Relocation of Hardware Tasks across Networked Partially Reconfigurable FPGAs

Aurelio Morales Villanueva
Faculty of Electrical and Electronics Engineering
Universidad Nacional de Ingeniería
Lima, Perú
E-mail: amorales@uni.edu.pe

***Abstract*—Partially reconfigurable (PR) field-programmable gate arrays (FPGAs) partition the FPGA into one static region and multiple PR regions (PRRs). This partitioning enables hardware multitasking in the PRRs, where preemption/ resumption of hardware tasks requires saving/restoring the preempted task's execution context with the possibility of relocating the task's context to another PRR. Prior works address the involved challenges, providing partial solutions and imposing limitations that prevent portability of relocating tasks across networked PR FPGAs. In this work, a portable solution for flexible task preemption/resumption/relocation across networked PR FPGAs is introduced, where experimental results evaluate these operations, enabling system designers to tradeoff task/PRR granularity based on application requirements.**

Keywords—field-programmable gate array, hardware task relocation, partial reconfiguration, hardware multitasking

I. Introduction and Motivation

Partially reconfigurable (PR) field-programmable gate arrays (FPGAs) partition the FPGA fabric into one static region and one or more PR regions (PRRs), enabling PRRs to time multiplex hardware (HW) tasks, where incoming tasks can be scheduled to any free/idle PRR (not executing another task) with sufficient resources. Time multiplexing HW tasks reduces the total FPGA area requirements and power consumption.

Reconfiguring a PRR is isolated, where the PRR halts operation during the reconfiguration, while the static region and other PRRs continue executing, enabling task preemption and resumption, where task preemption/resumption pauses/ resumes task execution by saving/restoring the task's execution state (i.e., context). Task preemption requires a context save (CS) operation, which saves the task's context off-chip in a CS bitstream. Task resumption requires a context restore (CR) operation using the previously saved CS bitstream to restore the task's context on the PRR.

There exist prior works on CS and CR (CSR) to the same PRR [1][2], which is too restrictive. HW task relocation (HTR) alleviates this restriction by relocating and resuming task execution in any candidate PRR (with sufficient resources) on the same FPGA. HTR is relatively easy when relocating a task to a homogeneous PRR (same size, shape, and resource distribution, but different fabric location), imposing many restrictions [3][4], and preventing portability across different FPGA device families. HTR for task relocation to a heterogeneous PRR (different size, shape, resource distribution, and fabric location) is significantly more challenging.

Prior works on HTR evaluated off-chip and on-chip implementations. In off-chip HTR between homogeneous PRRs [1], the FPGA is attached to a host CPU that executes software-based HTR. Autonomous on-chip HTR eliminates the CPU-FPGA communication overhead, and can be implemented entirely on the FPGA either using custom HW [3][5], which is not portable across different FPGA device families, or software running on a soft-core processor, or a combination of these two solutions [6].

On-chip software-based HTR enables portability across different FPGA device families, does not generate area overhead (except for the static region) and does not impose device-specific constraints. In order to address limitations in prior CSR/HTR works, on-chip HTR software for different-sized heterogeneous PRRs that executes on a soft-core processor in the FPGA's static region was introduced in [7].

This work leverages the on-chip HTR software [7] for interconnected PR FPGAs, where each FPGA in the network is an autonomous system. Application domains for this work include but are not limited to, dynamic load balancing of HW tasks between FPGAs, distributed fault tolerant systems, and distributed processing of HW tasks in a network of FPGAs.

This work maximizes PRR usage per FPGA, enabling the execution, preemption, and resumption of HW tasks in different-sized heterogeneous PRRs by relocating the task to any PRR (with sufficient resources) on a different FPGA. Also, the experimental results in this work enable system designers to trade off execution times and task granularity (i.e., the task's PRR size) based on application requirements.

II. Background and Related Work

There exists some prior works in CSR for homogeneous PRRs on FPGAs. Joswik [2] presented not portable off-chip CSR methods producing reduced CSR times using direct memory access (DMA) for the internal configuration access port (ICAP) on a Xilinx Virtex-4 device, using a task-specific access structure (TSAS), and inserting custom logic for each task's flip-flop. However, this method incurred significant HW overhead, and lacked portability. In prior work [8], a portable on-chip software CSR for PR FPGAs was introduced, which alleviated these drawbacks, but did not relocate the HW task to a different heterogeneous PRR.

