

# FPL-BENCHMARK ACTIVITIES

---

Andreas Beyer

Udo Keschull

Kirchhoff Institute for Physics

University of Heidelberg, 02.09.2010

# Motivation

- Observations:
  - Authors select a different set of design examples to prove their results
    - It is a hard job to implement a full featured design example from scratch
    - There is only poor comparison to other approaches
  - Most well-known benchmark suits are rather outdated
    - e. g. MCNC 93
- First discussions during FPL 09 in Prague

# Motivation

- Idea:
  - Turn **fpl.org** into a community web page
  - Use **fpl.org** as a platform to collect and provide FPGA related benchmarks
    - Collect various design examples used in publications and make them available for other researchers who are working at the same research topic
    - Download area for benchmarks
    - Chat room and forum
    - Bring in touch researchers who work in the same research area
- Community driven benchmark collection

# Motivation

- Current status:
  - This is only a preliminary platform implementation to demonstrate how such a benchmark suite could look like
  - We will not be able to implement and maintain all provided designs
    - We need your help!
- However:
  - **fpl.org** could be seen as a basic infrastructure to collect and offer designs which can easily be used for benchmarking your research

# Presentation

- Motivation
  - Introduction
  - Benefits
- Implementation
  - Structure
  - Representation
- Future work

# Definition of “benchmarking”:

- process of comparing one's performance metrics to
  - industry bests
  - best practices
- dimensions typically measured are
  - quality / time / cost
- Improvements from learning mean doing things
  - better / faster / cheaper

# Current situation

- Free (popular) benchmark suites for FPGAs
  - MCNC (Circuits) → 1993
  - RAW (Algorithms) → 1997
  - MiBench (various applications) → 2002
  - IWLS (Circuits) → 2005
- Commercial suites
  - EEMBC
  - AcceIFPGA
- mostly 1:1 comparison in papers
  - no possibility for comparison with several projects

# Goals

- assembling a set of benchmarks
  - to represent state of the art technologies
  - offer a starting point for further developments
- creating presentation platform
  - structured by fields of operation
  - make implementations comparable
  - follow the evolution process of the codes



# Abstract view on development



- This is a community driven benchmark suite
  - We want to support your workflow
  - It's not our intention to keep you busy
  - overhead of maintainers and contributors has to be kept to a minimum

