Project

Modeling of BlueTooth
Link Manager Protocol (LMP)

(PART C of the Bluetooth Specification)

Table of Contents

1 Modeling of BlueTooth's Link Manager Protocol (LMP)

1.1 Project participants:

David Cypher and Yunming Song

1.2 Goal:

To create a model of the Bluetooth Layer Management Protocol (LMP) using PROMELA, and then verify and validate the model using SPIN.

1.3 Background:

This project started in December 1998, with a review of the Bluetooth specification version 0.6. Comments were generated, but never submitted as a newer version of the specification was available by this time. Version 0.7 was then reviewed and comments generated and submitted to the Bluetooth chatroom for LMP (PART C) on the Bluetooth website. Additional comments were submitted shortly thereafter. No responses were ever received on these comments.

At the end of January 1999 a new version of the specification (0.8) was released. Comments were created by updating previous comments and adding new ones. These were submitted to the new Forum, the replacement of the chatroom, on the Bluetooth Web site.

May 11, 1999 version 0.9 was released. Comments for version 0.9 were generated on May 17th from the previous version 0.8 comments. While doing so it was found that all editorial comments on version 0.8 were either accepted into version 0.9 or rendered not applicable by other modifications. As for the technical comments only a few were accepted. Most are still unanswered. As for the assumptions #1, #6, and #7 were addressed. #6 and #7 have final resolution, while #1 only addressed the solutions, not the issue.

Modeling was started in February using version 0.8 of the LMP. There are twenty separate subsections (i.e. procedures) contained in LMP (Part C) to be modeled.

During the months of December and January time was spent learning the Promela language and SPIN. This learning is continuing as we create the models and run the simulations

1.4 Current Status (as of 5/17/99):

There are now twenty three (23) procedures from LMP (Part C). Three new procedures were added to version 0.9 and most of the previous twenty (20) procedures requires modifications.

1.5 Current Conclusions (as of 3/31/99):

The base specification changed significantly from 0.6 to 0.7, 0.7 to 0.8, and 0.8 to 0.9. Modeling is difficult because our assumptions have not always matched the changes from version to version. Too many assumptions create too many non-interoperable models. If the review comments would be addressed, modeling assumptions could be reduced and perhaps a single model could be produced.

2 Modeling experiences:

This section describes our experiences during the modeling of the LMP based on the use of Promela and SPIN and the BlueTooth specification.

2.1 Promela/SPIN Notes

SPIN running on a Windows 95 or NT 4.0 PC is not stable.
The output of SPIN cannot be assumed to be correct on first execution. Multiple tries, changing platforms, and rebooting often produce different results.
No easy way (non-error generating) to model termination of a process, which is needed for the BlueTooth Detach procedure.

2.2 BlueTooth not fully specified

Being that there were some key questions unanswered, we had to make many assumptions that produced many models. Only those models that were our final understanding are contained. However, we did testing on two different views of the channels that a Bluetooth device might use to communicate with. These two cases were where the channels were modeled as being synchronous (i.e. no buffers in channel) or asynchronous (i.e. buffering allowed in channels).

There were twenty (23) individual functions with the associate procedures to be modeled and verified. Figure 1 shows by graphical representation of the SPIN output (the number of states stored and matched and number of transitions reached) for the twenty individual models. Some functions have two models: one for synchronous and one for asynchronous. Where two models exist, only the asynchronous output is shown because it provides the upper bound on number of states stored and matched and number of transitions. Both Promela models and SPIN outputs are listed in Annex B.

