Bit-Serial Pipeline Synthesis and Layout for Large-Scale Configurable Systems *

Tsuyoshi Isshiki†, Wayne Wei-Ming Dai‡ and Hiroaki Kunieda†

† Tokyo Institute of Technology, Electrical and Electronics Engineering
2-12-1 Ookayama, Meguroku, Tokyo 152, Japan

‡ University of California at Santa Cruz, Computer Engineering
Applied Sciences Building, Santa Cruz, CA 95064

Abstract— In this paper, we present our datapath synthesis and layout tools which are targeted toward *large-scale configurable systems* with the logic capacity of up to millions of gates which consists of an easy design entry using C++, customized bit-serial circuit library for SRAM-based FPGAs, bit-serial pipeline circuit generator, and a circuit partitioner.

I. Introduction

Large-scale configurable systems composed of a number of FPGA devices have been developed by many research groups which demonstrate the capability of high-performance computing in various areas such as signal and image processing, data base search, pattern recognition, encryption, and embedded real-time systems. The main features of these configurable systems are the drastic speed up achieved by the highly parallelized hardwired computations whose architecture is strictly customized to a specific application, and also the large flexibility of the SRAM-based FPGAs for implementing a wide variety of applications.

There are a number of challenging problems in developing such a large-scale configurable system, especially on the software environment. We have been working on developing such a software system for the large-scale configurable system. Our focus has been on the following subjects:

Design entry on C++ Since configurable systems are not intended to be used by expert hardware engineers but by the software-oriented application designers and researchers, we have developed a C++ design entry system which allows the designers to enter their designs on the algorithm-level by means of *difference equations* which is widely used in discrete time system specifications.

Bit-serial circuit library We have developed a collection of bit-serial operator circuits such as adder, subtracter, multiplier, shifter, scaler, rounder and saturater. Because of the bit-serial architecture, all the critical paths are minimized to a single 5-input look-up table, and thus capable of functioning at a very high clock speed.

Bit-serial pipeline synthesis The algorithm captured by the C++ design entry system is mapped directly onto the pipelined hardware. In other words, every distinct operations are mapped onto distinct bit-serial circuits, and the data-flow is pipelined at the word-level as well as the bit-level.

Circuit partitioning The bit-serial pipeline circuit is partitioned into subcircuits in order to fit the target FPGA device. We have developed a partitioning algorithm based on Fiduccia and Mattheyeses' bipartitioning heuristic which maximizes the logic utilization while minimizing the IO utilization.

In this paper, we will introduce our work on these topics in some detail, and devote the rest of the paper on the layout results which exhibit some drastic improvement in device utilization, and also its implications of the effect on the future FPGA architecture and also on standard-cell designs.

II. C++ Programming for Hardware Design Capture

We have developed a design capture tool using C++ as the input. Unlike existing hardware description languages such as Verilog and VHDL, the designers are merely required to describe their design at the algorithm-level in terms of difference equations.

1. The C++ code is compiled by the standard GNU C++ compiler (`g++`), therefore the code can include any type of expressions supported by C++. Existing hardware design tools using various high-level programming languages all have dedicated compilers where a

*This work is supported in part by ARPA under ONR Grant N00014-93-1-1334, and also by CAD21 Research Body of Tokyo Institute of Technology.

