Efficient Reconfigurable On-Chip Buses for FPGAs

Dirk Koch, Christian Haubelt, and Jürgen Teich
Hardware/Software Co-Design, Department of Computer Science,
University of Erlangen-Nuremberg, Germany
{dirk.koch, haubelt, teich}@cs.fau.de

ABSTRACT

This paper presents techniques for generating on-chip buses suitable for dynamically integrating hardware modules into an FPGA-based SoC by partial reconfiguration. In contrast to other approaches, our generated buses permit direct connections of master and slave modules to the bus in combination with a flexible fine-grained module placement and with minimized latency and area overheads. The generated reconfigurable buses can be easily integrated into classical design flows and can consequently be used to reduce system cost by time variant FPGA resource sharing and/or to enhance component-based system design by specifying encapsulated and exchangeable interfaces that allow to compose systems based on completely routed components.
The performance of our techniques will be demonstrated by a test system that is capable to transfer 800 MB/s and a comparison with the best competing design will show a 7-8 times area improvement, while providing a higher placement flexibility.

Keywords

FPGA, On-chip-bus, Partial runtime reconfiguration

1. INTRODUCTION

The ability of partial reconfiguration allows FPGA-based systems to adapt to different demands that can occur anytime at runtime. As a consequence, such systems can react on resource defects, time variant work load scenarios, or changing system environments (e.g. the remaining battery life time of a mobile system) by modifying fractions of the hardware.

The practical applications for partial runtime reconfiguration are manifold. For example, it may be used to exchange some modules after the start-up phase. In this case, an FPGA may be configured with test modules, boot-loader modules, or a cryptographic accelerator to speed up some authentication processes in a secure embedded system at start-up. In the following, when these modules are not longer required, we can use the same resource area for the application tasks of the system. In another embodiment, reconfiguration may be used to adapt a system to time-variant demands. For example, at daytime we may have a high demand for voice-over-IP packets in a network processing system, while later the system may adapt to some other protocols by partial reconfiguration. At nighttime, we may change the configuration to packet processing units optimized for reduced power consumption.

The main goal of utilizing partial runtime reconfiguration is to reduce the FPGA size and consequently cost and power consumption. This may help to apply FPGA technology in systems where cost and power constraints would otherwise demand (Structured-) ASIC implementations. However, partial runtime reconfiguration comes along with 1) resource overheads for providing the communication to partially reconfigurable modules and for the reconfiguration management, 2) timing penalties for the communication, and 3) a more complicated design process. These circumstances have prevented a wide usage of partial runtime reconfiguration in commercial applications. In these days, the runtime reconfiguration facilities of some FPGAs (such as all Xilinx Virtex FPGA families) are used at the most for field updates but seldom for a time-variant resource sharing.

In order to take more benefit from partial runtime reconfiguration, we have developed an efficient reconfiguration interface module [13] that allows module relocation and high speed reconfiguration with up to 400 MB/s through bitstream decompression (on Xilinx Virtex-V devices) while only consuming about a 100 look-up tables. In this paper, we present a *reconfigurable bus architecture* and corresponding tools which extends current SoC designs on FPGAs such that not only static peripherals may be integrated, but also hardware modules may be dynamically integrated into the SoC as shown in Figure 1b). This work presents methodologies that allow to change modules connected to a reconfigurable bus offering the following main features:

i) *Direct interfacing*: Reconfigurable modules have a direct interface to the on-chip bus (OCB) with the option to include bus masters to the bus.

ii) *Module relocation*: Instead of binding modules to a fixed reconfigurable area, they can be placed freely within the span of the reconfigurable OCB.

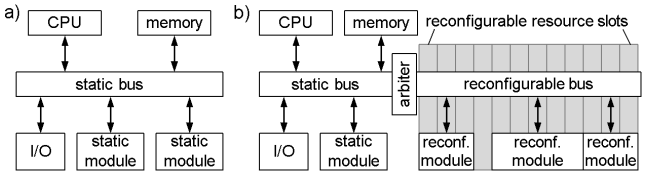

Figure 1: a) Example of a traditional FPGA-based System on a Chip (SoC). b) Alternative system that allows exchanging reconfigurable master or slave modules through partial reconfiguration in order to share reconfigurable resource slots over time.

iii) *Flexible widths*: Modules can be connected to the bus with a *fine module grid*. The module grid is a quantity for the granularity of how fine the bounding box of a module can be adjusted to just fit the area demands of a particular module.

iv) *Multiple instances*: Modules can be instantiated a couple of times and connected together to the bus.

v) *Low logic overhead* to implement the bus architecture.

vi) *High performance*: The communication bandwidth and the latency of the reconfigurable bus can compete with traditional static only systems.

All points together permit to exchange communicating hardware modules in a system at runtime in a way that can be compared to an exchange process of a card plugged into a backplane bus. But here, partial reconfiguration is used to automatically plug-in a new module to the system and the complete system may be integrated on a single FPGA.

The paper continues as follows: After giving an overview of the related work in the field of integrating reconfigurable hardware modules into SoCs at runtime in the following section, we will present our new techniques for building flexible and fast reconfigurable buses in Section 3. Afterwards, in Section 4, we demonstrate the capabilities and performance of our techniques by some examples.

2. RELATED WORK

Systems using FPGA resources in a time variant manner by exploiting partial reconfiguration have been presented in various academic publications. The main issue of this work is the interfacing of hardware accelerators to on-chip buses (OCBs) and to the static system containing typically a CPU, the memory, and the I/O interface modules. Existing approaches for the on-chip communication of partially reconfigurable modules are based on i) *circuit switching* techniques, ii) *packet switching* mechanisms, and iii) on-chip *buses* and will be outlined in the following.

Circuit Switching

Circuit switching is a technique where physically wired links are established between two or more modules for a certain amount of time. Typically, these links are implemented by more or less complex crossbar switches. The switching state of the switches may be controlled centralized by the static part of the system [19] or distributed by some logic in the crossbar switches itself with respect to the currently used routing resources within the switches [7, 2]. All these approaches prevent any partially reconfigurable module from directly accessing any memory of the system. In addition, the large multiplexers required for circuit switching allow only a very coarse-grained placement of modules. The large multiplexers have also significant propagation delays leading to decreased throughputs when using circuit switching.

Another approach [10] for two dimensional circuit switching is based on templates for turning, forking, crossing, or routing through a set of signals. By attaching these templates together through partial reconfiguration, it is possible to set a fixed routing path between some modules. As modules are obstacles for this kind of routing, the area required for routing cannot be used to implement a module's logic.

Packet Switching

The motivation for packet switching comes from the ASIC domain where more and more functional units were integrated. Here, networks on a chip (NoC) [5] have been proposed in order to deal with multiple clock domains, modularity for IP-reuse, and in order to support parallelism in communication and computation This was the motivation in [1] to integrate a dynamically reconfigurable network on an FPGA called a *DyNoC*. Here, a grid of routers is used for the communication among the reconfigurable modules. Each router decides locally based on the destination address where to send a packet further and the routers are capable to deal with obstacles. However, the approach in [1] demands a relatively fine grid of routers while the presented implementation results revealed that the logic for a single router requires several hundreds or even thousands of look-up tables (depending on the supported packet sizes).