# Abstract view on development



- Define the problem you are working on

# Abstract view on development



- Define the problem you are working on
- Describe idea and approach

# Abstract view on development



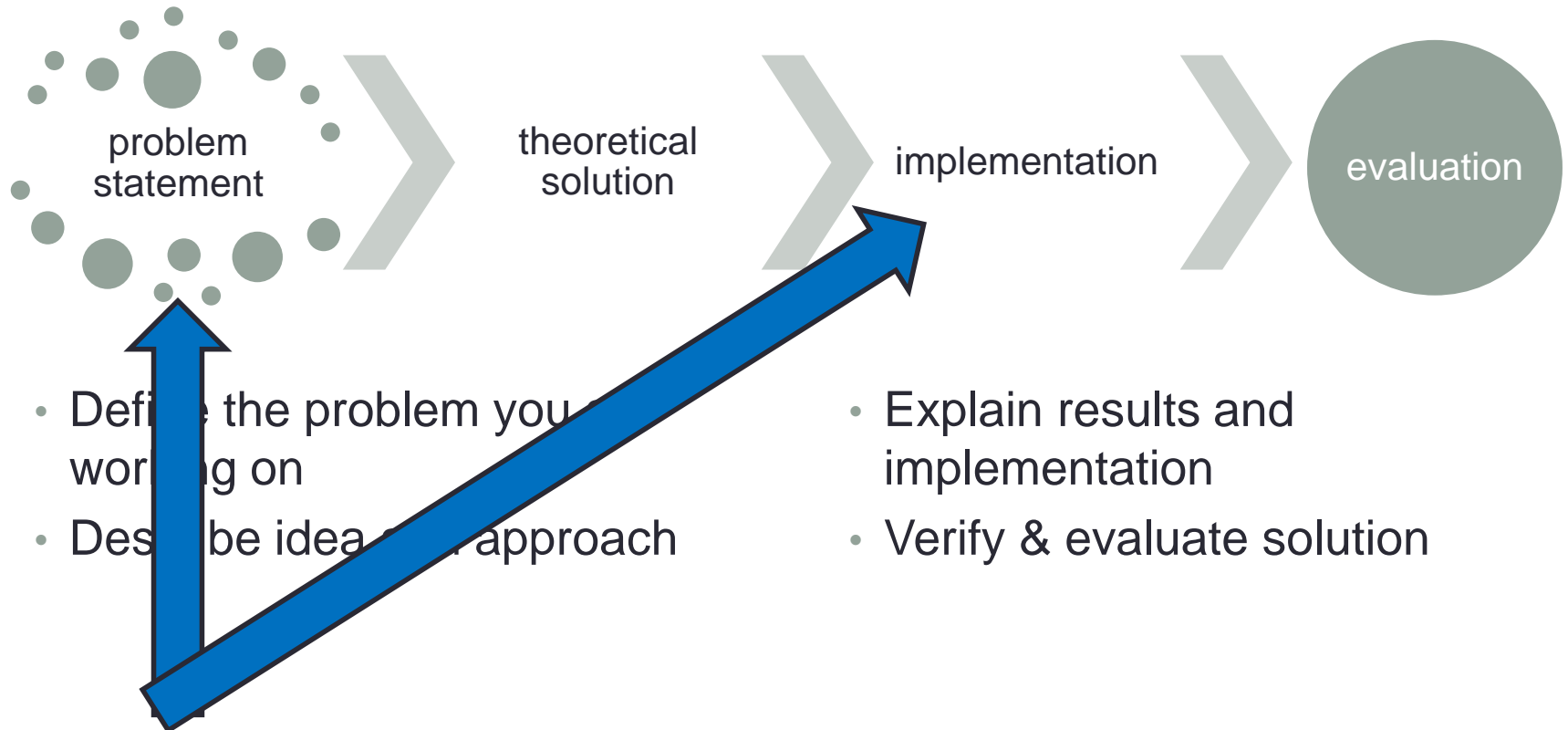
- Define the problem you are working on
- Describe idea and approach
- Explain results and implementation

# Abstract view on development



- Define the problem you are working on
- Describe idea and approach
- Explain results and implementation
- Verify solution & evaluate selected metrics