ASP-DAC '97
0-89791-851-7$5.00

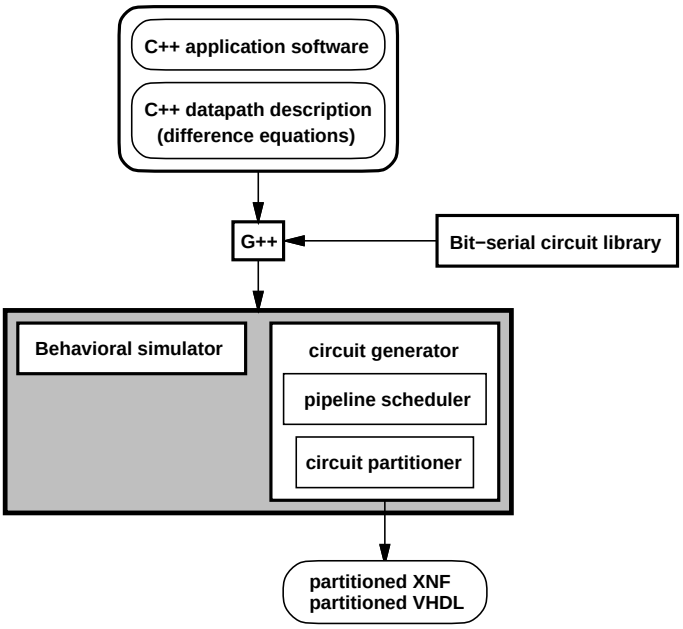


Fig. 1. C++ capture system compilation flow.

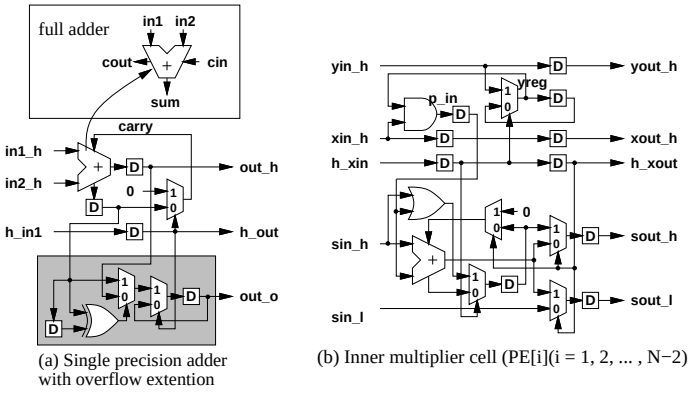
```
1D_FIR(int & in, int & out, Interface & bus, Single x[8],
Single & y, Double a[8], double coef[8])
{
  set_signal_precision(16);
  // Setting the signal word length to 16 bits

  for(int j = 0; j < 8; j ++){
    if(j == 0){
      x[j] = bus.read(in); // x[0] is the data input
      a[j] = x[j] * coef[j];
    }
    else {
      x[j] = delay(x[j - 1]); // x[j] is the delayed by
      // one sampling period from x[j - 1]
      a[j] = x[j] * coef[j] + a[j - 1];
    }
  }
  y = round(a[7]); // conversion from Double to Single
  bus.write(out) = y; // y is the output data
}
```

Fig. 2. 1D FIR filter C++ description. The underlined codes refers to the hardware description.

```
1D_FIR_main(int in[], int out[], double coef[8])
{
  Interface bus;
  Single x[8], y;
  Double a[8];
  for(int i = 0; i < TMAX; i ++){
    1D_FIR(in[i], out[i], bus, x, y, a, coef);
    // calling the convoluter routine
    clocker(); // incrementing sampling clock for simulation
  }
}
```

Fig. 3. 1D FIR filter C++ description including the application software codes.



(a) Single precision adder with overflow extention

(b) Inner multiplier cell (PE[i](i = 1, 2, ... , N−2)

Fig. 4. Bit-serial operator circuits.

great deal of effort is needed in developing and maintaining these tools. The compilation flow is illustrated in Fig.1.

2. The computational algorithm is described in difference equations. This form of design specification is widely used in many of the High-Level Synthesis systems for applications on digital signal processing such as SILAGE.

3. Distinction between hardware operators, computations which are to be implemented on the configurable hardware, and software operators is made by the C++ compiler which examines the data types of the argument variables. Expressions with predefined *hardware class objects* are automatically translated into various hardware object functions which construct the hardware description.

4. By utilizing the *operator overloading* feature of C++, the C++ coding style for hardware description becomes very similar to ordinary C++ coding. An example of a C++ description of a 1D FIR filter is shown in Fig.2.

5. The C++ code is both a hardware simulator and a synthesizer. Software codes can reside within the hardware description codes for simulation purposes such as reading in test vectors or even creating one, adding break points, displaying the simulation results, or emulating the whole application entirely (Fig.3). This enables the designer to perform a thorough and easy design verification. This feature is also unique compared to other tools.