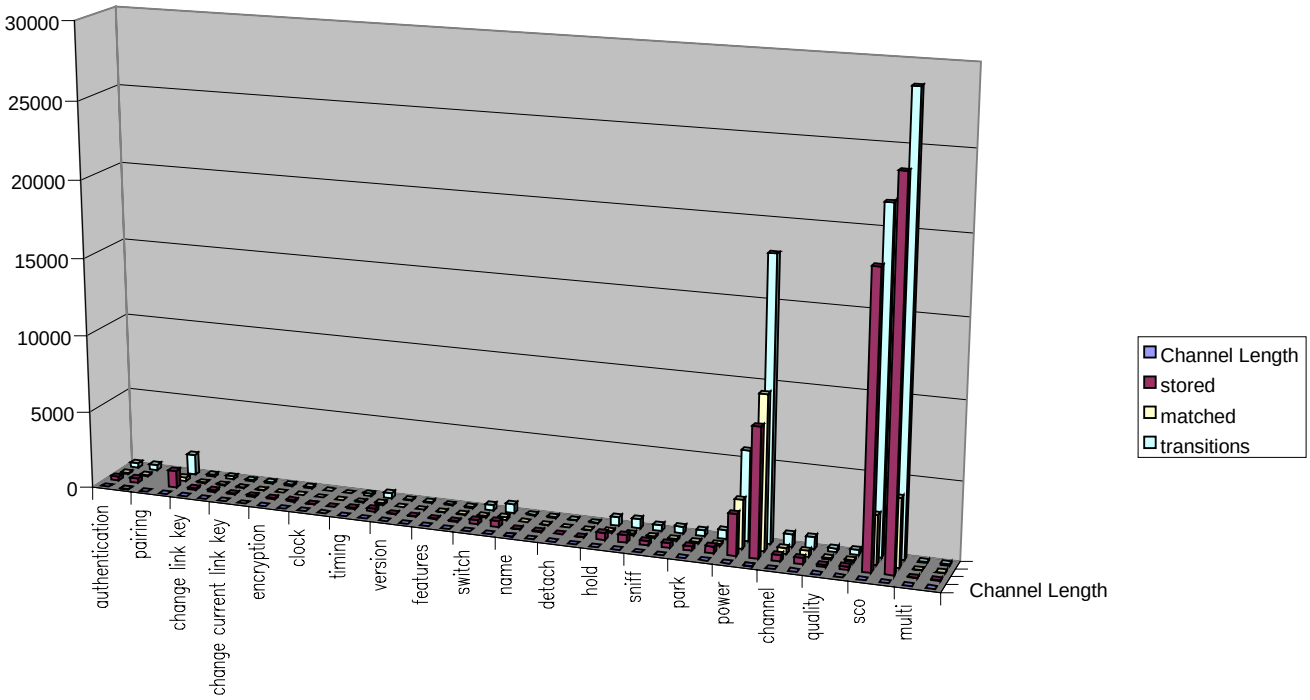


Figure 1 Twenty (20) single function models and their SPIN output in number of states stored and matched and the number of transitions and size of channel.

When using the asynchronous channel type, states must be defined within the models. These states are not part of the BlueTooth LMP specification. States were not needed in the synchronous channel type models, since only one transaction can exist at a time.

Even though on an individual basis the models created were easily manageable, when combining them into one complete system, the number of states and transitions were unmanageable. We grouped the individual functions into related functions, thus making the models more manageable. A single model with all 23 functions is not possible due to the state space explosion.

One such example of grouping related functions into a small model is the first five functions. The functions described in sections 3.2 through 3.6 are grouped together since they are all security related. This division is also supported by the text from section 4 of the Bluetooth specification. This model assumes a synchronous channel (i.e. no buffers). An asynchronous model was created for the first 4 functions, but when the fifth was added an error occurred with conflicting states, which is caused by the fact that multiple message exchanges could be outstanding, which leads to indistinguishable replies.

Figure 2 shows the state space explosion as one function (sections 3.2 through 3.5) after another is added to a single model.