# Abstract view on development



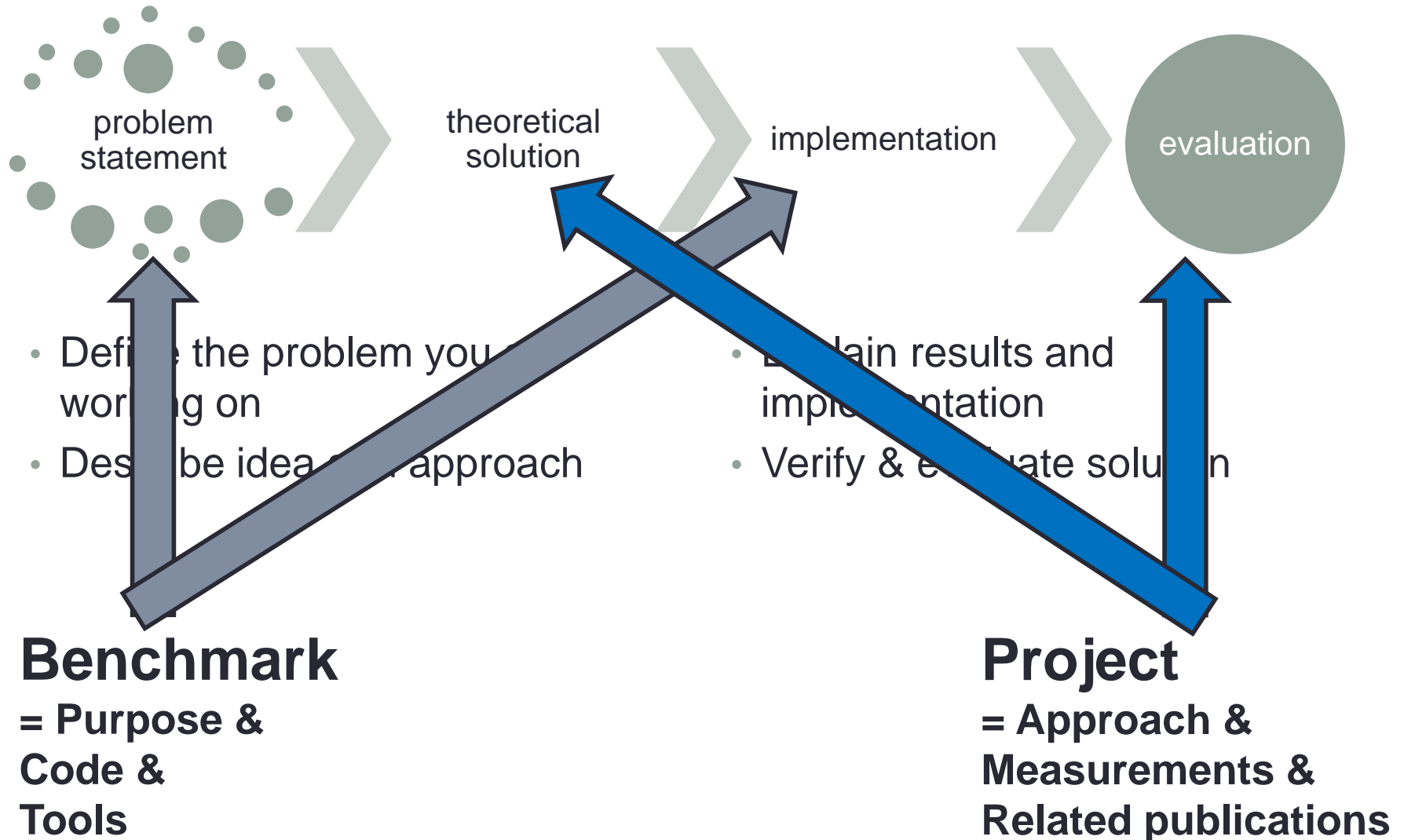
- Define the problem you are working on
- Describe idea of your approach

- Explain results and implementation
- Verify & evaluate solution

**Benchmark**

**= Purpose &  
Code & Tools**

# Abstract view on development



# Realization

- The suite consists of Projects and Benchmarks
- Each benchmark is used in different projects
  - as input data
  - as basis for further implementations
- Each project is based on one benchmark
  - the posted results are comparable
    - to the original benchmark
    - to other projects aiming for different features
  - projects are arranged
    - in categories according to their field of operation
    - in the same line as the corresponding benchmark



# Structure (1) – Overview

**Benchmark-suite** Besuchersicht

[Welcome](#)  
[Overview](#)  
[Benchmarks](#)  
[Projects](#)  
[Contact](#)  
[Participating groups](#)

The following table provides a set of typical applications or implementations. Each one poses a challenge in at least one of the given categories in current FPGA-research. The first column lists sources and their descriptions, entries in each row refer to a project using them for the appropriate category.

If the category you are looking for is not listed or you want to participate with your research - [let us know](#).  
 In case you successfully applied your solution to one of the implementations listed below, tell us about it - so we can refer to your work.