978-1-5386-2278-0/17/$31.00 ©2017 IEEE

There are a few prior works that focus on HTR. In off-chip HTR, Kalte [1] extended off-chip CSR by incorporating the on-chip custom HW relocator REPLICA on older Xilinx Virtex-E device that only supported one-dimensional (1-D) PRRs, and thus is not applicable to newer Xilinx devices that support two-dimensional (2-D) PRRs. In on-chip HTR, Iturbe et al. [6] leveraged their prior work in [5] to propose the R3TOS system using one MicroBlaze and three PicoBlaze on-chip soft-core processors on a Xilinx Virtex-4 device. However, Iturbe's works used embedded random access memory blocks (BRAMs), which are location-specific resources, imposing area overhead and specific constraints, which prevented portability of Iturbe's on-chip HTR across Xilinx FPGA device families.

In order to address the drawbacks of prior off- and on-chip CSR and HTR methods, the on-chip HTR software for 2-D different-sized heterogeneous PRRs was introduced [7]. The on-chip HTR software executes on a soft-core processor in a Xilinx Virtex-5 FPGA's static region, using the ICAP for reconfiguration. The on-chip HTR software does not incur off-chip communication overhead, does not introduce device overhead (except the static region, including a MicroBlaze soft-core processor), has no special constraints on the PRRs and static region (e.g., as in [4][6]), and is portable (with minor changes) across different Xilinx FPGA device families.

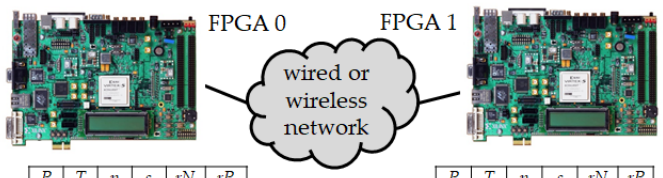
Since low-level device details are critical for understanding CSR and HTR fundamentals, the reader is advised to review the Xilinx Virtex-5 FPGA device configuration [9]. A summary of this device architecture and configuration for the purpose of CSR/HTR is included in [7][10], which can be extended to newer Xilinx FPGA device families.

III. Relocation of HW Tasks for Networked FPGAs

This work extends the on-chip HTR software for 2-D heterogeneous PRRs presented in [7] to enable dynamic context relocation of HW tasks across networked autonomous FPGAs (local wired or wireless). Context relocation of HW tasks across networked FPGAs enables HW multitasking in a distributed fashion, where each FPGA in the network would be able to time multiplex HW resources, such as configurable logic blocks (CLBs), BRAMs, etc., saving the context of a preempted HW task, and resuming the preempted HW task in another FPGA in the network.

This work assumes that the application has already been synthesized and partitioned into multiple HW tasks, each HW task has been assigned a priority execution level, the PRRs and soft-core processors (executing a scheduler that maps and schedules incoming HW tasks to PRRs), and all full and partial bitstreams for all FPGAs in the network have been created.

Even though the network of FPGAs may contain a mixture of different types of FPGA devices, this work focuses on a network with identical FPGAs. However, this work is equally applicable to any network of different FPGA devices. Also, PRRs in this work contain CLBs, LUTRAM for CLBs, and BRAM resources only, since access to flip-flops in digital signal processing blocks (DSPs) is restricted [2]. Additionally, all FPGAs are interconnected over a local wired Gigabit Ethernet network, but any interconnection network could be used for remote context relocation of HW tasks.



	P	T	p	s	rN	rP
t_1	0	A	1	ex	1	0
t_2	1	B	2	ex	1	1
t_3	1	E	1	sc	1	0
t_4						
t_5						
t_6						
t_7	0	A	1	pr	1	0
t_8	0	F′	2	ex	1	1

	P	T	p	s	rN	rP
t_1	0	C	0	ex	0	0
t_2	1	D	3	ex	0	1
t_3						
t_4	0	C	0	pr	0	0
t_5	0	E′	1	ex	0	1
t_6	1	F	2	sc	0	0

Fig. 1. Network of two autonomous FPGA systems, with a sequence of events that involve remote context relocation (rHTR).

