Interface Design for the Mapping of Generalized Template Matching on Reconfigurable Systems

Xuejun Liang, Jack Jean
Department of Computer Science and Engineering
Wright State University
Dayton, OH 45435, USA

Abstract

Algorithms considered as generalized template matching (GTM) operations [1] can be accelerated by using reconfigurable systems with field programmable gate array (FPGA) hardware resources. This paper proposes a method of designing the interface among FPGA, host processor, and on-board memory for GTM operations. The goal is to support the automation of the interface design and to improve the portability of GTM design among different FPGA boards.

Keywords: Template Matching, Configurable Computing, Field Programmable Gate Array, Reconfiguration

1 Introduction

Image processing algorithms for 2D digital filtering, morphologic operations, motion estimation, and template matching share some common properties. They all involve massively parallel computations that can benefit from using massive FPGA hardware resources. In addition, each algorithm can be considered a special case of a generalized template matching (GTM) operation. As a result, designing optimal implementations for these operations can be characterized similarly and be solved systematically. In our previous work [1], we described the GTM operation and characterized the mapping of the GTM operation in terms of data allocation and buffering. We presented several mechanisms (fully buffering image data on FPGA chip, using small internal buffer on the FPGA, duplicating image data on the external memory, etc.) to support different levels of parallelism (function level, pixel level and template level). This paper proposes a method of designing the FPGA interface for GTM operations. The interface design is to put together interface components, such as host interface, memory interface, multiplexers, and controllers in VHDL according to interfacing specifications. In addition, the estimate on the area/time should be produced. Because different FPGA boards have different architectures and require different interfaces, automating the interface design will greatly speed up the GTM FPGA design and improve the design portability. The concept of such an interface design tool is illustrated in Figure 1.

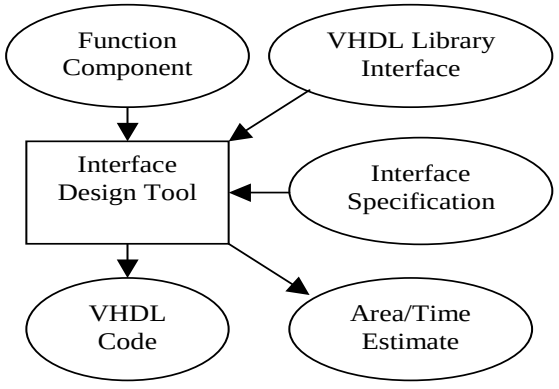

Figure 1: Interface Design Tool Concept

Section 2 presents a target board architecture, a GTM operation scenario and the target FPGA design structures. Section 3 gives the interface specification. Section 4 describes

the interface design generation. Section 5 concludes the paper.

2 Methodology Overview

2.1 Target Board Architecture

The target FPGA board may contain multiple FPGA chips, each with an array of heterogeneous memory ports. Different ports may have different sizes in terms of storage capacity and port width. Here an assumption is that all FPGA chips on the board have the same structure and there is no direct connection between them. This is because the GTM operations are relatively easy to partition among different FPGA chips. Different FPGAs can work on different regions of an image frame or with different templates in parallel without the need for inter-FPGA communications. As a result, the interface design method considered in the paper focuses on single FPGA chip. The target board architecture can be described by a set of parameters that denote the number of FPGAs, the number of memory ports, the width of memory data ports, and so on.

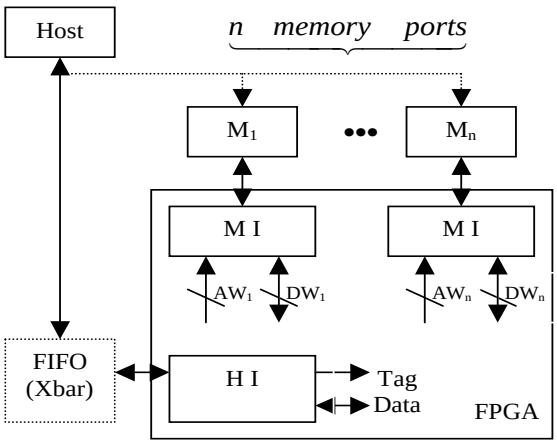

Figure 2: Target Board Architecture

Figure 2 shows such a board structure with one FGPA where the dotted line and box are optional. The host may access the on-board memory either directly or through the FPGA chip. Also the host may access the FPGA through a FIFO (and/or a Xbar). DWi and AWi represent the width of the i-th memory data port and address port, respectively. A library of VHDL components for the memory interface (MI) and the host interface (HI) is assumed to be available.