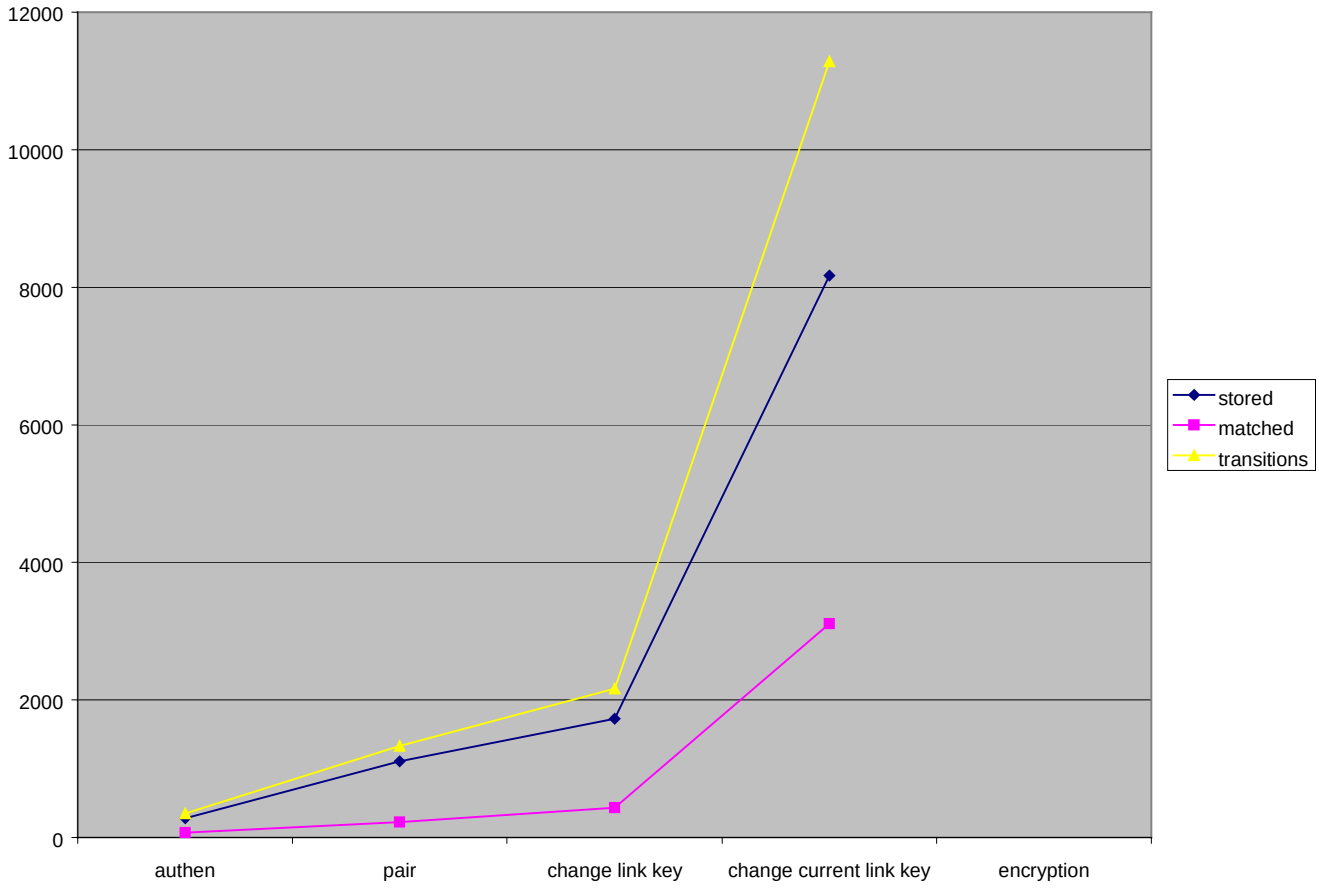


Figure 2 Growth of state space as one function after another is add to create a single model. (Authentication, Pairing, Change Link Key, and Change Current Link Key)

Figure 3 shows the state space explosion as one function (sections 3.6 through 3.11) after another is added to a single model.

