01 Jan 2006

Model Checking Control Communication of a FACTS Device

Bruce M. McMillin
Missouri University of Science and Technology, ff@mst.edu

J. K. Townsend

David Cape

Recommended Citation

B. M. McMillin et al., "Model Checking Control Communication of a FACTS Device," *Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2006.
The definitive version is available at https://doi.org/10.1109/ICPPW.2006.54

Model Checking Control Communication of a FACTS Device

David A. Cape
dacvdc@umr.edu

Bruce M. McMillin *
University of Missouri-Rolla
Department of Computer Science
Intelligent Systems Center
Rolla, MO 65409-0350
ff@umr.edu

James K. Townsend
jtown@umr.edu

Abstract

This paper concerns the design and verification of a real-time communication protocol for sensor data collection and processing between an embedded computer and a DSP. In such systems, a certain amount of data loss without recovery may be tolerated. The key issue is to define and verify the correctness in the presence of these lost data frames under real-time constraints. This paper describes a temporal verification that if the end processes do not detect that too many frames are lost, defined by comparison of error counters against given threshold values, then there will be a bounded delay between transmission of data frames and reception of control frames. This verification and others presented herein were performed with the model checkers SPIN and RT-SPIN.

Keywords:*model-checking, verification, real-time, lossy, control, communication, protocol, FACTS*

1. Introduction

The context for the development of this communication protocol is an attempt to understand how to control electric power flow in a power transmission system (network) by means of Flexible A/C Transmission System (FACTS) devices [1]. A FACTS device has a Digital Signal Processor (DSP) board that is responsible for sensory data acquisition, transforming the data into a usable format, communicating the data to an Embedded PC (EPC), receiving control information from a dynamic controller in the EPC, and applying the control information to power electronics that modify power flow on a particular power line [9]. The DSP provides continuous sensor readings which are expected to vary slowly, so some message loss is tolerable. The communication between the DSP and the EPC is via a Controller Area Network (CAN) bus. This paper focuses on the design and verification of a lossy communication protocol between the DSP and the EPC over the CAN bus.

The problem of reliable data transfer has been solved by several protocols: the alternating-bit protocol, the go-back N protocol, the selective repeat protocol, and hybrids of these including the method used by TCP [7]. In verification work of the correctness properties of these protocols, Bull and de Villiers [3] have done some work on the alternating-bit protocol and TCP. Holzmann [5] describes a verification of some standard safety properties and two liveness properties for the go-back N protocol. The first liveness property is that sent messages are eventually received, and the second liveness property is that messages are delivered in the order sent.

For this paper, there are two reasons why the protocols just discussed are inappropriate. First, there is no need for 100% reliability of the communication, because the control loop need not furnish new settings every cycle; it can use settings from the previous cycle without much harm being done to the integrity of the system. Second, and more important, is that the CAN bus bandwidth is inadequate to support message retransmission. Therefore, a protocol which allows some message (data) loss but which guards against excessive losses is called for. Perfect functioning is defined as operating without message loss, normal functioning as operating with not too much message loss (quantified later in this paper), and malfunctioning as operating with too much message loss.

A formal correctness argument for a protocol, particularly one embedded as deeply as the proposed EPC-DSP protocol in a FACTS device, is vital for correct operation of the system. Simply using the features of the CAN protocol, which has an inherent reliability, still does not provide confidence in data and control exchange. Verification provides this increased level of confidence. However, there appears

*This research is supported in part by the NSF MRI grant CNS-0420869 and in part by the UMR Intelligent Systems Center.

Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06)
0-7695-2637-3/06 $20.00 © 2006 IEEE

to be little prior published work on formal verification of a protocol that meets real-time constraints in a sensor network communication by allowing a tolerable level of data loss. Protocol design and verification of its correctness are the tasks undertaken in this paper.

The organization of this paper is as follows. In section 2, the protocol is described. In section 3, the correctness specification and the actual method of verification is explained. In section 4, we mention a validation of the fact that Assertion 1 expresses the desired property and show what happens when some of the error counters are disabled. Finally, the discussion is concluded in section 5.

