Scheduling tasks on Reconfigurable FPGA architectures

Mauro Marinoni

ReTiS Lab, TeCIP Institute

Scuola superiore Sant'Anna - Pisa

Agenda

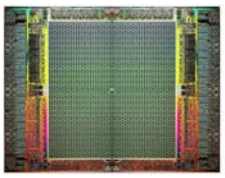
- The topics that will be addressed are:
  - Overview on basic characteristics of the FPGA;
  - FPGA reconfiguration capabilities;
  - Timing analysis for reconfigurable FPGA platforms;
  - Kernel mechanisms to support reconfigurable systems.

FPGA

Overview

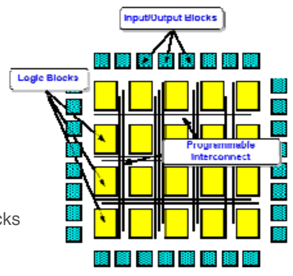
Definition

- A *field-programmable gate array* (**FPGA**) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable". The FPGA configuration is generally specified using a *hardware description language* (**HDL**).



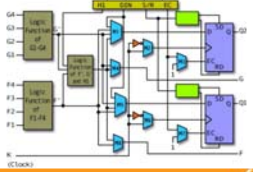
General structure

- 2-D array of **logic blocks** with electrically programmable **interconnections**
- They provide:
  - Configurable logic blocks (CLB)
  - Connection lines
  - Interconnection matrixes
  - Custom blocks
- User can configure:
  - Intersections between logic blocks
  - The function of each block



Characteristics of the CLB

- These blocks contain the logic for the FPGA. It contains:
  - enough logic to create a small state machine
  - RAM enough for creating arbitrary combinatorial logic functions, also known as lookup tables (LUTs)
  - flip-flops for clocked storage elements
  - multiplexers in order to route the logic within the block and to and from external resources



Pros and Cons

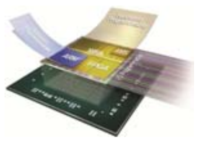
- **Advantages**
  - Performance: Online analysis of high-rate data streams
  - Reliability: Deterministic hardware dedicated to every task
  - Reconfigurability: Nonrecurring engineering expenses
  - Durability: Radiation Hardened and Program Integrity
  - Time to market: Flexible and rapid prototyping and debugging
- **Drawbacks**
  - Lower working frequencies
  - Higher power consumption
  - Higher cost

Performances

- FPGAs excel at computing **non-data** dependent algorithms **in parallel**.
- **Customizable data path** and ALU allow very large amounts of data to be transferred and computed within several clock cycles.
- Despite **lower clock frequencies**, FPGA's can outperform conventional CPU's on certain data processing tasks

Integration with microprocessors

- In order to provide an execution environment to those tasks **not fitted** for the FPGA execution paradigm
  - **Soft-core**: a microcontroller wholly implemented inside the FPGA (*NIOS II*)
  - **System on Chip (SoC)**: integrates a microcontroller and an FPGA inside a single chip (*Zynq*)



Programming technologies

- Fuse and anti-fuse:
  - fuse makes or breaks link between two wires
  - smaller and faster
  - one-time programmable
- Flash:
  - high density
  - dedicated production process (*in the past…*)
- RAM-based:
  - memory bit controls a switch that connects/disconnects two wires
  - can be programmed and re-programmed easily (using bitstreams)
  - standard technology
  - volatile SRAM memory

RAM-based programming

- Initially seen as a drawback imposing an initialization phase
- the volatility of SRAM-based FPGAs is **not a liability**, but was in fact the key to many new types of applications.
- the programming of such an FPGA could be changed by a completely electrical process…

INSTITUTE OF COMMUNICATION, INFORMATION AND PERCEPTION TECHNOLOGIES

Scuola Superiore Sant'Anna

Retis Real-Time Systems Laboratory

RECONFIGURABLE COMPUTING

Characteristics

FPGA reconfiguration

- While the previous uses of FPGAs still treat these chips purely as methods for implementing **digital logic**, the reprogrammability of modern FPGAs allows to download algorithms onto the FPGAs and **change** these algorithms just as general-purpose computers can change programs.

Selecting a Target FPGA

Characteristics of the different reconfiguration approaches:

- Granularity
- Dynamic reconfigurability
- Partial vs Full reconfiguration
- Reconfiguration time

"Fine-grain Dynamic Partial Reconfigurable devices"

Granularity

Two main architectures:

- **Course grained**: consists of small number of large logic blocks
  - Small bitstream is required to configure them (low config. time)
  - Faster because of easy routing
  - Less complexity and less flexibility
- **Fine grained**: consists of large number of small logic blocks
  - Customization at the bit level => Greater flexibility & more complexity
  - Large bitstream is required to configure them (high config. time)
  - Easy conversion to ASIC
  - Less speedy

Dynamic reconfigurability

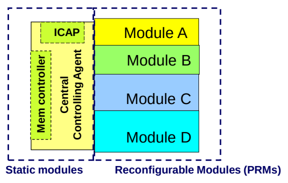
- It is the ability of a FPGA to modify operation during **runtime**
- The primary advantages of runtime reconfiguration in devices
  - Power/Size/Cost Reduction
  - Hardware reuse and flexibility
  - Application Portability
- The disadvantage of dynamic reconfiguration
  - Suffer from the time needed to load the configuration bitstream **before** starting its execution

