Evaluation of an Object-Oriented Hardware Design Methodology for Automotive Applications

N. Bannow, K.Haug
Robert Bosch GmbH, Automotive Electronics – Driver Assistance Systems
Nico.Bannow@de.bosch.com, Karsten.Haug@de.bosch.com

Abstract

In this paper we present results in using the new object-oriented design approach OSSS (ODETTE System Synthesis Subset). The methodology and tools of the ODETTE (Object-oriented co-DEsign and functional Test TEchniques) project have been developed within the context of the IST programme of the European Commission. Main focus of OSSS lies in the field of hardware design and in synthesis capability. The strategy is based on an extension of the synthesizable subset of standard SystemC. The approach supports real object-oriented and synthesizable design features like classes, inheritance, templates, polymorphism and global object access. Therefore OSSS promises high efficiency in sense of capability to handle complex designs, faster development time, improved code quality and faster time to market. In contrast, standard SystemC is also based on C++ constructs, but no object-oriented constructs are available yet for a synthesizable system description.

We have evaluated OSSS on an automotive design example. It was chosen for the implementation of a component that is part of all video projects: A camera's exposure control unit (ExpoCU). The first main goal that was achieved is a synthesizable design by the automatic generation of an FPGA netlist from an OSSS description. Furthermore we have also proved the methodology to fulfill industrial requirements such as usability for complex system development, integration of existing IP, improved code quality and decreased development effort. Comparison will be done against existing VHDL based design flow. We especially focus on the implementation and testability by comparing the new object-oriented synthesis approach with a standard VHDL flow by laying emphasis on synthesizability.

OSSS and equivalent kinds of methodology show a large potential to handle new generations of complex HW-SW systems. Moreover the gap between increasing design complexity and available methodologies already now gets bigger and bigger and thus needs to be closed by new solutions such as OSSS.

1 Introduction

The complexity of electronic systems is significantly growing. Especially in the automotive industry new features required by customers go far beyond standard applications. In modern cars the number and especially the complexity of used controllers is steadily growing. This new situation requires new ways to deal with increasing complexity, growing verification effort and pressure of time to market while keeping cost efficiency and development time affordable. Thus, design implementation and test strategy have to be steadily adapted to new requirements. Otherwise, there is a risk that development costs and time to market get out of control. Therefore, Bosch started the evaluation of new approaches and methodologies to be able to handle the challenges of steadily growing system requirements in current and future times in the automotive industry.

There is a high demand for such new methodologies in case of compatibility with existing design flows, merging of hardware and software development and faster conception, implementation and verification. One big goal we have focused on is to bring modern software approaches like object orientation into the area of hardware design. From this approach we expect to be able to handle more complex designs on a higher abstraction level and therefore to close the gap between design requirements and existing design development methodologies. With SystemC [1][2] as a basis platform and object-oriented extensions provided by the OSSS [3] methodology, developed in the project ODETTE [4], we have selected a very promising methodology. Within the following sections we will present OSSS and show achieved results.

2 Design Example ExpoCU

Exemplary, we have chosen to evaluate the new methodologies on a camera's Exposure Control Unit (ExpoCU) that is a fundamental part of any video system. In the automotive area, video based applications are 'night vision enhancement', 'lane departure warning',

1530-1591/04 $20.00 (c) 2004 IEEE

'rear view camera', 'road sign recognition'. In between these applications have left the prototypic state and will emerge on the market within next years. Considering the importance and the average complexity of an ExpoCU, (Figure 1) it is an ideal candidate for evaluation by keeping even more complex designs in focus. Especially the different scope of the modules implementation is very interesting for us: A system clock frequency of 66 MHz has to be reached by every component but the cycle time of some modules is just one clock cycle and thus pipelining is needed. Other components do have a budget of some thousand clock periods. Furthermore, some modules are data flow oriented with high transfer rates where others have control flow functionality.

