A Hierarchical C2RTL Framework for FIFO-Connected Stream Applications

Shuangchen Li * Yongpan Liu* Daming Zhang * Xinyu He* Pei Zhang† Huazhong Yang*
{ypliu,yanghz}@tsinghua.edu.cn zhangpei@gmail.com
* TNList, EE Dept.,Tsinghua University Y Explorations Inc.
Beijing, 100084, China San Jose, CA 95134, U.S.

***Abstract*—In modern embedded systems, the C2RTL (high-level synthesis) technology helps the designer to greatly reduce time-to-market, while satisfying the performance and cost constraints. To attack the performance challenges in complex designs, we propose a FIFO-connected hierarchical approach to replace the traditional flatten one in stream applications. Furthermore, we develop an analytical algorithm to find the optimal FIFO capacity to connect multiple modules efficiently. Finally, we prove the advantages of the proposed method and the feasibility of our algorithm in seven real applications. Experimental results show that the hierarchical approach can have an up to 10.43 times speedup compared to the flatten design, while our analytical FIFO sizing algorithm shrinks design time from hours to seconds with the same accuracy compared to the simulation based approach.**

I. Introduction

With continuous scaling down of CMOS technology, the gap between design productivity and transistor resources becomes ever larger. To resolve the challenge, design community is seeking a higher abstraction rather than register transfer level(RTL). Furthermore, in modern SoC designs, extensive use of embedded processors, huge silicon capacity, reuse of behavior IPs, extensive adoption of accelerators and more time-to-market pressure are needed. Compared with the traditional RTL approach, the C2RTL flow provides magnitudes of improvements in productivity to better meet those requirements. Recently, people observed a rapid rising demand for the high quality C language to RTL (C2RTL) tools [1].

In reality, designers have successfully developed various applications using C2RTL tools with much shorter design time, such as face detection [2], 3G/4G wireless communication [3], digital video broadcasting [4] and so on. Among those tools, many [5]–[8] are focusing on stream applications. They create design architectures including different modules connected by first-in first-out (FIFO) channels. There are some other tools focusing on general purpose applications. For example, Catapult C [9] takes different timing and area constraints to generate Pareto-optimal solutions from common C algorithms. However, little control on the architecture leads to suboptimal results. As [10] has shown, FIFO-connected architecture can generate much faster and smaller results in stream applications.

Among C2RTL tools for stream applications, GAUT [5] transforms C functions into pipelined modules consisting of processing units, memory units, and communication units. Global asynchronous local synchronous interconnections are adopted to connect different modules with multiple clocks. ROCCC [6] can create efficient pipelined circuits from C to be re-used in other modules or system codes. Impulse C [7] provides a C language extension to define parallel processes and communication channels among modules. ASC [8] provides a design environment for users to optimize systems from algorithm level to gate level, all within the same C++ program.

All above tools leave the user to determine the FIFO capacity between modules, which is nontrivial. As shown in Section II, the FIFO capacity has a great impact on the system performance and memory resources. Though determining the FIFO capacity via extensive simulations may work for several modules, the exploration space will become prohibitive large in the multiple-module case. Therefore, previous simulation-based method is neither time-efficient nor optimal.

To design a stream application, researchers also had investigated on the input stream rates to make sure that the FIFO between PEs will not overflow, while the real-time processing requirements are met. On-chip traffic analysis of the SoC architecture had been explored [11]. However, their simulation-based approaches suffer from a long executing time and fail in exploring large design space. A mathematical framework of rate analysis for stream applications have been proposed [12]. Based on the network calculus, [13] extended the service curves to show how to shape an input stream to meet buffer constraints. Furthermore, [14] discussed the generalized rate analysis for multimedia processing platforms. However, all of them adopts a more complicated behavior model for PE streams, which is not necessary in the hierarchical C2RTL framework.

This paper proposed a novel C2RTL framework, which supports a hierarchical way to implement complex stream applications and determines the FIFO capacity automatically. It is noted that this framework may be applicable to other applications, but it has the best improvement in stream applications. Our contributions are listed as below: 1) Unlike treating the whole algorithm as one module in the flatten design, we cut the complex stream algorithm into modules and connect them with FIFOs. Experimental results showed that the hierarchical implementation provides 10.43 times speedup compared to the flatten design. 2) We formulate the parameters of modules in stream applications and give out analytical results for the optimal FIFO capacity in two-module case, which is validated by exhaustive simulations. Furthermore, we develop a heuristic algorithm to find the optimal FIFO capacity in a multiple-module design. 3) We demonstrate the proposed method in seven real applications. Compared to the uniform FIFO capacity, our method can save memory resources by 14.46 times. Furthermore, the algorithm can optimize FIFO capacity in seconds, while extensive simulations may need hours.

The rest of the paper is organized as follows. Section II describes the motivation of our work. We present our model framework in Section III. The algorithm for optimal FIFO size

[0]This work was supported in part by the NSFC under grant 60976032, National Science and Technology Major Project under contract 2010ZX03006-003-01, eXCite software donated from Y Explorations Inc. and High-Tech Research and Development (863) Program under contract 2009AA01Z130.

