



A Reconfigurable Design Framework for FPGA Adaptive Computing

Ming Liu^{†‡}, Wolfgang Kuehn[†], Zhonghai Lu[‡],
Shuo Yang[†], Axel Jantsch[‡]

[†] Justus-Liebig-University Giessen (JLU) , Germany

[‡] Royal Institute of Technology (KTH), Sweden



Outline

- ◆ **Introduction & Motivation**
- ◆ **Reconfigurable Framework for Adaptive Computing**
 - **HW infrastructure**
 - **OS, device drivers & scheduler SW**
 - **Context saving and restoring**
- ◆ **A Case Study**
- ◆ **Technical Perspectives in applications**
- ◆ **Conclusion & Future Work**





Introduction & Motivation

- ◆ Adaptive computing: algorithms adapted to ambient conditions during system run-time.
- ◆ Benefits:
 - Higher performance
 - Lower power consumption
 - Multitasking on limited resources
 - ...
- ◆ Conventional adaptive multitasking on general-purpose CPUs + OSES: well-fledged as the development of OSES & scheduler
 - Computing resources (CPU) intelligently and efficiently utilized
- ◆ In the FPGA world???
 - Static designs? Not adaptive
 - Partial Reconfiguration (PR)? Technical support
- ◆ Motivation: a complete design framework for more efficient hardware resource management, based on FPGA PR technology.

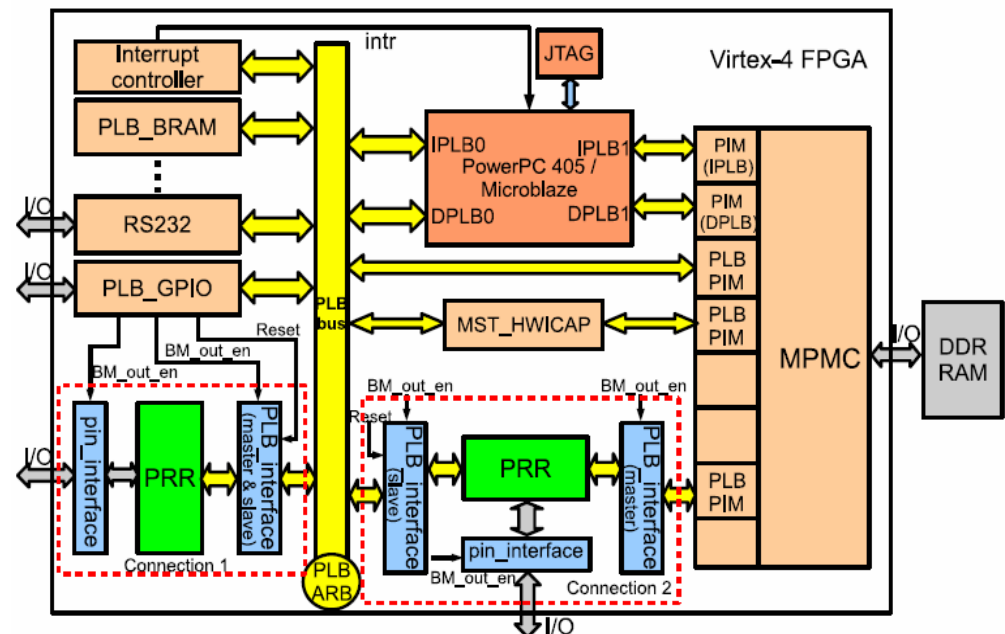


Framework (HW Infrastructure)

- ◆ Xilinx PR design flow [1]
- ◆ Modular algorithm designs (A1, A2, ...)
- ◆ PR Region (PRR)
- ◆ PR communication interface (BMs)
- ◆ System manager (GP CPU)
- ◆ Peripherals, memories, ...
- ◆ ICAP design [2]
 - mst_hwicap
 - Conf. speed: ~235 MB/s
 - Conf. overhead: xx μ s to xxx μ s

[2] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration", *In Proc. Of the International Conference on Field Programmable Logic and Applications*, Aug. 2009.