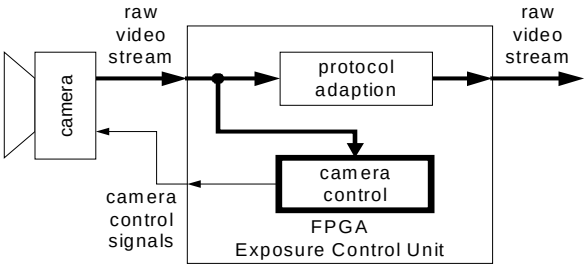

Figure 1 : Exposure Control Unit

The ExpoCU's 'Camera control' module consists of the following main parts:

- Camera data synchronization
- Histogram acquisition
- Threshold calculation
- Parameter calculation
- I²C bus control
- Reset control

The goal we have focused on is to implement the whole ExpoCU using the OSSS synthesizable hardware description syntax. Furthermore, some components like multipliers and FPGA specific constructs are to be integrated as existing VHDL IP.

3 Existing solutions

Object oriented solutions for high level synthesis are not very frequent on the market. One known tool is the 'Cynthesizer' by 'Forte Design Systems' [5] that provides object-oriented features for hardware synthesis by using the SystemC language subset. Nevertheless, only support of classes, inheritance and templates is being provided. Polymorphism and global object access are currently not supported.

Besides this solution, Synopsys [6] provides synthesis of SystemC as well, but the synthesizable subset is restricted to non object-oriented elements.

There are other evaluations and implementations of different object-oriented design methodologies ongoing like object-oriented extensions to VHDL [7], language 'e' synthesis extensions [8], automated implementation of communication protocols by V++ [9], OpenJ [10] and others [12][13]. Nevertheless, these approaches are not compatible to existing design flows or have been found to be to specific for some fields of application. Furthermore they were to complex or did not get established in real industrial environments for other reasons.

As the approach of OSSS promises a seamless flow from object-oriented specification down to gates with the capability of using all abstraction levels of a design description, we found it to be a very promising approach. Therefore, we decided to implement an ExpoCU design using the OSSS approach.

4 Evolution of OSSS

In the beginning of the ODETTE project, the focus has lain on object-oriented extensions based on VHDL. Nevertheless, during project time when the SystemC 2.0 methodology and simulation kernel got developed, much better options for object-oriented synthesis appeared and seemed more promising. This made it evident to skip to SystemC as a development base instead of extending a language that was never planned to support object orientation.

5 Approach

As design entry language, an extension of standard SystemC is being used. OSSS extends the SystemC language subset with additional C++ constructs. The specific feature is synthesizability of those constructs. The capability to compile the design and generate a binary executable file with any C++ compiler to support simulation stays untouched as known from SystemC.

In the following, we will differentiate between simulation issues for verification reasons and synthesis issues for hardware synthesis. In case of results, we differentiate between a binary executable program file for simulation and a netlist past synthesis.

We found that the very basic classes and templates are the most powerful features that motivated us to step towards real object-oriented design. Thus, we will focus on them while touching polymorphism and global objects only.

6 Features

The new methodology developed in ODETTE mainly provides the following new synthesizable features:

- Classes / class members / inheritance
- Templates (even complex types like classes)
- Polymorphism
- Global Object Access

Furthermore, prototypic support of automated fixed point number resolution has been implemented.

The usage of classes and their instantiation by creating objects is one of the basic principles of object-oriented programming. This feature is fully supported by OSSS and covers e.g. class members, member access privileges (public, private, protected), inheritance, operator overloading, etc. Classes can be instantiated inside a SC_MODULE or inside a process. Access takes part inside a process by using member function calls. The object data can be transferred via sc_signal<object> between different processes. Figure 2 shows a shortened class member declaration of the class 'SyncRegister'. This class is mainly being used to synchronize incoming data of a camera.

```
class SyncRegister
{
  private:
    sc_bv<REGSIZE> RegValue;

  public:
    SyncRegister();    // constructor

    // methods
    void Reset();
    void Write( const sc_bit& NewValue );
    bool RisingEdge( const unsigned int& RegIndex ) const;
    inline bool operator ==
      (const SyncRegister& ObjectRef) const;
};
```

Figure 2 : Example class members of SyncRegister

Templates (Figure 3) are a simple yet very powerful instrument to specify parameterized classes or functions. They enable designers to create generic functions, class instances and even SystemC modules. The types of templates can be basic types like 'int' or complex types like class identifiers. Indeed, using C++ templates is nothing unusual, but now, template usage is available even for synthesis for the first time. The advantages can be understood like automatic resolution of the operation '+' to its corresponding instantiation of an adder. This is in fact nothing exceptional, but compared to manual resolution we do not want to miss this powerful feature anymore.

```
template<unsigned int REGSIZE, unsigned int RESETVALUE>
class SyncRegister
{  ...  };
```

