Dedicated Module Access in Dynamically Reconfigurable Systems

J. Hagemeyer, B. Kettelhoit, and M. Porrmann
Heinz Nixdorf Institute, University of Paderborn
System and Circuit Technology
33102 Paderborn, Germany
{jenze, kettelhoit, porrmann}@hni.upb.de

Abstract

Modern FPGAs, such as the Xilinx Virtex-II Series, offer the feature of partial and dynamic reconfiguration, allowing to load various hardware configurations (i.e., HW modules) during run-time. To enable communication with these modules and for controlling purposes, dedicated access to each module as well as dedicated signals to control the global communication are required. This paper discusses several ways of implementing dedicated signals and addresses the impact on dynamically reconfigurable systems. Two new approaches are introduced, which allow a permanent access to the modules and to the communication infrastructure even during reconfiguration.

1. Introduction

Dynamic reconfiguration enhances the potential of FPGAs by enabling the change of hardware structures during run-time. This enables embedded systems based on reconfigurable hardware to adapt to changes of the environment. Until the rise of reconfigurable hardware this has been an exclusive property of microprocessors, which can easily change their behavior by processing different software. For prototypic implementations of dynamically reconfigurable architectures, Xilinx FPGAs are used in almost all cases since they offer the highest logic density and performance compared to other fine-grained, dynamically reconfigurable devices.

As described in detail in the following sections, reconfigurable systems are typically divided into static and dynamic hardware. While the static hardware consists of components which are needed at any time (e.g., interface hardware or memory), the dynamic hardware provides dynamically reconfigurable resources to which hardware modules can be loaded dynamically. A module consists of a set of logical resources and memory elements with usually rectangular shape. For the arrangement of several modules on one reconfigurable area different placement approaches are known. In this context, the overall resource usage, the amount of memory needed to store the configuration data, and the possibility to communicate with modules are of special interest. In this paper we focus on communication among modules or between static and dynamic components. This requires communication infrastructures like on-chip busses or networks-on-chips to be adapted to the special requirements of dynamically reconfigurable systems, such as the changing number of modules and the unknown module states during the reconfiguration process. All kinds of communication structures require dedicated control signals per subscriber, e.g., chip-enable or chip-select lines. We explain the problems that have to be solved to load many different modules onto one FPGA and common techniques to dynamically place modules in chapter 2. In chapter 3, we focus on different routing techniques for dedicated signals and their impact on module relocation, multiple module instances, and possible implementations. Besides known approaches we introduce and evaluate two new routing techniques which

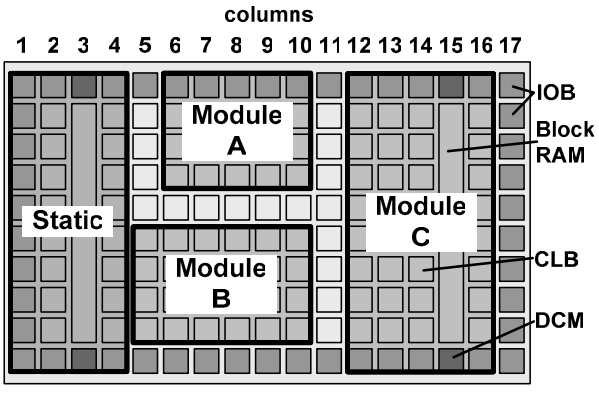

Figure 1. FPGA schematic with rectangular 2D module placement

1-4244-0054-6/06/$20.00

allow permanent control of global communication infrastructures. We conclude this paper with a summary and future work in chapter 4.