2. Protocol Description

The DSP and the EPC are two independent computers with independent clocks, and, thus, have no inherent synchronization. Each end process communicates asynchronously over the CAN bus that connects them. In the protocol operation (see Figure 1), the DSP periodically receives a control setting (ctrl) from the EPC over the CAN bus and applies this to embedded power electronics (not shown). The DSP also returns data frame sets (3 data) regarding the state of the power electronics to the EPC. The EPC periodically receives these data frame sets, computes new control information obtained from a dynamic controller (not shown), and sends this control setting to the DSP.

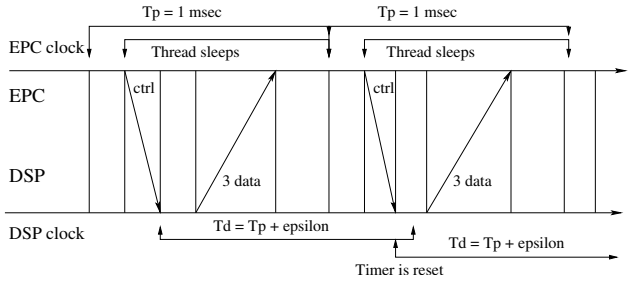

Figure 1. EPC-DSP Protocol Operation.

The basic protocol operation is complicated by a number of factors. The DSP and the EPC operate on two independent, but interleaved, cycles (of length T_d and T_p respectively). If information (control or data) is lost or delayed, each side must have the ability to continue operation and not block. If information is missing, however, both the DSP and EPC can continue asynchronously, using data/control from a previous cycle. There is a tolerable bound on the amount of data/control loss and a specific upper bound on the gap between control and data frames that the protocol can tolerate. The protocol design must deal with all these issues.

For the message-passing, each end process has variables to store the received values (Figures 2 and 3). On the

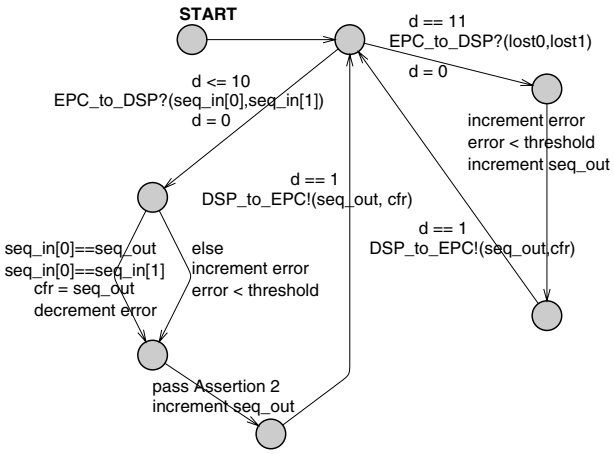

Figure 2. Timed Automaton: DSP Board with receive errors modeled.

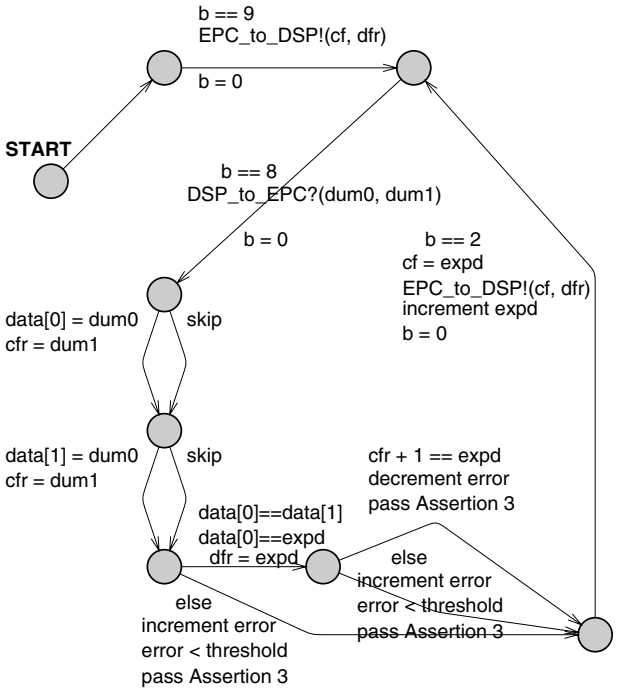

Figure 3. Timed Automaton: EPC with receive errors modeled.

