Run-Time Reconfiguration of Hardware

João Canas Ferreira

CERN, 25-26 February 2010

Table of contents

1 Introduction to reconfigurable computing

2 Run-time reconfiguration of hardware
- General aspects
- Creating configurations
- Potential advantages

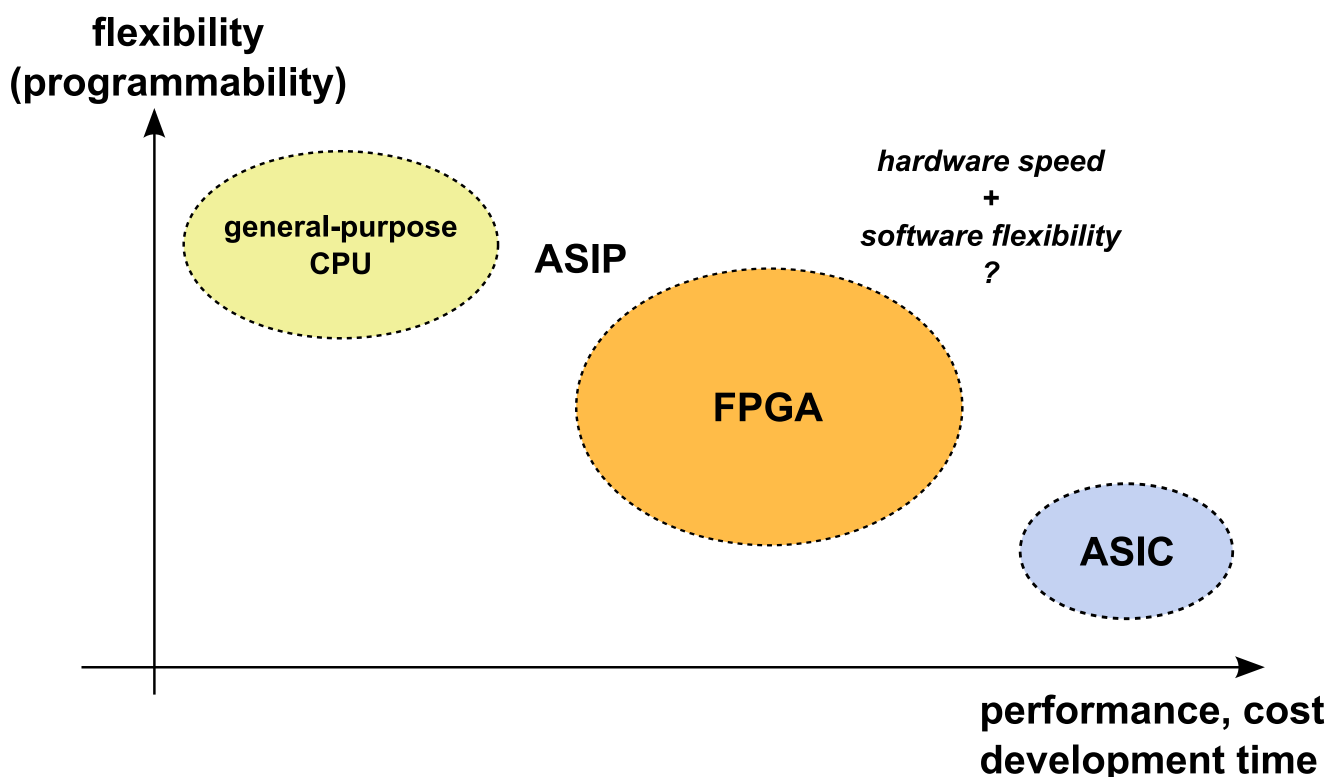
3 Some devices with support for RTR

4 Flexible bitstream generation
- General approach
- Pragmatic aspects
- Bitstream assembly
- Bitstream assembly at run-time

5 Opportunities

1. Introduction to reconfigurable computing
2. Run-time reconfiguration of hardware
   - General aspects
   - Creating configurations
   - Potential advantages
3. Some devices with support for RTR
4. Flexible bitstream generation
   - General approach
   - Pragmatic aspects
   - Bitstream assembly
   - Bitstream assembly at run-time
5. Opportunities

Three ways to perform computations



(No talk on reconfigurable computing is complete without this graphic!)

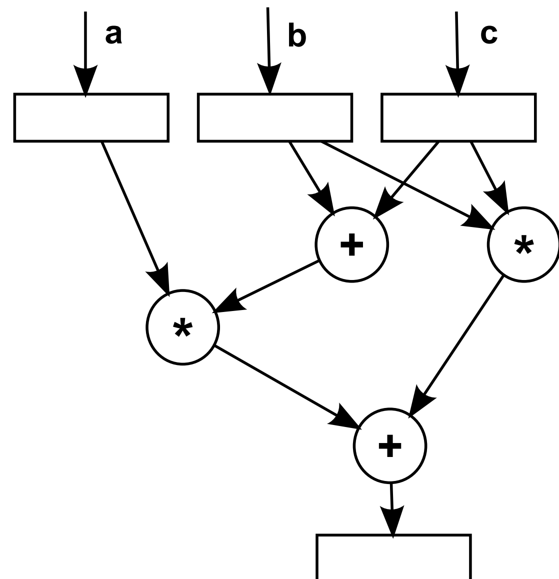
Reconfigurable computing

Reconfigurable computing (RC) vs. conventional (CPU) computing

- Reconfigurable hardware infrastructure instead of CPU
- Computation performed by a circuit rather than by executing instructions

res = a * (b + c) + b*c

```
lw     $t1, 0($s0)      # load word
lw     $t2, 4($s0)
lw     $t3, 8($s0)
add    $t6, $t2, $t3    #  b + c
mult   $t6, $t1         # a * ( )
mflo   $t1              # keep 32 msb
mult   $t2, $t3         # b * c
mflo   $t2
add    $t1, $t1, $t2    # final addition
sw     $t1, 0($s1)      # store res
```



Advantages of RC

Key advantage 1

Naturally concurrent computation

- “Natural” functioning mode of hardware
- In the absence of resource constraints, only dependencies restrict operation

Key advantage 2

Circuit can be tailored precisely to the requirements of application

- Bit-width optimization
- Partial evaluation
  - Example: embedding constants in the circuit
  - May improve latency and power consumption
- Memory organization

Reconfigurable fabric architecture

Two large classes of reconfigurable fabrics:

- Fine-grained:
  Allow for the manipulation of data at the bit-level, both for processing and for communication.
  Most commercial FPGA fabrics fall in this category, although resources are usually grouped in larger blocks (CLB: Configurable Logic Block).

  Advantages: data of any size can be processed without wasting reconfigurable resources; more versatile.
- Coarse-grained:
  Manipulate groups of bits via complex functional units (ALUs or processing elements—PEs) and interconnect networks organized at word level.

  Advantages: speed (for large, complex calculations with uniform data sizes); smaller amount of configuration data.

Many approaches to system organization

➠ RC hardware has appeared (or been proposed) in many sizes and shapes, targeting different market segments:

- only FPGAs
  - custom computing machines: PAM, Splash
  - accelerators (Teramac)
- FPGA with CPU(s)
  - embedded CPU core / CPU soft core
- SOCs with reconfigurable IP cores
- CPU with reconfigurable units
- supercomputers (Cray, Silicon Graphics)

➠ “Pure” FPGAs have evolved to reconfigurable platforms with dedicated blocks for:

- memory
- multipliers, DSP functions
- clock generation
- input/output (high-speed serial communications)

The "curse" of programmability

The programmability issue

How do we go from algorithms to circuits?

- Hardware designer: use hardware description languages
  - (+) Efficient at circuit design
  - (-) Lack integration with software, exposes hardware
- Automatic translation from standard languages (C, Matlab)
  - (+) Leverages existing code
  - (-) How to express/extract parallelism? Data formats?
  - (-) Quality of the final results? (Design space exploration)
- Use of new languages (e.g., Handel-C)
  - (+) Reduces the semantic gap (e.g., timing, pipelining, parallelism)
  - (-) Languages are *new* and embody a different computational model

Appropriateness

Are we using the appropriate algorithms for a given task in RC?

1. Introduction to reconfigurable computing
2. Run-time reconfiguration of hardware
   - General aspects
   - Creating configurations
   - Potential advantages
3. Some devices with support for RTR
4. Flexible bitstream generation
   - General approach
   - Pragmatic aspects
   - Bitstream assembly
   - Bitstream assembly at run-time
5. Opportunities

Implementation strategies for reconfigurable systems

Reconfigurable systems can follow two main strategies:

- Configure-once: (ASIC-like operation)
  - Single, system-wide configuration
  - FPGAs configured prior to operation

  Variant: For some applications, input data remains constant for hours or days: the bitstream is regenerated occasionally.

  Example: acceleration of the SNORT packet filter by translating regular expressions into hardware [Hutchings et al., 2002].
- Run-time reconfiguration:
  Application consists of *multiple* configurations for each device.

  During normal execution, the FPGA is potentially reconfigured many times (configuration steps).

General implementation strategies for RTR

Classification of RTR-based systems according to scope of reconfiguration:

- Global RTR:
  All resources are reconfigured in each configuration step.

  Example: Back-propagation training of artificial neural network divided in 3 mutually exclusive phases: idle circuitry in each phase is eliminated [Eldredge and Hutchings, 1996].
- Local RTR: (partial reconfiguration)
  A subset of the resources is reconfigured in each step.

  ➠ Part of a single FPGA (or a whole FPGA in a multi-FPGA system)

  ➠ Ideally, the operation of the remainder of the system is not affected.

  ➠ Use of hardware resources adapts to run-time profile of application.

  ➠ Several tasks may be *independently* supported in hardware at the same time (multiple hardware modules).

  ➠ Shorter reconfiguration time (individual hardware modules).

1 Introduction to reconfigurable computing

2 Run-time reconfiguration of hardware
- General aspects
- Creating configurations
- Potential advantages

3 Some devices with support for RTR

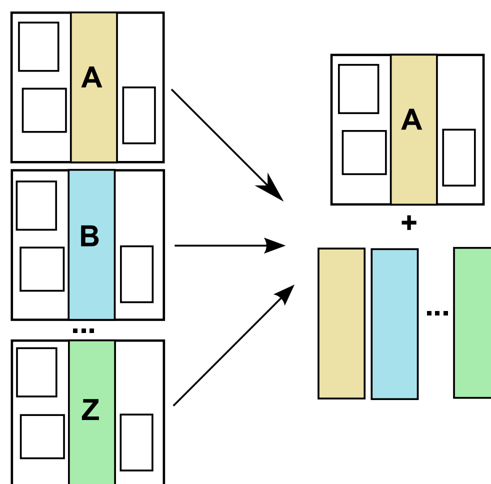
4 Flexible bitstream generation
- General approach
- Pragmatic aspects
- Bitstream assembly
- Bitstream assembly at run-time

5 Opportunities

Creation of configuration data: basic scenario

Default scenario: *local RTR* for a single-context, fine-grained FPGA.

➠ In the simplest development scenarios, configuration data is created at *design time*, together with the rest of system.



If sequence of configuration steps is known: partial *difference* bitstreams can be used to take the hardware from one configuration to another.

Characteristics of the basic scenario

Main characteristics of the basic scenario:

- Regular development flow and tools

  Just a modification of the bitstream generation procedure to create *partial bitstreams*.

- Full design for each configuration

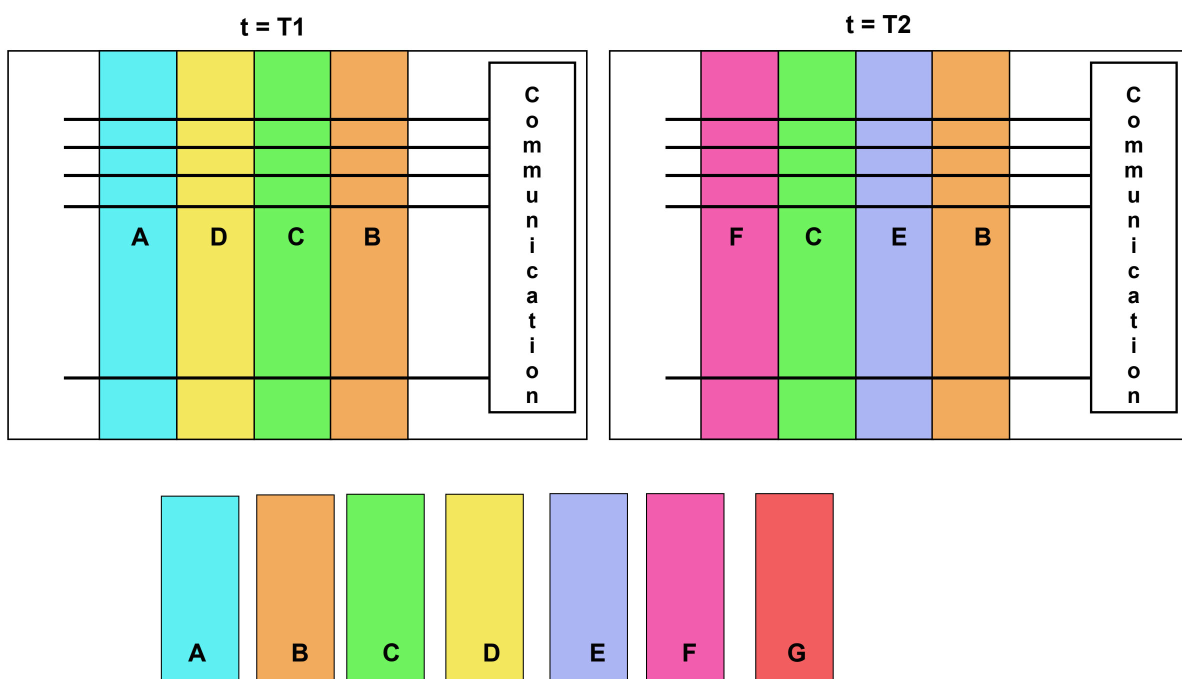
  Advantage: Safe—each configuration can be validated independently, including timing restrictions

  Disadvantage: Time-consuming, with a great amount of redundant work.

