



# Back-End Issues in Hardware/Software Compilation

Andreas Koch

Embedded Systems and Applications Group  
Department of Computer Science  
Technical University Darmstadt, Germany

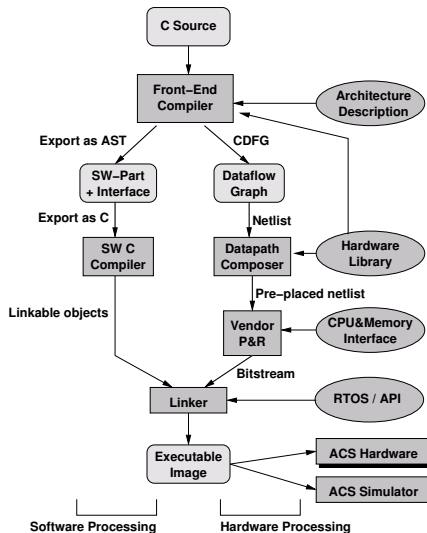


- 1 Project Overview
- 2 Target Architecture
- 3 (Re-)Configuration Management
- 4 Floorplanning
- 5 Module Generation
- 6 Integrating external IP blocks

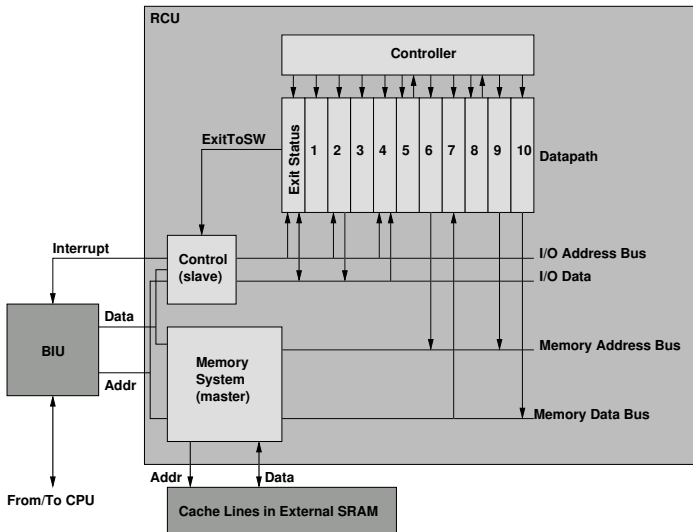


# Project COMRADE

- Compiler for adaptive computer
  - Processor and reconfigurable fabric
- Spiritual successor to NIMBLE Compiler
- Full ANSI C as input language
- Profile-based HW/SW partitioning
- ILP and speculative execution
- Fine granularity of HW/SW execution
- Optimized reconfigurations

