Moving Run-Time Reconfiguration into the Mainstream

Oliver Diessel
UNSW Asia, Singapore

Overview

1. Outline goals & challenges of Reconfigurable Computing
2. Design flows for Reconfigurable Computing with focus on high-level modeling & synthesis
3. Look at the implementation layer & run-time support
4. Sketch research vision & thrusts intended to make reconfigurable technology more accessible

1. What is Reconfigurable Computing?

- **Use of reconfigurable devices to achieve a benefit over processor-based computing and/or custom devices**
  - Currently involves FPGAs implementing algorithms as digital circuits
  - Look for enhanced performance, reduced power, reduced part count, greater flexibility, greater reliability
  - Small, but expanding niche; conditions most favorable in applications/markets with one or more of following characteristics:
    - The need to **prototype**
    - Move towards **higher levels of integration**
    - **Rapid development/need to support alternative** protocols, standards, algorithms, architectures
    - **Small to medium volume**

Sample Reconfigurable Computing applications

- Network packet processing & sniffing
- Switching
- Encryption
- HD video (de)compression
- Image & video processing
- Signal processing
- Systolic algorithms

Run-Time Reconfiguration (RTR)

- **RTR is the restructuring of a system's hardware components while the system is operating**

- **Also known as *dynamic reconfiguration (DR)***

⇒ **Allows computational structures to be adapted to present need**
⇒ **Enhances flexibility & robustness**
⇒ **Facilitates higher levels of integration**
⇒ **Enhances the functional density of the device**

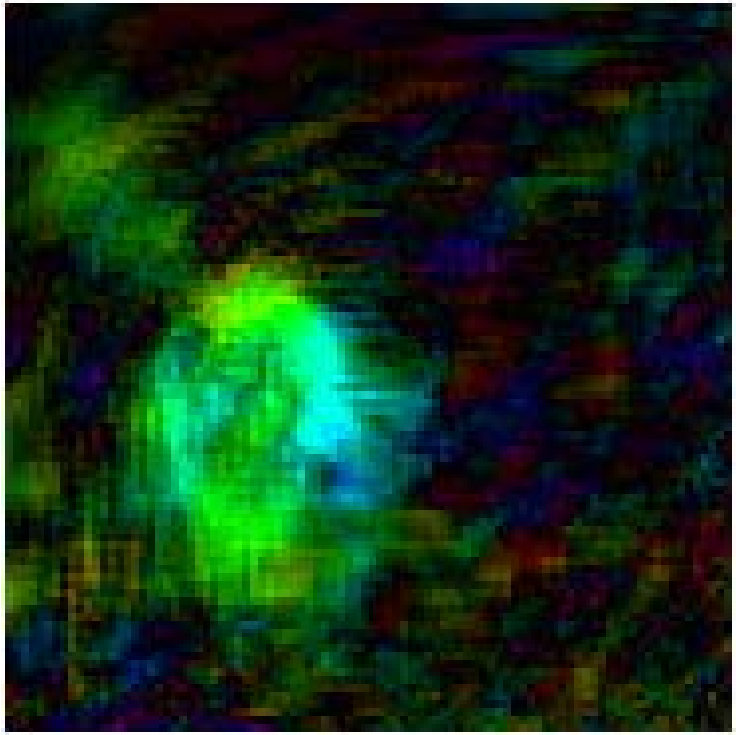
Example: Real-time optical flow computation

- **Implement real-time optical flow algorithms using an FPGA**

- **Why?**
  - Prototype a variety of hardware-based techniques
  - Faster processing = faster movement
  - Multiplex multiple functions onto limited hardware
  - Adapt to changing environment

Optical Flow

- **Determines velocity of pixels from frame to frame**
  ⇒ Closer objects have higher relative velocity

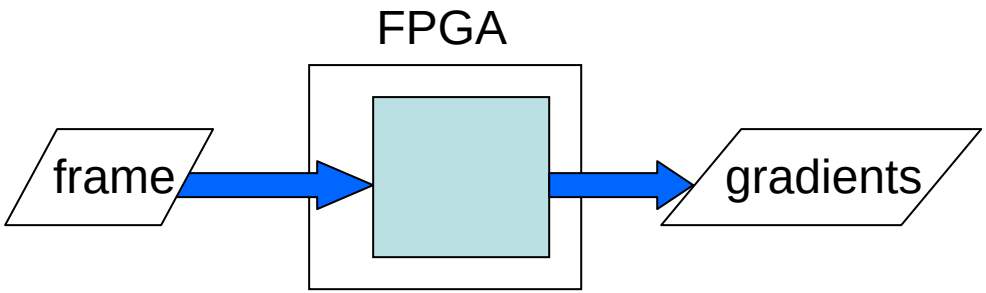


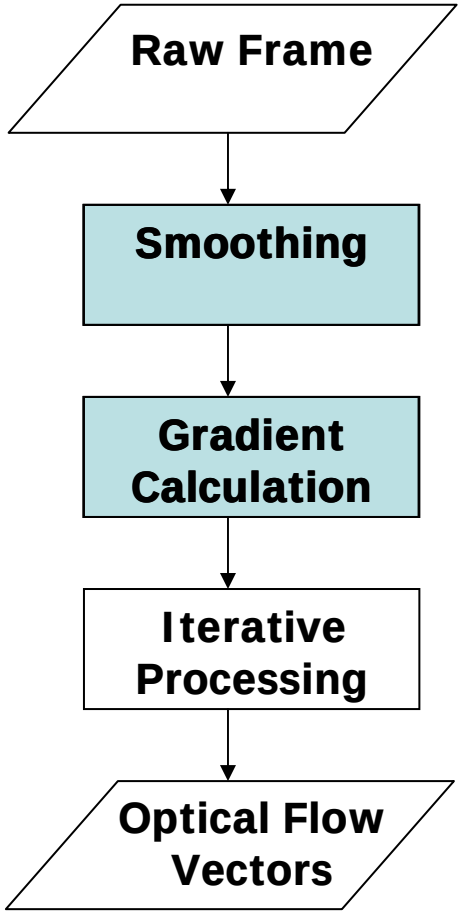
Overview of algorithm & mapping

Core iterative operation:

$$u_i^{k+1} = \frac{\sum_{j \in N(i)} u_j^k - \frac{1}{\alpha}(I_{x,y,i}.v_i^k + I_{x,t,i})}{|N(i)| + \frac{1}{\alpha} I_{y,y,i}}$$