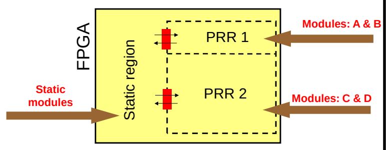
Partial vs Full reconfiguration

- Partial Reconfiguration (PR) allows the ability to **reconfigure a portion** of an FPGA
- It allows for critical parts of the design to continue operating while loading a partial design into a reconfigurable **module**
- Reconfiguration time of partial reconfiguration is much smaller (~4-5 ms) than full reconfiguration(~12 ms)
- Wide variety of dynamically reconfigurable FPGA devices available in the market offer PR today
  - Lattice ORCA Architecture
  - Atmel AT40K Architecture
  - Altera APEX 20K
  - Xilinx Virtex FPGAs

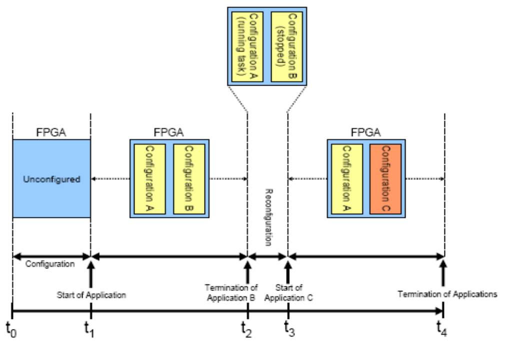
Partial Reconfiguration

- Partial reconfiguration (PR) allows the ability to reconfigure a portion of an FPGA
- Real advantages arise when PR is done during runtime also know as **dynamic partial reconfiguration**
- Dynamic Reconfiguration allows the reconfiguration of a portion of an FPGA while the remainder continues operating without any loss of data
- Two types of Regions
  - Static – Keeps operating
  - Reconfigurable – Can be reconfigured with a new module





Partial Reconfiguration - Timing



How to choose the FPGA model?

- Exploitation of Partial Reconfiguration for a design requires significant knowledge on targeted device
  - An evaluation of the performance and limitations of your selection is required
  - Also the support provided by design tools must be evaluated
- Correct FPGA selection matters!
- Xilinx's FPGA's (the Virtex Family is a widespread solution) is chosen as the example FPGA

Xilinx Virtex-II/Pro Architecture

- Composed of a fine-grain 2D heterogeneous array that includes
  - Configurable logic blocks (CLBs)
  - Memory blocks (BRAMs)
  - DSP units (MULTs)
  - I/O blocks (IOBs)
  - FIFOs buffers
- Each CLB contains LUTs, FFs, Gates & Multiplexers that can be configured to implement any design efficiently.
- This 2D array can configured either externally or internally.

Basic Xilinx FPGA Layout

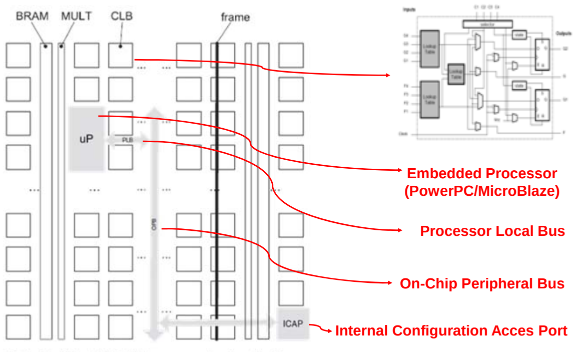

BRAM
MULT
CLB
frame
uP
PLB
OPB
ICAP
Embedded Processor (PowerPC/MicroBlaze)
Processor Local Bus
On-Chip Peripheral Bus
Internal Configuration Acces Port

Fig. 1. The Virtex II-Pro FPGA resources and configuration frames.

Xilinx FPGA Layout – Inside CLBs

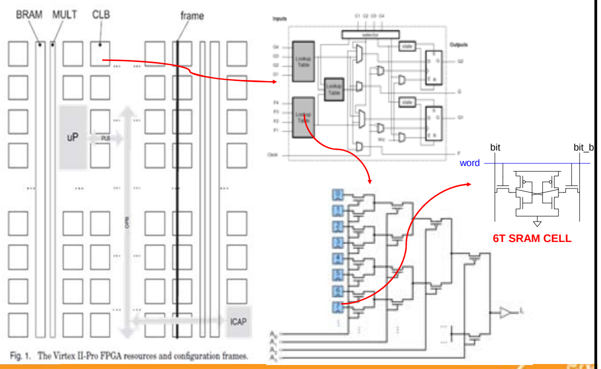


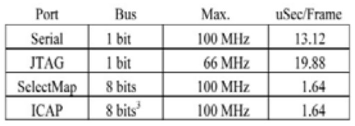
Fig. 1. The Virtex II-Pro FPGA resources and configuration frames.

