R3TOS Overview and Architecture

Enrico Rossi

1

R3TOS in a Nutshell

R3TOS is a Reliable Reconfigurable Real-Time Operating System.

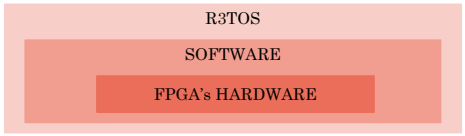
It ease the exploitation of online specialization offered by partially reconfigurable FPGAs, combining computation in space and time to obtain the best performance per transistor and unit of consumed energy.

It abstracts the FPGA's hardware resources and allows to exploit them indifferently for carrying out computation and communication tasks in hardware at different times.

2

R3TOS in a Nutshell

R3TOS creates a unified hardware-software runtime execution environment, whereby a software-centric application developer can easily use the underlying hardware resources to benefit from increased computing speed compared to a conventional processor.



3

Highlights

R3TOS most important capabilities are:

- **Real-Time:** R3TOS gives the necessary support for exploiting the inherent predictability of pure hardware in order to achieve (soft) real-time performance.
- **Dependability:** R3TOS gives the necessary support for exploiting the flexibility of FPGAs to build a system that can reconfigure its own resources in order to maintain the functionality in presence of faults and defects.

4

Highlights

- **High-Performance:** R3TOS gives the necessary support for exploiting the flexibility of FPGAs to load specialized circuits upon demand, each performing a specific type of computation.
- **High-Level Programming:** R3TOS provides the means to make the aforementioned capabilities easy-to-use without requiring any knowledge of low-level FPGA details.

5

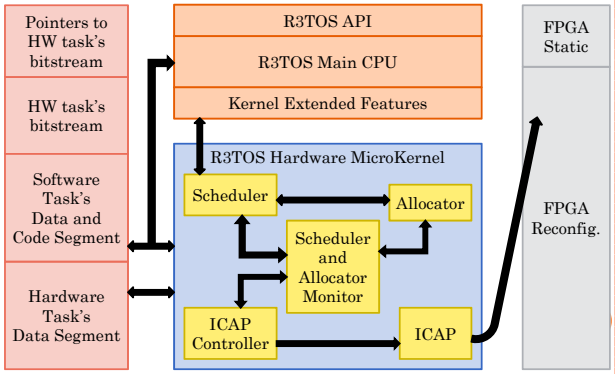
Limitations

The limitations associated to R3TOS mainly come from the reconfiguration bottleneck provoked by ICAP port that takes care of the hardware task allocation and inter-task communication:

- The configuration of an hardware task delays its execution by a non-negligible amount of time.
- The configuration of on-demand communication channels among the task adds a non-negligible overhead, greater then establish a communication on a NoC or on a bus.

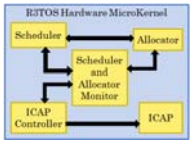
6

Block Diagram



Hardware MicroKernel

The Hardware MicroKernel (HWuK) gives the support to the main CPU to deal with the hardware tasks serving as the substrate upon which the hardware-related services are built.



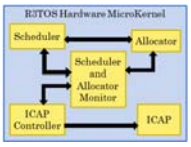
The main offered services are:

- Hardware task queues management;
- FPGA area management;
- Bitstream configuration, allocation and relocation;
- Fault detection of the FPGA's resources.

8

Hardware MicroKernel

The HWuK's internal architecture is structured around the Xilinx PicoBlaze and has some other custom hardware blocks:

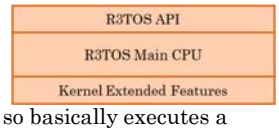


- **Scheduler**, expressly designed to schedule hardware tasks;
- **Allocator** that manages the FPGA resources;
- **ICAP Controller** that translates the high-level operations dictated by the Scheduler and Allocator into reconfiguration commands.

9

Software MicroKernel

The main CPU is based on a Software MicroKernel (SWuK) that provides the basic platform to execute application software routines which cannot be hardware accelerated or parallelized by computation specialization.



It is based on FreeRTOS so basically executes a program which is conceptually similar to a traditional RTOS but it is extended with extra functionality to interact with the HWuK.

10

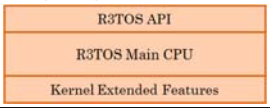
Software MicroKernel

The SWuK's extra features consist of:

- Scheduling and Hardware task;
- Forwarding hardware tasks to the HWuK.

This main CPU is implemented using a Xilinx MicroBlaze plus additional communication peripheral (e.g. Ethernet, UART, ecc).

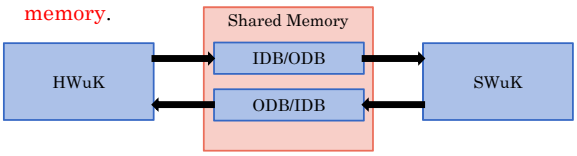
The POSIX-like API layer allows to interact with the FPGA's hardware through high-level functions.



11

HWuK and SWuK Communication

The communication between HWuK and SwuK occurs through a fixed, shared region of the main memory.



The HWuK cannot directly access the data segments of the tasks in the main memory and the mainCPU cannot access the hardware tasks in the FPGA, guaranteeing no interference.

12

Hardware Task types

The term "Hardware Task" is used to indicate that the task relies on specific purpose custom circuitry to perform computation.

- **Data Stream Processing Tasks:** typically these are High Bandwidth Communication tasks processing large amount of data in a short time.
- **Hardware Acceleration Tasks:** typically these are Low Bandwidth Communication tasks processing reduced amount of data in a large time.

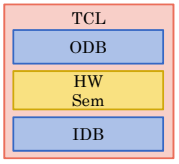
13

Hardware Task types

A generic hardware task is composed by the logic circuit plus a communication interface named TCL (Task Control Logic).

A TCL is made by two FIFO buffers and one hardware semaphore:

- The FIFOs are used to share the data;
- The hardware semaphore is used to regulate the access to the hardware task.



14

Hardware Task Allocation & Relocation

The FPGA can be seen as a field of programmable connections and each connection is mapped into a bitstream-memory.

Loading a specific configuration file (bitstream) in the memory would mean program the FPGA to implement a specific hardware circuit.