$$v_i^{k+1} = \ldots$$



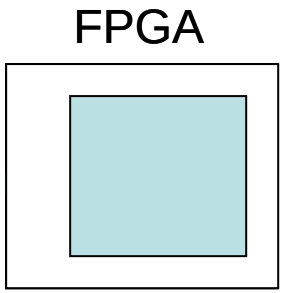


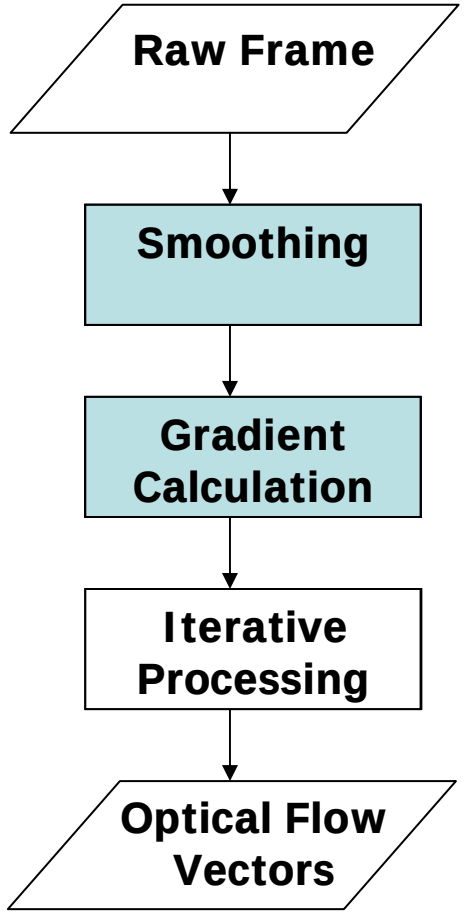
Overview of algorithm & mapping

Core iterative operation:

$$u_i^{k+1} = \frac{\sum_{j \in N(i)} u_j^k - \frac{1}{\alpha}(I_{x,y,i}.v_i^k + I_{x,t,i})}{|N(i)| + \frac{1}{\alpha} I_{y,y,i}}$$

$$v_i^{k+1} = \ldots$$





Example: Adaptive System

- **Change in requirements:**
  - Optical flow → Optical flow + template matching
- **Change in environment**
  - Outdoor navigation → navigate indoors
- **Fault tolerance**
  - Adapt control equations
  - Share additional load

So if RTR is so great, why isn't it being used?

- Lack of compelling applications?

- Lack of tools & support
- Difficult

Features of a Reconfigurable Computing design flow

- **Static:**
  - Support high-level and component modeling using multiple modalities
  - Guide partitioning through understanding of tradeoffs
    - Hardware & software components, interfaces, memory, buses, power, cost
  - Efficient mappings
  - Support co-simulation and co-verification of integrated subsystems
  - Rapid prototyping
- **Dynamic:**
  - As above, PLUS
    - Model dynamism
    - Multiple partitions
      - Active set is event dependent
      - Optimize over all partitions
    - System management
      - Dynamic system

2. High-Level Specification and Synthesis for RC

Goals

- To **simplify** the **specification** of reconfigurable systems
- To **automate** the **generation** of dynamically reconfigurable systems

Approach

- Model dynamic reconfiguration at the hardware level, i.e. capture capabilities of the hardware
- Develop compilation techniques that target these capabilities
- Embed syntactic structures into appropriate languages

Use bottom-up techniques to derive a model of the hardware capabilities & performance that can be exploited top-down via the design flow

Modeling Dynamic Reconfiguration using *process algebra*

- **Model 2 facets of dynamic systems**
  - Behavioral change
    - Change in function as mediated by change in logic
  - Structural change
    - Change in composition as mediated by change in interconnection
- **In a process algebra**
  - Behavioral change equates to process evolution – transition from one state to another
  - Structural change equates to dynamic composition – composition guarded by some event

[Milne, 1999]

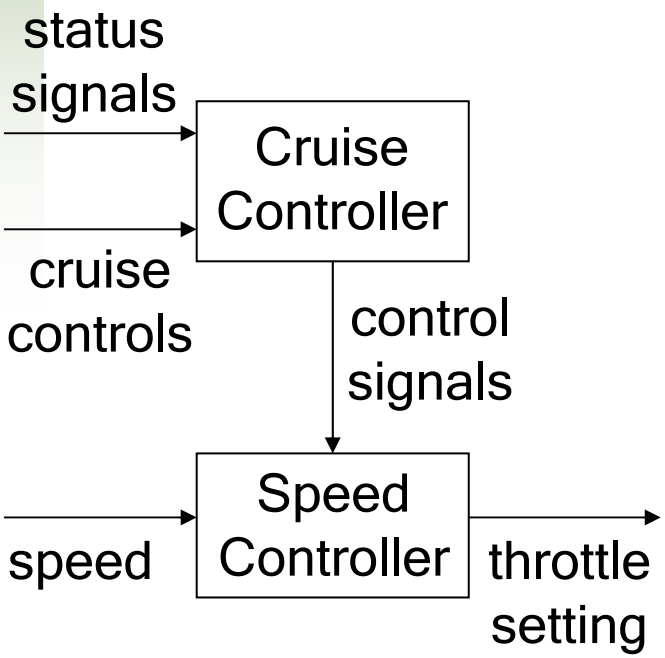
Progress to date

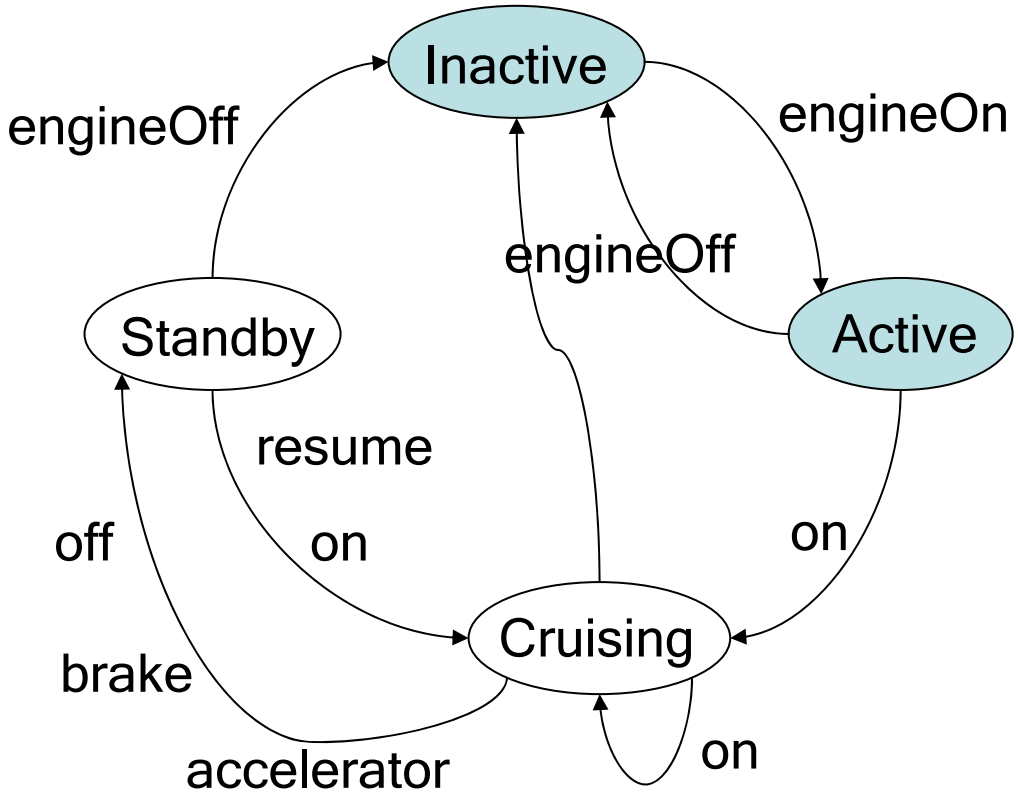
- **Process descriptions mapped to hardware structures via syntax-directed translation**
  - Process behaviours implemented as FSMs in compact logic blocks
  - Hierarchical design achieved through event abstraction and local process synchronisation

