ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

IJIRSET
International Journal of Innovative Research in Science, Engineering and Technology

Volume 3, Special Issue 3, March 2014

2014 International Conference on Innovations in Engineering and Technology (ICIET'14)
On 21st & 22nd March Organized by

K.L.N. College of Engineering, Madurai, Tamil Nadu, India

Energy Aware Optimized Resource Allocation Using Buffer Based Data Flow In MPSOC Architecture

Achyut karthikeya.T.G[#1],Gowtham.M[#2],Gautam.R[#3],Selva kumar [#4],Janakiraman.N [#5]

K.L.N. College of Engineering, Sivagangai , Tamilnadu,India

ABSTRACT— This project presents a methodology for reducing interconnect resources in reconfigurable platforms for communication between processing units such as FPGA(Field Programmable Gate Array), MPSoC(Multiple Processor System On Chip) and NoC(Network-On-Chip). The evaluation of NoC architectures are NP-complete problem in the design of MPSoC. This proposed methodology implements the buffer based data flow architecture to reduce the overall design time and increase reconfigurability using dynamic virtual channel allocation. In this architecture the memory and bus allocation is shared to achieve high dynamic energy consumption and low interconnect-resource utilization. The energy requirement for memory and bus allocation is continuously monitored and controlled within certain limit. This entire architecture is implemented in Xilinx 12.2 System Edition Software and realized in Xilinx Virtex 5 XUP FPGA kit. This proposed methodology expects the better optimization results in terms of energy compare to previous work. The mathematical representation of proposed work can be done using DAG (Data Acyclic Graph) and CDFG (Control Data Flow Graph) methods.

INDEX TERMS—Buffer-based dataflow, interconnect resource reduction, field-programmable gate array (FPGA), reconfigurable architecture in MPSoC Architecture.

I.Introduction

For complex designs, the greatest impact on performance,costs, and functionality can be made at the architectural level [1]. Top-down design methodologies proceeding from the architectural to lower levels, are thus becoming popular. This design trend relaxes redesign efforts at higher levels of system assembly [2], and most designers endeavor to simplify the complex system under design from an early stage. coarse-grained dataflow graphs is widespread and is adopted in many high-level design frameworks [3]. When a dataflow graph is synthesized in a fine-grained reconfigurable platform such as field-programmable gate arrays (FPGAs), the interconnects of the dataflow consume more resources than the functional units do [4]. In addition, the implementation of multiplexors for data routing is very expensive in typical FPGA design [5]. In [6], Conget al.presented a method that can bind a dataflow graph to a parameterizable register-file microarchitecture in order to reduce the interconnect and multiplexor complexity. However, this approach is limited in terms of reducing the number of multiplexors because the microarchitecture still uses multiplexors for data routing.

In this paper, we use the techniques developed for bufferbased dataflow representations [7], in which buffers are inserted between the nodes of a dataflow graph. This technique is applicable to a subtype of synchronous dataflow (SDF) graphs [8] where the number of samples produced and consumed is always unity since the input and output data of the buffers have the same size. Due to the buffers, data transfers between nodes can be separated from the functional executions of nodes [9]. Thus, a buffer-based dataflow can increase the reusability and reconfigurability of the interconnects between nodes. Moreover, the data transfers between nodes can betimely multiplexed with the buffer controller parametersat the level of a dataflow. This allows the designer to use tristate buffers for efficiently routing data transfers; the tristate buffers are activated by the timely multiplexed signals represented with buffer controller parameters. Since the buffers in a buffer-based dataflow represent the interconnects between nodes, we propose a methodology for sharing buffers in a buffer-based dataflow in order to reduce the number of the interconnects required. Buffer sharing is a well-known technique to reduce system memory in register transfer level (RTL) designs. The register allocation problem occurring in compiler design is also related. Existing buffer-sharing techniques merge those buffers that have non-overlapping lifetimes into one [10]–[13]. These approaches are based on buffer lifetime analysis such that they are limited to control the lifetimes of buffers provided by a target architecture. In this paper, buffers are inserted in order to parameterize data transfers through edges at the level of a dataflow graph. Sharing buffers and buses can thus be manipulated with the parameters of buffer controllers to reduce interconnect resources.One of the most popular simplifying methods is to represent a target application as a dataflow graph. Especially,modeling digital signal processing (DSP) applications through

II.BACKGROUND

A. Data Flow Representation

Most real-time signal processing algorithms consider sets of data as frames. Such systems possess two unique characteristics of execution. First, they can be represented as dataflow graphs which represent data dependency between processing blocks (nodes). Second, each node in the dataflow executes a set of data elements per frame (iteration).There are few existing dataflow models such as synchronous dataflow, scalable synchronous dataflow, cyclo-static dataflow, and Boolean dataflow. For clarity and suitability, we use buffer based data flow. This type of data flow model helps in reconfigurable platforms by supporting dynamic buffer allocation. The buffer based data flow is given in the fig. 1.

