Fast Start-up for Spartan-6 FPGAs using Dynamic Partial Reconfiguration

J. Meyer*, J. Noguera†, M. Hübner*, L. Braun*, O. Sander*, R. Mateos Gil‡, R. Stewart† and J. Becker*
*Institute for Information Processing Technology
Karlsruhe Institute of Technology, Karlsruhe, Germany
Email: {Joachim.Meyer, Michael.Huebner, Lars.Braun, Oliver.Sander, Becker}@KIT.edu
†Xilinx Inc., Ireland
Email: {Juanjo.Noguera, Rodney.Stewart}@Xilinx.com
‡Departamento de Electrónica, Universidad de Alcalá
Madrid, Spain
Email: Raul@depeca.uah.es

***Abstract*—This paper introduces the first available tool flow for Dynamic Partial Reconfiguration on the Spartan-6 family. In addition, the paper proposes a new configuration method called *Fast Start-up* targeting modern FPGA architectures, where the FPGA is configured in two-steps, instead of using a single (monolithic) full device configuration. In this novel approach, only the timing-critical modules are loaded at power-up using the first high-priority bitstream, while the non-timing critical modules are loaded afterwards. This two-step or prioritized FPGA start-up is used in order to meet the extremely tight start-up timing specifications found in many modern applications, like PCI-express or automotive applications. Finally, the developed tool flow and methods for Fast Start-up have been used and tested to implement a CAN-based automotive ECU on a Spartan-6 evaluation board (i.e., SP605). By using this novel approach, it was possible to decrease the initial bitstream size and hence, achieve a configuration time speed-up of up to 4.5x, when compared to a standard configuration solution.**

I. Introduction

In many of modern applications, electronic embedded systems have to meet extremely tight timing specifications. One of these timing requirements is the *start-up time*, i.e., time the electronic system has to be operative after power-up. Examples of electronic systems with such a start-up timing specification are PCI-express systems or CAN-based Electronic Control Units (ECU) in automotive applications. In both of these examples, the electronic system has to be up and running within 100ms after system power-up. Otherwise, in the case of PCI-express, the system will not be recognized by the root complex [1], or, the system might miss important communication messages in the case of CAN-based automotive ECU's.

The technology trend in semiconductor industry, as predicted by Moore's Law, has enabled today's FPGA manufacturers to significantly increase the amount of resources in their devices. But with an increasing amount of resources, the bitstream size grows proportionally, so does the time to configure the device. Therefore, even with medium-sized FPGAs, it is not possible to meet the start-up timing specification using low-cost configuration solutions. Figure 1 shows the configuration time for different Spartan-6 FPGAs using the low-cost SPI/Quad-SPI configuration interface. Even when using a fast configuration solution (i.e., Quad-SPI running at 40MHz configuration clock) only the small FPGAs meet the 100ms start-up timing specification.

978-3-9810801-7-9/DATE11/©2011 EDAA

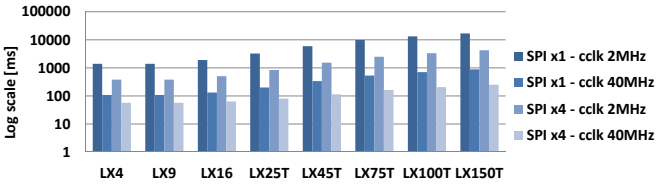

Fig. 1. Logarithmic illustration of calculated Spartan-6 configuration times

This paper tackles this problem of increasing configuration time in modern FPGAs. The paper explains a new configuration method called *Fast Start-up*, where the FPGA is configured in two-steps, instead of using a single (monolithic) full device configuration. In this novel approach, only the timing-critical modules are loaded at power-up using the first high-priority bitstream, while the non-timing critical modules are loaded afterwards. This approach minimizes the initial configuration data, and thus minimizes the FPGA start-up time for the timing-critical design.

Therefore, this paper features the following key novel contributions:

- It introduces the first available tool flow for dynamic partial reconfiguration on Spartan-6 FPGAs. Currently, available design tool-flow only supports Virtex families.
- The paper describes a complete new method to create partial initial bitstreams for Fast Start-up on FPGAs with *2-dimensional* configuration memory architectures (e.g., Spartan-6, Virtex-6). The previous existing method only works for FPGA families with *1-dimensional* configuration memory architectures (i.e., Spartan-3E, Virtex-II).
- The proposed design tool flow and techniques have been applied to a CAN-based automotive application. The design flow and techniques have been verified/tested in

hardware using a SP605 Spartan-6 development board.