Buses

A bus is the most common way for linking together communicating modules within an SoC. All major FPGA vendors offer tools that allow easily integrating a set of user-defined modules or IP cores into complete systems by the use of on-chip buses. Consequently, buses are good candidates for integrating also partially reconfigurable modules into a system at runtime. Most work done in this field is based on older Xilinx Virtex FPGA architectures that provide wires spanning over the complete horizontal device width and that can be used to build buses with tristate drivers [20, 16, 12, 6, 18]. However, tristate buses come along with some place and route restrictions and require timing parameters that must be met to turn buffers on-and-off. This leads typically to lower clock speeds as compared to multiplexer-based buses. Consequently, most established bus standards including AVALON, CoreConnect, and Wishbone provide multiplexer-based bus implementations with unidirectional wires and all newer FPGA architectures possess no more internal tristate drivers.

It is advantageous to implement the communication infrastructure for partially reconfigurable modules in such way that logic and routing resources that are related to the module communication are located in a homogenous manner. This allows modules to be exchanged and relocated to different positions on the chip. In the case of a bus, this means that it should be constructed in a regular fashion with regular tiles, each having exactly the same internal logic and routing layout for providing the connectivity among the tiles and to the modules. This is straightforward to implement for shared bus signals, when studying Figure 2, but it comes along with some difficulties for providing dedicated signals (e.g., Interrupts or module selects), because of their inhomogeneous routing nature.

In [20] and [15], systems are proposed where fixed resource areas with also fixed connection points to a bus interface are used for the integration of partially reconfigurable modules into a runtime system. Here, the dedicated signals are separated from the modules what is unlikely to implement on some FPGA architectures. A further drawback of this solution is that all modules have to fit into such a resource area

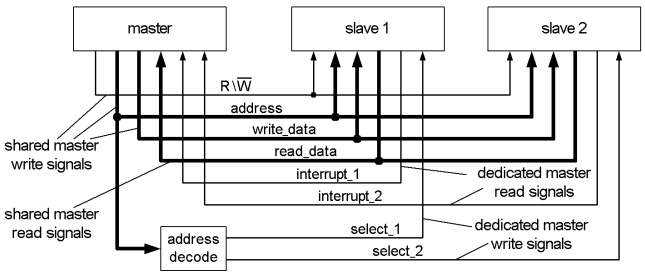

Figure 2: Example of a system consisting of a master and two slave modules communicating through a bus.

and that the resource areas could not be shared by multiple modules, even if the logic of multiple modules would fit into the same resource area. A suitable bus infrastructure for integrating reconfigurable modules should be able to connect modules of different sizes efficiently to the system.

In [9], a more flexible approach is presented that uses on-chip tristate drivers in order to build buses for reconfigurable systems. In addition, [9] presented different approaches for distributing dedicated signals to the modules connected to the bus. These approaches were either inflexible in terms of module relocation and multiple instantiation or came along with a significant area overhead or signal latency.

The related work will be discussed more detailed over the rest of the paper where we point out the advantages and novelties of our technologies against state of the art methodologies.

3. NEW TECHNIQUES FOR DYNAMICALLY RECONFIGURABLE BUSES

In this section, we will present techniques for building buses that support a direct plugging or unplugging of modules into an SoC through partial runtime reconfiguration. These buses may meet any common standards including, e.g., AMBA, AVALON, CoreConnect, or Wishbone. In addition, our methodologies aim for reconfigurable buses with numerous sockets for a fine-grained module placement in combination with high throughputs, low latencies and low area overheads. We prevent the usage of tristate drivers in order to support a wide range of different FPGA architectures which in particular includes newer devices like Xilinx Virtex-IV and Virtex-V FPGAs.

A bus is a set of signals having different purposes (see Figure 2) that establish a shared communication medium between different modules connected to the bus. These modules can be divided into two classes namely 1) master modules and 2) slave modules. Only masters are allowed to control the bus, whereas slaves are only allowed to respond to requests from a master. However, by the use of interrupt lines, slaves can notify a master to initialize a communication with a master. Table 1 presents an overview of signals found in typical on-chip buses. The table distinguishes whenever signals are driven by masters (write) or by slaves (read). Furthermore, the table differentiates if a signal is shared among different modules, or if there is a dedicated connection between a slave or a master module provided by the bus. Each quadrant in the table represents an own problem class requiring individual solutions for being efficiently implemented on FPGAs.

If multiple masters are connected to the same bus, an arbiter is required to resolve conflicts. The communication of an arbiter takes only place with the master modules by the use of dedicated bus signals for each master (e.g., `bus_request` or `bus_grant`). However, with respect to reconfigurable on-chip buses, these bus signals form no extra problem class. For instance, an interrupt request of a slave to a master is basically the same problem as a bus request to an arbiter. Thus, we will not further discuss the communication with an arbiter.

Table 1: Classification of Bus signals (see also Fig. 2).

	shared	dedicated (non-shared)

write	`address`
	`data_out`
	`write_enable`
	`read_enable`
	`byte_select`
		`module_enable`
		`bus_request`*

read	`data_in`
	`address_in`
		`interrupt`
		`wait_request`
		`bus_grant`*

* Signal for master-arbiter communication

In the following, we assume that partially reconfigurable modules are bound to rectangular regions. This can be achieved by applying some place and route constraints to the interchangeable modules. In addition, dedicated signals may be defined and constrained to be mapped always to the same wires so that a static part of a system can communicate with a dynamic reconfigurable part of the system over these wires. Such wires will be used to link together the static and the dynamic part of the system among different modules placed into an area allocated for dynamically partial reconfiguration. There exists a lack of adequate constraints on wires inside commonly used synthesis tools that prevent the tools to map a signal to a specific (user-defined) wire or a set of wires. However, this lack can be avoided by constructing macros built of logic resources that will be used as connection points in combination with the desired wires between this resources. Such macros allow to constrain signals to fixed wires in such a way that communication between a static part and a dynamic reconfigurable part of a system can be performed among different partial modules.

After introducing a generic FPGA architecture model, we will present our new techniques for implementing each problem class that is listed in Table 1 separately in the following four sections. All these different approaches are suitable for building special macros and come along with some specific advantages that can be discussed with respect to the resource requirements, latency aspects, and system integration issues.

3.1 Generic FPGA Architecture Model

Our assumed FPGA architecture model is presented in Figure 3. The shown *basic elements* (BE) contain the programmable switches and the logic resources. On Xilinx FPGAs a BE would be equivalent to a so-called *complex logic block* (CLB) [21] that consists of multiple look-up tables (LUT) for implementing the logic functions. The equivalent to an CLB on FPGAs from Altera Inc. is called a *logic array block* (LAB) [3]. However, all commercial available FPGAs follow the assumed FPGA architecture model in Figure 3. Hence, all FPGAs are suitable to implement our reconfigurable bus architecture.