978-1-4673-0772-7/12/$31.00 ©2012 IEEE

is formulated in Section IV. Section V presents experimental results. Section VI concludes this paper.

II. Motivation

This section provides the motivation of the proposed hierarchical C2RTL framework for FIFO-connected stream applications. We first compare the hierarchical approach with the flatten one. And then we point out the importance of the FIFO sizing.

A. Hierarchical vs Flatten Approach

The flatten C2RTL approach automatically transforms the whole C algorithm into a large module. However, it faces two challenges in practice. 1) The translating time is unacceptable when the algorithm reaches hundreds of lines. In our experiments, compiling algorithms over one thousand lines into HDL codes may cost several days to run or even failed. 2) The synthesized quality for larger algorithms is generally not so good as small ones. Though the user may adjust the code style, unroll the loop or inline the function, the effect is usually limited.

Unlike the flatten method, the hierarchical approach splits a large algorithm into several small ones and synthesizes them separately. Those modules are then connected by FIFOs. It provides the flexibility of architecture as well as small modules with better performance. For example, we synthesized the JPEG encode algorithm into HDLs using eXCite [15] directly compared to the proposed solution. The flatten one costs 42'475'202 clock cycles with a maximal clock frequency of 69.74MHz to complete one computation, while the hierarchical method spends 4'070'603 clock cycles with a maximal clock frequency of 74.2MHz. It implies a 10.43 times performance speedup and a 7.2% clock frequency enhancement.

B. Performance with Different FIFO Capacity

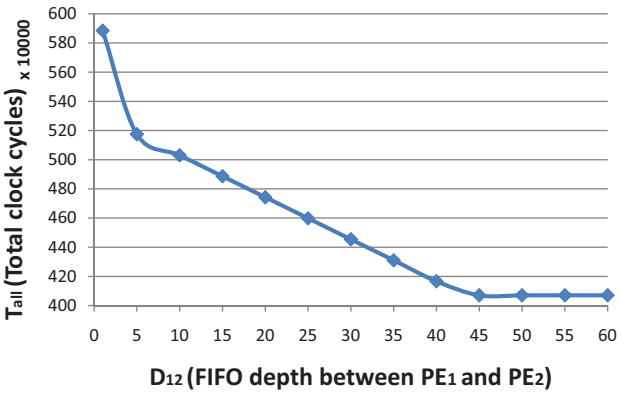

Fig. 1. FIFO size effect

In the hierarchial method, determining the FIFO size becomes relevant. We demonstrate the clock cycles of a JPEG encoder under different FIFO sizes in Figure 1. As we can see, the FIFO size will lead to an over 50% performance difference. It is interesting to see that the throughput can not be boosted after a FIFO size. The threshold may vary from several to hundreds of bits for different applications in Section V. However, it is impractical to use large FIFOs (several hundreds) due to the area overheads. Furthermore, designers need to decide the FIFO size in an iterative way when exploring different function partitions in the architecture level. Even worse, considering several FIFOs in a design, the optimal FIFO size of each module may interact with each other. Thus, determining the proper FIFO size accurately and efficiently is important but complicated. Analytical methods are preferred due to its ability to find global optimal solution very fast.

III. Hierarchical C2RTL Framework

This section shows the diagram of the proposed hierarchical C2RTL framework.[1] We define two major stages: function partition and FIFO interconnection.

A. System Diagram

The framework consists of three steps in Figure 2. In Step 1, we partition C codes into appropriate-size functions. In Step 2, we use C2RTL tools to transform each function into a hardware process element (PE), which has a FIFO interface. In Step 3, we connect those PEs with proper sized FIFOs. Given a large-scale stream algorithm, the framework will generate the corresponding hardware module efficiently. The synthesizing time is much shorter than that in the flatten approach. The hardware module can be encapsulated as an accelerator or a component in other designs. Its interface supports handshaking, bus, memory or FIFO. We denote several parameters for the module as below: the number of PEs in the module is denoted as N, the module's throughput as TH_{all}, the clock cycles to finish one computation as T_{all}, the clock frequency as CLK_{all} and the area as A_{all}.

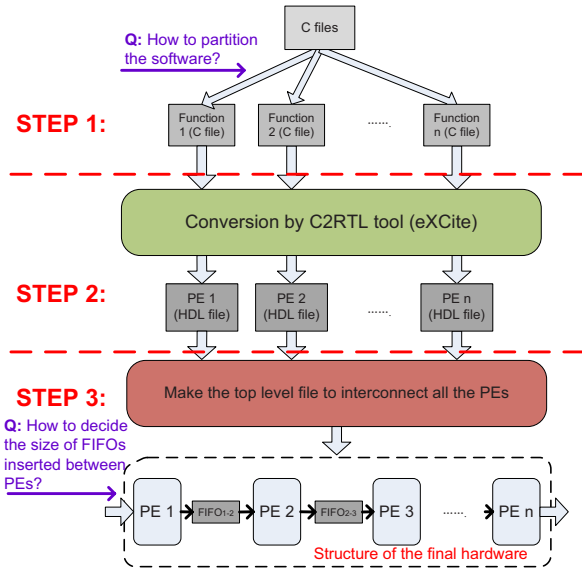

Fig. 2. Hierarchical C2RTL Flow

As C2RTL tools can handle the small-sized C codes synthesis (Step 2) efficiently, two main challenges exist: how to partition the large-scale algorithm into proper-sized functions (Step 1) and how to decide the optimal FIFO size (Step 3). We will discuss them separately.