EPC side, there is an array `data[]` for storing data frame sequence numbers, and a variable `cfr` for storing the sequence number of the last control frame that the DSP received. For verification purposes, the model used only two data frames (instead of three) in order to reduce the complexity without affecting the essential features of the design. There is also a variable `expd` (expected), which is

```
DSP (){
 byte error = 0; byte threshold = 5;
 byte lost; byte cfr = 1;
 *[atomic{when{d<=10} reset{d}
   EPC_to_DSP?msg(seq_in[0],seq_in[1])->
     [((seq_in[1]==seq_out)
        &&(seq_in[0]==seq_in[1]))->
          cfr = seq_out;
          [(error > 0) ->
               error = error - 1
           []else -> skip]
      []else ->
          error = error + 1;
          [(error>=threshold)->QUIT
           []else -> skip]];
     /* Assertion 2 */
     seq_out=(seq_out+1)%wrap}
progress: atomic{when{d==1}
          DSP_to_EPC!msg (seq_out,cfr)}
  []atomic{when{d==11} reset{d}
     EPC_to_DSP?msg(lost, lost) ->
     error = error + 1;
     [(error>=threshold)->QUIT
      []else -> skip];
     seq_out=(seq_out+1)%wrap}
     atomic{when{d==1} skip ->
       DSP_to_EPC!msg (seq_out, cfr)}]}
```

Figure 4. Pseudocode: DSP Board.

incremented in each cycle. The EPC will send a control frame with two sequence numbers: the first is simply equal to `expd`, and the other is the sequence number of the most recently received data frame set (which enables the DSP to know how fresh the control information is). On the DSP side, there is an array `seq_in[]` for the control frame sequence number received and the sequence number of the last fresh control frame sent, and a variable `seq_out` for the current data frame sequence number of a set. The defined constant `wrap` was set at 128, and instead of incrementing sequence numbers by one each cycle, they were incremented by eight units to reduce the state space.

During normal operation, messages will be received sequentially in order without gaps, and error counters will be decremented by one unit per cycle (with zero as a minimum value). To address the issues of lost or delayed data/control, the gap between data/control, and to ensure the protocol does not block, error counters and timers are maintained locally in each end process. The clock variables b and d regulate the timing. They simulate local clocks for the two end processes which run at *exactly* the same rate, but which can be reset independently.

```
EPC (){
 byte error = 0; byte threshold = 5;
 byte dum0, dum1; byte cf = start;
 byte dfr = start; byte cfr = 0;
 atomic{expd=expd+1;
 when{b==9} reset{b}
   EPC_to_DSP!msg (cf, dfr)}
 *[atomic{when{b==8} reset{b}
     DSP_to_EPC?msg (dum0, dum1)->
    [(1) -> data[0] = dum0; cfr = dum1
     [](1) -> skip];
    [(1) -> data[1] = dum0; cfr = dum1
     [](1) -> skip];
    [((data[0]==data[1])
         &&(data[0]==expd))->
         dfr = expd;
         [((cfr+1)%wrap!=expd)->
             error = error + 1;
             [(error>=threshold)->QUIT
              []else]
          []else ->
             [(error > 0) ->
                 error = error - 1
              []else]]
     []else ->
         error = error + 1;
         [(error>=threshold)->QUIT
          []else -> skip]]
    cf = expd;
    /* Assertion 3 */
    expd=(expd + 1)%wrap}
    atomic{when{b==2} reset{b}
     skip->EPC_to_DSP!msg (cf, dfr)}]}
```

Figure 5. Pseudocode: Embedded PC.

The error counters play a major role in the correctness and are designed to limit the amount of timeouts that can occur. Since timeouts occur due to lost, missing, or delayed data/control, the error counters form a record of the number of accumulated errors. An error counter will be incremented by the DSP when control frames are delayed or lost causing a timeout at $T_d = T_p + \epsilon$ milliseconds (ϵ may be taken to be 0.05 milliseconds) or when the current control frame received is not fresh. The EPC will increment its error counter when message loss or delay causes a situation in which a complete data frame set with expected sequence number is not available in a cycle or when it receives an indication that the DSP did not receive a fresh control frame its previous cycle. In each cycle of the EPC, a control frame is sent with sequence number echoing the expected value back via the CAN bus to the DSP. If either error counter reaches a

given threshold value, the protocol QUITs in an error state.

