Methodical Aspects of VHDL-based Field Bus Modelling with a Controller Area Network

Sven Altmann, Ulrich Donath
Fraunhofer-Institut für Integrierte Schaltungen,
Zeunerstr. 38, D 01069 Dresden

Abstract
Methodical aspects are described to develop VHDL models of field bus systems. Field bus systems used in automatic control engineering are examples of distributed computing systems. Modelling of such a system is demonstrated with a Controller Area Network (CAN). The consideration ranges from the process to the control devices. Targets are functional verifications as well as performance predictions. Since the bus is a decisive component modelling is focused on questions concerning the signal abstraction and the protocol used. VHDL code formulation is based on Statecharts. A design and simulation environment including the Synopsys VHDL simulator is presented.

1 Introduction
Field buses considered here are communication media between components of distributed automation systems [1]. In terminology of automatic control engineering, they are installed on the lowest level of the automation hierarchy; i.e. they connect sensors, actuators, and control devices to control a process. For modelling field bus systems, different approaches are applied such as Queueing Networks [2] or Coloured Petri Nets [3]. These approaches consider the devices or instruments as abstract service stations with mean arrival rates and service rates or times, respectively. However, function and performance of complex systems are determined by a variety of influence variables, e.g. by network structure, service types, priorities, response times of application programs, load-dependent adjustments, and data-dependent function settings. So a more general approach is needed that takes into account the bus activity and the behaviour of the subsystems connected by the bus. For modelling digital systems, VHDL has proved to be an expressive and effective means. Therefore, VHDL is chosen to model field bus systems as well. Simulation on this basis allows to verify the design of the components and, on the other hand, to predict performance features such as latency or response times. A Controller Area Network (CAN) is applied to illustrate the methodical aspects. More generally speaking, the task belongs to the field of modelling parallel processes. The main idea consists in utilizing VHDL features for covering concurrency and hierarchy. If both time adjustments and transferred data items are taken into account a combined simulation can be achieved showing performance and function of complex systems. In the near future, VHDL will be extended to analog domain so that analog signal processing devices can be incorporated.

2 System View
The object of investigation is a controller network arranged as depicted in figure 1. Distinct nodes participate in controlling a process or sub-processes interacting. A bus connects all nodes and transfers messages between them in bit serial order. To find a general approach, all nodes are assumed to be subdivided into an application processor and a protocol unit. The application processors interact with the process via sensors and actuators. The protocol units
access the bus to send or receive data. While the application processors treat data according to different programs, the protocol units realize the same protocol.

Concerning the CAN technique, the following features are given [2][4][5]:
- Multi-master system
- Medium access: CSMA/CD (Carrier Sense Multiple Access / Collision Detection) + AMP (Arbitration on Message Priority)
- Multicast message transfer
- Message identifiers determine their priorities
- Acceptance filtering by the receiver
- Data transfer service and remote data request service
- Data length: 0 - 8 Byte
- Bit rate: 10 - 1000 KBit/s
- Error detection and error signalling.

As depicted in figure 1, the VHDL model of the system will be composed with the components Process, Sensor, Actuator, and Node. The component Node is divided into the sub-components Application Processor and Protocol Unit. In terms of VHDL, the transmission medium Bus will be modelled as a Resolved Signal driven by a Resolution Function. The signal flow directions are shown in figure 1, too. First of all, the signal types have to be declared to achieve the interface descriptions of the components and to set up the basis of component modelling.

1) μP - Microprocessor, PID - Proportional-Integral-Derivative Element
3 Bus and Protocol Units

3.1 Signal abstraction

In accordance with the Open Systems Interconnection (OSI) Reference Model, the CAN controller is organized hierarchically in the Data Link Layer and the Physical Layer, see figure 2. In the approach chosen, the Protocol Unit behaviourally performs the tasks of the Data Link Layer as defined in the ISO Standard. The Physical Layer specifies an electrical circuit realization that connects the node to the bus. Since electrical features are not significant for behavioural modelling, this specification is not included.

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Link Layer</td>
</tr>
<tr>
<td>Logical Link Control (LLC)</td>
</tr>
<tr>
<td>Medium Access Control (MAC)</td>
</tr>
<tr>
<td>Physical Layer</td>
</tr>
</tbody>
</table>

Figure 2. Layers of the CAN Protocol

The Logical Link Control offers two services to its user: Data Transfer and Remote Data Request. The data transfer service carries only a data frame from a transmitter to a receiver, whereas the remote data request service transmits a remote frame to a receiver to request a data frame from this node. Figure 3 shows in which manner the contents of a data frame will be extended and arranged if a message passes from the Logical Link Control to the Medium Access Control. Omitting the data field, a remote data request is equal to a data transfer.