The paper is organized as follows: Section II gives an introduction into the existing tool-flows for dynamic and partial reconfiguration of Xilinx FPGAs, followed by presenting existing techniques to reduce the configuration time of FPGAs. In section III after introducing Fast Start-up, a tool flow to build the necessary configuration bitstreams for Fast Startup using Spartan-6 FPGAs is presented. Section IV describes the implementation of an example use case for Fast Start-up and presents measurements of the configuration time for a Spartan-6 FPGA when using different configuration techniques. The paper is closed in section V by the conclusions.

II. Related Work

A. Dynamic Partial Reconfiguration for Xilinx FPGAs

Dynamic Partial Reconfiguration describes the technique to change the configuration data for a specific part of a reconfigurable device, while the other parts stay operative. The high-end FPGA family from Xilinx, Virtex, supports Dynamic Partial Reconfiguration for quite a long time now. There also have been successful implementations of Dynamic Partial Reconfiguration for Spartan-3 FPGAs (see [2]), however the low-cost Spartan family never officially supported Dynamic Partial Reconfiguration.

The oldest methods from Xilinx to build partial bitstreams are realized by two options of BitGen, the low-level tool of Xilinx to produce bitstreams. The first of those options is called Partial Mask. It allows determining which configuration columns will be included in the bitstream and which will be rejected. This feature enables a designer to cut out modules of a full design, if the exact location of this module is known. The option is well documented in the Virtex-2 Pro User Guide [3], but since the introduction of Virtex-4, this option is not available anymore. The other Partial Reconfiguration related BitGen option is about Difference-based Partial Reconfiguration [4]. It was created in order to be able to capture small design changes. Therefore, the typical flow is to generate small changes to a design by hand using the FPGA Editor, followed by using the BitGen program to produce a bitstream that only includes the differences between the original and the new design. This allows switching the configuration of a module from one implementation to another.

The first full tool flow for Partial Reconfiguration of Xilinx, the Early Access Partial Reconfiguration (EAPR) flow, introduced some new features like providing a graphical user interface using the PlanAhead software [5]. The flow is module based and available as a patch for the ISE design tools but was dropped with ISE 10.1.

With the release of ISE version 12.1, Xilinx introduced a new flow for Partial Reconfiguration. This flow is based around partitions and it provides several improvements over the old flow like timing analyses for nets which cross PR-borders. Again PlanAhead supports this flow and provides a graphical user interface. The new flow supports Virtex-4, Virtex-5 and Virtex-6 devices, but it does not support the Spartan-6 family. For more information see [6].

B. Reducing FPGA configuration

The problem of increasing configuration times has been tackled by several research groups (see [7] and [8]), whereby the usual approach was to decrease the amount of data which has to be transferred to the FPGA by compression. Therefore different kind of algorithms have been analyzed and compared in order to find a good compromise of compression rate and resource requirements for the decompression module which has to be inside the FPGA. However, since these approaches would need dedicated data-decompression logic inside the FPGA these methods cannot be used for the initial configuration without changes in the FPGA fabric.

III. Fast Start-up for Spartan-6

A. Fast Start-up

Fast Start-up is a two-step configuration technique which enables an FPGA design to start critical design parts as fast as possible, much faster than they can be made available using a standard full configuration technique.

Although Fast Start-up is using Dynamic Partial Reconfiguration, there are differences compared to the traditional concepts of this technique. While the concept of Dynamic Partial Reconfiguration intends a full design to be used as initial configuration which can be modified during runtime, Fast Start-up already uses a partial bitstream in order to only configure a specific part of the FPGA for start-up. In this first configuration, only those parts of the full FPGA design are contained, which have a high priority to be up and running quickly. The not yet configured parts of the FPGA can be accessed later during runtime by using Dynamic Partial Reconfiguration. The different concepts are illustrated in Figure 2.

