

<b>Users</b>	<ul style="list-style-type: none"> <li>• Software developers/embedded system engineers with little to no knowledge of the hardware</li> </ul>
<b>Key Features</b>	<ul style="list-style-type: none"> <li>• Design parallel application intuitively with a graphical editor of dataflow graph, independently from any implementation consideration.</li> <li>• Simulate and generate correct-by-construction parallel code for a wide range of embedded systems, including customizable heterogeneous multi- and many-core systems.</li> <li>• Optimize automatically the latency, the core loads, and the memory footprint of applications.</li> </ul>
<b>Benefits for the User</b>	<ul style="list-style-type: none"> <li>• PREESM automates inter-PE (processing element) communications. Communication libraries are distributed with PREESM that automatically synchronizes cores.</li> <li>• PREESM offers performance predictability at the early stages of heterogeneous digital system design.</li> <li>• PREESM fosters legacy code reuse. Existing functions can be easily called from PREESM generated code</li> </ul>
<b>Inputs</b>	<ul style="list-style-type: none"> <li>• Block-based hierarchical description of the application: PiSDF Dataflow graph</li> <li>• Imperative code describing the internal behaviour of dataflow “blocks”: C code</li> <li>• High-level description of the targeted architecture: S-LAM (based on IP-XACT standard)</li> <li>• Deployment scenario: Specify constraint for a given pair of application and architecture</li> </ul>
<b>Outputs</b>	<ul style="list-style-type: none"> <li>• Optimized parallel (Multi-threaded) code for heterogeneous embedded platforms.</li> <li>• Simulation of application deployment: Gantt diagram, core load &amp; memory metrics</li> </ul>
<b>Block Design (Preesm Overview)</b>	<p>The diagram illustrates the PREESM Workflow. On the left, under 'Developer Inputs', are 'Graph Annotations and Scripts', 'PiSDF Graph', 'Scenario', 'S-LAM Archi.', and 'Actors C Code'. These feed into 'Hierarchy Flattening', which leads to 'Single-rate DAG Transfo.'. This then branches into 'Build MEG' and 'Static Scheduling'. 'Build MEG' leads to 'Memory Allocation', which then leads to 'Compute Memory Bounds' and 'Display Gantt and Metrics'. 'Static Scheduling' leads to 'C Code Generation'. The 'Single-rate DAG Transfo.' also feeds into a 'HW Specific Compiler' (part of the 'Legacy Development Toolchain'), which also receives 'Code Gen. Support Libraries'. The compiler outputs to a 'Heterogeneous MPSoC' block, which lists various components like CPU, DSP, and Acc.</p>
<b>Example of Usage (PiSDF graph)</b>	<p>The left graph, 'Stereo Matching application graph', shows a sequence of blocks: 'Read RGB' (x2) feeds into 'Pre Process' (x1), which feeds into 'Disparity Computation' (x1), which feeds into 'Median Filter' (x1), which finally feeds into 'Display' (x1). The right graph, 'Hierarchical sub-graph of the Pre-Process actor', shows a more complex dataflow with blocks like 'RGB2 Gray', 'Get Left', 'Brd0', 'Brd1', 'Census', and 'Compute Weights', with various data streams and scaling factors (x2, x1).</p>
<b>Example of Usage (S-LAM graph)</b>	<p>The diagram shows a network of processors. On the left, four ARM processors (ARM_0, ARM_1, ARM_2, ARM_3) are connected to a 'SharedMem_0' block. This block is connected to a 'Nav' block, which is in turn connected to another 'SharedMem_1' block. This second shared memory block is connected to eight C66 processors (C66_0 through C66_7).</p>
<b>Role in the Toolchain</b>	<p>Crossing layers from high-level models to embedded SW implementation</p>