Interface based system modeling of a CAN using SVE

Denny Brem, Dietmar Müller
Professorship Circuit and Systems Design, Chemnitz University of Technology

www.ip-qualifikation.de

1. Motivation

The rapid development speed of silicon technology permits to integrate more and more complex systems on a microchip (System on Chip). However, the productivity of design methods for such System on Chip develops slower. Therefore, the effort and the time for development, verification and test increases. In spite of manifold efforts the discrepancy between the size of the technologically realizable systems and the design effort of such systems (the so-called Design Gap) becomes larger. For the solution of these problems, the consequent use of reusable components, so-called Intellectual Property (IP) modules, is considered to be essential. In order to be able to use IP in as many target systems as possible, they must be designed for flexibility and must be easily adaptable.

Interface based design is a methodology that should help designing IP to be flexible and to minimize the adaptation effort.

2. Interface based design

Interface based design is a design methodology that orthogonalizes functionality and communication of a digital system, aiming at the separate, independent exploration of the corresponding design spaces. An interface based design methodology using a formal language called *SVE* is described in this poster.

3. Interface based design using SVE

The methodology described in this poster is based on a formal mixed-multi-level specification of a communication protocol. The formalism used for protocol specification has been realized as an extension to the system description language SystemC, called SVE. This ensures that protocol specifications become an integral part of the system description and can be simulated and verified directly in the system context.
The name SVE is a reference to SuperVISE, ICL's design methodology that first used the concept of interfaces. To formalize the concepts of this design methodology ICL has developed an extension to VHDL, called VHDL+. SVE now combines the advantages of the interface modeling concepts of VHDL+, the high simulation speed and HW/SW-domain unifying language approach of C++ based system modeling and the user extensibility of SystemC.
In context to State-of-the-Art approaches this approach generates both transaction producer and consumer controllers (which both usually are distinct types of interacting controllers) in form of RTL models from one single protocol specification in the same run of the synthesis algorithm. This ensures that the behaviours of both transaction producer and consumer conform to the protocol specification. In case of verification of the controller implementation the protocol specification also serves as a protocol checker during system simulation.

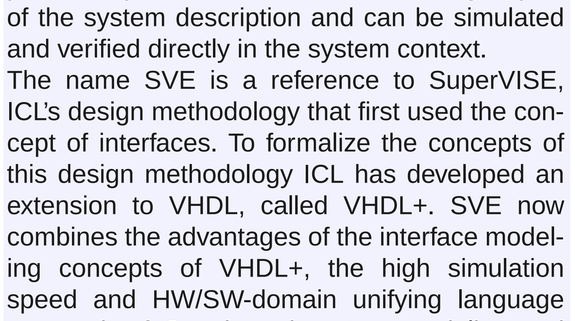

Figure 1: Mixed multi-level communication modeling in SVE

SVE enables communication modeling in terms of high level message passing between behaviours. It introduces a new design unit to SystemC, called the *Interface* (Figure 1). The interface describes communication between a number of behaviours at different levels of abstraction in terms of *interface items* (Figure 2). An interface item describes communication between behaviours at a specific level of abstraction. As the design advances both, system functionality and system communication, are to be implemented at lower levels of design abstraction. For this purpose interface items can be iteratively refined completely independent from the module's functional implementations. This refinement is done by defining *item compositions* for interface items. Item compositions describe how a particular item is decomposed into a set of lower level items and how these items are scheduled (executed in time). In reverse, the item composition describes how the item is constructed of lower level items.

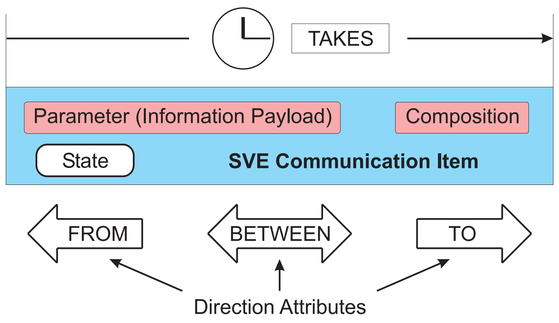

Figure 2: SVE Interface Item

4. Modeling a Controller Area Network (CAN) Protocol in SVE

CAN is a serial bus system especially suited to interconnect smart devices to build smart systems or sub-systems. The properties of a CAN are:

- multi-master capabilities
- broadcast messaging
- sophisticated error detecting mechanism and retransmission of faulty messages

The CAN protocol is an international standard defined in the ISO 11898. The protocol defines messages which are distinguished by using a message identifier. Such message identifier defines not only the content but also the priority of the message.

Figure 3 shows the topology of the SVE CAN model. The CAN protocol is captured in a SVE interface called „CAN_interface". Each CAN device is an instance of the same SystemC module class. Such a device contains two instances of a „CAN_interface", called interface signal. Each interface instance is dedicated to produce transactions (send interface) or to consume transactions (receive interface).
As communication medium a SystemC sc_signal is used, so the send interface has to decompose transactions down to signal level and the receive interface will compose transactions based on the values of this sc_signal.