2. Run-time reconfiguration

Xilinx FPGAs are composed of an array of Configurable Logic Blocks (CLBs) surrounded by Input/Output Blocks (IOBs), which are connected to the FPGA's pins (see Figure 1). Most common FPGAs contain different types of processing elements, some for realization of general logic, some more specialized blocks for functions often needed in applications, like memory and multipliers, or functions that can only be implemented in dedicated hardware, like clock generation or clock distribution circuitry. The term "run-time reconfiguration" in general means to change the function of some of these processing elements while leaving the rest in functional operation. Configuring a contiguous set of processing elements during run-time is also called "loading a module". In principle, all modules could be of arbitrary shape. Nevertheless, all implementations known to the authors only consider modules with rectangular shapes. On the one hand this is caused by the limited support for dynamic reconfiguration of currently available design tools, which only support rectangular shapes. On the other hand, it is much easier to solve the task of finding available and suitable resources on a two dimensional CLB array for rectangular modules rather than for arbitrarily shaped modules.

2.1. Run-time reconfiguration on Xilinx FPGAs

In addition to Configurable Logic Blocks, Xilinx FPGAs provide, e.g., dedicated memory blocks (BlockRAM) and Digital Clock Managers (DCMs). These additional elements disrupt the homogeneity of the reconfigurable resources, which does not affect the communication infrastructures discussed in the following.

Although modules always consist of a contiguous set of CLBs, the configuration data (i.e. the partial bitstream) cannot necessarily be loaded to the FPGA continuously. Xilinx Virtex FPGAs can only be (re-)configured in a column-wise manner ([1], [2]). This means that reconfiguring only a few CLBs (or IOBs, etc.) implicates a reloading of the configuration data of all other configurable elements in the affected columns. An example to this problem can be seen in Figure 1: By loading module A to the FPGA, the columns 6 to 10 have to be reconfigured. This implicates reloading the configuration data of the columns 5 to 10 of module B. This means, loading this module will also affect the columns 6 to 10 of module B, which currently uses columns 5 to 10. Thus, a partial bitstream has to be composed from the configuration data of two modules sharing identical columns during run-time.

In general, partial bitstreams consist of command and data sequences [2]. The data sequences define the functionality of the reconfigurable resources while the command sequences define *which* resources will be reconfigured. Speaking of modules this means that the command sequences of an according bitstream defines the position on the FPGA to which a module will be loaded. This means in turn that loading one module to two different positions requires two different bitstreams.

2.2. Communication macros

Like static FPGA designs, dynamic modules can freely use the available resources within a given area. Exceptions are the routing resources used for signals, which are crossing the module's borders. These signals, e.g., bus signals used for communication, have to be continued within neighbouring modules or static components. The position on a module's border where a signal leaves or enters the module can be considered a module pin. To allow the exchange of modules during run-time, all pins used for communication or for dedicated signals have to be at the same positions for all modules. In the Xilinx design flow this can be realized by means of macros that explicitly define the routing resources to be used for a given signal. Only if the signals that are leaving or entering a module are realized with macros, and if the same macros are used for the implementation of all modules, a communication across module borders can be guaranteed among any neighbouring modules. Xilinx introduced the use of macros for bus communication signals in [3].

2.3. Placement approaches

As mentioned before, all practically used placement approaches are based on rectangular module shapes. These approaches can be divided into one-dimensional and two-dimensional placement, both of which can be realized with fixed module slots or with variable positions and module sizes (called fixed 1D/2D and free 1D/2D, respectively). The free 2D approach is shown in Figure 1. A module can be placed everywhere, as long as it provides a rectangular shape. This leads to some non-trivial problems. First, this approach requires a more sophisticated placement algorithm, compared to all other approaches. Furthermore, the communication infrastructure has to be adapted to the modules during each reconfiguration, which is not an easy task and will most likely disturb the whole communication structure during reconfiguration. As a third issue, the partial bitstream for this dynamic reconfiguration has to be composed at run-time (see section 2.1) since it cannot be