## Categories

Showing 16 items

Benchmarks	Fault Tolerance	HPC	Place&Route	Reconfiguration	DSP Performance	Power Consumption
<a href="#">converted MCMC (.net)</a>			<a href="#">ASII</a>			
<a href="#">Audio Filters</a>				<a href="#">DPR-Framework for PDL Language</a>		
<a href="#">MIPS compatible CPU</a>	<a href="#">Fault-tolerant CPU</a>			<a href="#">Rapid Customization of Soft Core Processors</a>		
<a href="#">Soft Core Processors</a>				<a href="#">DRAFT NoC</a>		
<a href="#">DRAFT Communication Ports</a>						
<a href="#">highly modified MCMC</a>			<a href="#">PipePlace</a>			
<a href="#">MB-Lite-CPU</a>						
<a href="#">AES implementation</a>						<a href="#">AES implementation (Catapult) for LEON3 with eFPGA</a>
<a href="#">AES encryption module</a>					<a href="#">BCDL: A High Speed Balanced DPL for FPGA with Global Precharge and no Early Evaluation</a>	
<a href="#">Matrices</a>		<a href="#">FPGA vs. GPU for Sparse Matrix Vector Multiply</a>				
<a href="#">Cd2dat &amp; wireless</a>				<a href="#">Pareto Efficient Design for Reconfigurable Streaming Applications on CPU/FPGAs</a>		
<a href="#">Scalar decoder (code-length 1638)</a>					<a href="#">Vector decoder for QC-LDPC codes</a>	
<a href="#">GridRT-Benchmark</a>		<a href="#">GridRT</a>				
<a href="#">FIR-Filter</a>						
<a href="#">Restricted Boltzman machine</a>		<a href="#">A Multi-FPGA Architecture for Stochastic Restricted Boltzmann Machines</a>				
<a href="#">Bitstreams for compression</a>					<a href="#">Hardware decompression modules for FPGA bitstreams</a>	

Showing 16 items

**Benchmarks** (vertical label on the left side of the table)

**Projects** (vertical label on the right side of the table)

# Structure (2) – Benchmark

## Benchmark-suite

Besucheransicht

Welcome  
Overview  
Benchmarks  
Projects  
Contact

[Benchmarks >](#)

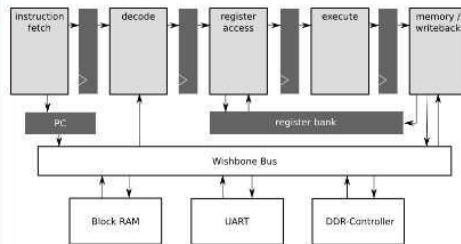
### MIPS compatible CPU

posted May 28, 2010 3:23 PM by Andreas Beyer [ updated a minute ago ]

The final version of this CPU is quite similar to the MIPS R2000/R3000 system and is supplied with

- 32 bit address and data width, byte addressed, big or little endian
- a five stage pipeline
- a hardware multiplication and division unit
- interrupt and exception handling as suggested by the MIPS specification
- a Wishbone bus interface

A graphical representation of the CPU system is shown below.



The CPU and any peripherals can be controlled using the standard GCC MIPS cross compiler. The implemented MIPS CPU is a three operand architecture with 32 general purpose 32 bit registers and several co-processor registers used for interrupt and exception handling. It neither has a cache nor a MMU at this time. The lack of separate instruction and data caches require that both, instruction fetch and memory access share the same bus. The pipelined execution of commands is separated into instruction fetch (IF), instruction decode (DE), register access (RA), execute (EX) and memory / writeback (MW). IF-stage fetches the next instruction from the bus and DE-stage decodes it and creates the select signals for the multiplexers in the later stages. RA-stage loads the contents of the registers according to these signals. EX-stage performs the requested logic combination of the previously loaded register contents. MW-stage redirects the result back into one of the registers or performs a bus access. The Wishbone bus currently grants access to an internal BlockRAM, a UART controller and a DDR-SDRAM controller.

This text describes the non-hardened version of the CPU as it is in the attached archive. See [Overview](#) to find the related "fault-tolerant CPU" project.