  Disadvantage: Any change of the common sections requires recreating all the designs.

Slot-based organization

➠ Exchangeable modules may be placed in one of various slots.



Characteristics of slot-based organization

- More flexible than basic scenario
  - At any given time, a different combination of modules may be active.
  - Modules for the slots may be created after the initial design.
- Mostly regular design flow
  - Partial bitstreams must be created before use:

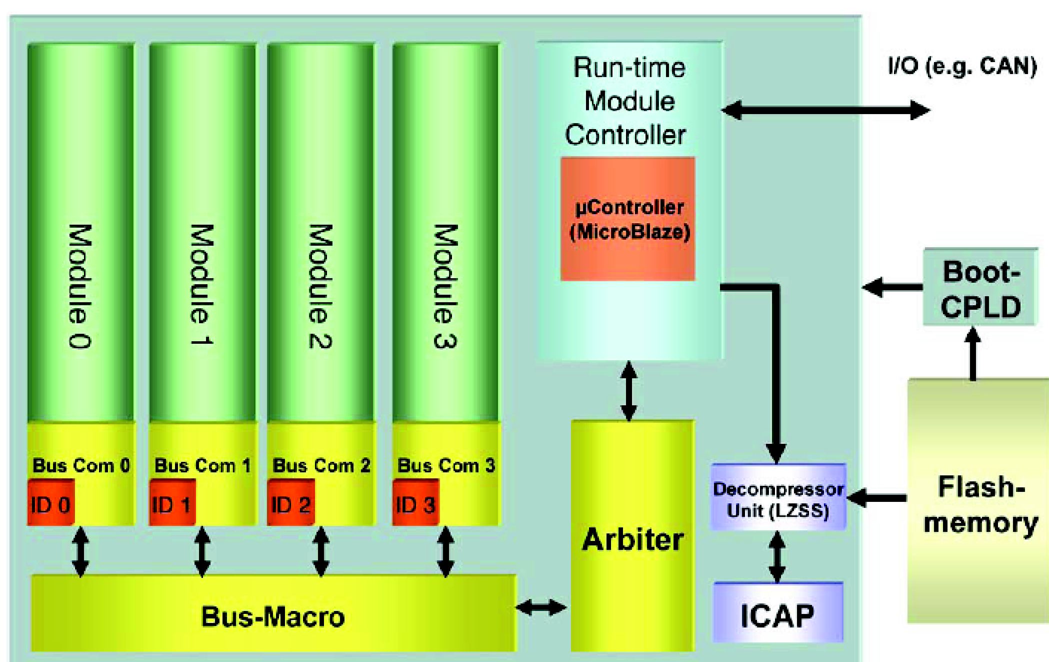
    (i) One version for each possible slot

    (ii) Single version relocated at load time by software or hardware
- Communication infrastructure:
  Connections among slots, and between slots and the "fixed" circuitry.
- Predefined slot size constrains design
- Use of some non-homogeneous resources is restricted or impossible

  Example: Block RAMs or dedicated DSP blocks.

Example: 1D slot-based system

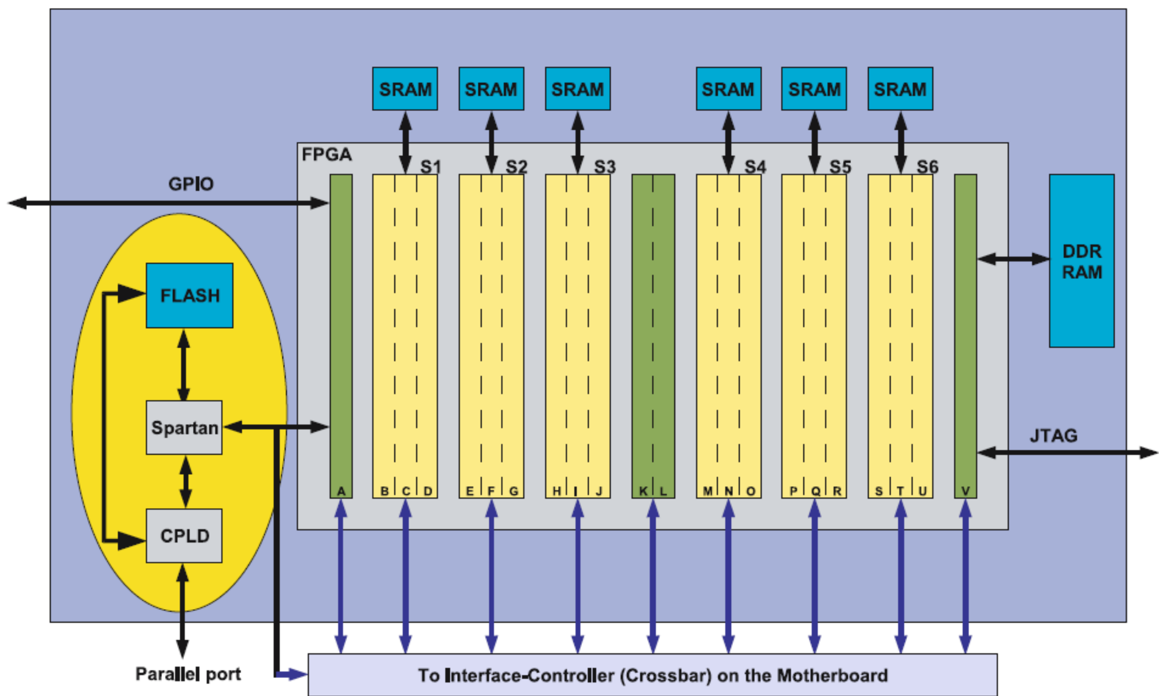
Used in automotive systems: run-time system assigns slots, may save state.



Source: [Becker et al., 2007]

Example: Erlangen Slot Machine

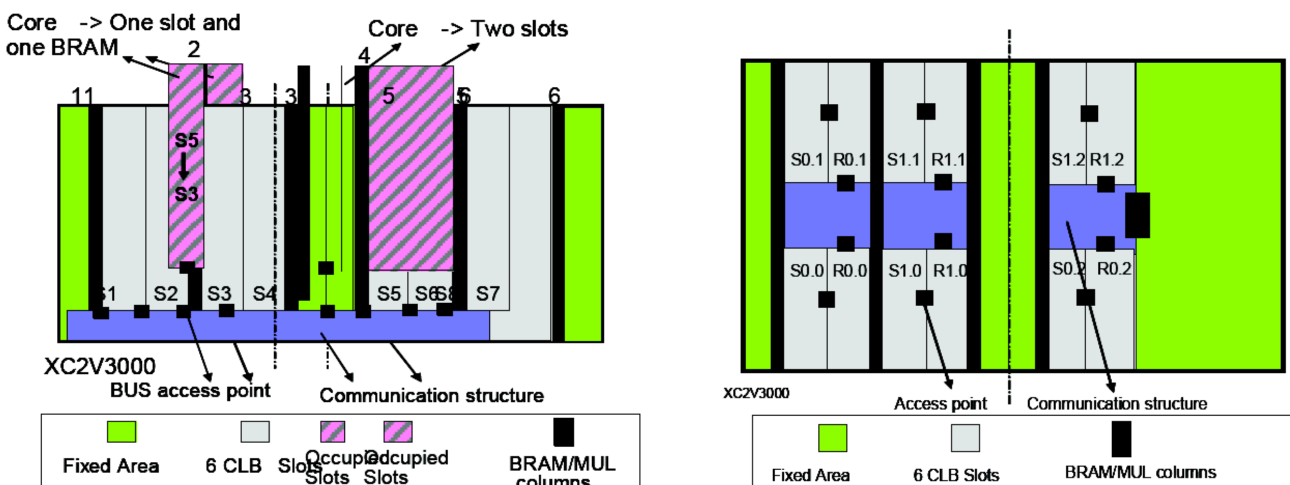
BabyBoard: (4-CLB-wide) micro-slots, macro slots, access to external SRAM.



Source: [Majer et al., 2007]

Flexible slot arrangement

Characteristics: 1D and 2D slot arrangements, multi-slot modules, and reconfigurable routing modules for configuration.



Source: [Krasteva et al., 2007]

1. Introduction to reconfigurable computing
2. Run-time reconfiguration of hardware
   - General aspects
   - Creating configurations
   - Potential advantages
3. Some devices with support for RTR
4. Flexible bitstream generation
   - General approach
   - Pragmatic aspects
   - Bitstream assembly
   - Bitstream assembly at run-time
5. Opportunities

Increased hardware utilization

- More specialized circuits require less hardware and may run faster.
  - T: total operation time
    $T = T_e + T_c$ T_e: execution time; T_c: configuration time
    $f = T_c/T_e$: relative configuration time
  - A: total area required for the implementation
- Functional density D measures the *computational throughput* (operations/s) of unit hardware resources [Wirthlin and Hutchings, 1998]:

$$D = \frac{1}{C} = \frac{1}{AT} \qquad D_{\max} = \frac{1}{AT_e}$$

- Improvement I in functional density of RTR circuit D_r over statically configured alternative D_s:

$$I = \frac{D_r - D_s}{D_s} = \frac{D_r}{D_s} - 1 \qquad I_{\max} = \frac{D_{\max}}{D_s} - 1$$

- To increase functional density over D_s: $f \leq I_{\max}$

Using functional density

Quick estimate of the suitability of RTR:

1. estimate area and time (T_e) of RTR version
2. calculate I_{max}
3. estimate T_c
4. find f and check condition $f \leq I_{max}$

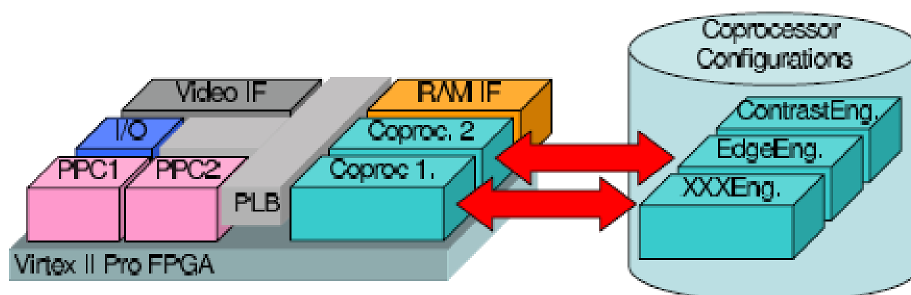
Application	I_{max}	f	I_{actual}	Break-Even
$RRANN_{60}$	4.28	9.9	−0.52	138 neurons
$RRANN\text{-}II_{global}$	1.68	1.62	.014	58 neurons
$RRANN\text{-}II_{partial}$	1.68	.78	.50	28 neurons
$Template_{global}$	.588	.394	.139	89 × 89 image
$Template_{partial}$	.588	.025	.564	18 × 18 image
$Edit_{5000}$	.79	2.64	-0.51	17,700 chars

Source: [Wirthlin and Hutchings, 1998]

Example: Extending hardware support

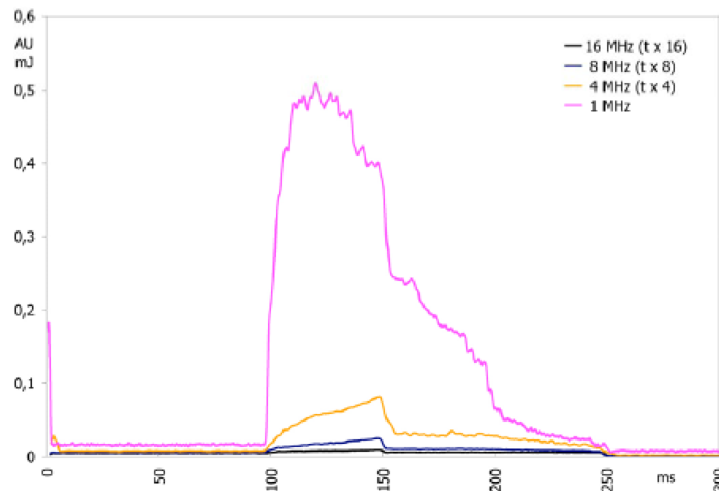
- RTR can be used to provide more hardware support than would fit in a static configuration.
- Example: image processing for driver assistance [Claus et al., 2007]

Environment	Shape Engine	Tunnel Engine	Contrast/ EdgeEng.	Taillight Engine	PPC
Highway	x	x			x
Tunnelentrance		x	x		x
Tunnel inside				x	x



Advantage: Energy consumption

Study for Atmel FPGAs AT40K20 [Lorenz et al., 2004]:



Energy consumption during reconfiguration is dominated by short-circuit and static power consumption (reconfiguration of interconnections), which increase with reconfiguration time.
Reconfiguration should be made at the highest frequency (16 MHz).