To simplify the figure, we illustrated only some horizontal routing wires. The wires may be arranged in any regular horizontal or vertical fashion. Onto such architecture, we will build the logic and the routing of our reconfigurable bus architecture. The system will be divided into a *static part* that typically contains a CPU for managing the reconfigurable area and the *resource slots* for hosting the partially reconfigurable modules. The system provides altogether R

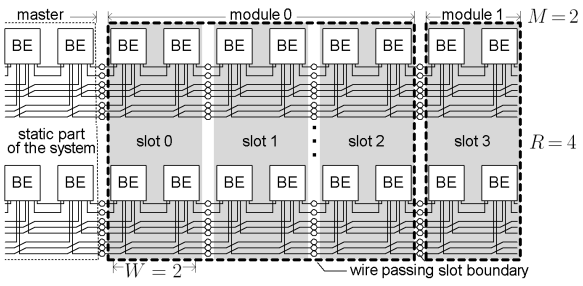

Figure 3: Simplified FPGA architecture model.

identical slots that are W BEs wide and up to M modules may be connected to the bus at the same time. A resource slot is equivalent to a socket in a traditional backplane system and modules may use resources from multiple slots in such way that only signals from the reconfigurable bus will pass a module's border. Note that some commercial FPGAs may contain a few irregularities in the otherwise homogeneous architecture of the device. For instance, some FPGAs provide dedicated RAM blocks that may require to widen selectively resource slots that contain such irregularities in order provide a homogeneous structure of the bus with respect to the wires that cross a resource slot boundary.

In all following figures that show some bus details, the static part is located at the left side of the bus and the bus is horizontally aligned. This matches best to Xilinx FPGAs that feature a column based configuration scheme. However, the presented techniques can be transferred easily to build reconfigurable on-chip buses towards any direction.

3.2 Shared Write Signals

Examples for write signals shared by multiple masters and slaves are address lines, write data lines, and read or write control signals. For dynamically reconfigurable buses, all approaches that can be found up to now are based on a *macro approach* [17] that requires some logic resources inside every module connection point for shared signals driven by a master. Thus, a 32-bit data bus would occupy also additional 32 look-up tables (or some other logic resources) multiplied by the number of resource slots R just for supplying the write data from the bus to the reconfigurable modules. Instead of connecting module signals with bus signals *explicitly* through connection macros (see Figure 4a)), our approach allows to connect shared write signals directly to the modules. Such a direct connection to a bus signal is not bounded to a fixed location. The only restriction is that such a connection does not utilize routing resources outside the bounding box of the module. Therefore, we call this an *implicit connection*.

In order to allow a relocation of modules connected to the bus, we have to constrain the routing such that bus signals occupy always the same equivalent routing wire within and at the edges of the resource slots. This requires a dummy termination resource right to the last module slot, as shown in Figure 4. Note that one logic resource may be used to terminate multiple bus wires of a master write signal. A LUT, for example, can terminate a wire at every input, thus making the overhead for the dummy sink negligible. When using the implicit connection technique, the amount of k-bit look-up tables L_{SW} required to distribute S_{SW} shared write signals is:

$$L_{SW} = S_{SW} + \left\lceil \frac{S_{SW}}{k} \right\rceil \quad (1)$$

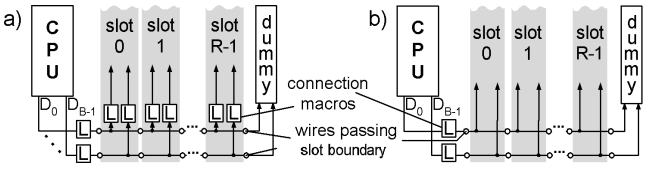

Figure 4: Distribution of shared write data from the master (CPU). In the common case a), each signal has to pass a connection macro (e.g., a LUT) in each slot of the bus (*explicit connection*). In contrast, our technique b) allows connecting modules directly to the wires forming the bus (*implicit connection*).

3.3 Dedicated Write Signals

A bus master usually demands some control wires connected directly to an individual slave where such a particular signal is not shared among other slaves. An example for such a dedicated control signal is a `module_select` signal that is equivalent to a `chip_select` or a `chip_enable` signal. In the following, we will describe how a master can activate a specific `module_select` signal by applying an according address onto the bus. A straight implementation for distributing dedicated master write signals would be to use an inhomogeneous routing scheme as presented in Figure 5a). This has been demonstrated in [18] and [6] and comes along with some drawbacks for flexible partial reconfiguration as mentioned in Section 2.

To overcome the limitations of a non-uniformed routing of module select signals, different approaches have been proposed and the most practical ones can be summarized into three groups: 1) A *slot-wise rotated module select distribution scheme*, as shown in Figure 5b), is discussed in [9]. The next group 2) is based on *fixed resource slot addresses* and has been proposed in [11] and was also used in [8]. This approach is tailored to Xilinx FPGAs, where the flip-flops in the basic elements keep their state despite a configuration process. Some of these flip-flops have been used to store an individual fixed address that is compared within each resource slot with an address supplied by a master to the bus. The last group 3) uses a counter in each resource slot to decrement the incoming address that is send further to the next consecutive slot performing the same operation. The slot getting a zero at its input will be the selected one. This approach results in an unlikely deep combinatorial path of counter stages. However in [14] this technique was efficiently used for a serial bus to access individual reconfigurable modules in a system where access times do not matter.

In all these three groups, the amount of look-up tables required for a complete bus implementation of a system providing R resource slots scales with $R \cdot log_2(R)$ LUTs for the address decoder in 1), the comparator units in 2), or the counters in 3). Thus, it is useful only for smaller systems providing only a few slots.

Reconfigurable Select Generators

So far, for all presented approaches, the system accessing a reconfigurable module must be aware of the resource slot position in order to access a particular module. For instance, this demands that a software driver for a reconfigurable module may have to update the base address if the module could be placed to different positions at the bus. In order to simplify the hardware-software interface, we designed so called *reconfigurable module select generators* where modules can be accessed individually independent to the placement by configuring a module address directly into the resource slots providing the bus sockets. This configuration process is carried out on two levels. Firstly, after loading a module onto the FPGA device through partial reconfiguration (device level), the module will be in a state where it is deactivated (e.g., by applying a module wide reset) and where

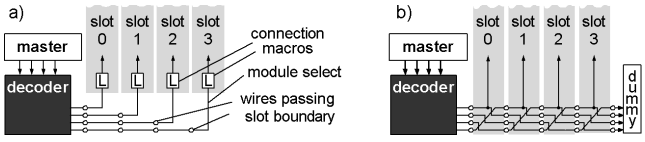

Figure 5: Distribution of module enable signals a) using inhomogeneous routing and b) homogeneous routing formed by rotating module select signals.

it is logically isolated from the bus. In this state, the module select generator can be configured on the second level, the system level, in order to write the particular module address upon which the specific `module_select` signal is to be activated in the following.

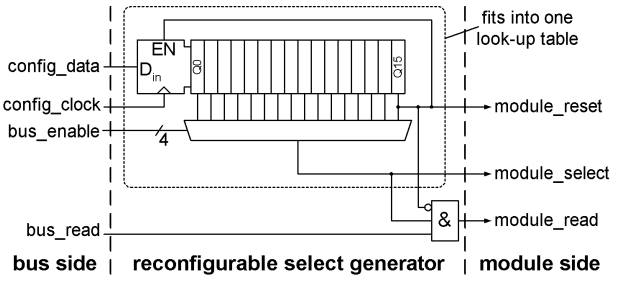

Figure 6: Reconfigurable select generator based on a shift register primitive. At FPGA reconfiguration time, the shift-register is enabled by configuring a `1` to all LUT values (`Q0`,...,`Q15`). In the following, a configuration word starting with a `0` is shifted into the LUT for self locking the register after shifting a complete new table into the LUT (`Q15`=`0`). This automatically releases `module_reset` and the register is used in the LUT mode to generate the `module_select` signal by decoding the `bus_enable` signal.

