A Reconfigurable Bus IP For Modular Function Integration

Neal S. Stollon, Ph.D, P.E.
Infinite Technology Corp.
1778 North Plano Rd. Suite 310
Richardson, Texas 75081
neal@itc-usa.com

Abstract: Mechanisms for initialization, control, and inter-communication of IP blocks; the ability to synchronize blocks; to integrate software control; and to reconfigure IP block interconnection to adapt to changing system requirements and conditions are all useful and advantageous features for an IP based design environment. This paper discusses Bus Level Encapsulation as a method of addressing these issues of initialization, control, synchronization, and inter-communication of IP blocks. As an example of the encapsulation approach, a Reconfigurable Bus Architecture, developed by Infinite Technology Corp (ITC) to support a Distributed Instruction Word driven encapsulation and dynamically reconfigurable interconnection is discussed. Reconfigurable Bussing was developed as an integration framework for ITC IP communication and will be licensed as a supported IP product. System implementation issues and control structure, for examples of datapath IP encapsulation will be discussed.

1. Introduction:

With the availability of silicon resources to leverage multiple IP blocks in a single design, integration has become a key issue in successful IP usage. Unlike traditional design cycles, (where designers have intimate knowledge of all of the design functions), in an IP centric design environment, functions are often understood only by their peripheral I/O signals and parameters. In this environment, efficient functional encapsulation of IP to support the integration issues of initialization, synchronization, control, and inter-communication of IP blocks is a vital consideration in maintaining both the edge in time to market and improved engineering productivity that IP reuse promotes.

Encapsulation mechanisms can be addressed in many ways, dependent on the applications and types of design being targeted. As a company with product focus on datapath oriented distributed architectures, Infinite Technology Corp (ITC) has developed a dynamically reconfigurable control and bussing architecture as a primary encapsulation mechanism of internal and third party IP functions. Among the advantages of dynamically reconfigurable integration are of software control of sub-system block connectivity and configuration, and encapsulation methodology that is efficiently scalable with regard to complexity vs. architecture size.

2. An Overview of Dynamic Reconfiguration:

Dynamic reconfiguration as applied to a general class of reconfigurable architectures is a multi-faceted problem. For purposes of this paper, dynamically reconfigurable architectures are defined as those where functionality and interconnection of logic processing blocks are externally controllable on a quasi-real time basis. This is in contrast to static reconfiguration, where the loading of reconfiguration information requires that real time processing must be halted and restarted. An example of static reconfiguration is reprogramming of FPGA parts, where the device is essentially halted and local information is lost during the FPGA data load.

Dynamic controllability of reconfiguration can be understood as a function of two factors:

1. The logical features of a block architecture to support reconfigurability and
2. The amount of local memory that must be integrated with the reconfigurable blocks to control differing dynamic reconfiguration modes.

Of these two factors, the most challenging appears to be the issues of reconfiguration memory. Memory resources are a performance bottleneck to dynamic reprogramming of the reconfigurable elements [1]. Reconfiguration at a larger architectural block level can be seen to be more memory efficient for dynamic reconfiguration than small blocked approaches, since the amount of control and distribution of the memory configuration data required per unit size of IP logic being controlled is minimized. This can be seen as follows:

Consider a reconfigurable block as one where all elemental logic operations may be reordered on a dynamic basis to implement one of a given number of different functions. Let two different modular block functions be F(a), a configuration that is established for time frame T(a), and a different functionality, F(b), that is established for time frame T(b). The issue of architectural reconfiguration becomes a

question of what is the level of granularity of the elemental logic operations that can be adequately reconfigured in order to efficiently implement functions F(a) and F(b). A reconfigurable system would want to work at the highest level of functional granularity possible, since this would minimize the overall memory requirements for reconfiguration between F(a) and F(b). [2]. Granularity requirements must be traded off against other design issues such as amount of block to block interconnection and block testability requirements.

To illustrate this tradeoff of configuration granularity vs. resources, consider IP implemented on a block based architecture (such as an FPGA). For an FPGA, the elemental logic block is typically a registered universal logic block that is functionally configured via memory based control. As such, reconfiguration requires a reload of memory information, and is not dynamic. Interconnection of the configured logic blocks creates basic logical functions such as adders, decoders or counters. Further interconnection of basic functions creates "system" functions such as filters, transforms, correlators, etc. One of the key block interconnection design challenges to maintain timing performance and logic block efficiency is to keep these functional blocks locally connected.