We can think to program only a portion of the memory in order to program only a part of the FPGA…

15

Hardware Task Allocation & Relocation

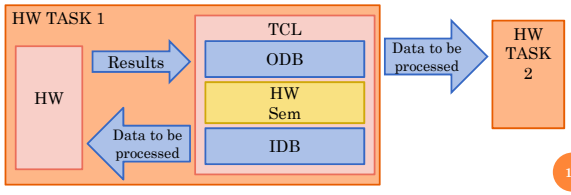
We can think about this flow:

- Divide the reconfigurable application in tasks;
- Synthesize the hardware task together with a wrapping Task Control Logic (input and output buffers and an hardware semaphore);
- Obtain a single, relocable partial bitstream for each of the hardware tasks that can be directly loaded into the bitstream-memory.

16

Hardware/Software Task Communication

TCL (Task Control Logic) blocks attached to hardware (or software) tasks provide support for Synchronization, Communication and Data Buffering.



17

Hardware/Software Task Communication

The TCL delivers the data to be processed from its IDB to the task and stores the results into the ODB.

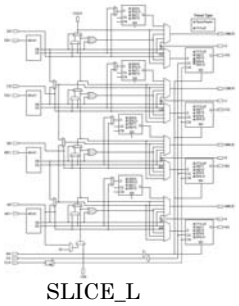
In case of communication between HW and SW tasks, the principle is the same but the TCL of the software task is mapped in the program memory.

The hardware tasks are provided with a "ghost software body" which include HWuK-related system calls with the objective of making them manageable in SWuK.

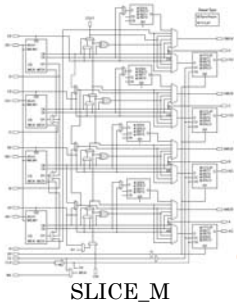
18

Appendice A: FPGA Logic Element

- One Configurable Logic Block contains two SLICE_M or a SLICE_L and a SLICE_M.



SLICE_L



SLICE_M

19

R3TOS

Reliable Reconfigurable Real-Time Operating System

Marco Pagani

ROS Overview

- **ROS** (*Reconfigurable operating system*) is an **operating system augmented** with functions to **manage reconfigurable hardware (FPGA)**
- **ROS hide complexity** by offering a set of basic **services**, accessible through an **API**, to the application developer:
  - task switching
  - intertask communication
  - synchronization
  - etc ...
- Provides **runtime support** for both **task management** and **reconfigurable hardware resource management**.

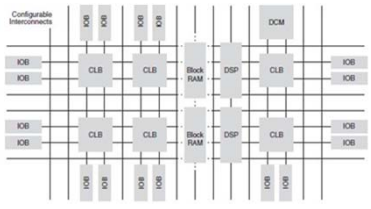
R3TOS

- **R3TOS** creates a **unified HW-SW runtime execution environment**
- **R3TOS** main features:
  - **Performance:** support for exploiting the **flexibility** of **FPGAs** to load **specialized circuits upon demand**, each performing a **specific** type of **computation**
  - **Soft real-time:** exploiting **predictability** of pure **hardware** to achieve (soft) real-time performance **(QoS)**
  - **Dependability:** exploiting the **flexibility** of **FPGAs** to **maintain functionality** in the presence of **permanent defects** and **spontaneous faults**

Dynamic partial reconfiguration

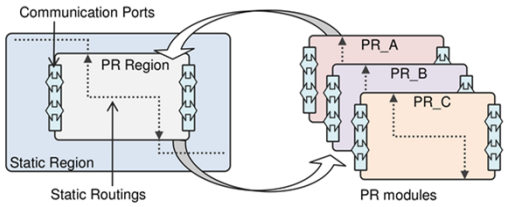
Reconfigurable device support

- An **FPGA** is a *non-homogeneous* **computing fabric** made of **reconfigurable resources**:
  - *Logic cells (CLB)*
  - *Specific function blocks (BRAM / DSP /…)*
  - *Input/Output blocks (IOB)*
  - *Routing resources (Switch Matrix)*



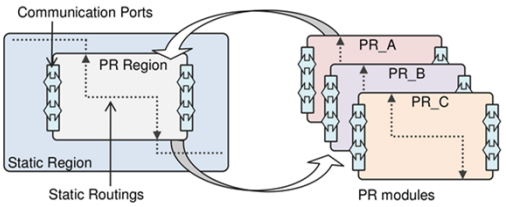
Reconfigurable device support

- **Dynamic Partial Reconfiguration (DPR)** allows some portions of the **FPGA** to be **reconfigured** at **runtime** while the rest continues to operate
- **DRP** is the **enabling technology** for **reconfigurable computing**



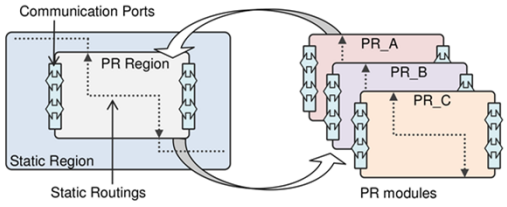
Reconfigurable device support

- When using **DRP** the **FPGA** is **partitioned** in:
  - **Static region** (remains **unchanged**, host **static system**)
  - **Reconfigurable partitions** (each can accommodate a set of **reconfigurable modules** in **time-sharing**)



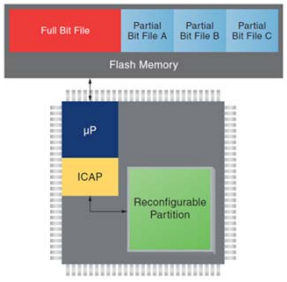
Reconfigurable device support

- When using **DRP** the **FPGA** is **partitioned** in:
  - **Static region** (remains **unchanged**, host **static system**)
  - **Reconfigurable partitions** (each can accommodate a set of **reconfigurable modules** in **time-sharing**)
- **Typically reconfigurable partitions** are **fixed** in the area and organized as **islands** or **slots**



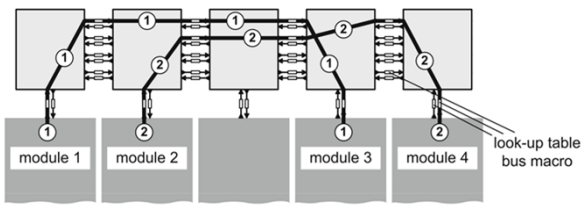
Reconfigurable device support

- A **reconfigurable modules** can be loaded on a **reconfigurable partition** by an **embedded microprocessor (softcore)** through the **Internal Configuration Access Port (ICAP)**



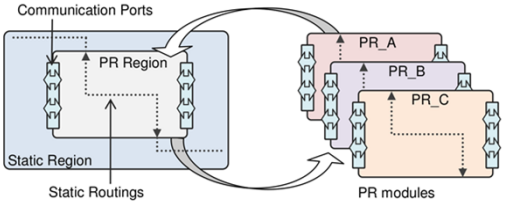
Reconfigurable device support

- Typically the **static system** contains a **communication infrastructure** that interconnects all **reconfigurable partitions**



