International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:4, 2013

High Level Synthesis of Digital Filters Based On Sub-Token Forwarding

Iyad F. Jafar, Sandra J. Alrawashdeh, and Ban K. Alhamayel

Abstract**—**High level synthesis (HLS) is a process which generates register-transfer level design for digital systems from behavioral description. There are many HLS algorithms and commercial tools. However, most of these algorithms consider a behavioral description for the system when a single token is presented to the system. This approach does not exploit extra hardware efficiently, especially in the design of digital filters where common operations may exist between successive tokens. In this paper, we modify the behavioral description to process multiple tokens in parallel. However, this approach is unlike the full processing that requires full hardware replication. It exploits the presence of common operations between successive tokens. The performance of the proposed approach is better than sequential processing and approaches that of full parallel processing as the hardware resources are increased.

Keywords**—**Digital filters, High level synthesis, Sub-token forwarding.

I. Introduction

THE increasing capabilities of silicon technology and the growing complexity of applications in recent times forced design methodologies and tools to move to higher abstraction levels. As a matter of fact, the process of hardware design has evolved tremendously. Simulation at the gate level appeared in the early 1970s, and cycle-based simulation became available by 1979. In the mid eighties, hardware description languages (HDL), such as Verilog and VHDL, have enabled large adoption of simulation tools. During the 1990s, the first generation of commercial high-level synthesis (HLS) tools was available commercially [1].

HLS is the term used to describe the automated design process that interprets an algorithmic description of a desired system behavior and creates the hardware that implements that behavior at the register-transfer level (RTL) [2] such that design constraints are satisfied and the cost function is optimized. The inputs to a HLS system are the behavioral specification, design constraints and an optimization function. The behavioral specification is expressed by the means of a data-flow graph (DFG) which shows the data dependencies between a number of ordered operations in an algorithm using nodes and arcs. The nodes represent the processes that exist in the system, while the lines between the nodes represent the links between those processes. On the other hand, the output of the HLS is a register-transfer-level implementation of the system.

The process of high level synthesis consists of several procedures: system definition, scheduling, hardware allocation, and generation of the control system. Out of these procedures, scheduling and hardware allocation are the most critical. Scheduling assigns the operation of the DFG nodes to control steps which are the fundamental sequencing unit in synchronous systems (clock cycle). Scheduling should be done such that the minimum number of processing elements is used and the rate and time optimality constraints are satisfied. On the other hand, allocation consists of assigning the operations to hardware units, i.e. allocating functional units, storage and communication paths.

Several HLS tools that aim at designing optimal digital systems are commercially available, such as CATHEDRAL II [3], ALPS [4], BSSC [5] and MARS [6]. The main differences between these tools lie in the way the design space is explored, the representation of the behavioral description of the system, and how the scheduling is performed. However, most of the scheduling algorithms in these HLS tools ignore the use of pipelined processing elements and assumed them to be unified functional units that can perform multiple operations. On the contrary, the scheduler found in the pipelined heterogeneous high level synthesis algorithm (PHHLS) [7] treats the processing elements as distinct functional units each of which is responsible for a single task instead of being unified functional units capable of performing multiple tasks. Additionally, it employs the concept of pipelining in the system design at the register-transfer level to reduce the lower iteration period and the hardware resources. In this case, the iteration period is not bounded by the highest node computational delay in contrast to systems with non-pipelined processing elements.

The PHHLS and other HLS algorithms were only tested when the system to be designed considers the processing of the data points one data point at a time, i.e. single token. This in turn limits the system performance at some point even if more hardware is available due to the sequential processing. The increase in hardware resources can be exploited to improve the performance of the system by processing multiple tokens in parallel. However, processing the tokens in parallel requires full replication of the hardware which might be expensive and not affordable in case of processing large number of tokens. This issue can be alleviated in the design of digital filters by observing that there are some operations that are repeated as the filter moves from one token to another. Taking this into account, we propose modifying the behavioral

I. F. Jafar is with the Computer Engineering Department at The University of Jordan, Amman 11942, Jordan (phone: +962-6-5355000; fax: +962-6-5300813; e-mail: iyad.jafar@ju.edu.jo).

S. J. Alrawashdeh is with the Computer Engineering Department at The University of Jordan, Amman 11942, Jordan (phone: +962-6-5355000; fax: +962-6-5300813; e-mail: sandra.jamal.6@gmail.com).