B. Function Partition

The C code partition has a great impact on the hardware performance. Figure 3 demonstrates the partition's impacts on operating cycles and logic elements under different combinations of six GSM's sub functions. We normalized the results by the last partition. As we can see, the improper partition can slow down the performance. For example, partition A1 simply clusters five sub functions into one module and leads to a quite slow PE. It becomes the bottleneck of the system performance. On the contrary, the most efficient partition leads to an identical throughput of each PE. For example, partition B2 adopts near equal-sized function cluster and provides the best performance with reasonable area overheads. Currently, we use a manual partition strategy. An integer linear programming based partition approach is presented in [16]. Integrating such an automatical partition stage into the framework is our future work.

C. FIFO Interconnection

To deal with the FIFO interconnection, we first define the parameters of FIFO and PE's interfaces. They will be used to analyze the performance in the next section. Figure 4 shows the signals of a FIFO. F_clk denotes the clock signal of the FIFO. F_we and F_re denote the enable signals of writing and reading. F_dat_i and F_dat_o are the input and output data bus. F_ful and F_emp indicate the full and empty state, which are active high. Given a FIFO, its parameters are shown in Table I. To connect modules with FIFOs, we need to determine $D_{(n-1)n}$ and $W_{(n-1)n}$.

Without considering the constraints posed by FIFOs connected, we formulate the parameters of the n^{th} PE interface

[1]Presently, this framework accepts the finished HW/SW partition from other tools or designers. It focuses on transforming C algorithms into RTL modules. However, it is possible to integrate HW/SW partition procedure into this framework by iterative design explorations.

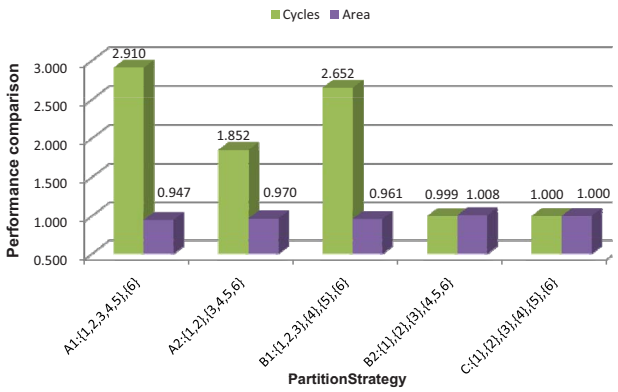

Fig. 3. Partition's Impacts

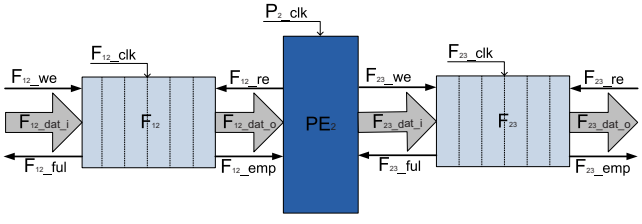

Fig. 4. Circuit diagram of FIFO blocks connecting to PE_2

TABLE I
THE PARAMETER OF FIFO BETWEEN PE_{n-1} AND PE_n

Name	Description	Examples[2]
$Fclk_{(n-1)n}$	Clock frequency (MHz)	50
$W_{(n-1)n}$	Data bus width	16
$A_{(n-1)n}$	Area: memory resource used (bit)	704
$D_{(n-1)n}$	FIFO depth	44
$f_{(n-1)n}(m)$ [1]	Number of data in FIFO at m^{th} cycle	

[1] m means m^{th} cycle.
[2] This example comes from the FIFO between PE_1 and PE_2 in the JPEG encode case.

TABLE II
THE PARAMETER OF THE N^{th} PE'S INPUT/OUTPUT INTERFACES

Name	Description	Examples[2]
Type	Interface type,I or II	II
T_n	Period of PE_n (cycles)	848
$K_{ni/o}$	Number of data input/output in T_n	64
$t_{ni/o}$	Reading or writing time in T_n (cycles)	128
$r_{ni/o}$	Reading or writing rate: $K_{ni/o}/t_{ni/o}$	0.5
$R_{ni/o}$	Input or output throughput $K_{ni/o}/T_n$	0.0755

[1] m means m^{th} cycle.
[2] Output of PE_2 in the JPEG encode case, as shown in Figre 5

in Table II. Based on a large number of PEs converted by $eXCite$, we have observed two types of interface parameters. Figure 5 shows the waveform of type II. As we can see, t_n is less than T_n in this case. In type I, t_n equals to T_n, which indicates the idle time is zero.