- **Interpret specifications at run time, and dynamically reconfigure process logic to cope with limited chip area**

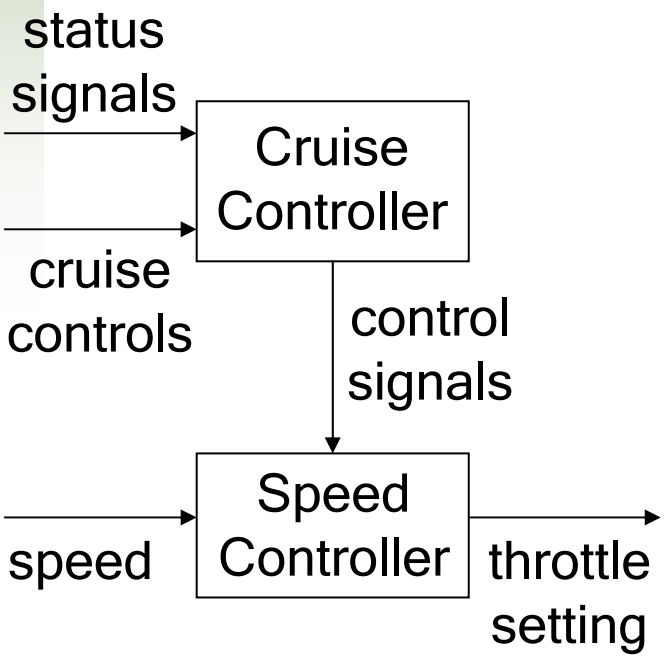
[Diessel & Milne, 1999][Malik, So & Diessel, 2002]

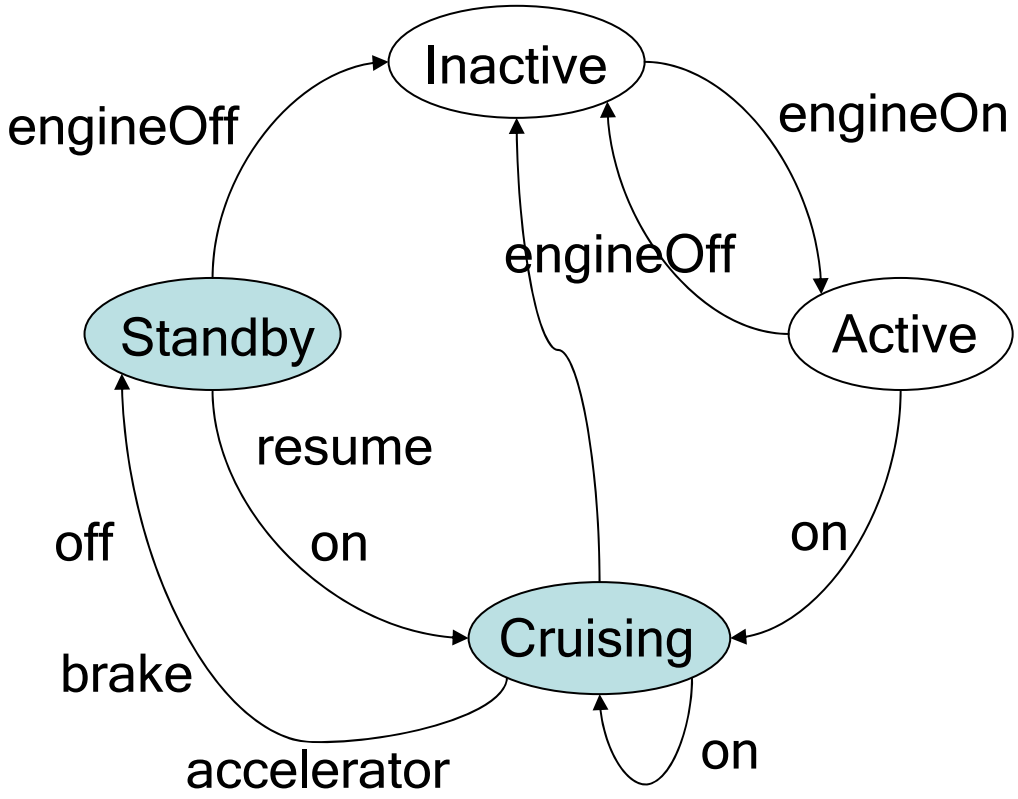
Example: Car Cruise Control – Initial configuration





Example: Car Cruise Control – Final configuration





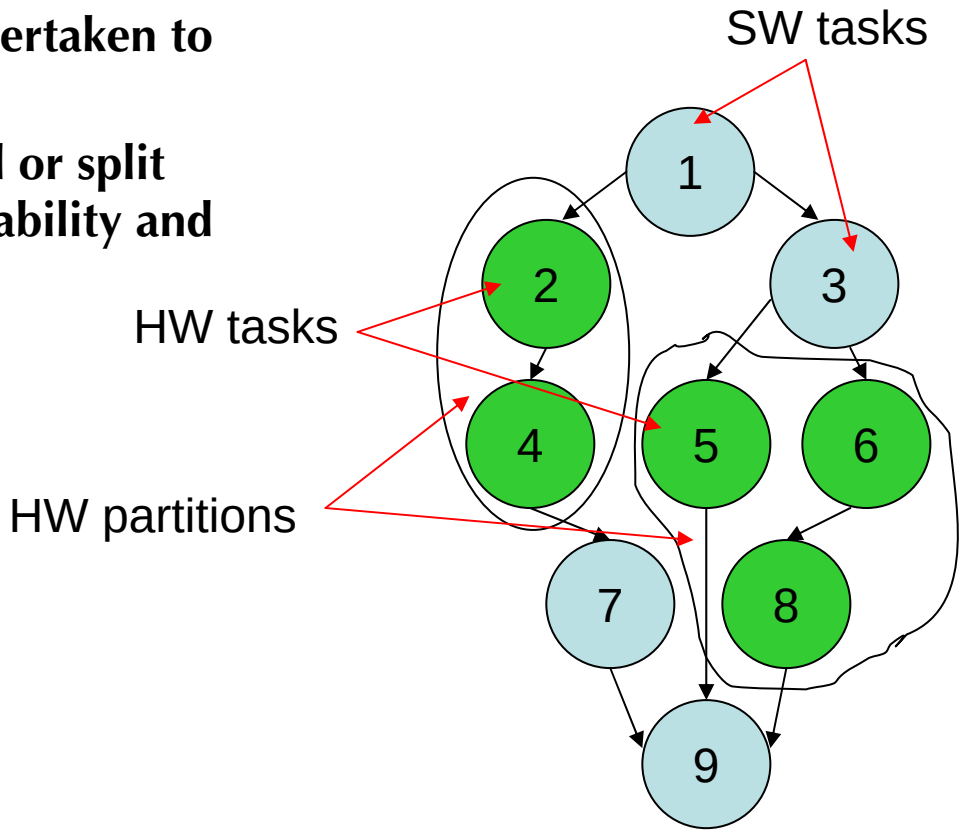
Applications

- **Implementing time-varying control strategies**
  - Mode switching
- **Adjusting to available resources**
  - Multi-tasking
  - Graceful degradation
- **Coping with dynamic updates**
  - User customizes system by selecting web-accessible modules