In Fig. 1(a), a node can be regarded as the source and destination of data. As shown in Fig. 1(b), this source-destination relationship can be isolated by inserting buffers between nodes.

By separating the relationships between nodes, nodes only represent their functionality. The isolation also facilitates to reconfigure the overall system.

In a buffer-based dataflow graph, inserting a buffer to an edge represents delivering a data frame from its source to destination. Thus, the size of data frames appearing at the input port of a buffer is identical to the size of data frames at the output port of the buffer. Furthermore, while a source node is writing data to a buffer, the corresponding destination node is able to read data from the buffer. Therefore, buffers in a buffer-based dataflow can be realized as the dual-port memory which allows simultaneous writing and reading access.

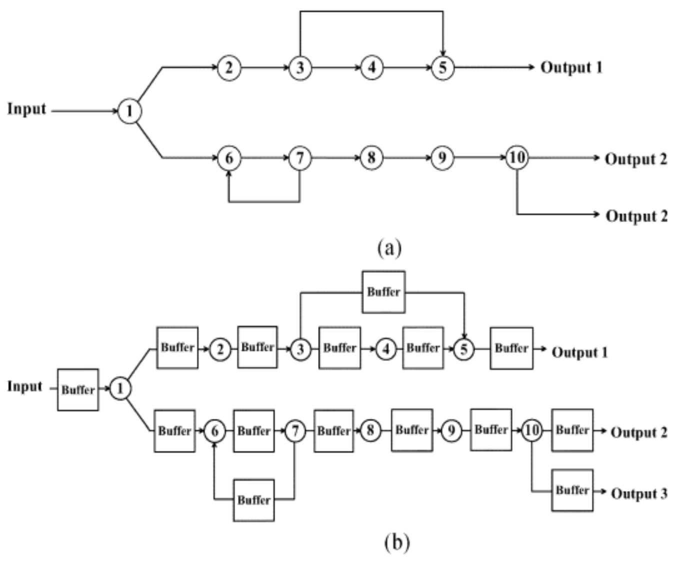


Fig. 1. Buffer-based dataflow derived from a dataflow by buffer insertion. (a) A dataflow graph. (b) Buffer insertion.

B. Notations

Let $BC_{i,j}$ denote the buffer between the producer node *i* and the consumer node *j*.

C. Primary parameters and limitations

Logic latency (L_i) , Write offset($nw_{i,j}$), Read offset($nr_{i,j}$) , Block size($M_{i,j}$) , and Delay factor($D_{i,j}$). Logic latency (L_i) is the latency of node *i*.

The Write offset ($nw_{i,j}$) represents the difference between reading data from the previous buffer and writing data to the current buffer without considering (L_i) .

The Read offset ($nr_{i,j}$) is the offset from the start of writing data to $BC_{i,j}$ to the start of reading data from when the writing speed of node *i* and the reading speed of node *j* are matched.

If the writing speed of node *i* is slower than the reading speed of node *j* , node *j* does not read valid data from $BC_{i,j}$. The delay factor ($D_{i,j}$). is to represent this rate mismatch between nodes *i* and *j*. The block size ($M_{i,j}$) characterizes the data size generated by node *i*. It also determines the maximum storage requirement of $BC_{i,j}$. The unit of all the parameters described above is the unit cycle of the target platform used.

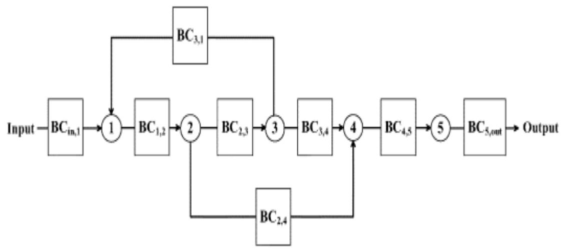


Fig. 3. Buffer-based dataflow derived from Fig. 2.