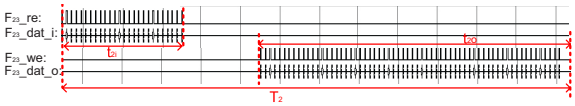

Fig. 5. Type II case: Interface of PE_2 in the JPEG encode

IV. ALGORITHM FOR FIFO-CONNECTED BLOCKS

This section formulates the FIFO interconnection problem. We then describe the equations of optimal FIFO capacity for two PEs. Finally, we propose an algorithm to solve the FIFO interconnection problem of multiple PEs[2].

A. FIFO Interconnection Formulation

Given a design consisting of N PEs, we need to determine the depth $D_{(i-1)i}$ of each FIFO[3], which maximizes the throughput TH_{all} and minimizes the area A_{all}.

$$MIN. \quad \sum_{i=2}^{N} D_{(i-1)i} \tag{1}$$

$$s.t. \quad TH_{all} \geq TH_{ref} \quad and \quad A_{all} \leq A_{ref} \tag{2}$$

where TH_{ref} and A_{ref} can be the user-specified constraints or optimal values of the design. Without losing generality, we set TH_{ref}=TH_{best} and A_{ref}=∞, TH_{best} means the maximum throughput the system can get. We assume that F_{01} never empty and $F_{N(N+1)}$ never full. That is for $\forall m$, $f_{01}(m) > 0$ and $f_{N(N+1)}(m) < D_{N(N+1)}$[4].

[2]Every module has the same hierarchical level.
[3]We assume that the $W_{(i-1)i}$ is decided by the application.
[4]This means that we only consider the operating state of the design instead of the halted state.

TABLE III
ANALYTICAL EXPRESSIONS FOR OPTIMAL FIFO SIZE WHEN $N = 2$

	I+I	I+II			II+I		
		$R_{1o} < R_{2i}$	$R_{1o} > R_{2i}$		$R_{1o} < R_{2i}$		$R_{1o} > R_{2i}$
			$R_{1o} < r_{2i}$	$R_{1o} > r_{2i}$	$R_{2i} < r_{1o}$	$R_{2i} > r_{1o}$	
D_{12}	1	$R_{1o}*(T_{2i}-t_{2i})$	$(r_{2i}-R_{1o})*t_{2i}$	1	$(r_{1o}-R_{2i})*t_{1o}$	1	$R_{2i}*(T_{1o}-t_{1o})$

TABLE IV
ANALYTICAL EXPRESSIONS FOR OPTIMAL FIFO SIZE WHEN $N = 2$ (II+II CASE)

	Conditions	$R_{1o} > R_{2i}$		$R_{1o} < R_{2i}$		
		$r_{1o} > r_{2i}$	$r_{1o} < r_{2i}$	$r_{1o} > r_{2i}$	$R_{2i} < r_{1o} < r_{2i}$	$r_{1o} < R_{2i}$
D_{12}	$T_1 > t_{1o} > T_2 > t_{2i}$	A [1]	$t_{2i}*(r_{2i}-r_{1o})+A$	$t_{1o}*(r_{1o}-R_{2i})$	$\max[t_{1o}*(r_{1o}-R_{2i}), (T_{2i}-t_{2i})*r_{1o}]$	B
	$T_1 > T_2 > t_{1o} > t_{2i}$			$\max[t_{1o}*(r_{1o}-r_{2i}), t_{1o}*r_{1o}-t_{2i}*r_{2i}]$	B [2]	
	$T_1 > T_2 > t_{2i} > t_{1o}$		N/A	$t_{1o}*(r_{1o}-r_{2i})$		

[1] $A = (T_1 - t_{2i})/R_{2i}$
[2] If $r_{1o} * t_{1o} > r_{2i} * t_{2i}$, $B = r_{1o} * min\{(T_2 - t_{2i}), t_{1o}\}$;
Else $B = min\{r_{1o} * (T_2 - t_{2i}), r_{2o} * t_{2o} * n - r_{1i} * t_{1i}\}$,
$n = \lceil (r_{2i} * t_{2i})/(r_{1o} * t_{1o}) \rceil$

B. Optimal FIFO Capacity for Two PEs

According to the interface classification in Section III-C, there are four interface combinations for two PEs: I + I, I + II, II + I and II + II.

We show the analysis results under the first three cases in Table III. We will show the reason progress by the following case. Considering data processing rate $r_{2i} > R_{1o} > R_{2i}$ in the I+ II case, we need TH_{all}=TH_{best}. Obviously, the design throughput TH_{all} is limited by the slowest PE, considering $r_{2i} > R_{1o} > R_{2i}$, we have $TH_{all} = TH_{best} = R_{2o}$ From the condition of parameter Table II we know only when the FIFO never affect PE_2, can PE_2 get its output throughput as R_{2o}. That means for $\forall m$,

$$f_{12}(m) = f_{12}(kT_2) + (R_{1o} - r_{2i}) * (m - kT_2) \geq 0 \quad (3)$$

Considering $r_{2i} > R_{1o} > R_{2i}$, the minimal value is achieved at $kT_2 + t_2$. Take it into equation 3, the minimal FIFO depth D_{12} can be obtained

$$D_{12} \geq f_{12}(kT_2) \geq (r_{2i} - R_{1o}) * t_2 \quad (4)$$

Similarly, we can get other FIFO size equations in Table III.