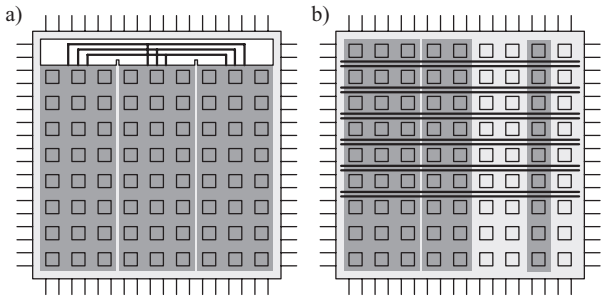

Figure 2: Two 1D placement approaches: a) 1D placement with fixed slots, b) free 1D placement

known in advance which modules will be involved into any occurring reconfiguration. For more details about run-time composition of bitstreams see, e.g., [4]. To prevent rebuilding the communication infrastructure during reconfiguration, either a fixed floorplan is needed [4], or a pseudo 2D approach can be chosen. In such an approach, the reconfigurable area is divided into fixed rectangular areas, e.g. the reconfigurable area will be split up into slots at the top and at the bottom, as shown in [4]. In this case, 1D communication structures can be used.

The bitstream composition at run-time can be avoided if modules always use the full height of the FPGA, leading to the 1D placement approaches, which is well suited for column-wise reconfiguration. In addition, the task of finding available resources for modules during run-time is reduced to a one dimensional problem and thus is heavily reduced in complexity. Figure 2 shows both 1D placement approaches with variable module width and fixed module widths, respectively. When using fixed module widths, each module that can possibly be loaded to the reconfigurable area has the same shape, leading to a rather simple managing task. An example communication structure for this *1D approach with fixed slots* is shown in Figure 2a. The disadvantage of this approach is that it fixes the amount of resources available and reserved per module. While this wastes resources for smaller modules, big modules cannot be implemented if they exceed the size of the slots. As an alternative, 1D placement with variable module width is a more general approach. In this case, a one dimensional module can be placed anywhere within a predefined grid in the reconfigurable area. In general, this grid is not identical to the FPGA columns. A module can have any width (with a minimum of one CLB column) and can be placed at any location provided by the grid. Current implementations of dynamically reconfigurable systems with 1D placement usually use grid widths of two or four CLB columns. A communication infrastructure for a grid based variable placement approach is shown in Figure 2b. Any possible module position contains communication connection points, typically realized as tristate buffers ([5]), or as a slice based communication ([6]). All current implementations known to the authors use one of these 1D communication infrastructures. Thus, we will focus on 1D communication infrastructures in this paper.

2.4. Partial modules – module relocation and multiple module instantiations

As mentioned before, loading a module to two different locations generally requires two different partial bitstreams. One possibility to deal with this fact is to generate separate bitstreams for all possible module locations and then downloading the bitstream that corresponds to the chosen position during run-time. For 1D-approaches with only few fixed slots this can be an easy to realize possibility. Still, the amount of bitstreams that has to be generated and stored for each module quickly grows with the number of slots, resulting in rising costs for and waste of memory. For free 1D placement (without constraints to the position), this multiple locations – multiple bitstreams approach is not realizable with reasonable effort even for small systems. A solution to this is run-time bitstream manipulation, as described in [7]. This technique uses the homogeneity of the FPGA's CLB area and relocates modules by changing the address information in the command sequences of a partial bitstream while keeping the configuration data for all configurable elements (CLBs, IOBs, etc.). With module relocation all modules can be placed to any position on the FPGA (for 1D-placement approaches) while only one partial bitstream per module has to be generated and stored (multiple locations – single bitstream). Although module relocation allows to realize 1D-placement in big reconfigurable systems (many modules, many locations), its disadvantage is that the reconfigurable area, which means every slot, has to be completely homogeneous. In case of an inhomogeneous FPGA, the content of the data segments in the bitstream need to be adapted in order to reinstantiate or relocate a module, which is practically impossible at runtime. As a solution to this problem, a set of bitstreams can be generated, which, in combination with module relocation, cover all possible positions of an inhomogeneous FPGA.

In principle, it is possible to load multiple instances of one module to the FPGA. E.g., when using FPGAs for digital control it could be sensible to use identical controllers to control different plants. In this case, however, one has to take care to use separate dedicated signals for each instance of one module. Otherwise, a reset, for example, would apply to all instances of one module, which is not desired in general.