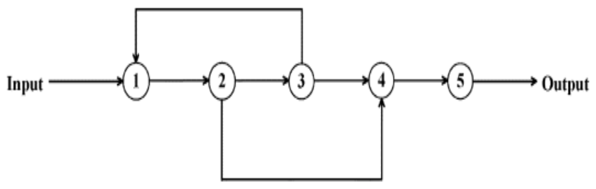
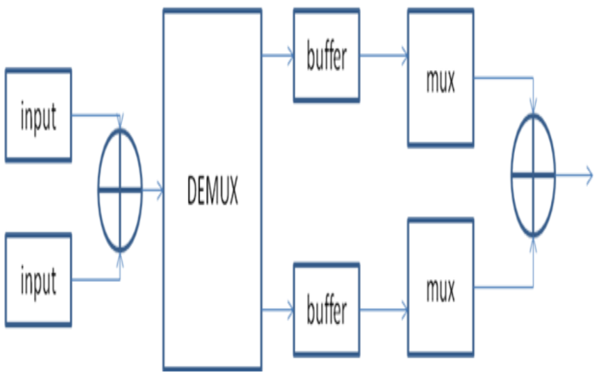


Fig. 2. Example dataflow.

III. Approach And objective

The approach is based on the application oriented. The main objective is to reduce the delay and power optimization between the communication links of processing units.ALU processing unit is considered as the processing unit. As a part of ALU processor here ripple carry adder is used. The purpose of usage of the Ripple carry adder is the advantage of reduced power. But in other hand the speed is low when comparing to power consuming and fast Look ahead carry.



IV.Proposed methodology

1) The power and delay

A. Design of a Full-Adder (FA)

The logic equations of a 1-bit FA with its signal are:

$S = A \oplus B + Ci$ (1)

$Co = AB + Ci\ (A + B)$ (2)

$P = A \oplus B$ (3)

Because Co will be an input of the next FA (in the worst case, the carry would propagate through all FAs), so it is on the critical path that is needed to be quickly computed. We adopt the mirror circuit for computing Co [6] as shown in Fig. 2(a).This circuit calculates the inverted value Co. Its transistors are sized to have the worst-case pull up and pull down times equivalent to a minimum-size inverter with switching threshold at 0.5VDD
as shown in Fig. 2(b)1.

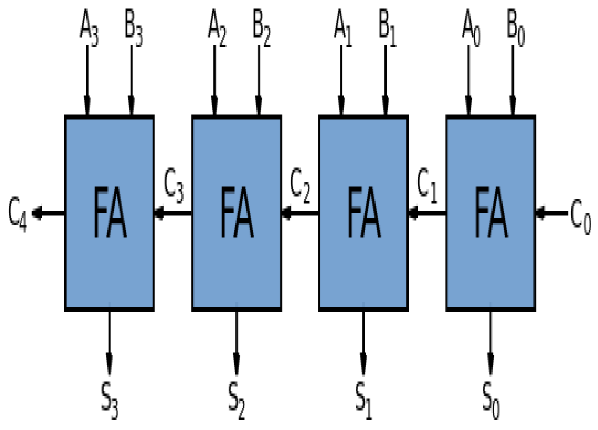
This mirror circuit is fast because the inverted Co is available as soon as A, B and Ci
occur. Its delay is equal to only one
minimum-size inverter. Besides that, the pull up and pull down networks only have stack of two transistors that should give a good Ion
/Io f f ratio [2].

The logic circuits for S and P are shown in Fig. 3(a) using XOR gates (Fig. 3(b)). Delay of P and S are one and two XOR gate delays, respectively. Since logic S is not on the critical path, its delay is negligible in compared to the delay of whole 32-bit adder (which is mainly determined by the carry propagation path).

B. Ripple-Carry Adder

A ripple carry adder is simply several full adders connected in a series so that the carry must propagate through every full adder before the addition is complete. Ripple carry adders require the least amount of hardware of all adders, but they are the slowest.

The following diagram shows a four-bit adder, which adds the numbers A[3:0] and B[3:0], as well as a carry input, together to produce S[3:0] and the carry output.



C. Propagation Delay in Full Adders

Real logic gates do not react instantaneously to the inputs, and therefore digital circuits have a maximum speed. Usually, the delay through a digital circuit is measured in gate-delays, as this allows the delay of a design to be calculated for different devices. AND and OR gates have a nominal delay of 1 gate-delay, and XOR gates have a delay of 2, because they are really made up of a combination of ANDs and ORs.