Reconfiguration may reduce power consumption

Under some simplifying assumptions (equal dynamic power consumption):

- K: number of reconfigurations
- N: factor by which the RTR circuit is smaller

Condition for reducing energy consumption:

$$E_{\text{static}} + E_{\text{dynamic}} + K \times E_{\text{reconf}} - N \times E_{\text{static}} - E_{\text{dynamic}} < 0$$

Therefore,

$$\frac{K}{N-1} E_{\text{reconfiguration}} < E_{\text{static}}; \qquad \text{typically} \qquad \frac{K}{N-1} \approx 1.4$$

For the Atmel FPGAs considered in the study, this results in:

$$\frac{T_{\text{process}}}{T_{\text{reconf}}} > \frac{1.4 \times P_{\text{reconf}}}{P_{\text{static}}} \approx 16$$

Advantage: Increased flexibility

Increased flexibility afforded by RTR can be used to:

- develop versatile framework field updates [Fong et al., 2003]
- develop sophisticated adaptive systems

➠ SAFES—Secure Architecture For Embedded Systems:

Support for security standards and defence against hardware attacks by using reconfigurable hardware [Gogniat et al., 2008]

➠ Autonomous System-on-a-Chip Adaptation

Uses Bayesian network to choose and activate appropriate filter to mitigate changing RF interference [French et al., 2008].

Interference identification (96 %), correct filter selection (65% plus 16% partial mitigation). Virtex-4 FX100 FPGA.

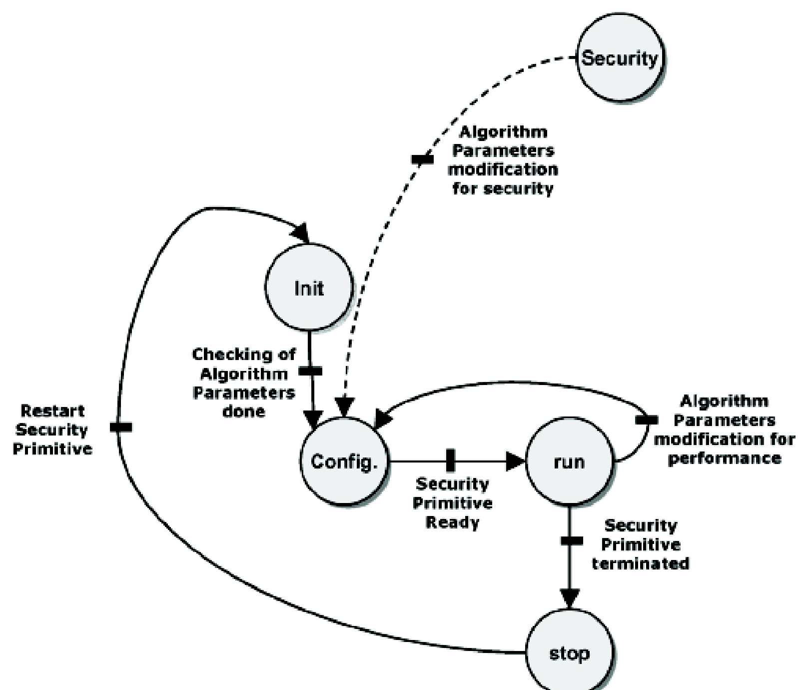
Reaction time is 112 ms (against 3–5 s for human operator).

SAFES

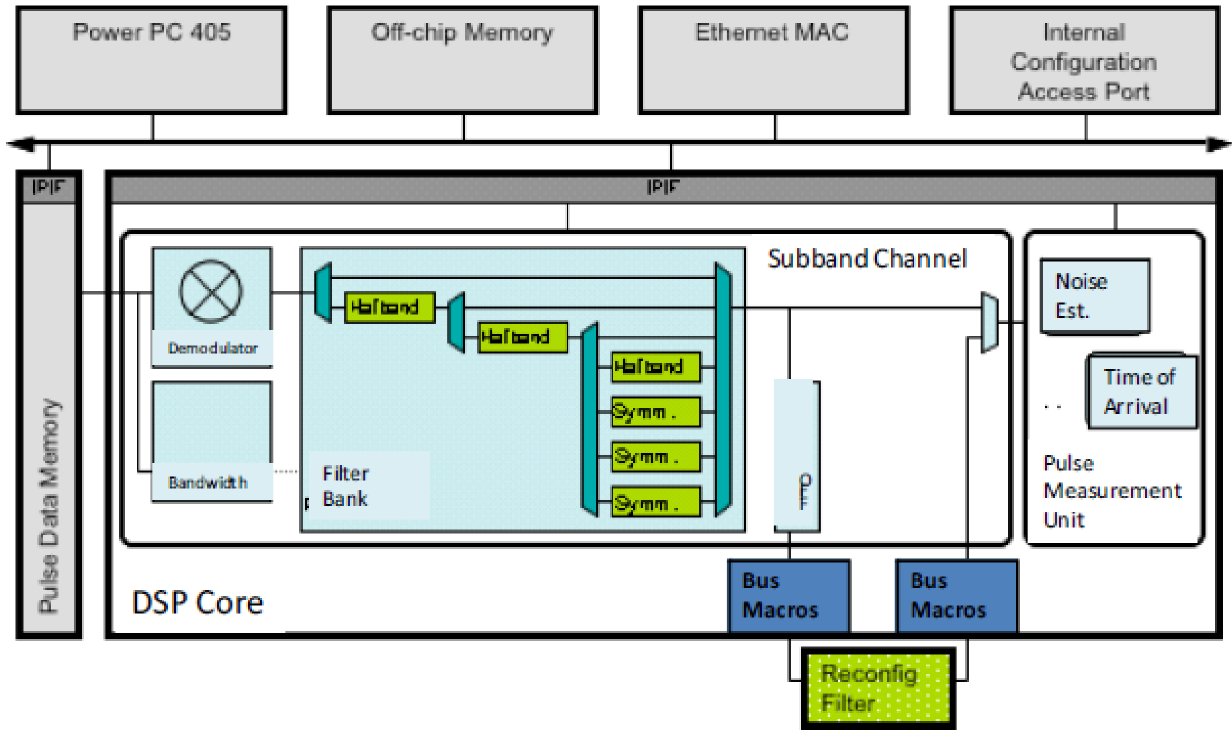
RTR security primitives: (i) speed up computation; (ii) allow switching between different primitives; (iii) provide trade-offs.

The security primitive controller selects the bitstream corresponding to the chosen algorithm and parameters (in the "configuration" state).

Source: [Gogniat et al., 2008]

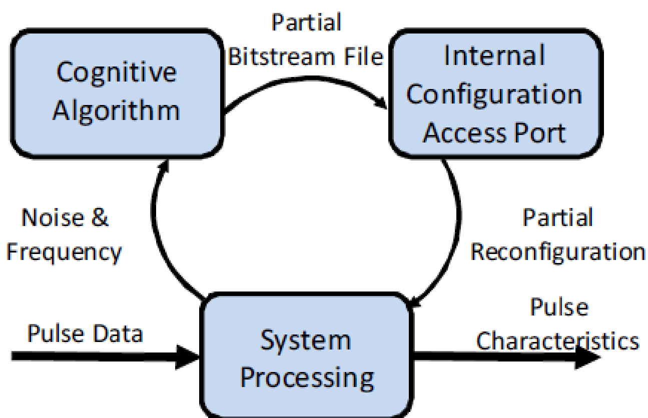


Example: Autonomous interference mitigation



Source: [French et al., 2008]

Autonomous system feedback loop



Operation	Time
Heuristics transfer to PowerPC	7 ms
BN execution	~300 us
Off chip bitstream acquisition	10 ms
Linux memory space copy	16 ms
Data transfer to ICAP	70.7 ms
Reconfiguration time	8 ms
Total	112 ms

Source:[French et al., 2008]

Some difficulties and challenges

Assuming a robust design flow, the development of complex, (embedded) systems involves the following RTR-related issues:

- Devising a (adaptive) reconfiguration strategy.
  Must take into account the algorithms to be used (choice?) and the performance constraints. Hardware/software co-design.
  Definition of the reconfiguration policy/scheduling.
- Design of the (partial) configurations.
  Includes the design of the logical infrastructure (for modular approaches).
- Validation of system operation with temporal change of hardware.
  Includes validation of (all) possible module combinations.
  Includes composition of information for each combination at various levels (functional and post-layout simulation).
- Debugging of a dynamically changing system.

1 Introduction to reconfigurable computing

2 Run-time reconfiguration of hardware
   General aspects
   Creating configurations
   Potential advantages

3 Some devices with support for RTR

4 Flexible bitstream generation
   General approach
   Pragmatic aspects
   Bitstream assembly
   Bitstream assembly at run-time

5 Opportunities

Configuration architectures

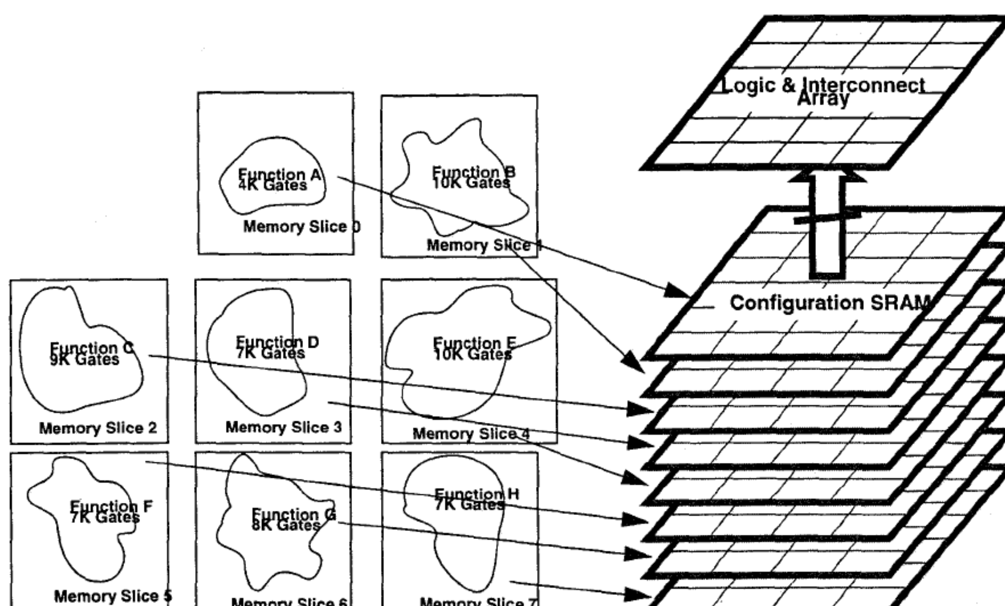
Two main configuration architectures:

- Single-context
  Device has only one configuration: most common in commercial FPGAs.

  Some devices: partial reconfiguration (addressable configuration memory).
- Multi-context
  Each configuration point is controlled by a multiplexer that chooses between several controlling values.
  - background loading of configurations
  - very fast switching between configurations
  - *but* reduces physical capacity (for the same chip area)
  - spikes in dynamic power consumption

Example: Multi-context FPGA I

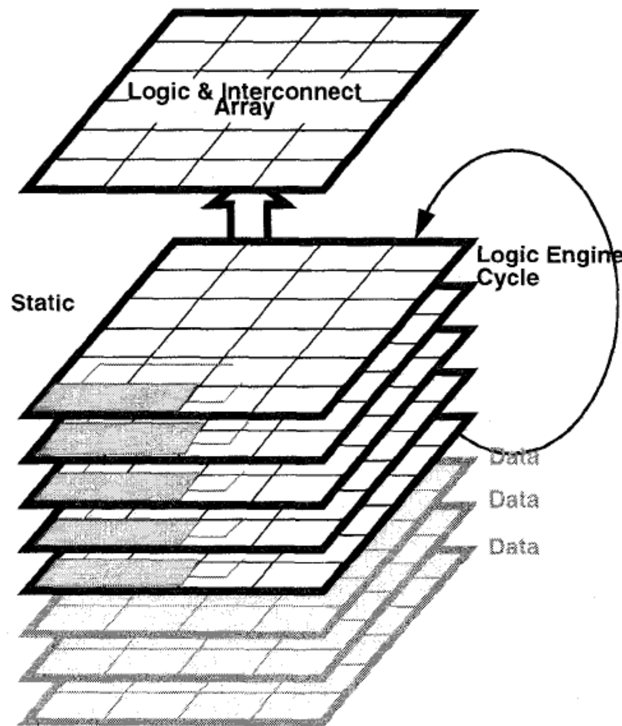
Time-sharing operation of 8-context FPGA: variant of Xilinx X4000E.
➠ Extend capacity by adding reconfiguration bits, not CLBs.



