DSX tool specification

1. DSX tool specification
    1. A) Goals
    2. B) General Principles
        1. B1) System Ressources Layer)
        2. B2) DSX/L language
    3. Describing the software application
    4. Describing the hardware architecture
    5. Mapping the software on the hardware
    6. Code generation

A) Goals

DSX stands for *Design Space eXplorer*. It helps the system designer to map a multi-threaded software application on a multi-processor hardware architecture (MP-SoC) modeled with the SoCLib components.

It supports the hardware software codesign approach, and allows the designer to define successively :

- the hardware architecture : number of processors, number of memory banks, peripherals, etc.
- the software application structure : Task & Communication Graph
- the mapping of the software objects on the hardware components

The software application must be statically defined as a **Task & Communications Graph**. The number of tasks (defining the coarse grain parallelism), and the communication channels between tasks should not change during execution. The two targeted application domains are the telecommunication applications (where the tasks are handling packets or packet descriptors), and multi-media applications (where the tasks are handling audio or video streams).

A specific goal of DSX is to allow the system designer to control not only the the placement of the tasks on the processors, but also to control precisely the placement of the software objects (execution stacks, communication buffers, synchronization locks, etc.) on the memory banks. In shared memory multi-processors architectures with several physically distributed memory banks, such control is mandatory to optimize both the performances and the power consumption.

B) General Principles

The general principles are the following:

- The coarse grain parallelism is
- The software tasks are supposed to be written in C or C++.
- ?Mutek : OS kernel for embedded MPSoCs

B1) System Ressources Layer)

We want to map the multi-threaded software application on several hardware platforms, without any modification of the task code. One important platform is a POSIX compliant workstation, as we want to validate the multi-threaded application by running it on a the workstation before starting the mapping on the MPSoC architecture.

DSX defines a **system Ressource Layer** API (SRL), that is an abstraction of the synchronization and

communication services provided by the various target platforms. The SRL API helps the programmer to distinguish the embedded application code from the system code used for inter-tasks communications and synchronizations.

Three platforms are supported :

- Any Linux (or Unix) workstation supporting the POSIX threads,
- MP-SoC architecture using the MUTEK/D operation system,
- MP-SoC architecture using the MUTEK/S operating system,

MUTEK/D is an embedded, POSIX compliant, distributed, operating system for MP-SoCs?, while MUTEK/S is an optimized version: the performances are improved, and the memory footprint is reduced, at the cost of loosing the POSIX compatibility.

B2) DSX/L language

DSX/L is a language based on PYTHON, that allows the system designer to describe :

- the Task & Communication Graph (TCG)
- the MP-SoC hardware architecture (using the hardware components available in SoCLib).
- the mapping of the TCG on the MP-Soc architecture.

The DSX/L script execution generates the binary code executable on the workstation, the SystemC model of the *top cell* correspondint to the MP-SoC architecture, and the binary code that will be uploaded in the MP-Soc embedded memory.

Describing the software application

Describing the hardware architecture

Mapping the software on the hardware

Code generation