2.2 GTM Operation Scenario

The computation of a GTM operation is to move a template (or mask) pixel by pixel in scanning line order, similar to the "Sliding Window-Based Operations" (SWO) as in [2]. The GTM is more general in that all the pixels (or samples) in a SWO "window" (or mask) are involved in the window computation while in GTM a template may be quite "sparse" and only a low percentage of pixels in a "window" is involved in the computation.

In this paper, the FPGA portion handling the computation involved in applying one template to a pixel location is called a Basic Function (BF). In the case that one memory port is wide enough to provide more than one pixel in parallel, a group of BFs may be used for parallel computation [1]. Such a group of BFs is called a Unit Function (UF). Combined with an appropriate controller, a UF may be used to apply a template to one image region (i.e., one set of image rows). In this case, the UF and the controller together are called a Region Function (RF).

Figure 3 shows the sequence of tasks involved in the GTM operation from the FPGA side. The first three tasks do not have to proceed in that order. After the first three tasks, the fourth task applies the template on each pixel location in the region. The host reads back the results in the fifth task. The operation may then loop back to previous tasks. Which task to start again depends on the particular GTM application or the FPGA board used. For example, if the storage area is not large enough to contain an image frame, the first task needs to be repeated. The operation may repeat the second task if there are multiple image regions. (Note that all pixels in an image region are subject to the same set

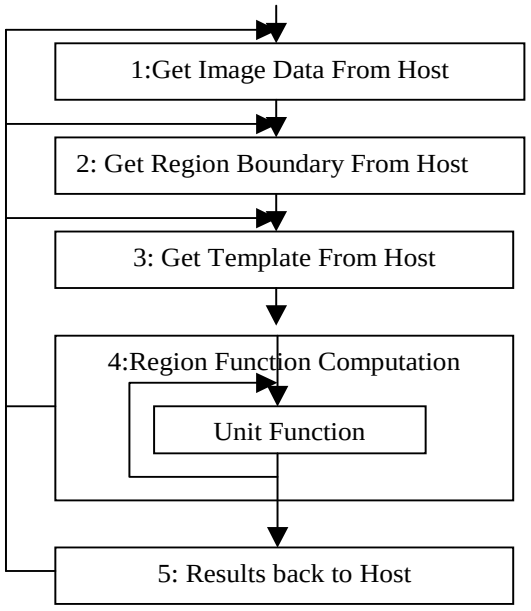

Figure 3: GTM Operation Scenario

of templates.) The third task may repeat if there is more than one template for the same image range. Note that the numbers of these iterations are not required to be incorporated into the interface controller. The computation continues until the host stops initiating tasks.

2.3 FPGA Design Structure

If the FPGA design for a GTM operation is represented hierarchically, Figure 4(a) shows all the possible top-level control/data flow when only one memory port and one RF block are considered. The dotted box HMI is a controller that coordinates the transfer of data between the host and the external memory. When the host interface (HI) and the memory interface (MI) are driven by two asynchronous clocks, a buffer inside HMI is needed. When the host is allowed to access the memory directly, the HMI is not required. The data transfer through HMI can be either unidirectional or bidirectional. Another function of HMI is to reorder the image data so as to support duplicated image data storage as necessary. The box GC is a global controller that controls all multiplexers at the top-level and coordinates the execution of RFs when there are multiple of them. In particular, the Mux_Sel signal controls the multiplexers, the Reset signal resets the RF Controller (RFC), and the Assert signal activates the RFC.

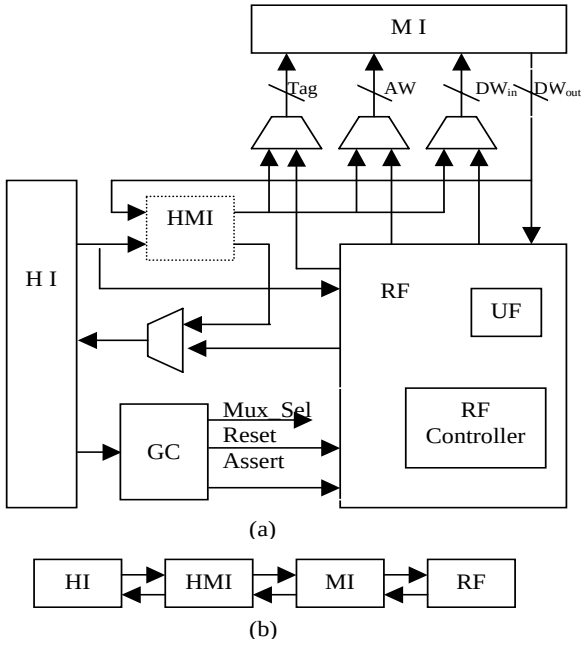

Figure 4: (a) A One-RF Design Inside an One-Memory-Port FPGA (b) The Corresponding Connectivity Graph

Figure 4(a) can be abstracted as a connectivity graph as in Figure 4(b). The GC, the RF-HI connections, and the multiplexers are not shown in Figure 4(b) because the GC and RF-HI connections are assumed necessary and the multiplexers can be deduced from the connectivity graph.