III. Bit-Serial Circuit Library

We have designed a set of bit-serial datapath circuits for implementing high-performance pipeline network [3][4] which is summerized below:

1. Four types of 2's complement bit-serial data format are provided:

TABLE I
CIRCUIT AREA AND OPERATION LATENCY OF THE VARIOUS BIT-SERIAL OPERATORS. XILINX 3000 ARCHITECTURE IS ASSUMED FOR CLB COUNTS. (SP = SINGLE PRECISION, DP = DOUBLE PRECISION, ESP = EXTENDED-SINGLE PRECISION)

	input	output	size (CLBs)	latency (cycles)
k-input parallel-serial	parallel	SP	$k(N+1)/2$	1
k-output serial-parallel	SP	parallel	$k(N+k)/2$	$N+1$
Add(I)	SP	SP	2	1
Add(II)	DP	DP	3	1
k-bit shift-right	SP, DP	DP	$\lceil 0.5k+2 \rceil$	$k+1$
k-bit shift-left	DP	DP	$\lceil 1.5(k+1) \rceil$	1
Multiplication	SP	DP	$5N-4$	$2N-1$
Scaling	SP	DP	$4N+2$	$2N+1$
Round-even	DP	SP	3	1
Saturation	ESP	SP	$2N+2$	$2N+1$

- *single precision* (sign bit, $N-1$ fraction bits, single data line).
- *double precision* (sign bit, $2N-1$ fraction bits, double data lines).
- *extended-single precision* (sign bit, N integer bits, $N-1$ fraction bits, double data lines).
- *extended-double precision* (sign bit, N integer bits, $2N-1$ fraction bits, triple data lines).

Double precision data formats are for data produced by $N \times N$ multipliers, and extended precision data formats are for accumulation operations which may create temporal overflows on the integer part.

2. A collection of bit-serial circuits are provided (Table I):
   - Adder (Fig4(a)), subtracter (all data precisions).
   - Multiplier (Fig4(b)), scaler (single precision inputs, double precision output).
   - Rounder ((extended) double precision input, (extended) single precision output).
   - Saturater (extended single precision input, single precision output).
   - Sampling delay (all data precisions).
   - IO interface for system bus and memory devices (single precision).
3. Each circuit was designed specifically for Xilinx 3000 architecture using its CLB primitives. Special attention has been paid to the circuit implementation so that all the bit-serial circuits can be mapped onto the LUT-based logic blocks and can operate at the highest speed. They are all implemented in a single-level LUT logic where the critical path delay is only one LUT plus the flip-flop clock setup delay excluding the routing delay.
4. We have compared our bit-serial multiplier against several bit-parallel multipliers on Xilinx XC4000 FPGA architecture. The comparisons on circuit size, sampling period and AT complexity is shown in Fig.5. We can see that bit-serial multiplier is as twice as slow as the Booth's multiplier, but has significantly smaller circuit area. And as a result, the AT (area-time) complexity which is the product of the sampling period and circuit area is significantly lower on bit-serial multiplier than Booth's multiplier.

IV. Bit-Serial Pipeline Synthesis

Applications in digital signal processing and image processing often have the same computational structure in which a set of computations are applied continuously on incoming stream of data. This type of computation can be most efficiently implemented by pipelining the individual operators. We have developed a bit-serial pipeline synthesis tool which automatically synthesizes this pipeline datapath using our bit-serial circuit library. The basic automated steps are described below.

Circuit Generation After the design is captured by C++ as described earlier, bit-serial circuit modules are created for each distinct arithmetic operation by calling the circuit library and the primary circuit netlist is constructed.

Minimum Sampling Period Calculation Sampling period of the bit-serial pipeline needs to be at least as long as the word size N. Also, if there are loops in the datapath, the sampling period needs to be long enough so that all the loop paths have positive weights. This can be done by iteratively solving the *single-source longest paths problem* to find the minimum sampling period.

