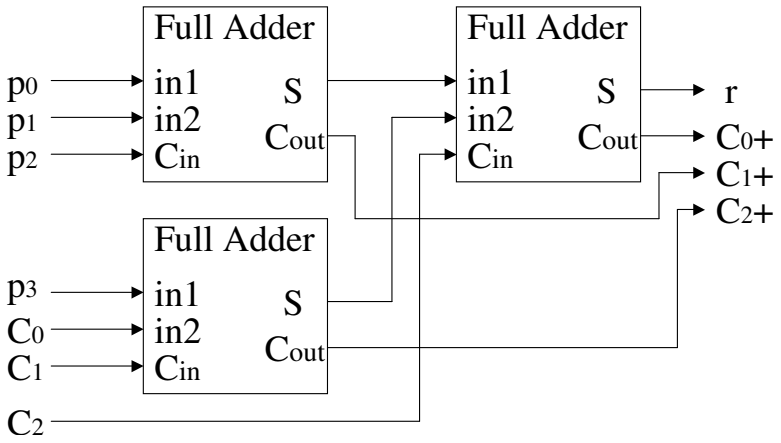

Figure 16. 4-bit-serial multiplier

The 4-bit-serial multiplier outputs r_i from port r and $c_{(i+1)j}$ from ports c_i+.

When a PE is in multiplication, the behavior of a PE is as follows. A PE stores two inputs to two data registers respectively. A PE reads all bit of two data registers that store input, then computes partial products of i-th column. The partial products are transfered to the multiplier. Concurrently, a PE reads all bits of another data register as carry propagations, then transfer them to the multiplier. Finally, a PE transfer the multiplier's output as a result to other elements. When the multiplication is finished, a PE resets all data registers.

Indeed, a PE has a 16-bit-serial multiplier, thus it can compute up to 16 bit by 16 bit multiplication. It cannot treat over 16 bit length data, but the multiplication method of Divide-and-Conquer [13] is available to solve this problem.

4.2 The I/O element

4.2.1 The features of the I/O element

The IOEs are the special element that mediate data transfer between the PEs and the controllers. The elements on the edge of the PE array is the IOEs and each IOE is connected to one of the controllers with the 16 bit shared bus. Figure 17 shows an overview of the IOE. The composition of the IOE is almost the same as the PE. The difference between the PEs and the IOEs is as follows.

- The IOE has 16 bit bus that is connected to the controller.
- The number of MUX is only one.
- The ALU has only two operations. One of them is input from the controller and output to the PEs. The other is input from the PEs and output to the controller.
- There are eight entries in 16 bit data registers.

PEs/controllers cannot transfer data to controllers/PEs without using an IOEs. As mentioned earlier, to make our architecture scalable, data transfer between the PEs and the controllers are restricted to be performed only through the IOEs. Without this restriction, the two problems occurs since all PEs and all controllers are connected. One of them is that wiring area would occupy most