A full adder block has the following worst case propagation delays:

- From A_i or B_i to C_{i+1}: 4 gate-delays (XOR → AND → OR)
- From A_i or B_i to S_i: 4 gate-delays (XOR → XOR)
- From C_i to C_{i+1}: 2 gate-delays (AND → OR)
- From C_i to S_i: 2 gate-delays (XOR)

Because the carry-out of one stage is the next's input, the worst case propagation delay is then:

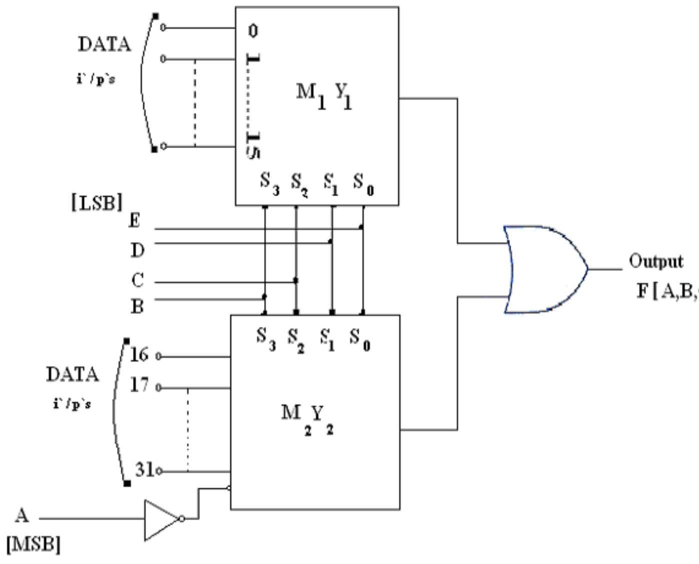
- 4 gate-delays from generating the first carry signal ($A_0/B_0 \rightarrow C_1$).
- 2 gate-delays per intermediate stage ($C_i \rightarrow C_{i+1}$).
- 2 gate-delays at the last stage to produce both the sum and carry-out outputs ($C_{n-1} \rightarrow C_n$ and S_{n-1}).

So for an n-bit adder, we have a total propagation delay, t_p of:

$$t_p = 4 + 2(n-2) + 2 = 2n + 2$$

This is linear in n, and for a 32-bit number, would take 65 cycles to complete the calculation. This is rather slow, and restricts the word length in our device somewhat. We would like to find ways to speed it up.

D. MUX AND DEMUX



- A demultiplexer has
  - N control inputs
  - 1 data input
  - 2^N outputs
- A demultiplexer routes (or connects) the data input to the selected output.
  - The value of the control inputs determines the output that is selected.

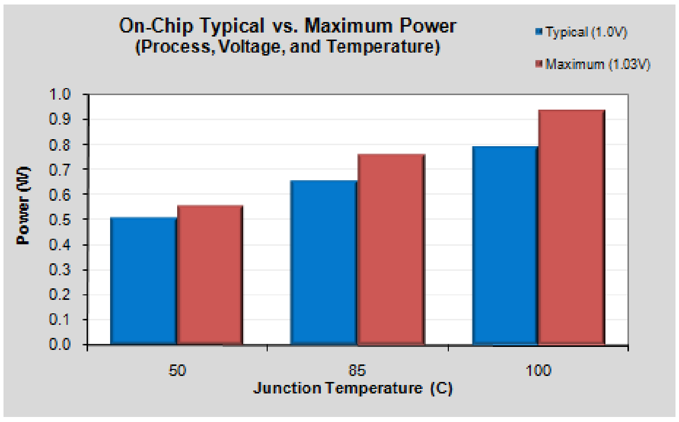
A demultiplexer performs the opposite function of a multiplexer

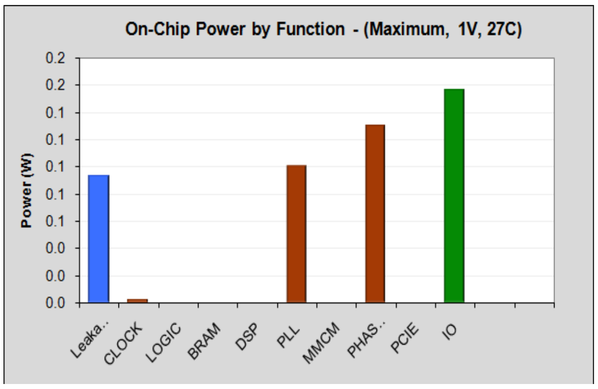
A multiplexer has

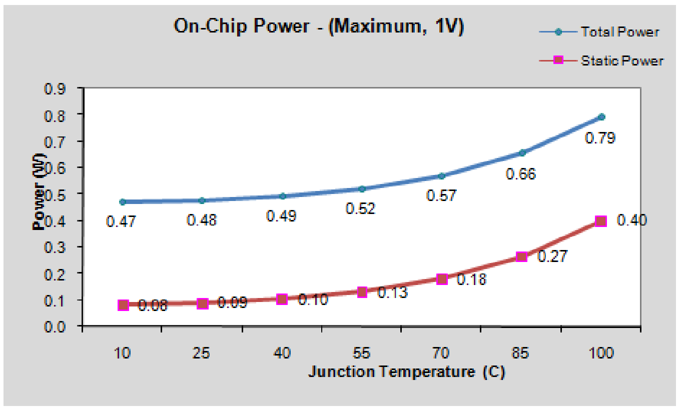
- N control inputs
- 2^N data inputs
- 1 output
- A multiplexer routes (or connects) the selected data input to the output.