Pipeline Scheduling Optimization After the minimum sampling period is determined, the pipeline network is synchronized by inserting retiming registers in order to equalize the latency on every path in the network. At the same time, the number of retiming registers are minimized in order to reduce the hardware overhead. This pipeline scheduling optimization can be formulated as a linear programming problem and known to be efficiently solved by *simplex method* [1][2].

V. Circuit Partitioning

After the bit-serial pipeline network has been synthesized, we have to deal with the problem of partitioning the pipeline network into subgroups in which each can be

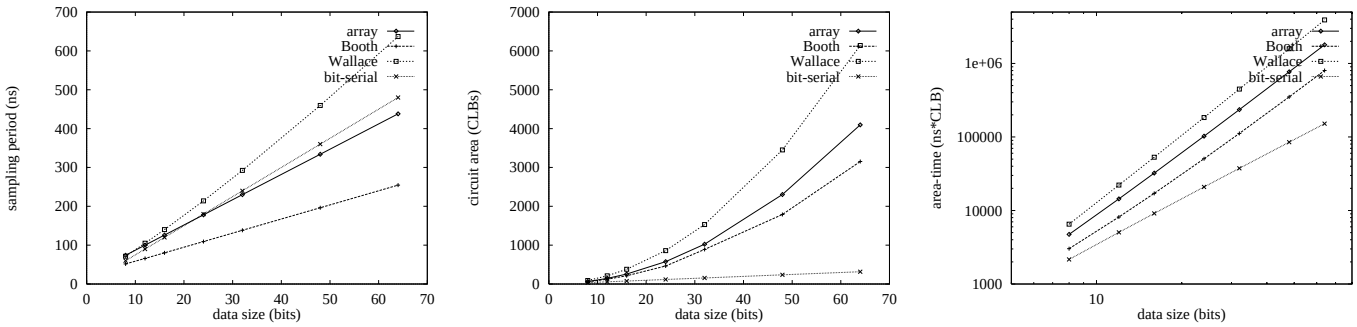


Fig. 5. Sampling period, circuit size and AT complexity of various multiplier circuits.

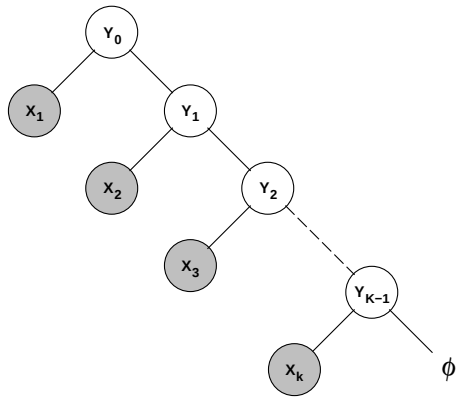


Fig. 6. Recursive bipartitioning method to obtain a K-way partition. Capacity constraints are imposed only to the shaded leaf partitions.

TABLE III
RENT'S PARAMETERS FOR BIT-SERIAL CIRCUITS.

design	γ	k
N-12-14-4	0.3724	4.916
N-8-8-4	0.3264	5.878
1D-FIR30-II	0.2218	6.018
2D-FIR8x8	0.3245	4.665
1D-IIR20-III	0.3264	5.878
adapt10T-7	0.3735	5.150
IDCT-I	0.3394	5.493

fitted inside a single FPGA. We have developed a simplistic multi-way partitioning method based on Fiduccia-Mattheyses bipartitioning heuristic [5] which maximizes the logic utilization while minimizing the IO utilization. Our partitioning algorithm is described below:

1. For a bit-serial pipeline network $HG(\overline{E}, V)$, let $Y_0 = V$. For $i = 1, 2, \cdots$, recursively partition the pipeline the network Y_{i-1} into X_i and Y_i (Fig.6):

$$\begin{array}{rcl} X_i \cup Y_i & = & Y_{i-1} \quad (i = 0, 1, \cdots) \\ X_i \cap Y_i & = & \phi. \end{array} \tag{1}$$

   We call $\{X_i | i = 0, 1, \cdots\}$ the *leaf partitions* and Y_i the *residue partition* after producing i leaf partitions : $V = (X_1 \cup X_2 \cup \cdots \cup X_i) \cup Y_i$.