<table>
<thead>
<tr>
<th>LLC Layer</th>
<th>Identifier</th>
<th>DLC</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC Layer</td>
<td>SOF</td>
<td>Identifier</td>
<td>RTR</td>
</tr>
<tr>
<td></td>
<td>1 Bit</td>
<td>11 Bit</td>
<td>1 Bit</td>
</tr>
</tbody>
</table>

Figure 3. Structure of the data frames

In order to declare the Resolved Signal, a unique signal type has to be defined. Therefore, the MAC data frame is chosen and adapted so that various tasks can be fulfilled:

- The Start Of Frame (SOF) bit marks the bus out as busy with a dominant ‘0’.
- Data frames and remote frames are distinguished by the Remote Transmission Request (RTR) bit with ‘0’ or ‘1’.
- The Control field gets the Data Length Code (DLC) from the LLC frame, but the Data field itself is fixed to 8 bytes.
- If a remote frame is sent the Data field will be empty.
- The Identifier field, the CRC field, and the ACK field are kept up, whereas the End Of Frame (EOF) mark is omitted.
- A time stamp is added to allow time measuring and event handling.

The result is a record type declared in the following type definition:
% TYPE bit_array IS ARRAY (0 TO 7) OF bit_vector(7 DOWNTO 0);

% TYPE telegram IS RECORD
  sof: bit;
  identifier: integer;
  rtr: bit;
  control: integer;
  data: bit_array;
  crc: integer;
  ack: bit;
  stamp: time;
END RECORD;

According to the ISO standard, any node may start to transmit a frame when the bus turns idle. A conflict will arise if two or more nodes start the transmission at the same time. In this case, an arbitration mechanism uses the signal identifiers to solve the conflict. The identifiers are positive integers having inverse priority, i.e. the identifier zero has the highest priority. The transmitter with the frame of highest priority gains the bus.

This arbitration is realized by the Bus Resolution Function, which checks all signal drivers on a dominant Start Of Frame. Comparison of the signal identifiers is carried out numerically, not bitwise. This is worth mentioning since the Bus Resolution Function significantly influences the simulation efficiency. For the same reason, the frames are transferred as a whole, not in a sequence of data items.

3.2 Specification of control activities with Statecharts

Statecharts are applied to specify the control activities of the Protocol Unit. The subdivision of the Data Link Layer into Logical Link Control and Medium Access Control are taken from the ISO Standard. Harel’s rules [6] are used to define the state transitions and to decompose the states into substates. The schematic entry can be done with the ExpressV-HDL tool [7].

Rounded boxes represent states, directed arcs symbolize state transitions. The general syntax of an expression labelling a state transition is shown in figure 4. This expression may be read as follows: when the source state is marked and Event occurs and Condition is true then a transition is carried out from the source state to the target state. Doing this Action is carried out. In the following, Events are signal changes or timeouts, Conditions are logical relations, and Actions are signal assignments or procedure calls. If several actions have to be done they are separated by semicolons.

Decomposition of states into substates is demonstrated with the top level chart of the Medium Access Control, see figure 5.
With respect to the fault management a node may be in one of the three states: Error Active, Error Passive, or Bus Off. A Transmit Error Counter and a Receive Error Counter, which belong to every node, determine the transitions among these states. Only in the Bus Off state a node is completely decoupled from the bus. The normal working state is the Error Active state. This state will be achieved after initialization or occurrence of a Restart event and will be left if a Go Passive event is produced internally. The prefix @ of a state identifier indicates a reference chart underlying.

![Figure 5. Top Level Statechart of MAC](image)

The Error Active state, see figure 6, is decomposed into four states: Ready, Send, Receive, and Error. The initial state is Ready. Send is considered here for instance. Preconditions to enter Send are the following: Ready is marked, a data frame or remote frame has to be transmitted, and the bus is free. Send will be quitted if one of the events Go Receive, Send OK, or Send Error is produced internally. The control activities for sending are represented in an additional chart.

![Figure 6. Error Active Statechart](image)
The first function performed after entering Send is the data transmission, see figure 7. That means, the data frame is assigned to the bus port. After that, the arbitration will be checked to find out whether the node gains the bus or not. In case the arbitration is lost the Go Receive event will be generated. If the node gains the bus, i.e. if the Arbitration OK event is generated, the time slot awaiting the acknowledgement will be computed. Then the Wait On ACK state will be entered. This state will be left if the acknowledgement arrives or a Timeout event occurs. The latter causes a Send Error event, which will be dealt with after leaving Send.