[1] Xilinx Inc., "Early Access Partial Reconfiguration User Guide for ISE 8.1.01i", UG208 (v1.1), Mar. 2006.





Framework (OS, Drivers & Scheduler)

- ◆ Embedded OS or Standalone
- ◆ Device drivers for algorithm modules
 - Software control registers
 - Interrupts
- ◆ Algorithm scheduler
 - Application programs (flexible & portable)
 - Monitors ambient conditions and triggers algorithm switching
 - HW processes are preemptable and comply with the scheduler
 - Flexible disciplines

```
int scheduling(void) {  
    if((data_in_fifo0 - data_in_fifo1) > THRESHOLD) {  
        switching_to_hw_process = 0; // Context switching to hw process 0,  
                                     // due to much buffered data in fifo0.  
    }  
    else if((data_in_fifo1 - data_in_fifo0) > THRESHOLD) {  
        switching_to_hw_process = 1; // Context switching to hw process 1,  
                                     // due to much buffered data in fifo1.  
    }  
    else {  
        switching_to_hw_process = switching_to_hw_process; // Keep unchanging,  
                                                           // to reduce reconfiguration overhead.  
    }  
    return switching_to_hw_process;  
}
```

```
int scheduling(void) {  
    if(event_in_fifo0 != 0) {  
        switching_to_hw_process = 0; // Context switching to hw process 0 at once,  
                                     // since algorithm 0 has higher priority.  
    }  
    else if(event_in_fifo1 != 0) {  
        switching_to_hw_process = 1; // Context switching to hw process 1.  
                                     // It has lower priority, but also RT requirement.  
    }  
    else {  
        switching_to_hw_process = switching_to_hw_process; // No event happened.  
                                                           // Keep unchanging,  
    }  
    return switching_to_hw_process;  
}
```




Framework (Context Switching)

◆ Context

- Control registers
- Buffered incoming data
- Intermediate calculation results
- To be saved and restored for algorithm modules in many cases

◆ Concrete approaches [3][4]:

- Register read and write
- Bitstream readout and analysis

[3] H. Kalte and M. Porrmann, "Context Saving and Restoring for Multitasking in Reconfigurable Systems", *In Proc. of the International Conference on Field Programmable Logic and Applications*, Aug. 2005.

[4] C. Huang and P. Hsiung, "Software-controlled Dynamically Swappable Hardware Design in Partially Reconfigurable Systems", *EURASIP Journal on Embedded Systems*, Jan. 2008.





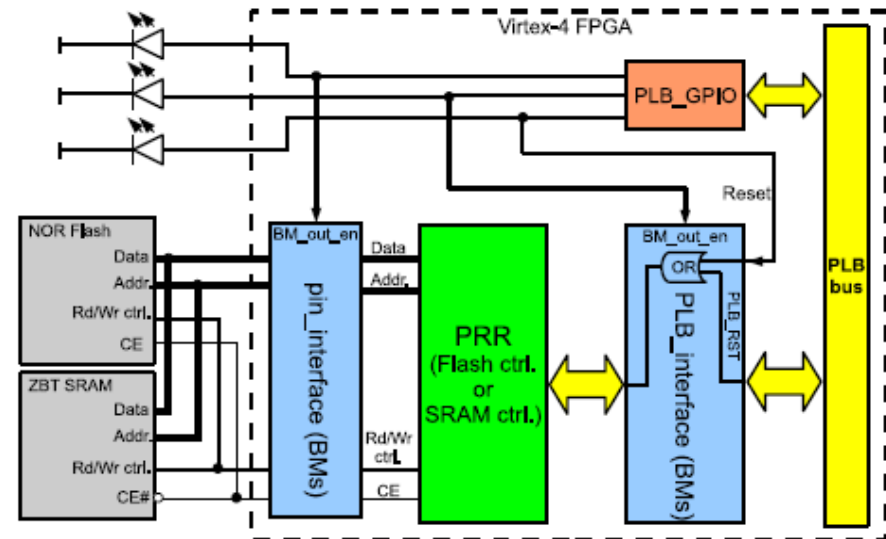
A Case Study

- ◆ A case study switching a NOR flash memory controller and an SRAM controller (V4-FX20)
- ◆ A pre-verification for algorithm switching in real applications
 - Existing IP cores and no need to modify
 - Same connection interfaces (PLB, I/Os)
 - To save I/O pins and resources on the FPGA
- ◆ Motivation:
 - NOR flash memory: embedded Linux kernel
 - SRAM: LUT storage for application-specific computation
 - To share FPGA resources and access different memories according to system requirements