Source [Trimberger, 1997]

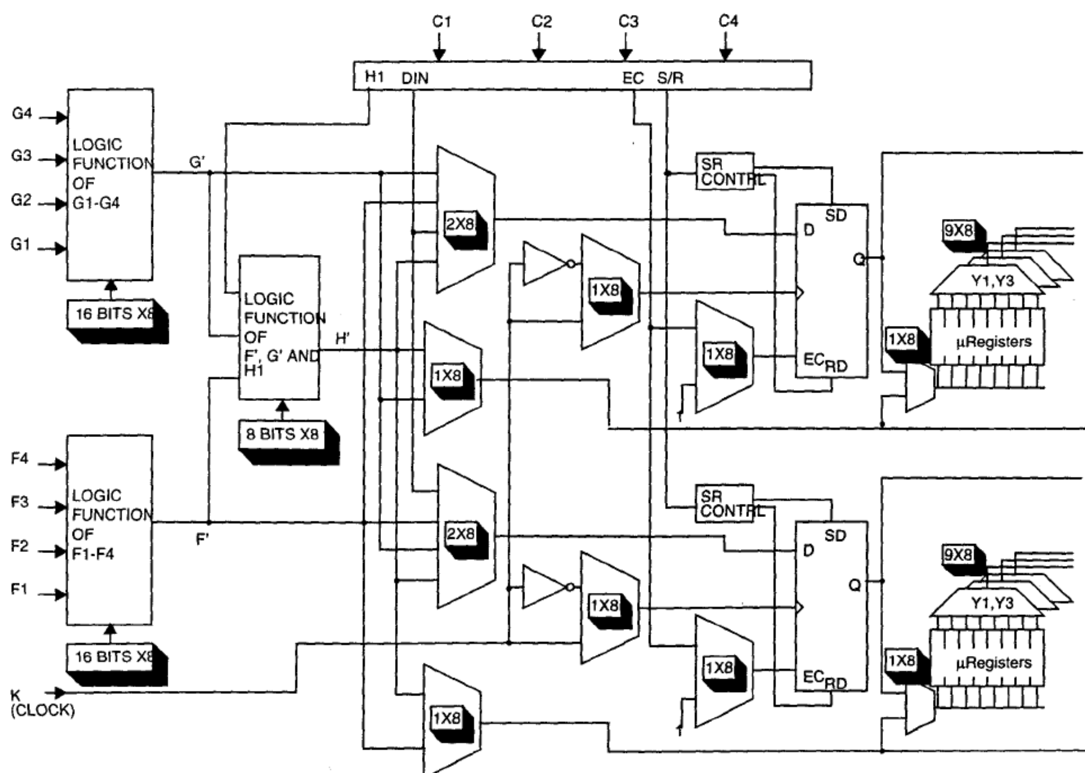
Example: Multi-context FPGA II

"Logic engine" mode: "user cycle" goes through several contexts



Source [Trimberger, 1997]

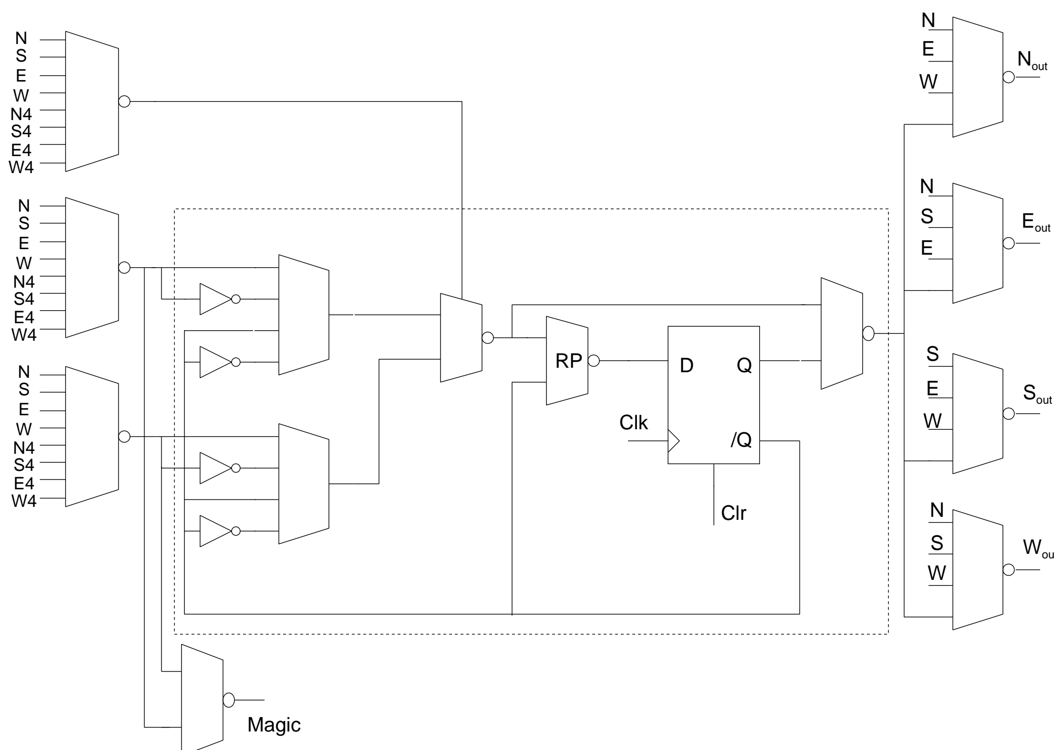
Example: Multi-context FPGA CLB



Source [Trimberger, 1997]

Partial reconfigurability: Xilinx XC6200

- Fine-grained family from Xilinx



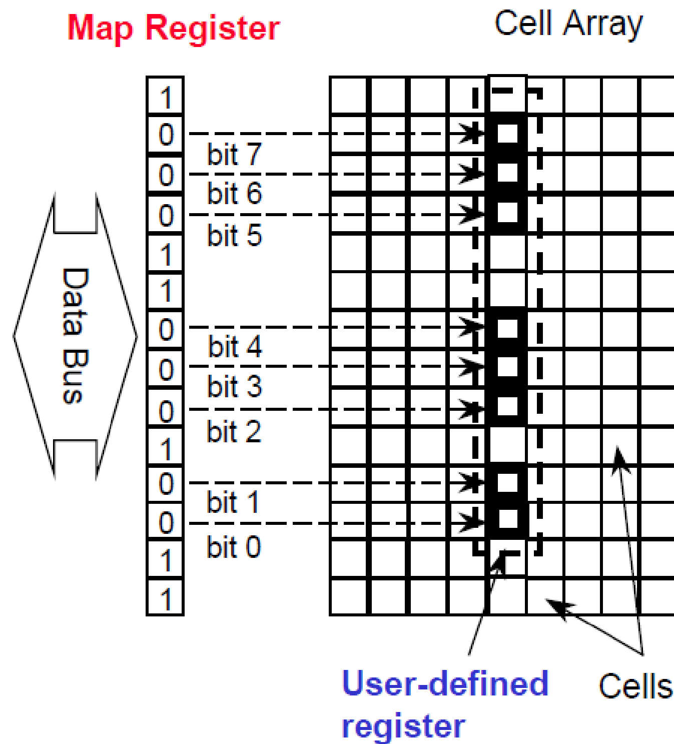
XC6200: FastMap interface

The FastMap interface of the XC6200 allows:

- random access to configuration memory
  - fast partial device configuration
  - only program the bits that affect specific LE or interconnections (no frames)
- direct access to *user registers*
- memory-mapped interface is “like SRAM”
- wildcard registers allow *don’t cares* in the address bits
  - same data can be written to several locations in one cycle
  - fast configuration of bit-slice type designs
  - broadcast of data to registers

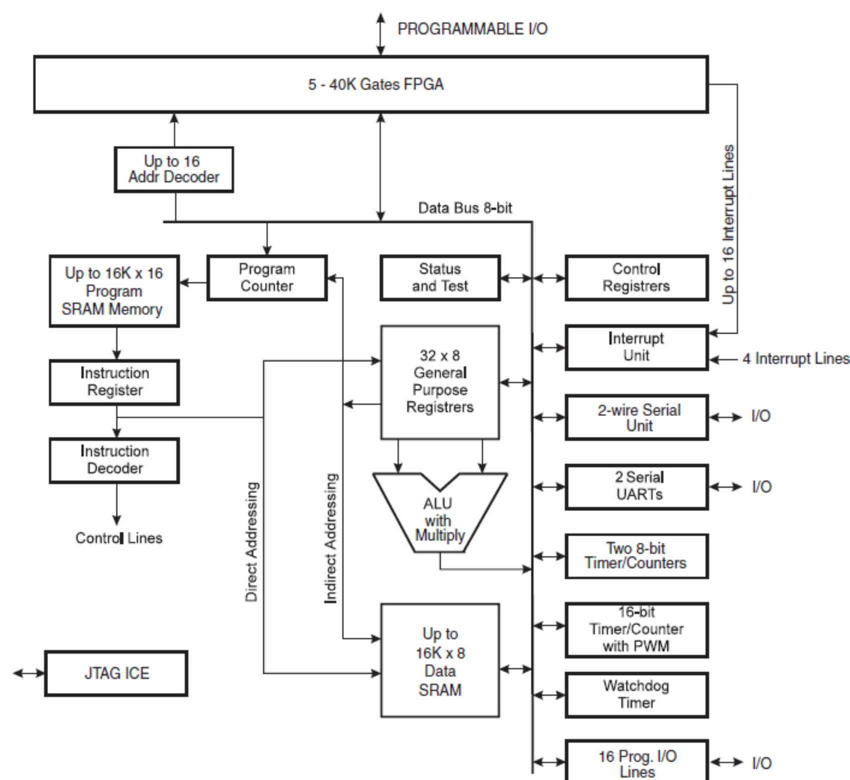
XC6200: Map register

Map register maps user register onto external data bus (8, 16, 32 bits):



Reconfigurable fabric as peripheral

Atmel AVR 8-bit RISC microcontroller + AT40K SRAM FPGA:



Interface AVR/FPGA

- AVR treats FPGA as a large I/O device
- AVR can configure the FPGA (Cache Logic)

Source: Atmel datasheet

Morpheus

Source [Voros et al., 2009]

1. Introduction to reconfigurable computing
2. Run-time reconfiguration of hardware
   - General aspects
   - Creating configurations
   - Potential advantages
3. Some devices with support for RTR
4. Flexible bitstream generation
   - General approach
   - Pragmatic aspects
   - Bitstream assembly
   - Bitstream assembly at run-time
5. Opportunities

General context

- Objective: flexible bitstream generation
- Approach inspired by traditional software development:

  Partial configurations are produced by assembling components from a previously created library

  Rationale: reduction of the effort involved in creating many partial bitstreams for RTR
- Additionally: Support for generation of bitstreams at run-time

➠ Target platform: Virtex-II Pro + external memory

Additionally: Insight into pragmatic aspects of RTR and implementation trade-offs.

Bitstream generation by component assembly

Problem: How to generate many similar configurations efficiently?

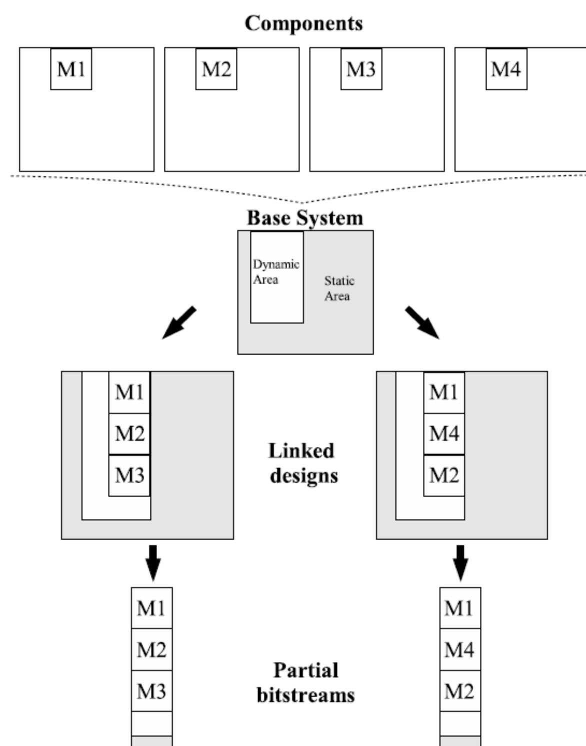
➠ Assemble configurations from partial bitstreams of smaller components.

Example: Creation of pipelines where each stage may have several variants.

Analogy: Linking several procedures to create one executable.