Basing the system on a formalism such as a process algebra aids validation and verification

Model definition: Tasks

- Application modeled as a task graph
- Tasks represent functional modules
- Task graph partitioning undertaken to allocate tasks to HW or SW
- HW partitions are clustered or split according to resource availability and capability

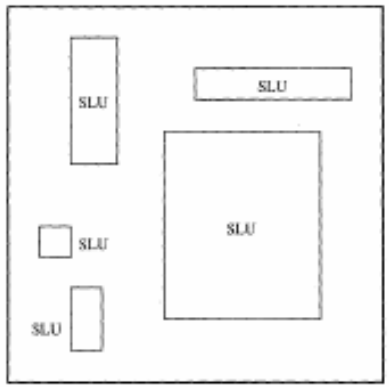


3. Implementation model: Swappable Logic Unit

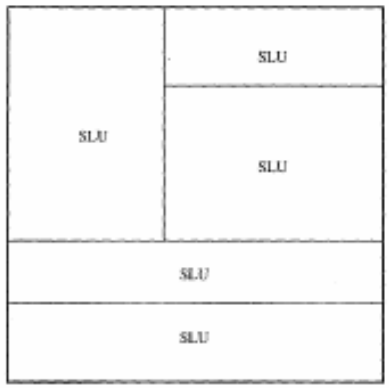
- **2 models:**
- **Sea of accelerators:**
  - Logic flexibility
  - Performance less compromised
  - Potential for high utilisation
  - Problems with fragmentation
  - Problems routing
- **Parallel wiring harness**
  - Ease of placement
  - Known delays
  - Lower performance
  - Reduced utilisation

[Brebner, 1996]

Sea of accelerators

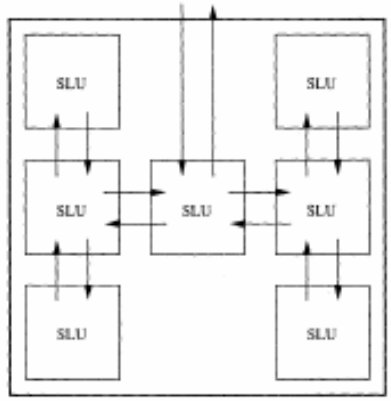


(a) Sea with five accelerators present, placed fairly randomly

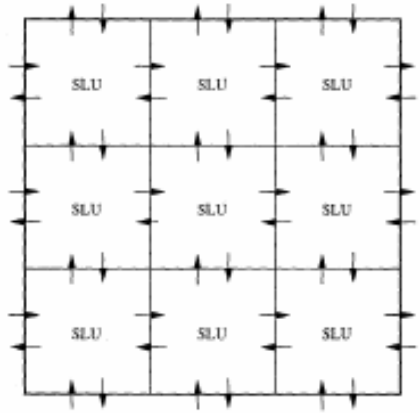


(b) Sea with five accelerators present, densely packed

Parallel wiring harness



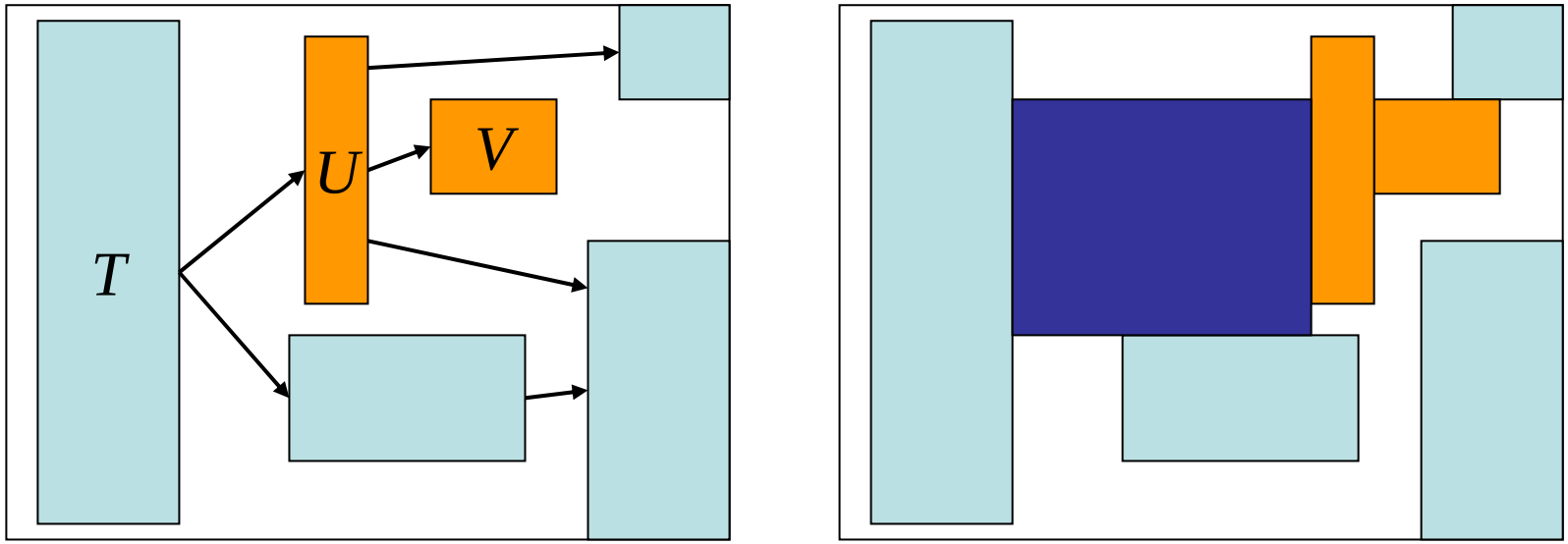
(a) Parallel harness with extra H-tree wiring supplied



(b) Parallel harness with no extra wiring needed

Ordered compaction deals with fragmentation

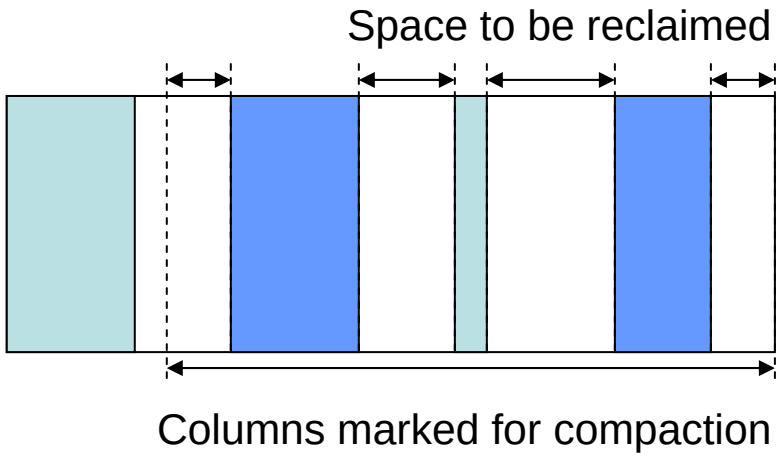
- "Slide" tasks along rows of FPGA cells to free space for incoming task