Elements	utilization
flip-flops	2254
look-up tables	3502
DSPs	4

#### Attachments (1)

hw\_nFT.tar.gz - on May 28, 2010 4:21 PM by Andreas Beyer (version 1)  
27k [Download](#)

Purpose

Sources

# Structure (3) – Project

## Benchmark-suite

Besucheransicht

Welcome  
Overview  
Benchmarks  
Projects  
Contact

Projects >

### Fault-tolerant CPU

posted May 28, 2010 3:24 PM by Andreas Beyer [ updated a minute ago ]

The usage of SRAM based reconfigurable hardware has become an important instrument for particle physics detector readout or space applications. However, in radiative environments, the accurate operation of SRAM based field programmable hardware cannot be guaranteed. Radiation can alter configuration and state of these devices and thus change their behavior. Common approaches use triple modular redundancy (TMR) in combination with majority voters to compensate radiation induced errors. However, this comes with a large area overhead. This work proposes a fault tolerant softcore CPU for FPGAs with reduced area overhead by using mostly double modular redundant logic to detect errors in combination with continuous FPGA configuration scrubbing to correct them. The effectiveness of the applied methods could be verified with both error emulation and particle beam experiments.

#### APPROACH:

The aim of this work is to apply fault tolerance techniques to an SRAM based reconfigurable FPGAs not by blindly triplicating all logic, but by exploiting mitigation techniques on different layers to reach the goal. The competing architecture (COTS) SRAM architecture is chosen not because of any improved radiation tolerance features compared to the competing architectures, but due to its flexibility and price. According to the requested application, a radiation hardened SRAM FPGA, a flash based FPGA or even an antifuse solution may give a significantly better radiation tolerance. The aim is rather to find out what is possible with these cheap and well spread SRAM based FPGAs, which are only present in current high energy physics experiments. Obtaining increased radiation tolerance with a fair effort would allow these FPGAs to be placed closer to the experiment saving costs and simplifying detector readout. Different SEL mitigation techniques are used in this work to exemplarily develop a radiation tolerant softcore CPU for SRAM based FPGAs, usable for controlling tasks. This Figure 3: System layout. A small flash FPGA is used to load bitfiles from a flash memory and perform scrubbing on the SRAM FPGA. The SRAM FPGA is the main part of the system holding the fault tolerant softcore CPU and its peripheral controllers. CPU is designed to be part of a radiation tolerant multilayer system spanning all layers of modern FPGA based embedded systems. This includes the FPGA configuration layer and the actual FPGA system architecture with CPU and peripherals. An operating system contributes an abstraction layer for the hardware and allows applications to use it. A skeleton of the layer structure is shown in figure 2 (in the attached paper). All these layers will give their share to an overall radiation hardened system by applying different error and radiation effect mitigation techniques on each level. As shown above, the FPGA configuration is extended with redundancy to detect or even correct single event effects. An operating system could implement redundant execution of codes or context saves and software. A radiation tolerant softcore CPU for SRAM based FPGAs as part of a multilayer system covers only the lowest abstraction layers. Operating system or software aspects have no effect as long as a base for executing single commands has not yet been created. The mitigation techniques used in this work therefore refer to the lowest two layers: the FPGA configuration and the user logic implementation. The main aspect of this work is the exploitation of the combination of both, scrubbing and redundant logic. Scrubbing guarantees that within a short period of time a configuration error will be corrected. With this knowledge, it is sufficient to use double modular redundancy to detect errors and wait for them to be corrected by the scrubbing mechanism.

(full text in attached paper.)

#### Synthesis:

Elements absolute (relative)	non-hardened version	fault-tolerant version
flip-flops	2254 (100%)	3605 (160%)
look-up tables	3502 (100%)	11687 (334%)
DSPs	4 (100%)	8 (200%)

#### related publications:

- Jano Gebelein, Heiko Engel, Udo Kobschull:  
*FPGA Fault Tolerance in Particle Physics Experiments*  
DPG Conference, Bonn, Germany, HK 10.2, 15. March 2010  
citation: <BibTeX>
- Heiko Engel, Jano Gebelein, Udo Kobschull:  
*Development of a Radiation Tolerant Softcore CPU for SRAM based FPGAs*  
DPG Conference, Bonn, Germany, HK 10.3, 15. March 2010  
citation: <BibTeX>
- Jano Gebelein, Heiko Engel, Udo Kobschull:  
*An approach to system-wide fault tolerance for FPGAs*  
Short Paper, FPL 2009 Conference, Prague, Czech Republic, session tba, 31. August - 2. September 2009  
citation: <BibTeX>