Functional reconfiguration can only be defined at a logic block level; for all higher level functions, reconfiguration can be seen to be an interconnection reconfigurability problem. Functional reconfiguration is difficult since it requires an understanding into the architecture typically only available to the device manufacturer to ensure loss of local information. Generalized functional reconfiguration mechanisms are not realistic due to logic block and system architecture differences. For these reasons, the implementations that we will discuss focus only on interconnection reconfiguration as an user based approach to dynamic reconfiguration. For general classes of VLSI systems, block (FPGA) and otherwise, interconnect reconfiguration is largely independent of specifics of the block architectures.

As an example of an interconnection reconfiguration approach being discussed, consider a portion of a "sea of blocks" as shown in Fig. 1. An IP Execution Unit (IP-EXU) is defined as a locally interconnected modular function implemented as logic blocks with well defined input and output ports. Alternately, IP-EXUs may be viewed as synthesized, placed, and routed "hard macro" blocks or functions. Forming an interface to all the IP-EXUs is a Reconfigurable Channel Bus (RCB); a set of parameterized, single source (one IP-EXU)-multiple destination (any and all other IP-EXUs) bus channels which form the reconfigurable bus. RCB connectivity is controlled in the quasi-real time basis by updates to a Distributed Instruction Word (DIW). A core function is further defined as a set of IP-EXUs that are logically inter-connected via the RCB. Since RCB connectivity of the IP-EXUs can be dynamically reconfigured under DIW control, the architecture of a core function can be dynamically reconfigured at the datapath level, assuming IP-EXUs as fixed building blocks. By having interconnection information (and optionally IP-EXU control parameters) updateable by the DIW, the connectivity reconfiguration speed is limited only by the ability to update the DIW.

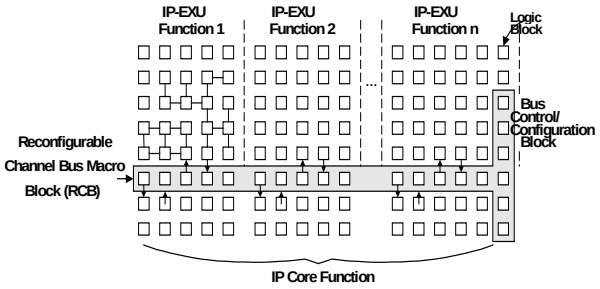

Figure 1: An interconnection reconfigurable Core

A local Bus Control Configuration block shown in Fig. 1 is part of the interconnection reconfiguration scheme. ITC's Bus Control is implemented as a loop counter and state machine driven controller that loads and stores the set of DIWs for a given operation and synchronizes and controls the access of DIW subsets for different configuration modes of the core. Single clock cycle RCB reconfiguration of a core can be achieved by synchronization of the DIW distribution.

3. Dynamic Interconnect Reconfigurability

As an example of how ITC has applied dynamically reconfigurable interconnection to generic datapath computing problems, consider a set of IP-EXUs implementing the following functions: I/O, ALU, MAC, 2-port RAM, and FIFO RAM. These are typical simple functions that may be OEM designed blocks or alternately available as soft or hard macro IP from third party sources. Other IP blocks that may be more appropriate to a given application may be substituted at the user's discretion. For the example under consideration, 8 instances of these IP-EXUs have been placed adjacent to each other and are interconnected by an 8 channel RCB to form an FPGA Core Function, as shown in Figure 2.

Each IP-EXU has at least one dedicated output that is a source to an assigned bus channel of the RCB. RCB

channel width is variable based on application. At ITC, channels are typically 17 or18 bits. When DIW control is applied to the RCB, connectivity (defined by the DIW) is established between RCB channels carrying data and the input to selected IP-EXUs. For an 8 channel RCB and 8 IP-EXUs with 2 inputs and one output, a 48 bit DIW (8 IP-EXUs x 2 inputs x 3 bit control for 8 RCB channels) provides connection control for all IP-EXUs and RCB channels. Dynamic loading and applying reconfiguration to an entire core is limited only by the DIW update rate. Additional control of different IP-EXU parameters, including configuring internal functionality of individual EXUs will increase DIW size.