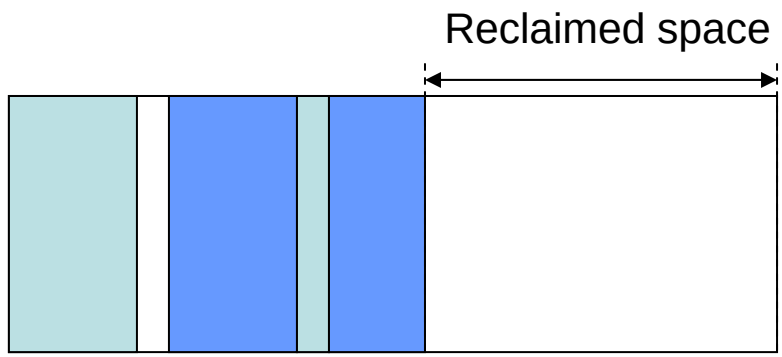


[Diessel & ElGindy, 1997]

Logic-based compaction

- **Ordered compaction frees required space by squeezing a subset of the tasks together**
- **Requires following enhancements:**
  - marking method: shorten pattern as space found
  - compaction method: reloading usually proposed



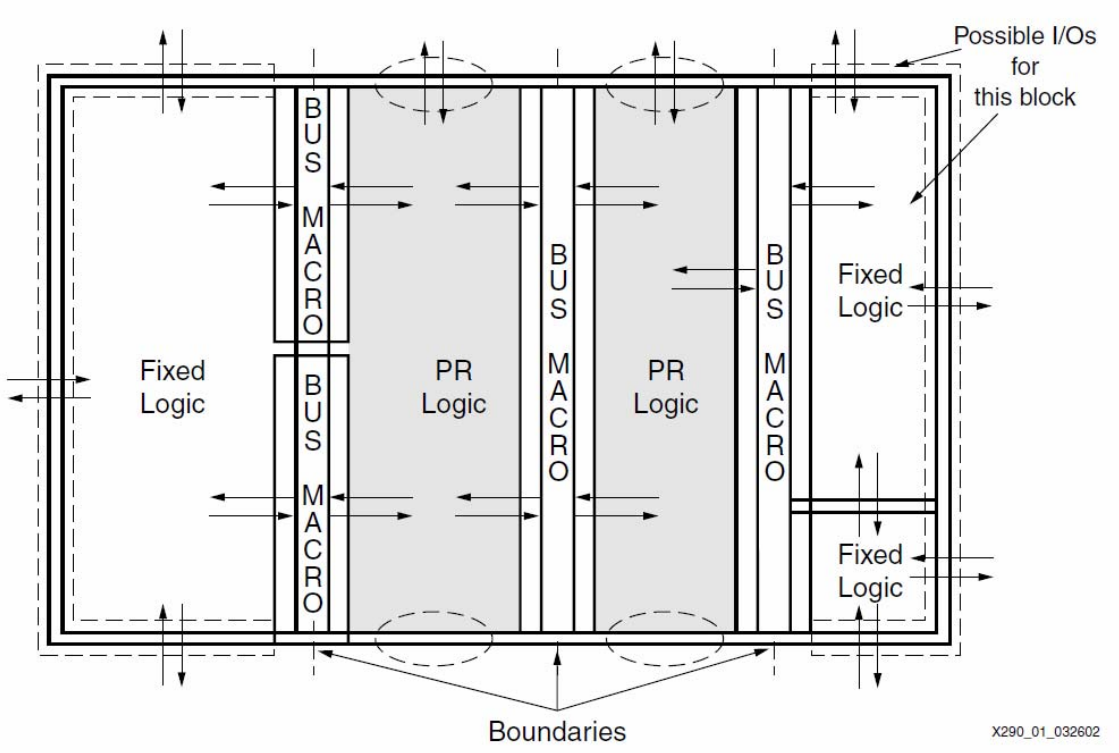


Before compaction

After compaction to left

[Brebner & Diessel, 2001]

Early communications approaches



[Xilinx XAPP290, 2003]

AMBA-based

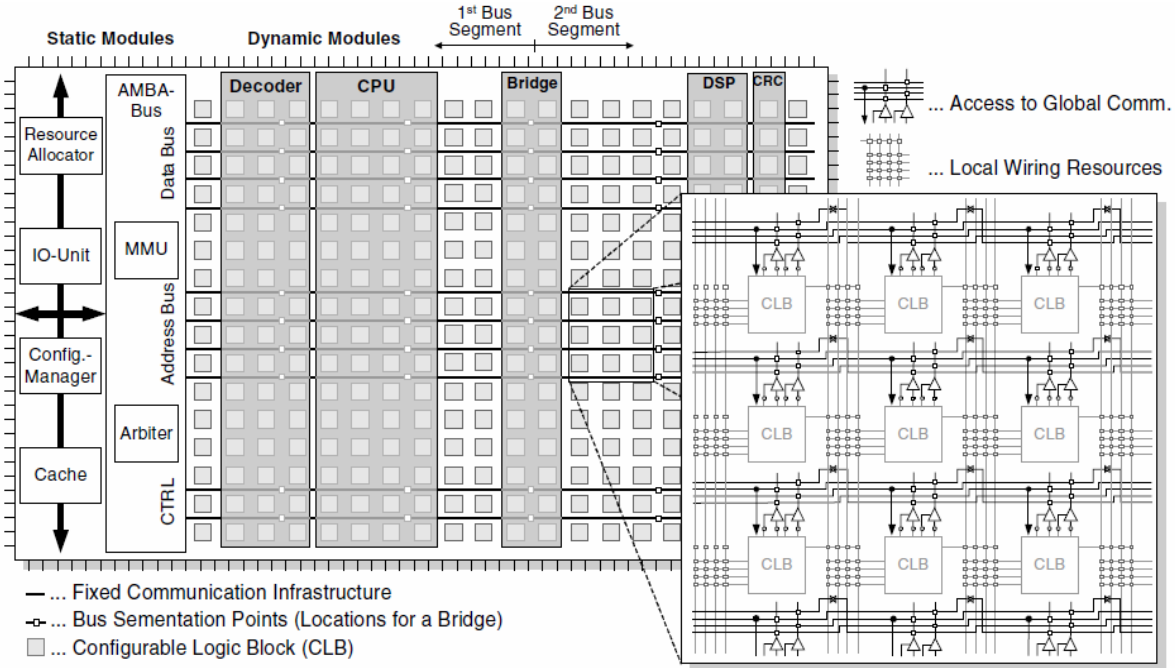
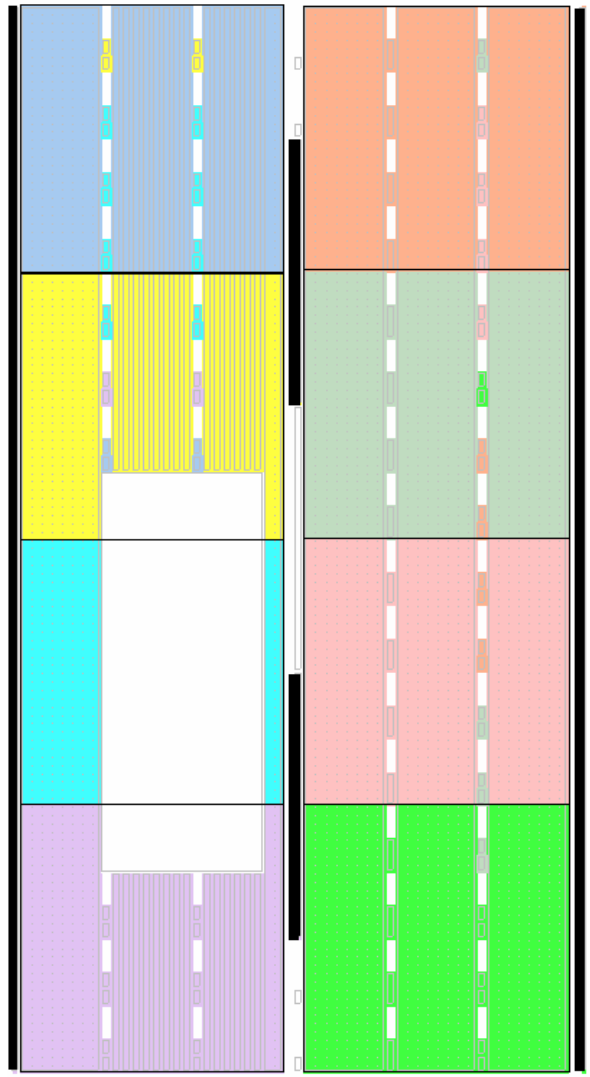