B. K. Alhamayel with the Computer Engineering Department at The University of Jordan, Amman 11942, Jordan (phone: +962-6-5355000; fax: +962-6-5300813; e-mail: bonbonsweetcandy_89@yahoo.com).

description, expressed as DFG, of the system to be designed to operate on multiple tokens at once. However, and unlike the full parallel approach, these multiple tokens are selected such that they have some operations in common. Thus, when the set of tokens are processed at the same time, the common operations found in these multiple tokens are performed once and saved in hardware registers such that they are forwarded to other tokens in the set. Essentially, this approach processes the tokens in parallel; however, forwarding the results of common operations reduces the hardware requirements of the full parallel processing. At the same time, the performance of the proposed approach is expected to be better than sequential processing, and it will approach that of the full parallel approach depending on the available hardware resources.

In this paper, we investigate the use of the concept outlined earlier in improving the performance of the PHHLS algorithm in the design of digital image filters. We refer to this technique as sub-token forwarding since some intermediate results of tokens are forwarded to other tokens in the set. The rest of the paper is organized as follows. In Section II, we explain the concept of token packaging when considered for the design of the 2-D arithmetic mean filter. Section III evaluates the benefits of employing token packaging in PHHLS. Finally, Section IV concludes the paper.

II. Sub-Token Forwarding

As outlined earlier, the behavioral description used by most HLS algorithms to design digital systems is specified such that tokens are assumed to be presented to the system one at a time. However, this approach may not utilize the available hardware efficiently, especially when there are common operations between successive tokens. In order to improve the performance of the HLS algorithms, we propose to make the input to the system to consist of multiple tokens. Processing a set of tokens in parallel by the system is expected to increase the hardware requirements since the processes have to be replicated. However, this cost can be reduced if some of the operations in different tokens are the same. In this case, common operations between different tokens in the set are performed once and their results are forwarded to the processing of the other tokens in the package. This observation is strongly applicable to the operation of many 1-D and 2-D digital filters. To demonstrate the idea, in the following we consider the design of the 2-D arithmetic mean filter that is commonly used in digital image processing for blurring and noise suppression.

For an image F(x,y), the output of an arithmetic mean filter of size (2m+1)×(2m+1) at pixel (x,y) is given by

$$G(x,y)=\frac{1}{(2n+1)^2}\sum_{i=-m}^{m}\sum_{j=-m}^{m}F(x+i,y+j) \qquad (1)$$

The filtered image is obtained by processing all tokens (pixels in this case) by rolling the filter mask over all pixels in the image and applying (1). Moving the filter mask can be in any direction across the image. If the filter mask size is 3x3 and the filter is moved in scan-line order, then the output of the filter at location (x,y) can be written as

$$\begin{aligned}G(x,y)=\frac{1}{9}(F(x-1,y-1)+F(x,y-1)+F(x+1,y-1)+\ldots\\ \left[F(x-1,y)+F(x,y+1)+F(x,y)+\ldots\right.\\ \left.F(x,y+1)+F(x+1,y)+F(x+1,y+1)\right])\end{aligned}$$
$$(2)$$

while the output at (x,y+1), which is the next token to be presented to the system, can be expressed by

$$\begin{aligned}G(x,y+1)=\frac{1}{9}(F(x-1,y+2)+F(x,y+2)+F(x+1,y+1)+\ldots\\ \left[F(x-1,y)+F(x,y+1)+F(x,y)+\ldots\right.\\ \left.F(x,y+1)+F(x+1,y)+F(x+1,y+1)\right])\end{aligned}$$
$$(3)$$

Comparing (2) and (3) reveals that the addition operations enclosed in brackets are the same when the two token are processed. Thus, if the behavioral description of the system is modified to accept the two tokens at once, these common operations can be performed once when processing the first token and saved in hardware registers such that they are used when the second token is processed. Apparently, for some given hardware resources, this would result in reducing the time required to process the two tokens as opposed when the two tokens are processed separately since some operations are eliminated when the second token is processed.

In this example, only 11 addition operations are needed to process two tokens in the sub-token forwarding approach, while processing the tokens sequentially requires 16 addition operations. If the two tokens are processed in a full parallel fashion, there are also 16 additions operations to process the two tokens; however, the time required to process them is half that of the sequential approach since the full parallel approach will have twice the number of adders. In this case, we can see that the performance of sub-token forwarding performing is somewhere between the sequential and full parallel approach, but with less hardware resources than the parallel approach. The performance of the proposed approach is expected to approach that of the full parallel approach if more hardware resources are available to the design process.