3. Dedicated signals

In nearly every reconfigurable system, some connections are dedicated to each module, which means that these signals are unique to a particular slot or position on the FPGA. These unique signals can easily cause an inhomogeneity in the reconfigurable area, preventing the use of principles like module relocation or multiple module implementations as described in the previous chapter. In this chapter, several ways to deal with these dedicated signals are presented.

3.1. Classification of dedicated signals

Dedicated signals generally can be divided into two classes: dedicated signals for the system's communication infrastructure and dedicated signals for each single module. Members of the first class must never be interrupted, not even during reconfiguration. Typical examples for this type of dedicated signals are *enable* signals that have to be driven by the modules in order to drive the data lines of a shared bus. For tristate based communication, as described in [5], erroneously driving the enable signal during reconfiguration will eventually cause electrical damage to the device. Erroneously setting the select signal of a slice based communication structure, will still cause corrupted data. Hence, such signals have to be at a defined level at any time. To achieve this, dedicated signals of this class are typically included in bus macros. If all modules use the same bus macro during synthesis, a signal implemented with this macro will never be interrupted during reconfiguration. For the second class of dedicated signals, the status during reconfiguration doesn't matter, since these signals only affect the loaded module itself. A typical signal of this class is a module dedicated reset signal that resets all module internal registers to a reasonable state before putting the module into operation. Such a signal is only needed after configuration and thus is not affected by any configuration issues.

Another classification of dedicated signals can be made depending on the requirements to signal latency. As will be shown in the following sections, different implementation techniques result in different latencies for dedicated signals. For some dedicated signals, such as bus grant or request signals, a latency of many clock cycles cannot be accepted. Other signals, such as a global disable for tristate drivers during reconfiguration, have much lower demands to signal latency.

3.2. Position-based dedicated signals

Figure 3 shows a straightforward approach for dedicated signals. As an example, the enable lines for the tristate buffer of a shared bus are chosen. To prevent modules from unintentionally driving the bus during module reconfiguration, in each slot the enable signal for all tristate drivers is generated by a logical AND-operation of an internal enable and a global enable from a static supervising component. The logical AND (realized as a LUT) and the global enable are implemented as macros. This allows the bus arbiter to reliably disable the tristate drivers even during partial reconfiguration, since the information for the macro is included in all modules that will ever be loaded to a specific slot. After a module has been loaded successfully, the arbiter activates the global enable of the modules position and thus passes the control over the tristate drivers to the module. Modules with a width bigger than one principally have more than one possibility to access the bus (see module A in Figure 3). Such modules have to choose one access point and leave all other possibilities unused. The dedicated signals for each bus access have to be routed at different vertical positions within the FPGA. Hence, the bus macros differ slightly for each grid position. Since the macros are part of the module's partial bitstreams, this means in turn that module relocation, as described above, is not possible.