![Send Statechart](image)

Figure 7. Send Statechart.

Further charts are not detailed here because they are decomposed in the same manner.

In figure 8, the time sequence diagram shows the interrelations between sender and receiver for the case of an acknowledgement.

Having transmitted a telegram, the sender awaits either the acknowledgement or the timeout of the Receive ACK Timer. This timer is set to the expiry of the ACK time slot computed according to the number of bits transmitted until here and the bit time. In the model, the frames are transferred without delay. So the receiver computes the time point for acknowledgement with the same rule, but with the number of bits reduced by one. Without having achieved an acknowledgement, the sender transmits an error flag if the Receive ACK Timer expires. Both sender and receiver adjust another timer - the EOF timer - to the remaining time interval of the transmission phase. If these timers expire a new transmission can be started.

Regarding all timers in the instantiated nodes, the simulator yields the timing of the data transfer on the bus.
3.3 VHDL code generation

The translation of Statecharts into VHDL code can be accomplished by the ExpressV-HDL tool [7] or manually. In order to get an optimal code for fast simulation, the manual translation was preferred. Furthermore, functions for initialization, producing log-files, computing statistics, or any data processing can be added quickly if the VHDL code is well known.

The top level statecharts of the components are translated into design entities with corresponding architectural bodies. All reference charts are mapped into procedures called in the architectural body. As an example, the top level statechart of the MAC is considered.

The MAC entity declaration describes the interface to the Bus and to the Logical Link Control. Here, Bus is declared as a bidirectional signal of the type Telegram. The Telegram type declaration and all the other ones belong to the CAN package developed.

Besides signal declarations, the MAC architectural body contains a single Process statement including a Case statement and a Wait On statement. The Case statement specifies the state transitions according to the Statechart representation in figure 5. Events in the Statechart are translated into signals with the attribute EVENT. The Wait On statement suspends the process until an event occurs related to its sensitivity list.

Translation of the MAC Statechart:

```vhdl
ENTITY CAN_MAC IS
    PORT ( Bus: INOUT Telegram; -- Bus interface
            LLC_Request: IN LLC_Frame; -- LLC interface
            LLC_Indication: OUT LLC_Frame;
            LLC_Confirm: OUT LLC_Conf )
END CAN_MAC;
```
ARCHITECTURE behavioral OF CAN_MAC IS

SIGNAL MacState: MacStateType := Start;
SIGNAL GoPassive, GoActive, GoBusOff, Restart: bit;
SIGNAL Toggle: bit;

BEGIN

CAN_MAC: PROCESS

CASE MacState IS

WHEN Start =>

WHEN ErrorActive =>

IF GoPassive EVENT THEN

MacState <= ErrorPassiv;
ELSE

execute_error_active;        -- procedure call

END IF;

WHEN ErrorPassive =>

WHEN BusOff =>

END CASE;

WAIT ON

MacState, LLC_Request, Bus,
GoPassive, GoActive, GoBusOff, Restart,
Toggle;

END PROCESS;

END behavioral;

All procedures detailing states have the same structure: a case statement specifies state transitions in dependency on EVENTS attributed to signals and/or logical relations. Any state transition will be indicated by a Toggle event, which will affect the progress of the process via the Wait On statement on the top level. In case a state has to be decomposed once more a further procedure will be called with the structure just described.

4 Application processors, sensors, and actuators

The tasks usually placed in the CAN application layer are the following:

- polling of sensors
- transmission of polled values
- transmission of events signalled by limit indicators
- data preprocessing
- passing of set-point values to actuators
- control settings.

In the model, these tasks will be performed by the application processors. Again, the VHDL description is formulated on behavioural level. For instance, a temperature control will be considered with a Poll Processor and a Heating Controller.

The Poll Processor only inserts the given temperature value in a request frame and directs this frame to its corresponding Protocol Unit. This is repeated isochronously according to a Wait
For statement adjusted to the required sample period. The signal identifier and the sample period are read from a network configuration file during the initialization step.