Next, we show the FIFO size equations for II+ II in Table IV. Assuming the $R_{1o} > R_{2i}$, $r_{1o} > r_{2i}$ and $T_1 > t_{1o} > T_2 > t_{2i}$, the maximal throughput is achieved when $TH_{all} = R_{2o}$. According to the condition of parameter Table II, $f_{12}(m) > 0$ should hold when $m \in [kT_2, kT_2 + t_2]$. As the output of PE_1 is not always working in this case, we discuss them respectively. If $m \in [kT_2, kT_2 + t_2]$ and $lT_1 < kT_2 < kT_2 + t_2 < lT_1 + t_1$, we have

$$f_{12}(m) = f_{12}(kT_2) + (r_{1o} - r_{2i}) * (m - kT_2) \geq 0 \quad (5)$$

Because we have $r_{1o} > r_{2i}$, equation 5 always holds. It means the FIFO will never be empty in this condition. In other conditions, we use the average writing rate R_{2i} of PE_2, and we have the following equations:

$$f_{12}(m) = f_{12}(lT_1 + t_1) - R_{2i} * (m - lT_1 - t_1) \geq 0 \quad (6)$$

Taking m as $lT_1 + T_1$, the minimal FIFO depth D_{12} for this case can be obtained

$$D_{12} \geq f_{12}(lT_1 + t_1) \geq R_{2i} * (T_1 - t_1) \quad (7)$$

Similarly, we can get other FIFO size equations in Table IV.

C. FIFO Capacity for Multiple Blocks

To determine the FIFO size for multiple PEs, we state an assumption to simplify the problem and explain the algorithm. After that, we explain how to merge two PEs into one, which is a key step in the algorithm.

In case of multiple PEs, the FIFO sizing will interact with each other. It will increase the exploration space greatly. We set TH_n as the throughput of the system consisting of PE_1 to PE_n. We have a recursive equation:

$$TH_n = \begin{cases} R_{no} & TH_{(n-1)o} > R_{ni} \\ K_{no}/K_{ni} * TH_{(n-1)} & others \end{cases} \quad (8)$$

As TH_{all}=TH_N, we can express TH_{all} as the followings:

$$TH_{all} = TH_{bn} \prod_{i=bn+1}^{N} (K_{no}/K_{ni}) \quad (9)$$

where we denote the bottleneck PE's index as bn. The assumption is that $D_{(bn-1)bn}$ and $D_{bn(bn+1)}$ are only affected by PE_{bn-1}, PE_{bn}, and PE_{bn+1}. Experimental results in Section V show the assumption holds for real stream applications.

Algorithm 1 FIFO Capacity Algorithm for $N > 2$

Input: $N, ParaG[N]$
Output: $D_{(n-1)n}$, for $\forall n \in [2, N]$
1: $n = N - 1$
2: **while** $n > 1$ **do**
3: $i = Get\_bottleneck\_index(ParaG)$
4: **if** $i > 0$ **then**
5: $D_{(i-1)i} = Fsize\_2(ParaG[i-1], ParaG[i])$
6: $Merge(ParaG[i-1], ParaG[i])$
7: **end if**
8: **if** $i < N$ **then**
9: $D_{i(i+1)} = Fsize\_2(ParaG[i], ParaG[i+1])$
10: $Merge(ParaG[i], ParaG[i+1])$
11: **end if**
12: $Update(ParaG)$
13: $n = n - 1$
14: **end while**

We describe the proposed method to determine the FIFO capacity for multiple PEs (N>2) in Algorithm 1. The inputs are the number of PE N, the parameters of PE interfaces $ParaG[N]$. $ParaG[N]$ includes $K_{ni/o}, r_{ni/o}, R_{ni/o}, t_{ni/o}, T_n$ shown in Table II. The output is each FIFO's optimal depth $D_{(n-1)n}$. In each loop, n means the number of PEs left to be processed. We get the bottleneck PE's index i from function $Get\_bottleneck\_index()$ in line 3; Line 4 and 8 judge whether this PE is the head or tail node of the PE chain. If not, they respectively generate the input and output FIFO for PE_i. As PE_i is the bottleneck in the chain, the assumptions in Section IV-B is satisfied. Therefore, we call function $Fsize\_2()$ to derive the FIFO capacity based on the equations in Table III and IV. After that, we call function $Merge()$ to merge two PEs into a new one, which will be discussed soon. Because the active PE is reduced, we adjust the array index in $ParaG$ according to the merge operation by function $Update()$. Finally, the number of PE decreases by one and the loop continues until all FIFO sizes are determined.

We give out the equations to compute the interface parameters for the merged PE in Table V. As defined in Table II, we use $R_{i/o}, r_{i/o}, t_{i/o}, T$ to characterize the parameters of the merged PE. We use a typical case to explain how to derive the results in Table V. Considering $r_{2i} > R_{1o} > R_{2i}$ in the I + II case, we first determine the output interface. Similar to Table IV, we have T_o =T_2. Because K_2 never changes, other output parameters remain the same.