Fig. 5. Overview of the Approach

[Kalte *et al.*, 2004]

The COMMA Approach
Module Placement

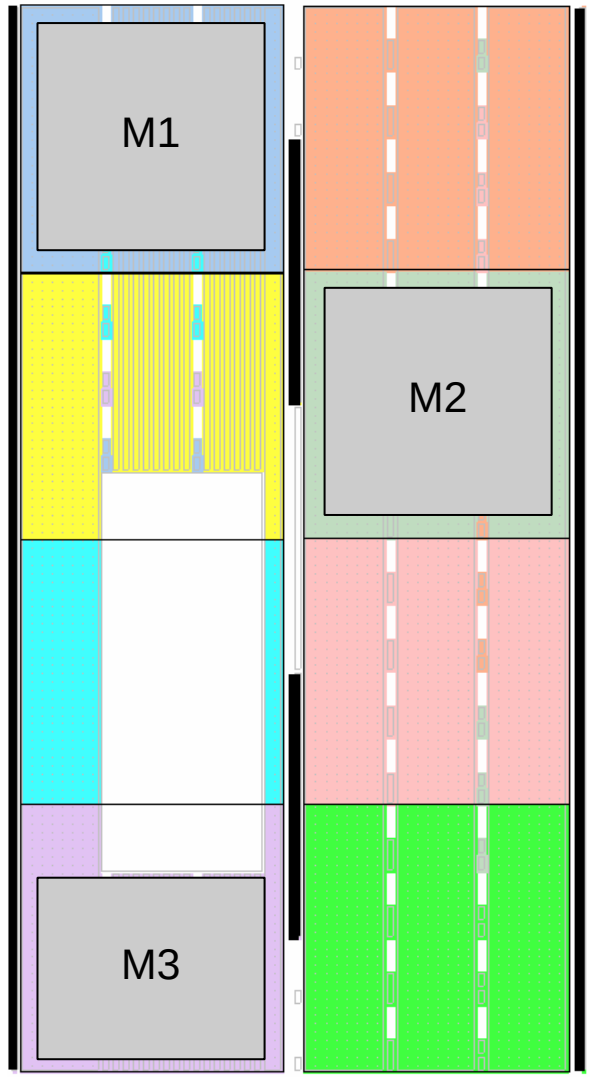
- **Reference target device – the Virtex-4**
  - 41-word vertical frame length (16 CLBs high)
  - External I/Os on sides and middle
    - I/Os on sides may not necessarily be on the periphery



[Koh & Diessel, 2005]

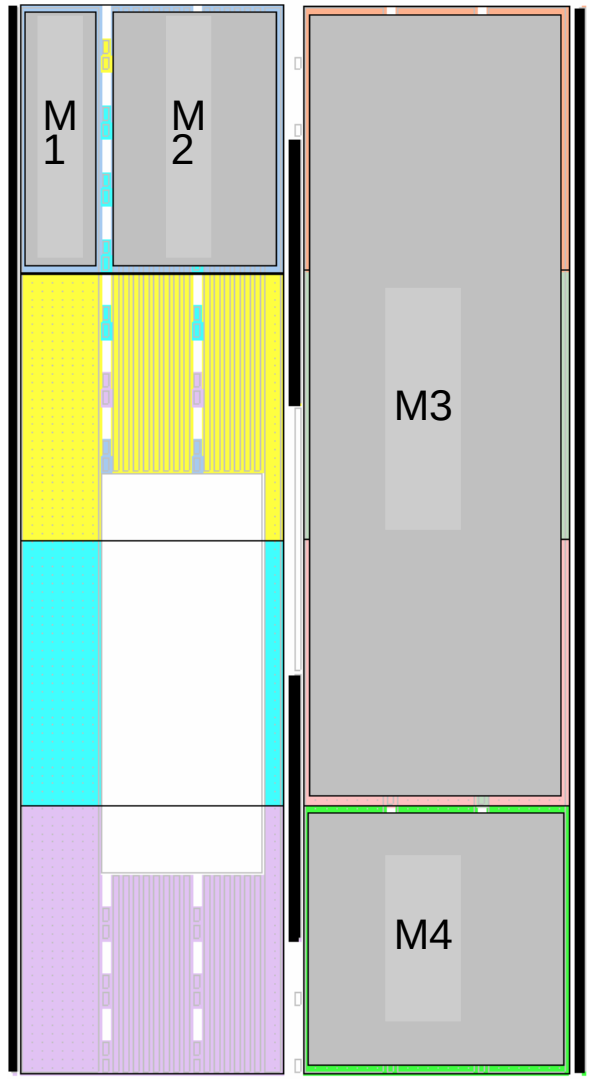
The COMMA Approach
Module Placement

- One module to be placed in each of these natural "slots"
- Can be reconfigured independently



The COMMA Approach
Module Placement

- **Slots may be subdivided**
  - Accommodates more modules
- **May also be aggregated**
  - Accommodates larger modules