For further improvement, we can consider increasing the number of tokens per package and search for common operations between the tokens in the package. For example, if the number of tokens is 3, there are 24, 15, and 24 addition operations for the sequential, sub-token forwarding, and full parallel approaches, respectively. The additional cut achieved by the sub-token forwarding over the sequential approach is obtained because some addition operations in the first token are also present in the third token, thus they can be forwarded from the first token to the third token.

Another one way to improve the performance of sub-token forwarding is to increase the filter size as this would result in more common operations between the tokens. For example, with a 5x5 filter mask, if we package five tokens then we will have common operations between all tokens in the set. The first token will require 24 additions, while each of the next four tokens will require four additions. Thus there are 40 additions in sub-token forwarding when compared to 120 additions in the sequential case. Of course the full parallel approach requires 120 additions as well but since the adders are replicated, the processing time of the full parallel approach is effectively that of performing 24 additions.

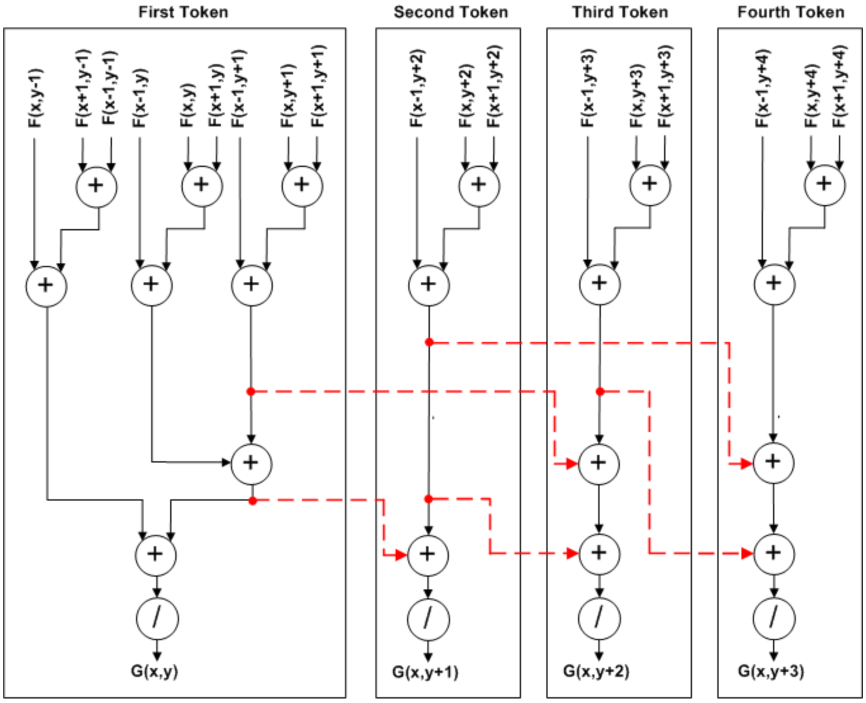


Fig. 1 Data-flow graph of the 3x3 arithmetic mean filter for different number of tokens

Again, it might be argued that token packaging is the same as processing the tokens in parallel and independently. Nonetheless, the additional hardware needed by sub-token forwarding is less than that of the parallel approach since common operations are done once and forwarded to other tokens in the set.