Figure 6 presents the approach of our methodology tailored to distributed memory FPGA architectures, such as all Xilinx Virtex devices, where look-up tables can also be used as memory and as shift registers. In addition, Figure 7 gives an example of how such reconfigurable select generators are used within the bus. Whenever the FPGA is configured for plugging in a new module, one look-up table that is implementing the logic of a reconfigurable select generator will be used in the shift-register mode and it will be initialized completely with ones. Thus, the cascade output `Q15` in Fig. 6 becomes one, too. This will enable the shift register and additionally activate the `module_reset` signal provided to the attached module. After this first configuration step, the look-up table of the reconfigurable module select generator is armed to be filled with a function table for decoding the `module_select` output based on the `bus_enable` input in a second configuration step. This is performed by shifting in a zero followed by typically a one-hot encoded value by the use of the `config_data` and the `config_clock` inputs. The zero value in the beginning will automatically lock the shift register and will further release the `module_reset` signal when the second reconfiguration process finishes. Note that the select generator can be used as a reconfiguration sensor that indicates at the `module_reset` output if a resource slot was involved in a reconfiguration process. The rest of the look-up table values depends on the address of the `bus_enable` signal at which the `module_select` output is to be activated. For instance, if we take the reconfigurable select generator in Figure 6 and if we want the `module_select` signal to be active only when the `bus_enable` signal is equal to `0011`, then we have to shift the value `0001 0000 0000 0000` into the look-up table. By configuring more than one `1` value to the table it is possible to use the configurable select generator for multicast operations. For instance, let us assume that we have a module 1 mapped to the `bus_enable` address `0001` and a module 2 mapped to `0010` and that both modules should further be accessible at the multicast address `0100`. Then we have to shift the value `0100 1000 0000 0000` into the look-up table of the select generator in module 1 and the value `0010 1000 0000 0000` into the LUT in module 2. Note that

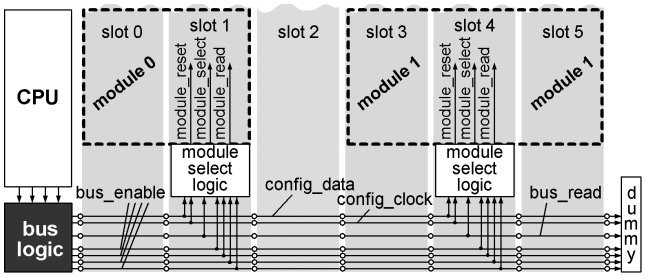

Figure 7: System composed of a CPU and some modules of different widths. Each module requires one module select block that contains the reconfigurable select generator shown in Figure 6.

in the case when `bus_enable` is equal to `1111`, no module is selected, because this address is reserved for the cascade output `Q15` locking the shift register. Thus, it is possible to generate $2^k - 1$ individual module select signals when using one k-input LUT as a shift-register inside the reconfigurable module select generator. On all Xilinx Virtex FPGAs it is possible to cascade the shift register with an additional flip-flop within the logic blocks, what can be used to generate `module_select` signals for up to 2^k individual modules.

With this enhancement, the complete amount of look-up tables L_{DW} required to provide S_{DW} dedicated master write signals in all R resource slots is for systems with up to M simultaneously running modules:

$$L_{DW} = R \cdot S_{DW} \cdot \left\lceil \frac{M}{2^k} \right\rceil + I_{config}. \quad (2)$$

Where I_{config} denotes the amount of LUTs required to implement the configuration interface for the second configuration step. As k is at least 4 in current FPGA architectures L_{DW} becomes $R \cdot S_{DW} + I_{config}$ for most practical systems. Thus, we can provide high flexibility and simple system integration at little hardware cost.

3.4 Shared Read Signals

As a counterpart to shared write signals, the bus has to provide some signals in backward direction for driving data from a selected module through some shared wires to the static part of the system. An example for shared read signals is the read data bus from several slaves to a master located in the static part of the system. Typical on-chip buses for completely static systems use unidirectional wires and some logic in order to avoid on-chip tristate buses as demonstrated in Figure 8a). This structure can be implemented in a regular *distributed read multiplexer chain* that is especially suitable for partial reconfiguration as presented in Figure 8b). This technique has been used in [11], [8], and [4]. However, using distributed read multiplexer chains for connecting partial reconfigurable modules by a bus leads to very long combinatorial paths and consequently to low bus throughputs. This is unlikely to be solved by the use of pipeline registers within each resource slot, because this would lead to position dependent latencies. Instead to this, we accelerated the simple read multiplexer chain by interleaving N chains in such a way that the output of an OR-gate from one resource slot is connected to a slot that is located N positions further towards the static part of the system as shown in Figure 9.

Beside the latency issues, the simple chained read multiplexer chains for connecting partial reconfigurable modules by a bus require $L_{SR}^{SC} = (1 + R) \cdot S_{SR}$ look-up tables for connecting S_{SR} shared master read signals in each resource

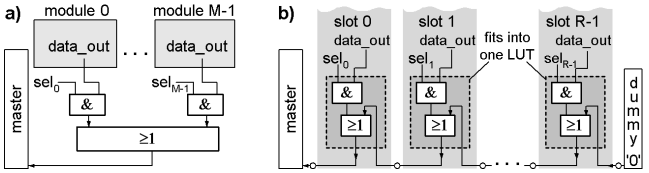

Figure 8: a) Common implementation of a shared read signal in a multiplexer-based bus. b) *Distributed read multiplexer chain* implementation that is suitable for partial reconfiguration.

slot and for the resources providing the connection to the static system (not shown in the figures). We do not further consider the required dummy sources as they can be shared with other dummy LUTs. In the case of N interleaved read chains, it requires $L_{SR}^{IC} = (\lceil \frac{N-1}{k-1} \rceil + R) \cdot S_{SR}$ k-bit LUTs to implement the final OR-gate and the read multiplexer chain. When $N \leq k$, then L_{SR}^{IC} is equal to L_{SR}^{SC}.

For increasing the placement flexibility (slot position and module width), it is desirable to reduce the resource slot width W while increasing the number of resource slots R. Unfortunately, this would enormously increase the look-up table count. In addition, this would waste logic inside the resource slot, because modules will typically occupy more than one slot and a module requiring r resource slots would leave $(r-1) \cdot S_{SR}$ look-up tables unused, because one plug would be sufficient to connect this module to the bus. Furthermore, practical experience showed that the interface width is usually related to the complexity (=size) of the according module and consequently to the amount of required resource slots. For instance, a master module may require more shared read signals for driving its address and data signals on the bus[1] and demands more logic resources for implementing the bus logic as compared to a simple slave module. Hence, wider interfaces with more shared master read signals are usually only required for modules occupying multiple resource slots.

This observation has led us to the *multi-slot read technique* that provides only a subset of all shared read signals within a resource slot while allowing to combine consecutive slots for connecting more read signals to the bus as demonstrated in Figure 10. In this approach, again N distributed read multiplexer chains are interleaved for reducing the signal latency. In order to establish a better relationship with existing systems, let us assume that each signal shown in the figure represents one byte and that the data width of the bus is up to 32-bit. Then `module 0` has a 16-Bit interface while `module 1` uses the complete interface width of the bus. In our example, modules may have interface widths of 8, 16, 24, and 32 bits, thus requiring at least 1, 2, 3, or respectively

[1]According to our classification of bus signals in Table 1, the address output of a master port is a shared read signal with respect to the static part of the system.