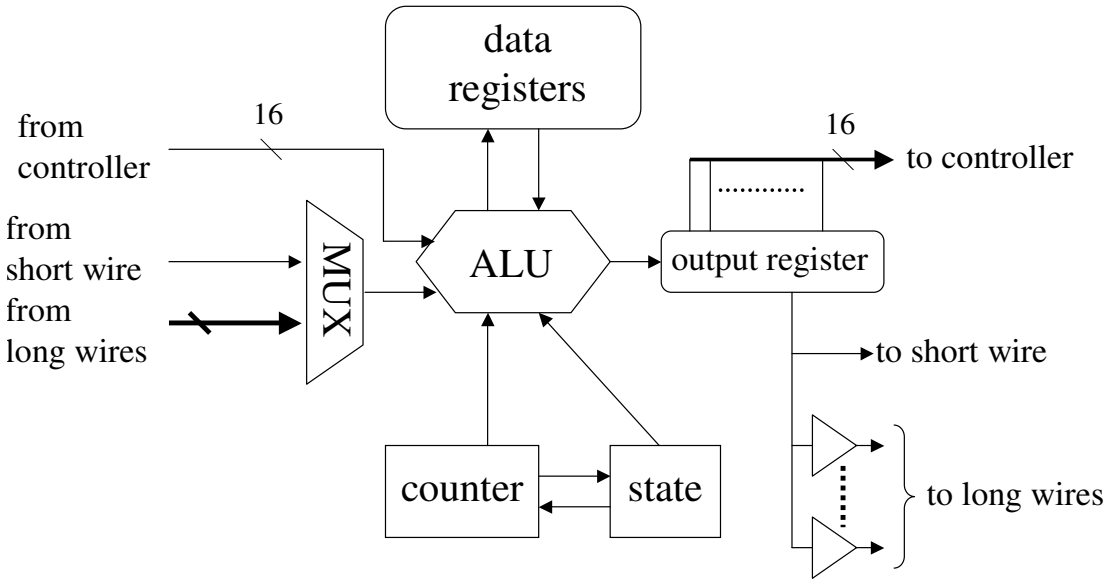

Figure 17. overview of the IOE

area of a chip. Therefor it becomes difficult to increase the number of elements. The other problem is that wiring delay becomes larger. This problem prevent the proposed architecture from working at high clock frequency.

4.2.2 Communication with the PEs and the controllers

Now we explain how to transfer the data to the PEs or to the controllers. Data transfer between the IOE and the PE is done by serial communication. So the protocol of data transfer between the IOE and the PE is the same as the protocol of data transfer between PEs.

On the other hand, the width of data transfered between an IOE and an controller is 16 bit. This data communication is realized by using 16 bit shared buses that connect the controllers and the IOEs. When the strobe port of an IOE is asserted and the RW port of the IOE is high level, the IOE stores the data from the controller. When the strobe port of an IOE is asserted and the RW port of the IOE is low level, the controller stores the data from the IOE.

The controllers convert the data length because host processor treats multiple of 8 bit data but the PEs and the IOEs always do not. When the width of data

from the IOEs is less than 16 bit, the controllers expand the data to 16 bit. The controllers expand the data to 8 bit when the IOE transfer the data whose width is less than 8 bit. Then the controller sends the expanded data to the host processor.

To support serial and parallel data transfer in one IOE, an IOE has eight entries of 16 bit data registers and two pointer registers. An IOE uses data registers as a FIFO buffer. Two pointer registers are 3 bit width registers and used to indicate the head and the tail position of the data.

When an IOE loads the data from the controller, a data register addressed by the head pointer is overwritten by the data from the controller. Then the head pointer is incremented (see Figure 18 (a)). When an IOE stores the data to the controller, the IOE outputs value of a data register addressed by the tail pointer to the controller. Then the tail pointer is incremented (see Figure 18 (b)).

Figure 18. data transfer with the controller

When an IOE loads the data from the PE, a particular bit of a data register selected by the head pointer is overwritten as follows: First, the ALU reads a 16 bit value of a data register selected by the head pointer (the data flow (I) in Figure19 (a)). Next, the ALU modifies the bit, specified by the counter, of read data into the data bit from the PE (the data flow (II) in Figure19 (a)). Then the ALU stores overwritten data to a data register selected by the head pointer (the data flow (III) in Figure19 (a)). Finally, the head pointer is incremented. When an IOE stores the data to the PE, a data register selected by the tail pointer is loaded, and output a data bit specified by the counter to the PE. The tail pointer

is incremented finally (see Figure 19 (b)).

