# SysML Model to SystemC

One of the interresting areas for this project as describes in the project scope is the transformation from the platform independent modeling language SysML to a platform specification model in SystemC. The most interesting part is which steps is needed, is it always a one-to-one relation between a SysML block and a SystemC module, and how is the notation in SySML to illustrate what goes where. There is not literature about going from SysML architecture to SystemC implementation, though one rapport[[1]](#footnote-1) has been used as inspiration. Whether this article has given us greater insight in the process, will be concluded together with our interpretation of how it could be done.

While SysML is a modeling language for describing system and architecture, including behavior and structure, SystemC is the platform independent implementation language written in C++ that support almost all hardware-software construct.

A lot of work have been done in making software tools that automatically can generate SystemC code from a SySML model by exporting the SySML module to a XML format. Such tools are normally quite large, and just learning the basic would require a long time. Therefore the project members have agreed upon not spending time learning new tools, because learning by doing it manually would benefit more. The process of automatically generating the SystemC template require that SysML modules is notated with both variables, and port type declaration.

The mapping between SysML parts and SystemC is done upon SySML structural diagrams. The mapping process can begin after the bdd are decomposed to a more detailed idb.

* **SysML blocks maps to SystemC modules (sc\_module)(Could be split)**
* **SysML flow ports maps to SystemC port (sc\_ports)**
* **SysML property maps to SystemC fifo/signal (sc\_fifo,sc\_signal)**
* **SysML action maps to SystemC method/thread (sc\_method, sc\_thread)**

The above mapping of structural diagrams can be illustrated by our idb Audio below . Audio.h is responsible for instansation clock, signals, and modules, used among all modules in Audio.

|  |  |
| --- | --- |
| **Mapping illustration** | |
| **Audio.h** | **Idb Audio** |
| SC\_MODULE(Audio)  {  // Input/Out decleration  // Initiate modules  ADCSource  DACSource  AudioEncoder  AudioDecoder  AudioSplitter  FeedBackFiltration  AudioControl  } |  |

Above mapping illustration is a one to one mapping, where one block has a corresponding sc\_module. Therby the process of creating header files and implementation files can begin. Each module in the ibd is then created in SystemC with a header file (.h) which must include module definition, port declaration, and process declaration. Then the .cpp file is created which contain all implementation code, that describes the modules behavior.

The illustration of the mapping process above is done manually, and therefore the SystemC template has to be written manually, which in most cases is trivial work when the SysML blocks is well defined. Well defined means that every block has decorated with port information and value types. We have not been using this approach in transforming SysML to SystemC but to get an overview of have it could have been accomplice, we have made an example for the Audio block. The dbb for Audio which can be used as input to automatically generate SystemC template is illustrated below.

|  |
| --- |
|  |

It is important that all connectivity as described in the above bdd is correct for the transformation to be correct. When design later changes the advantage of going up in abstraction also serves the purpose of making it more easy for designer to change design constrain compared to grasp a whole SystemC simulation, the benefit of model from a higher abstraction is obvious. The bdd uses a UML 2.1 extensions like ports, connectors, but still has the SySML diagram definition header. The diagram can be translated to XMI (XML Metadata Interchange), which most code generation tools can translate to an SystemC template from which futher implementation can begin.

**Advantage of auto transformation from SysML to SystemC**

After the idb is created for a module you can start the SystemC development. This has been our approach in this project. The downside is late decision of the blocks connectivity between each other. The datatype, flowtype and timing constraint is first considered late in implementation face. Even though you don’t autogenerate SystemC templates from the detailed bdd, our experienced so far, is that both the idb and the detailed bdd compliments each other. To transform the above detailed bdd, a tool like Enterprice Architect (EA) could be used. We have made experiments transforming som SySML block to SystemC code through EA, but the compiled SystemC code is quite defectively for some reason, the lack of flow information is one thing. It therefor decided not to use more time experimenting with EA.

One of the valuable thinks using a tool like EA is that later changes in the Audio block is synchronized both ways, that is changes in the SysML model would be reflected in the SystemC code, and changes in the SystemC code would be reflected in the SySML bdd. This synchronization would help developers who sometimes “forget” to update the model when they change the code base.

**Describing the SysML behavior diagrams with SystemC allocation**

The approach described above is concerning SysML structural diagrams, and their SystemC transformation, but it is also possible to annotate SySML behaviors diagrams with both SysML and SystemC allocation. Some says that one of the most important job for a developer is to make readable code for third part developer, and we don’t disagree, and this said the below illustration makes is easier for system engineers to get an overview of the whole system.

|  |
| --- |
|  |
|  |

1. SySML Profile for SoC Design and SystemC Transformation (University of Lugano) [↑](#footnote-ref-1)