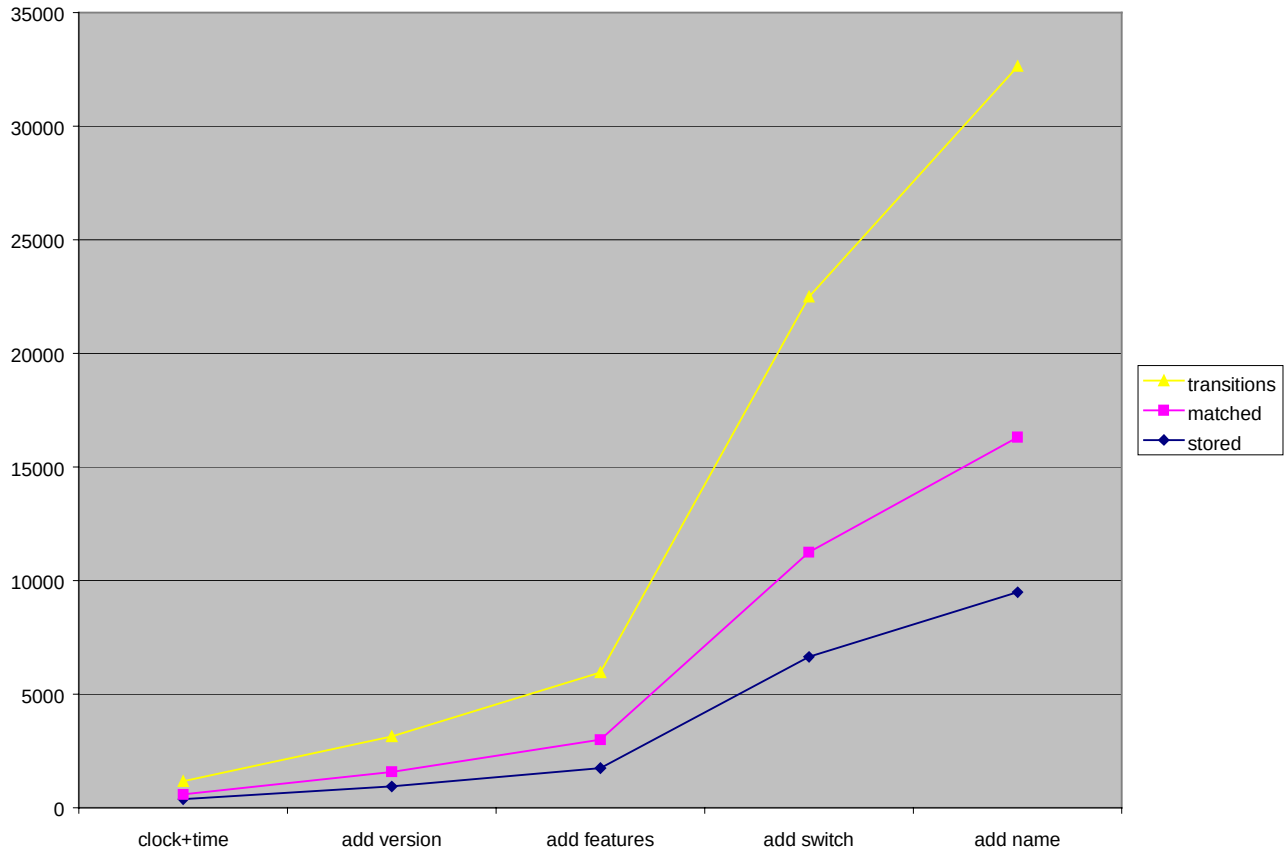


Figure 3 Growth of state space as one function after another is add to create a single model. (Clock Offset request, Timing Accuracy Information Request, LMP Version, Supported Features, Switch of Master/Slave Role, Name Request)

2.3 Conclusion

This tool (Promela/SPIN) provides many opportunities for trying the various assumptions and attempted fixes for protocol development, but the LMP protocol is too immature to perform an exhaustive test of all the possibilities based on our assumptions.

3 Future plans for modeling other PARTs/Protocols of BlueTooth

Future plans for modeling depend upon our capability to have access to the developing specification. At this time we have access to version 0.9. The final version is not scheduled for release until the end of June 1999 (i.e. version 1.0). We should at least update our current models to align with the new versions (0.9 and 1.0) of the specification. To test the protocol for completeness, as well as prepare for conformance testing of the protocol.

BlueTooth is designed to be used in a mobile network where any system can communicate with any other system. A system is part of a piconet where at most one device is considered as MASTER while the rest of the devices are considered as SLAVEs. For our current modeling we only modeled two devices (one master and one slave) and one ACL channel. We could expand this to cover multiple devices and multiple ACL channels. We also could add simulated DM/DH packets and SCO channels. Currently we just model their invocation functions as described in the specification.

If we want to expand our modeling, we could begin to model the wireless (channels) interface (PART B). However, timing could present a problem when trying to do detailed work.

As for conformance testing and interoperability testing, it is too premature to attempt because if we cannot model the behavior, then how could we possibly test it.

4 Assumptions:

The Bluetooth specification has a number of holes (due to the fact that the specification is under construction) that required us to make assumptions in order to complete our models. These assumptions need to be confirmed, and if correct, need to be stated within the Bluetooth specification before the specification is finalized. What follows is our list of assumptions, their problem and solution(s).