Xilinx Virtex – Reconfiguration

- FPGA is reconfigured by writing bits into Configuration Memory (CM).
- CM is arranged in vertical frames (1bit wide) stretching from top to bottom.
- So Configuration data is organized into frames that target specific areas of the FPGA through frame addresses.
- To reconfigure any portion of that frame the partial bitstream contain configuration data for a whole frame.

Xilinx Virtex – Reconfiguration (2)

- The (device) array can be configured:
  - **Externally**
    - Serial Peripheral Interface (SPI) port
    - JTAG (Boundary Scan) port (serial)
    - SelectMap port (parallel)
  - **Internally**
    - Internal configuration access port (**ICAP**) which allows for **(internal) partial configuration** only.



Port	Bus	Max.	uSec/Frame
Serial	1 bit	100 MHz	13.12
JTAG	1 bit	66 MHz	19.88
SelectMap	8 bits	100 MHz	1.64
ICAP	8 bits^{3}	100 MHz	1.64

*Table: The configuration speeds of 4 input ports.

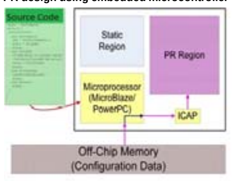
Xilinx Virtex – Reconfiguration (3)

- **ICAP details:**
  - Operational Frequency:
    - 100 MHz
  - Width Size:
    - 8 bits (Virtex-II Pro)
    - 16/32 bits (Virtex-4 and Virtex-5)
  - ICAP BRAM
    - Caches the configuration bits before being loaded into the config. Memory.
  - Connection with the bus:
    - OPBHWICAP (IP Core in Virtex II/Pro)
    - XPSHWICAP (IP Core in V-4 and V-5) (Lower latency due to connection to the PLB bus)

PR using an Embedded Microcontroller

- Reconfiguration Steps
  - Reconfiguration is triggered within the FPGA
  - Processor core loads the desired configuration data from external non-volatile memory.
    - This could be from ROM, PROM, Flash, SPI Flash loaded at startup or filled up by the FPGA itself
  - Processor reconfigures the PR region through the reconfigurable controller (ICAP primitive)
    - OPBHWICAP or XPSHWICAP (v)
    - Customized reconfigurable controllers (c)

PR design using embedded microcontroller


Source Code
Static Region
PR Region
Microprocessor (MicroBlaze/ PowerPC)
ICAP
Off-Chip Memory (Configuration Data)

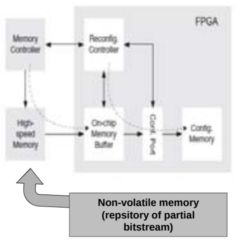
- Reconfiguration Time (RT):
  - Time required to pull the bitstream from
    - off-chip memory -> Local memory of processor
    - Local memory of processor -> ICAP
    - ICAP -> FPGA configuration region (PR Region)

- **RT Dependencies**
  - Storage type
  - Controller type
  - ICAP Premitive

Partial Reconfiguration (in detail)

- Reconfiguration throughput
  - To evaluate the system holistically, the overhead added by all the system components involved in the reconfiguration process should be considered
  - The flow of data across different components should also be taken into account for the said purpose

- Reconfiguration in detail
  - Memory Controller is instructed to load the partial bitstream.
  - Bitstreams is copied from off-chip memory to On-chip memory buffer.
  - Reconfig.Controller loads the bitstream to the FPGA Config. Memory through Conf.Port.
  - These phases occur in succession untill the entire bitstream is copied.



A Survey of the Reconfiguration Times (RT)

(with different used technologies)

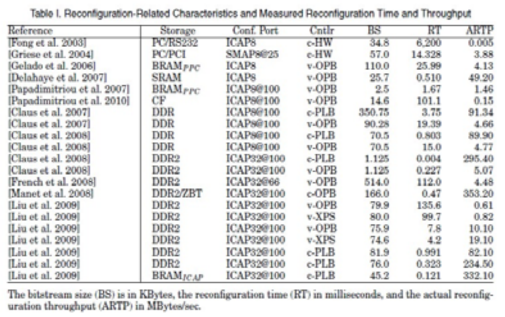


Table I. Reconfiguration-Related Characteristics and Measured Reconfiguration Time and Throughput