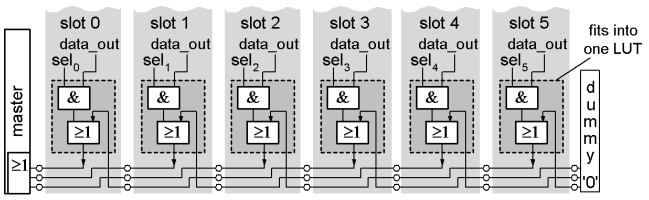

Figure 9: Multiple interleaved read multiplexer chains allow reducing the bus latency.

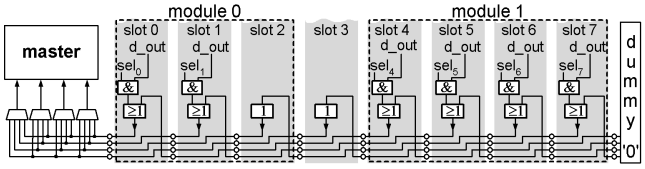

Figure 10: Multi-slot read technique with N=4 interleaved read multiplexer chains and additional attached alignment multiplexers.

4 resource slots. Additional *alignment multiplexers* in front of the master arrange the order of the incoming sub-words according to the master interface and they allow to place modules regardless to the first bus socket. This requires a small register file, that stores for each module the position of the most left used bus socket modulo N for controlling the alignment multiplexers, e.g., a 0 for `module 1` in the example in Figure 10. This register file can be implemented efficiently on distributed memory FPGA architectures by using a few LUTs as a dual-ported RAM primitive.
If the alignment multiplexer is implemented with two LUTs for each shared read signal S_{SR}, then the amount of look-up tables for implementing the multi-slot read technique with N interleaved chains and R resource slots is:

$$L_{SR}^{MR} = 2 \cdot S_{SR} + \left\lceil \frac{S_{SR}}{N} \right\rceil \cdot R. \quad (3)$$

Under all circumstances, the bus logic has to prevent modules to drive data to a shared signal if the module is not selected. This is ensured by the reconfigurable select generators (see Section 3.3) by connecting the select inputs of a read multiplexer chain (abbreviated with `sel` in the figures) for one module to the `module_read` output of the according reconfigurable select generator within the same slot.

3.5 Dedicated Read Signals

The last class of bus signals are dedicated (non-shared) read signals that allow one module to transmit some state information to the static part of the system, e.g., an interrupt request to a master. In [18] and [6], this was implemented using an inhomogeneous routing scheme (analogous to the scheme in Figure 5a) with changed signal directions) what is inapplicable for flexible module placement. A homogenous distribution of dedicated read signals was implemented in [8] by the use of a demultiplexer within each resource slot for connecting the read signal from a specific module to an associated wire within the bus. This requires one wire per resource slot inside the bus for allowing one dedicated read signal per module. Furthermore, the demultiplexers in [8] are implemented by using one switching resource (either a LUT or a tristate driver) for every dedicated read bus wire within every resource slot. Therefore, the resource requirements scale with R^2 with respect to the number of resource slots R offered by the bus, thus, being unacceptable for our purpose.

Time-Multiplexing
In many systems, it may be acceptable to distribute the state of a dedicated read signal with a latency of a few clock cycles, e.g., in the case of interrupt requests. Then, our approach presented in Figure 11 is best fitting, where interrupts are captured time-multiplexed from a reconfigurable bus. Whenever a module has an interrupt output, this signal will be connected to a distributed read multiplexer chain (analogous to the one in Figure 8b)). These multiplexers are used for periodically driving the `module_IRQ` state mod-

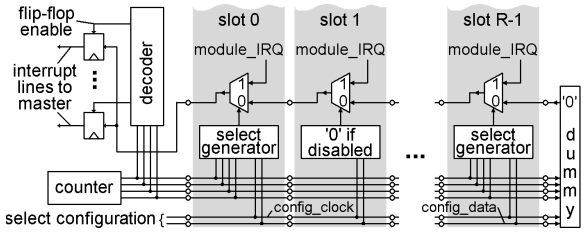

Figure 11: Time-multiplexed connection scheme for dedicated master read signals (interrupts).

ule wise to the read multiplexer chain. The respective states are stored in flip-flops connected to the master. In all Xilinx Virtex FPGAs, there exist multiplexers that are controlled by attached look-up table outputs, thus one LUT per resource slot is sufficient for implementing the select generator (as shown in Figure 6) and the attached multiplexer. This holds true for up to 15 interrupt sources in the case of 4-bit LUTs, what is sufficient for most practical systems. When implementing the time-multiplexing technique, the total amount of look-up tables required to connect the interrupt outputs of M modules to a master in a system with R resource slots is therefore:

$$L_{DR}^{TMUX} = R + M + log_2(M) + I_{config}. \quad (4)$$

The logic for the configuration interface I_{config} can be partially shared with the interface required for the chipselect generation (see Section 3.3). Note that the configurable select generators in Figure 11 can be used for flexible assigning the interrupt sources of the different modules to a specific interrupt signal connected to the master, thus, supporting systems with hardware prioritized interrupt processing. The worst case latency to store a `module_IRQ` state in the according output flip-flop is $M + 1$ clock cycles.

Configurable Bus Request Demultiplexers
While it might be sufficient to distribute an interrupt signal in a time-multiplexed manner, this kind of distribution is disadvantageous for latency critical signals, like a bus request signal from a master to an arbiter. We solved this issue by the help of *configurable bus request demultiplexers* as presented in Figure 12. In this technique, multiple configurable multiplexer chains are interleaved along the bus (like three chains in the example in Figure 12). Let us assume an example, where we want to plug in `module 1` by partial reconfiguration to the slots 3 to 5 as shown in Figure 12 and that this module must connect the dedicated master read signal $\texttt{bus\_request}_1$ to a master (or arbiter) located in the static part of the system. Then we will connect $\texttt{bus\_request}_1$ to all possible multiplexer inputs within the resource slots occupied by `module 1`. After completing the partial reconfiguration process, all shown multiplexer control flip-flops in the resource slots 3 to 5 are temporary enabled by a signal supplied from the reconfigurable enable generators inside the according slots (see Section 3.3). This allows us to write a one-hot encoded configuration word over the `config_data` lines into these flip-flops such that $\texttt{bus\_request}_1$ is demultiplexed inside one resource slot (here slot 4) to one of the multiplexer chains. The multiplexers of all other chains will then let pass the signals from the preceding slots.

If the system provides R resource slots with N interleaved configurable multiplexer chains and if up to S_{DR} dedicated read signals have to be connected to the static part of the system, the amount of LUTs required when using the configurable bus request multiplexer technique is:

$$L_{DR}^{BRM} = \left\lceil \frac{S_{DR}}{N} \right\rceil \cdot R + I_{config}. \quad (5)$$

Note that if N is chosen less or equal to the resource slot count of the smallest module requiring such a dedicated read signal, a simple first-fit strategy will always find a valid configuration for the multiplexer chains.

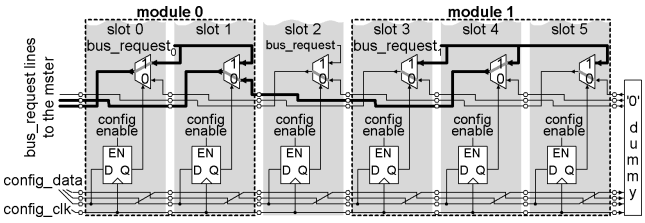

Figure 12: Configurable bus request demultiplexers allow to connect a `bus_request` signal to a dedicated interleaved multiplexer chain.