1. **Assumption:** Only one initated message exchange can occur/exist on an ACL (i.e. the communication link between Bluetooth devices).
   Problem: If more than one message exchange is initiated by a device, it is impossible to differentiate a received response of LMP_accepted or LMP_not_accepted. For example, if a device sends an LMP_SCO_link_req and an LMP_switch_req, then receives an LMP_accepted, there is no way to determine which request this LMP_accepted is to be matched.
   Solution(s): Accept the assumption and not permit multiple requests – OR – Use separate messages for each and every exchange (i.e. no sharing of response messages (This could be done in at least two ways: Add a field to the LMP_accepted and LMP_not_accepted PDUs which indicates which type of PDU this is in response to – OR – define new messages). Version 0.9 accepted comment to add a field to LMP_accepted and LMP_not_accepted to differentiate responses. However, the main issue was not addresed.
2. **Assumption:** A device does not have to respond to a received PDU in its next slot.
   Problem: This permits a collision situation in some message exchanges. Eventhough there is a bit indicating which device initiated the message exchange, it is possible for both devices to initiate the same or similar message exchanges. In this case, How is the procedure to be completed? There are two positions to take. One is that you treat each message exchange as separate exchanges (i.e. reason for the added bit in version 0.8). The second is to define new procedures for the detection and resolution of these collisions.
   Solution(s): The first position fails because there is interaction that cannot be separated, therefore it is not a solution. The second position requires defining new procedures, which are numerous. The LMP_hold and LMP_hold_req procedures presents a good example of this problem. (a similar one is LMP_sniff procedures). The final solution is to accept assumption #1 and not allow more than one exchange at a time over the ACL.
3. **Assumption:** (Related to the first two assumptions and effects the next two assumptions) Only one transaction at a time from either side can exist on the ACL. If either side initiates a transaction, then that transaction must complete before any other transaction can be initiated with the exception of the detach procedure.
   Problem: No way to resolve the hold or sniff procedures, if both side can initiate the message exchange (i.e. do not respond immediately to other sides request. Eventhough there exists a bit to differentiate which side began a message exchange, it is impossible to resolve the collision situation without defining many more procedures.

Soultion: (1) Accept the assumption. Side effect is that slots will be wasted, if a device can not return a response in the next time slot and can not use that time slot for sending another PDU. (2) Define many more procedures to allow the interactions.

4. **Assumption:** An LMP_hold message can only be sent when there is nothing outstanding or waiting for a response or being received.
   Problem: Following the above line of assumptions, there is a possibility that a device may want to force a hold on the other device, but what happens to the possible queue of messages when a hold is received?
   Solution(s): Dequeue all messages when an LMP_hold is received or sent. – OR – Ignore all message received while in hold state (this appears to be impossible as the master does not transmit while in hold state, but what prevents a slave from transmitting?).
5. **Assumption:** (Related to the LMP_hold assumption) If a LMP_hold_req is received while awaiting a response to a previously sent message, then what is the action.
   Problem: If the LMP_hold_req is accepted, then what is done about the outstanding response? If the LMP_hold_req is not accepted, then what happens to the outstanding response by the side that transmitted the LMP_hold_req?
   Solution(s): (1) Create a pending hold state for both transmitter and receiver. The pending state is entered after transmitting the LMP_hold_req. While in this state if any PDU other than LMP_hold_req, LMP_accepted, LMP_not_accepted, LMP_hold is recevied, then the messages are responded to normally – or – take this as an implicit negative response (i.e. LMP_not_accepted received). If a LMP_hold_req is received in response to its own LMP_hold_req, then all is well, proceed as normal. If the LMP_hold_req is not in response to its own LMP_hold_req, then (i) assume connection held (i.e. collision) or (ii) treat separately. If LMP_accepted is received, then go to hold state. If LMP_not_accepted is received, then go to a normal state. If LMP_hold is received, then what? (send LMP_hold and go to hold state) The pending state is entered when receiving a LMP_hold_req. If there are any outstanding requests, then no response to the LMP_hold_req until the response is received or reply with LMP_not_accepted and return to a normal state. This solution impacts greatly the current procedures of the specification. (2) if any message is awaiting a response when a LMP_hold_req is received, then send a LMP_not_accepted.
6. ~~**Assumption:** No timeouts because the link protocol guarantees delivery.~~
   ~~**Problem:** The link protocol can acknowledge the delivery of a LMP PDU, but it does not guarantee that a device has acted upon that message. This creates a situation where at the link level the device knows that an LMP PDU was delivered, but does not indicate as to how long to wait before the device can determine that the message needs to be sent again or declared as a failure.~~
   ~~**Solution(s):** This problem would not exist, if a device had to respond in the next slot, but then this violates Assumption #2. Establish a timeout period on how long to wait and then define procedures accordingly.~~ Version 0.9 added a maximum time limit of 30 seconds.
7. ~~**Assumption:** Error handling or unrecognized OPCODE is different from not implemented OPCODEs.~~

~~**Problem:** There are a number of OPCODEs currently defined in the specification. Some used by mandatory procedures. Others are used by optional procedures. How is receipt of an OPCODE that is defined in the specification, but not implemented by the device to be handled? Is it treated as an unrecognized or not supported LMP PDU?~~

~~**Solution(s):** Add text to section 7 of the specification clarifying the subtle difference. "If the Link Manager receives a PDU with an unrecognized OpCode, it responds with LMP_not_accepted with the reason code unknown PDU. If the Link Manager receives a PDU with an OpCode that is not implemented by this device, it responds with LMP_not_accepted with the reason code not supported~~. Version 0.9 makes mandatory recognition of all PDUs and allows the LM to send LMP_not_accepted PDU with error code unsupported feature.