Figure 3 : Example parameterization via templates

Figure 4 exemplary presents the instantiation of the class 'SyncRegister' with two template parameters '0' and '4' inside a SC_MODULE:

```
SC_MODULE(sync) {
  SyncRegister< 4, 0 > data_sync_reg;  // SyncRegister instance
  ...
  SC_CTOR(sync) {
    SC_CTHREAD(sync_input, clk.pos());
    watching(reset.delayed() == true);  // (synchronous) reset
    ...
  }
};
```

Figure 4 : Parameterized class instantiation

Now after the instantiation of a class, the next code lines (Figure 5) inside a SC_CTHREAD show the access by class member function calls. In the reset part, the class instance gets initialized by a Reset() method. Later in the main while loop, the class data is updated using a Write() method. The contents of the class can be read and evaluated using different methods. One could be a rising edge detection:

```
// (access to 'SyncRegister')
void sync::sync_input() {
  data_sync_reg.Reset();  // reset data_sync_reg
  wait();

  while (true) {          // none reset behavior
    data_sync_reg.Write(data.read());    // update data_sync_reg

    if ( data_sync_reg.RisingEdge(0) ) { // detect rising edge
      ...
    wait();
  }
}
```

Figure 5 : Object access

Another classic but advanced design approach of object-oriented design style is polymorphism. With the OSSS methodology and tools, the synthesis of polymorphic objects is supported. This feature can be used to call different operations through the same interface on different objects. One example could be to simply select between different ALU (Arithmetic Logic Unit) instantiations (e.g. '+', '*', '-') but keeping the same access methods like 'read()' / 'write()' / 'execute()'.

Often, components of a system have to be accessed by different modules or processes. Those components are either shared resources (like an ALU) or used for intercommunication (like buses or memories). Such parts of a system can be implemented as global objects. The methodology of OSSS provides simple definition and access capabilities for global objects. The access and scheduling of a global object gets automatically included for synthesis. A designer can use a standard scheduler or implement an own – according to the required needs.

7 Flow

For simulation some parts of the OSSS extensions like classes or usage of templates may simply be compiled by a standard C++ compiler. Only the SystemC kernel needs to be included. Other features like global objects and polymorphism additionally require the OOWHLIB (object-oriented hardware library) available at [3] that in fact provides some classes and macros to support the OSSS features.

For synthesis of OSSS down to SystemC (Figure 6), two tools developed in ODETTE are being used. The first tool is an analyzer that parses OSSS source code and generates a library where it holds information of the whole design structure. The library will be written out and used by a second tool, the synthesizer. This synthesizer

generates standard SystemC files with a strong focus on synthesizability.

After the generation of SystemC files out of the original OSSS files, we can simply apply the standard synthesis flow to generate an FPGA netlist (Figure 6).

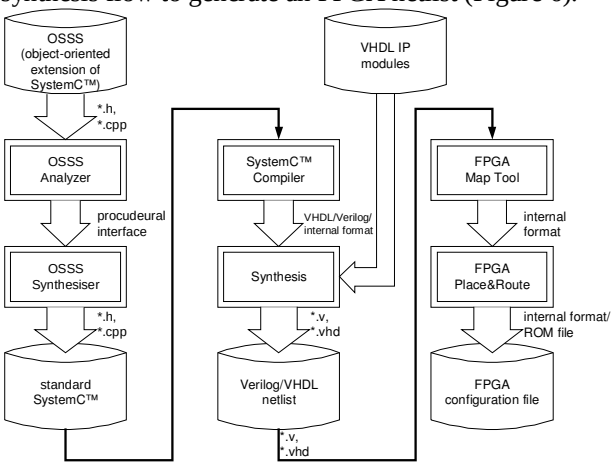

Figure 6 : OSSS flow down to an FPGA netlist

For the integration of existing VHDL IP modules into the ExpoCU design, we generate the netlists separately. Then – on the netlist level – the synthesis tools connect the whole design automatically (Figure 6).

8 Resolution of OSSS constructs

As explained in the flow, the generated SystemC files are in fact resolved OSSS constructs.