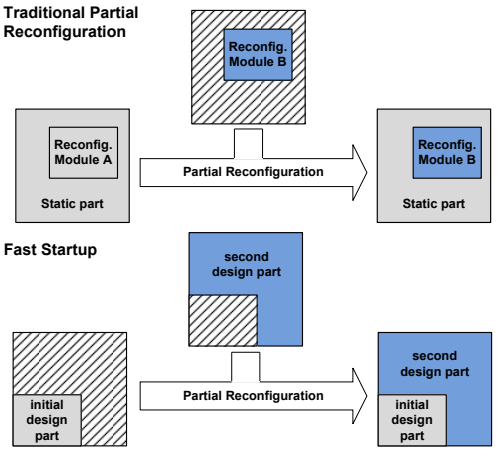

Fig. 2. Comparison of Traditional Partial Reconfiguration and Fast Start-up

The concept of Fast Start-up was introduced in [9] focusing on Spartan-3E FPGAs. Since the implementation techniques used in [9] are not supported by the newest FPGAs anymore, the following sections describe a new way to perform Fast Start-up on those devices by focusing on Spartan-6.

B. Fast Start-up tool flow overview

In order to implement the two step configuration of the Fast Start-up technique, the first step is to partition the complete FPGA design into two parts, one initial part and one for the second configuration. For both of those parts a partial configuration bitstream has to be built, but while the second bitstream would be a standard bitstream for Partial Reconfiguration, the first one needs to meet some special requirements, like including the configuration of the global clock resources.

Since there is no support for Spartan-6 by any of the available Partial Reconfiguration tool flows, both creation processes afford a non-standard procedure. The basic concept of this flow can be seen in figure 3. In order to get a partial initial bitfile which is holding the initial design configuration, first a full bitstream of the initial design is created (A). This full bitstream (A) is edited on a binary level to remove the configuration data which is not required what gives you the partial initial bitstream (C).

In order to get the partial bitstream for the Dynamic Partial Reconfiguration of the second design, it is possible to use the BitGen option "-r" for Difference-based Partial Reconfiguration which is still available for Spartan-6. Applied on a full design (B), using the full bitstream of the initial design (A) as reference, this option produces a partial bitstream (D) containing only the configuration data of the second design.

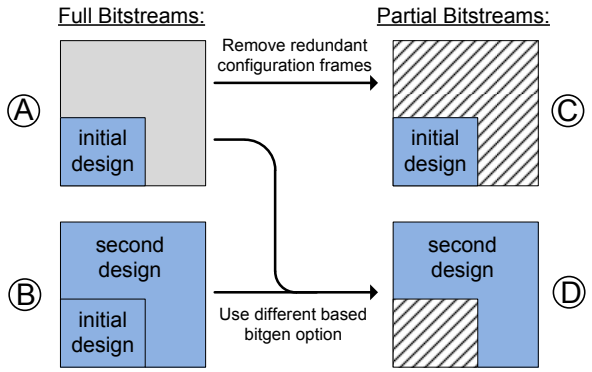

Fig. 3. Basic approach to create the partial bitstreams for Fast Start-up

C. Generation of the initial partial bitstream

As mentioned before, in order to get the initial partial bitstream (C) all redundant configuration data of a full bitstream has to be removed. This affords a deep knowledge of the configuration memory structure and the bitstream composition. The following low level information about bitstream composition and configuration procedure is based on the configuration user guides like [10] or [11].

The configuration of a Xilinx FPGA is organized in several configuration rows each consisting of multiple columns of resource elements like e.g. the Configuration Logic Blocks (CLBs). Such a configuration column can be broken down into several configuration frames which are the smallest addressable segments of the configuration memory space and therefore an operation always affects a whole frame. A configuration frame can be thought of as a one bit wide column which spans a whole configuration column. Thus one frame holds only little configuration data of one specific resource element but therefore it holds this information for all the resources in the corresponding configuration column.

In order to reduce the configuration bitstream size, the compress option of the Xilinx BitGen tool can be used. This option avoids writing similar frames multiple times into the FPGA. Instead, it writes this frame one time into the FDRI and afterwards the combination of updating the Frame Address Register (FAR) with the first of the corresponding addresses for the frame and triggering a Multiple Frame Write follows. A Multiple Frame Write (MFW) is a special configuration command which uses the actual frame inside the FDRI to configure the configuration memory addressed by the actual value of the FAR. After some No-operation commands, the procedure of updating the address and triggering an MFW gets repeated until all addresses for the frame are written.