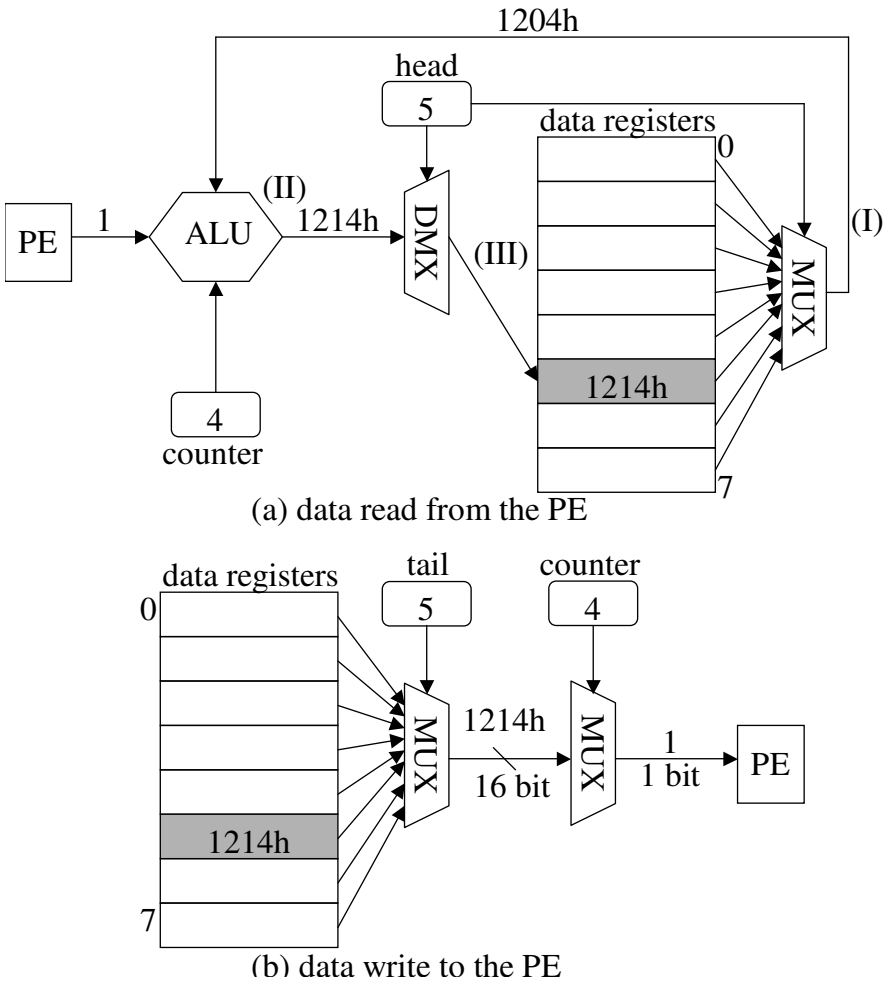

Figure 19. data transfer with the PE

5. Chip implementation

We implemented the proposed architecture on a chip to evaluate the chip area and clock frequency. The developing environment is shown in Table 1. We

Table 1. developing environment

Tool type	Tool name (vendor)	version
HDL	Verilog-HDL	—
simulator	Verilog-XL (Cadence)	2.13
synthesis	Design Compiler (Synopsys)	2000.11-sp4
layout	ApolloII (Avant!)	2000.2.3.4.0.2
process	ROHM 0.35μmEXD	3metal layer

implemented the PE and the IOE but did not implement the controller. This is because the design of the controller depends on the external interface and we did not specify the external interface.

The parameters of the PE is described in Table 2. The parameters of the IOE is the same as the PE's one. The maximum number of the long wires in a

Table 2. parameters for the PE

distance	6
step	1
selector length	5bit
decoder length	5bit
counter register length	5bit
delay register length	5bit

row/column (L_{max}) becomes 7 because the distance is 6 and the step is 1. Each direction (north, south, east, west) of each PE has L_{max} long wires and a short wire for input and a short wire for output. Therefor a PE needs a 32-1 selectors for input selection. For the output, a 28 bit decoder is needed since a PE outputs a result to the adjacent elements with no reservation. To represent the decimal

value 32 requires 5 binary digits. So a selector needs a 5 bit registers. Like a selector, the decoder needs a 5 bit registers. We configured the counter register for 5 bit so that a PE can process an up to 32 bit data. We configured the delay register for 5 bit so that a PE can wait up to 31 clock cycles.

The clock frequency of the PE and the IOE is $133MHz$. The area of the PE was $0.1681mm^2$ and the area of the IOE is $0.1444mm^2$. Then we could successfully implement a 7×7 element array on a $3mm \times 3mm$ chip.