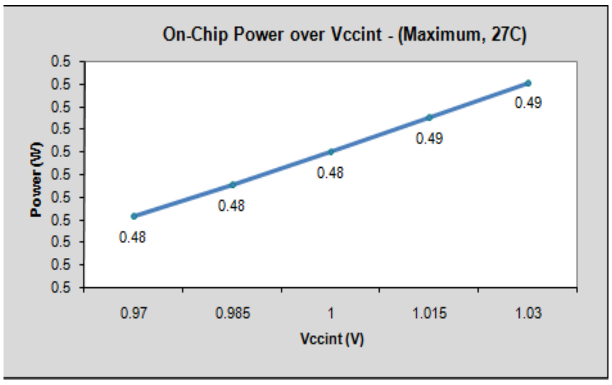
– The value of the control inputs determines the data input that is selected.

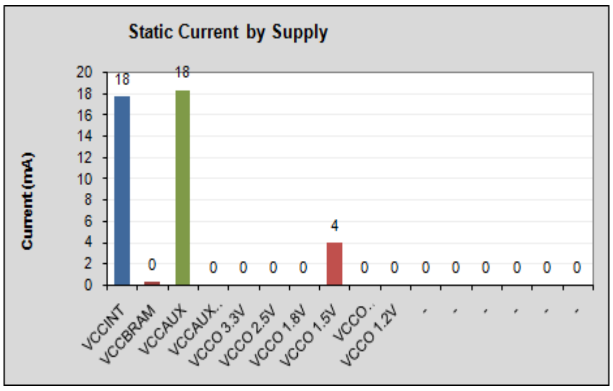
E. Power estimation











1. *Power comparison with FIFO logic*

In FIFO logic the shift register logic will come. In FIFO logic the power consumed per buffer is 10mW. For 64 buffers the power consumed is 0.32W in total. The power consumed in dynamic power allocation for the entire design of the project is 56.5%

2. Dynamic Buffer allocation

Based on the Mux select bit the buffers are dynamically allocated in the buffer

3. In which the each buffer had the certain adderss in which the bits are allocated in the dynamic manner.
4. By using the mux we are selecting the buffer in the dyanamic manner.
5. A MUX is a digital switch that has multiple inputs (sources) and a single output (destination).
6. The select lines determine which input is connected to the output.
7. A DEMUX is a digital switch with a single input (source) and a multiple outputs (destinations).

8. The select lines determine which output the input is connected to.

V. Conclusion

In this module we compare the existing and proposed power and number of circuit used. Finally we compare the performance of our proposed system.

References

[1] K. Kundert, *"Design of mixed-signal systems on a chip," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1561–1571, Dec. 2000.

[2] McCorquodale, M. S. Gebara, F. H. Kraver, K. L. Marsman, E. D. Senger, and R. M. Brown, "*A top-down microsystems design methodology and associated challenges*," inProc. Des., Autom. Test Eur. Conf.Exhibition, 2003, pp. 292–296.

[3] B. Bhattacharya and S.Bhattacharyya, "*Parameterized dataflow modeling for DSP systems,"IEEE Trans. Signal Process*., vol. 49, no.10, pp. 2408–2421, Oct. 2001.

[4] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "*Efficient circuit clustering for area and power reduction in FPGAs,"ACM Trans. Des. Autom. Electron*. Syst., vol. 7, no. 4, pp. 643–663, Oct. 2002.

[5] D. Chen, J. Cong, and Y. Fan, "*Low-power high-level synthesis for FPGA architectures*," inProc. Int. Symp. Low Power Electron. Des., Aug. 2003, pp. 134–139.

[6]P. K. Murthy and S. S. Bhattacharyya, "Buffer merging—*A powerful technique for reducing memory requirements of synchronous dataflow specifications*, "ACM Trans. Des. Autom. Electron. Syst., vol. 9, no. 2,pp. 212–237, Apr. 2004

[7] L. Shang, A. S. Kaviani, and K. Bathala, "*Dynamic power consumption in Virtex™-II FPGA family," in Proc. ACM/SIGDA 10th Int. Symp.Field-Program. Gate Arrays*, 2002, pp. 157–164