2. Impose the size capacity and IO capacity constraints only to leaf partitions (X_i) which reflect the actual physical capacities of the FPGA :

$$\begin{array}{rcl} |X_i| & \leq & F_s \\ |X_i|_{IO} & \leq & F_{IO} \end{array} \tag{2}$$

   where F_s and F_{IO} are represents the size capacity and IO capacity of the target FPGA device, respectively, $|X_i|$ is the size of X_i, and $|X_i|_{IO}$ is the IO count of X_i.

3. Continue the above procedure until Y_{K-1} becomes empty. The K leak partitions essentially becomes the K-way partitioning solution $(X_1, X_2, \cdots, X_{K-1}, X_K)$ in which all the partitions satisfy the size capacity and IO capacity constraints.

In Table II, partition results of the various bit-serial designs developed by our bit-serial synthesis system are shown. We can see that the average logic utilization of each design is consistently well over 90%. In fact, most of the partitioned subcircuits are close to 100% utilization, except for the *last* leaf partition since there is little control over the size of the last partition. Considering the low IO utilization of 17% $\sim$ 30% and also the fact that *all* of these partitioned subcircuits were completely routed, we can observe the efficiency of bit-serial circuits in terms of layout, where the consumption of IO and routing resource is significantly low. In the next section, we conduct further analysis on the routability of the bit-serial circuits using Rent's rule.

VI. Routability Analysis for Bit-Serial Circuits

In analyzing the circuit structure to evaluate the area required for VLSI layout, Rent's rule is commonly used [8][6][7]. This rule defines the relationship between the average number of pins and the average number of logic circuits in a subcircuit. It is expressed as

$$P = k \cdot G^{\gamma} \tag{3}$$

TABLE II

Synthesis results of bit-serial pipeline networks. "N-12-14-4" is a digital neural network with 12, 14, 4 nodes on each layer. "N-8-8-4" is a digital neural network with 8, 8, 4 nodes on each layer. "1D-FIR-30-II" is a 30-tap 1D FIR filter. "2D-FIR8x8" is an 8×8-tap 2D FIR filter. "1D-IIR20-III" is a 20-tap recursive filter with 2 sampling period lookahead on the feedback loop. "adapt10T-7" is a 10-tap 1D adaptive FIR filter where the coefficient updating cycle delay is 7 sampling periods. "IDCT-I" is an Inverse-DCT circuit.

design	signal precision	# CLBs	# gates	# chips (XC3164A)	CLB util.	IO util.	critical delay	sampling frequency (@40MHz)
N-12-14-4	8	6964	127962	32	97.2%	30.3%	24.6ns	2.5MHz
N-8-8-4	8	3082	56465	14	98.3%	30.7%	24.6ns	2.5MHz
1D-FIR30-II	16	2004	35813	9	99.4%	17.4%	20.3ns	1.25MHz
2D-FIR8x8	8	2272	39957	11	92.2%	21.1%	25.0ns	2.5MHz
1D-IIR20-III	16	2132	41165	10	98.1%	22.9%	23.4ns	1.25MHz
adapt10T-7	8	871	16036	4	97.5%	27.3%	22.8ns	2.5MHz
IDCT-I	16	1038	15909	5	90.4%	23.7%	23.2ns	1.25MHz

where P is the average number of external pins in a subcircuit, and G is the average number of modules in a subcircuit. k is the *Rent's constant* which has empirically been found to correspond to the average number of pins per module. γ is the *Rent's exponent* which ranges between 0 and 1. First discovered by E. F. Rent of IBM in the late 1960's, Donath [6] derived the same relationship from a stochastic model of hierarchical design process. From his model which assumes a two-dimensional layout, he has derived the average wire length r of the circuit using the Rent's parameters as