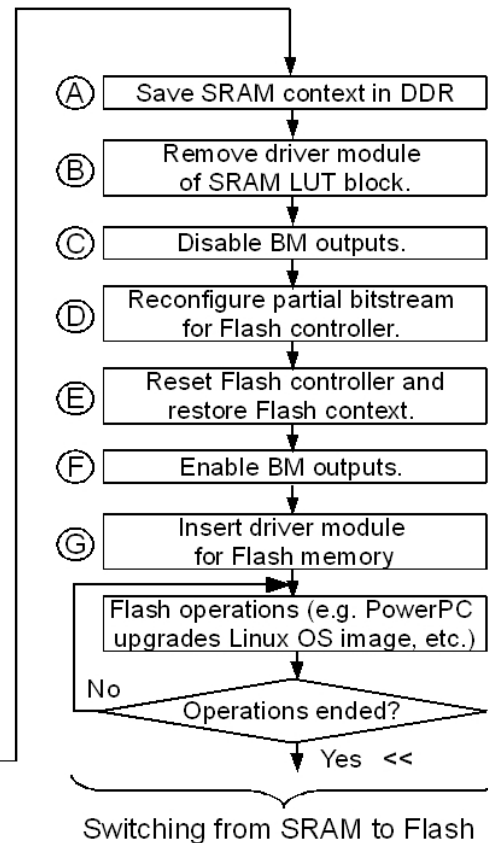
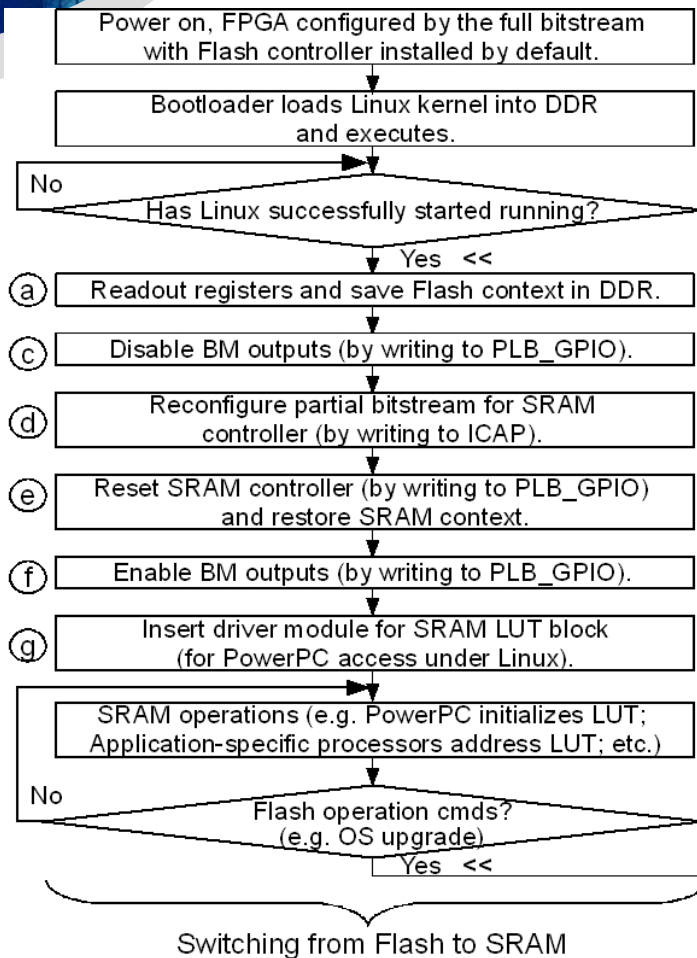