A. Remote HTR Overview

Fig. 1 shows an overview of context relocation of HW tasks between two networked autonomous FPGA systems, with a sequence of events t_x. Each FPGA has two heterogeneous PRRs, where P = PRR number, T = HW task name, p = HW task's priority execution level (3 = highest level, 0 = lowest level), s = HW task status (*ex* = executing, *sc* = scheduled for execution, *pr* = preempted), rN = remote FPGA for remote HTR (rHTR), and rP = remote PRR for rHTR.

At t_1 and t_2, HW tasks A and B start execution in FPGA 0, on PRRs 0 and 1, respectively, while tasks C and D start execution in FPGA 1, on PRRs 0 and 1, respectively. At t_3, HW task E is scheduled for execution in FPGA 0 on PRR 1, however, task E cannot start execution on PRR 1, since task B has a higher priority execution level. Alternatively, task E can start execution in FPGA 1 (rN = 1), on PRR 0 (rP = 0), since task C has a lower priority execution level. Then, at t_4, task C is preempted and at t_5, task E is remotely relocated to FPGA 1 to execute as task E′. At t_6, task F is scheduled for execution in FPGA 1 on PRR 1, however, task F cannot start execution on PRR 1 since task D has a higher priority execution level. Alternatively, task F can start execution in FPGA 0 (rN = 0), on PRR 0 (rP = 0), since task A has a lower priority execution level. Then at t_7, task A is preempted and at t_8, task F is remotely relocated to execute in FPGA 0 as PRM F′.

B. Remote HTR across Networked PR FPGAs

Remote HTR (rHTR) uses a network of N FPGAs, where i represents the FPGA number ($0 \leq i < N$), FPGA i contains P_i PRRs, where prr_{ij} denotes PRR j on FPGA i ($0 \leq j < P_i$), and prr_{ij} can time multiplex up to M HW tasks, where $task_{ijk}$ denotes a HW task with index k for PRR j in FPGA i ($0 \leq k < M$). Due to rHTR's distributed nature, inconsistencies in the network of FPGAs must be avoided by broadcasting all task and PRR status changes to all of the FPGAs every time there is a change in a FPGA. Thus, all FPGAs have the same tasks' status, and which FPGAs/PRRs the tasks are assigned to.

In order to maintain data consistency and perform context relocation of HW tasks across networked FPGAs, rHTR uses the client/server model for remote procedure call (RPC). RPC [11] is a widely-used model for communication in distributed systems, and using RPC in rHTR enables programs in one *local* FPGA to remotely execute code in a *remote* FPGA.

Fig. 2 depicts the rHTR's flow for two networked PR FPGAs, with a local FPGA *i* and a remote FPGA *m*, FPGA *i* has one PRR prr_{ij} and FPGA *m* has one PRR prr_{mn}, and FPGA *i* has one HW task $task_{ijk}$ while FPGA *m* has one HW task $task_{mnp}$. Since rHTR uses RPC's client/server, a portion of the rHTR code executes on the client side in local FPGA *i*, and a portion of the rHTR code is called from the client side of FPGA *i* and is executed on the server side of remote FPGA *m*. Unshaded rectangular and decision boxes are executed on the client side of local FPGA *i*. Light gray rectangular and decision boxes are called from the client side of FPGA *i* and executed on the server side of remote FPGA *m*. Boxes with cross lines are called from the client side of FPGA *i* and executed sequentially (FPGA *0*, *1*, … , *N-1*) on the server side of remote FPGAs. Finally, the dark gray box is for remote relocation of a HW task, executed entirely in the remote FPGA.

In Fig. 2, lock() defines if a PRR is locked (coded as '1'), or unlocked (code as '0'), where a PRR is locked if a task assigned to the PRR is changing the status. Also, avail() defines if a PRR is free (coded as '1') or busy (coded as '0'), where a free PRR is not currently executing a task. Finally, prio() defines the priority level of current task in the PRR.