Next, we decide the input interface. From nT to $nT + t_i$, $n \in [1, 2, ...]$, we have the number of data in FIFO at m time as below:

$$f_{12}(m) = f_{12}(nT) + R_{1o} * (m - nT) - r_{2i} * (m - nT) \quad (10)$$

TABLE V
PARAMETERS OF THE NEW PE BY MERGING PE_1 AND PE_2

	I+I		I+II		
	$R_{1o} < R_{2i}$	$R_{1o} > R_{2i}$	$R_{1o} < R_{2i}$	$R_{1o} > R_{2i}$	
				$R_{1o} < r_{2i}$	$R_{1o} > r_{2i}$
R_i	R_{1i}	$R_{2i}/R_{1o}*R_{1i}$	R_{1i}	r_i*t_i/T_i	$R_{2i}/R_{1o}*R_{1i}$
r_i	r_{1i}	$r_{2i}/R_{1o}*R_{1i}$	r_{1i}	R_{1i}	$r_{2i}/R_{1o}*R_{1i}$
t_i	t_{1i}	$t_{2i}*R_{1o}/R_{1i}$	t_{1i}	$t_{2i}*r_{2i}/R_{1o}$	t_{2i}
T_i	T_1	T_2*R_{1o}/R_{1i}	T_1	T_2	T_2
R_o	$R_{1o}/R_{2i}*R_{2o}$	R_{2o}	$R_{1o}/R_{2i}*R_{2o}$	R_{2o}	R_{2o}
r_o	$r_{1o}/R_{2i}*R_{2o}$	r_{2i}	r_{2o}	r_{2o}	r_{2o}
t_o	$t_{1o}*R_{2i}/R_{2o}$	t_{2i}	t_{2o}	t_{2o}	t_{2o}
T_o	T_1*R_{2i}/R_{2o}	T_2	T_2*R_{2i}/R_{1o}	T_2	T_2
	II+I			II+II	
	$R_{1o} < R_{2i}$		$R_{1o} > R_{2i}$	$R_{1o} < R_{2i}$	$R_{1o} > R_{2i}$
	$r_{1o} < R_{2i}$	$r_{1o} > R_{2i}$			
R_i	R_{1i}	R_{1i}	$R_{2i}/R_{1o}*R_{1i}$	R_{1i}	$r_{1i}*t_{1i}/T_i$
r_i	r_{1i}	r_{1i}	r_{1i}	r_{1i}	$r_{1i}*t_{1i}/r_i$
t_i	t_{1i}	t_{1i}	t_{1i}	t_{1i}	$[t_{1i}, t_{1i}+\Delta t]$[1]
T_i	T_1	T_1	T_1*R_{1o}/R_{2o}	T_1	$r_{1o}*t_{1o}/R_{2i}$
R_o	$R_{1o}/R_{2i}*R_{2o}$	r_o*t_o/T_o	R_{2o}	$r_{2o}*t_{2o}/T_o$	R_{2o}
r_o	$r_{1o}/R_{2i}*R_{2o}$	R_{2o}	r_{2o}	$r_{2o}*t_{2o}/t_o$	r_{2o}
t_o	t_{1o}	$t_{1o}*r_{1o}/R_{2i}$	t_{2o}	$[t_{2o}, t_{2o}+\Delta t]$[1]	t_{2o}
T_o	T_1	T_1	T_2	$r_{2i}*t_{2i}/R_{1o}$	T_2

[1] $\Delta t = |T_1 - T_2|$
[2] Here the input and output interfaces' type of one PE are the same.

When $m \in [nT + t_i, (n+1)T]$, the new PE's input interface will be idle, which means F_{12} is full:

$$f_{12}(nT + t_i) = f_{12}(nT + T) = f_{12}(nT) = D_{12} \qquad (11)$$

Based on equation 10 and 11, we have $t_i = r_{2i} * t_{2i}/R_{1o}$ when $m = nT + t_i$. Because $r_{2i} > R_{1o}$, r_i remains as R_i. When $m \in [nT + t_i, (n+1)T]$, $f(m) < D$ always holds. That means F_{12} can not be full and PE_1 will run at full speed without halting. Similarly, we have other parameters for the merged PE in other conditions in Table V.

V. EXPERIMENTS

In this section, we first explain our experimental configurations. After that, we compare the flatten approach with the proposed hierarchical method. Finally, we demonstrate the effectiveness of our algorithm using seven real applications.

A. Experimental Configurations

In our experiments, we use eXCite to do the C2RTL conversion. The HDL files are simulated by ModelSim to get the timing information. The area and clock information is obtained by Quartus II from Altera, where Cyclone II is selected as the target hardware. We derive seven large stream applications from high-level synthesis benchmark suits CHstone [17]. They come from real applications and consist of programs from the areas of image processing, security, telecommunication, and digital signal processing.

- $JPEG\ encode/decode$: JPEG transforms image between JPEG and BMP format.
- $AES\ encryption/decryption$: AES (Advanced Encryption Standard) is a symmetric key cryptosystem.
- GSM: LPC (Linear Predictive Coding) analysis of GSM (Global System for Mobile Communications).
- $ADPCM$: Adaptive Differential Pulse Code Modulation is an algorithm for voice compression.
- $Filter\ Group$: The group includes two FIR filters, a FFT and an IFFT block.