- library of basic components (medium granularity) with
  - bitstream format (black box)
  - interface information
- bitstream manipulation to create new assemblies
  - relocation of component bitstreams + merging
  - interconnection: simplest (but less flexible) is by abutment
  - no additional restrictions on the *internal* organization of the dynamic area
- fast process

Example of bitstream assembly



Source: [Silva and Ferreira, 2006]

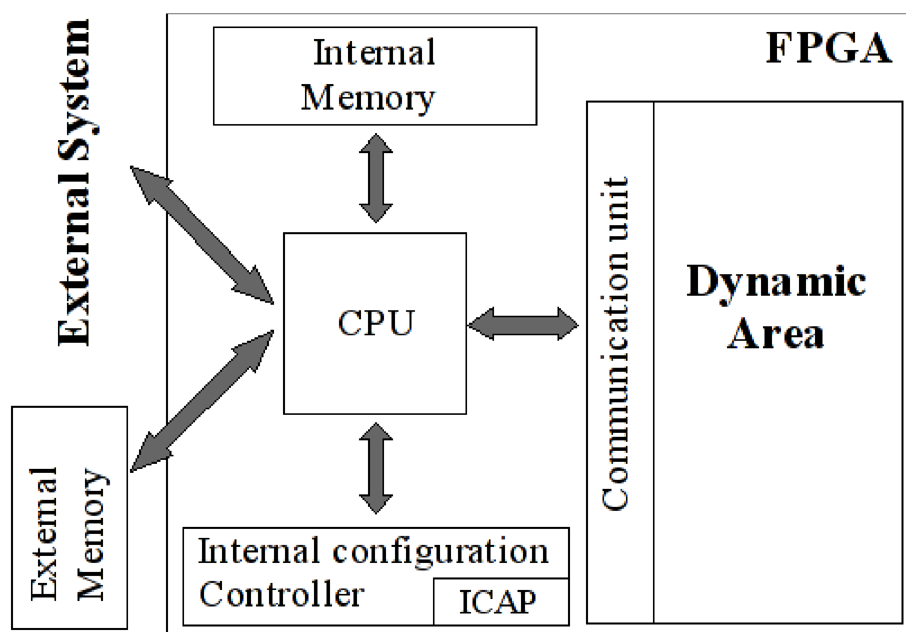
Implementation context

- Device with support for RTR (partial reconfiguration)
- Closely-coupled CPU
- Internal partial reconfiguration control
  - ICAP: Internal Configuration Access Port
    for downloading and reading-back of configurations
  - the system can be *self-reconfigurable*
- FPGA divided in two areas:
  - *Static area:*
    CPU, memory, reconfiguration control, data transfer control
  - *Dynamic area:* Time-shared to support multiple tasks

    Includes interface to the fixed-area (the "dock")

Generic system organization

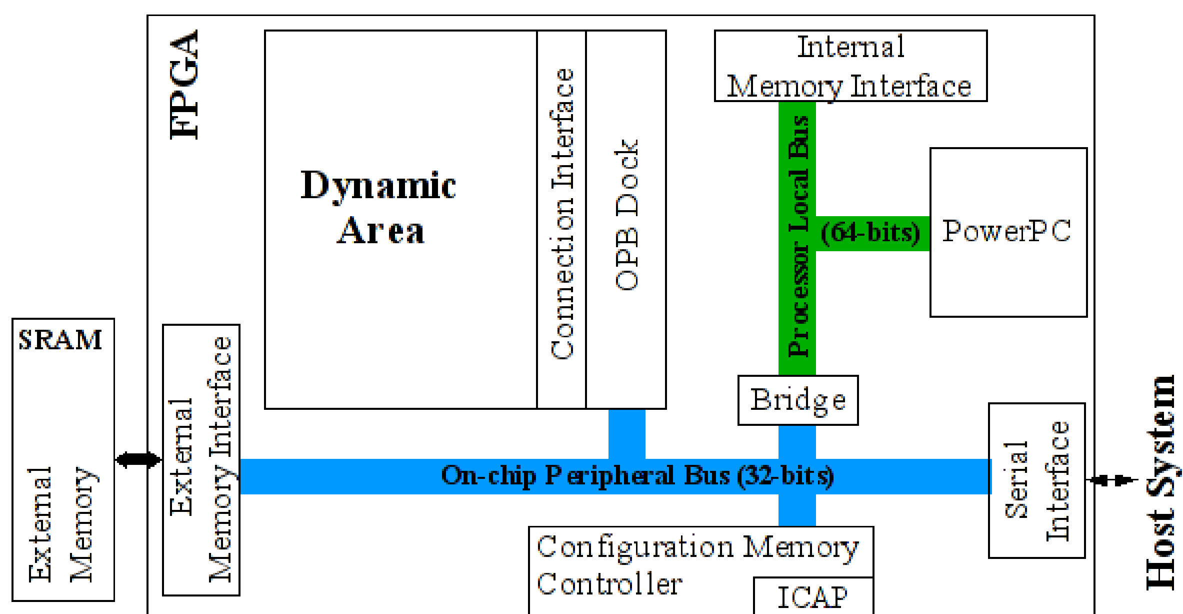
➠ Software application running on the CPU manages run-time tasks on the dynamic area.



1. Introduction to reconfigurable computing
2. Run-time reconfiguration of hardware
   - General aspects
   - Creating configurations
   - Potential advantages
3. Some devices with support for RTR
4. Flexible bitstream generation
   - General approach
   - Pragmatic aspects
   - Bitstream assembly
   - Bitstream assembly at run-time
5. Opportunities

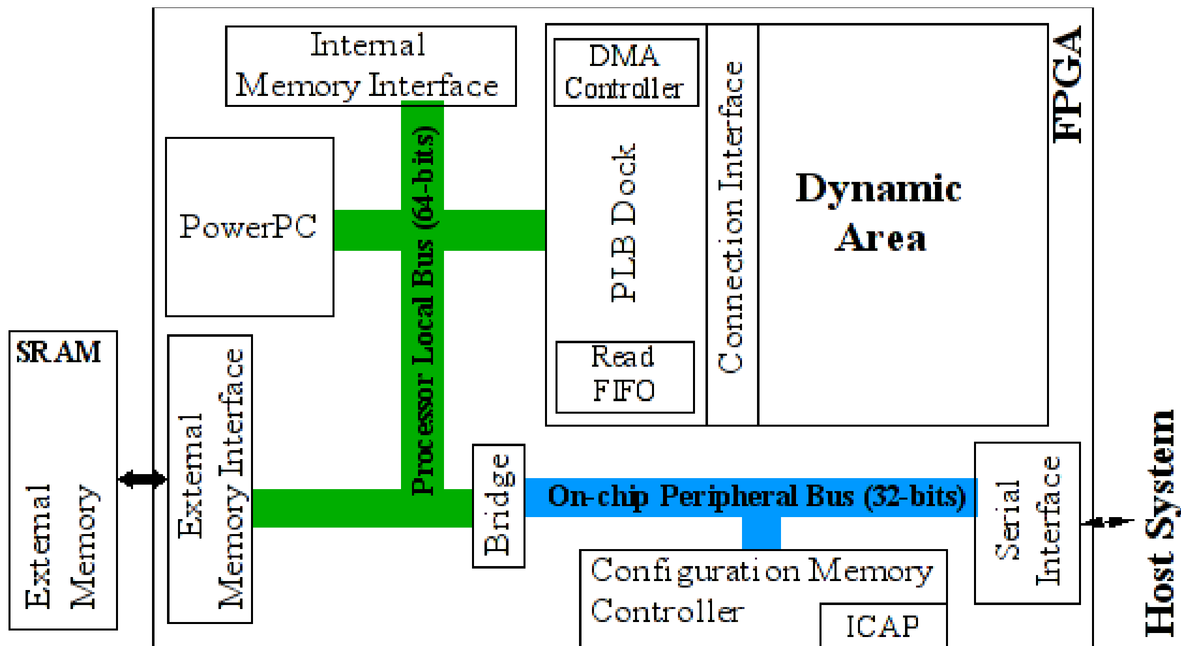
32-bit system implementation

32-bit OPB used for communication with: external memory (data), dynamic area through OPB Dock, configuration controller.



64-bit system implementation

64-bit PLB used for communication with: external memory (data), dynamic area through PLB Dock.



System comparison

32-bit system

- Xilinx XC2VP7
- 36% slices used
- CPU at 200 MHz
- Buses at 50 MHz
- Dynamic area: 25% slices
- 1 CPU

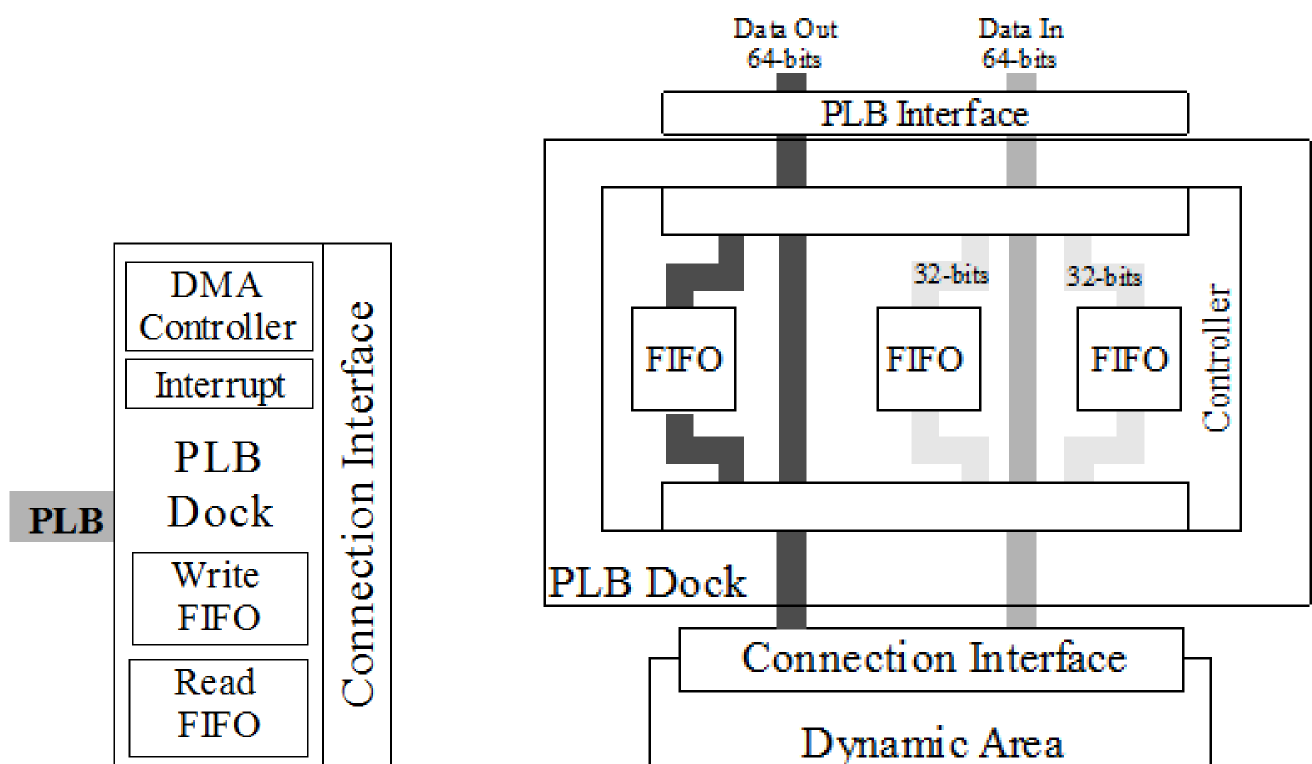
64-bit system

- Xilinx XC2VP30
- 31% slices used
- CPU at 300 MHz
- Buses at 100 MHz
- Dynamic area: 22.4% slices
- 1 of 2 CPUs used

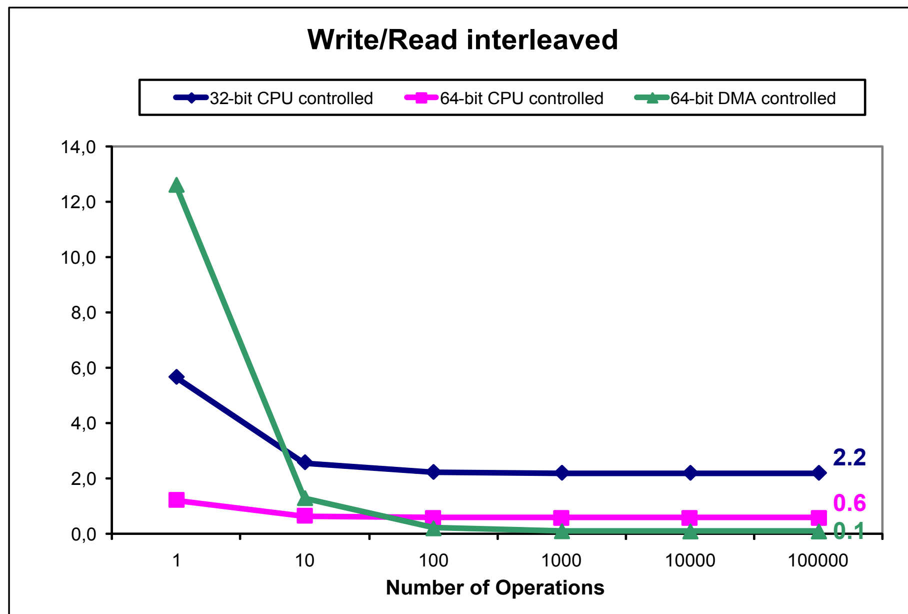
Data and configuration transfers

- Data transfers to/from dynamic area may limit performance
- CPU-controlled transfers are inefficient and don't use 64 bits
  - load/store instructions operate on 32 bits
- DMA can use 64-bit width and frees CPU for other tasks
- DMA transfers restrict:
  - data organization
  - access patterns
  - have a high setup overhead
- Speed of configuration transfers to the ICAP can also be improved by DMA

More versatile 64-bit dock with DMA



Baseline performance for data transfers



$$\frac{t_{\text{CPU32}}}{t_{\text{CPU64}}} = 3.7 \quad \text{to} \quad 6.5 \qquad \frac{t_{\text{CPU64}}}{t_{\text{DMA64}}} = 4.7 \quad \text{to} \quad 6.7$$