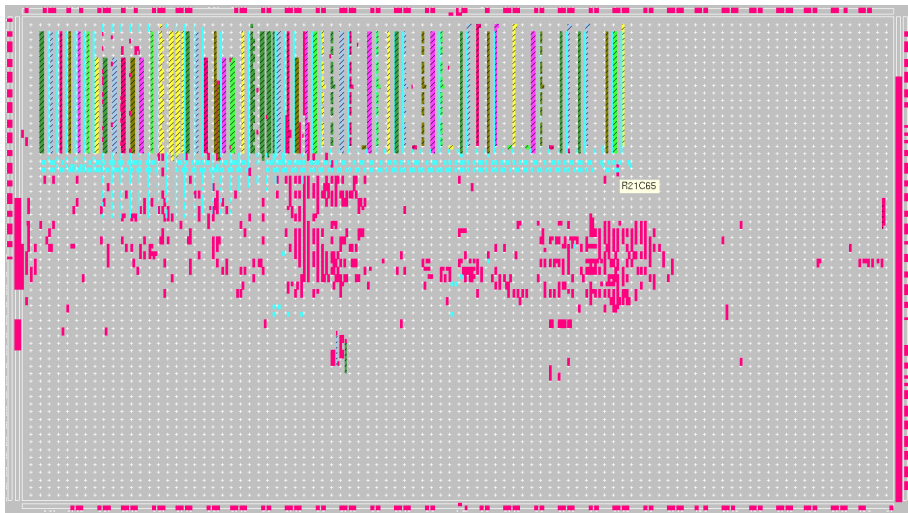


# Current Target Architecture





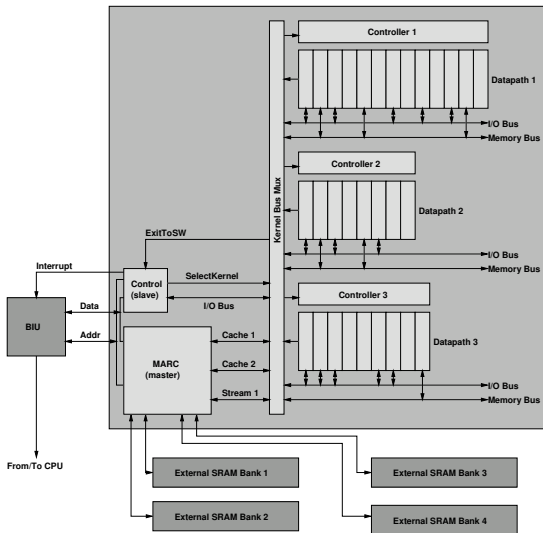
# On-Chip Layout: CFAR Kernel





- Strictly linear data path layout
  - Does not scale with
    - device geometry
    - data path complexity
  - Long delays
- One hardware kernel per configuration
- Single port cached memory interface

# New Target Architecture





- Constrained two-dimensional placement
  - Data paths can span multiple rows
  - Regular structure within data path (aligned busses)
- Multiple hardware kernels (data paths) per configuration
  - Rapid kernel **switching** instead of reconfigurations
- Flexibly configurable memory interface
  - Multi-port, caching and streaming
  
- **How to achieve this goal?**



# Configuration Management

- Map kernels to configurations
  - Possibly duplicating kernels
  - Reducing number of reconfigurations
- Two approaches realized
  - 1 Fast heuristics (independent of dynamic behavior)
    - Suitable for use in partitioning cost function
  - 2 Exact solution (analyzes execution trace)
- Example: Wavelet image compression

	Configurations	Reconfigurations	Computed in
Before	8	5381	-
Optimal	4	4	2.59s
Heuristic	4	5	<0.01s

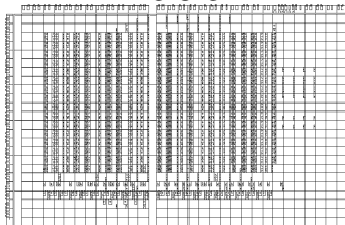


- Initial prototype CIAP (Clustering and Placement)
- Strategy
  - 1 Clustering of suitable hardware operators
    - Shared interconnections
    - Matching topologies (height, bus pitch)
  - 2 Simulated annealing based timing-driven placement
    - Moves at operator and cluster scopes

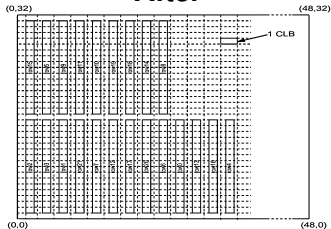


# Example: Wavelet Kernel

Before



After



**Discovered two clusters,  
delay reduced by  $\approx 20\%$**

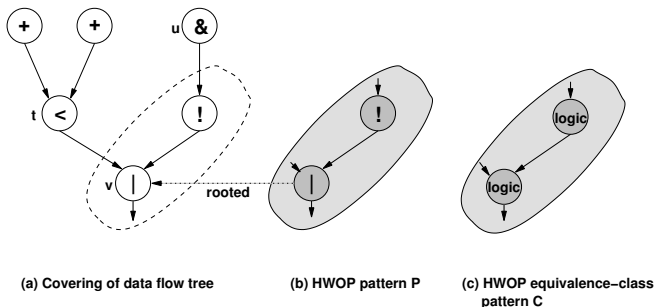


- Procedural module generators
  - GLACE library
  - Primitive functions: Add, Mult, Div, Logic, . . .
- Works very well in general
- Discovered two limitations
  - Generating fine-grained logic inefficient
    - C semantics insufficient (unwieldy bit manipulation)
    - No logic optimization
  - New generators awkward to develop
    - Manual characterization of area/delay characteristics
    - Difficult for complex/flexible generators

# Current Logic Generator

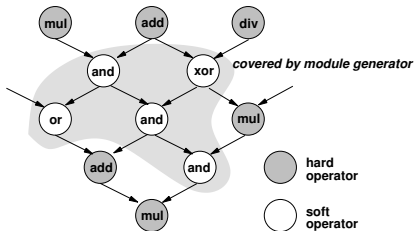
- Accepts arbitrary-width logic function of up to four inputs  

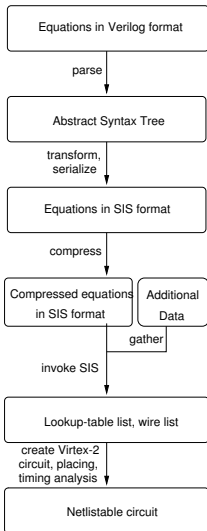
```
int y, a, b, c, d;  
y = ((a&0xff) <<24) | ((b&0xff) <<16) | ((c&0xff) <<8) | (d&0xff);
```
- Generates pre-placed regular hardware operator
- Called from pattern-based tree-covering algorithm