3.6 Appraisal of Results

In this section, we have presented various techniques for efficiently implementing regular bus structures for any kind of bus signal. Even if we design systems with lots of resource slots, the amount of additional logic required for providing the communication will remain within reasonable limits. The benefit of our techniques becomes obvious if we compare it with the most advanced competing solution. In [8] an advanced bus based communication infrastructure was presented that aims also on a flexible module placement, varying module sizes, and the capability to instantiate a module multiple times. The authors presented a case study of a 32-bit Wishbone compatible bus implementation with 16 resource slots, each capable to host a master or a slave module. Already the bus required 4072 dense packed slices (=logic units providing two 4-bit LUTs) and 2380 tristate drivers what is equivalent to 18% of the complete logic resources available on the test device (a Xilinx VirtexII-4000-4 FPGA). By applying our techniques, the implementation of the case study in [8] would need only 1054 LUTs (≈550 slices) and no tristate drivers or 2.3% of the logic resources. In this case we would offer the double density of resource slots ($R = 32$) within the same overall width of the bus. Note that we are the first providing such high densities for the placement of reconfigurable modules. If we interleave $N = 4$ chains, the look-up table count would be as follows:

- $L_{SW} = 88$ LUTs for altogether $S_{SW} = 70$ *shared master write* signals (32-bit data, 32 address lines plus 4 byte select signals, and 2 control signals) (see Eq. 1).
- $L_{DW} = 96$ LUTs for together $S_{DW} = 3$ *dedicated master write* signals (a module select, a read enable, and a master grant) (see Eq. 2).
- $L_{SR} = 714$ LUTs for all $S_{SR} = 69$ *shared master read* signals (32-bit data, 32 address lines plus 4 byte select signals, and 1 read/write signal) (see Eq. 3).
- $L_{DR} = 128$ LUTs for $S_{DR} = 16$ *dedicated master read* signals (bus request or interrupt) (see Eq. 5).
- 28 LUTs for the *configuration interface* used at the system level to assign addresses and interrupts to the individual modules.

Beside the massive logic savings, our solution would further allow to place modules in a finer grid and to take modules in smaller bounding boxes. In addition, as shown in the following section, our solution has lower latencies for providing a higher throughput.

We can also roughly estimate how much logic a completely static design would require to implement a bus for a system that is not capable for exchanging modules by partial reconfiguration. If we assume a system with altogether 6 masters (each one also providing a slave interface) and 6 additional slaves, it requires $(32 + 4) \cdot 5 = 180$ LUTs[2] for the address-multiplexer and another $32 \cdot 10 = 320$ LUTs for the read-data-multiplexer with 12 inputs. If we assume that all dedicated signals and shared master write signals are distributed without additional logic, our technique requires about the double amount of logic resources to provide a reconfigurable system backplane bus. Thus, we can conclude that applying our technology starts to amortize even if just relatively small parts of the resources are shared over time and the benefit increases rapidly with the amount of shared FPGA resources.

4. EXPERIMENTAL RESULTS

In this section, we will demonstrate that our reconfigurable bus architecture allows a high throughput on available FPGAs. In order to determine the throughput on different FPGAs, we will firstly present a parametrizable timing model and some synthesis results to determine the signal latencies for different implementation variants of the bus. After this, we reveal details of an test system that is capable to transfer up to 800 MBytes/s over our bus technology without any timing or logic influence due to partial reconfiguration processes.

4.1 Timing Behavior

The maximum throughput of a bus is given by the data word width B of the bus multiplied by the reciprocal of the propagation delay t_{prop} of the most critical combinatorial path. *With respect to our proposed reconfigurable bus architecture, we found the critical path on data read operations.* This path consists of a forward path with the module se-

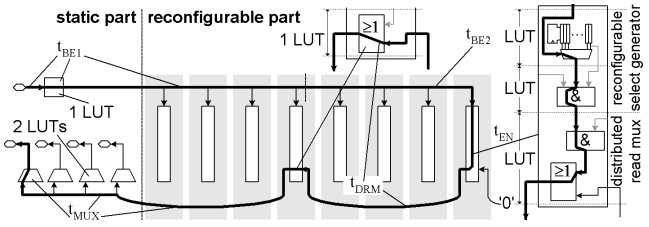

Figure 13: Propagation delays of read operations.

lect logic (see Figure 6 and Figure 7) and a backward path consisting of a distributed read multiplexer followed by a final multiplexer for aligning the correct byte order (see Figure 10). We can define a timing model for this path that is independent of the final FPGA implementation based on the bus architecture presented in Figure 13. If the bus architecture is built upon N interleaved read multiplexer chains, the annotated delays in the architectural view have the following meanings:

[2]The values for the required 4-bit look-up tables to build a 6:1-multiplexer are from the QuartusII IP-MegaCore-Wizard obtainable by the Altera cooperation. According to this, a single 6:1-multiplexer requires 5 LUTs and a 12:1-multiplexer requires 10 LUTs, respectively.

Table 2: Propagation delays (in ns) reported for some examined FPGAs.

FPGA	Spartan 3 -4		Virtex II -6		Virtex IV -11	
	$W = 1$	$W = 2$	$W = 1$	$W = 2$	$W = 1$	$W = 2$
t_{BE1}	1.637	1.972	1.095	1.581	1.828	1.415
t_{BE2}	0.335	0.670	0.486	0.972	0.374	0.748
t_{EN}	1.239	1.239	1.988	1.988	1.244	1.244
t_{DRM}	1.563	1.535	0.986	1.053	0.658	0.691
t_{MUX}	3.169	4.098	2.068	2.099	1.531	1.801

t_{BE1} denotes the time required for a `bus_enable` signal to pass a connecting dummy resource plus the propagation delay for the wiring to the reconfigurable select generator within the first N resource slots (see Fig. 7).

t_{BE2} specifies the additional time to reach the select generator that is N resource slots further away.

t_{EN} gives the propagation delay for the reconfigurable select generator (see Fig. 6) plus the chunk of a distributed read multiplexer that feeds in the data.

t_{DRM} specifies the time to pass the look-up table of one distributed read multiplexer stage plus the propagation delay on the wiring between two chunks.

t_{MUX} is the propagation delay for the alignment multiplexers and the routing within the first N resource slots.

All these timing values specify the worst case propagation delay over all considered wires within a particular case, thus, leading to conservative estimations. The values allow extrapolating the propagation delay to any size of the bus:

$$t_{prop} = \underbrace{t_{BE1} + t_{EN} + t_{MUX}}_{fixed} + \underbrace{\lambda \cdot (t_{BE2} + t_{DRM})}_{variable} \qquad (6)$$

The factor λ specifies the width in addition to the first N resource slots of the complete reconfigurable bus as a multiple of the number of interleaved distributed read multiplexer chains N. Thus, the complete bus offers $R = (\lambda + 1) \cdot N$ resource slots, each capable to connect a module.

As we have obtained all experimental results with Xilinx FPGAs, where a basic building block is called a CLB (complex logic block), we will bound our following examinations to CLBs only. A CLB provides 8 look-up tables, thus a resource slot being W CLBs wide and H CLBs high provides $8 \cdot W \cdot H$ LUTs and the complete reconfigurable resource area provides $8 \cdot (\lambda + 1) \cdot N \cdot W \cdot H$ LUTs respectively. We have to reduce these values by the bus logic requiring a few percent of the logic, as shown Section 3.6).