In case that a FPGA has two memory ports, the FPGA can accommodate two RFs that can in parallel handle two regions or handle two templates on the same region. (Note that there is no benefit in having multiple RFs sharing one memory port.) An alternative is to use only one RF that gets its input data from both memory ports at the same time. This should speed up the evaluation of a template on a pixel location. The connectivity graphs for these two design options are shown in Figure 5.

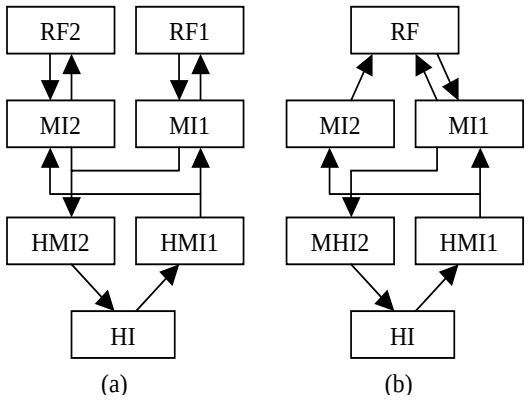

Figure 5: Two Possible Connectivity Graphs

RFC Refinement The RF controller inside the RF block is responsible for the following functions: (1) receiving the region boundary from the host, (2) receiving the template data from the host, (3) sending the result data and the status information to the host, and (4) controlling the UF to iterate through pixels in an image region and the data transfer between FPGA and memory. Therefore, the RF controller can be divided into four separate parts, one for each function, as shown in Figure 6 where RC stands for Region Controller, TC the Template Controller, RSC the Result and Status Controller, and LC the Loop Controller. The "Ok" signal from the UF means that the UF computation results are ready while the "Done" signal indicates that the UF computation is finished.

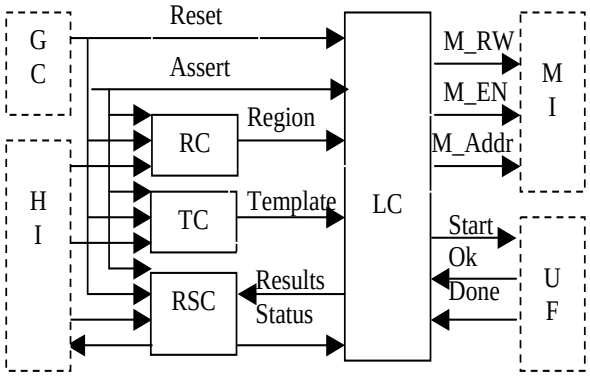

Figure 6: RF Controller Structure

2.4 Interface Design

From the above discussion, the interface design for the GTM operation is to generate the Connectivity Graph (CG), the Global Controller (GC), and the RF Controller (RFC). Note that MHI, HI, MI, and UF are assumed to be available. In order to automate the interface design process, we need to provide some interface specifications aside from the target board architecture and the interface components in libraries. They are Task Schedule (TS) and UF Timing (UFT). Figure 7 shows the relationship between the generators and the specifications, and identifies the sections where each topic is presented in more detail.

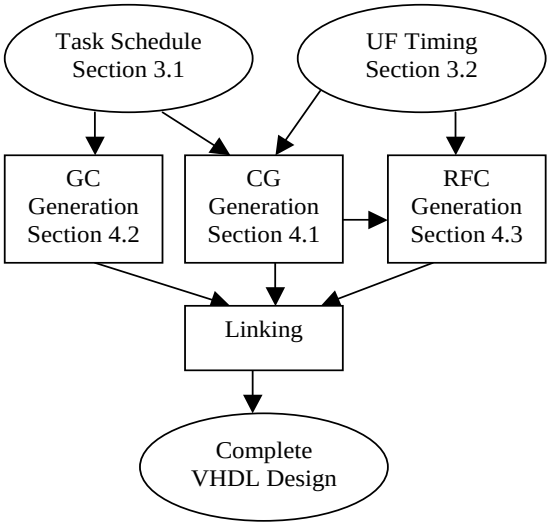

Figure 7: Specification And Generation

3 Interface Specification

3.1 Task Schedule

Figure 3 shows five tasks for a GTM operation. Each RF is supposed to execute these tasks. The task schedule is an ordering of the tasks performed by each RF. All tasks scheduled at the the same time are to be executed in parallel. When there is only one RF, the task schedule can be as trivial as (T1, T2, T3, T4, T5) where Ti stands for the i-th task. When

there are multiple RFs, there are more options in the scheduling of tasks.