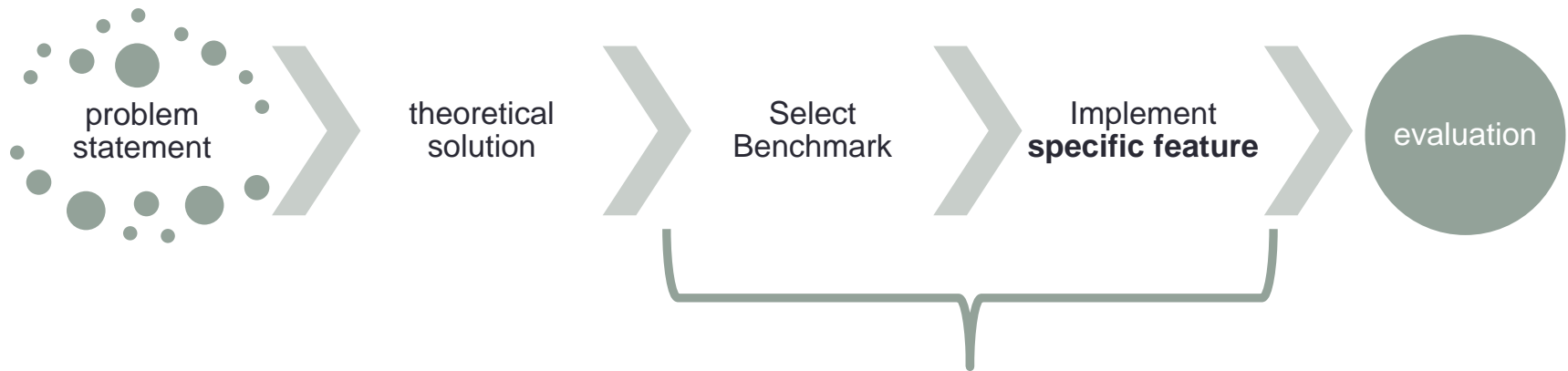
#### Attachments (1)

paper.pdf - on May 28, 2010 4:47 PM by Andreas Beyer (version 1)  
224k [View](#) [Download](#)