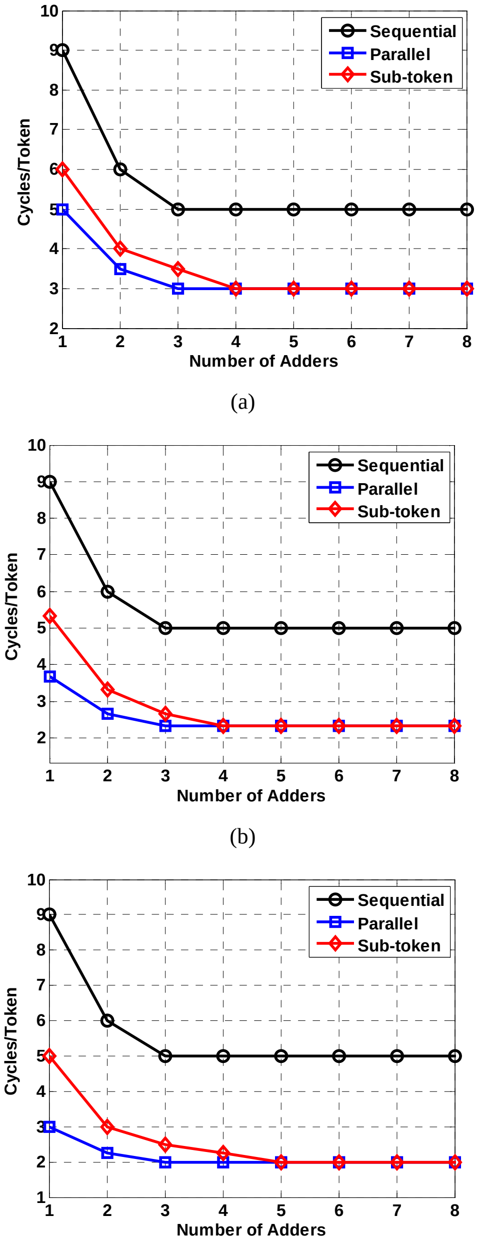
In order to design a system that exploits the sub-token forwarding concept, the data-flow graph of the system has to be modified before it is processed by the HLS algorithm. The modifications introduced to the DFG depend on the number of tokens and type and number of the common operations that are determined by investigating the filtering operation. For example, the left most box in Fig. 1 is the DFG for the 3×3 arithmetic mean filter that operates on a single token. Here, eight addition and one division operations are required. In case of full parallel processing of K tokens, the DFG is simply K replicas of the DFG included in this box. In sub-token forwarding, when the number of tokens is two, this requires expanding the DFG in the left most box by adding the DFG in the second left most box. As can be seen in the figure, the part of the intermediate results in the computation of the first token are forwarded to the second token. This is indicated by the dashed red arc. In case of three and four tokens, the right most boxes are added to the DFG. Again, note how the third token is reusing some of the values that are computed in the first and second tokens. The same applies when the number of tokens is four where the processing of the fourth token reuses values from the second and third tokens.

At the hardware level, forwarding is achieved by storing these values in registers until they are needed. Hardware registers are not only needed in the sub-token forwarding. The sequential approach in the 3x3 arithmetic mean requires three registers to save three intermediate values between successive cycles in the processing of the token. In case of the full parallel approach, the number of required registers is 3xK, where K is the number of tokens. As for the sub-token forwarding, the number of registers is 3 + (K-1) since we need to add one register per additional token to save the new intermediate result.

III. Experimental Results

In this section, we evaluate the performance of the proposed approach, sub-token forwarding, in the design of a 3x3 arithmetic mean filter and compare it with the performance of sequential and full parallel approaches when the PHHLS scheduler is used. In sequential approach, tokens are processed one after another. On the other hand, the full parallel approach accepts more than one token at once and processes them in parallel. However, the system in this case is designed assuming that each token has dedicated hardware units that are not used by other tokens. Since the common operations found between different tokens in the arithmetic mean filtering are addition operations, our evaluation is based on varying the number of available adders that can be used in designing the system, and fixing the number of dividers to one. The number of tokens in the full parallel and sub-token forwarding is varied from 2 to 4.

To quantify the performance of different approaches, we use the number of cycles required to finish one token in each approach as a performance metric when changing the number of adders and the number of tokens. Higher performance is achieved if lower number of cycles is required to process the tokens. The cycle duration is defined to be the delay of the slowest unit in the system, which is the divider in this case.



(a)

(b)

(c)

Fig. 2 Performance results for sequential, full parallel, and sub-token forwarding approaches for (a) two tokens (b) three tokens (c) four tokens

In the following experiments, it is assumed that the system is defined using the DFGs shown in Fig. 1. The sequential approach is basically represented by the box labeled “First Token” in Fig. 1, while the DFG of the full parallel approach is basically K copies of this DFG. In sub-token forwarding, the DFG in the “First Token” box is expanded by including other boxes depending on the number of tokens.

TABLE I
NUMBER OF REGISTERS IN DIFFERENT APPROACHES

No. tokens	Sequential	Full Parallel	Sub-token Forwarding
1	3	-	-
2	3	6	4
3	3	9	5
4	3	12	6

In the first experiment, the number of tokens used in sub-token forwarding and parallel approaches is set to 2. The number of adders is varied between 1 and 8. The results are shown in Fig. 2(a). For the sequential case, the number of cycles per token is 9 while it is 6 and 5 cycles, for the sub-token forwarding and parallel approaches, respectively, when the system has one adder (two adders in the case of full parallel design). As the number of adders is increased, the performance of all approaches improves and the performance of sub-token forwarding approaches that of the full parallel approach. When the number of adders is above three, the performance of sub-token forwarding and parallel approaches are the same.