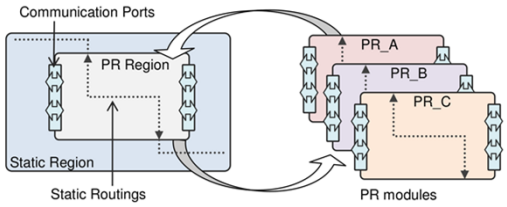
Reconfigurable device support

- The **static system** might **cross** a **reconfigurable partition** to carry out the **routing**



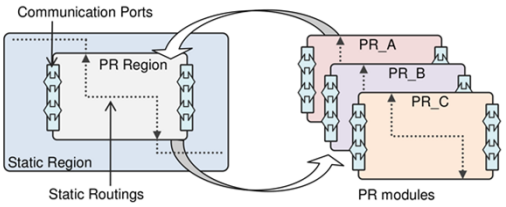
Reconfigurable device support

- The **static system** might **cross** a **reconfigurable partition** to carry out the **routing**
  - Therefore **reconfigurable modules** may **include** information about the **static routing**



Reconfigurable device support

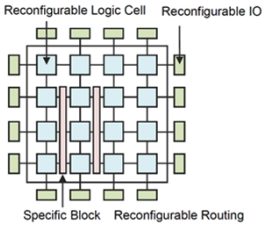
- The **static system** might **cross** a **reconfigurable partition** to carry out the **routing**
  - Therefore **reconfigurable modules** may **include** information about the **static routing**
  - **This prevents reconfigurable modules relocability** (not supported by Xilinx DPR flow)



R3TOS overview

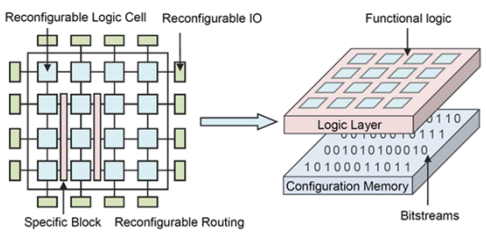
Reconfigurable device model

- An **FPGA** can be **modelled** as a **two layers** architecture:



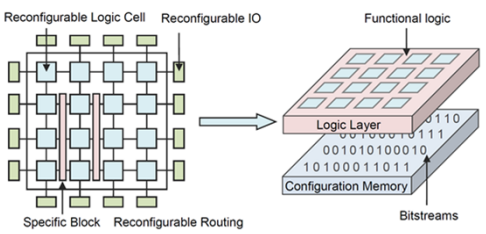
Reconfigurable device model

- An **FPGA** can be **modelled** as a **two layers** architecture:
  - **functional layer**: contains the **physical resources** used to perform **computation**



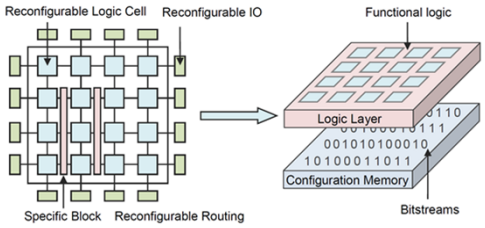
Reconfigurable device model

- An **FPGA** can be **modelled** as a **two layers** architecture:
  - **functional layer**: contains the **physical resources** used to perform **computation**
  - **configuration layer**: **controls** and contains the **configuration** of the **functional layer**



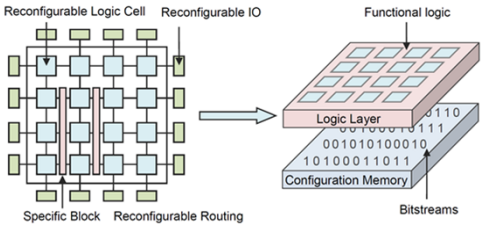
Reconfigurable device model

- The **ICAP port** act as an **interface** between the **configuration layer** and the **functional layer**



Reconfigurable device model

- The **ICAP port** act as an **interface** between the **configuration layer** and the **functional layer**
- Theoretical **bandwidth** of 400 MB/s
  - **reconfigurable module configuration time** is proportional to the **module size**



R3TOS

- **R3TOS** approach to tackle those issues:
  - **No static communication infrastructure:**

R3TOS

- **R3TOS** approach to tackle those issues:
  - **No static communication infrastructure:**
    - ***Dynamic*** *communication infrastructure*

R3TOS

- **R3TOS** approach to tackle those issues:
  - **No static communication infrastructure:**
    - ***Dynamic*** *communication infrastructure*
    - *Communications between* ***modules*** *through the* ***configuration layer*** *(ICAP virtual channels) or through shared* ***functional resources*** *(BRAM)*

R3TOS

- **R3TOS** approach to tackle those issues:
  - **No static communication infrastructure:**
    - ***Dynamic*** *communication infrastructure*
    - *Communications between* ***modules*** *through the* ***configuration layer*** *(ICAP virtual channels) or through shared* ***functional resources*** *(BRAM)*
  - **Reconfigurable region** is organized as **one single large reconfigurable partition**

R3TOS

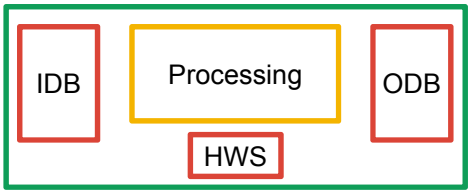
- **R3TOS** approach to tackle those issues:
  - **No static communication infrastructure:**
    - ***Dynamic*** *communication infrastructure*
    - *Communications between* ***modules*** *through the* ***configuration layer*** *(ICAP virtual channels) or through shared* ***functional resources*** *(BRAM)*
  - **Reconfigurable region** is organized as **one single large reconfigurable partition**
    - *Complete* ***modules*** *relocability*
    - *Simpler online allocation strategy*

R3TOS

- in **R3TOS the basic hardware unit** for **computation** or **communication**: **hardware tasks,** are **implemented** as **reconfigurable modules**:

R3TOS

- in **R3TOS the basic hardware unit** for **computation** or **communication**: **hardware tasks,** are **implemented** as **reconfigurable modules**:
  - **HW task logic** is **wrapped** with an **hardware container (TCL)** to build a **self contained**, **relocatable module**, with a **standard hardware interface** for **data exchange (IOB/ODB)** and **synchronization (HWS)**