Approach  
Metrics  
related publications

# “cheaper, better, faster”

- Benefits of using benchmarks during development

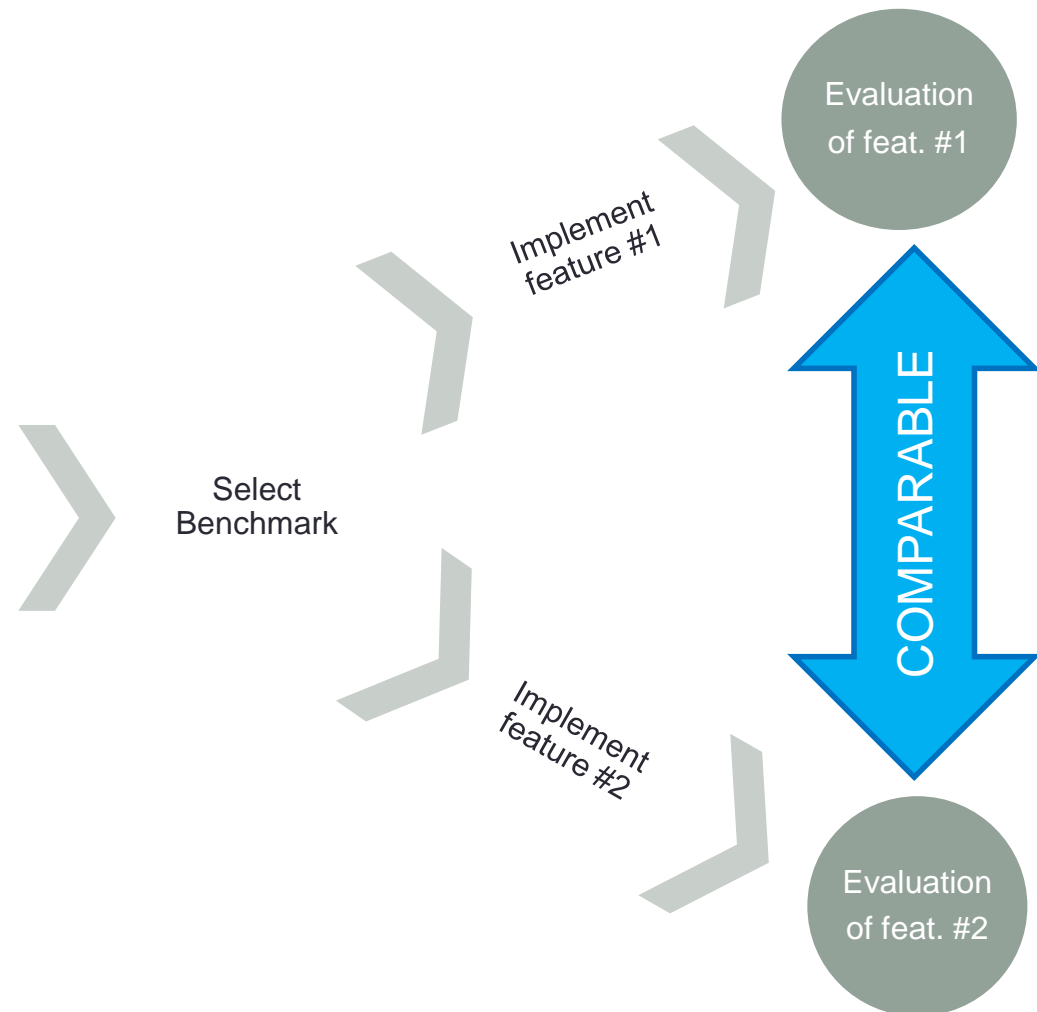


Implementation efforts reduced:

- Start with existing sources from Benchmark
- Implement specific feature or apply mechanism

# “comparing one's performance metrics”

- two groups with the same field of operation
- they choose the same benchmark to start from
- implement their feature or apply mechanism
- results are comparable due to using same basis