Reference	Storage	Conf. Port	Cntlr	BS	RT	ARTP
[Fong et al. 2003]	PC/RS232	ICAP8	c-HW	34.8	6.200	0.005
[Griese et al. 2004]	PC/PCI	SMAP8@25	c-HW	57.0	14.328	3.88
[Gelado et al. 2006]	BRAM_{PPC}	ICAP8	v-OPB	110.0	25.99	4.13
[Delahaye et al. 2007]	SRAM	ICAP8	v-OPB	25.7	0.510	49.20
[Papadimitriou et al. 2007]	BRAM_{PPC}	ICAP8@100	v-OPB	2.5	1.67	1.46
[Papadimitriou et al. 2010]	CF	ICAP8@100	v-OPB	14.6	101.1	0.15
[Claus et al. 2007]	DDR	ICAP8@100	c-PLB	350.75	3.75	91.34
[Claus et al. 2007]	DDR	ICAP8@100	v-OPB	90.28	19.39	4.66
[Claus et al. 2008]	DDR	ICAP8@100	c-PLB	70.5	0.803	89.90
[Claus et al. 2008]	DDR	ICAP8@100	v-OPB	70.5	15.0	4.77
[Claus et al. 2008]	DDR2	ICAP32@100	c-PLB	1.125	0.004	295.40
[Claus et al. 2008]	DDR2	ICAP32@100	v-OPB	1.125	0.227	5.07
[French et al. 2008]	DDR2	ICAP32@66	v-OPB	514.0	112.0	4.48
[Manet et al. 2008]	DDR2/ZBT	ICAP32@100	c-OPB	166.0	0.47	353.20
[Liu et al. 2009]	DDR2	ICAP32@100	v-OPB	79.9	135.6	0.61
[Liu et al. 2009]	DDR2	ICAP32@100	v-XPS	80.0	99.7	0.82
[Liu et al. 2009]	DDR2	ICAP32@100	v-OPB	75.9	7.8	10.10
[Liu et al. 2009]	DDR2	ICAP32@100	v-XPS	74.6	4.2	19.10
[Liu et al. 2009]	DDR2	ICAP32@100	c-PLB	81.9	0.991	82.10
[Liu et al. 2009]	DDR2	ICAP32@100	c-PLB	76.0	0.323	234.50
[Liu et al. 2009]	BRAM_{ICAP}	ICAP32@100	c-PLB	45.2	0.121	332.10

The bitstream size (BS) is in KBytes, the reconfiguration time (RT) in milliseconds, and the actual reconfiguration throughput (ARTP) in MBytes/sec.

Partial Reconfiguration Phases (Cost Model)

PowerPC/Microblaze is currently used as the embedded processor to controll the reconfiguration.

Reconfiguration Process is carried out in three distinct phases.

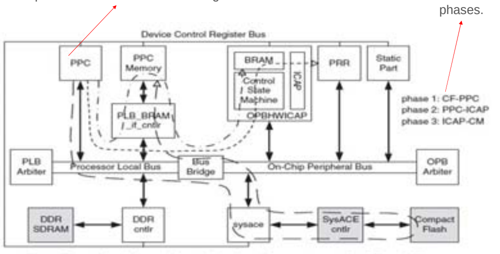
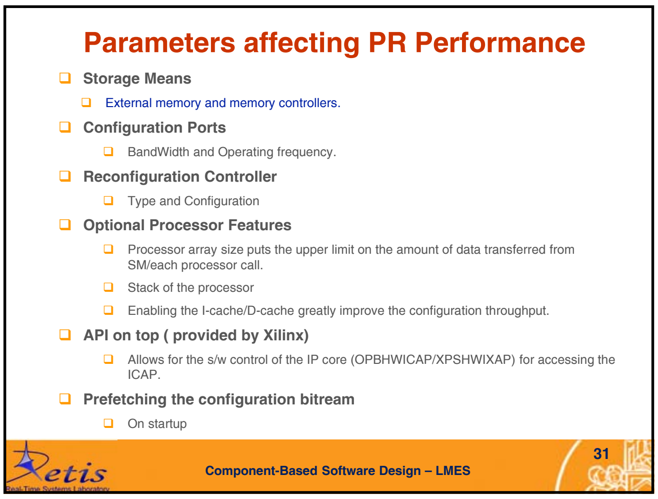


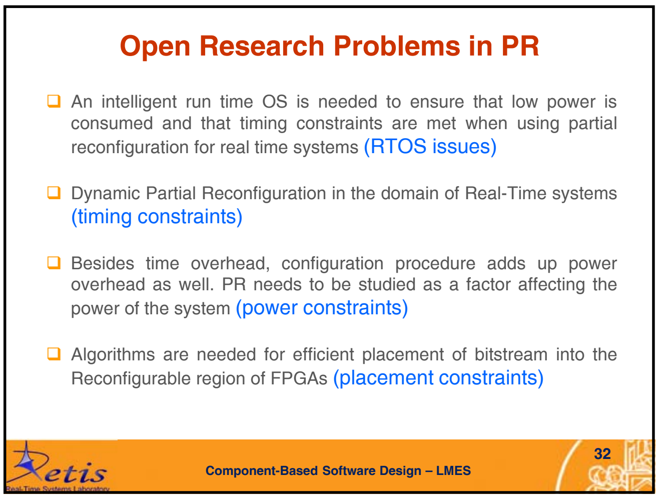
Fig. 3. Flow of partial reconfiguration in the reference system with a Virtex-II Pro.