The COMMA Communications Challenge

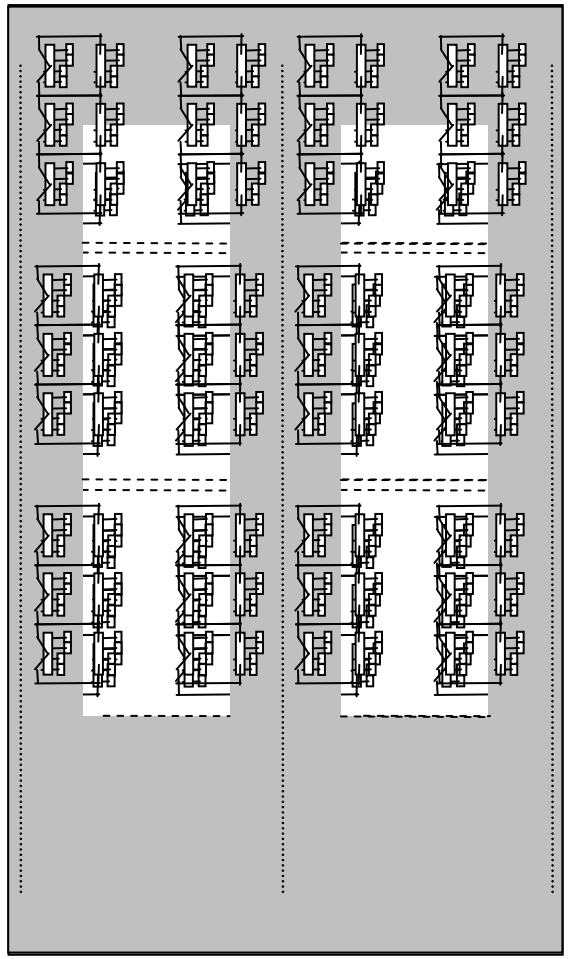
- **Assemblies of dynamically swapped modules have dynamic communication needs**
  - Different protocols
  - Different communication patterns
  - Different ports: Communication with different modules over time
  - Varying bandwidths: Port sizes
- **To minimise reconfiguration overheads, module-based reconfiguration needs to be supported with a communications infrastructure that supports:**
  - Different interfaces, behaviour and timing of modules
  - Management of dynamic reconfiguration i.e. pin reassignments and wire reuse/switching

We're focusing on providing the interconnecting wires and minimising the overheads of reconfiguring the interconnect

The COMMA Approach
Wiring Harness

- Supports arbitrary inter-pin interconnection
- Allows modules and wiring infrastructure to be reconfigured independently

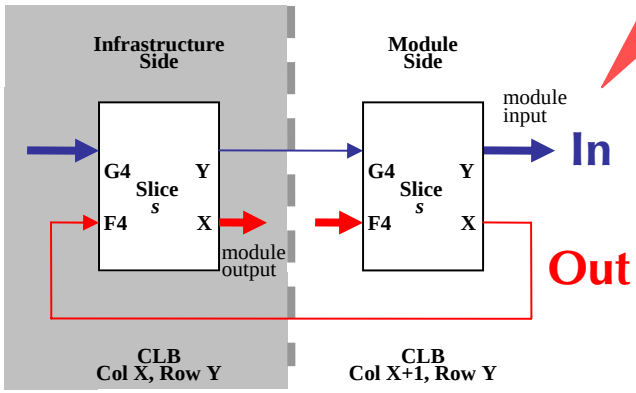
Infrastructure is tailored to the application requirements

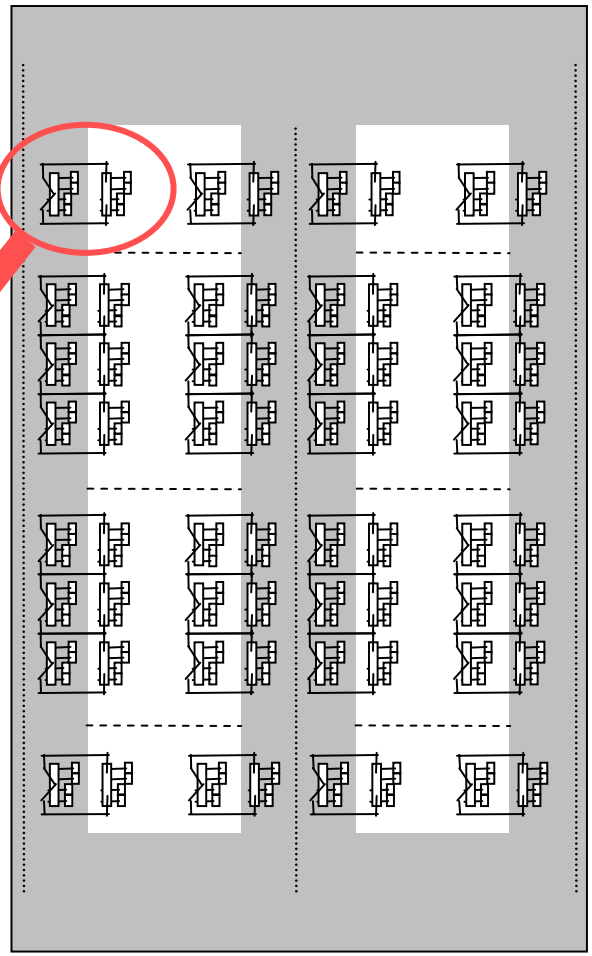


The COMMA Approach
Pin Virtualisation – Slice Macros

- Slice macros straddle module and infrastructure boundaries
- Support *arbitrary* one-pin and two-pin IO combinations

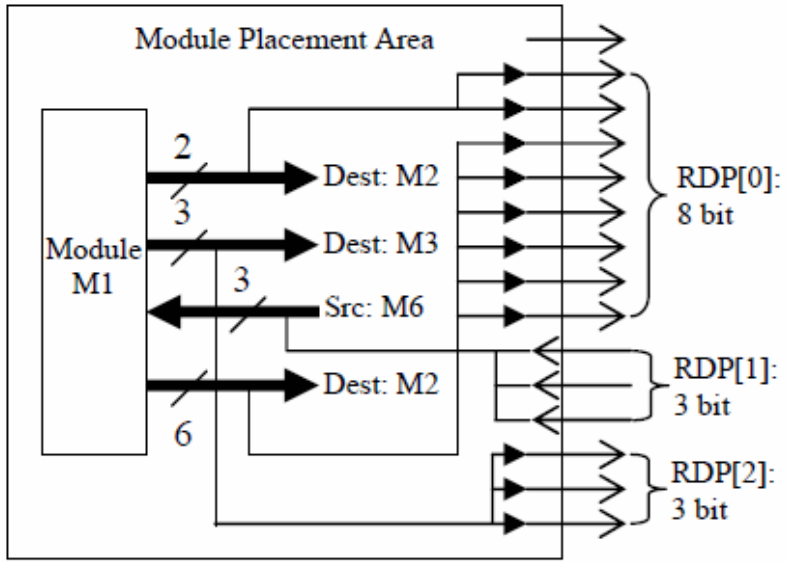
1 input and 1 output:



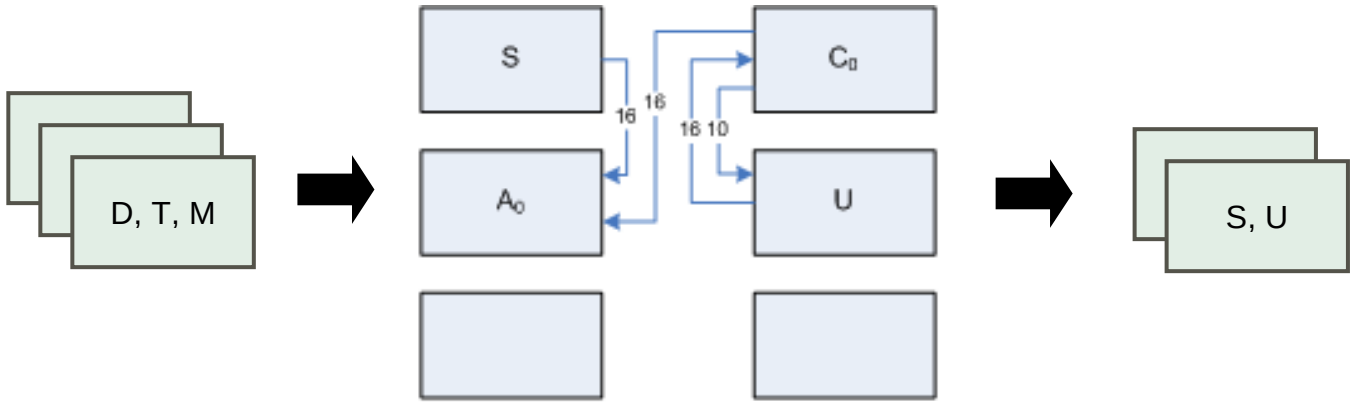


The COMMA Approach
Pin Virtualisation – "Reconfigurable Data Ports"