# New Logic Generator

- Accepts **multiple** arbitrary logic functions
- Described using Verilog language capabilities
  - Busses: Splitting and composition  
 $y[31:0]=\{a[7:0],b[7:0],c[7:0],d[7:0]\};$
  - Variable shifts
  - Bit Permutations
  - Constant Bits
- Applied to DFG at the block level during compilation





- Integrates UCB SIS for
  - logic optimization
  - technology mapping
- Problem: SIS not bus-oriented
  - Requires pre-/post-processing
- Regularity analysis of expressions
  - Reduce problem complexity
- Replication of results
- Timing estimation
- Compact regular placement



- Based on BYU JHDL
  - Structural hardware description
  - Supported by all Java constructs (inheritance, iterators, ...)

## Extended with

- Automatic analysis of timing and topology characteristics
- Replaces explicit expressions

```
T.opcy+(T.n-1)/2*T.net+(T.n-3)/2*T.byp+T.sum+T.reg*(T.ick+T.cko)
```

- .. with calls

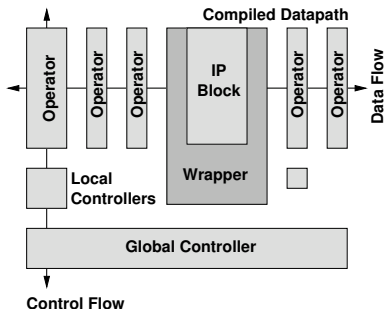
```
targetVirtex2.getDelay(addCell)
```

- Supports
  - Target device specific features **MUXF<sub>x</sub>**, **SOP<sub>x</sub>**, **SRL<sub>x</sub>**, ...
  - Reads timing directly from vendor tools



# Integration of External Hardware Functions

- Compiler performance will not be sufficient for all functions
- Allow integration of external IP blocks
  - Manually optimized
  - Similar to calling assembled from high-level language
- Integration in C
  - Idiomatic programming style
  - Description of IP characteristics

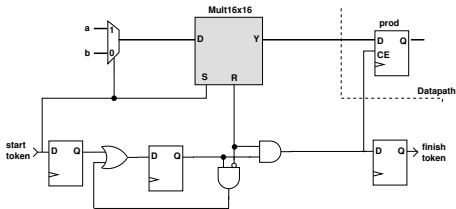




# Controller Integration

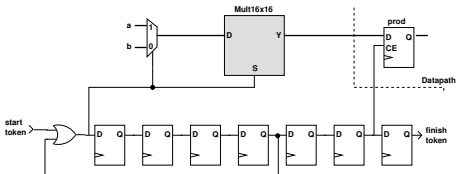
- Generate local controllers in wrapper
- Exchange data with compiled data path
- Logical-to-physical interface conversion
  - Example: Sequential loading of operands
- Startup/shutdown IP processing
  - Generate IP-specific signal sequences
  - Allow variable-latency computations
  - Present simple START/DONE protocol to central controller
- Pipelined execution
  - Parallel threads of control
  - Stopping/draining the pipeline
- Goals
  - Concise description
  - Quickly and efficiently synthesizable

## Variable-latency Multiplier



```
POSEDGE (S 1) (D[15:0] a[15:0]);
POSEDGE (S 0) (D[15:0] b[15:0]);
CONTINUE (R 1);
POSEDGE (Y[31:0] prod[31:0]);
```

## Pipelined Multiplier



```
START;
POSEDGE (S 1) (D[15:0] a[15:0]);
POSEDGE (S 0) (D[15:0] b[15:0]);
POSEDGE; POSEDGE;
RESTART;
POSEDGE; POSEDGE;
POSEDGE (Y[31:0] prod[31:0]);
```

# Example: Wrapping Xilinx LogiCore 16-Point FFT

```
; initialize
POSEDGE (CE 1) (SCALE_MODE 0) (FWD_INV 1) (START 1)
POSEDGE (START 0)
; start of steady-state
START
; wait for acceptance of first FFT block
CONTINUE (MODE_CE 1)
; write 16 time domain samples
POSEDGE *16 (DI_R[15:0] time_r[15:0]) (DI_I[15:0] time_i[15:0])
; fork control flow for pipelining
RESTART
; wait for transformed data
CONTINUE (DONE 1)
; read 16 frequency domain samples
POSEDGE *16 (XK_R[15:0] freq_r[15:0]) (XK_I[15:0] freq_i[15:0])
```

Synthesis Style	Virtex-II Slices	Max. Clock [MHz]
Direct FFs	25	467
Counter	13	248
SRL16	8	243



- Tool flow overview
- Target architecture
- Reconfiguration management
- Floorplanning
- Module generation
- IP integration

**TODO: Finally tie everything together!**