(Cost Model) $RT = RT_{SM-PPC} + RT_{PPC-ICAP} + RT_{ICAP-CM}$



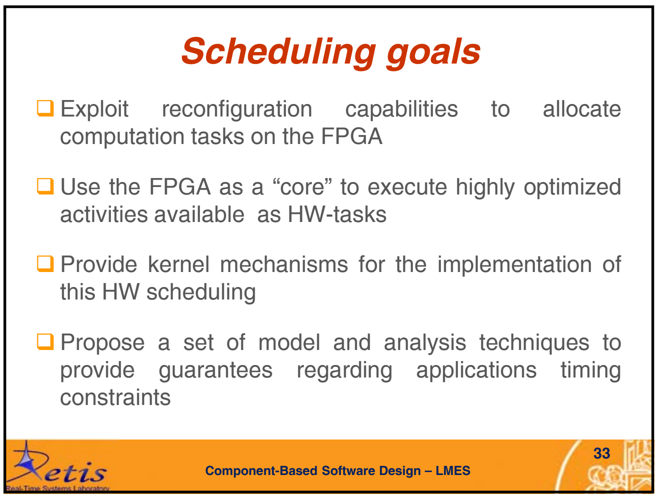
Parameters affecting PR Performance

- **Storage Means**
  - External memory and memory controllers.
- **Configuration Ports**
  - BandWidth and Operating frequency.
- **Reconfiguration Controller**
  - Type and Configuration
- **Optional Processor Features**
  - Processor array size puts the upper limit on the amount of data transferred from SM/each processor call.
  - Stack of the processor
  - Enabling the I-cache/D-cache greatly improve the configuration throughput.
- **API on top (provided by Xilinx)**
  - Allows for the s/w control of the IP core (OPBHWICAP/XPSHWIXAP) for accessing the ICAP.
- **Prefetching the configuration bitream**
  - On startup



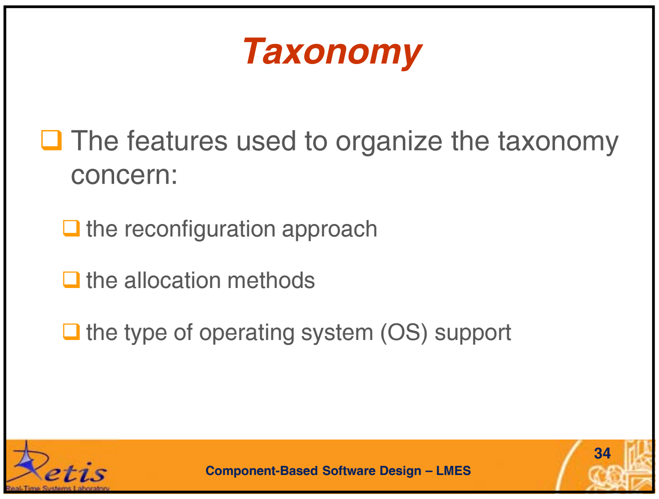
Open Research Problems in PR

- An intelligent run time OS is needed to ensure that low power is consumed and that timing constraints are met when using partial reconfiguration for real time systems (RTOS issues)
- Dynamic Partial Reconfiguration in the domain of Real-Time systems (timing constraints)
- Besides time overhead, configuration procedure adds up power overhead as well. PR needs to be studied as a factor affecting the power of the system (power constraints)
- Algorithms are needed for efficient placement of bitstream into the Reconfigurable region of FPGAs (placement constraints)



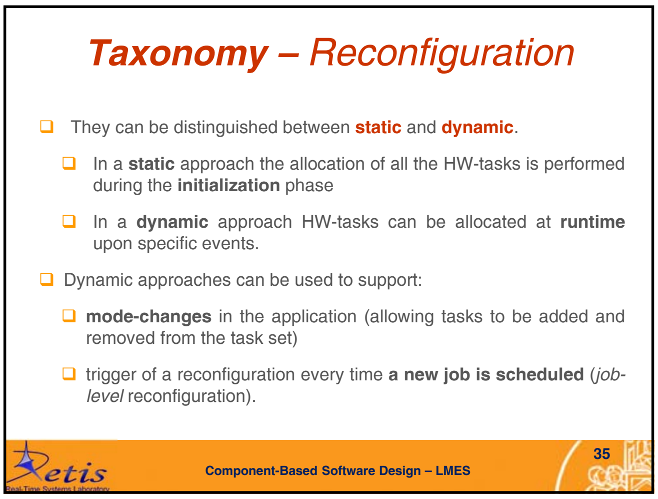
Scheduling goals

- Exploit reconfiguration capabilities to allocate computation tasks on the FPGA
- Use the FPGA as a “core” to execute highly optimized activities available as HW-tasks
- Provide kernel mechanisms for the implementation of this HW scheduling
- Propose a set of model and analysis techniques to provide guarantees regarding applications timing constraints



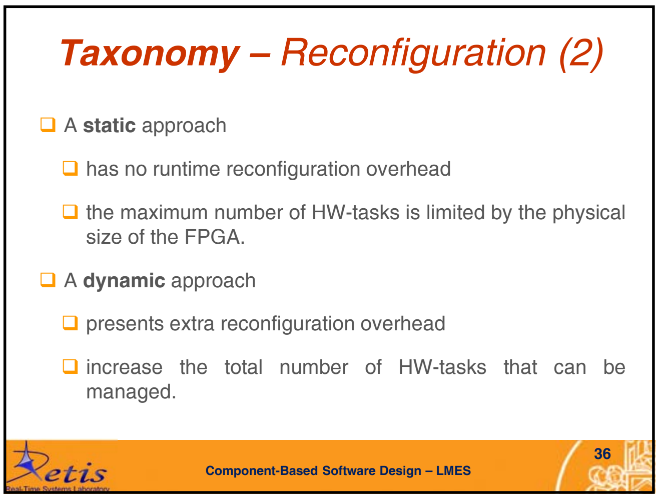
Taxonomy

- The features used to organize the taxonomy concern:
  - the reconfiguration approach
  - the allocation methods
  - the type of operating system (OS) support