Because of that it is possible to replace multiple similar frames of an ordinary bitstream, which for example for Spartan-6 usually contain 65 configuration words, with a sequence of 4-5 configuration words. The efficiency of the compress option therefore obviously depends on the amount of similar frames in a design. For Xilinx FPGAs the configuration data for resources which are not used in a design are only zeros. Thus an FPGA design which only uses a small amount of logic of the FPGA contains a lot of frames only consisting of zeros and therefore using compress with such a design will decrease the configuration bitstream size significantly.

However, all the memory addresses, the Multiple Frame Write commands and the No-operation words are still inside the bitstream. But for Zero-frames this is redundant information, because after the house cleaning process, all configuration memory should be initialized with zero anyway. While for an ordinary configuration bitstream removing the entire configuration data of resources which are not used and adding the necessary address updates by hand is very hard, this is much easier for a compressed bitstream. This is because the compressed bitstream structure already separates the Zero-frames by putting them into Multiple Frame Writes. Therefore the Zero-frames can be removed easily from the bitstream by removing all Multiple Frame Writes of Zero-frames. A comparable approach was used in [12] to decrease the amount of non-volatile memory for an initial configuration bitstream using Virtex 4.

D. Dynamic Partial Reconfiguration for Spartan-6

While it is possible for Virtex architectures to use a standard Partial Reconfiguration tool flow in order to create the partial bitstream for the second configuration, Spartan-6 is not supported by Xilinx for Partial Reconfiguration. Nevertheless, with the right combination of standard implementation techniques and the BitGen option for Difference-based Partial Reconfiguration it is possible to create partial bitstreams which were successfully used for Dynamic Partial Reconfiguration.

As mentioned before and shown in Figure 3, the difference based BitGen option can be used to extract the difference of the full design (B) and the initial design (A). Therefore, the key element of the flow is to create those two designs in a way which ensures, the initial design part doesn't change. This makes sure the partial bitstream for the second configuration only contains information of the second design part.

Keeping the initial design part from changing during the two implementations can be achieved by design preservation using partitions [13]. Those Partitions create logical boundaries between hierarchical modules and thus make it possible to reuse the implementation information of partitions already implemented in a previous design. To preserve the complete routing of the initial design, all IO buffers which are driven by signals from this design part should be instantiated inside the corresponding hierarchical sub-module.

For nets which leave a logical module of the initial design part in order to build a connection to the second design part, the strategy is to route them through an interface logic which is placed outside of the area of the initial design part but belonging logically to the initial design part module and thus to the preserved partition. This can be used to make sure no frames in the area with the first design part are reconfigured when the Dynamic Partial Reconfiguration adds the second design and the connection to the mentioned interface logic. This logic should also provide an enable signal which makes it possible to disable the connection. This is used to avoid glitches, resulting from the configuration of the second design, to reach the first design part. In order to avoid the nets from the second partition to get routed through the area of the first design part the "contained route" constraint should be used for the partition of the second design.

E. Summary of the Fast Start-up Tool Flow for Spartan-6

Figure 4 visualizes the tool flow. It is composed by two runs, one of them is creating the full design (B) and the other run builds the initial design only (A).

In a partition based flow there is always a toplevel partition and at least one sub-level partition. For the Fast Start-up approach the second design part is implemented as the sub-level partition. In order to use partitions in a design, a valid partition description file called "xpartition.pxml" has to be located in the implementation directory to be to be recognized by the frontend tool of the implementation flow (Ngdbuild), compare (2) in the figure. Information on the structure and syntax for such a file can be found in [13]. During the first of the two runs, both partitions are implemented as new partitions.

For the second run, the toplevel partition gets reused (3) but the sub-level partition gets replaced by an empty dummy module (1) and implemented again. By doing so, everything, including the initial design part, gets reused but the second design part. The dummy module is needed since it is not allowed to have empty partitions.