The resetting of timeouts based on successful receipt of messages has the effect of synchronizing the interleaved cycles of the EPC and DSP.

3. Correctness and Verification of Models

3.1. Correctness

We define a *fresh* control frame as one which has just been computed based on a complete data frame set, so both sequence numbers which it contains are equal.

Correctness of this protocol is defined by ensuring that a fresh control frame follows a complete set of data frames, that there is (on the DSP side) a bounded gap between fresh control frames received and data frames sent, that there is also (on the EPC side) a similar bounded gap between acknowledged fresh control frames and control frames sent, and that the protocol does not block. Four formal assertions, taken together, comprise this definition of correctness, which is shown by verifying that if the error counters stay below a threshold, the protocol satisfies the four assertions.

Assertion 1 states, formally, that a fresh control frame is only generated from a complete set of data from that cycle.

Assertion 1

$$\begin{aligned} p &= data[0] == data[1] \\ &\quad \wedge (data[0]+1)\%wrap == expd, \\ t &= seq\_in[1]! = data[0]) \\ &\Rightarrow \ \Box(t \rightarrow ((tUp)||(\Box t))) \end{aligned}$$

Figure 6. The control frame based on sequence number data[0] is not received by the DSP before the embedded PC receives a complete data frame set with that sequence number.

Assertions 2 and 3 are composed of three cases. Consider Assertion 2. If `seq_in[1]` is not about to wrap-around, then the data frame sequence number is larger than `seq_in[1]`, but not more than `max_gap` larger. Similarly, if the data frame sequence number has not just wrapped around, then `seq_in[1]` is small, but not more than `max_gap` smaller than `seq_out`. The third case is the exception and asserts that the data frame sequence number is not too far past zero relative to the difference between the `seq_in[1]` and `wrap`, which is the wrap-around modulus (1 + the maximum sequence number). Assertion 3 is similar.

Assertion 2

```
if
::(seq_in[1]<(wrap-max_gap)) ->
    assert(seq_in[1]+max_gap>=seq_out);
    assert(seq_out>=seq_in[1])
::(seq_out>=max_gap + 1) ->
    assert(seq_out-max_gap<=seq_in[1]);
    assert(seq_in[1]<=seq_out)
::else ->
    assert(seq_in[1] >=
      seq_out+(wrap-max_gap))
fi;
```

Figure 7. The difference between the data frame sequence number and most recent fresh control frame sequence number is bounded by max_gap.

Assertion 3

```
if
::((cfr+1)%wrap<(wrap-max_gap)) ->
   assert((cfr+1)%wrap+max_gap>=expd);
   assert(expd>=(cfr+1)%wrap)
::(expd>=max_gap+1) ->
   assert(expd-max_gap<=(cfr+1)%wrap);
   assert((cfr+1)%wrap<=expd)
::else ->
   assert((cfr+1)%wrap >=
     seq_out+(wrap-max_gap))
fi;
```

Figure 8. The DSP acknowledges at least intermittent receipt of control frames. The EPC will be assured that the DSP is not experiencing too many cycles without a fresh control frame.

Assertion 4 *No non-progress cycles: if the protocol runs without terminating due to an error condition, then the DSP will successfully receive control frames infinitely often.*

Assertion 4 is necessary to show that the protocol does not block, and more importantly, that Assertion 2 is checked infinitely often. The error occurrences relate to time because each end process operates (at least approximately) on a periodic cycle. Thus, no long period of time can pass without an exchange between the EPC and the DSP. This synchronization relates the data to the control frames.

3.2. Verification of Models

Both SPIN and RT-SPIN were used to perform verifications of the assertions which are described above. Promela [6] and RT-Promela [10] are used as the modeling languages. SPIN is well-known and easier to use than RT-SPIN, but lacks inherent timing features; without some timing assumptions or control of interleaving, the model can exhibit many interactions that are not representative of the protocol's timed operation.