Tasks classification

- **R3TOS** authors **classify tasks**, as **logical** entities, in **two classes** based on the **communication requirements:**

Tasks classification

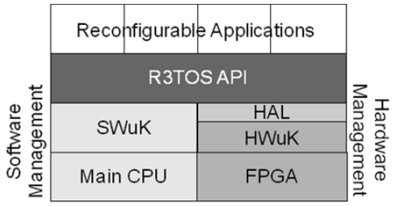
- **R3TOS** authors **classify tasks**, as **logical** entities, in **two classes** based on the **communication requirements:**
  - **High-Bandwidth Communication (HBC)** tasks:
    - *process a* ***high amount of data*** *within a relatively* ***short*** *amount of* ***time****, i.e.* ***communication dominates computation.***
    - ***Data stream processing tasks***

Tasks classification

- **R3TOS** authors **classify tasks**, as **logical** entities, in **two classes** based on the **communication requirements:**
  - **High-Bandwidth Communication (HBC)** tasks:
    - *process a* ***high amount of data*** *within a relatively* ***short*** *amount of* ***time****, i.e.* ***communication dominates computation.***
    - ***Data stream processing tasks***
  - **Low-Bandwidth Communication (LBC)** tasks:
    - *process a* ***reduced amount of data*** *within a relatively* ***long*** *amount of* ***time****, i.e.* ***computation dominates communication.***
    - ***Hardware acceleration tasks***

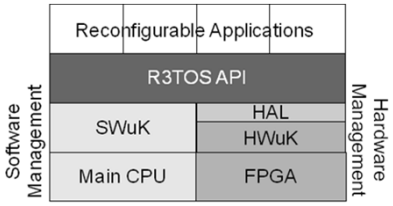
R3TOS API

- From the **developer** perspective **R3TOS** provide a **POSIX-like API** to access the **low-level services** implemented by the **HWuK** (i.e., **HAL**) and to exploit FPGA resources



R3TOS API

- From the **developer** perspective **R3TOS** provide a **POSIX-like API** to access the **low-level services** implemented by the **HWuK** (i.e., **HAL**) and to exploit FPGA resources
- **HAL** is **wrapped** with a **software OS layer**, the **SWuK**, which is executed on the main CPU

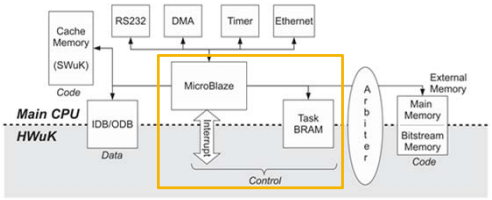


R3TOS

- The **SWuK** is a modified version of **FreeRTOS**
  - *Scheduler has been modified to provide support for hardware tasks (preemption is disabled for hw tasks)*
  - *New ISRs to enable communication with the HWuK*

R3TOS

- The **SWuK** is a modified version of **FreeRTOS**
  - *Scheduler has been modified to provide support for hardware tasks (preemption is disabled for hw tasks)*
  - *New ISRs to enable communication with the HWuK*
- **SWuk** and **HWuK** shares **control information** and task parameters through a **shared memory** (Task BRAM) with interrupt signaling



R3TOS

- The **R3TOS** stack allow to **manage HW** and **SW tasks** in a **uniform way**

R3TOS

- The **R3TOS** stack allow to **manage HW** and **SW tasks** in a **uniform way**
  - **Hardware tasks** have a **"software body"** to make them manageable in **SWuK**

R3TOS

- The **R3TOS** stack allow to **manage HW** and **SW tasks** in a **uniform way**
  - **Hardware tasks** have a **"software body"** to make them manageable in **SWuK**
  - The **software body** of an **hardware task** may also include SWuK system calls and **regular software code**
    - **HW accelerated SW Task** model… (LBC tasks)

R3TOS

- The **R3TOS** stack allow to **manage HW** and **SW tasks** in a **uniform way**
  - **Hardware tasks** have a **"software body"** to make them manageable in **SWuK**
  - The **software body** of an **hardware task** may also include SWuK system calls and **regular software code**
    - **HW accelerated SW Task** model… (LBC tasks)
- **HW-SW** application **framework**:

R3TOS

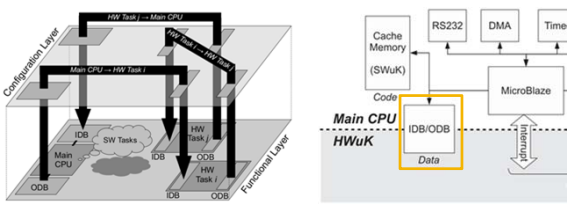
- The **R3TOS** stack allow to **manage HW** and **SW tasks** in a **uniform way**
  - **Hardware tasks** have a **"software body"** to make them manageable in **SWuK**
  - The **software body** of an **hardware task** may also include SWuK system calls and **regular software code**
    - **HW accelerated SW Task** model… (LBC tasks)
- **HW-SW** application **framework**:
  - **Hi-level SWuK software API**: **higher abstraction** level, trade off with **performances loss**

R3TOS

- The **R3TOS** stack allow to **manage HW** and **SW tasks** in a **uniform way**
  - **Hardware tasks** have a **"software body"** to make them manageable in **SWuK**
  - The **software body** of an **hardware task** may also include SWuK system calls and **regular software code**
    - **HW accelerated SW Task** model… (LBC tasks)
- **HW-SW** application **framework**:
  - **Hi-level SWuK software API**: **higher abstraction** level, trade off with **performances loss**
  - **Low-level HAL API**: direct access to **HWuk** services, **higher performance**, **lower abstraction**

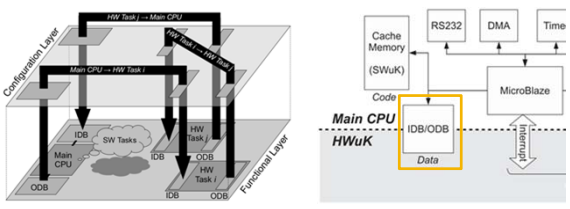
HW-SW inter-task communications

- **Data** are **exchanged** between **HW** and **SW tasks** through a fixed main **memory region shared** between the **CPU** and the **HWuK** and organized in the form of an **ODB** and an **IDB**