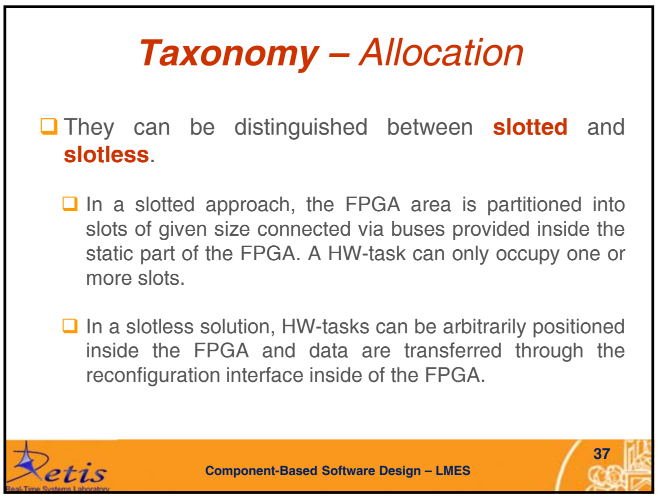
Taxonomy – Reconfiguration

- They can be distinguished between **static** and **dynamic**.
  - In a **static** approach the allocation of all the HW-tasks is performed during the **initialization** phase
  - In a **dynamic** approach HW-tasks can be allocated at **runtime** upon specific events.
- Dynamic approaches can be used to support:
  - **mode-changes** in the application (allowing tasks to be added and removed from the task set)
  - trigger of a reconfiguration every time **a new job is scheduled** (*job-level* reconfiguration).



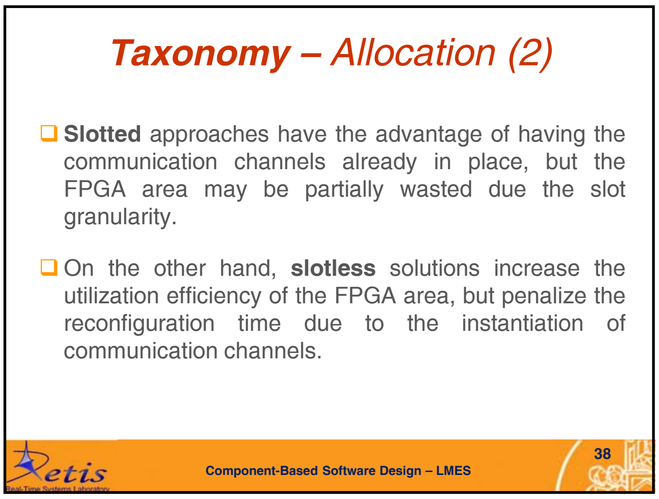
Taxonomy – Reconfiguration (2)

- A **static** approach
  - has no runtime reconfiguration overhead
  - the maximum number of HW-tasks is limited by the physical size of the FPGA.
- A **dynamic** approach
  - presents extra reconfiguration overhead
  - increase the total number of HW-tasks that can be managed.



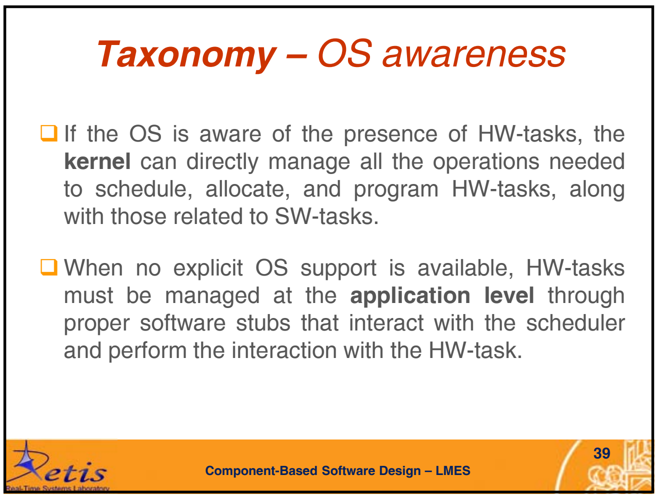
Taxonomy – Allocation

- They can be distinguished between **slotted** and **slotless**.
  - In a slotted approach, the FPGA area is partitioned into slots of given size connected via buses provided inside the static part of the FPGA. A HW-task can only occupy one or more slots.
  - In a slotless solution, HW-tasks can be arbitrarily positioned inside the FPGA and data are transferred through the reconfiguration interface inside of the FPGA.