Additionally, note that how the performance of the sub-token forwarding and the full parallel approaches does not improve when the number of adders is greater than four (effectively 8 adders in the full parallel approach), respectively. This can be explained by referring to Fig. 1. In the full parallel approach, the maximum number of additions that can be performed in parallel per token is 3. The next addition operations use the results of previous additions and cannot be performed even if more adders are available. The same discussion applies to sub-token forwarding. Investigating the left most two blocks of Fig. 1, reveals that the maximum number of additions than can be carried out for the two tokens is 4. Thus, adding more adders to the systems is also of no use.

An important factor to be considered in the evaluation is the required number of registers. Table I shows that for the sequential approach, the number of registers is always 3 regardless of the number of tokens. This is logical since three registers are required to process one token and these same three registers are reused when the second token is processed. For the full parallel approach, the number of adders is twice that of the sequential case, since each token is processed independently and the registers in one token cannot be used in the other token. Note how the sub-token forwarding with two tokens requires only 4 registers which is less than that of the parallel approach. This can be easily seen in Fig. 1 where three registers are used in the first token and only one register is needed to save the results of the second token.

The results for the three-token and four-token cases are shown in Fig. 2(b) and Fig. 2(c). In these figures, we see that the performance of the sequential approach is the worst, and the performance of sub-token forwarding approaches that of the full parallel approach as the number of adders is increased. Nonetheless, we need to keep in mind that the number of adders in the parallel approach is K times that in the sub-token forwarding approach, where K is the number of tokens. In terms of required registers, Table I shows that the sub-token forwarding requires few additional registers when compared to the sequential approach while the number of registers in the full approach is the highest.

Comparing the performance of the three approaches using different number of tokens, we see that the sequential approach requires the same number of cycles per token. However, the performance of the full parallel approach increases since lower number of cycles are required to process one token at the expense of replicating hardware units. In case of the sub-token forwarding, the performance also improves as the number of tokens is increased since forwarding is exploited better as shown in Fig. 1.

In summary, the sub-token forwarding approach utilizes the available processing hardware units efficiently to achieve a performance that is the same or very close to that of the full parallel approach depending on the number of tokens and the available hardware resources. The only requirement for the sub-token forwarding approach when compared to the sequential approach is additional registers.

IV. Conclusion

High level synthesis is extensively used in the design of digital systems as it gives full control over optimizing the design process. In this paper, we investigate the concept of sub-token forwarding in specifying the behavioral description of digital systems, specifically digital filters, that is used in the synthesis process. The proposed approach relies on parallelizing the processing of tokens with lower hardware resources by exploiting the existence of common operations between different tokens with less hardware requirements when compared to full parallel processing of the tokens. Thus, these values can be saved in hardware registers and forwarded to successive tokens when needed. The performance of the proposed approach is much better than that of sequential processing and approaches that of full parallel processing without the need to replicate hardware resources. The proposed idea can be easily extended to the design of other filters by simply investigating the common operations between successive tokens.

References

[1] P. Coussy, M. Meredith, D. Gajski, and A. Takach, “An Introduction to high-level synthesis,” ,” *IEEE Design & Test of Comuters*, vol. 26, no. 4, pp. 8-17, Aug. 2009.

[2] P.Coussy and A. Morawiec, *High-Level Synthesis from Algorithm to Digital Circuit*, Springer, 2008.

[3] H. Deman, J. Rabaey, P. Six, and L. Claesen, “Cathedral II: A silicon compiler for digital signal processing,” *IEEE Design & Test of Comuters*, vol.3, no. 6, pp. 13-25, Dec. 1986.

[4] C.-T. Hwang, J.-H. Lee and Y.-C. Hsu, “A formal approach to the scheduling problem in high level synthesis,” *IEEE Trans. Computer-Aided Design Integrated Circuits Syst.*, vol.10, no. 4, pp. 464–475, 1991.

[5] F. F. Yassa, J.R. Jasica, R.L. Hartley, and S.E. Noujaim, “A silicon compiler for digital signal processing: Methodology, implementation, and applications,” *Proceedings of IEEE*, vol. 75, no. 9, pp. 1272–1282, Sept. 1987.

[6] C.-Y. Wang and K.K Parhi, “High-level DSP synthesis using concurrent transformations, scheduling, and allocation,” *IEEE Trans. Computer-Aided Design Integrated Circuits Syst.*, vol. 14, no. 3, pp. 274–295, Mar. 1995.

[7] A. Shatnawi, J.Ghanim, and M. Ahmad, “High level synthesis of integrated heterogeneous pipelined processing elements for DSP applications,” *Computer and Electrical Engineering*, vol. 30, no. 8, pp. 543-567, Nov. 2004.