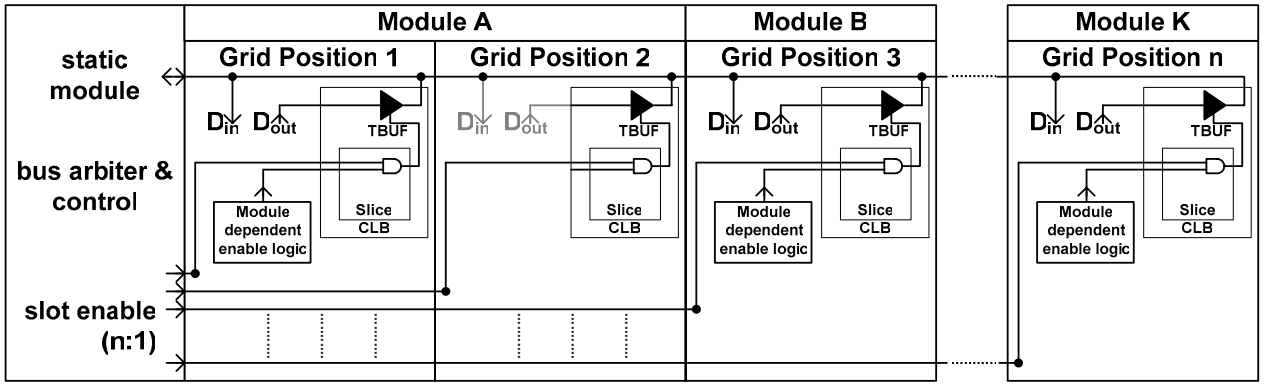

Figure 3: Position-based dedicated enable signals

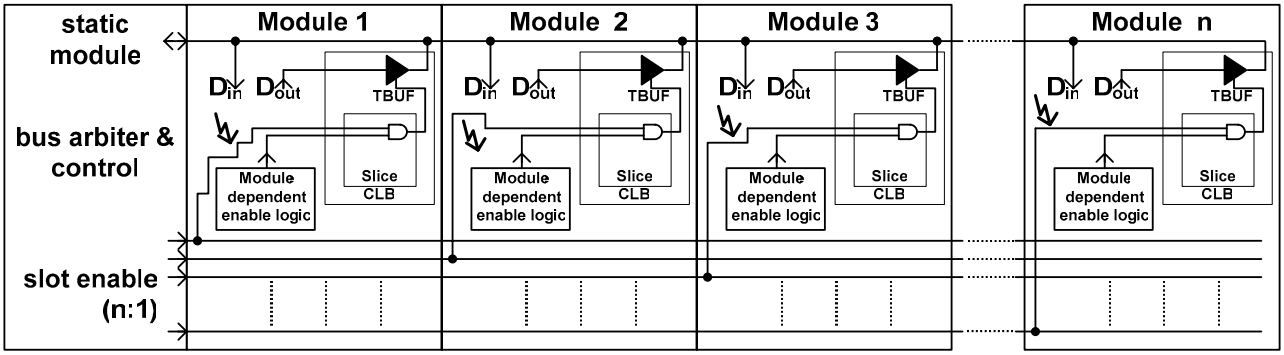

Figure 4: Module-based dedicated signals

Thus, choosing a module's position during run-time requires a separate bitstream for each module at any possible position. In general, this method of routing dedicated signals is applicable to both free 1D placement and 1D placement with fixed slots.

3.3. Module-based dedicated signals

Instead of using dedicated signals for every position, the module-based dedicated signal approach uses one dedicated signal per module (see Figure 4). As for the position-based approach, all dedicated signals are routed horizontally at different vertical positions. If these vertical routing resources are spanned across the whole FPGA, any module can connect to its dedicated signals at any position on the FPGA. The vertical connection from the horizontal routing resources to, e.g., a tristate enable port, is different for each module and hence no macro can be used for this connection. This means in turn that during reconfiguration, an external control over the bus enables is not possible and, even worse, an erroneous driving of the bus cannot be prevented. This of course means that module-based dedicated signals should not be used for signals which need a permanent connection during configuration, such as the tristate enables used in the example. For other signals, though, this can be an easy to realize alternative, which needs only very few resources.

Loading multiple instances of one module also is not possible with this approach, since all instances would be controlled via the same dedicated signal. However, since the connection to the dedicated signals is not dependent on the position to which a module will be loaded, module relocation can easily be used.

In this approach, the number of modules has to be known at design time of the system in order to provide sufficient signals for all modules. Furthermore, the associations between the dedicated signals and the modules have to be made in advance to be able to synthesize the modules. Hence, it is not possible to load modules to the system, which were not known at design time. A separate development of static hardware and software (modules) thus is not possible.