Resolution of class member functions is done by the generation of non-member functions. Template parameters are forwarded to their location where they are being used. The data members of a class instance are mapped to a single bit vector. This vector stays where it has been declared: Inside a SC_MODULE or as a process member. The access to object data is therefore being translated to a read/write to parts (slices) of the generated vector. The class 'SyncRegister' and SC_MODULE 'sync' get analyzed and translated into the following synthesizable SystemC code (Figure 7, Figure 8).

```
void
_SyncRegister_Reset_1_( sc_biguint< 4 > & _this_ )
{  _this_ = 0;  }

void
_SyncRegister_Write_1_(
  sc_biguint< 4 > & _this_, const sc_bit & NewValue )
{
  sc_biguint< 4 > _temp_0_;
  _temp_0_[0] = NewValue;
  _temp_0_.range(3, 1) = ( (sc_biguint<4>)_this_ ).range(2, 0);
  _this_ = _temp_0_;
}

bool
_SyncRegister_RisingEdge_1_(
  const sc_biguint<4> & _this_, const sc_bigint<32> & RegIndex )
{  ...  }
```

Figure 7 : Non-member functions

```
SC_MODULE( sync )
{
  sc_biguint< 4 > data_sync_reg;
  ...
  void sync_input()
  {
    _SyncRegister_Reset_1_(data_sync_reg);
    wait();

    while ( true ) {
      _SyncRegister_Write_1_( data_sync_reg, data.read() );
      ...
  }
  ...
  SC_CTOR(sync) {  ...  }
}
```

Figure 8 : Translated module 'sync'

In the code translation (Figure 8) it can be seen that no additional logic has been added when using classes and templates. The resolution of object-oriented design features like classes and templates do not create an additional overhead. This is not only valid for the given example but for the whole OSSS approach. During compilation down to standard SystemC this structures are fully resolved. In case of polymorphism, multiplexers are being inserted to select the function and object. When global objects are being instantiated and accessed, some scheduling logic of course has to be added. But in any case: If described in conventional approach, logic would have to be added anyway for global-object-like as well as for polymorphism-like design descriptions.

9 Additional implementation issues

Implementation of 'sc_trace' and furthermore the overloading of the streaming operator '<<' is recommended to enable the tracing of an object's contents and to allow a dump of object data at any time (Figure 9):

```
#ifndef SYNTHESIS
  // overloading operator '<<'
  template <unsigned int REGSIZE, unsigned int RESETVALUE>
    inline ostream& operator << (ostream& OStream,
      const SyncRegister<REGSIZE, RESETVALUE>& ObjectReference);

  // overloading method 'sc_trace'
  template <unsigned int REGSIZE, unsigned int RESETVALUE>
    extern void
    sc_trace(sc_trace_file* TraceFile,
      const SyncRegister<REGSIZE, RESETVALUE>& ObjectReference,
      const sc_string& ObjectName);
#endif SYNTHESIS
```

Figure 9 : Operator '<<', sc_trace

To enable 'sc_trace' access to object data it has to be declared as a friend function (Figure 10):

```
#ifndef SYNTHESIS
  #ifndef WIN32
    friend void sc_trace<>(
      sc_trace_file* TraceFile,
      const SyncRegister& ObjectReference,
      const sc_string& ObjectName);
  #endif WIN32

  #ifdef WIN32
    friend void sc_trace(
      sc_trace_file* TraceFile,
      const SyncRegister& ObjectReference,
      const sc_string& ObjectName);
  #endif WIN32
#endif SYNTHESIS
```

Figure 10 : sc_trace friend declaration

Furthermore, for comparison of whole complex objects, we may overload operators like '==' (Figure 11):

```
// overloading operator '=='
inline bool operator == (const SyncRegister& ObjectRef) const;
```

Figure 11 : Operator '==' overloading

10 Benefits

OSSS, together with SystemC, has many benefits compared to conventional HW-design methodologies like VHDL. The most important benefits we identified are:

- Real object-oriented design approach allows to reach higher abstraction level,
- Intermediate output format is (readable and simulatable) standard SystemC,
- Seamless synthesizable design flow from OO concept and design down to hardware (gates),
- SystemC kernel can be extended for simulation by public or internal extensions to custom requirements because source code is open source,
- Simulation tools are not strictly needed as the SystemC kernel provides signal-tracing capabilities comparable to existing HDL simulators, debugging capability usually is provided by C++ compilers,
- Standard C++ compilers can be used to create an executable binary file for simulation, debugging is straight forward either by using a C++ debugger or by printing simple text to output, using 'cout' calls,
- Possible separation between synthesizable and non synthesizable constructs by using #define directives or macros,
- Separation of pure hardware / pure software design gets abolished by support of C++ constructs,
- HW/SW co-design becomes much easier,
- Encapsulation, preferable on mid and lower granularity, can additionally be applied by class member function calls instead of using ports, interfaces and channels only,
- Class libraries can be used for IP transfer,
- Reduced development time, faster time to market and therefore better cost efficiency,
- Much higher simulation speed than conventional RTL simulators.

11 Open issues

There are some open issues that currently prevent OSSS to be used for series projects within Bosch. Most of them are based on restrictions in SystemC usage:

- Currently OSSS tools are in a prototypic but advanced state,
- SystemC synthesis tools are very rare,
- Some parts of synthesizable subset of HDL's like VHDL are still not supported by tools for synthesis yet like 'generate' loops or recursive function calls,
- Some restrictions exist in available standard SystemC synthesis tools that require workarounds.

12 Results

The question why we look for new methodologies instead of keeping established design flow is obvious: Already now and even in very close future, SOC's (System on a Chip), multiprocessor circuits and HW/SW co-design become too complex or may even not be handled at all by common pure HDL's. They are not powerful enough to handle the new challenges in acceptable time with affordable resources. Big advantages can be seen when looking to standard SystemC already. But even SystemC's synthesizable subset is restricted to structures that do not allow the usage of real object-orientation. Looking to VHDL, which is Europe's dominating hardware description language, the benefits of the new methodology OSSS are very promising.

We have set up the evaluation of OSSS in parallel to a standard development flow with existing methodologies like C++ and VHDL. This, of course, was necessary because it prevents dependency on new methodology and its tools. Because of the two flows, we have the chance to compare both flows in case of results and their efficiency.

If we compare the required area of a synthesized ExpoCU netlist in a conventional and an OSSS approach, they are almost equivalent. The frequency of the FPGA achieved in OSSS design is below the frequency in the VHDL flow. Further evaluation on this issue will be done.

The development time in the beginning of the project was restricted by the prototypic tool implementation. To the end of the project major tool issues got solved and the development effort could be spent on implementation mainly. Thus, development time dramatically decreased. The implementation of a complete I²C master module e.g. took a single day. We assume an implementation effort of two days in case of pure SystemC implementation by keeping same hierarchical module structure. The VHDL implementation took slightly longer using the RTL coding style. The difference in behavioral OSSS and SystemC coding time versus VHDL RTL is mainly based in the controlling functionality that the I²C interface is based on. Especially in the implementation of controlling functionality the behavioral description has advantages versus RTL coding. Nevertheless, in data flow oriented modules where RTL coding might be preferred, OSSS may also be applied very efficiently.

We have been debugging the generated intermediate files on all possible levels of synthesis to get an idea of realization strategy. What we found out is that the behavior on every stage is bit and cycle accurate and fully

complies with its original description. In other words does it mean that even with OSSS full control of synthesis results in case of design functionality stays in the hands of designers. Unfortunately, in synthesis steps during behavioral synthesis of SystemC code, the tools have some restrictions and produce some unnecessary overhead. Thus, nevertheless the influence on area and speed are partly tool specific issues. Figure 12 represents a screenshot of the synthesized main components that are connected on the top level of the ExpoCU. Synthesis has been done by using the OSSS approach.