We have implemented buses with a data word width of $B = 32$ for the resource slot width of $W = 1$ and $W = 2$. Note that such high densities have never been reported before and the best competing solutions [8] and [12] achieve just a W of 4. The reported propagation delays for the critical read data path are listed in Table 2 and Figure 14 displays the extrapolated delay over the number of attached resource slots λ. The Figure allows to determine possible bus parameters for a given time budget. In the case of high speed buses e.g., with 100 MHz ($t_{prop} = 10$ ns) we see that only narrow buses with only a few resource slots are implementable. For instance, the slowest device, the Spartan3-4 would provide up to $R = (2+1)*4 = 12$ slots when W is chosen to 1. Note that for $W = 2$ the system may only provide 8 resource slots for all partial modules, but the complete Bus would be $W \cdot R = 16$ CLB-colums wide.

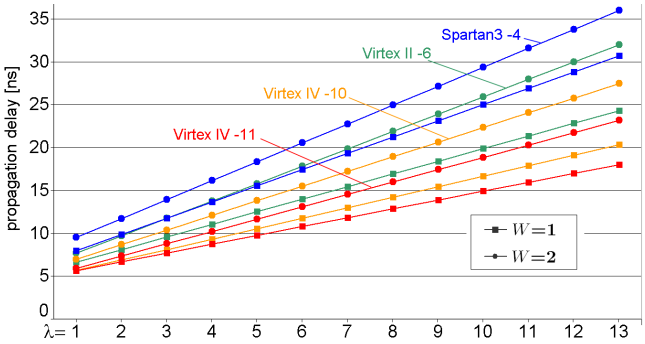

Figure 14: Extrapolated propagation delays for different FPGAs over λ for the widths W=1 and W=2.

In order to overcome this limitation, we propose to pipeline the bus communication. Note that also static designs require additional effort for the implementation of high speed systems. This means that we are still able to start a communication at any clock cycle, but this time the data arrives with a latency of one additional clock cycle at the output while transferring a new data word each cycle. Such a bus protocol is supported for the embedded hard IP processors (a PowerPC405 Core) available on Xilinx Virtex2-Pro FPGAs [21]. We want to ensure that the latency of the reconfigurable bus is independent to a module location. Thus, pipelining is not suitable within a distributed read multiplexer.

By analyzing the timing model in Figure 13, we determined to place a pipeline register between the forward path and the backward path containing the distributed read multiplexer chains. We have the choice to place the pipeline register behind or in front of the block containing the reconfigurable select generator. As a consequence t_{EN} will be considered either in the forward path or in the backward path. The worst case propagation delay of the reconfigurable bus with one pipeline register is:

$$\begin{aligned} t_f &= t_{BE1} + \lambda \cdot t_{BE2} \\ t_b &= t_{MUX} + \lambda \cdot t_{DRM} \\ t_{wc}^{pipe} &= max\Big\{max\{t_f, t_b\}, min\{t_f, t_b\} + t_{EN}\Big\} \quad (7) \end{aligned}$$

The combined worst propagation delay t_{wc}^{pipe}, when using pipelining, is the maximum of the faster path plus the enable time t_{EN} and the slower path (in both cases the forward path and the backward path).

Analogous to Figure 14, we extrapolated the pipelined worst case propagation delay t_{wc}^{pipe} with respect to the number of attached interleaved resource slots λ. The results are presented in Figure 15 and reveal a significant improvement as compared to the unpipelined version in Figure 14. Now the Spartan3 device allows to implement buses at a speed of 100 MHz with $R = (3{+}1){\cdot}4{\cdot}2 = 32$ resource slots, each being just two CLB-columns wide. The 32 CLB columns would span over the half chip in the case of a Spartan3 XC3S2000 device that provides the equivalent of 2M system gates.

4.2 Example System

Current synthesis tools are not able to directly use our techniques for building reconfigurable buses. As a consequence, we built a tool called ReCoBus-Builder that allows to parameterize and to generate a monolithic macro containing the complete logic and routing of a reconfigurable bus. Currently, the tool is supporting all Spartan 3, Virtex II(Pro), and some Virtex IV FPGAs from Xilinx.

In order to demonstrate our methodology, we built a system containing a stimuli pattern generator in the static part

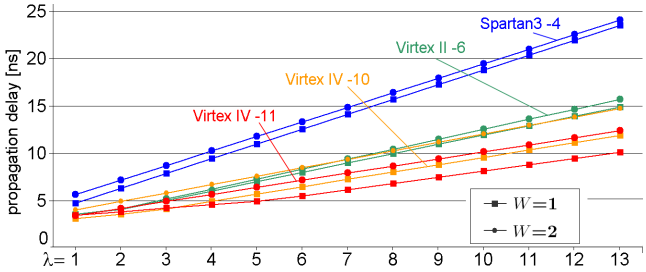

Figure 15: Extrapolated worst case propagation delays for buses with one pipeline register over λ.

of the system that is connected to a reconfigurable bus on a Xilinx Virtex II-6000-6 FPGA. After specifying the bus parameters with B=32 bit and R=8 resource slots that are not interleaved (N=1) and that are W=2 CLB-columns wide, our tool ReCoBus-Builder generated a macro containing the complete logic and routing of the bus. In addition to these parameters, the user can chose between simple slave or master mode and it is possible to select up to two address buses (in the slave mode) for simultaneous read and write operations. We also generated macros that are completely compatible to the Wishbone standard and that support to integrate given IP-cores without any interface modifications into a reconfigurable bus-based system. Note that these macros can be directly instantiated by the Xilinx tools. However, in these systems, we achieved only clock rates of up to 50 MHz that was limited by the IP-cores but not by the bus. As a consequence, we implemented a special stimuli generator for testing our buses at full speed. The stimuli generator contains a set of simple test modules that consist of a small amount of logic and an output register (adders, Boolean functions, and permutations). A chip view on this system is presented in Figure 16.

Based on this design, we used an incremental design flow to build the same set of test modules once again, but this time the modules are connected to the reconfigurable bus. All partial modules have been built one after another at the rightmost possible resource slot that is located at the farthest position from the stimuli generator as shown in Figure 17. This ensures that the timing can be guaranteed for all reconfigurable modules at any position at the bus. With each synthesis step of a partial module, we generate the partial reconfiguration data exactly for this particular module. Note that it is possible to assign a slack to all signals of the bus, thus, allowing to (re-)synthesize the static part of the system independent to the partial modules. This hold true, if the bus is constrained to a fixed location.

The stimuli generator can write a portion of data to its internal test module and simultaneously to the equivalent module connected to the bus, thus being capable to verify the data transfers. By designing the bus using two independent address buses for read as well as for write operations,

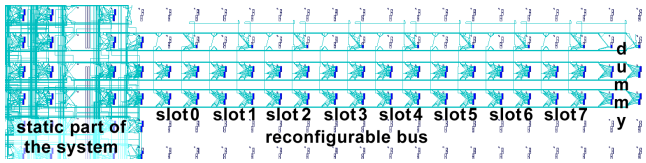

Figure 16: Unpipelined reconfigurable bus with eight resource slots and a dummy resource slot at the left hand side of the bus. Each resource slot is W=2 CLB columns wide and allows for connecting a slave with a data bus width of up to 32 bit.

we have doubled the number of bus transfers in order to gain the number of tests per time. The complete configuration process and the work of the stimuli generator is controlled by an external host. In this system, the bus is the limiting factor restricting the clock rate to 104 MHz when using the unpipelined transfer mode between the static part and any generated partial test modules. Therefore, the accumulated bus throughput on the 32 bit wide data channels with simultaneous read and write capability was 800 MB/s. More than 20.000 tests with changing configuration patterns of test modules connected to random bus positions have demonstrated that the configuration process can be performed reliable without any corruption to data transferred over the reconfigurable bus.