$$r \sim \begin{cases} f(\gamma) & (\gamma < 0.5) \\ \log G & (\gamma = 0.5) \\ G^{\gamma - 0.5} & (\gamma > 0.5) \end{cases} \quad (4)$$

where $f(\gamma)$ is a function independent of G. The intuitive explanation for this is that the case $\gamma = 0.5$ is the transition between planar and non-planar circuits, and the circuits whose Rent's exponent is lower than 0.5 can be placed such that all connections essentially lie between nearest neighbors with an average wire length being independent of G. For circuits where $\gamma > 0.5$, the wire length grows exponentially with G. Therefore, the Rent's exponent is a good indicator for estimating the amount of routing resources needed for the physical layout. Landman and Russo [7] performed an extensive study on large "real-life" circuits, and observed the Rent's exponent to be between 0.47 and 0.75. For FPGA layout synthesis, the Rent's exponent is very critical both in IO pin consumption and routability.

We have obtained the Rent's parameters for our bit-serial designs by applying multi-way partitioning with different size constraints. The method for deriving Rent's parameters is to plot the average CLB counts and the average IO counts per partition on the log-log scale and use linear regression to estimate the slope and the intercept. Note that Rent's rule on log-log scale forms a linear equation $\log P = \gamma \log G + \log k$. Figure 7 shows this plot for 12-14-4 digital neural network. The Rent's parameters are summerized in Table III. Compared to Landman and Russo's work where they reported the Rent's exponent to be between 0.47 and 0.75, our bit-serial circuits have a significantly lower Rent's exponent between 0.22 and 0.37. This implies that our bit-serial circuits are very highly routable circuits on two-dimensional plane. There have not been any studies on logic circuits which reveal such a low Rent's exponent for *real* applications. Our bit-serial circuit design and our bit-serial pipeline network synthesis strategy led to such an extremely routing-efficient circuit structure.

VII. Bit-Serial Pipeline Synthesis on Optimized FPGA Architecture and Standard Cell Designs

So far, we have discussed the important properties of bit-serial circuits in terms of layout. Their impact on the FPGA architecture is very important if we observe the relationship of routing channel density against chip area. As an example, let us consider the Xilinx LCA FPGA devices [10]. XC3000 series has identical routing channel density for all chip size : XC3020 which is composed of 8×8 logic block array has the same routing channel width as XC3090 which is composed of 20×16 logic block array. For those "real-life" circuits whose Rent exponent is over 0.5, it will become harder to route as the chip size increases. In other words, logic utilization tends to be lower for larger FPGAs because of the insufficient routing resource, which in fact has been pointed out by many users. XC4000 series handles this problem by increasing the routing channel density as the device size increases and therefore attempts to sustain high logic utilization even for large devices. Recently announced XC4000EX/XL series is the upgrade models of XC4000 aimed for delivering over 100k gate capacity where the routing resource is virtually doubled from the former XC4000 series devices in order to provide sufficient routing resource. It easy to imagine how much silicon resource is spent on routing and not on the actual logic.

Our routability analysis implies that since many of the bit-serial pipeline network have a Rent exponent of lower than 0.5, the required routing resource is *independent of*

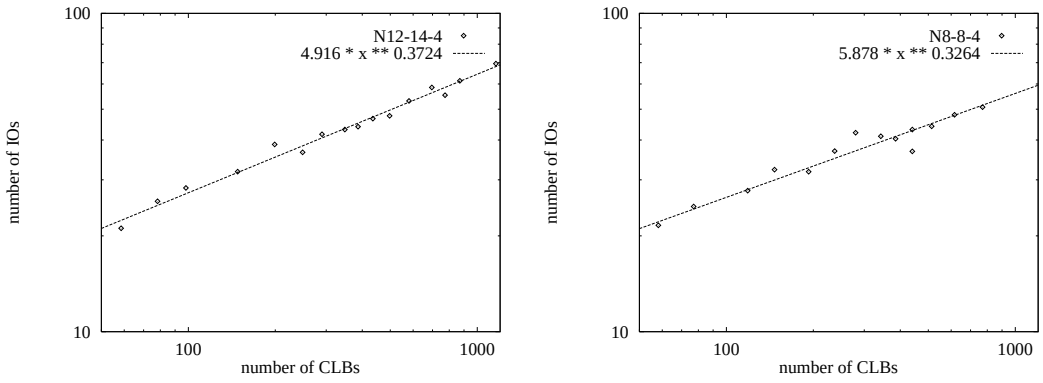


Fig. 7. Rent's exponent approximation for N12-14-4 and N8-8-4. The two axes are log-scale.