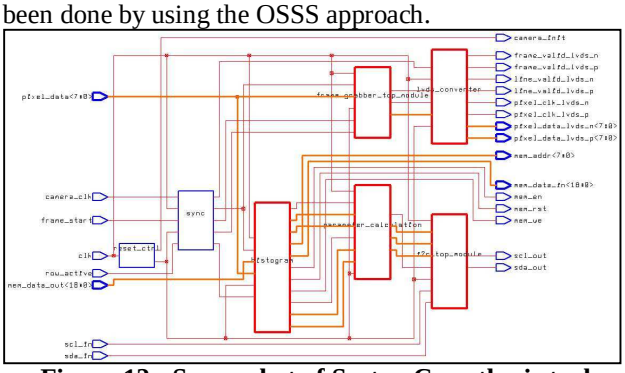

Figure 12 : Screenshot of SystemC synthesis tool with main ExpoCU modules

On higher hierarchy level, SystemC ports, interfaces and channels should be used to: Keep independent communication and synchronization, to simplify a modular synthesis, to be able to use different channel implementations or refinement strategies and for further hardware specific issues. Thus the usage of OSSS classes and their access via class member functions should be preferred on lower hierarchy level. Furthermore, if blocking object access is applied via wait() calls, other modules still must continue their execution. This can only be provided by independent, parallel module execution.

More information about ODETTE participants and OSSS results can be found on [4][13][14].

13 Conclusions

We have shown that an object-oriented design methodology has been successfully applied to a real industrial design example. One major result is the generation of a silicon netlist, working on an FPGA. First results and experiences with OSSS are very promising. The OSSS approach is not fixed but open for possible additional features. Standard SystemC sources are a subset of OSSS and thus fully supported. The development time is supposed to be decreasing because of different reasons: Many issues are done implicitly within OSSS while they need to be done explicitly in SystemC. Class encapsulation, the usage of templates, additional structures and basis for concept and prototyping, better integration into existing C++ test-environments, etc. are the main features of the new methodology and toolset. OSSS extends the powerful possibilities which SystemC already offers by providing a higher level of abstraction.

14 Future Work

Bosch sees a high potential in this new methodology developed in the ODETTE project.

The approach and evaluation of OSSS is based on a prototypic tool chain. But even in this state we obtained very promising results. Thus, Bosch will keep track on any further evaluations of OSSS and all similar approaches. Further we will continue investigation on the synthesis results of OSSS and SystemC versus VHDL flow. Especially we will focus on the reasons of the achieved lower frequency of the ExpoCU and on the generated overhead during behavioral synthesis.

Bosch decided to play an active role in providing valuable feedback for ongoing activities in tool and methodology development. This includes internal and external evaluations to be able to move to new promising design techniques and methodology once they are available for industrial usage.

References

[1] Open SystemC Initiative. *SystemC, Version 2.0.* www.systemc.org, 2001

[2] T. Grötker, S. Liao, G. Martin, S. Swan. *System Design with SystemC.* Kluver Academic Publishers, 2002

[3] E. Grimpe, B. Timmermann, T. Fandrey, R. Biniasch, F. Oppenheimer. *SystemC Object-Oriented Extensions and Synthesis Features.* Forum on Design Languages FDL '02

[4] http://odette.offis.de. Homepage of ODETTE project.

[5] Forte Design Systems *Forte Cynthesizer* www.forteds.com

[6] Synopsys, Inc. *Synopsys SystemC™ Compiler.* www.synopsys.com

[7] S. Swamy, A. Molin, B. Covnot. *OO-VHDL: Object-Oriented Extensions to VHDL.* IEEE Computer, Oct. 1995

[8] T. Kuhn, T. Oppold, M. Winterholer, W. Rosenstiel, M. Edwards, Y. Kashai. *A Framework for OO-Hardware Specification, Verification, and Synthesis.* DAC 38th, 2001

[9] S.-T. Cheng, P. C. McGeer, M. Meyer, T. Truman, and P. Scaglia, R. K. Brayton, A. L. Sangiovanni-Vincentelli. *The V++ System Design Language.* EECS and ERL of University of California, Berkley, 1998

[10] J. Zhu, D. D. Gajski. *OpenJ: An Extensible System Level Design Language.* Date '99

[11] M. Radetzki, A. Stammermann, W. Putzke-Röming, W. Nebel. *Data Type Analysis for Hardware Synthesis from Object-Oriented Models.* Date '99

[12] J. Zhu. *MetaRTL: Raising the abstraction level of RTL Design.* Date '01

[13] E. Grimpe. OO Features supported by the SystemC™ Plus Methodology. ODETTE OFFIS, V0.2, http://odette.offis.de/systemc-plus/, Oct. 2002

[14] W. Fornaciari, L. Pomante. *Generic Class Library User Guide.* ODETTE Siemens ICN, R1.5, http://odette.offis.de/systemc-plus/, May 2003