Due to the growing complexity of embedded systems, serious challenges are imposed to engineers who are supposed to efficiently design these systems. This resulted in a design gap which motivated the strive for higher levels of abstractions in electronic design automation. The Electronic System Level (ESL) provides description means that allow for first implementations and simulations of an embedded system, while, at the same time, abstracts from the increasing complexity [8]. SystemC, a C++-library, became the de-facto programming language for this abstraction level. However, even at the ESL, quite large designs have to be considered. In order to facilitate the Design Understanding needed for the corresponding ESL projects, visualization schemes have been proven beneficial [7]. In this work, we propose an improved visualization tool for SystemC which assists a designer in communicating a system’s structure and behavior by means of the following features:

- The visualization is interactive, allowing for a smooth browsing of the given design.
- Points of interest can be stored and re-visited later on.
- The system’s behavior can be visualized based on a given set of input stimuli, giving the designer the ability to interactively check e.g. the coverage of certain simulation runs.
- Information which is not directly derived from SystemC elements, but further in-depth extraction methods, can be visualized as well.

This leads to entirely new applications as illustrated in Figure 1 and Figure 2. More precisely, Figure 1 shows the coverage of simulation runs. Edges between the components represent signals and are colored based on the amount of toggles they become imposed with. By this, the designer gets an indication about active (green) and inactive (blue) parts of the design. He/She can use this information to interactively generate further stimuli in order to get a better coverage for less active parts.

In Figure 2, a visualization of data which is not available in the SystemC code but obtained by our extraction methods is shown. More precisely, the memory of the respective SystemC modules is mapped to respectively colored blocks. Each byte of a module and its value is represented by a virtual slice of the blocks and a corresponding color, respectively. In the figure, it can clearly be seen that large areas of the considered modules remain unassigned for SystemC objects and, hence, result in only few slices not being the gray.

While this significantly aids designers in their work, determining the required information for the visualization represented a significant challenge. This is caused by the fact that SystemC does not come with reflection or introspection capabilities that would allow accessing program’s properties at run-time. Instead, existing approaches followed the ideas of either parsing the source code statically to determine the properties of the design [5], [2], [1], [4] or executing at least parts of the program that are creating the simulated system and combining this data with the static information [9], [6], [3]. However, all these methods are tied to their specific code tools and/or compilers – making them hardly applicable for generic workflows.

In order to overcome these limitations, we developed an analysis tool for SystemC which relies on a combination of the SystemC API and the program’s debug symbols to extract the desired information about the system [10], [11]. This led to an unintrusive and portable solution which allowed for the extraction of the system’s properties without requiring major changes to a given SystemC project. Besides that, this methodology can easily be extended to visualize further design tasks e.g. the feature localization scheme proposed in [12].

REFERENCES