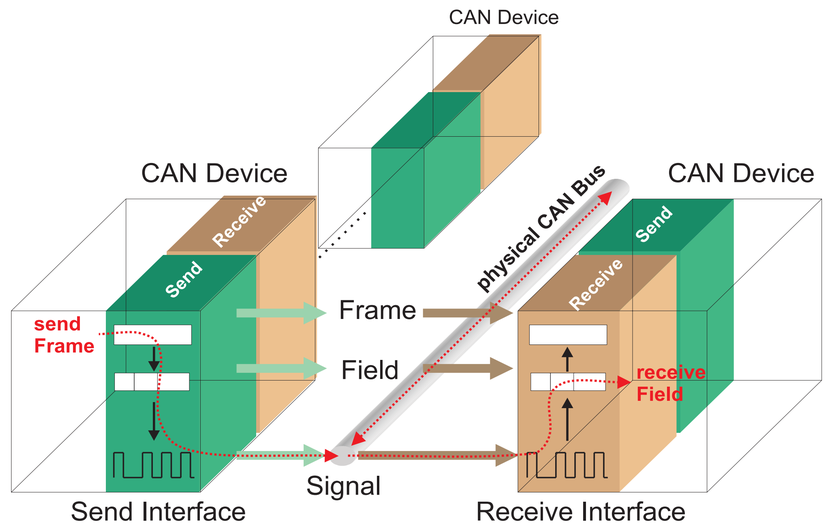

Figure 3: CAN System Overview

SVE. Figure 4 shows as an example the declaration of the remote frame item (extended format).

```
// Message Item REMOTE_FRAME_extended
SV_MESSAGE( REMOTE_FRAME_extended ){
    // declare item parameters
    SV_Param<sc_uint<18> > ext_id;
    SV_Param<sc_uint<4> > data_bytes;
    SV_Param<bool> ack_slot_value;
    SV_Param<bool> abort_frame;

    // declare compositional item references
    SV_MessageRef IDE_r_ref;
    SV_MessageRef EXTENDED_IDENTIFIER_ref;
    SV_MessageRef RTR_r_ref;
    SV_MessageRef r1_ref;
    SV_MessageRef CTRL_ref;
    SV_MessageRef CRC_ref;
    SV_MessageRef ACK_ref;
    SV_MessageRef EoF_ref;

    // method to get interface status
    void get_abort_frame(){
        abort_frame = SV_INTERFACE_CONTEXT(CAN_interface)->get_abort();
    }

    // message constructor
    SV_MESSAGE_CTOR( REMOTE_FRAME_extended ){
        // message attributes
        SV_FROM << "GENERIC_SEND_END";
        SV_TO << "GENERIC_RECEIVE_END";
        SV_TAKES( 8);

        // parameter registration
        SV_PARAMETER(ext_id);
        SV_PARAMETER(data_bytes);
        SV_PARAMETER(ack_slot_value);

        // link compositional items
        SV_LINK_REFERENCE( IDE_r_ref, CAN_interface );
        SV_LINK_REFERENCE( EXTENDED_IDENTIFIER_ref, CAN_interface );
        SV_LINK_REFERENCE( RTR_r_ref, CAN_interface );
        SV_LINK_REFERENCE( r1_ref, CAN_interface );
        SV_LINK_REFERENCE( CTRL_ref, CAN_interface );
        SV_LINK_REFERENCE( CRC_ref, CAN_interface );
        SV_LINK_REFERENCE( ACK_ref, CAN_interface );
        SV_LINK_REFERENCE( EoF_ref, CAN_interface );

        abort_frame = false;

        // message composition
        SV_COMPOSITION( SV_SERIAL(IDE_r_ref(),// extended frame
                                  EXTENDED_IDENTIFIER_ref(ext_id),
                                  RTR_r_ref(),
                                  SV_SEQBEHAVIOUR(get_abort_frame),// set variable abort_frame
                                  SV_SELECT<bool>( SV_MATCH(abort_frame),
                                                   bool(false) <= SV_SERIAL( r1_ref(),
                                                                             CTRL_ref(data_bytes),
                                                                             CRC_ref(),
                                                                             ACK_ref(ack_slot_value),
                                                                             EoF_ref()),
                                                   bool(true) <= SV_NULL)// abort composition
                                  )
                      );
    }
};
```

Figure 4: Item REMOTE_FRAME_extended

During simulation a Finite State Machine (FSM) generated by the SVE kernel attempts to compose items based on the received values according to the item compositions. If a value does not match the expected value (some bits are of a fixed value in accordance to CAN Spec 2.0) a protocol error is detected, i.e. an implicit frame check is performed. Figure 5 shows how de-/composition is defined over the complete protocol hierarchy.