3.4. Slot dedicated signals

As one alternative to the straight forward approaches presented before, Figure 5 shows the same example implemented as a system using slot-based dedicated enable signals. Next to each slot, an adaptor CLB column redirects the enable signal from its vertical routing resources to another vertical position. From this position, the signal is then handed over vertically to the slot and thus to a module. Within the module, the global enable

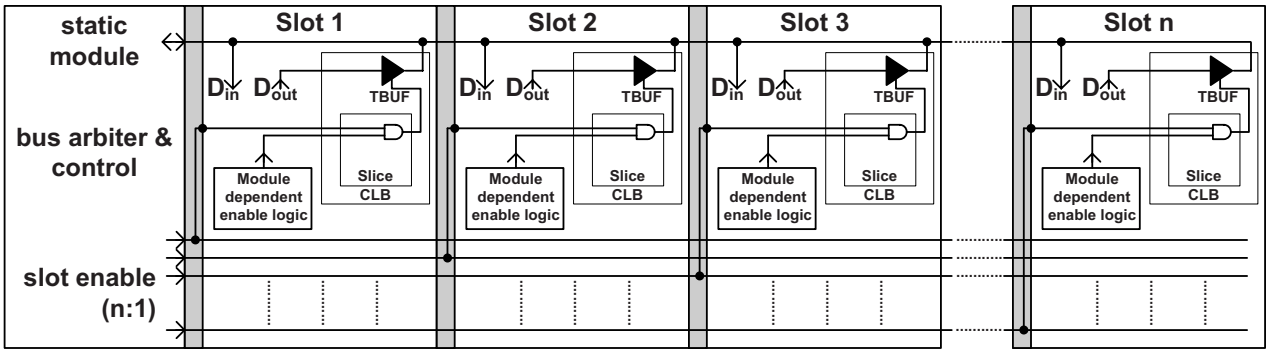

Figure 5: Slot-dedicated enable signals with adaptor

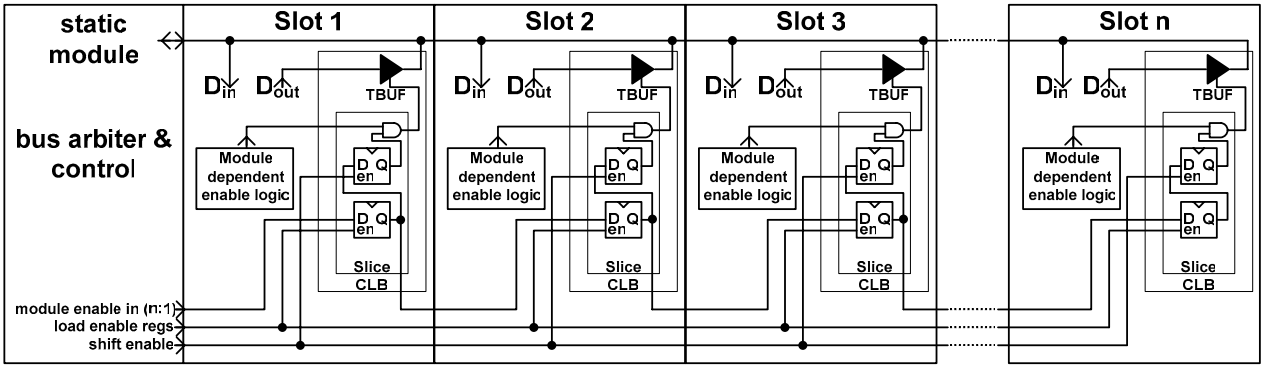

Figure 6: Enabling/Disabling of bus accesses via shift registers

signal is routed to a logical AND, which in turn controls the tristate enable. As in 3.2, these signals have to be implemented with macros to prevent the tristate drivers from accidentally driving the bus.

Since each slot provides the access to the enable signal at the same position, all modules can use the same hard macro, no matter to which slot they are loaded, i.e., module relocation can be used. The association of a dedicated signal and a module has to be made during run-time, allowing for a separate development of static hardware and modules. It is possible to use different sets of modules in the same static environment, i.e., the static hardware can be reused for different tasks. Loading multiple instances of one module into different slots at the same time is also possible. The main disadvantage of this approach is a certain waste of resources. At least one CLB column per slot is needed additionally and can not be used for module dedicated logic, but only very few routing resources are actually used in these columns. So this approach is better suited for implementations with a big number of columns per slot. An interesting issue is the possibility of implementing multiplexed busses in a reconfigurable system using slot-dedicated signals. In future FPGAs, possibly supplying much more logic and routing resources, this approach could be able to use the benefits of multiplexed buses to reconfigurable systems without relocation or multiple instantiation trouble.