A ANNEX A

Milestones:

December 1998 – January 1999: Learn Promela and SPIN by reading Design and Validation of Computer Protocols by Gerard J. Holzmann and doing the exercises contained within.

February 1- 12, 1999: Review and understand LMP specification

February 8 – March 19, 1999: Model the 20 procedures (approximately one day for each procedure)

- February 8 – 16: Authentication procedure (This takes a long time, since it is the first attempt at coding and verifying the protocol)
- Week 1
  - February 16: Pairing
  - February 17: Change Link Key
  - February 18: Change Current Link Key
  - February 19: Encryption
- Week 2
  - February 22: Clock Offset
  - February 23: Timing Accuracy
  - February 24: LMP Version
  - February 25: Supported Features
  - February 26: Switch of Master Salve Roles
- Week 3
  - March 1: Name Request
  - March 2: Detach
  - March 3: Hold
  - March 4: Sniff
  - March 5: Park
- Week 4
  - March 8: Power Control
  - March 9: Channel Quality
  - March 10: Quality of Service
  - March 11: SCO Links
  - March 12: Control of Multi Slot Packets
- Week 5
  - Integrate all parts into one model

March 22 – 31, 1999: Generate Report

B ANNEX B
Models

This annex lists the Promela models followed by their SPIN output for the LMP. Some models are only for synchronous. Some models are only for asynchronous. Others have both asynchronous and synchronous models.

B.1 AUTHENTICATION

```
#define     LMP_not_accepted               4
#define     LMP_detach                     7
#define LMP_au_rand                   11
#define     LMP_sres                      12

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define s_auth     1
#define s_detach  12

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored) */


proctype device (chan in, out; bit device_type, linkey)
{
   byte state=NORMAL;
   bit outstanding=NO, auth_done=NO;


BEGIN:
   do
/*************************************/
/* 3.1 Authentication                */
/*************************************/
   :: (((outstanding==NO)&&(auth_done==NO))&&(state==NORMAL)) ->
                  out!LMP_au_rand;
                  outstanding=YES;
                  state=s_auth;

   :: in?LMP_au_rand ->
      if
      :: (state==s_detach) -> skip;
      :: !(state==s_detach) ->
         if
         :: (linkey==YES) -> out!LMP_sres;
         :: (linkey==NO) -> out!LMP_not_accepted;
         fi;
      fi;

   :: in?LMP_sres ->
      if
      :: (state==s_auth) ->
```

```
         if
         :: auth_done=YES; outstanding=NO; state=NORMAL;
         :: auth_done=NO; outstanding=NO; state=NORMAL;
         :: out!LMP_detach; outstanding=YES; state=s_detach;
         fi;
      :: (state==s_detach) -> skip;
      fi;

/*************************************/
/* ACEPTED and NOT_ACCEPTED          */
/*************************************/

   :: in?LMP_not_accepted ->
      if
      :: (state==s_auth) -> state=NORMAL; outstanding=NO;
      fi;

   :: in?LMP_detach -> break;

   :: timeout ->
      if
      :: (state==s_detach)&&(len(out)==0) -> break;
      :: !(state==s_detach) ->
          if
          :: (auth_done==YES) -> skip;
          :: (auth_done==NO) -> skip;
          fi;
      fi;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
           run device (AB, BA, MASTER, YES);
           run device (BA, AB, SLAVE, YES);
        }
}
```

B.1.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 48, errors: 0
     238 states, stored
      74 states, matched
     312 transitions (= stored+matched)
```