To build timing into the Promela model, [4] presents an attractive approach to adding timers to the model to regulate the timing of the end process cycles. This leads to a rough scheduler which gives the DSP and EPC opportunities to execute in an alternating manner (creating some implicit synchronization), but with the additional possibility of one of them drifting ahead and executing more frequently (see Figure 9). the DSP_CLK message initiates the DSP cycle, and the EPC_CLK message initiates the EPC cycle.

```
active proctype CLK ()
{
    byte P = 128;
end0:  do
    ::atomic{DSP_CLK!msg(0)->
        P = P - 1;
        if
        :: (P<=127) -> break
        :: else
        fi}
    :: atomic{EPC_CLK!msg(0)->
        P = P + 1;
        if
        :: (P>=130) -> break
        :: else
        fi}
    od
}
```

Figure 9. Scheduler for SPIN Verification.

For the SPIN verification, an untimed, multiple data frame model using channels (one channel for each data frame) was created in Promela and was verified against Assertions 1, 2, and 4 using SPIN. By varying the number of channels that could potentially lose messages, the complexity grew dramatically. To reduce the complexity of verification, thresholds for the error counters and the amount of drift between the cycles of the EPC and the DSP were limited to small values.

As a validation of the difficulty and complexity of verification of a communication protocol, a comparison to the work of [2], which focuses on model checking of a zero-cost secured protocol (ZCSP) using SPIN was made. That work also mentions difficulties with the capability of SPIN model checking software to handle complicated models. The number of states visited was comparable for the ZCSP verification (on the order of 10^8 states) and the SPIN verification (on the order of 10^6 states).

Next, a real-time model was developed based on the untimed model. RT-SPIN allows better modeling of the real-time aspects of the system. It was a simple task to modify the untimed models for use with RT-SPIN, by substituting a timing constraint for the signal from the scheduler (Figure 9).

In the RT-Promela model (Figures 4 and 5), timing constraints that were introduced force the DSP and the EPC to operate on cycles with periods of either 10 or 11 units, where units represent tenths of milliseconds. For both the untimed and timed models, the complexity of the verifications of Assertion 1 is similar to the complexity of the verifications of Assertion 4 (progress). The untimed and timed model verifications were found to have similar complexity.

With RT-SPIN, it was eventually possible to explore the entire state space for infinite runs despite complexity considerations, so progress verifications were possible. This was accomplished by replacing the explicit CAN bus process with buffered channels. The issue of deadlock was also considered for the final verifications, and neither SPIN nor RT-SPIN discovered any invalid end-states in the safety verifications.

The RT-SPIN verification uses Assertion 2 to check that (during normal operation) control frames received by the DSP have a sequence number (`seq_in[1]`, which may be interpreted as an acknowledgement number) within a certain range relative to the most recent sequence number of data frame sets sent by the DSP. Taken together, if the communication continues forever, it will do so correctly.

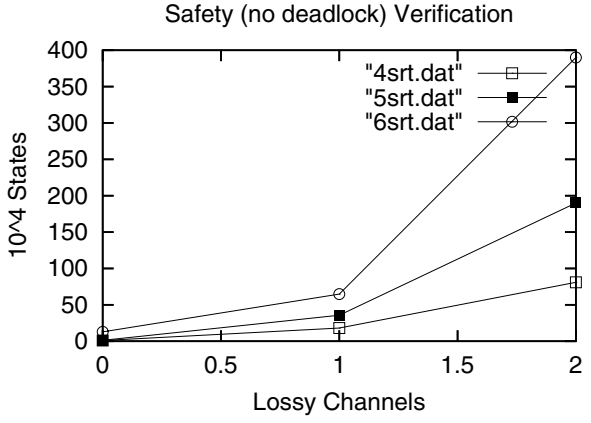

Figure 10. Complexity of Safety Verification for RT-SPIN.