Since this work focuses on rHTR across networked PR FPGAs, the reader is advised to read [7] and [8] for a complete understanding of HTR and CSR, respectively. Fig. 2 assumes that $task_{ijk}$ is scheduled for execution in a locked prr_{ij} or prr_{ij} is executing a higher priority execution task. If $task_{ijk}$ is mapped in remote candidate prr_{mn}, rHTR checks for lock() and avail() of prr_{mn} (T_{chk_lock}). If prr_{mn} is free and not locked, rHTR locks prr_{mn} (T_{lock}) and checks if $task_{ijk}$ was previously preempted (CS bitstream exists for $task_{ijk}$) or not (T_{ftp_bit}). Then, remote context relocation (rHTR) of $task_{ijk}$ is executed (T_{rhtr}), and finally the status of the relocated task is broadcasted (T_{broad1}). If prr_{mn} is executing a lower priority execution level $task_{mnp}$ (T_{chk_pri}), rHTR locks prr_{mn} (T_{lock}), preempts $task_{mnp}$ (T_{cs_rtask}), broadcast the status of the preempted task to other FPGAs (T_{broad2}), and performs the remote context relocation (rHTR) of $task_{ijk}$.

From Fig. 2, the total execution time for a remotely relocated HW task to start execution (or resume execution) in a free and unlocked remote candidate PRR (denoted as T_{exe1}), and in a busy and unlocked remote candidate PRR executing a lower priority HW task (denoted as T_{exe2}), are expressed as:

$$T_{exe1} = T_{chk\_lock} + T_{lock} + T_{ftp\_bit} + T_{rhtr} + T_{broad1} \quad (1)$$

$$T_{exe2} = T_{chk\_lock} + T_{chk\_pri} + T_{lock} + T_{cs\_rtask} + T_{broad2} + T_{ftp\_bit} + T_{rhtr} + T_{broad1} \quad (2)$$

C. Remote HTR Portability across FPGAs device families

Remote HTR (rHTR) portability is ensured by using a portable language for the HTR software application for different FPGA device families. HTR uses the standard C language, using configuration files selected at compilation time for a device's specific architecture. The configuration files (*.far) used in the HTR specify the FAR addresses [9] for each resource column (CLB, DSP, etc.) per row in the specific FPGA device. The bram.far, clb.far, clk.far, dsp.far, and iob.far files in the HTR software specify the FAR addresses for all of the BRAM, CLB, CLK, DSP, and IOB columns [9] in the FPGA, respectively. Also, a C header file (htr.h) defines all of the ICAP commands' coding using the #define directive for the Virtex-5 FPGA device family. For HTR compatibility across FPGA device families, changes to htr.h and *.far are needed.

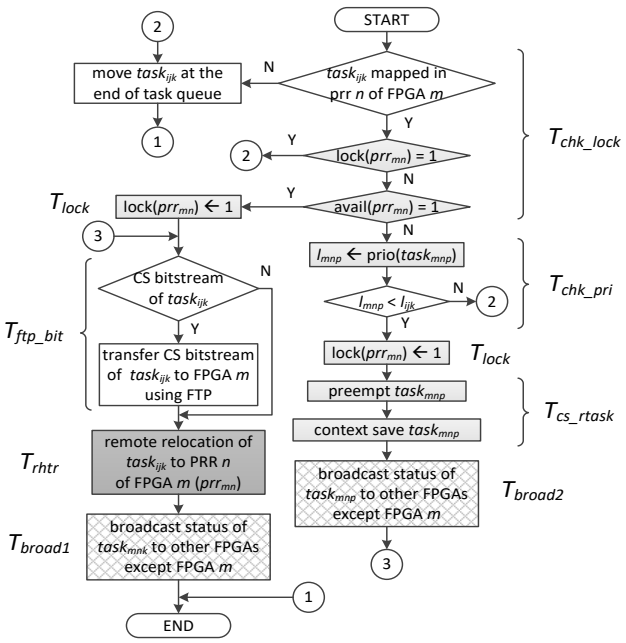


Fig. 2. Flow for the execution of remote context relocation of a HW task to a remote candidate PRR (rHTR).

IV. Experimental Results

This section presents the experimental results of the on-chip remote context relocation (rHTR) of HW tasks between 2-D different-sized heterogeneous PRRs across networked PR FPGAs, for varying PRR sizes and organizations. Note that these experimental results may be improved by using a high-speed hardcore processor embedded in the FPGA fabric, as in Zynq-7000, which was not available for the experiments.