3.5. Shift register based module access

The approaches introduced in the last chapters used direct connections from a supervising component to the modules. These connections have to use routing resources on different vertical positions and thus either cause inhomogeneity (through the vertical routing) or do not support permanent connectivity. This leads either to additional effort to compensate this inhomogeneity (as for the slot-dedicated enable signals) or does not allow multiple module instantiations or module relocation. Figure 6 shows a fourth approach, again using the tristate enable example, which has no such inhomogeneity and thus does not share the mentioned disadvantages. The main idea is that the enable signals are not connected directly to the modules, but are stored locally in registers within the modules (the registers are part of the communication hard macro). These registers are cascaded to a *shift register*, which can be filled by the arbiter. The number of registers corresponds to the number of slots or possible module positions, respectively. If any of the slots shall be enabled or disabled, a new corresponding content is written to the shift register. To avoid glitches on the enable signals during a write process to the shift register, a second stage of *enable registers* is inserted. After writing to the shift register, its content is copied to the enable registers and thus the tristate enables are enabled or disabled, respectively. Modules, whose enable state is not changed, are not affected by the whole process at all. As an example, let us imagine that slot 1 and 3 are actually used on the FPGA and communicating via the tristate bus. Slot 2 now gets used by loading a module to its place. Before the reconfiguration, both contents of the shift register (SR) and the enable registers (ER) are SR = ER = (1, 0, 1, 0, ...). After loading a module to slot 2, its global enable is activated by writing (1, 1, 1, 0, ...) to SR. In a second step this vector is loaded to ER and thus module two gains access to the bus (ER = 1, 1, 1, 0, ...). To ensure that the enable register of any newly loaded module is initially set to 0 (i.e. to prevent the tristate drivers from driving the bus during reconfiguration), the enable registers have to be part of the hard-macro used for the tristate drivers and the logical AND. This enables writing a new content to the shift register even during reconfiguration. The activation delay of n clock cycles needed to load the shift register plus one cycle to load the enable registers thus can be reduced to one clock cycle. Still, it is this rather high delay, which is the biggest disadvantage of the shift register based dedicated signals. For signals like bus grant signals, such a delay is probably not acceptable.

Table 1: Comparison of the three approaches

	placement approach	permanent connection	module relocation	multiple instantiations	area costs	signal latency
position-based	free 1D, fixed slots	✓	✗	✗	-	1
module-based	free 1D, fixed slots	✗	✓	✗	-	1
slot-based	fixed slots	✓	✓	✓	one column per slot	1
shift register	free 1D, fixed slots	✓	✓	✓	two registers per slot	N + 1 (1)

N: number of modules loaded

The shift register based approach supports both module relocation as well as multiple module instantiations. Although it was presented as implementation possibility for 1D placement with fixed slots, it can also be used for free 1D placement without any remarkable waste of resources. In this case, though, every possible vertical position within the grid needs a separate access point (comparable to the position-based dedicated signals), quickly leading to big registers and big delays. For the chosen example, the global tristate enable, this might still be a good solution, since even big shift registers (> 100 stages) can be loaded within microseconds, whereas the configuration time for a module is in the order of a millisecond.

3.6. Summary

In table 1 the different routing techniques for dedicated signals are compared. Position-based dedicated signals provide a direct connection, which can be controlled even during reconfiguration. Since this technique does not resolve the inhomogeneity caused by dedicated signals, module relocation is not possible. Module-based dedicated signals forgo the use of bus macros for vertical routing. This makes the problem of inhomogeneity obsolete and thus allows module relocation. The prize to pay is that the dedicated signals are not connected during reconfiguration and that multiple instances of one module cannot be controlled independently. Still, both position-based and module-based dedicated signals require simple bus macros only and can easily be implemented.