A Case Study (HW Design)

- ◆ PR Region (PRR) reserved
- ◆ Flash and SRAM controller to be loaded: standard IP cores and no need to modify
- ◆ Shared I/O pins to external devices
- ◆ Customized BM interfaces
 - To lock signal routing between static & PR designs
 - BM_out_en to isolate unpredictable outputs during reconfiguration
 - Reset to solely reset the newly loaded core after reconfiguration
- ◆ GPIO controls
 - BM_out_en
 - Reset



A Case Study (Operation Flow)

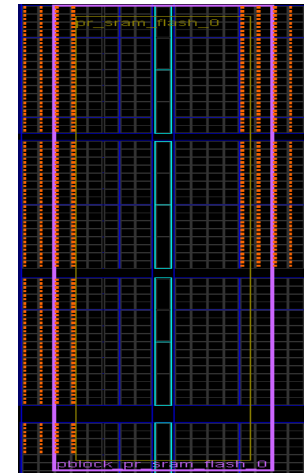


- ◆ Operations in Linux
- Context saving
 - Remove device driver
 - Disable BM outputs
 - Module reconfiguration
 - Reset the module
 - Re-enable BM outputs
 - Insert device driver

A Case Study (Results)

◆ Evaluation

- $L_{static} = L_1 + L_2 + \dots + L_n$ (static LUTs)
- $F_{static} = F_1 + F_2 + \dots + F_n$ (static Flip-Flops)
- $L_{PR} = L_{PRR} + 1/2 * L_{BM}$ (PR LUTs)
- $F_{PR} = F_{PRR}$ (PR Flip-Flops)
- $L_{PRR} = F_{PRR} = \text{Max}[(L_1 + 1/2 * L_{BM}), \dots (L_n + 1/2 * L_{BM}), F_1, \dots F_n] + R_{margin}$ (to reserve the PR region)



◆ Resource utilization benefits with PR

	■ static	● pr design	◇ util. factor
I/O pins	■ 56(s)+61(f)	● 61	◇ 52.1%
4-LUTs	■ 954(s)+923(f)	● 1624(prr+1/2*BM)	◇ 86.5%
Slice FFs	■ 728(s)+867(f)	● 1296(prr)	◇ 81.3%

◆ Reconfiguration overhead: 299 μ s for 71 KB bitstream

◆ More benefits foreseen when more and larger algorithm modules multiplex the PR region





Application Perspectives

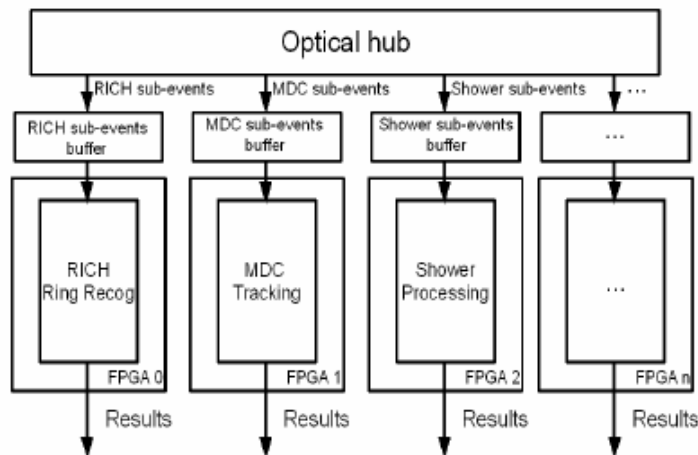
- ◆ To be applied and verified in nuclear and particle physics experiments (HADES, PANDA, WASA, etc..)
- ◆ Large-scale massive computing for online data acquisition (DAQ) and triggering based on FPGA clusters
- ◆ Motivation:
 - Multiple pattern recognition algorithms
 - Multiple cores for parallel processing
 - Different computation features for algorithms (computation-bounded, memory-bounded, ...)
 - Conventional approach: algorithm partitions are statically distributed on FPGA nodes by designers
 - Too complicated to manage and modify the design