A. Experimental Setup

This work evaluates rHTR across networked PR FPGAs using four Xilinx XUPV5 boards (a Virtex-5 LX110T device in each board), and interconnected in a local Gigabit Ethernet network, where each FPGA in the network has the same setup (static region and PRRs) and Linux-like OS as for the HTR setup [7]. The PRR sizes for the HW tasks in the experiments contained one row, one BRAM column (RAMB36), and multiple CLB columns ranging from one to twelve. On each CLB column that includes a SLICEM slice type, a LUTRAM (RAM32M) was included. The partial bitstreams generated for the experiments range from 31.9 KB to 95.4 KB, and the CS bitstreams for the preempted tasks range from 22 KB to 28.9 KB. The execution of rHTR considers that the remote PRR has twice as many CLB columns as compared to the local PRR [7].

The execution times denoted as T_x (measured as in [10]) consider one local predefined PRR and one remote candidate PRR for rHTR per each HW task. Adding more remote PRRs for rHTR enhances context relocation flexibility, maximizing PRR usage in each FPGA. However, adding more remote PRR candidates will negatively impact the execution times.

TABLE I. RHTR EXECUTION TIMES (ms) FOR RELOCATING A HW TASK IN A REMOTE CANDIDATE PRR EXECUTING A LOWER PRIORITY TASK

Execution times	HW task flip-flops in the PRRs (each PRR includes one BRAM column)											
	160	**320**	**480**	**640**	**800**	**960**	**1120**	**1280**	**1440**	**1600**	**1760**	**1920**
T_{chk_lock}	16.4	16.4	16.4	16.4	16.4	16.4	16.5	16.4	16.1	16.4	16.4	16.4
T_{chk_pri}	32.8	32.8	32.8	32.7	32.8	32.9	32.8	32.8	32.8	32.8	32.8	32.8
T_{lock}	7.3	7.4	7.4	7.3	7.3	7.3	7.3	7.3	7.4	7.3	7.3	7.3
T_{cs_rtask}	31.6	33.4	36.5	38.0	41.9	43.5	47.0	48.3	52.1	53.3	57.3	58.3
T_{broad2}	29.1	28.7	28.7	27.4	28.8	27.5	27.5	30.3	28.8	28.6	27.5	28.6
T_{ftp_bit}	416.2	412.7	408.8	407.7	414.4	414.0	414.4	406.2	414.7	411.7	412.6	411.9
T_{rhtr}	75.2	102.6	141.6	193.5	261.5	337.3	428.9	532.7	649.2	778.5	920.7	1075.7
T_{broad1}	27.5	27.3	27.5	27.3	27.5	27.6	27.4	27.4	27.4	27.4	27.4	27.4
T_{exe2}	683.1	708.0	746.9	797.0	877.4	953.4	1048.7	1148.2	1275.6	1402.9	1548.8	1705.3

B. Remote HTR Execution Times

Table I shows the execution times in T_{exe2} for relocating a HW task in a remote candidate PRR executing a lower priority execution level task. T_{exe2} depends on the behavior of T_{cs_rtask} and T_{rhtr}. All other times are almost constant. T_{cs_rtask} depends on the number of LUTRAMs, BRAMs and flip-flops of the preempted remote task, showing linear behavior. T_{rhtr} shows similar behavior as the small-to-large PRR HTR [7]. Execution times for T_{exe1} can be obtained from Fig. 2 and Table I.

No prior work evaluated rHTR of HW tasks across networked FPGAs for 2-D different size heterogeneous PRRs. Not even in [12] (using old Virtex II Pro FPGA that only supports 1-D PRRs, scrubbing the PRRs, and without performing CS and CR), or in [13] (using non-PR Altera Cyclone FPGAs configured with a Nios-II soft-core processor).

V. CONCLUSIONS AND FUTURE WORK

This work extends the on-chip hardware (HW) task relocation (HTR) software to work with multiple partially reconfigurable (PR) field-programmable gate arrays (FPGAs) to enable context relocation of HW tasks that execute on two-dimensional (2-D) different-sized heterogeneous PR regions (PRRs) across networked FPGAs. Each FPGA in the network is an autonomous stand-alone system running Linux OS on a MicroBlaze soft-core processor, without introducing device overhead (except the static region in each FPGA), having no special constraints on the PRRs, and being portable (with minimum changes) across different FPGA device families.