Whenever ncd-files for both designs are available, the method described in section III-C can be used in (5) in order

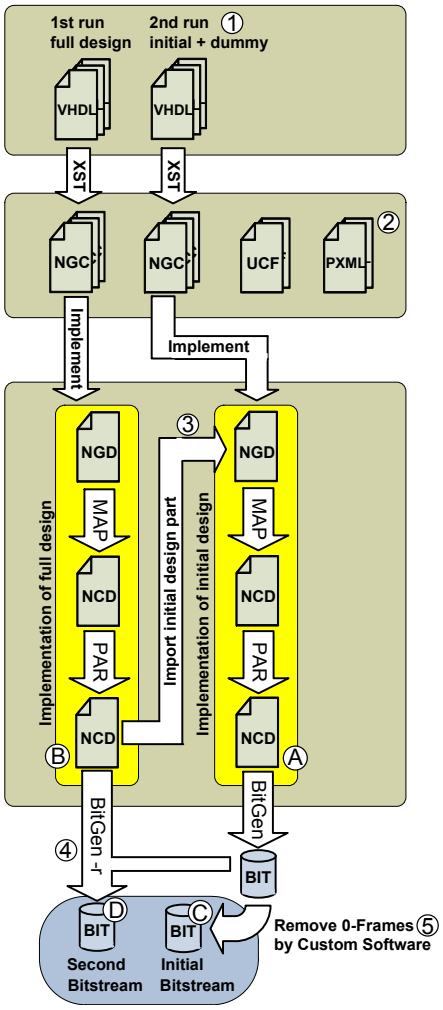

Fig. 4. Fast Start-up flow for Spartan-6

to create the partial bitstream of the initial design part (C), the BitGen option "-r" is used (4) to create the partial bitstream for the second design part (D).

Beside the custom program which was written to automate the removal of the Zero-frames of the initial bitstream, the flow uses standard Xilinx tools only. The approach is not limited to Spartan-6, it can for example also be used for Virtex-5/6. However, when using the Virtex devices we would recommend using the officially supported Partial Reconfiguration Flow to generate the bitstream for the second design part.

IV. Experiments and Results

A. Use case scenario

In order to verify the Fast Start-up technique for Spartan-6 a realistic industrial scenario from the automotive domain was chosen. In today's automotive Electronic Control Units (ECU), sometimes FPGAs are used to implement custom functionality and thus support the main application processing sub-system. Beside the main application sub-system there is usually also a system controller sub-system which handles communication and coordination tasks. Although the FPGA could easily

also implement this system controller using already existing IP-cores, fast start-up requirements add extra system cost therefore inhibiting adoption.

The major reason for these requirements is the need for a very deep sleep mode to meet the tight power budget. The sleep mode is realized by disconnecting almost all components of the ECU, including the system controller, from power. When waking up the system controller has only a limited amount of time to boot and be ready to process the first communication data. For ECUs using the CAN bus for communication this boot-time limit is typically 100ms. As illustrated in Figure 1, it is hard to beat this time limit using a big Spartan-6 with a low cost configuration interface like (Quad-)SPI, but using a faster and therefore more expensive configuration interface is inacceptable in the automotive domain.

B. Measurement setup

The measurement setup is presented in Figure 5. On the left side there is an X1500 automotive platform based on a Spartan-3 implementing a *Traffic Generator* for the CAN bus, which is able to send and receive CAN messages and measure time between messages using hardware timers. On the right side of Figure 5 is the target platform, a Spartan-6 SP605 Evaluation Kit, which is not connected directly to the CAN bus but uses the CAN transceiver from an additional custom board. Besides providing a CAN PHY the mentioned custom board also controls the power supply of the target board.

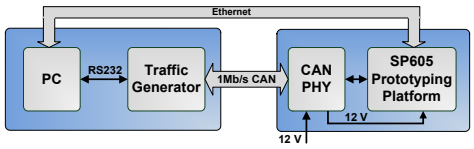

Fig. 5. Measurement setup

C. FPGA design

Figure 6 shows a block diagram of the full FPGA design. A multiplexer is used to separate the designs and implement a defined interface.