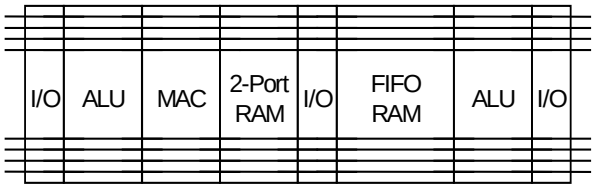

Figure 2. An 8 IP-EXU Core Function

Empirical modeling of different datapath functions has shown that most functions of interest can be supported with a memory range of 16 configurations. For more intricate or inclusive control, a number of memory options, including memory segmentation of the IP-EXU parameters have been investigated and prototyped. These options become more important as the size of the RCB and EXUs increase. ITC has implemented RCB based reconfiguration on cores consisting of up to 16 EXUs. Above this level, an "array of cores" based approach has been adopted to minimize issues of RCB routing and control modularity. RCB based integration of IP-EXU functions can be hierarchically extended to the core level (Fig.7). An advantage of an reconfigurable interconnection approach is that it is cascadable to allow independent reconfigurations of multiple core and mixed IP-EXU and core architectures.

To demonstrate the application of the reconfigurable bus architecture to example datapath applications, consider again the 8 IP-EXU core function from Figure 2. The core contains sufficient functionality for dynamically reconfigurable mapping into several algorithmic functions of interest in high-speed signal processing applications.

Figure 3a illustrates an implementation of a Discrete Cosine Transform (DCT) algorithm that, from a block dataflow viewpoint, utilizes the IP-EXU resources previously defined.

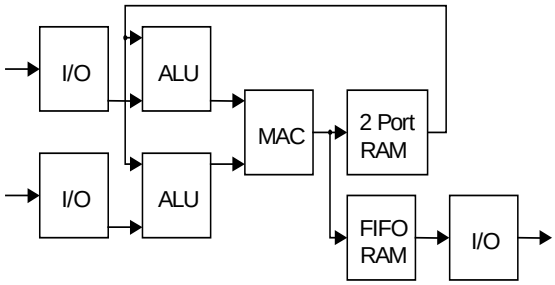

Figure 3a. A DCT Dataflow diagram

By setting the DIW control of the RCB channels to establish the block diagram connectivity as shown in Fig 3A between the IP-EXUs, the DCT function can be implemented within the core function.

Figure 3b illustrates connectivity between IP-EXUs as heavier lines across the RCB channels connecting them. The vertical dash on the connectivity line for each channel which IP-EXU that is the dedicated input source for that channel of the RCB.

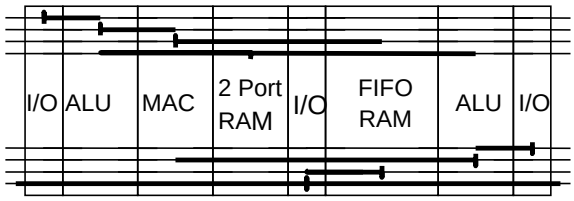

Figure 3b. An DCT Core Function

Figures 3a and 3b illustrate that a core function made up of a general number of IP-EXU macros can be connected by the RCB to implement a system level function of interest. Figure 4a and 4b exemplify how the same general IP-EXU block resources may be reconfigured to map into a different datapath function, a Finite Impulse Response (FIR) filter.

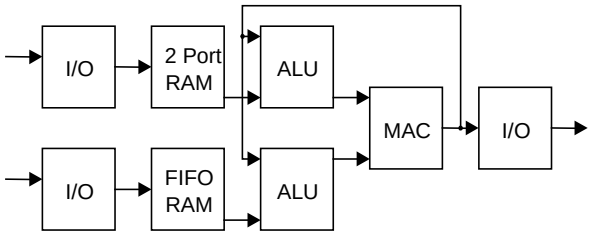

Figure 4a. A FIR Dataflow diagram

Figure 4a illustrates a block diagram of a different signal processing function that is implemented using the IP-EXU set in Figure 2. Figure 4b shows the bus level reconfiguration that implements the FIR. Since

reconfiguration is implemented solely on the DIW controlled bus, IP-EXUs form a new datapath by only by differences at the interconnect level.