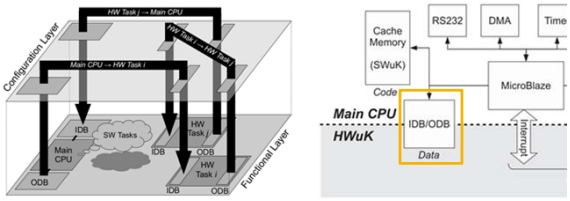
HW-SW inter-task communications

- **Data** are **exchanged** between **HW** and **SW tasks** through a fixed main **memory region shared** between the **CPU** and the **HWuK** and organized in the form of an **ODB** and an **IDB**
  - **CPU to HW task:** The data written by the **CPU** in the **ODB** is delivered by **HWuK** to the **hardware task** running on the FPGA



HW-SW inter-task communications

- **Data** are **exchanged** between **HW** and **SW tasks** through a fixed main **memory region shared** between the **CPU** and the **HWuK** and organized in the form of an **ODB** and an **IDB**
  - **CPU to HW task:** The data written by the **CPU** in the **ODB** is delivered by **HWuK** to the **hardware task** running on the FPGA
  - **HW task to CPU:** The data written by **HWuK** in the **IDB** is relocated by the **CPU** to the data segment assigned to the corresponding software task in the main memory



Management of an HW task in SWuK - 1

```
void vTask_ID (void *pvParameters)
{
   insert_task(task_ID,task_params);
   wait_scheduling(task_ID);
   wr_ODB(&input_data_base_addr, length);
   wait_computing(task_ID);
   rd_IDB(&output_results_base_addr, length);
}
```

Management of an HW task in SWuK - 2

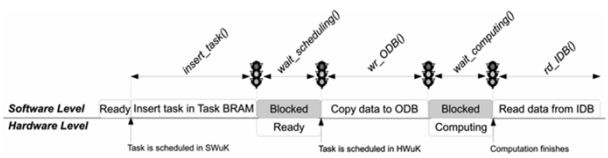
- **HW tasks (SW bodies)** have the **highest priority** in **SWuK**
  - they are **immediately set in execution** by **SWuK's scheduler**

Management of an HW task in SWuK - 2

- **HW tasks (SW bodies)** have the **highest priority** in **SWuK**
  - they are **immediately set in execution** by **SWuK's scheduler**
- Immediately after a **HW** task is inserted into the ***HW task queue*** [`insert_task()`], it is **blocked** in the **software** level [`wait_scheduling()`] until the **HWuK's** scheduler selects it to be executed on the **FPGA**

Management of an HW task in SWuK - 3

- When the **HW task** is **scheduled** by **HWuK** scheduler, and **allocated** by the allocator, the task is **awakened** in the **software level**

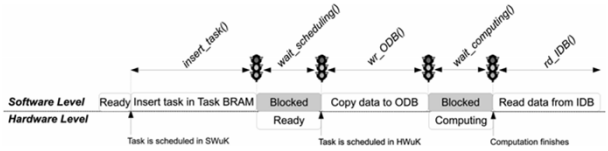


Management of an HW task in SWuK - 3

- When the **HW task** is **scheduled** by **HWuK** scheduler, and **allocated** by the allocator, the task is **awakened** in the **software level**
- After **transferring** the **input data** to the task's **ODB** [`wr_ODB()`], the task is **blocked** in the **software level** [`wait_computing()`] and **starts hardware computation**

Management of an HW task in SWuK - 4

- When the **HW computation** finishes, the **software task** is **awakened** and retrieves the **computed results** from the **IDB** [`rd_IDB()`]



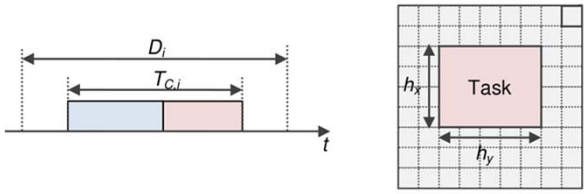
Management of an HW task in SWuK - 4

- When the **HW computation** finishes, the **software task** is **awakened** and retrieves the **computed results** from the **IDB** [`rd_IDB()`]
- **Task blocking** and awakening mechanisms in the software level are based on **binary semaphores** provided by **FreeRTOS**

R3TOS hardware task model

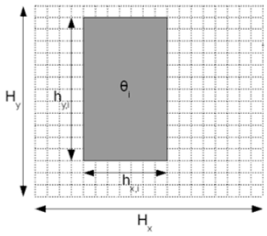
Hardware task model

- An **hardware task** θ_i , executing on the **FPGA**, can be modelled in two domains: **Area** and **Time**



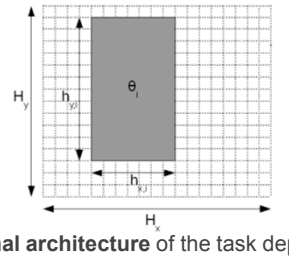
Hardware task model - Area domain

- **Area domain**: the **task** θ_i occupy a **rectangular region** of the **FPGA** defined by its **width** and **height**: $h_{x,i}$, $h_{y,i}$
  - The **area** of the **entire FPGA** is: H_x , H_y



Hardware task model - Area domain

- **Area domain**: the **task** θ_i occupy a **rectangular region** of the **FPGA** defined by its **width** and **height**: $h_{x,i}$, $h_{y,i}$
  - The **area** of the **entire FPGA** is: H_x , H_y



  - The **internal architecture** of the task depends on the location where it was **originally synthesized**

Hardware task model - Time domain - 1

- **Time domain**: the complete model of a **task** θ_i consist of **five** different **phases**:

Hardware task model - Time domain - 1

- **Time domain**: the complete model of a **task** θ_i consist of **five** different **phases**:
  - **Set-up** phase: $t_{A,i}$
    - *Task is configured on the FPGA*

Hardware task model - Time domain - 1

- **Time domain**: the complete model of a **task** θ_i consist of **five** different **phases**:
  - **Set-up** phase: $t_{A,i}$
    - *Task is configured on the FPGA*
  - **Input data delivery** phase: $t_{D,i}$
    - *Fill the task IDB with input data*

Hardware task model - Time domain - 1

- **Time domain**: the complete model of a **task** θ_i consist of **five** different **phases**:
  - **Set-up** phase: $t_{A,i}$
    - *Task is configured on the FPGA*
  - **Input data delivery** phase: $t_{D,i}$
    - *Fill the task IDB with input data*
  - **Execution** phase: $t_{E,i}$
    - *Temporal isolation, hardware determinism: worst-case predictable timing behaviour*