# Inspected publications

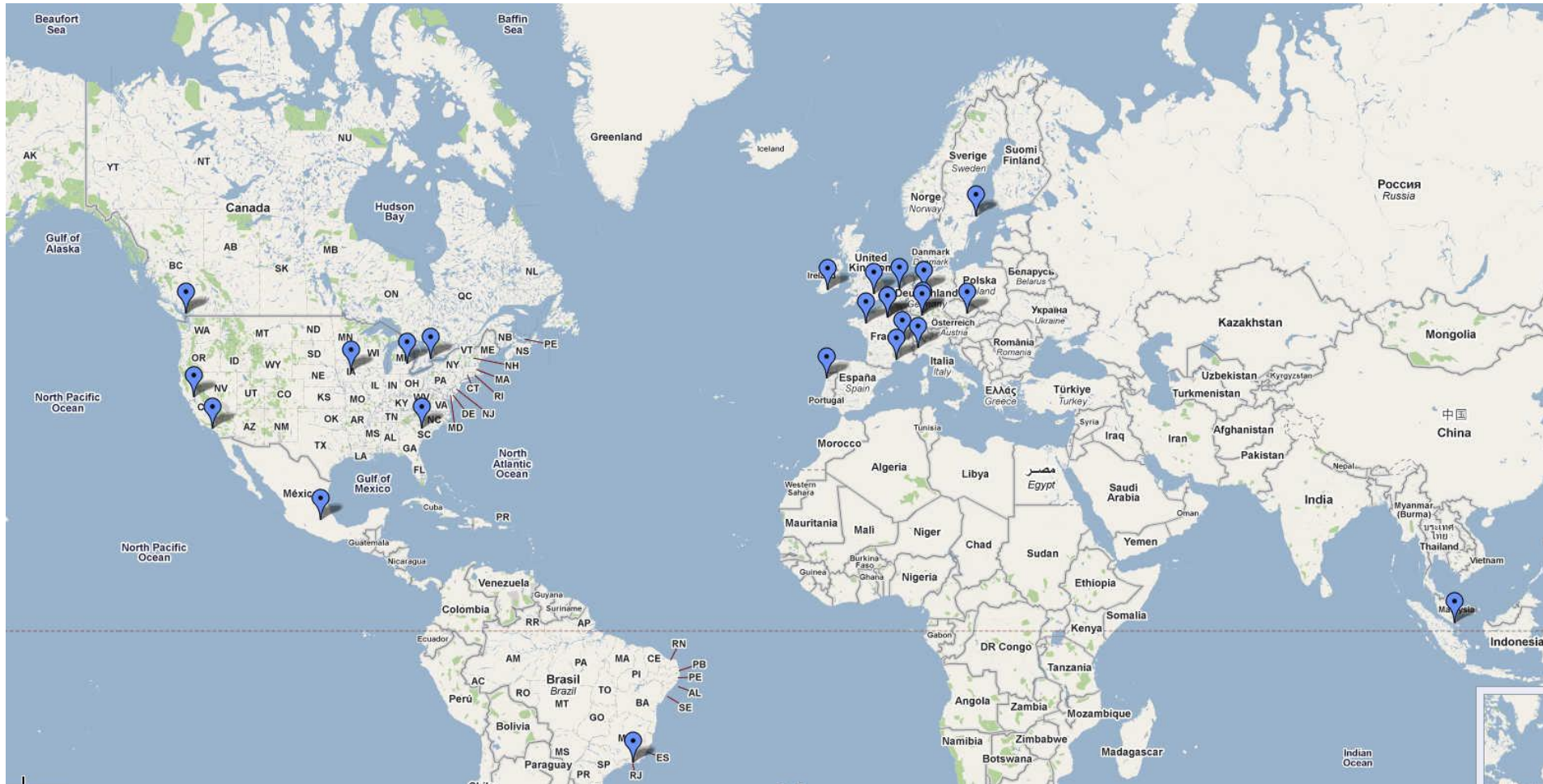
- All papers of following conferences (~2000)
  - DAC 2008 & 2009
  - Date 2009 & 2010
  - DSD 2010
  - FPL 2008 & 2009
  - FPT 2009
  - ReConFig 2009
  - SOCC 2009
- Criteria
  - Quantifiable results / metrics & measurements
  - Aims for FPGA
  - Preferably evaluated with own benchmarks
  - willing to share sources

# Areas of operation

- Relevant for FPL-BS
  - Fault Tolerance
  - HPC
    - FPGA and GPU
  - Synthesis
  - Place&Route
  - Reconfiguration
    - Tools and Application
  - DSP Performance
  - Power Consumption
  - ...
- Unnoticed areas
  - Spiking neural networks
  - Random number generator
  - Motion detection
  - Altered FPGA hardware



# Contributors



200 requests → 50 responses → 35 Dialogs → currently 25 candidates → 17 contributions



# How to contribute

- Procedure to publish a new “Benchmark”
  1. contact [benchmark@fpl.org](mailto:benchmark@fpl.org)
  2. provide your contribution
    - describe purpose or problem statement
    - assemble sources
    - give instructions how to compile
  3. provide the corresponding “Project”
  4. maintain your benchmark
    - Updates
    - Answer questions from other users (forum)

# How to contribute

- Procedure to publish your results as “Project”
  1. contact [benchmark@fpl.org](mailto:benchmark@fpl.org)
  2. provide your contribution
    - describe your approach
    - present metrics & measurements
    - list related publications
    - poste contact information / affiliation
    - tell us the desired category
      - where you fit in
      - to create
  3. maintain your Project
    - Updates
    - Answer questions from other users (forum)

# Future work (planned services)

- Automated evaluations
  - Cluster and web interface already set up
  - sources will be “nightly build”
    - for various platforms
    - using different compilers / P&R tools
- RSS-Feed
  - Errata / Updates / News

# FPL-Benchmark Suite

Web-link [benchmarksuite.beyer-andreas.net](http://benchmarksuite.beyer-andreas.net)  
( → will become [www.fpl.org/benchmark](http://www.fpl.org/benchmark) soon)

Thank your for your attention