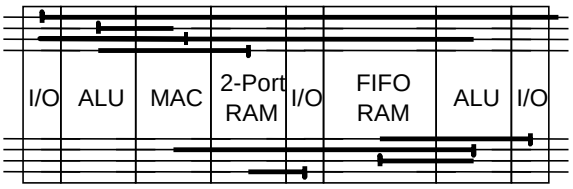

Figure 4b. An FIR Core Function

Figures 5a and 5b show another datapath DSP function (IIR filter) that may be implemented using the same IP-EXU set controlled by a different DIW driven bus interconnection. Since DSP operations use a minimal set of functions (MACs, ALUs etc.), the options of dynamic reconfiguration of a set that can be supported can be extended to include most major filter and transform types, including those used in image recognition and surveillance applications. One factor that makes reconfiguration an especially useful and interesting approach to system encapsulation is that in many application domains, key functions use largely the same types of computational resources in differing interconnection configurations.

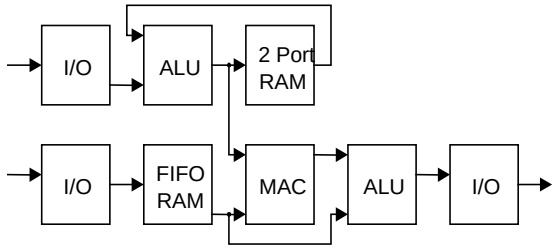

Figure 5a. An IIR Dataflow diagram

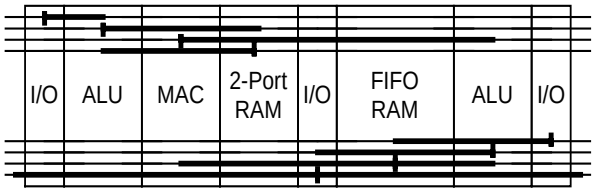

Figure 5b. An IIR Core Function

One of the keys to bus reconfiguration success, as opposed to other reconfiguration methods is that by treating the IP-EXUs as static computing resources, the memory overhead (both in size and distribution) is reduced significantly. Memory overhead has typically been one of the key issues of reconfigurable logic based approaches. By limiting reconfiguration to the interconnection and IP-EXU parameter level, memory resources can be implemented locally. The examples that have been shown are of datapath functions; however, the encapsulation concept can be applied to any type of interconnected logic blocks that are implemented in a block driven architecture with clearly defined and partitioned interconnections.

4. Hierarchical Reconfigurable Encapsulation

Provided that resources are available to implement Reconfigurable Channel Bus structures and their control, bus level approaches to reconfiguration can be applied in a relatively open ended hierarchical manner over the range of logic functions.

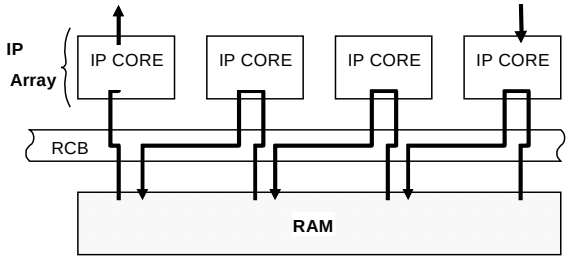

Figure 6. An RCB Based IP to Memory Interface

Reconfigurable bus encapsulation also provides a high performance mechanism for interfacing IP to memory. Innovative pipelined processing with integrated memory support can be implemented with both multi-port and partitioned memory structures. Since the RCB allows DIW driven reconfiguration of the connectivity of multiple IP cores to both data and address busses of a memory, as shown in figure 6, the bus architecture simplifies memory resource sharing. Latency is reduced compared to static interfacing methods such as crossbar switch based connectivity due to DIW control. In alternate configurations, memory enhancement IP cores (encoding, decoding, compression, error correction, etc.) may be integrated into memory architectures as a reconfigurable approach to intelligent memory implementations. Figure 7 illustrates bus architecture expansion for a number of IP blocks integrated through a hierarchical set of RCBs. For the architecture shown in Figure 7, IP core functions are dynamically reconfigured by two levels of Reconfigurable Busses (RCB 1-6, each configuring a parallel system macro function and RCB 7, forming communications path between two controller blocks). Bus reconfiguration at the intra-core level as discussed is not shown in Figure 6 or 7, but can be inferred from the previous examples. The distribution of instruction word control allows the partitioning of memory and control resources

needed for each IP core. By allowing modular control to be addressed independently for core and arrays, IP encapsulation overhead typically increases linearly with number and size of the cores. The encapsulation approach allows levels of dynamic reconfiguration to be hierarchically and independently implemented using top-down or bottoms-up methodologies.