the circuit size, and therefore independent of the device as well. This means that if we are able to optimize the FPGA architecture strictly for bit-serial circuits, we would not need to increase the routing channel density, or at least not as drastically as recent Xilinx FPGAs, as the chip size grow. As a result, we can expect this new FPGA optimized for bit-serial implementation to have a much higher logic density than a general-purpose FPGA.

Also, because of the fine grain nature of the bit-serial circuits, it is possible to migrate these circuit library into the standard-cell design environment. Standard-cells are normally used only on control circuits which has the structure of so called "random logic", where there are no regularity in the pattern of logic connections. Layout of the datapath circuits are still done manually since automatic placement and routing tools have not yet matured. We believe that by using bit-serial architecture for the datapath circuits, it is possible for the automatic place and route tool to handle the layout of the datapath circuits as well.

VIII. Summary

In this paper, we have presented the work on our bit-serial pipeline synthesis system which is composed of C++ design entry, bit-serial circuit library optimized for Xilinx FPGAs, bit-serial pipeline synthesis and circuit partitioner for maximum logic utilization. Our routability analysis for a number of bit-serial designs shows that the routing density required for these circuits are significantly lower than the "real-life" circuits observed by others. We believe that by utilizing this bit-serial architecture, we will be able to redesign the existing FPGAs by properly allocating sufficient routing resource only for bit-serial implementation so that the logic density becomes significantly higher. Also, we believe that our bit-serial architecture can be adopted for standard-cell designs for ASICs as well.

Acknowledgement

Authors would like to thank the members of CAD21 Research Body of Tokyo Institute of Technology and members of FPMCM project of University of California at Santa Cruz for their suggestion and cooperations.

References

[1] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," *J. VLSI and Comput. Syst.* , vol. 1, no. 1, pp. 41-67. Oct. 1983.

[2] X. Hu, S. C. Bass and R. G. Harber, "Minimizing the Number of Delay Buffers in the Synchronization of Pipelined Systems," *IEEE Trans. Computer Aided Design*, pp. 1441-1449, Dec. 1994.

[3] T. Isshiki and W. W. -M. Dai, "High-Level Bit-Serial Datapath Synthesis for Multi-FPGA Systems," *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, Feb. 1995.

[4] Tsuyoshi Isshiki, "High-Performance Bit-Serial Datapath Implementation for Large-Scale Configurable Systems," *PhD Thesis, University of California, Santa Cruz*, March 1996.

[5] C. M. Fiduccia and R. M. Mattheyses, "A Lenear-Time Heuristic for Improving Network Partitions," *Proc. 19th ACM/IEEE Design Automation Conference*, pp.241-247, 1982.

[6] Wilm E. Donath, "Placement and Average Interconnection Lengths of Computer Logic," *IEEE Trans. Circuits and Systems*, pp.272-277, April 1979.

[7] B. Landman and R. Russo, "On a Pin Versus Block Relationship for Partitioning Logic Graphs," *IEEE Trans. Computers*, pp.1469-1479, 1971.

[8] L. Hagen, A. B. Kahng, F. J. Kurdahi, C. Ramachandran, "On the Intrinsic Rent Parameter and Spectra-Based Partitioning Methodologies," *IEEE Trans. Computer-Aided Design*, pp.27-37, Jan. 1994.

[9] Abba A. El Gamal, "Two-Dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits," *IEEE Trans. Circuits and Systems*, pp.127-138, Feb. 1981.

[10] "The Programmable Gate Array Data Book," Xilinx, 1995.