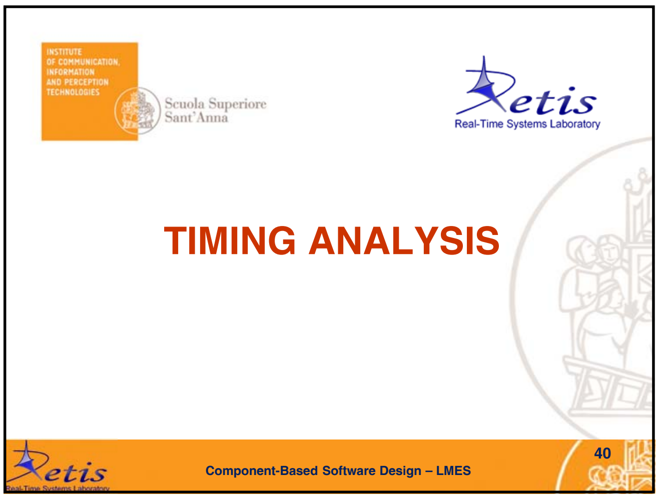
Taxonomy – Allocation (2)

- **Slotted** approaches have the advantage of having the communication channels already in place, but the FPGA area may be partially wasted due the slot granularity.
- On the other hand, **slotless** solutions increase the utilization efficiency of the FPGA area, but penalize the reconfiguration time due to the instantiation of communication channels.



Taxonomy – OS awareness

- If the OS is aware of the presence of HW-tasks, the **kernel** can directly manage all the operations needed to schedule, allocate, and program HW-tasks, along with those related to SW-tasks.
- When no explicit OS support is available, HW-tasks must be managed at the **application level** through proper software stubs that interact with the scheduler and perform the interaction with the HW-task.

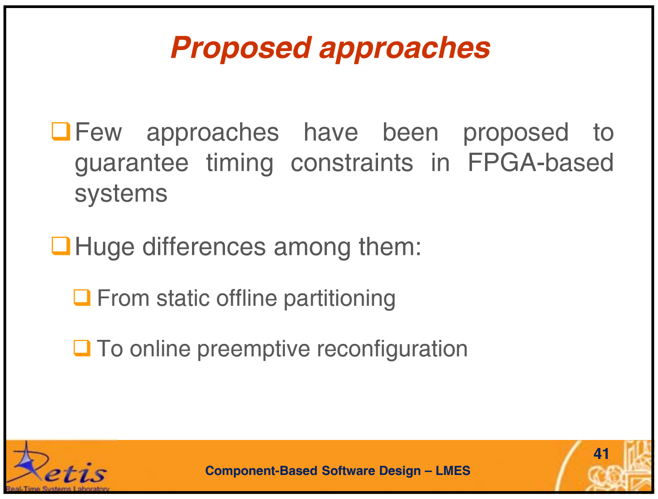


INSTITUTE OF COMMUNICATION, INFORMATION AND PERCEPTION TECHNOLOGIES

Scuola Superiore Sant'Anna

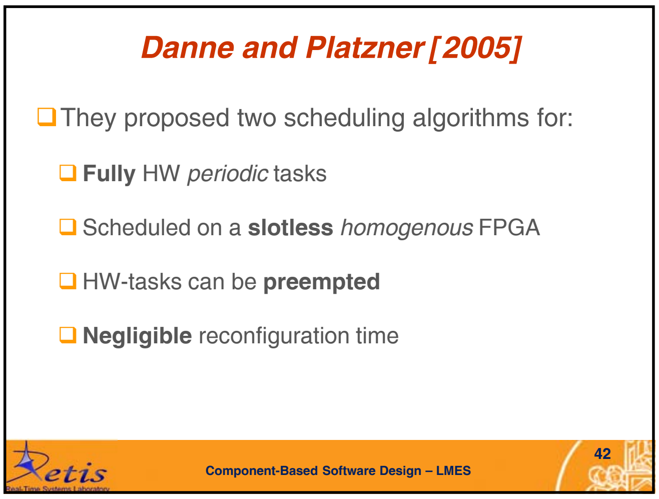
Retis Real-Time Systems Laboratory

TIMING ANALYSIS



Proposed approaches

- Few approaches have been proposed to guarantee timing constraints in FPGA-based systems
- Huge differences among them:
  - From static offline partitioning
  - To online preemptive reconfiguration



Danne and Platzner [2005]

- They proposed two scheduling algorithms for:
  - **Fully** HW *periodic* tasks
  - Scheduled on a **slotless** *homogenous* FPGA
  - HW-tasks can be **preempted**
  - **Negligible** reconfiguration time

EDF-NF algorithm

```
Alg. 1 Earliest Deadline First - Next Fit
Require: list $Q$ of ready tasks, sorted by increasing absolute deadlines
 1: procedure EDF-NF($Q$)
 2:     $R \leftarrow \emptyset$
 3:     $A^{running} = 0$
 4:     for $i \leftarrow 1, |Q|$ do
 5:         if $A^{running} + A_i \leq 1$ then
 6:             $R \leftarrow R \cup T_i$
 7:             $A^{running} = A^{running} + A_i$
 8:         end if
 9:     end for
10:     return $R$
11: end procedure
```

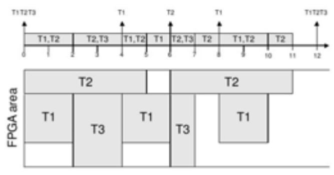
Allocate and execute all tasks fitting in the FPGA, sorted by absolute deadline.

EDF-NF Example

- Scheduling of a small task set composed by 3 periodic tasks.