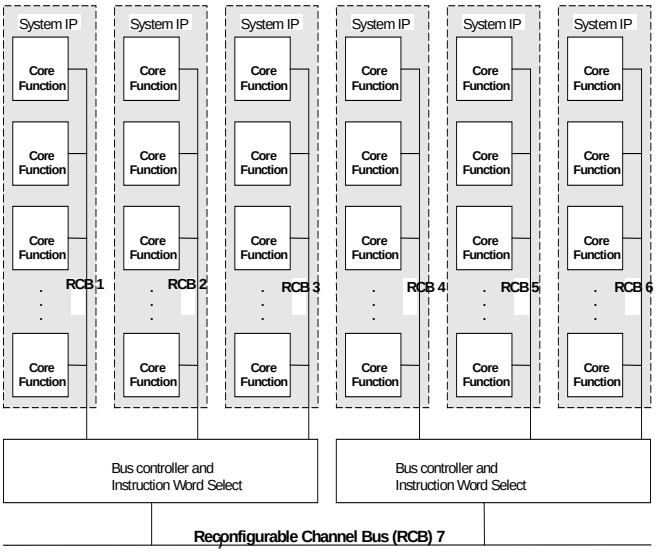

Figure 7: A Hierarchical RCB Implementation

5. Reconfigurable Bussing Architecture

In this section, we present a technical overview of the Reconfigurable Bus Encapsulation architecture. From an architectural perspective, every encapsulated core function is configured with a variable number of different IP based execution units that are made up of one of more locally interconnected functional elements (IP-EXUs), plus the following invariant components as shown in Figure 8:

- Bus/Control Configuration block
- DIW Memory
- I/O block for Initialization Control
- Reconfigurable Channel Bus
- IP-EXU Initialization Bus
- IP-EXU Flag Bus

All IP Execution units, the Controller, and the DIW Memory are assumed to be synchronous.

5.1 Bus/Control Configuration Block

The Bus/Control Configuration Block has been developed as an interface between the host processor or main controller and the encapsulated IP-EXUs. Primary controller functions include:

- Synchronization of small data transfers that use the Core Data Bus to the I/O block. During initialization, the Controller initializes and coordinates loading of data.
- Receiving host instructions, reporting status, and coordinating host or data transfers between system memory and the encapsulated subsystem.
- Program Counter control for the DIW Memory, usually by sequentially incrementing through a range of DIWs. Controller Instructions can allow the Program Counter to loop on DIW set or to branch to non-sequential DIWs. ITC's controller block supports up to four loop counters to control 4 nested loops in a DIW program.

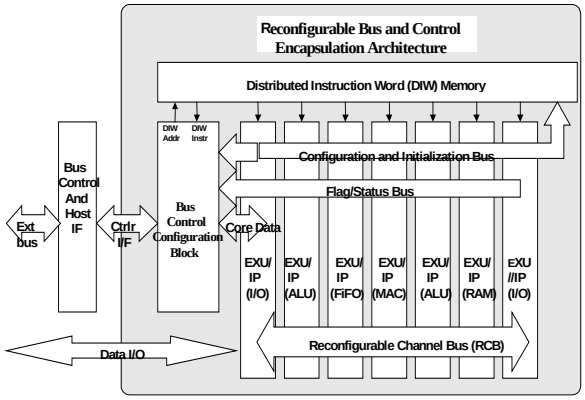

Figure 8: Reconfigurable Encapsulation Architecture

5.2 Reconfigurable Encapsulation Busses

There are five types of internal busses that are available for Reconfigurable Busing encapsulation. The Reconfigurable Channel Bus, Initialization Bus, and DIW Instruction Bus are required for architecture encapsulation and control and are discussed in more detail. The Core Data and Flag busses have been implemented for ITC developed architectures, but are optional based on specific user applications.