B. Flatten vs Hierarchical Approach

We compare two approaches using seven benchmarks. Table VI shows the clock cycles saved by the hierarchical approach. As we can see, an up to 10.43 times speedup can be reached. Furthermore, Table VI also shows the maximal clock frequency of two approaches. Obviously, the hierarchical approach is also faster. Among those benchmarks, the area overheads of them are generally less than 5% except the GSM case. It has a 15.77% larger area using the hierarchical method. [5]

TABLE VI
FLATTEN AND HIERARCHICAL APPROACH COMPARISON

Benchmark		Flatten approach	Hierarchical approach	Speedup
JPEG encode	T[1]	42,475,202	4,070,603	10.43
	C[2]	69.74	74.2	1.064
JPEG decode	T	623,090	456,821	1.364
	C	71.15	71.3	1.002
AES encryption	T	1,904,802	719,263	2.648
	C	71.24	91.06	1.278
AES decryption	T	2,185,802	915,222	2.388
	C	75.56	87.35	1.156
GSM	T	620,802	204,356	3.038
	C	55.73	59.16	1.062
ADPCM	T	35,691	12,464	2.864
	C	53.29	68.32	1.282
Filter groups	T	6,537,416	1,702,406	3.840
	C	93.41	96.69	1.035

[1] T means the min operation cycles T_{all}(cycles).
[2] C means the max clock frequency CLK_{all}(MHz).

TABLE VII
OPTIMAL FIFO SIZE(D_{12}) WHEN N=2

Condition	I+I	I+II			II+I		
		$R_1 < R_2$	$R_1 > R_2$		$R_1 < R_2$		$R_1 > R_2$
			$R_1 < r_2$	$R_1 > r_2$	$r_1 > R_2$	$r_1 < R_2$	
Analytical:	1	6	43	1	1	13	11
Simulation:	2	6	44	1	2	14	12

C. Optimal FIFO Capacity

We first valid our equations to determine the optimal FIFO size for $N = 2$ (Table III and IV). We compared our results with exhaustive simulations under random inputs. We listed those results in Table VII and VIII. As we can see, the analytical results fit the simulation-based ones quite well for the FIFO connecting two PEs.

Next, we compare the analytical FIFO size with the simulated results for real designs with multiple PEs. First of all, we show the relationship between the FIFO size and the running time T_{all}. Figure 6 shows the JPEG encoding case. As we can see, the FIFO size has a great impact on the performance of the design. In this case, the optimal FIFO capacity should be D_{12}=44, D_{23}=2.

Table IX lists both the analytical results and the experimental ones on FIFO size in seven cases. It shows that our algorithm is accurate enough for those real cases. Though little mismatch exists, the difference is very small. Compared to the magnitudes of speedup to determine the FIFO size, our

[5] We use the number of logic elements (LE) to represent the area.

TABLE VIII
OPTIMAL FIFO SIZE(D_{12}) WHEN N=2 FOR CASE II+II

Condition		$R_1 > R_2$		$R_1 < R_2$		
		$r_1 > r_2$	$r_1 < r_2$	$r_1 > r_2$	$R_2 < r_1 < r_2$	$r_1 < R_2$
$T_1 > t_1 >$ $T_2 > t_2$	Analytical:	24	26	96	32	1
	Simulation:	25	25	97	33	2
$T_1 > T_2 >$ $t_1 > t_2$	Analytical:	24	32	64	32	1
	Simulation:	17	30	65	33	2
$T_1 > T_2 >$ $t_2 > t_1$	Analytical:	28	N/A	2	20	1
	Simulation:	25		3	21	2

TABLE IX
OPTIMAL FIFO CAPACITY ALGORITHM EXPERIMENT RESULT IN 7 REAL CASES

Benchmark		D_{12}	D_{23}	D_{34}	D_{45}	D_{56}	T_{all}
JPEG encode	Analytical	43	2	-	-	-	4080201
	Experiment	44	2	-	-	-	4070603
JPEG decode	Analytical	2	33	17	2	-	456964
	Experiment	2	33	18	2	-	456821
AES encryption	Analytical	2	2	2	-	-	719364
	Experiment	3	2	3	-	-	719263
AES decryption	Analytical	2	257	2	-	-	867407
	Experiment	3	249	3	-	-	867306
GSM	Analytical	54	2	2	2	2	204554
	Experiment	55	2	2	2	2	204356
ADPCM	Analytical	2	2	2	2	2	12464
	Experiment	2	2	2	2	1	12464
Filter group	Analytical	2	2	86	2	2	1701896
	Experiment	2	2	87	2	2	1701846

TABLE X
FIFO AREA SAVED

Benchmark	Memory resource used(bit)		Savings
	FIFOs with [1] enough size	FIFOs with optimized size	
JPEG encode	10,048	2,624	x3.83
JPEG decode	38,776	8,376	x4.63
AES encode	92,160	67,968	x1.36
AES decode[2]	92,160	75,808	x1.22
GSM	36,028	8,602	x4.19
ADPCM	54,040	3,736	x14.46
Filter groupe	114,400	76,736	x1.49

[1] We set each FIFO depth as 128.
[2] In this case we set each FIFO depth as 256.

algorithm is quite promising to be used in architecture level design space exploration.