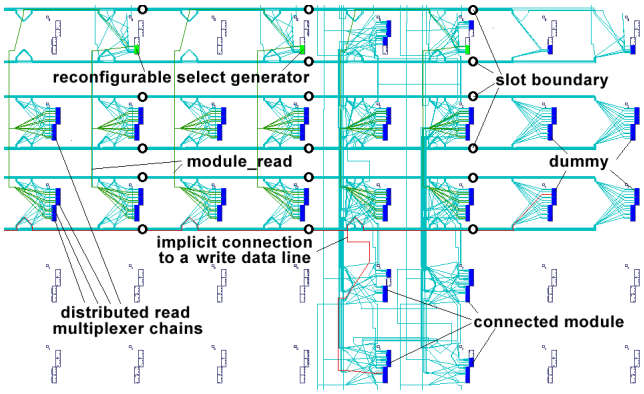

Figure 17: Enlarged view on the bus with a connected 32-bit wide slave test module.

5. CONCLUSIONS

In this paper, we presented a set of new techniques that enhance the flexibility of integrating hardware modules *efficiently* to a reconfigurable on-chip bus by dynamically partial reconfiguration. With these techniques, we are the first to offer *high placement flexibility* and *simple system integration* of reconfigurable slave modules as well as of master modules in combination with *high bus bandwidths* and *low resource overheads*. We achieved a reduction of logic resources for implementing the bus to almost one eighth as compared to the best existing approach. By implementing a test system, we demonstrated that our buses are suitable for high speed systems. Our reconfigurable buses permits to swap modules in an embedded system without any user intervention. Here, was before only known from racks where cards where plugged into backplane buses (e.g., VME-buses).

Current work targets on supporting more FPGA architectures including all Xilinx Virtex-V devices and on improving the ReCoBus-Builder tool.

6. REFERENCES

[1] A. Ahmadinia, C. Bobda, M. Majer, J. Teich, S. Fekete, and J. van der Veen. DyNoC: A Dynamic Infrastructure for Communication in Dynamically Reconfigurable Devices. In *Proc. of the Int. Conf. on Field-Programmable Logic and Applications*, pages 153–158, Tampere, Finland, Aug. 2005.

[2] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, J. Teich, S. Fekete, and J. van der Veen. A Practical Approach for Circuit Routing on Dynamic Reconfigurable Devices. In *Proc. of the 16th IEEE Int. Workshop on Rapid System Prototyping*, pages 84–90, Montreal, Canada, Jun 2005.

[3] Altera Inc. *Alera Devices*, jun 2007. `www.altera.com/products/devices/dev-index.jsp`.

[4] Shekhar Bapat and SridharKrishna Murthy. *High Speed Bidirectional Bus with Multiplexers*. WO Patent WO001998016012A1, issued 10 October 1996.

[5] L. Benini and G. D. Micheli. Networks on Chips: A New SoC Paradigm. *Computer*, 35(1):70–78, Jan. 2002.

[6] C. Bieser and K.-D. Mueller-Glaser. Rapid Prototyping Design Acceleration Using a Novel Merging Methodology for Partial Configuration Streams of Xilinx Virtex-II FPGAs. In *17th IEEE Int. Workshop on Rapid System Prototyping*, pages 193–199, Los Alamitos, CA, USA, 2006.

[7] H. A. ElGindy, A. K. Somani, H. Schroeder, H. Schmeck, and A. Spray. RMB - A Reconfigurable Multiple Bus Network. In *Proc. of the 2nd. Int. Symp. on High-Performance Computer Architecture*, pages 108–117, Feb. 1996.

[8] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann. Design of Homogeneous Communication Infrastructures for Partially Reconfigurable FPGAs. In *Proc. of the Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, USA, June 2007.

[9] J. Hagemeyer, B. Kettelhoit, and M. Porrmann. Dedicated Module Access in Dynamically Reconfigurable Systems. In *Proc. of the 20th Int. Parallel and Distributed Processing Symposium (IPDPS)*, Rhodes Island, Greece 2006.

[10] M. Hübner, C. Schuck, M. Kühnle, and J. Becker. New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-time Adaptive Microelectronic Circuits. In *Proc. of the IEEE Annual Symp. on Emerging VLSI Technologies and Architectures*, page 97, Washington, DC, USA, 2006.

[11] M. Huebner, T. Becker, and J. Becker. Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration. In *Proc. of the 17th Symp. on Integrated Circuits and Cystem Design*, pages 28–32, New York, NY, USA, 2004.

[12] H. Kalte, M. Porrmann, and U. Rückert. System-on-Programmable-Chip Approach Enabling Online Fine-Grained 1D-Placement. In Proc. *11th Reconfigurable Architectures Workshop*, page 141, New Mexico, USA, 2004.

[13] D. Koch, C. Beckhoff, and J. Teich. Bitstream Decompression for High Speed FPGA Configuration from Slow Memories. In *Proceedings of the Int. Conf. on Field-Programmable Technology*, 2007.

[14] D. Koch, M. Körber, and J. Teich. Searching RC5-Keys with Distributed Reconfigurable Computing. In *Proc. of Int. Conf. on Engineering of Reconfigurable Systems and Algorithms*, pages 42–48, Las Vegas, USA, June 2006.

[15] S. Koh and O. Diessel. COMMA: A Communications Methodology for Dynamic Module Reconfiguration in FPGAs. In *14th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM 2006)*, pages 273–274, Los Alamitos, CA, USA, 2006.

[16] Y. E. Krasteva, A. B. Jimeno, E. de la Torre, and T. Riesgo. Straight Method for Reallocation of Complex Cores by Dynamic Reconfiguration in Virtex II FPGAs. In *Proc. of the 16th IEEE Int. Workshop on Rapid System Prototyping*, pages 77–83, Washington, DC, USA, 2005.

[17] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford. Invited Paper: Enhanced Architecture, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs. In *Proc. of the 16th Int. Conf. on Field Programmable Logic and Application*, pages 1–6, Aug. 2006.

[18] J. C. Palma, A. V. de Mello, L. Möller, F. Moraes, and N. Calazans. Core Communication Interface for FPGAs. In *Proc. of the 15th Symp. on Integrated Circuits and Systems Design*, page 183, Washington, DC, USA, 2002.

[19] M. Ullmann, M. Hübner, and J. Becker. An FPGA Run-Time System for Dynamical On-Demand Reconfiguration. In *Proc. of the 18th Int. Parallel and Distributed Processing Symp.*, Santa Fe, New Mexico, Apr. 2004.

[20] H. Walder and M. Platzner. A Runtime Environment for Reconfigurable Operating Systems. In *Proc. of the 14th Int. Conf. on Field Programmable Logic and Application*, pages 831–835, 2004.

[21] Xilinx Inc. *Xilinx : Silicon Devices*, jun 2007. `www.xilinx.com/products/silicon_solutions/`.