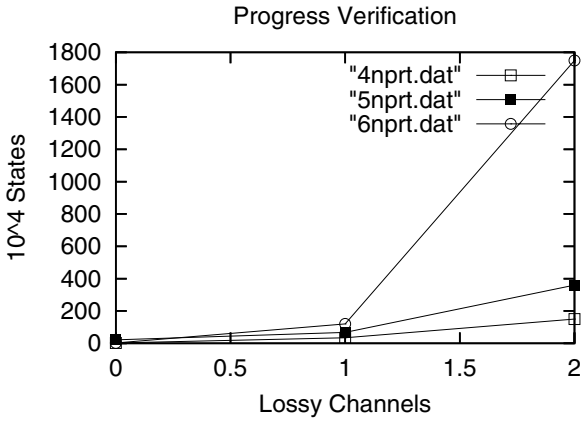

Figure 11. Complexity of Progress Verification for RT-SPIN.

The complexity of the real-time model verification is shown in Figures 10 and 11. A parameter set with a threshold value of 4 was used with `max_gap` equal to 6 for 4srt.dat and 4nprt.dat. The verifications for 5srt.dat and 5nprt.dat used a threshold of 5 with `max_gap` equal to 7, and for 6srt.dat and 6nprt.dat a threshold of 6 was used with `max_gap` equal to 7 as well. These parameter sets were chosen to demonstrate the increased complexity of the verifications when the threshold is increased; it is also apparent that the complexity increases as more message channels are allowed to become lossy.

4. Validation

Because Assertion 1 expresses a non-trivial temporal property, it was validated against a simpler model program with simpler predicates (p and t), but with the same temporal formula.

The Table 1 shows how the error counters in the DSP and EPC are independently capable of detecting the error state and how without at least one of them unacceptable errors can occur.

Error Counter(s) Disabled	Result
EPC	Assertions 2 and 3 passed
DSP	Assertions 2 and 3 passed
Both DSP, EPC	Assertions 2 and 3 violated

Table 1. Result of disabling error counters.

5. Conclusion

The software used in the modeling and model checking in this paper provides a method for design and some simple verifications. RT-Promela and RT-SPIN are enhancements of the popular model checker SPIN, which is an advantage. The GUI XSPIN was very useful during the simulation phase of the project. As some expertise was developed, simulation became less important, and verification was done relying on XSPIN only for LTL formula conversion. One major advantage that any real-time model checker could offer is the possibility of partial-order reduction, the theoretical foundation of which has been laid by Minea [8]. This could significantly increase the capability of any model checking software.

Despite some limitations of SPIN and RT-SPIN, it has been verified that with a reasonable definition of normal operation (possibly with some message loss), which can be monitored by the end processes, an acceptable level of information integrity can be guaranteed in the CAN bus communication protocol (EPC-DSP) which may be used in a FACTS device. This protocol and verification can be used for continuous sensor readings and control to tolerate some degree of message loss.

References

[1] A. Armbruster, B. McMillin, and M. Crow. Controlling power using FACTS devices and the maximum flow algorithm. In *Proc. 5th International Conference on Power Systems Operation and Planning*, pages 158–163, 2002.

[2] V. Beaudenon, E. Encranaz, and J.-L. Desbarbieux. Design validation of ZCSP with SPIN. *IEEE*, pages 102–110, 2003.

[3] D. Bull and P. D. Villiers. Using SPIN to verify protocols at the implementation level. In *ACM International Conference Proceeding Series; Vol. 30 Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 195–204, 2002.

[4] Z. Gu and K. Shin. Model-checking of component-based event-driven real-time embedded software. *Eighth IEEE Internation Symposium on Object-Oriented Real-Time Distributed Computing*, pages 410–417, 2005.

[5] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[6] G. Holzmann. *The SPIN Model Checker Primer and Reference Manual*. Addison-Wesley, 2004.

[7] J. Kurose and K. Ross. *Computer Networking: A Top-down Approach Featuring the Internet*. Pearson Education, Inc., 2005.

[8] M. Minea. *Partial Order Reduction for Verification of Timed Systems*. PhD thesis, Carnegie Mellon University, 1999.

[9] M. Ryan, S. Markose, Y. Cheng, F. Liu, and B. McMillin. Structured object-oriented co-analysis/co-design of hardware/software for the FACTS power system. In *29th Annual International Computer Software and Applications Conference*, volume 1, pages 396–402, 2005.

[10] S. Tripakis and C. Courcoubetis. Extending promela and spin for real time. In *Second International Workshop, Tools and Algorithms for the Construction and Analysis of Systems*, pages 329 – 348, March 1996.