The memory resource savings by well designing FIFO are listed in Table X. Compared to the large enough design strategy, the memory savings are significant. Moreover, compared to the simulation based method to decide FIFO capacity, our work is extremely time efficient. Considering a hardware with N FIFO to design, each FIFO size is fixed using a binary searching algorithm. It will request $log_2(p)$ times simulations with the initial FIFO depth value $D_{(n-1)n}$=p. Assume the average time cost by ModelSim simulation as C, the entire exploration time is $N * log_2(p) * C$. Considering the Filer Group case with $N = 5$, $p = 128$ and $C = 170$ seconds, which are typical values on a normal PC, we have to wait 100 minutes to find the optimal FIFO size. However, our analytical solution can finish the exploration in seconds.

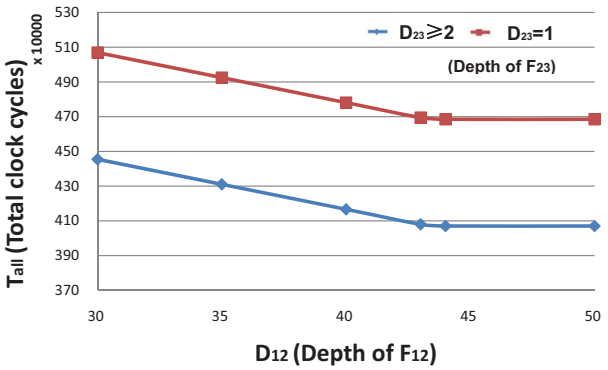

Fig. 6. FIFO capacity in JPEG encode case

VI. CONCLUSIONS

Improving the booming design methodology of C2RTL to make it more widely used is the goal of many researchers. Our work of the hierarchical framework for FIFO-connected stream applications does have achieved the improvement. We propose a hierarchical C2RTL design flow to increase the performance of the flatten one. Moreover, we develop an analysis based heuristic algorithm to find the optimal FIFO capacity in a multiple-module design. Experimental results show that hierarchical approach can improve performance by up to 10.43 times speedup. What's more, the FIFO sizer works accurately in seconds compared with the simulation based approach in hours. The future work includes the automatical C code partition and using our algorithm in more complex architectures with feedback and branches.

REFERENCES

[1] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 4, pp. 473–491, 2011.
[2] B. Schafer, A. Trambadia, and K. Wakabayashi, "Design of complex image processing systems in esl," in *Asia and South Pacific Design Automation Conference*. IEEE Press, 2010, pp. 809–814.
[3] Y. Guo and D. McCain, "Rapid prototyping and vlsi exploration for 3g/4g mimo wireless systems using integrated catapult-c methodology," in *Wireless Communications and Networking Conference*. IEEE, 2006, pp. 958–963.
[4] M. Rossler, H. Wang, U. Heinkel, N. Engin, and W. Drescher, "Rapid prototyping of a dvb-sh turbo decoder using high-level-synthesis," in *Specification & Design Languages*. IEEE Forum on, 2009, pp. 1–6.
[5] G. Lhairech-Lebreton, P. Coussy, and E. Martin, "Hierarchical and multiple-clock domain high-level synthesis for low-power design on fpga," in *International Conference on Field Programmable Logic and Applications*. IEEE, 2010, pp. 464–468.
[6] J. Villarreal, A. Park, W. Najjar, and R. Halstead, "Designing modular hardware accelerators in c with roccc 2.0," in *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2010, pp. 127–134.
[7] M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented fpga computing in the streams-c high level language," in *Field-Programmable Custom Computing Machines*. IEEE Symposium, 2000, pp. 49–56.
[8] O. Mencer, "Asc: a stream compiler for computing with fpgas," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 9, pp. 1603–1617, 2006.
[9] "http://www.mentor.com."
[10] A. Agarwal, "Comparison of high level design methodologies for algorithmic ips: Bluespec and c-based synthesis," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.
[11] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing on-chip communication architectures," in *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 20, no. 6, 2001, pp. 768–783.
[12] R. Cruz, "Quality of service guarantees in virtual circuit switched networks," in *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 6, 1995, pp. 1048–1056.
[13] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele, "Rate analysis for streaming applications with on-chip buffer constraints," in *Asia and South Pacific Design Automation Conference*. IEEE Press, 2004, pp. 131–136.
[14] Y. Liu, S. Chakraborty, and R. Marculescu, "Generalized rate analysis for media-processing platforms," in *International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*. IEEE, 2006, pp. 305–314.
[15] "http://www.yxi.com."
[16] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Partitioning of behavioral descriptions with exploiting function-level parallelism," in *Fundamentals,IEICE Trans on*, vol. E93-A, 2010, pp. 488–499.
[17] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "Chstone: A benchmark program suite for practical c-based high-level synthesis," in *Circuits and Systems*. IEEE International Symposium on, 2008, pp. 1192–1195.