```
       1 atomic steps
hash conflicts: 1 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 39, state 12, "out!4"
      line 51, state 31, "(1)"
      line 60, state 36, "state = 0"
      line 60, state 37, "outstanding = 0"
      line 59, state 38, "((state==1))"
      line 71, state 49, "(1)"
      (6 of 57 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.2 PAIRING

```
#define     LMP_not_accepted               4
#define     LMP_detach                     7
#define LMP_in_rand                     8
#define LMP_comb_key                    9
#define LMP_unit_key                   10
#define LMP_au_rand                    11
#define     LMP_sres                       12

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define TEMP1   7
#define s_auth     1
#define s_ukey     2
#define s_ckey     3
#define s_detach  12

#define CHAN_LEN  2  /* length of channel (number of messages to be
stored) */


proctype device (chan in, out; bit device_type, linkkey)
{
   byte state=NORMAL, linkey=linkkey;
   bit outstanding=NO, auth_done=NO;


BEGIN:
   do
/*************************************/
/* 3.2 Pairing                       */
/*************************************/
   :: (((outstanding==NO)&&(auth_done==NO))&&(state==NORMAL)) ->
      if
      :: (linkey==NO) -> out!LMP_in_rand; linkey=TEMP1;
```

```
      :: !(linkey==NO) -> skip;
      fi;
      out!LMP_au_rand;
      outstanding=YES;
      state=s_auth;

   :: in?LMP_in_rand ->
      if
      :: (state==s_detach) -> skip;
      :: !(state==s_detach) -> linkey=TEMP1;
      fi;

   :: in?LMP_au_rand ->
      if
      :: (state==s_detach) -> skip;
      :: !(state==s_detach) ->
         if
         :: (linkey==YES) -> out!LMP_sres;
         :: (linkey==TEMP1) -> out!LMP_sres;
         :: (linkey==NO) -> out!LMP_not_accepted;
         :: !((linkey==NO)||((linkey==YES)||(linkey==TEMP1))) -> skip;
         fi;
      fi;

   :: in?LMP_sres ->
      if
      :: (state==s_detach) -> skip;
      :: (state==s_auth) ->
         if
         :: auth_done=YES ->
            if
            :: (linkey==TEMP1) ->
               if
               :: out!LMP_unit_key; outstanding=YES; state=s_ukey;
               :: out!LMP_comb_key; outstanding=YES; state=s_ckey;
               fi;
            :: !(linkey==TEMP1) -> outstanding=NO; state=NORMAL;
            fi;
         :: auth_done=NO; outstanding=NO; state=NORMAL;
         :: out!LMP_detach; outstanding=YES; state=s_detach;
         fi;
      fi;

   :: in?LMP_unit_key ->
      if
      :: (state==s_detach) -> skip;
      :: (state==s_ukey) -> outstanding=NO; state=NORMAL;
      :: (state==s_ckey) -> outstanding=NO; state=NORMAL;
      :: !((state==s_ukey)||(state==s_ckey)||(state==s_detach)) ->
         if
         :: out!LMP_unit_key;
         :: out!LMP_comb_key;
         fi;
      fi;
      linkey=YES;

   :: in?LMP_comb_key ->
```

```
      if
      :: (state==s_detach) -> skip;
      :: (state==s_ukey) -> outstanding=NO; state=NORMAL;
      :: (state==s_ckey) -> outstanding=NO; state=NORMAL;
      :: !((state==s_ukey)||(state==s_ckey)||(state==s_detach)) ->
         if
         :: out!LMP_unit_key; printf("refuse to change link key\n");
         :: out!LMP_comb_key; printf("accept change of link key\n");
         fi;
      fi;
      linkey=YES;


/*************************************/
/* ACEPTED and NOT_ACCEPTED          */
/*************************************/

   :: in?LMP_not_accepted ->
      if
      :: (state==s_detach) -> skip;
      :: (state==s_auth) -> state=NORMAL; outstanding=NO;
      fi;

   :: in?LMP_detach -> break;

   :: timeout ->
      if
      :: (state==s_detach)&&(len(out)==0) -> break;
      :: (state==s_detach)&&!(len(out)==0) -> skip;
      :: !(state==s_detach) ->
          if
          :: (auth_done==YES) -> skip;
          :: (auth_done==NO) -> skip;
          fi;
      fi;
   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
           run device (AB, BA, MASTER,NO);
           run device (BA, AB, SLAVE,NO);
        }
}
```