Speeding up reconfiguration

- Original ICAP driver is inefficient (EDK 8.2)

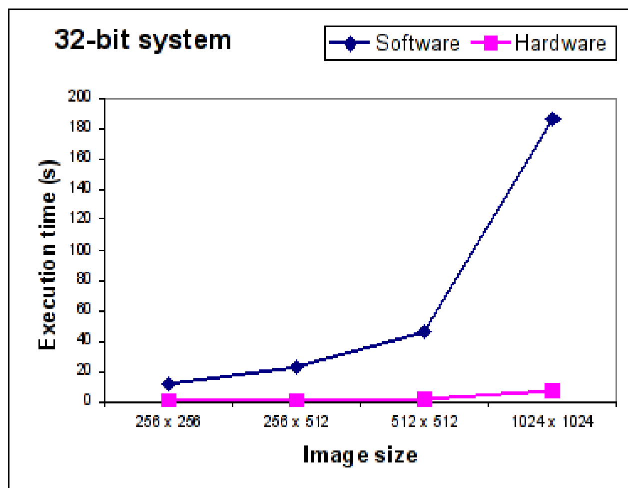
  Reconfiguration of the dynamic area takes 0.17 s

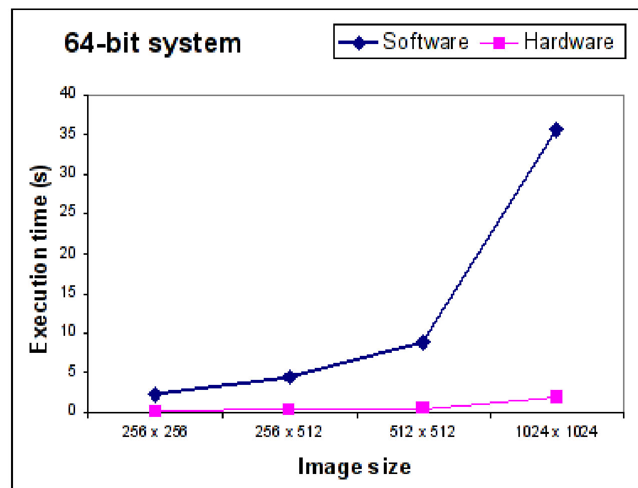
  The bottleneck lies not in the ICAP, but in the transfer process from memory to the HWICAP (wrapper around ICAP).
- Driver that uses the DMA controller improves reconfiguration

Size	Reconfiguration time (ms)	
	Original driver	DMA capable driver
One frame	0.32	0.01
CLB column (22 frames)	7.13	0.31
Dynamic area (24 columns)	171.07	7.39

Case study: pattern matching (images)

Pattern matching task for bilevel images:
(i) Search pattern, 8x8 pixels; (ii) Implemented as an 8 stage pipeline; (iii) 8-bit transfers (Write/Read interleaved).





Average speedup = 27

Average speedup=19

Case study: SHA1 algorithm

Implementation of SHA1 algorithm (RFC 3174)

- Only fits in the dynamic area of the 64-bit system
- 32-bit transfers (N writes, 5 reads)

Data length (bytes)	Execution time (ms) Software	Execution time (ms) Hardware	Speedup
64	24.35	0.02	1217.5
640	30.54	0.24	127.3
6400	47.91	2.35	20.4
64000	261.95	23.51	11.1
640000	2402.56	235.09	10.2

Case study: Simple image processing

32-bit system: (no DMA)

Task	Execution time per output pixel (μs)			Speedup
		Hardware		
	Software	Data preparation	Total	
Brightness adjustment	2.18	n.a.	0.55	4.0
Additive blending	2.96	0.72	1.27	2.3
Fade effect	3.94	0.72	1.38	2.9

64-bit system: (no DMA)

Task	Execution time per output pixel (μs)			Speedup
		Hardware		
	Software	Data preparation	Total	
Brightness adjustment	0.48	n.a.	0.01	48.0
Additive blending	0.64	0.19	0.20	3.2
Fade effect	0.90	0.19	0.20	4.5

Case study: Simple image processing with DMA

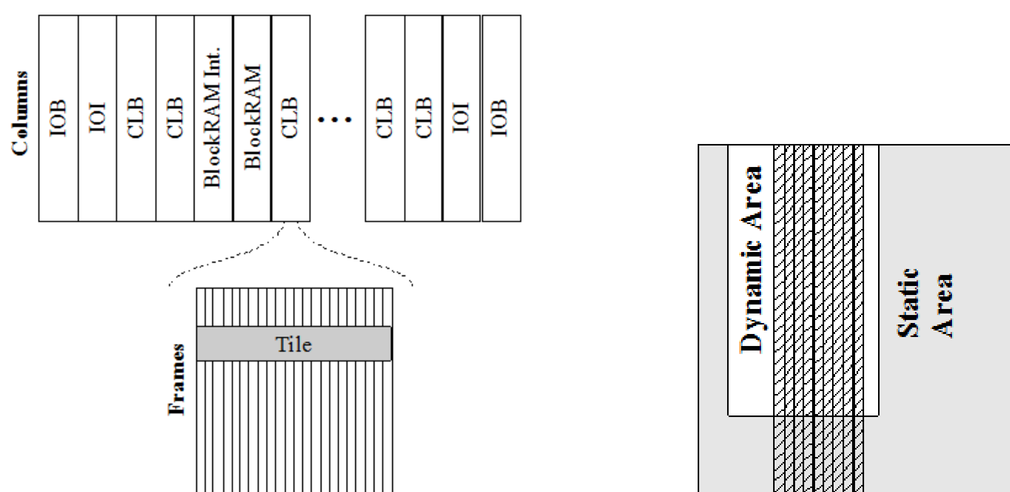
64-bit system with DMA:

Task	Execution time per output pixel (μs)		Speedup
	Software	Hardware	
Brightness adjustment	0.48	0.01	48.0
Contrast adjustment	0.73	0.01	73.0
Additive blending	0.64	0.04	16.0
Fade effect	0.90	0.04	22.5
Motion detection	0.79	0.04	19.8
Motion overlay	0.82	0.04	20.5

Average values for image sizes 256×256, 256×512, 512×512 and 1024×1024.

1. Introduction to reconfigurable computing
2. Run-time reconfiguration of hardware
   - General aspects
   - Creating configurations
   - Potential advantages
3. Some devices with support for RTR
4. Flexible bitstream generation
   - General approach
   - Pragmatic aspects
   - Bitstream assembly
   - Bitstream assembly at run-time
5. Opportunities

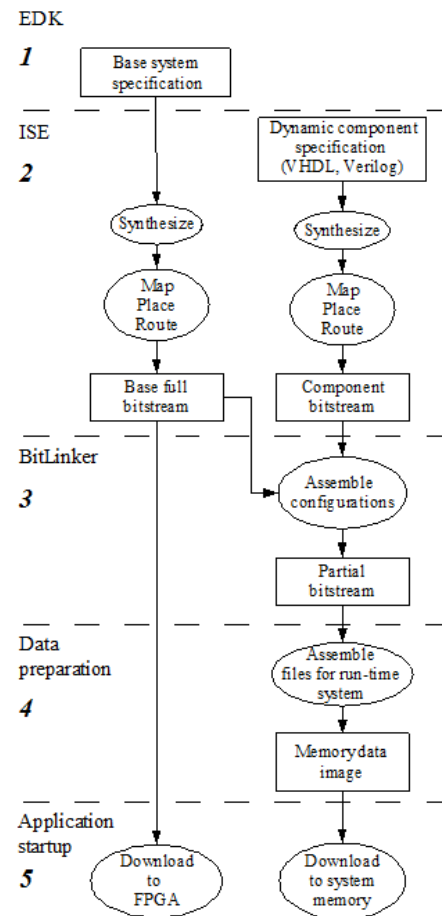
Bitstream organization of the Virtex-II Pro



- Configuration data organized in frames
- Frames are grouped in different sections
- Frame: 1.bit column that spans to the total height of the device
- Limitations imposed by host board don't allow the dynamic area to be of full height

Design flow

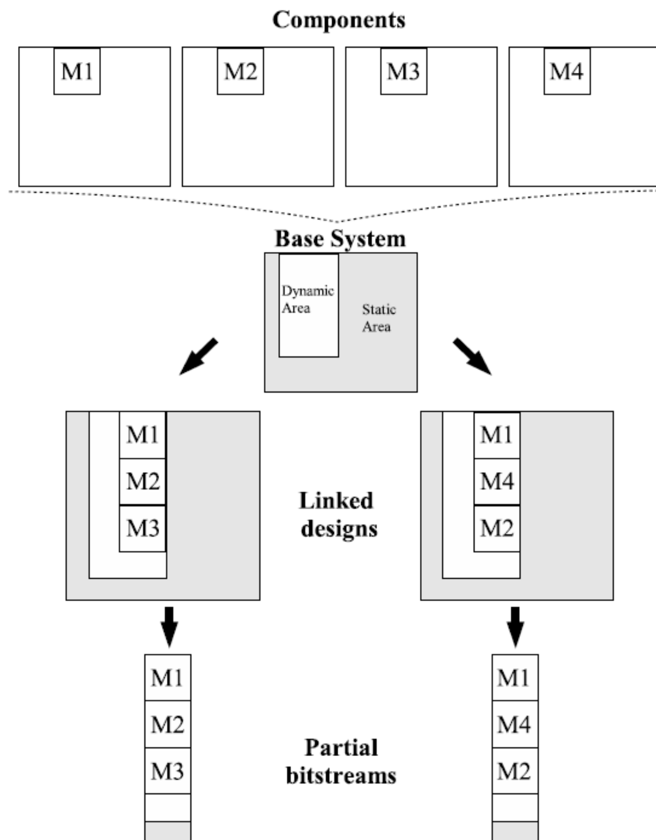
- Base system implementation
  - fixed logic
  - interface to dynamic area
  - dynamic area must be unused
- Hardware component implementation
  - One project per component
  - Components must follow a common pattern for I/O placement



Configuration assembly

- Individual components assembled together to produce loaded configuration
- Communication between components done through *fixed* connection macros
- Multiple possible arrangements
- Modules can be relocated and replicated to different positions of the device
- Multiple configurations can be built on the fly from existing modules without needing to synthesize a new design
- BitLinker
  - Configuration information of hardware components extracted from complete bitstream
  - Component relocation to compatible areas
  - Ensures assembly created compatible with dynamic area
  - Produce correct partial bitstream for whole area affected by the reconfiguration

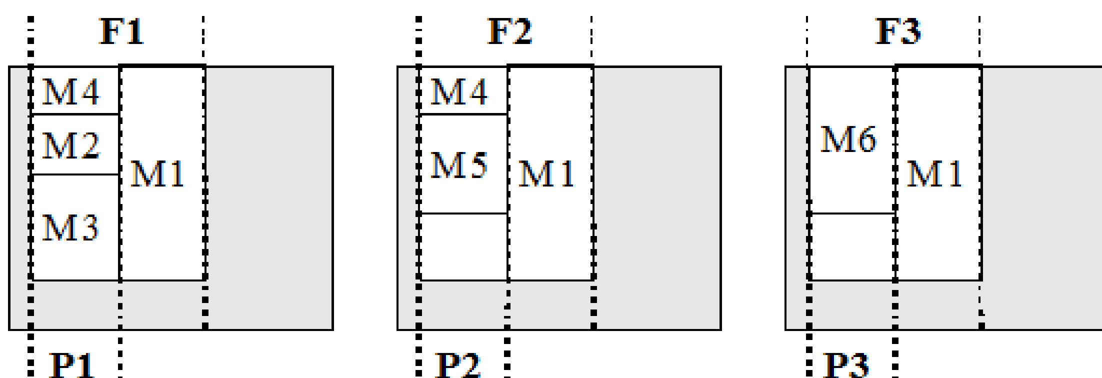
BitLinker: basic tasks



Small partial bitstreams

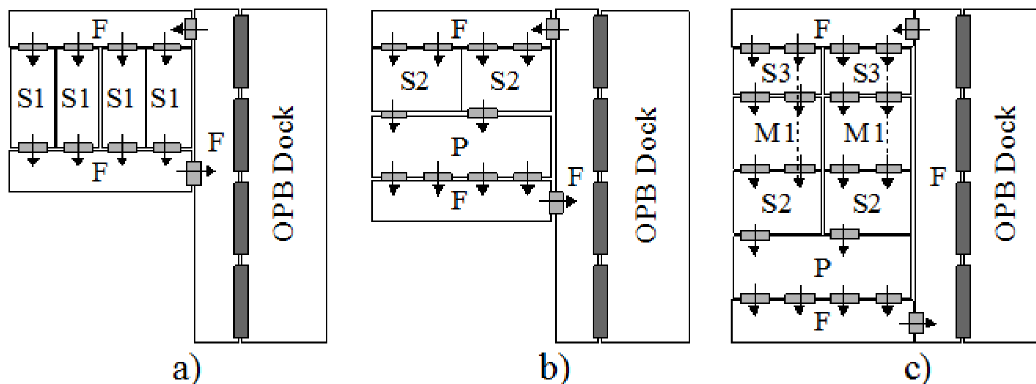
When there are sections common to a set of partial bitstreams:

- Partial bitstream corresponds to assembled configuration or to a vertical section
- Extracting parts of an assembly creates smaller bitstreams
- Decreases time to switch between assemblies with common sections



Example: sub-word operations for image processing

Symbol	Operation
S1	Saturating addition with signed constant
S2	Saturating addition
S3	Saturating subtraction
M1	Saturating multiplication with constant
P	32 bit unsigned accumulator
F	Feedthrough components



All assemblies mentioned before were produced in this way.