For example, if two RFs work with the same templates but on different regions, a possible task schedule is shown in Figure 8. Scheduling the first tasks of both RF1 and RF2 at the same time implies broadcasting the image data from the host to RF1 and RF2. Since RF1 and RF2 work on different regions, they cannot receive the region boundary (T2) at the same time. And because they share the template, they can receive templates (T3) simultaneously.

Order	1	2	3	4	5	6	7
RF1	T1	T2		T3	T4	T5	
RF2	T1		T2	T3	T4		T5

Figure 8: A Possible Task Schedule

3.2 Unit Function Timing

In this paper we assume that a pipelined FPGA design of the UF is created prior to the design of interface, even though an optimal UF design is impossible without considering the interface part. In order to control the UF computing through pixels in a region, the feeding of data to UF, and the storing of UF results in memory, the number of pipeline stages and the timing of the pipelined UF design must be specified. The timing refers to when the input data are consumed or the results are produced, and so on.

Figure 9(a) shows a simple UF timing where stage R reads data from memory, stage W writes data to memory, and stage C computes. This specification tells us that the UF takes the first six clock cycles to consume data from memory (and possibly compute at the same time), then three clock cycles to do other operations, and finally two clock cycles to write results to memory. If the UF accesses two or more memory ports at specific stages, these stages are labeled with distinct R's or W's. Figure 9(b) shows such an UF timing where R1 and R2 denote reading of different memory ports, while W1 denotes writing to the same memory port as used by R1.

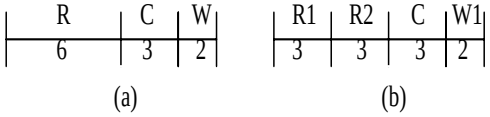

Figure 9: UF Timing

4 Interface Design Generation

4.1 CG Generation

The connectivity graph can be generated with the information provided by the task schedule, the UF timing, and the target board architecture. The task schedule and the UF timing are supposed to be produced by a GTM optimization tool still under study.

The task schedule provides the number of RFs. From the UF timing the minimum number of memory ports per UF is known. It is easy to bind the memory ports to RFs when the memory port information can be obtained from the target board architecture. There may exist more than one CG that satisfies all the constraints. In that case, it provides us an opportunity to select the best design.

The following example illustrates how the connectivity graphs are generated. Suppose that there is only one RF with a trivial task schedule, and that the UF timing is as shown in Figure 9(a). If there is only one memory port available, two connectivity graphs as shown in Figure 10(a) and (b) can be produced. If two memory ports are available, then there exists at least one more CG as shown in Figure 10(c).

It is worth mentioning that we implemented and compared these three different designs on an Annapolis Micro Systems's StarFire board with one Virtex XCV1000 FPGA chip. The FPGA areas used by the designs (a), (b) and (c) increase in that order, but the design (a) can run at 35MHz clock, the design (b) at 40MHz, and the design (c) at 45MHz. This

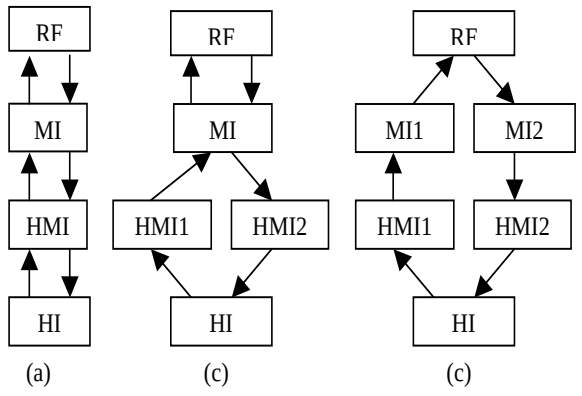

Figure 10: Three Designs

shows an area/speed tradeoff and a need to provide the area/time estimate in the case when an optimal design is desirable.

4.2 GC Generation

After the connectivity graph is obtained, the requited multiplexers can be added to support the CG. Then, the global controller, or the timing of the Mux_Sel and Assert signals can be generated from the task schedule. This is possible because the task schedule contains the information about when the host and which RF will access (read or write) the memory, and when RFs start working.