6. Application mapping

In this section, the result of the application mapping to the proposed architecture is described. The DCT circuit is employed as a target application. First we mention the performance of the implementations by other methods, i.e., FPGA, MorphoSys, and software. We compare them with our architecture. Then, we explain our target application, DCT. We, then, show how we can map the DCT on to our architecture. Finally, we provide comparison results and some evaluation.

6.1 Targets to be compared with our architecture

We employ the FPGA, MorphoSys, and software implementation to compare the performance and the chip area with the proposed architecture. We implemented the proposed architecture on a chip by $0.35\mu m$ process technology. Since we would like to compare the potential of our architecture with other methods, we evaluated the performance of the comparison target with $0.35\mu m$ process technology as will be mentioned.

MorphoSys [14] is a coarse-grained reconfigurable hardware that is proposed by the University of California, Irvine. MorphoSys consists of 16bit PEs, context memory, frame buffer, and the tiny RISC processor (controller). The number of the PEs is 64 and mapping applications by using context switch. MorphoSys is fabricated by various process technologies. Among them, we chose one fabricated by $0.35\mu m$ process technology since we implemented the proposed architecture on a chip by $0.35\mu m$.

An FPGA is a fine-grained reconfigurable hardware. We estimated the performance and the number of elements for mapping application by referring commercial library for Altera's FPGA which is called MegaFunction [18]. Since Altera does not release the data about chip area of FPGA, we estimated the area of an element that will be shown in Table 3 by referring the FPGA that is fabricated by the same process technology we used [19].

For the software implementation, we could not prepare the CPU with $0.35\mu m$. Therefore, we used the CPU (Pentium II) with $0.25\mu m$ process technology. The environment for the software implementation is as follows: CPU is Pentium II 350MHz, Memory type is PC100 and it's capacity is 128MB, software is running

over Windows2000, and we used Borland C++ 5.5.1 to Compile the software.

6.2 Mapping DCT circuit to the proposed architecture

The discrete cosine transform (DCT) is a technique for converting a signal into elementary frequency components. It is widely used in image compression. Especially the 2D-DCT is used in JPEG, MPEG, and other image compression. In JPEG and MPEG, an image is divided into the set of the macro blocks that is 8×8 pixel array. Then 2D-DCT computation is done to the all macro blocks. Therefore the 2D-DCT is one of the highest cost processes.

The 2D-DCT circuit consists of first 1D-DCT, matrix transform, and second 1D-DCT circuit. They are connected serially. The data flow of the 2D-DCT circuit is as follows.

1. Each row/column of the macro block is input to the first 1D-DCT circuit.
2. The 1D-DCT circuit outputs a result of each row/column to the matrix transform circuit.
3. The matrix transform circuit sorts the row/column data to the column/row data.
4. The matrix transform circuit outputs each row/column of the transposed macro block to the second 1D-DCT circuit.
5. The second 1D-DCT circuit outputs each row/column of the macro block.

We mapped the 2D-DCT circuit on our architecture manually as follows: The mapped 2D-DCT circuit outputs a row/column result of the macro block every 30 clocks after the first result is output. This is because the mapped 2D-DCT circuit works as pipeline, so only the first result takes additionally clock cycles of circuit latency. The data length of each result is 15 bit. It takes 30 clocks to output a row/column result since the 2D-DCT circuit outputs 30 bit precision data as an intermediate result. Then the 2D-DCT circuit shrink the intermediate result to 15 bit data and output as a final result.

6.3 evaluation

We evaluated the chip area and the performance of each reconfigurable hardware architecture and the software. We evaluate the total costs for processes the $194,400$ macro blocks by each method. This value is the same amount of the MP@ML of MPEG2 ($720 \times 576 pixel/frame, 30 frame/sec$). We estimated the time to finish of the process.

The result is shown in Table 3. *Clock frequency* in the table is running speed of

Table 3. area, performance comparison

	CPU	FPGA	MorphpSys	Ours
clock frequency $[MHz]$	350	17.45	100	133
clocks/Macro Block	2,500,000	64	37	240
elements	1	4386	64	322
process time $[msec]$	2163	1069	107	525
chip area $[mm^2]$	118	267	180	60

each method. *Clocks/Macro Block* is how many clock cycles each method spends to process a macro block. *Elements* means the number of elements to map a 2D-DCT circuit. Process time is . *process time* is how many seconds *Chip area* is computed by $elements \times elementarea$. The terms that should be paid attention in the table are the process time and the chip area.