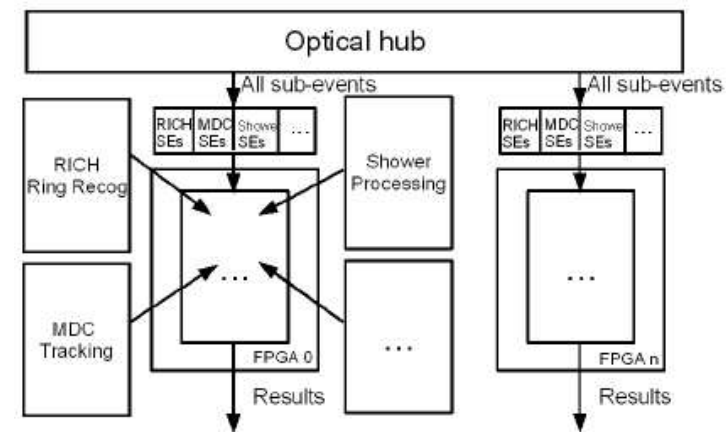
Application Perspectives

- ◆ Expected benefits from adaptive computing:
 - Easy dynamic design management
 - Efficient resource utilization for higher performance
 - Reduced FPGA size/count (costs)

Static design:



Adaptive design:



- Uniform design in adaptive computing – easy to maintain system designs
- No data distribution requirements for optical hubs (all kinds of sub-events fed into all FPGAs)
- Balanced computing and more efficient FPGA resource utilization [5]

[5] M. Liu, Z. Lu, W. Kuehn, and A. Jantsch, "FPGA-based Adaptive Computing Framework for Correlated Multi-stream Processing", In Proc. of the Design, Automation & Test in Europe Conference 2010, to appear.



Conclusion and Future Work


Conclusion:

- ◆ A design framework for FPGA-based adaptive computing
- ◆ Key aspects discussed
- ◆ A case study using general memory controllers
- ◆ Technical perspectives in target applications

Future Work:

- ◆ Individual in-depth research in different aspects of the framework
- ◆ Verification with real algorithms for physics experiments

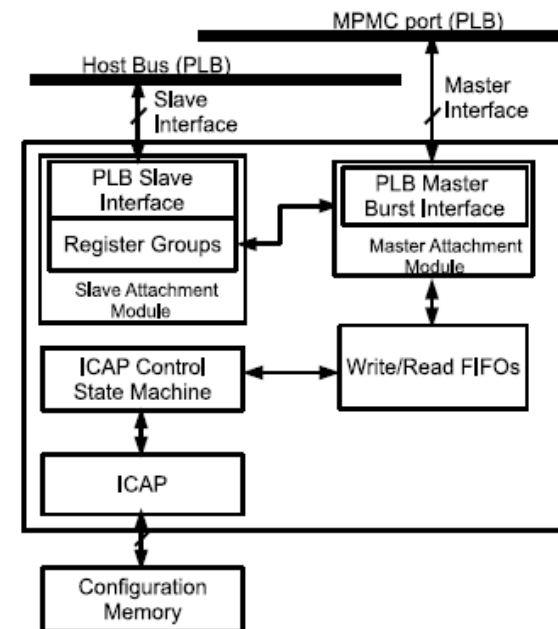




Thank You !

Framework (PR Tech. Support)

- ◆ ICAP for FPGA configuration
- ◆ ICAP designs [2]
 - Xilinx opb_hwicap
 - Xilinx xps_hwicap
 - Improved mst_hwicap
 - Improved bram_hwicap
- ◆ Practical mst_hwicap
- ◆ Conf. speed: ~235 MB/s (from DDR memory)
- ◆ Conf. overhead: $xx \mu s - xxx \mu s$



[2] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration", *In Proc. Of the International Conference on Field Programmable Logic and Applications*, Aug. 2009.