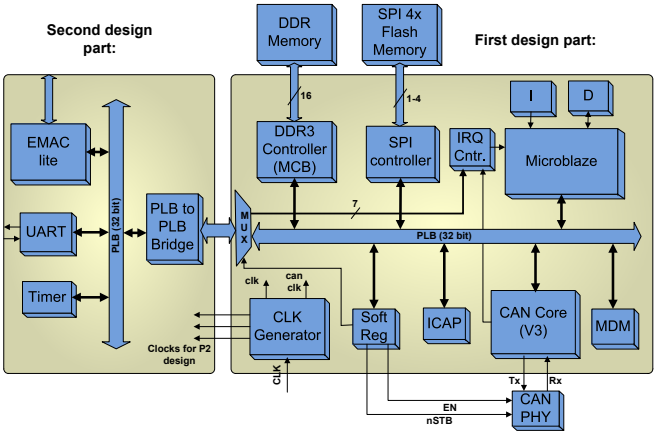

Fig. 6. Block diagram of the full FPGA design

The first design part on the right hand side includes all components of a typical automotive ECU system controller: A Microblaze microprocessor, interfaces to volatile and non-volatile memory, a CAN core for communication and other common EDK modules. A simple register was used to control the enable pin of the multiplexer and the status of the external CAN PHY. Beside the multiplexer the other custom core used in the design is an interface to the ICAP primitive used for configuring the second part of the design. With this the ICAP can be accessed through the PLB bus and can be run with a slower clock than the rest of the system. This was necessary in order to avoid a slow system frequency because running the Spartan-6 ICAP is only specified for a maximum clock frequency of 20 MHz. To create these designs and to run the tool-flow the System Edition version 11.5 of the Xilinx ISE Design Suite was used.

The operating system RTA-Osek from ETAS was run on the Microblaze. RTA-OSEK is a real-time operating system suitable for applications in all areas of automotive ECU design. Different tasks were implemented to process CAN messages, start the configuration for the second design part or start a software application which uses the second design part.

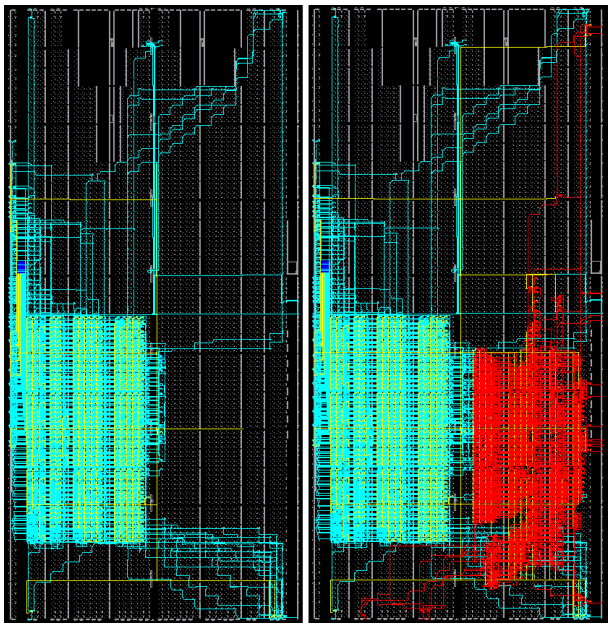

Fig. 7. FPGA Editor view of the initial (left side) and the full (right side) FPGA design. The system clock is highlighted in yellow

As second design an UART core, an Ethernet core and a hardware timer were implemented and connected to the system controller sub-system using a PLB bus. In order to be extendable easily and additionally have a clean separation of the designs a PLB to PLB Bus Bridge was used, which also minimized the nets crossing the border of the two design parts. The second design part is clocked with the same system clock as the first design part. The FPGA editor view of both, the initial design on the left and the full design on the right is illustrated by Figure 7.

D. Measurement process

The procedure to measure the configuration time starts with the Traffic Generator in idle status, the CAN transceiver on the CAN PHY board in sleep mode and therefore the SP605 disconnected from power. In the next step the Traffic Generator starts a hardware timer and sends a CAN message. The activity on the CAN bus is recognized by the CAN PHY which awakes from sleep mode and reconnects the SP605 to the power supply. The FPGA then starts to load the initial bitstream from SPI flash. Because there is no receiver acknowledging the message send by the Traffic Generator, the message will be resent immediately until the FPGA finished its configuration and also configured the CAN core with the valid baud rate. Whenever the message gets acknowledged by the CAN core of the Spartan-6 design, the CAN core of the Traffic Generator triggers an interrupt which stops the hardware timer. This timer is now holding the boot time for the SP605 design. Measurements, which included an additional hardware timer inside the SP605 design, have shown that when executing the software to configure the CAN core from internal BRAM memory, the software start-up time is negligible.

E. Results

The resource consumption for each partition is presented in table I. The percentage information refers to the total amount of available resources of the used XC6S45LXT device.