6.3.1 comparison with the software and the FPGA

Below we provide our observation about the comparisons. Comparing with the CPU (software) and an FPGA, the proposed architecture processes 2D-DCT computation faster and the chip area is smaller. We focus on an FPGA. The area of an element of an FPGA is one third of our architecture However it takes 13 times the number of elements. Thus, the necessary chip area of an FPGA is larger than our architecture. Moreover, the large number of elements makes mapping process more complex, while the proposed architecture only needs a simple mapper.

6.3.2 comparison with the MorphpSys

Our architecture processes the application slower than MorphpSys. However our architecture finishes computation in about $0.5sec$. This is sufficient to the real-time processing for MPEG2 encoding. The chip area of our architecture is smaller than MorphpSys. This is because the area of one element is smaller than MorphpSys although the necessary number of elements becomes larger for our architecture. As described above, the large number of elements make a mapper more complex. Our architecture needs five times as many elements as MorphpSys.

The DCT, the IDCT, and the motion estimation occupy almost computation time of MPEG encoder system. If user map a MPEG encoder system to the MorphoSys, application mapping with context switch is needed. This is a very complex problem. Furthermore, there is an overhead to switch process. On the other hand, whole MPEG encoding system can be mapped to our architecture when the implemented chip has the sufficient number of elements. Then the mapping process becomes simpler and there is no overhead. This advantage is obtained because our architecture can increase the number of the elements easily unlike the case of MorphoSys.

7. Conclusion

7.1 Features of the proposed architecture

In this paper, we present our new reconfigurable hardware architecture for processing the bit-serial data paths. The features of the proposed architecture are high clock frequency, low chip area, and scalability. The three ideas are employed to obtain these features. First, we employed the coarse-grained architecture and the precision of its processor element is one bit. Next, we proposed new wiring architecture with two parameters (the distance and the step) to obtain flexible and low area wiring resources. Finally, we limited the connection between the elements and the controllers. Then the proposed architecture can increase the number of the elements easily.

We implemented a proposed architecture on a chip to estimate the clock frequency and the area per element. When the $distance = 6$ and the $step = 1$, each element runs at $133MHz$ and its area is $0.1681mm^2$ using ROHM $0.35\mu m$ EXD process technology.

We mapped the DCT circuit to the proposed architecture and compare the performance and the area with other reconfigurable hardwares and software. Then our architecture processed the DCT faster than the software and the FPGA, and slower than the Morphosys. However our architecture has sufficient performance to process real time MPEG2 MP@ML encoding. The chip area for mapping application was the smallest in the four comparison objects.

7.2 Future works

We proposed, implemented, and evaluated the architecture. Then, we have some future works as follows. The first issue is to map other applications. If the proposed architecture shows advantages in other applications, it would become more valuable reconfigurable architecture. The next issue is how to configure the number of elements, the distance, and the step. Whereas a designer can freely set these parameters, this issue might occur. One way to solve this issue is to provide various type of evaluation functions for different objects. Then designers use these functions appropriately. Another issue is developing the mapping tool

for the proposed architecture. The proposed architecture is the coarse-grained one and has an affinity to the high level synthesis. Moreover the proposed architecture does not have the context switch, so mapping process become simple. Thus, the following mapping processes are sufficient for our purpose, and to develop such a mapper is one of our future works.

1. Analyze source code which is written in C, Pascal, and so on. Then make a CDFG.
2. Analyze the data flow timing of the CDFG. Then insert delays to synchronize the data input to the nodes.
3. Assign each node of the CDFG to the each PE of the target chip.
4. Assign long wires. If the lack of the wiring resources is occurs, go back to the previous processes.

Developing a simulator for the proposed architecture is also a future work.

Acknowledgements

I would like to express my sincere appreciation to Professor Katsumasa Watanabe of Nara Institute of Science and Technology for his continuous guidance and encouragements for this research.