Hardware task model - Time domain - 1

- **Time domain**: the complete model of a **task** θ_i consist of **five** different **phases**:
  - **Set-up** phase: $t_{A,i}$
    - *Task is configured on the FPGA*
  - **Input data delivery** phase: $t_{D,i}$
    - *Fill the task IDB with input data*
  - **Execution** phase: $t_{E,i}$
    - *Temporal isolation, hardware determinism: worst-case predictable timing behaviour*
  - **Synchronization** phase: $t_{S,i}$
    - *Wait on the task's HWS to detect computation completion*

Hardware task model - Time domain - 1

- **Time domain**: the complete model of a **task** θ_i consist of **five** different **phases**:
  - **Set-up** phase: $t_{A,i}$
    - *Task is configured on the FPGA*
  - **Input data delivery** phase: $t_{D,i}$
    - *Fill the task IDB with input data*
  - **Execution** phase: $t_{E,i}$
    - *Temporal isolation, hardware determinism: worst-case predictable timing behaviour*
  - **Synchronization** phase: $t_{S,i}$
    - *Wait on the task's HWS to detect computation completion*
  - **Output result retrieval** phase: $t_{R,i}$
    - *Read the result from ODB*

Hardware task model - Time domain - 2

- When **two HW tasks: θ_j (producer), θ_i (consumer) communicate** each other:
  - The **output data retrieval** phase of the **producer** task θ_j (**ODB read**) is immediately followed by the **input data delivery** phase of the consumer task θ_i (**IDB write**)

Hardware task model - Time domain - 2

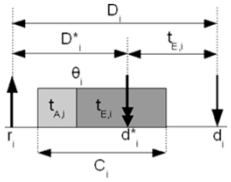
- When **two HW tasks: θ_j (producer), θ_i (consumer) communicate** each other:
  - The **output data retrieval** phase of the **producer** task θ_j (**ODB read**) is immediately followed by the **input data delivery** phase of the consumer task θ_i (**IDB write**)
  - Those **two phases** are to be **preceded** by the **set-up phase** of the data **consumer task θ_i**

Hardware task model - Time domain - 2

- When **two HW tasks: θ_j (producer), θ_i (consumer) communicate** each other:
  - The **output data retrieval** phase of the **producer** task θ_j (**ODB read**) is immediately followed by the **input data delivery** phase of the consumer task θ_i (**IDB write**)
  - Those **two phases** are to be **preceded** by the **set-up phase** of the data **consumer task θ_i**
  - The latter **three phases** are **merged together** with the **synchronization phase** of the **data producer task θ_j** to form a **single ICAP access period** for **each task θ_i**

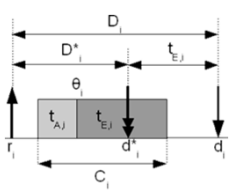
Hardware task model - Time domain - 3

- **Real-Time** constraints in **R3TOS** for a **task θ_i** :
  - **Relative deadline D_i**
    - **Absolute** deadline: $d_i = D_i + r_i$



Hardware task model - Time domain - 3

- **Real-Time** constraints in **R3TOS** for a **task θ_i** :
  - **Relative deadline D_i**
    - **Absolute** deadline: $d_i = D_i + r_i$
  - **Absolute set-up deadline d^*_i**
    - Latest instant **(delay)** for a task to **start the computation** in order to meet its deadline



Hardware task model - Time domain - 4

- Set-up **deadlines** are different for **HW tasks** which **communicate** with **other HW tasks** and for those which **deliver data** to **SW tasks**

Hardware task model - Time domain - 4

- Set-up **deadlines** are different for **HW tasks** which **communicate** with **other HW tasks** and for those which **deliver data** to **SW tasks**
  - The **data retrieval phase** is **included** in the **model** of the **data consumer HW tasks** used by **HWuK**

Hardware task model - Time domain - 4

- Set-up **deadlines** are different for **HW tasks** which **communicate** with **other HW tasks** and for those which **deliver data** to **SW tasks**
  - The **data retrieval phase** is **included** in the **model** of the **data consumer HW tasks** used by **HWuK**
  - **Not included** in the model of **data consumer SW tasks** used by **SWuK**

Hardware task model - Time domain - 4

- Set-up **deadlines** are different for **HW tasks** which **communicate** with **other HW tasks** and for those which **deliver data** to **SW tasks**
  - The **data retrieval phase** is **included** in the **model** of the **data consumer HW tasks** used by **HWuK**
  - **Not included** in the model of **data consumer SW tasks** used by **SWuK**
    - The **data retrieval operation** of a data consumer **SW task** must be **invoked** from the **data producer HW task** itself

Hardware task model - Time domain - 4

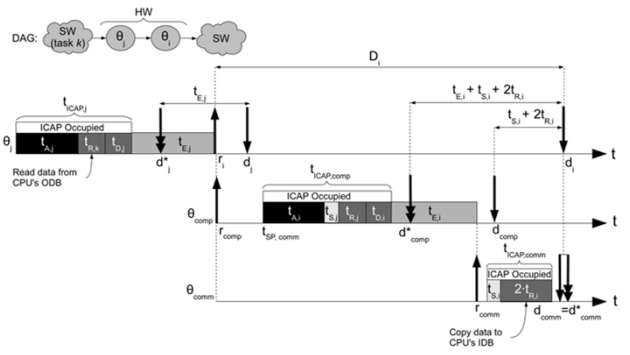
- Set-up **deadlines** are different for **HW tasks** which **communicate** with **other HW tasks** and for those which **deliver data** to **SW tasks**
  - The **data retrieval phase** is **included** in the **model** of the **data consumer HW tasks** used by **HWuK**
  - **Not included** in the model of **data consumer SW tasks** used by **SWuK**
    - The **data retrieval operation** of a data consumer **SW task** must be **invoked** from the **data producer HW task** itself
- **HW tasks** that **deliver data** to the **CPU** are modelled as **two separate tasks**

Hardware task model - Time domain - 4

- Set-up **deadlines** are different for **HW tasks** which **communicate** with **other HW tasks** and for those which **deliver data** to **SW tasks**
  - The **data retrieval phase** is **included** in the **model** of the **data consumer HW tasks** used by **HWuK**
  - **Not included** in the model of **data consumer SW tasks** used by **SWuK**
    - The **data retrieval operation** of a data consumer **SW task** must be **invoked** from the **data producer HW task** itself
- **HW tasks** that **deliver data** to the **CPU** are modelled as **two separate tasks**
  - **Computing task**
  - **Communicating task** (copy data to the CPU's IDB)

Hardware task model - Time domain - 5

- Execution of two hardware tasks:

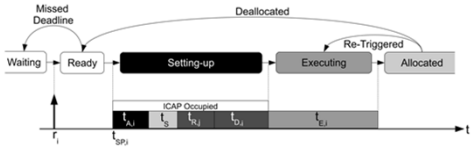


Hardware task states

- An **HW task** goes through several **states** during its life cycle:

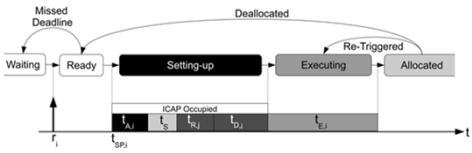
Hardware task states

- An **HW task** goes through several **states** during its life cycle:
  - **Waiting state**



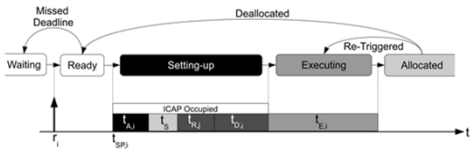
Hardware task states

- An **HW task** goes through several **states** during its life cycle:
  - **Waiting state**
  - **Ready state**
    - waiting to be **scheduled** and **assigned** to a **set of resources** on **FPGA**



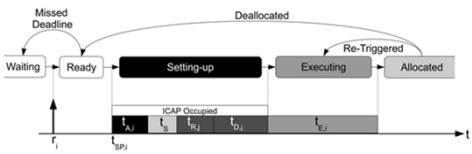
Hardware task states

- An **HW task** goes through several **states** during its life cycle:
  - **Waiting state**
  - **Ready state**
    - waiting to be **scheduled** and **assigned** to a **set of resources** on **FPGA**
  - **Setting-up state**
    - **Configured** on the FPGA and **provided** with **input data**



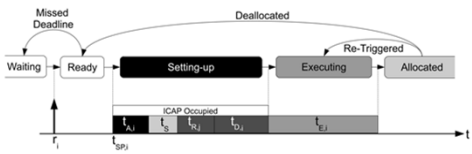
Hardware task states

- An **HW task** goes through several **states** during its life cycle:
  - **Waiting state**
  - **Ready state**
    - waiting to be **scheduled** and **assigned** to a **set of resources** on **FPGA**
  - **Setting-up state**
    - **Configured** on the FPGA and **provided** with **input data**
  - **Execution state**
    - Active computation



Hardware task states

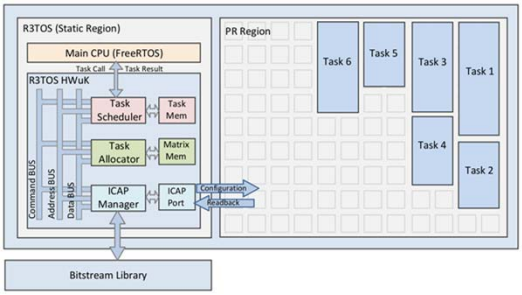
- An **HW task** goes through several **states** during its life cycle:
  - **Waiting state**
  - **Ready state**
    - waiting to be **scheduled** and **assigned** to a **set of resources** on **FPGA**
  - **Setting-up state**
    - **Configured** on the FPGA and **provided** with **input data**
  - **Execution state**
    - Active computation
  - **Allocated state**
    - **remains configured** in the FPGA after computation



R3TOS scheduling and allocation

R3TOS scheduling and allocation

- **HWuK** services:
  - **Scheduling** (*time domain*)
  - **Allocation** (*area domain*)
  - **Device Configuration**



R3TOS scheduling and allocation

- **Hardware Scheduling:**
  - More **overhead** with respect to **software scheduling**

R3TOS scheduling and allocation

- **Hardware Scheduling:**
  - More **overhead** with respect to **software scheduling**
    - **Search** for a **location** (**allocation** problem)
    - **Modify** the task **bitstream** (relocation)
    - **Loading** the **bitstream** through **configuration port**

R3TOS scheduling and allocation

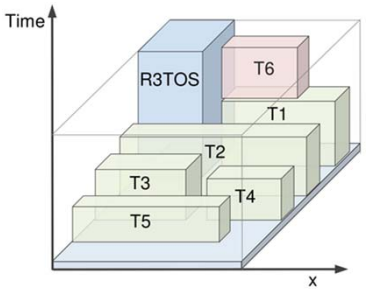
- **Hardware Scheduling:**
  - More **overhead** with respect to **software scheduling**
    - **Search** for a **location** (**allocation** problem)
    - **Modify** the task **bitstream** (relocation)
    - **Loading** the **bitstream** through **configuration port**
- **Preemption** is possible but **context switchings** are **expensive**:
  - **Read back**, **store**, and **write the bitstream** to **restore context**

R3TOS scheduling and allocation

- **Hardware Scheduling:**
  - More **overhead** with respect to **software scheduling**
    - **Search** for a **location** (**allocation** problem)
    - **Modify** the task **bitstream** (relocation)
    - **Loading** the **bitstream** through **configuration port**
- **Preemption** is possible but **context switchings** are **expensive**:
  - **Read back**, **store**, and **write the bitstream** to **restore context**
- **FPGA**: **Real HW tasks concurrency**
  - **Bottleneck: ICAP** accesses must be **mutual exclusive** and **sequential**
  - **Serialize ICAP accesses**

R3TOS scheduling and allocation

- **FPGA scheduling and allocation** in general:
  - **3D bin packing problem**
  - **2 Area** dimensions + **1 time** dimension (DPR)



R3TOS scheduling algorithms

- **R3TOS** base scheduling algorithm:
  - Non-preemptive **EDF**
  - Online scheduling to cope with HW faults

R3TOS scheduling algorithms

- **R3TOS** base scheduling algorithm:
  - Non-preemptive **EDF**
  - Online scheduling to cope with HW faults

```
inputs:
   R: list of ready hw tasks sorted by increasing allocation
   deadline (d* - t_icap)

   MER: Free area information provided by the allocator
   (maximum empty rectangle)

procedure:
   for (task i: R)
      if (i fits in MER) {
            allocate i;
            return i;
      }
   return null;
```

R3TOS scheduling algorithms

- **Finishing-Aware EDF (FAEDF)**
  - Extension of NP-**EDF**
  - Look ahead: aware of the tasks finishing time
  - Compensate for the lack of preemption

R3TOS scheduling algorithms

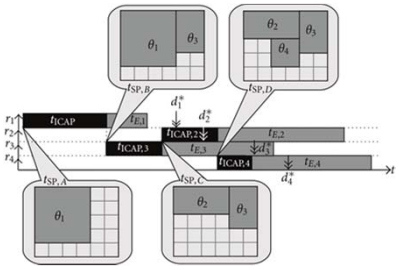
- **Finishing-Aware EDF (FAEDF)**

procedure:

- NP-EDF: try to allocate **task i**
- If unsuccessful (not enough resources) and real-time constraints are "loose"
  - Scan the **list** of **executing** tasks
  - If find a **task j** that **finish execution** (L.A.), **freeing resources**, before **task i allocation deadline** ($d^* - t_{icap}$)
    - set **insertion flag**
    - **task i** will be **delayed**
- If **insertion flag** is set
  - Scan **ready tasks** queue
  - If finds a **task m** that can **finish its allocation** before **task i allocation deadline**
    - Schedule **task m**

R3TOS scheduling algorithms

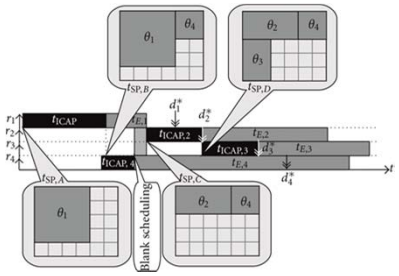
- **NP-EDF example:**



- At $t_{SP,B}$ θ_2 cannot be allocated, not enough area
- θ_3 is scheduled instead but this delays the subsequent allocation of θ_2 too long, and, as a result, misses its deadline

R3TOS scheduling algorithms

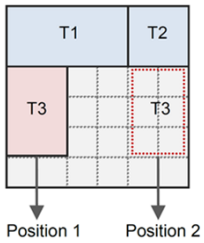
- **FAEDF example:**



- At $t_{SP,B}$ **FAEDF** finds out that θ_2 can be allocated later using the resources that will be released by θ_1
- The time until then is used to allocate θ_4

R3TOS allocation algorithms

- **Allocation problem:**
  - **Where** an **hardware tasks** should be **allocated** to reduce the **external fragmentation** in the **FPGA's** reconfigurable partition



R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - Empty Area / Volume Compaction heuristics (EAC / EVC)

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - Empty Area / Volume Compaction heuristics (EAC / EVC)
- The **FPGA** is modelled as a **grid**

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - Empty Area / Volume Compaction heuristics (EAC / EVC)
- The **FPGA** is modelled as a **grid**
- **Scan** the **area** (grid) of the FPGA in the **horizontal direction** (**1D**)
  - Count the **consecutive** number of **available resources** (not occupied or damaged)

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - Empty Area / Volume Compaction heuristics (EAC / EVC)
- The **FPGA** is modelled as a **grid**
- **Scan** the **area** (grid) of the FPGA in the **horizontal direction** (**1D**)
  - Count the **consecutive** number of **available resources** (not occupied or damaged)
- **Scan** the **area** (grid) of the FPGA in the **vertical direction** (**2D**)
  - Count **consecutive available resources** to find the **greatest empty rectangle** that can be formed at **each position**

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - Empty Area / Volume Compaction heuristics (EAC / EVC)
- The **FPGA** is modelled as a **grid**
- **Scan** the **area** (grid) of the FPGA in the **horizontal direction (1D)**
  - Count the **consecutive** number of **available resources** (not occupied or damaged)
- **Scan** the **area** (grid) of the FPGA in the **vertical direction (2D)**
  - Count **consecutive available resources** to find the **greatest empty rectangle** that can be formed at **each position**
- The process produces **four matrices** (URAM, ULAM, DRAM, DLAM) that represent, for **each grid position**, the **widest empty rectangle** which can be formed in **each orientation** (N, S, W, E)

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - Empty Area / Volume Compaction heuristics (EAC / EVC)
- The **four matrices** (URAM, ULAM, DRAM, and DLAM) are **added** to build the **2D Adjacency Matrix (2DAM)**
  - The **2DAM** values represent in what measure **each position** contributes to form **adjacent pieces** of **empty area**

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - Empty Area / Volume Compaction heuristics (EAC / EVC)
- The **four matrices** (URAM, ULAM, DRAM, and DLAM) are **added** to build the **2D Adjacency Matrix (2DAM)**
  - The **2DAM** values represent in what measure **each position** contributes to form **adjacent pieces** of **empty area**
- **EVC extends** the **area analysis** to include the **time domain** (greatest execution time in the taskset)
  - Build **3D Adjacency Matrix (3DAM): temporal** and **spatial adjacency**

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - At runtime, when task should be allocated, the set of **feasible allocations** are **evaluated** based on the values stored in the **2DAM** (EAC) or in the **3DAM** (EVC)

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - At runtime, when task should be allocated, the set of **feasible allocations** are **evaluated** based on the values stored in the **2DAM** (EAC) or in the **3DAM** (EVC)
  - **Unfeasible allocation** are **discarded**

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - At runtime, when task should be allocated, the set of **feasible allocations** are **evaluated** based on the values stored in the **2DAM** (EAC) or in the **3DAM** (EVC)
  - **Unfeasible allocation** are **discarded**
  - **EAC** and an **EVC score** is assigned to each **feasible allocation**

R3TOS allocation algorithms

- **R3TOS allocation algorithms**
  - At runtime, when task should be allocated, the set of **feasible allocations** are **evaluated** based on the values stored in the **2DAM** (EAC) or in the **3DAM** (EVC)
  - **Unfeasible allocation** are **discarded**
  - **EAC** and an **EVC score** is assigned to each **feasible allocation**
  - The **placement quality** is (inversely) **proportional** to the **EAC** or **EVC** scores

R3TOS performance evaluation

- **Performance** figures **measured** in the **developed proof-of-concept R3TOS implementation** (Xilinx Virtex-4 running at 100 MHz)

	Min.	Max.
Scheduling		
Scheduling algorithm execution	<1 μs	100 μs
Queues and task state updating	<1 μs	300 μs
Allocation		
Allocation algorithm execution	<1 μs	100 μs
Empty Area Descriptor updating	10 μs	200 μs
Inter-task communications		
Transfer LUT data buffer (ICAP)	3.7 μs	3.7 μs
Transfer BRAM data buffer (ICAP)	60.18 μs	60.18 μs
Transfer BRAM data buffer (DRTS): configuration layer	36.18 μs	39.9 μs
Transfer BRAM data buffer (DRTS): functional layer	81.92 μs	81.92 μs
Switching brams between neighbor tasks	10.03 μs	10.03 μs
Inter-task synchronization		
Polling of a HWS	1.6 μs	1.6 μs
Activation of a HWS	3.7 μs	3.7 μs