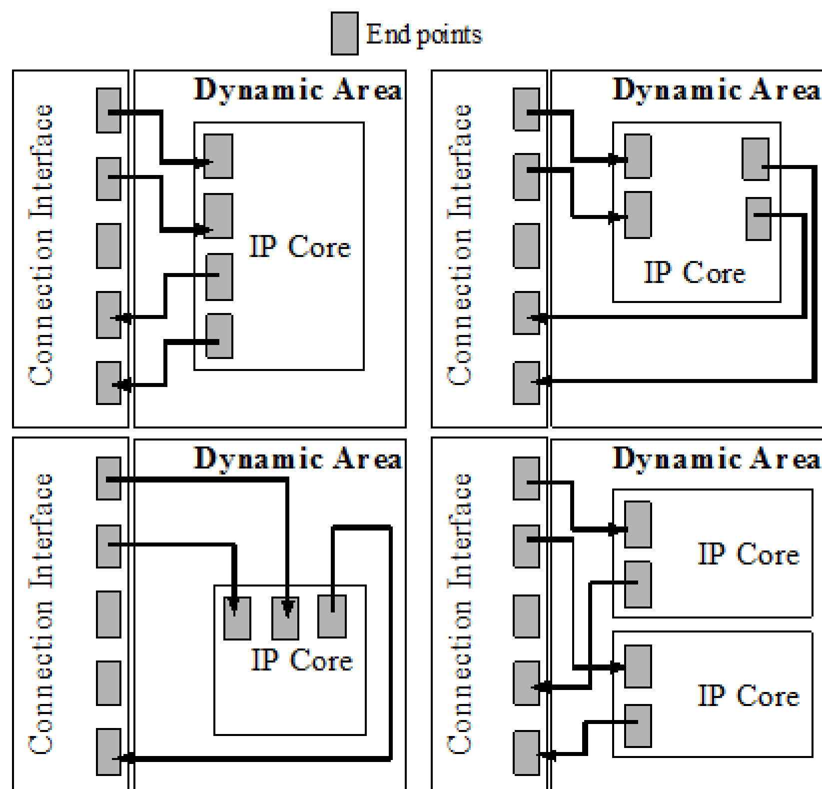
IP cores in RTR systems

- In a traditional development flow, IP cores are closely connected to a base system.
- Requirements for using IP cores in RTR systems
  - The IP cores must fit in the dynamic area
    Area, aspect ratio, resource usage
  - IP cores must have a compatible connection interface
- We assume the IP core is a completely implemented design
  - The IP core has been synthesized, placed and routed
  - No knowledge about the implementation details
- Assumptions about data availability (black box)
  - Configuration bitstream
  - I/O and bounding-box information

Using bitstream IP cores

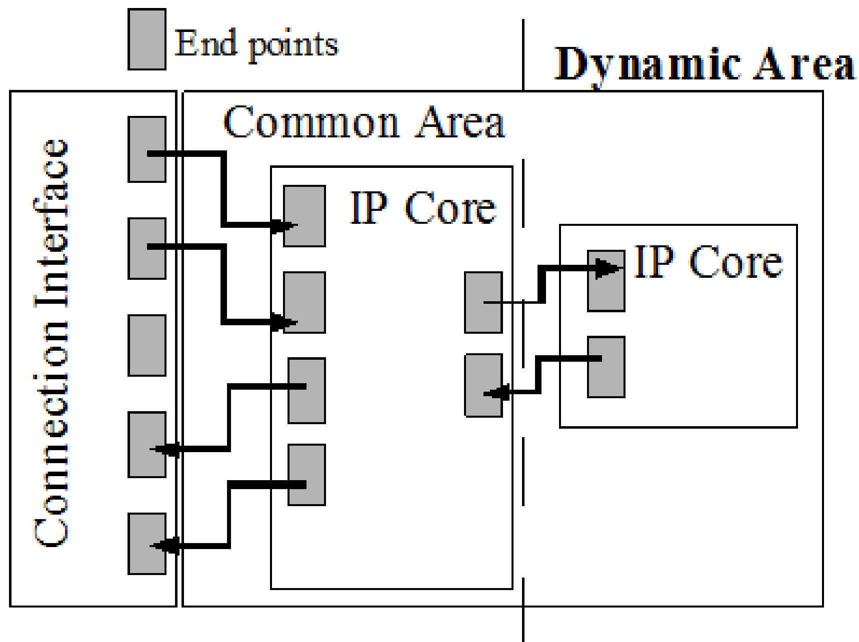
- *Problem:* How can we make third-party cores with different physical characteristics time-share the same area?
- Extension: more flexible connections between components. Bitstream IP cores can adapted to the dynamic area by
  - Limited place and route operations
  - and bitstream manipulation
- The same bitstream can be used
  - with different base systems
  - combined with other cores in different arrangements
- Output: Partial bitstreams that can be loaded at run-time into a given base system.

Bitstream IP Cores (Examples)

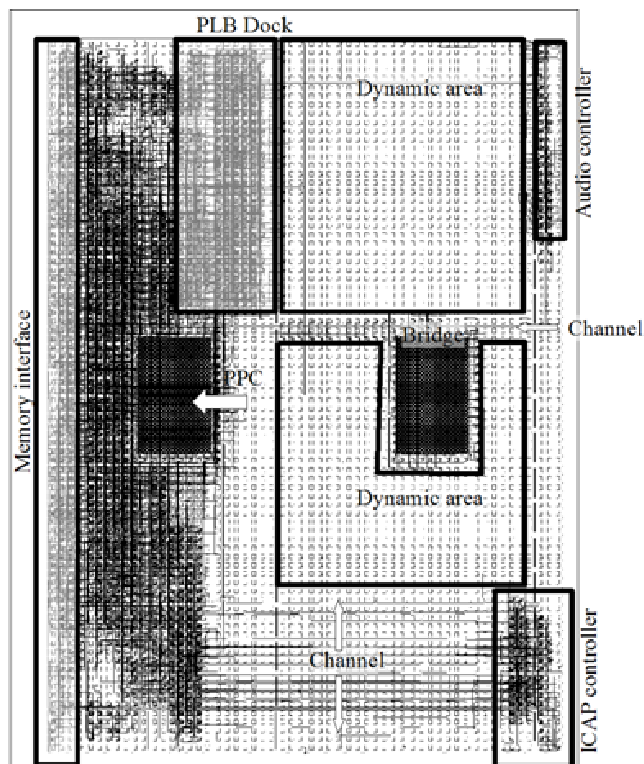


Another supported scenario

- multiple IP core combinations differ by only in one core
- only one core must be swapped at run-time

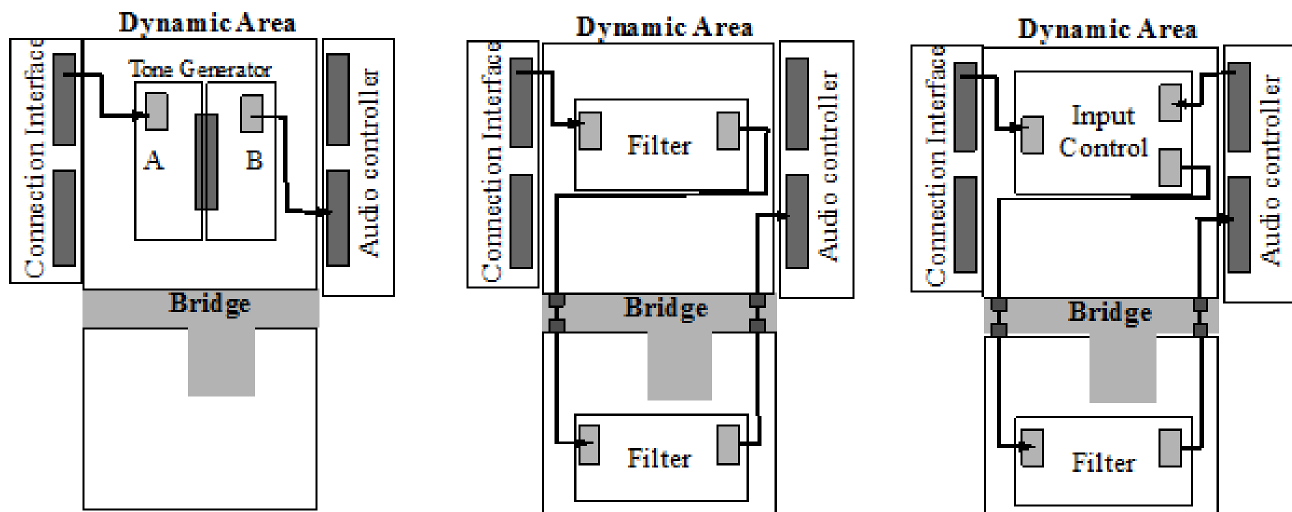


Layout of the system



➠ Another extension: The dynamic area is not a single rectangle.

Example: cores for sound processing



	Number of connections	Assembly time routing (s)
Example 1	48	3,5
Example 2	104	7,3
Example 3	72	5,3

1. Introduction to reconfigurable computing
2. Run-time reconfiguration of hardware
   - General aspects
   - Creating configurations
   - Potential advantages
3. Some devices with support for RTR
4. Flexible bitstream generation
   - General approach
   - Pragmatic aspects
   - Bitstream assembly
   - Bitstream assembly at run-time
5. Opportunities

The next step

➠ *Approach:* application modifies dedicated hardware according to needs determined at execution time
Reasons for on–line generation:

- Increasing system "flexibility", while preserving good performance
- Allowing trade-off between performance / hardware usage

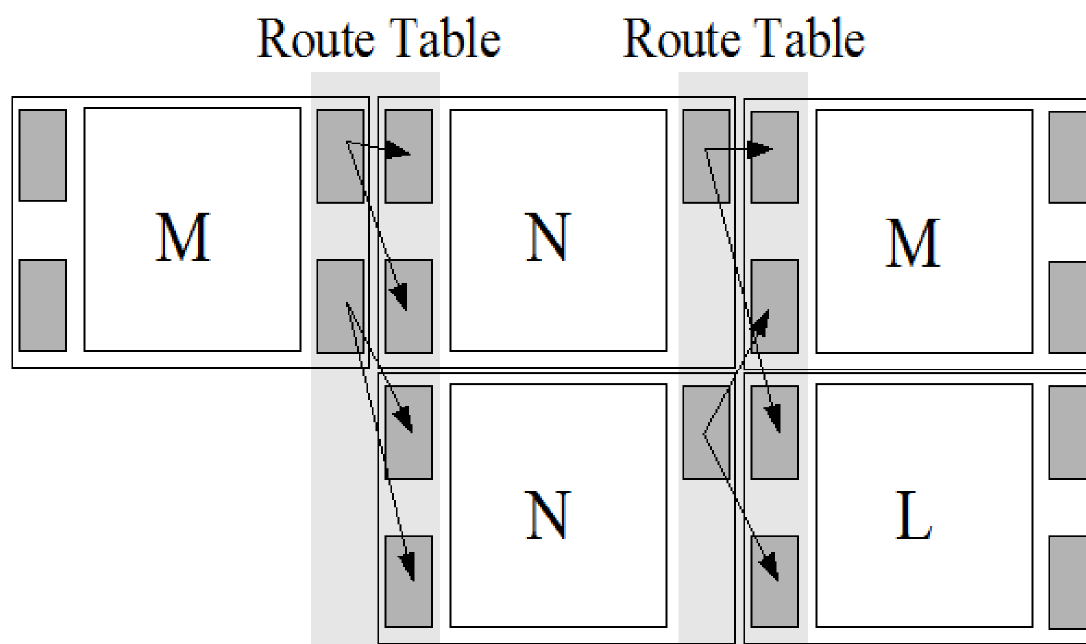
  If some function is done often in software, create a corresponding hardware version
- Avoid the need to pre-generate large numbers of partial configurations
- Increased application "portability" (different dynamic area layouts)

➠ Bitstream assembly must use relatively little CPU power (embedded systems) and be "fast".

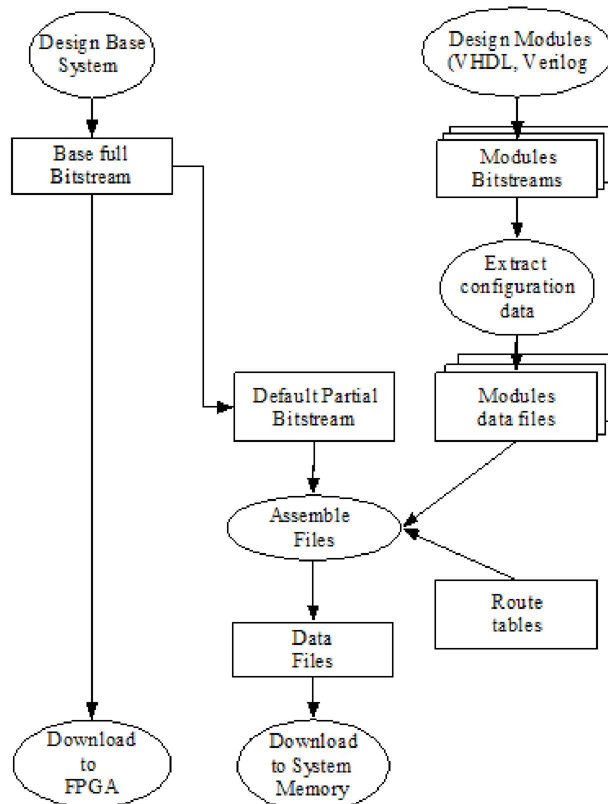
Component connection

➠ Enable flexible connections between components in adjacent columns

A route table contains information about all possible connections.