- Map module pins to slice macro pins
- Perform necessary multiplexing or demultiplexing between multiple pins
- Implemented as simple module adapters

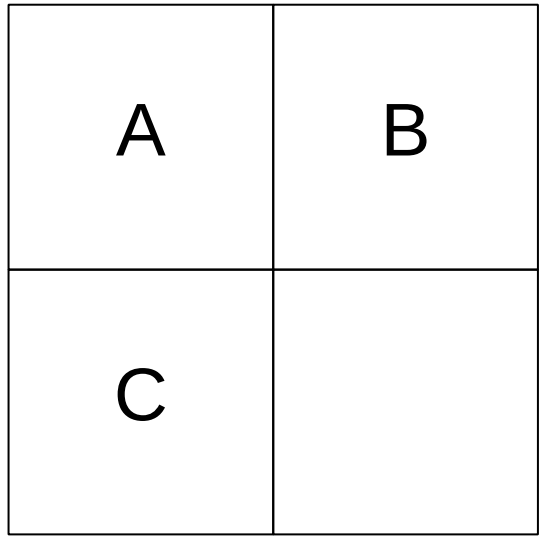


Optimisation problem



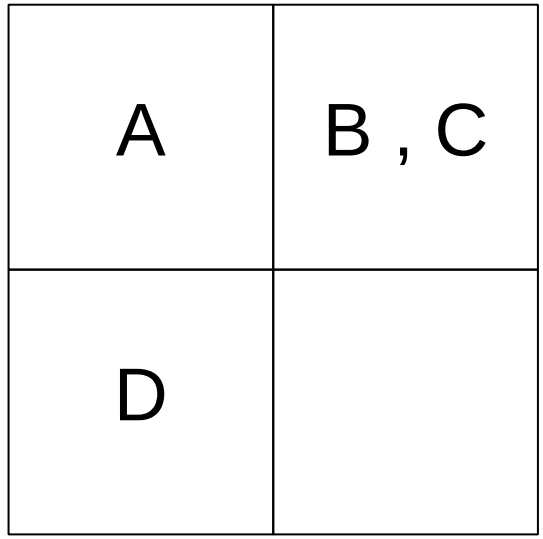
A Hierarchy of Reconfigurable Modular Systems

- **Static**
  - Module arrangement and communication patterns fixed at design time



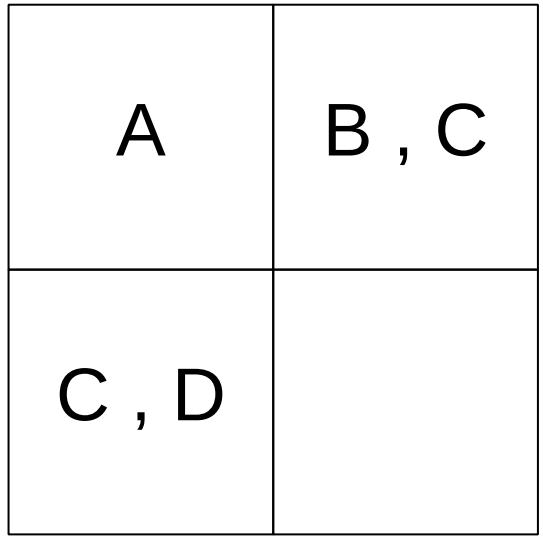
A Hierarchy of Reconfigurable Modular Systems

- Static
- **DR1**
  - Modules are swapped at run-time
  - Communications patterns known at design-time



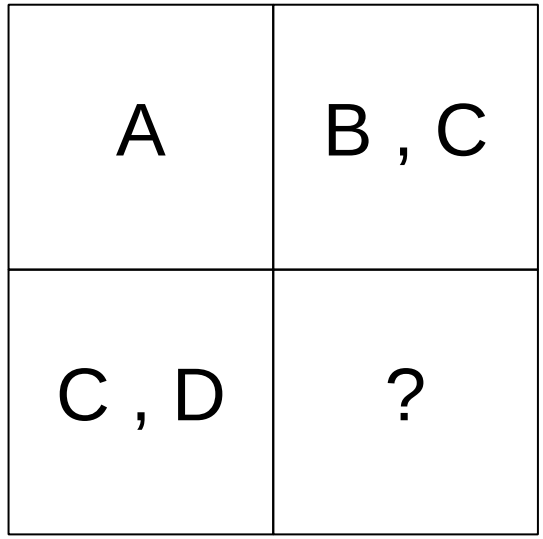
A Hierarchy of Reconfigurable Modular Systems

- Static
- DR1
- **DR2**
  - Module placement may not be known at design-time
  - May occur when the order in which modules are placed is unknown



A Hierarchy of Reconfigurable Modular Systems

- Static
- DR1
- DR2
- **DR3**
  - An unknown module may be dynamically placed at run-time



4. Research Vision

- **Minimize the barriers and reduce costs of using reconfigurable hardware**
- **Bridge the gap between vendors and end users**

- **Integrate reconfigurable devices into mainstream design flows for embedded and high performance systems by assisting in**

  - ***Developing coherent sets of tools for defining and elaborating the space of hardware configurations covered by a proposed system***

  - ***Developing effective run-time support methods that can be automatically generated***

  - ***Developing benchmarks that allow improvements in techniques, algorithms and devices to be measured***

Thrust 1 – Design Exploration

- **Design exploration tools that allow RTR to be rapidly modelled and assessed at a high level**
  - When does reconfigurable hardware confer a benefit over software? What are the performance requirements?
  - Capture the triggers of reconfiguration
  - Scope the complexity of hardware configurations and understand the timing requirements; can the overheads be managed?
  - How is power to be managed?
  - What about $$$ costs?
  - Rapid elaboration of system architecture

Thrust 2 – Synthesis

- **Synthesis tools**
  - Optimize across hardware configurations (partitions)
    - Minimize area & power; maximize performance of individual configurations
    - Minimize reconfiguration overheads (delay, energy, buffer size)
  - Automate the provision of supporting run-time infrastructure
    - Controllers, OS,

Thrust 3 – Validation and Verification

Conclusion

- RTR promises better performance for less cost
- Industry appears to be poorly supported in making use of the technology
- … perhaps that is why compelling applications are hard to find
- More integrated tools that assist with exploration, synthesis, and verification of dynamically reconfigurable systems are needed
- Now is the time to work on these