- Reconfigurable Channel Bus: Each IP-EXU input can access any one of the available data channels of the RCB under DIW bus selection control. In current implementations, RCB have been configured with up to 15 data busses per encapsulated core. Each EXU output is allocated to one dedicated bus of the 15 that are available.
- Initialization Bus: The Initialization Bus contains data and address busses and control signals that allow initialization data to be sent to each EXU as well as to load the DIW (and other memories).
- DIW Instruction Bus: Instructions to the IP-EXUs are sourced by the DIW memory and the Instruction Bus as shown is made of a series of

concatenated instruction word wide busses that distribute instructions to an IP-EXU block.

5.3 Instruction Word Control

The ability to maintain a coherent pipelined structure during processing operations via instruction word control is a key advantage of DIW encapsulation.

DIW based execution unit control offers precision in control of resources and operations while maintaining flexibility for software control at a systems level. DIW control allows partitions of IP-EXUs to be quasi-independent. For IP-EXUs that are temporally independent in operational dataflow, conditional branching operations, a major consideration in IW control, are minimized.

DIW control of hardware functionality can be applied to optimize application specific characteristics of a design. For example, DIW controlled gating of clock distribution to blocks supports power reduction strategies by DIW clock disabling during times when datapath indicates that the block is not being utilized. For signal processing datapath algorithms such as imaging, optimizing the pipelined operations allows fewer data dependencies and higher throughput.

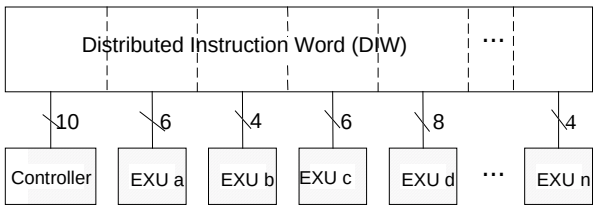

Figure 9: DIW Control Implementation

A notable advantage of DIW over other methods of distributed control and interfacing is the ability to pass parameters to a given core on a real time basis. For different EXU sets, the overall DIW length is variable. Figure 9 shows a DIW with IP-EXU blocks (with different resources) having different control lengths. DIW control allows efficient synchronization and pipelining of operations

To support use of the reconfigurable encapsulation architecture, ITC has developed software tools based on a visual spreadsheet format. The spreadsheet method of programming facilitates visualization of temporal reconfiguration operations and simplifies concurrent instruction word generation. ITC's DIW tools implementation is adaptive to each given set of EXUs used in a core since the instruction length is variable. Ongoing tool development will support high level language driven reconfiguration tools in 1999.

6. Summary

A reconfigurable encapsulation framework and architecture that provides a low overhead cost and a flexible reconfiguration and control mechanism for multiple IP blocks has been discussed. The approach has been primarily applied to the integration and encapsulation of datapath IP, but is applicable to any domain where communication and control of pipelined or concurrent IP functions are required. A distributed instruction word (DIW) driven bus control mechanism is used as an approach to implement dynamic reconfiguration.

Infinite Technology Corporation has used variations of the Reconfigurable Bus architecture as a means of encapsulation and control on several projects [3]. In parallel with hardware and architecture development, supporting software tools have been developed for automating instruction word and system verification. ITC is licensing the Reconfigurable Bus IP to third parties to address the design gap that exists between IP core availability and robust approaches to address integration, communication, and control of multiple IP blocks. Additional information is available on the ITC website: http://www.itc-usa.com.

[1] "A New Direction for Computer Architecture Research" C. Kozyrakis, D. Patterson, IEEE Computer, Nov. 1998 p24-31

[2] "Reconfigurable Architectures for General Purpose Computing" A. DeHon , MIT/AIL Technical Report 1586, Oct. 1996

[3] "A High Performance Reconfigurable Signal Processor with Distributed IW Architecture" N. Stollon, G. Landers, T. Lahutsky, Proceedings of DesignCon99/IPWorld Forum, Feb. 1999

Neal Stollon is Director of Reconfigurable System-on-a-Chip efforts and a Senior Member of the Technical Staff at Infinite Technology Corp. Dr. Stollon has previously worked in diverse areas of digital design, design automation, and program management at Texas Instruments and DSC Communications. Dr. Stollon holds a Ph.D. in EE from Southern Methodist University and is a licensed P.E.(Texas). He has published numerous papers and holds several patents in the areas of VLSI design and architecture and design automation.