I would also like to express my appreciation to Professor Hideo Fujiwara of Nara Institute of Science and Technology for his helpful guidance and comments as a supervisor.

I would also like to express my appreciation to Associate Professor Shigeru Yamashita of Nara Institute of Science and Technology for his helpful suggestions and accurate criticisms.

I would also like to express my appreciation to Research Associate Masaki Nakanishi of Nara Institute of Science and Technology for his helpful advise about implementing and designing architecture.

Thanks are due to all members of the Professor Watanabe's Laboratory for their discussions and helpful supports.

References

[1] W M Smith and B Hutchings, *Reconfigurable Architectures: The Road Ahead*, In Reconfigurable Architectures Workshop, Geneva, Swizerlang, 1997, pp.81-96.

[2] S. Brown and J. Rose, *Architecture of FPGAs and CPLDs: A Tutorial*, IEEE Design and Test of Computers, Vol.13, NO.2, Nov.1996, pp.42-57.

[3] S Hauck, *The Roles of FPGAs in Reprogrammable Systems* Proceedings of the IEEE, Vol. 86, No. 4, April.1998, pp.615-639.

[4] J R Hauser and J Wawrzynek, *Garp: a MIPS processor with a reconfigurable coprocessor* 5th IEEE Symposium on FPGA-Based Custom Computing Machines, Apr.97, pp.12-21.

[5] G. Lemieux and D. Lewis, *Design of Interconnection Networks for Programmable Logic*, Kluwer Academic Publishers, 2004.

[6] H Amano and A. ouraku and K Anjo, *A Dynamically Adaptive Switching Fabric on a Multicontext Reconfigurable Device*, 13th International Conference, FPL2003, Sep.2003, pp.161-170.

[7] L Caglar and B Salefski, *Re-Configurable Computing in Wireless*, 38th Conference on Design Automation, Jun.2001, pp.178-183.

[8] IPFlex Inc., *http://www.ipflex.com*

[9] M. Katayama and H. Kai and J. Yoshida and H. Yamada and K. Shiomoto and N. Yamanaka, *10Gb/s Firewall System using Reconfigurable Processors*, 4th Reconfigurable system workshop, Sep. 2004, pp.67-72.

[10] A Ohta and T Isshiki and H Kunieda, *A New FPGA Architecture for High Performance Bit-Serial Pipeline Datapath*, IEICE Trans. Fundamentals, Vol.E83-A, NO.8, Aug.2000, pp.1663-1672.

[11] N Ohsawa and O Sakamoto and M Hariyama and M Kameyama, *Design of a Field Programmable VLSI processor Based on Bit-Serial-Pipeline Architectures*, IPSJ SIG Technical Report, 2003-SLDM-111, Oct.2003, pp.145-149.

[12] P. Ienne and M. A. Viredaz, *Bit-Serial Multipliers and Squarers*, Computers IEEE Trans, Vol.43, NO.12, Nov.1994, pp.1445-1450.

[13] Behrooz Parhami, *Computer Arithmetic Algorithms and Hardware designs*, Oxford University Press, 2000.

[14] H. Singh and M. Lee and G. Lu and F. J. Kurdahi and N. Bagherzadeh, *MorphoSys: An Integrated Reconfigurable System for Data-Parallel Computation-Intensive Applications*, Computers IEEE Trans, Vol.49, NO.5, May. 2000, pp.465-481.

[15] T Miyamori and K Olukotun, *REMARC: Reconfigurable Multimedia Array Coprocessor*, 6th International Symposium on Field-Programmable Gate Arrays, Feb. 1998, pp.261.

[16] Xilinx Home Page, *http://www.xilinx.com/*

[17] Altera Home Page, *http://www.altera.com/*

[18] Altera FPGA's MegaFunction Data Sheet: Discrete Cosine Transform, *http://166.111.64.217/datasheet/altera/sb/sb009.pdf*

[19] S. Kimura and T. Horiyama and M. Nakanishi and H. Kajiwara, *Folding of Logic Functions and Its Application to Look Up Table Compaction*, Proc. on ICCAD 2003, Nov. 2002, pp.694-697.