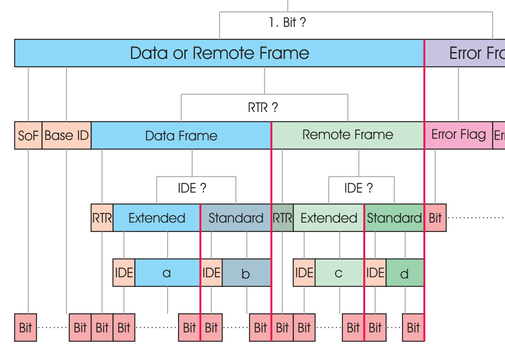

Figure 5: Protocol hierarchy

matically done by the SVE simulation kernel during simulation. **Bit stuffing** is described in the lowest level interface item called "Bit" and also automatically checked. If an error of these two types is detected, a protocol error message is generated. The corresponding receive interface stops matching the actual interface item immediately and will automatically start matching an new item.

CRC generation and check is also implemented in the interface directly. However, because of the delayed signalling of crc errors defined in CAN Spec 2.0, error handling can not be left to the SVE simulation kernel, which would stop receiving the item immediately, but must be implemented explicit by embedding sequential behaviours in item compositions.

Monitoring also uses sequential behaviours: The receive interface needs to know about the value sent by the send interface to compare sent and received values. In case of mismatch the receive interface indicates an error to the send interface, which then stops sending the item. The following transmission of an error frame has to be induced by the CAN device itself.

The transmission of **ACK** can not be described as separate transaction in relation to the protocol description. Therefore ACK is sent by a process in the CAN device, which is triggered by the receive interface. However, ACK is part of data and remote frames and always sent by the send interface as 'recessive' value, which can be overwritten by a 'dominant' value (i.e. positive ACK) on the bus. In the composition both 'recessive' and 'dominant' values are accepted for ACK.

Simulating the system

The simulation starts with the elaboration phase. Beside the normal SystemC elaboration, the executable (system model) generates FSM to produce and to consume transmissions for each interface instance. The number of states of these FSM depends on the complexity of the protocol specification: The number of states increases with the number of defined abstraction levels and the complexity of item compositions. After elaboration the model runs a specified time just like ordinary SystemC programs. The states of interface items and values of item parameters can easily be traced by tracing the corresponding interface signal. A result of tracing an interface signal is shown in Figure 6.

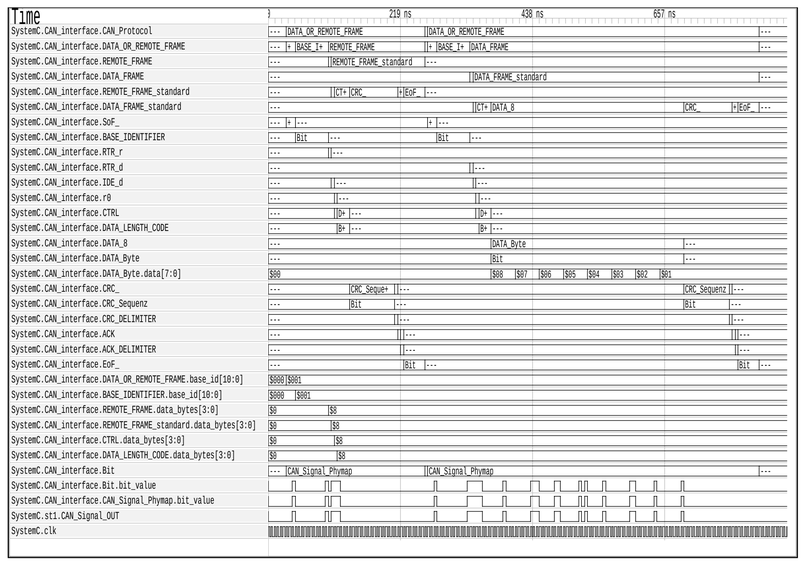

Figure 6: Trace of an interface signal

Frame Formats

In CAN Spec 2.0 Part A and B four frame formats are defined: data frame, remote frame, error frame and overload frame.

According to these frame formats interface items are declared. Following the composition and scheduling defined in CAN Spec 2.0 leads to a specification of the complete CAN protocol in

Detecting and signalling errors

For error detection the CAN protocol defines five mechanisms: CRC, frame check, acknowledgement, monitoring and bit stuffing.
As part of the system communication these mechanisms should be implemented inside the interface:
As previously mentioned, **frame check** is auto-

5. Conclusion and Outlook

The CAN protocol was described in SVE and successfully simulated within a system environment. Using SVE interfaces a testbench could be retained along most stages of design exploration. Conversions between incompatible protocols can easily be described by two interfaces which internally exchange abstract data.

The synthesis of interface specifications to controller hardware descriptions is in preparation. In the near future SVE will be extended to enable designers to describe protocols of parallel character, of what SVE is currently not capable.

Acknowledgements:

This work was realized in association with AMD and is part of the „IP Qualification Project (IPQ)" funded by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung) in section „Design Platforms for complex Systems and Circuits (Entwurfsplattformen für komplexe angewandte Systeme und Schaltungen der Mikroelektronik - EkompaSS)"

Contact: Denny Brem
Technische Universität Chemnitz
Professur Schaltungs- und Systementwurf (413201)
09107 Chemnitz, Germany

brem@infotech.tu-chemnitz.de
http://www.infotech.tu-chemnitz.de/~sse
phone: +49 371 531 3158
fax: +49 371 531 3186