The slot-based approach moves the inhomogeneity from the modules (and the partial bitstreams) to adaptor columns, which are not reconfigured during run-time. This allows the use of module relocation. Since the dedicated signals are not dedicated per module but per slot, multiple instances of one module can be loaded to the FPGA. The unique bus macro, which is used to implement all modules, enables a direct connection, also during configuration. The disadvantage of this slot-based approach is that a whole CLB column per slot has to be used for rerouting and cannot be used for other purposes. This is only acceptable for wide slots, making this approach inapplicable for free 1D placement.

As a last alternative we have discussed a shift register based variant. By locally storing the signals in each module, all signals can be transmitted via the same routing resources, preventing the occurrence of inhomogeneity. The complete routing of the dedicated signals can be made via one unique bus macro, allowing module relocation, multiple module instances, and permanent connection. The two registers needed per slot can be implemented in one slice only, so that this approach can be used for both free 1D and fixed slots placement. Still, the registers cause an additional signal delay per slot. This indirect control cannot be used for a certain group of dedicated signals.

To solve the tristate enable problem, Blodget et al. suggest in [4] to configure a module in two steps. In a first step, the module with all internal logic and tristate drivers is loaded onto the FPGA. In a second, additional configuration step only the connections between the tristate drivers and the tristate lines are activated. This makes dedicated signals for controlling the tristate drivers obsolete. Still, with this solution two bitstreams and two configuration steps are required to configure only one module.

A very important issue for all approaches, which has not been addressed yet, is the development effort needed to implement the macros. In most current applications, one macro contains a small number (e.g four or eight)

signal lines per row. These macros are then instantiated multiple times to realize a complete bus structure. In case of the global enable signal, which is used as an example above, where one signal controls 32 or 64 tristate drivers, this “copy and paste” design flow for the bus macros is not possible any more. Instead, one big bus macro has to be generated. Doing this manually is a tough, error-prone job. EDA tools for an automated macro design are still not available, but would be a great help for the implementation of dynamically reconfigurable systems.

4. Conclusion and further work

In this paper we have discussed different methods of routing dedicated signals to dynamic modules in reconfigurable systems. We have analyzed the requirements of currently used placement approaches such as support for module relocation, multiple module instantiations, and connectivity during reconfiguration. With regard to these requirements we have shown different ways of routing dedicated signals to dynamic modules and we have classified them by means of area costs and signal latency. For the use of dynamically reconfigurable hardware and for the evaluation of different placement approaches, communication via module specific dedicated signals is a basic requirement, which has been ignored in many studies in the past.

We are currently developing a tool, which is able to automatically generate various kinds of macros and which supports all of the placement approaches and routing for dedicated signals shown in this paper. This framework will significantly reduce the time needed for the implementation and evaluation of different placement approaches for FPGA-based dynamically reconfigurable systems.

5. References

[1] Xilinx. Virtex Series Configuration Architecture User Guide, *Xilinx Application Note 151.* Xilinx, March 2003.

[2] Xilinx. *Xilinx Virtex-II Platform FPGA User Guide.* March 2005.

[3] D. Lim and M. Peattie. Two flows for partial reconfiguration: module based or small bit manipulation, *Xilinx Application Note 290.* Xilinx, 2002.

[4] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght and T. Becker. Modular partial reconfiguration in Virtex FPGAs. In *Proceedings of the 15th International Conference on Field Programmable Logic and Applications*, Tampere, Finland, August 2005.

[5] H. Kalte, M. Porrmann and U. Rückert. System-on-Programmable-Chip Approach Enabling Online Fine-Grained 1D-Placement. In *Proceedings of the 11th Reconfigurable Architectures Workshop*, 2004.

[6] J. Becker, T. Becker and M. Hübner. Real-Time LUT-based Network Topologies for Dynamic and Partial Self-Reconfiguration. In *Proceedings of the 12th International Conference on Very Large Scale Integration VLSI-SoC*, Darmstadt, December 2003.

[7] H. Kalte, L. Gareth, M. Porrmann and U. Rückert. REPLICA: A bitstream manipulator filter for module relocation in partial reconfigurable systems. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium, Reconfigurable Architectures Workshop*, 2005.