ENTITY POLL_PROCESSOR IS
  GENERIC ( NodeNo: natural;          -- node number
            ConfigFile: string );     -- configuration file
  PORT ( Temperature: IN real; -- input from sensor
         Indication: IN LLC_Frame; -- input from protocol unit
         Confirm: IN LLC_Conf; -- input from protocol unit
         Request: OUT LLC_Frame ) -- output to protocol unit
END POLL_PROCESSOR;

ARCHITECTURE behavioral OF POLL_PROCESSOR IS
BEGIN
  PROCESS
    VARIABLE Init: bit; SignalID: integer; Period: time;
    BEGIN
      IF NOT Init THEN
        read_config_file ( NodeNo, ConfigFile, SignalID, Period );
        Init := í1í;
      END IF;
      WAIT FOR Period;
      Request <= real_to_frame ( SignalID, Temperature );
    END PROCESS;
  END;
END;

The Heating Controller receives the temperature value as part of an indication frame from the Protocol Unit coupled. In dependency on the range settings, the control signal Heating is assigned to '1' or '0'. This operation will be repeated if a new indication occurs.

ENTITY HEATING_CONTROLLER IS
  GENERIC ( NodeNo: natural; -- node number
            ConfigFile: string ); -- configuration file
  PORT ( Heating: OUT bit; -- output to actuator
         Indication: IN LLC_Frame; -- input from protocol unit
         Confirm: IN LLC_Conf; -- input from protocol unit
         Request: OUT LLC_Frame ) -- output to protocol unit
END HEATING_CONTROLLER;

ARCHITECTURE behavioral OF HEATING_CONTROLLER IS
BEGIN
  PROCESS
    VARIABLE Init: bit; SignalID: integer;
    VARIABLE Temperature, LowTemp, HighTemp: real;
    BEGIN
      IF NOT Init THEN
        read_config_file ( NodeNo, ConfigFile, SignalID, LowTemp, HighTemp );
        Init := í1í;
      END IF;
      WAIT ON Indication;
      Temperature := frame_to_real ( SignalID, Indication );
      IF Temperature < LowTemp THEN
        Heating <= í1í;
      ELSIF Temperature > HighTemp THEN
        Heating <= í0í;
      END IF;
    END PROCESS;
END;
Sensors and actuators are also modelled in this way. If a process model is available signals will be sent to or received from this model. The signals may be typified in any kind or composition. With the sampling method, for instance, a discrete-time model of the process can be created [8]. In the near future, analog extensions of VHDL (VHDL-A) will allow to incorporate continuous process models. However, if a process model is not available signal sequences have to be obtained by reading files containing test patterns or utilizing approximation functions. If only a traffic load has to be produced sensors and application processors will be replaced by event generators. These generators issue telegrams in isochronously, uniformly, or exponentially distributed periods, the latter ones are quantified.

5 Simulation

A design and simulation environment using the Synopsys VSS simulator is shown in figure 9. The model library of this simulator has been extended by the Protocol Unit, which references the Bus Resolution Function. Further extensions are application processors in the manner described above, but enhanced by adjustable signal generators or receivers.

![Diagram](image)

**Figure 9.** Design and simulation environment for field bus systems with the Synopsys VSS simulator

With the schematic editor the model components are composed to a network as represented in figure 10. Generic parameters such as transmission rate, signal identifier, or sample period will be set at the simulation run time via control files, which are only read in the initialization step. To define the control files, a Configurator is offered to the user. Monitoring facilities of the
Simulator are applied to depict the traffic or to report lost data. The Wave Monitor shows the messages on the transmission medium in their timing. Details referring to occupied buffers, latent periods, response times, or lost data are directed to the Pad Window. The system is extended by a Bus Observer which writes selected data or events into a log-file. Similar reports can be produced by the Protocol Unit, too. The analysis of these data items is done in a post-processing step with various programs.

Figure 10. Composition of a controller network

Simulation experiments have been carried out with different systems using CAN-Bus, PROFI-BUS, or INTERBUS-S. For the latter ones, equivalent simulation environments were installed.

With the CAN simulation, the medium access times of a sensor/actuator system consisting of 12 participants were determined. Figure 11 and figure 12 show the simulation results. Here, the medium access time is the time interval between generating a message and its complete transmission.

Figure 11. Mean medium access times of the CAN example
In the CAN example, the sensors transmit broadcasts in isochronously or exponentially distributed periods. The receivers and actuators, respectively, select the messages according to the identifiers given to them. Among the participants, a controller periodically transmits a remote frame to all other nodes to request data.

Figure 11 shows the development of the mean medium access times during 1s simulated real time with the bit rate of 500 KBit/s. The controller curve represents a set of signals since the controller sends broadcasts with various priorities. For this simulation 6.5 min CPU time elapsed on a Sun Sparc 10 station. Figure 12 shows the relations between the mean value and the current values of the medium access time of single node. The peaks illustrate the need of detailed investigations in time critical applications.

References