This work maximizes the use of PRRs in each FPGA, enabling the execution, preemption, and resumption of HW tasks across networked FPGAs, for application domains such as dynamic load balancing of HW tasks, distributed fault tolerant systems, distributed processing of HW tasks across networked FPGAs, etc. This work shows results for a network of multiple PR Virtex-5 FPGAs, however, the fundamentals are applicable to newer Xilinx device families, such as Virtex-6, the 7 series, Zynq-7000, and UltraScale.

The results show the growth rates (quadratic behavior) of rHTR times as the PRR size increases, revealing that this work is most attractive for smaller PRRs, allowing the system designer to determine the application partitioning in tasks and the task-to-PRR mappings (considering PRR sizes), according to the application requirements. Since the HTR executes on a soft-core processor in each networked FPGA, most of execution times in this work may be reduced using a high-speed hardcore processor embedded in the FPGA.

Future work will extend the on-chip HTR to support different FPGA architectures, optimize HTR times, and incorporate HTR with a run-time reconfiguration scheduler for preemptive HW multitasking on multiple PR FPGAs.

REFERENCES

[1] H. Kalte and M. Porrmann, "Context saving and restoring for multitasking in reconfigurable systems," Proc. 15th Int'l Conf. Field Programmable Logic and Applications (FPL), pp. 223-228, 2005.

[2] K. Jozwik, H. Tomiyama, M. Edahiro, S. Honda, and H. Takada, "Comparison of preemption schemes for partially reconfigurable FPGAs," IEEE Embedded Systems Letters, vol. 4, no. 2, pp. 45-48, Jun. 2012.

[3] C. Beckhoff, D. Koch, and J. Torresen, "Portable module relocation and bitstream compresion for Xilinx FPGAs," Proc. 24th Int'l Conf. Field Programmable Logic and Applications (FPL), pp. 1-8, 2014.

[4] T. Drahonovsky, M. Rozkovec, and O. Novak, "A highly flexible reconfigurable system on a Xilinx FPGA," Proc. Int'l Conf. on Reconfigurable Computing and FPGAs (ReConFig), pp. 1-6, 2014.

[5] X. Iturbe, K. Benkrid, T. Arslan, R. Torrego, and I. Martinez, "Methods and mechanisms for hardware multitasking: executing and synchronizing fully relocatable hardware tasks in Xilinx FPGAs," Proc. 21st Int'l Conf. Field Programmable Logic and Applications (FPL), pp. 295-300, 2011.

[6] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, and T. Arslan, "Microkernel architecture and hardware abstraction layer of a reliable reconfigurable real-time operating system (R3TOS)," ACM Trans. Reconfigurable Technology and Systems (TRETS), vol. 8, no. 1, pp. 5:1-5:35, Feb. 2015.

[7] A. Morales-Villanueva and A. Gordon-Ross, "HTR: on-chip hardware task relocation for partially reconfigurable FPGAs," Proc. 9th Int'l Symp. Applied Reconfigurable Computing (ARC), pp. 185-196, 2013.

[8] A. Morales-Villanueva and A. Gordon-Ross, "On-chip context save and restore of hardware tasks on partially reconfigurable FPGAs," Proc. 21st Ann. IEEE Int'l Symp. Field-Programmable Custom Computing Machines (FCCM), pp. 61-64, 2013.

[9] Xilinx, Virtex-5 FPGA Configuration User Guide (UG191 v3.12), 2017.

[10] A. Morales-Villanueva, R. Kumar, and A. Gordon-Ross, "Configuration prefetching and reuse for preemptive hardware multitasking on partially reconfigurable FPGAs," Proc. 2016 Design, Automation and Test in Europe Conference & Exhibition (DATE), pp. 1505-1508, 2016.

[11] A. S. Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms, 2nd ed. Upper Saddle River, NJ: Pearson Education Inc., 2007, pp. 125-140.

[12] S. Wichman et al., "Partial reconfiguration across FPGAs," Proc. 2006 Military and Aerospace Applications of Programmable Logic Devices and Technologies, pp. 26-28, 2006.

[13] C. Haubelt, D. Koch, F. Reimann, T. Streichert, and J. Teich, "ReCoNets – Design methodology for embedded systems consisting of small networks of reconfigurable nodes and connections," Dynamically Reconfigurable Systems: Architectures, Design Methods and Applications, M. Platzner, J. Teich, and N. Wehn, Eds., Heidelberg, Germany, pp. 223-243, 2010.