Preprocessing flow



Route table information

A complete route table contains three sets of data:

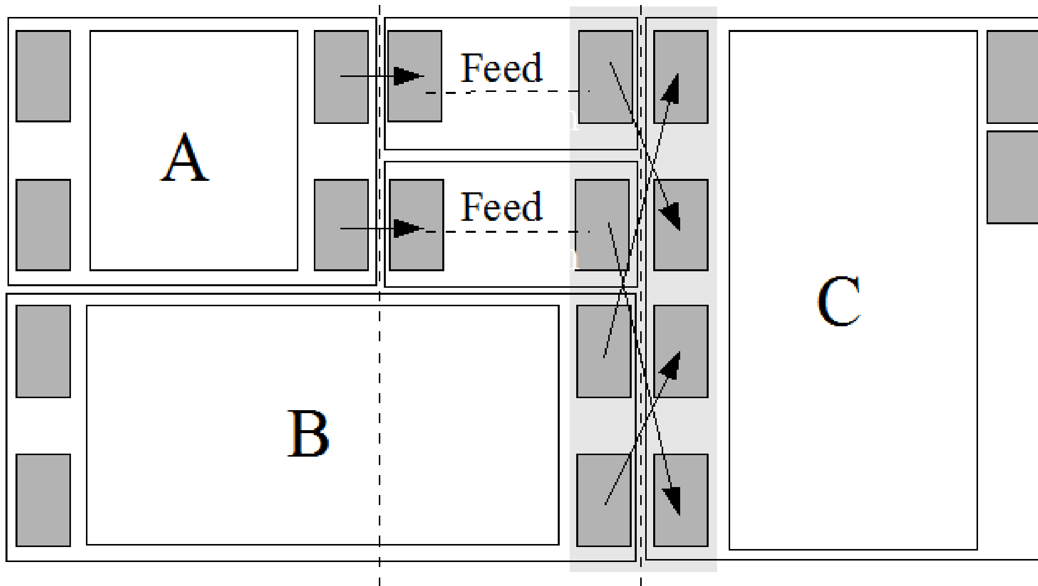
- connection identification: relative coordinates of the connection endpoints;
- route specification: list of frame and bit indexes of all switching points in the route;
- incompatibility information: set of routes that are mutually incompatible

Height	Number of connections	Size(KB)	Time
4	128	4	~4s
32	8192	416	~30m
80	49928	6826	~4h

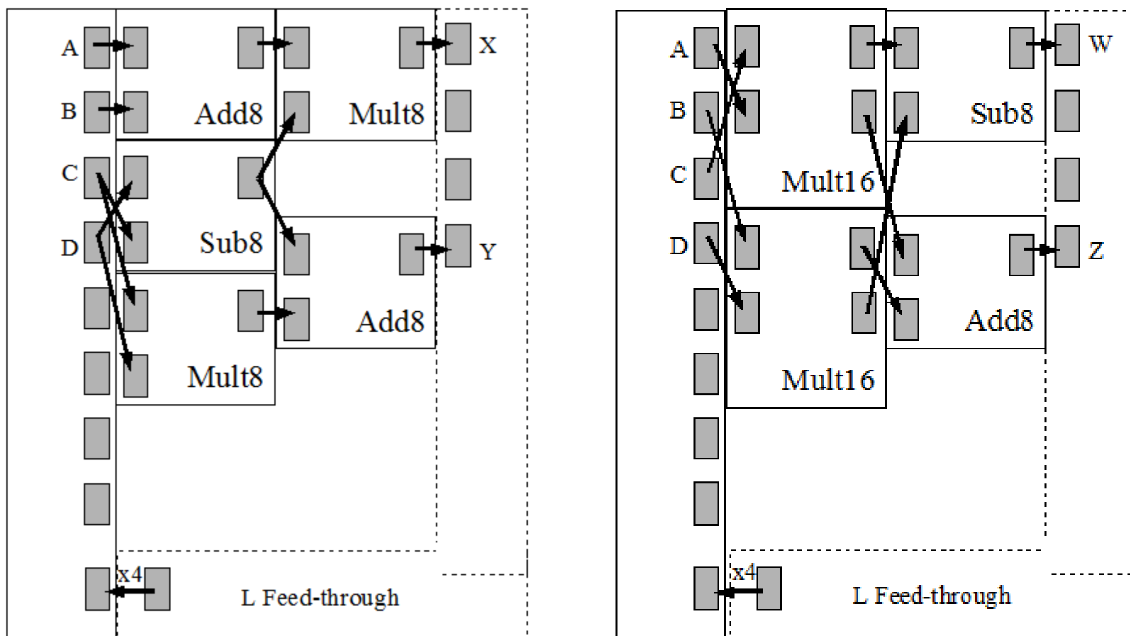
One-time-only generation (per device and dynamic area).

Avoiding some connection limitations

➠ Feed-through modules are inserted automatically.



Examples for arithmetic expressions



Example 1: 24 ms Example2: 22 ms

Both partial bitstreams have 528 frames; reconfiguration time is 7.5 ms.

Source: [Silva and Ferreira, 2008]

1. Introduction to reconfigurable computing
2. Run-time reconfiguration of hardware
   - General aspects
   - Creating configurations
   - Potential advantages
3. Some devices with support for RTR
4. Flexible bitstream generation
   - General approach
   - Pragmatic aspects
   - Bitstream assembly
   - Bitstream assembly at run-time
5. Opportunities

Limitations of commercial devices and tools

- Addressability of configuration memory

  Accessing configuration memory is time-consuming and complex

  Good: Basic unit of configuration access is getting smaller (Virtex-4)

  Side note: Bitstream formats are generally not public.
- Large bitstream size
- Power consumption

  Reconfiguration costs a (relatively) large amount of energy.
- Commercial tool support is limited

  Low-level support is available (experimentally), but no higher-level tools.

  Positive: Large number of academic tools, flows and methodologies.

  For many examples, see: [Hauck and DeHon, 2008], [Voros et al., 2009]
- No portability to different device families (let alone vendors)

Opportunities (formerly known as obstacles)

- Including hardware change (temporal dimension) in design,implementation and validation.
- High-level, general-purpose models for specification and verification are not (yet!) available.
- Debugging adaptive systems is difficult.
- Design space exploration is more complex (but can achieve better results).
- Benefits of RTR are (currently) application-dependent, but RTR may enable new classes of systems.

➠ Trend towards complex, autonomous, adaptive embedded systems makes RTR more attractive:

- Increased use of heterogeneous, many-core SoCs (including RTR fine- and coarse-grained fabrics).
- Areas: domestic robots, smart camera networks, cars, mobile broadband wireless access (IEEE 802.16j) . . .

➠ And wouldn't hardware JIT compilation be nice?

Challenges in complex embedded systems

HIPEAC identified several challenges associated with RTR [Bosschere et al., 2007]:

- Mapping computations on accelerators
  - The problem is especially challenging for less conventional computing nodes such as FPGAs.
- Run-time support for reconfiguration
  - Multi-tasking
  - Transparent hardware/software boundaries
- Improved run-time reconfiguration
  - Novel configuration memories (fewer soft errors, multi-context)
- Dynamically adaptable network-on-chip technologies and main memory interfaces
- Tool chains (again. . .)

High-level algorithms to hardware

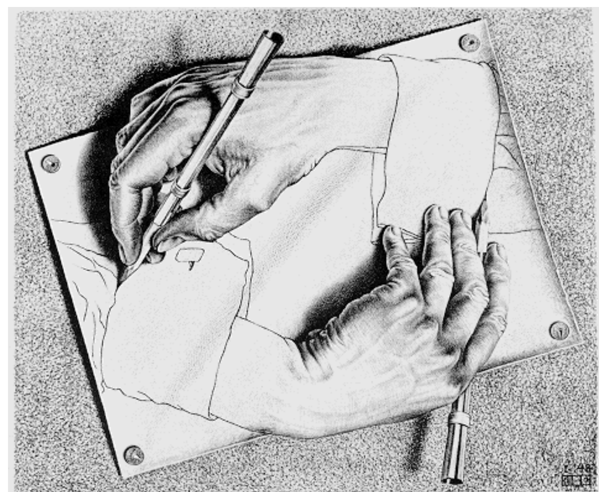
- REFLECT: Rendering FPGAs to Multi-Core Embedded Computing (FP7 Strep)
- Objective: To develop, implement and evaluate a novel compilation and synthesis system approach for FPGA-based platforms.
- Aspect-Oriented specifications will convey domain knowledge to a mapping engine
- Keep the advantages of a high-level imperative programming paradigm
- How: extensible intermediate mapping language
  - exploration of alternative architectures and run-time adaptive strategies
  - generation of flexible hardware cores that can be easily incorporated into larger multi-core designs

Demonstrators: audio/video processing and real-time avionics

Start date: January 2010
Details at: `http://www.reflect-ist.org/`

RTR is not just circuits

Modular and dynamically reconfigurable



Self-reconfigurable

Thank you!

References I

J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka. Dynamic and partial FPGA exploitation. *Proceedings of the IEEE*, 95(2):438 –452, Feb. 2007. ISSN 0018-9219.

Koen De Bosschere, Wayne Luk, Xavier Martorell, Nacho Navarro, Mike O'Boyle, Dionisios Pnevmatikatos, Alex Ramirez, Pascal Sainrat, André Seznec, Per Stenström, and Olivier Temam. *High-Performance embedded architecture and compilation roadmap*, pages 5–29. Number 4050 in LNCS. Springer Verlag, 2007.

Christopher Claus, Johannes Zeppenfeld, Florian Müller, and Walter Stechele. Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system. In *Proceedings of the conference on Design, automation and test in Europe*, pages 498–503, Nice, France, 2007. EDA Consortium.

James G. Eldredge and Brad L. Hutchings. Run-Time reconfiguration: A method for enhancing the functional density of SRAM-based FPGAs. *The Journal of VLSI Signal Processing*, 12(1): 67–86, 1996.

R. J. Fong, S. J. Harper, and Peter M. Athanas. A versatile framework for FPGA field updates: an application of partial self-reconfiguration. In *Propc. 14th IEEE International Workshop on Rapid Systems Prototyping*, pages 117 – 123, June 2003.

References II

Matthew French, Erik Anderson, and Dong-In Kang. Autonomous system on a chip adaptation through partial runtime reconfiguration. In *16th International Symposium on Field-Programmable Custom Computing Machines (FCCM '08)*, pages 77–86, 2008.

G. Gogniat, T. Wolf, W. Burleson, J.-P. Diguet, L. Bossuet, and R. Vaslin. Reconfigurable hardware for High-Security/ High-Performance embedded systems: The SAFES perspective. *IEEE Trans.Very Large Scale Integration (VLSI) Systems*, 16(2):144 –155, February 2008. ISSN 1063-8210.

S. Hauck and André DeHon, editors. *Reconfigurable Computing*. Morgan Kaufmann, 2008.

B. L. Hutchings, R. Franklin, and D. Carver. Assisting network intrusion detection with reconfigurable hardware. In *Proc. 10th Annual IEEE Symp. Field-Programmable Custom Computing Machines*, pages 111–120, 2002.

Yana Esteves Krasteva, Eduardo de la Torre, and Teresa Riesgo. Reconfigurable heterogeneous communications and core reallocation for dynamic HW task management. In *Proc. ISCAS 2007*, pages 873–876. IEEE, 2007.

Michael G. Lorenz, Luis Mengibar, Mario G. Valderas, and Luis Entrena. Power consumption reduction through dynamic reconfiguration. In *Field Programmable Logic and Application*, pages 751–760. 2004.

References III

Mateusz Majer, Jürgen Teich, Ali Ahmadinia, and Christophe Bobda. The Erlangen Slot Machine: A dynamically reconfigurable FPGA-based computer. *VLSI Signal Processing*, 47 (1):15–31, 2007.

Miguel L. Silva and João C. Ferreira. Support for partial run-time reconfiguration of platform FPGAs. *Journal of Systems Architecture*, 52(12):709–726, 2006.

Miguel L. Silva and João Canas Ferreira. Generation of partial FPGA configurations at run-time. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 367–372, 2008. ISBN 978-1-4244-1960-9.

S. Trimberger. A time-multiplexed FPGA. In *Proc. 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines*, pages 22 –28, April 1997.

N. S. Voros, A. Rosti, and M. Hübner, editors. *Dynamic System Reconfiguration in Heterogeneous Platforms*. Springer, 2009.

M. J. Wirthlin and B. L. Hutchings. Improving functional density using run-time circuit reconfiguration. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 6(2):247–256, 1998. ISSN 1063-8210. doi: 10.1109/92.678880.