T_i	P_i	C_i	A_i	U_i^T	U_i^S
T_1	4	2	1/2	1/2	1/4
T_2	6	5	1/4	5/6	5/24
T_3	12	3	3/4	1/4	3/16
				1.58	0.65

$$U^T(\Gamma) = \sum_{T_i \in \Gamma} \frac{C_i}{P_i} \qquad U^S(\Gamma) = \sum_{T_i \in \Gamma} \frac{C_i}{P_i} A_i$$

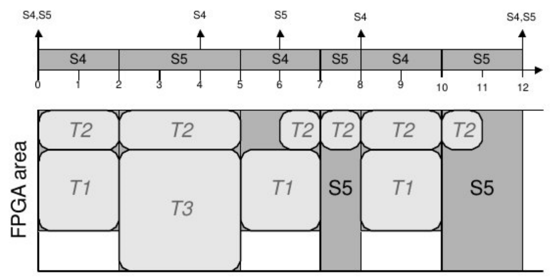


Merge Server Distribute Load (MSDL) algorithm

```
Alg. 2 Merge Server - Distribute Load
 1: procedure MSDL($\Gamma$)
 2:     $\Omega \leftarrow \emptyset$
 3:     for all $T_i \in \Gamma$ do                                    ▷ init
 4:         $S_i \leftarrow (\{T_i\}, P_i, C_i, A_i)$
 5:         $\Omega \leftarrow \Omega \cup S_i$
 6:     end for
 7:     loop
 8:         $S_x, S_y \leftarrow selectValidPairToMerge(\Omega)$
 9:         if no pair found then
10:             return $\Omega$                                          ▷ exit
11:         end if
12:         $S_z \leftarrow (R_x \cup R_y, P_x, C_x, A_x + A_y)$            ▷ $P_y \leq P_x$
13:         $C_x \leftarrow C_x - takeOverTime(S_x, S_z)$
14:         $\Omega \leftarrow \Omega \cup S_z$                                 ▷ add server
15:         $\Omega \leftarrow \Omega \setminus S_y$
16:         if $C_x \leq 0$ then
17:             $\Omega \leftarrow \Omega \setminus S_x$
18:         end if
19:     end loop
20: end procedure
```

Merge tasks together in servers for parallel execution. Obtained servers are executed in a sequential way

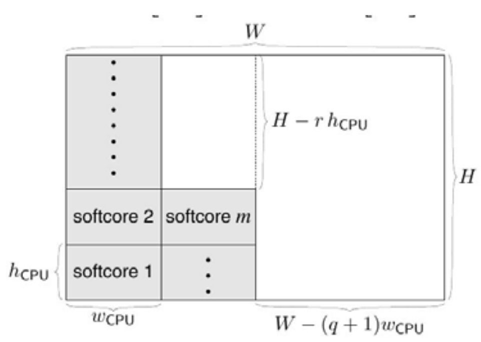
MSDL Example



Di Natale – Bini [2007]

- **Offline optimization** technique for the allocation of tasks on the FPGA
- Each task has **both** a HW-task implementation and a SW-task one
  - A task can be executed in the SW or HW version
- A solution based on an Integer Linear Programming (**ILP**) is used to select the HW-tasks and assign the remaining to a set of softcores.

Allocation model



Pellizzoni et. al. [2007]

- They proposed an admission control test to guarantee feasibility for:
  - tasks having **both** a HW-task implementation and a SW-task one
  - Scheduled on a **slotted** FPGA
  - **Negligible** reconfiguration time
  - **Relocable** at runtime

Relocation example

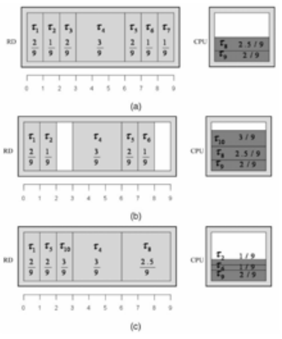


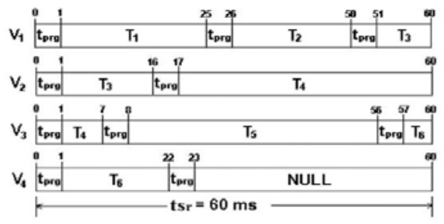
Fig. 6. Example relocation. (a) Example: initial allocation. (b) Example: intermediate allocation. (c) Example: final allocation.

Saha et. at. [2015]

- They proposed two scheduling algorithms for:
  - **Fully** HW *periodic* tasks
  - Scheduled on a common-size **slotted** FPGA
  - HW-tasks can be **preempted**
  - **Fixed** reconfiguration time

Saha et. at. [2015]

- At every deadline each task is allocated a share of the next time slice proportional to its utilization



Main issues

- The more critical issues to be addressed are:
  - The FPGA model is too simple
  - The reconfiguration time is almost not considered
  - Limitations in preemption and relocation are negletted

INSTITUTE OF COMMUNICATION, INFORMATION AND PERCEPTION TECHNOLOGIES
Scuola Superiore Sant'Anna

Retis Real-Time Systems Laboratory

KERNEL MECHANISMS

ROS

- Different solutions have been presented to create kernel level support to **Reconfigurble Operating Systems (ROS)**.
- Available solutions are:
  - R3TOS
  - ReconOS

Questions

Mauro Marinoni - m.marinoni@sssup.it

thank you!

http://retis.sssup.it/people/nino