4.3 RFC Generation

RFC includes RC, TC, RSC and LC. RC, TC and RSC control the corresponding storage (region boundary, template, or result) on the FPGA. In order to detect which types of data are coming from the host and when, there is a need to create a memory map that shows a one-to-one mapping between storage types and HI signals. This map can be built by using a HI specification that will be part of the interface library. Because different FPGA boards may have different tagging patterns to indicate data communication between the host and the FGPA, different FPGA boards need to have different HI components and HI specifications in the library.

LC Generation For the LC generation, we need to know the MI specification in order to set up the memory access to and from UF. The MI specification should be associated with a set of parameterized VHDL components in the library. These components need to deal with various memory access operations, such as burst read, burst write, and so on.

The RF loop control performed by LC can be generated by using the UF timing and the connectivity graph. For example, Figure 9(a) shows an UF pipeline timing. If the connectivity graph is as shown in Figure 10(a) or (b), then the RF loop pipelining will be as shown in Figure 11(a). If the connectivity graph is as shown in Figure 10(c), then the RF loop pipelining will be as shown in Figure 11(b) where "Read" and "Write" accesses to different memory ports. In this case it should be ensured that no hardware sharing exists between "Read" and "Write" stages.

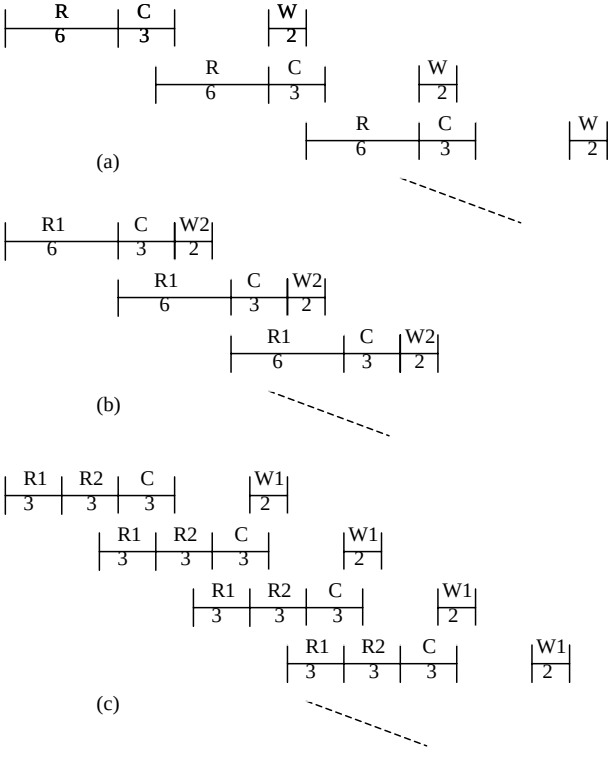

Figure 11: Loop Pipelining

Note that the UF computation has 11 stages. With the first solution, the UF computation

has a pipeline period of 8. That is, a new UF computation is started every 8 clock cycles. The second solution has a better pipeline period of 6 at the expense of using one more memory port. In both cases, the throughputs are constrained by the memory bandwidth.

Take another example. Suppose that we have an UF timing shown in Figure 9(b) and a connectivity graph shown in Figure 5(b). Then the RF loop pipelining as shown in Figure 11(c) can be used. The pipeline period of this design is 5 which is better than that of the second solution although they both use two memory ports. But the tradeoff is that with the last solution no hardware sharing should exist between the two "Read" periods and the control logic will be more complicated.

5 Conclusions

This paper describes the design of interface among FPGA, host processor, and on-board memory for the mapping of generalized template matching (GTM) on reconfigurable computers. The controller required for such an interface is also proposed. The interface design is critical to the implementation of GTM operations on different FPGA boards. In addition, solving the interface problem systematically is an important step in the development of a software tool that automatically generates FPGA designs for GTM operations. Researchers of Wright State University are at the initial stages of developing such a tool.

References

[1] J.S.N. Jean, X. Liang, and K. Tomko, "Data Buffering and Allocation in Mapping Generalized Template Matching on Reconfigurable Systems," in the Proc. of Parallel and Distributed Processing Techniques and Applications Conference, pp. 1111–1117, June 1999.

[2] C. Thibeault and G. Begin "A Scan-Based Configurable, Programmable, and Scalable Architecture for Sliding Window-Based Operations," in IEEE TRANSACTIONS ON COMPUTERS, pp. 615-627, 1999.