TABLE I
OCCUPIED FPGA RESOURCES

Resource Type	Partition			
	1st design part	%	2nd design part	%
Flip-flop	3480	6%	1941	4%
LUT	3507	13%	1843	7%
IO	58	20%	20	7%
RAMB	12	10%	2	2%

Table II shows the results of the configuration time measurements. For these measurements, a standard bitstream of the full design, a compressed bitstream of the full design and the Fast Start-up technique using a partial initial bitstream were implemented and compared. The table lists the configuration times for different SPI bus width's and different Config Rate (CR) settings. The Config Rate is an option to determine the target configuration clock frequency in MHz. As expected the configuration times are proportional to the bitstream sizes. Because using a fast configuration clock does not affect the house cleaning process the ratio in percentage stays not the same for high Config Rate settings. Also keep in mind that those numbers are measured and not worst case!

V. CONCLUSION

In this work the first available design tool flow for Dynamic Partial Reconfiguration on Spartan-6 FPGAs has been introduced. This tool flow enables the novel *Fast Start-up* configuration mechanism for modern FPGA's with 2-dimensional configuration memory architectures.

TABLE II
CONFIGURATION TIMES

Configuration Interface	Configuration Technique		
	Traditional 1450 KB	Compressed 920 KB	Fast Start-up 314 KB
SPIx1 CR2	5297 ms	3382 ms	1157 ms
SPIx1 CR40	292 ms	196 ms	85 ms
SPIx2 CR2	2671 ms	1699 ms	596 ms
SPIx2 CR40	161 ms	113 ms	58 ms
SPIx4 CR2	1348 ms	872 ms	311 ms
SPIx4 CR40	97 ms	73 ms	45 ms

Fast FPGA Start-up, which configures the device in two steps (i.e., prioritized FPGA start-up), is essential to address the challenge of increasing configuration time in modern FPGAs, which in other case, would prevent the use of FPGAs in many modern applications, like PCI-express or CAN-based automotive applications. A method to create the high-priority initial configuration was proposed and verified in hardware.

Finally, the developed tool flow and methods for Fast Start-up have been used and tested to implement a CAN-based automotive ECU on a Spartan-6 evaluation board (i.e., SP605). By using this novel approach, it was possible to decrease the initial bitstream size, and hence, achieve a configuration time improvement of up to 78% when compared to a standard configuration solution.

REFERENCES

[1] PCI-SIG, *PCI EXPRESS BASE SPECIFICATION, REV. 1.1*, PCI-SIG, March 2005.

[2] I. Gonzalez, E. Aguayo, and S. Lopez-Buedo, "Self-reconfigurable embedded systems on low-cost fpgas," *Micro, IEEE*, vol. 27, no. 4, pp. 49 –57, 2007.

[3] *Virtex-II Pro and Virtex-II Pro X FPGA User Guide, UG012, v4.2*, Xilinx, November 2007, available at www.xilinx.com.

[4] *Difference-Based Partial Reconfiguration, XAPP290, v2.0*, Xilinx, December 2007, available at www.xilinx.com.

[5] *Early Access Partial Reconfiguration User Guide, UG208, v1.2*, Xilinx, September 2009, available at www.xilinx.com.

[6] *Partial Reconfiguration User Guide, UG702, v12.1*, Xilinx, May 2010.

[7] Z. Li and S. Hauck, "Configuration compression for virtex fpgas," in *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, 2001, pp. 147 – 159.

[8] R. Stefan and S. Cotofana, "Bitstream compression techniques for virtex 4 fpgas," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 2008, pp. 323 –328.

[9] M. Huebner, J. Meyer, O. Sander, L. Braun, J. Becker, J. Noguera, and R. Stewart, "Fast sequential fpga startup based on partial and dynamic reconfiguration," in *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, July 2010, pp. 190 –194.

[10] *Spartan-6 FPGA Configuration User Guide, UG380, v2.1*, Xilinx, February 2010, available at www.xilinx.com.

[11] *Virtex-5 FPGA Configuration User Guide, UG191, v3.8*, Xilinx, August 2009, available at www.xilinx.com.

[12] B. Sellers, J. Heiner, M. Wirthlin, and J. Kalb, "Bitstream compression through frame removal and partial reconfiguration," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 312009-sept.2 2009, pp. 476 –480.

[13] *Hierarchical Design Methodology Guide, UG748, v12.1*, Xilinx, May 2010.