B.2.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
```

```
      invalid endstates +

State-vector 40 byte, depth reached 90, errors: 0
    1105 states, stored
     225 states, matched
    1330 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 4 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 45, state 14, "(1)"
      line 56, state 28, "out!4"
      line 57, state 30, "(1)"
      line 63, state 37, "(1)"
      line 113, state 102, "(1)"
      line 114, state 104, "state = 0"
      line 114, state 105, "outstanding = 0"
      line 112, state 106, "((state==12))"
      line 112, state 106, "((state==1))"
      line 122, state 114, "(1)"
      line 126, state 119, "(1)"
      (10 of 127 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.3 CHANGE LINK KEY

```
#define LMP_comb_key                  9
#define LMP_unit_key                 10

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define TEMP1   7
#define s_ckey     3

#define CHAN_LEN  2  /* length of channel (number of messages to be
stored) */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL;
   bit outstanding=NO, auth_done=YES;


BEGIN:
   do
/*************************************/
/* 3.3 Change Link Key               */
```

```
/************************************/
   :: ((outstanding==NO)&&(auth_done==YES)) ->
          out!LMP_comb_key; outstanding=YES; state=s_ckey;

   :: in?LMP_unit_key ->
      if
      :: (state==s_ckey) -> outstanding=NO; state=NORMAL;
      :: !(state==s_ckey) -> skip;
      fi;
      printf("link key not changed\n");


   :: in?LMP_comb_key ->
      if
      :: (state==s_ckey) -> outstanding=NO; state=NORMAL;
                            printf("link key changed\n");
      :: !(state==s_ckey) ->
         if
         :: out!LMP_unit_key; printf("refuse to change link key\n");
         :: out!LMP_comb_key; printf("accept change of link key\n");
         fi;
      fi;

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
           run device (AB, BA, MASTER);
           run device (BA, AB, SLAVE);
        }
}
```

B.3.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 59, errors: 0
     128 states, stored
      57 states, matched
     185 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)
```

```
unreached in proctype device
      line 32, state 10, "(1)"
      line 51, state 32, "-end-"
      (2 of 32 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.4 CHANGE THE CURRENT KEY

```
#define     LMP_accepted                    3
#define LMP_temp_rand                 13
#define LMP_temp_key                  14
#define LMP_use_semi_permanent_key 50

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define TEMP1   7
/* use of previous indicates linkey==YES */
#define PREVIOUS 2
#define s_semi     4

#define CHAN_LEN  2  /* length of channel (number of messages to be
stored) */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL, linkey=NO;
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.4 Change Current Link Key       */
/*************************************/
   :: ((len(out)==0)&&((outstanding==NO)&&(device_type==MASTER))) ->
          out!LMP_temp_rand;
          out!LMP_temp_key;

   :: in?LMP_temp_rand -> skip;

   :: in?LMP_temp_key ->
          linkey=TEMP1;
          printf("link key changed to temporary key\n");

   :: ((len(out)==0)&&((outstanding==NO)&&(device_type==MASTER))) ->
          out!LMP_use_semi_permanent_key;
          outstanding=YES;
          state=s_semi;
```

```
   :: in?LMP_use_semi_permanent_key ->
          out!LMP_accepted;
          linkey=PREVIOUS;

/*************************************/
/* ACEPTED and NOT_ACCEPTED           */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_semi) ->  state=NORMAL; outstanding=NO;
linkey=PREVIOUS;
      :: !(state==s_semi) -> skip;
      fi;


   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
           run device (AB, BA, MASTER);
           run device (BA, AB, SLAVE);
        }
}
```

B.4.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 57, errors: 0
      85 states, stored
      26 states, matched
     111 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 55, state 22, "(1)"
      line 62, state 29, "-end-"
      (2 of 29 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.5 ENCYPTION

```
#define     LMP_accepted                    3
#define     LMP_not_accepted                4
#define LMP_encryption_mode_req       15
#define LMP_encryption_key_size_req   16
#define LMP_start_encryption_req      17
#define LMP_stop_encryption_req       18

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
/* security (0=NO, 1=YES, 2=STAGE1, 3=STAGE2) YES=ENCRYPTION ON*/
#define STAGE1   2
#define STAGE2   3
#define s_neg_en   50  /* negotiation of encryption */
#define s_neg_size 51  /* negotiation of key size */
#define s_bg_en    52  /* begin encryption */
#define s_end_en   53  /* end encryption */

#define CHAN_LEN  2  /* length of channel (number of messages to be
stored) */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL, security=NO;
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.5 Encryption                    */
/*************************************/
   :: ((security==NO)&&((outstanding==NO)&&(device_type==MASTER))) ->
          out!LMP_encryption_mode_req;
          outstanding=YES;
          state=s_neg_en;

   :: in?LMP_encryption_mode_req ->
      if
      :: out!LMP_accepted; security=STAGE1;
      :: out!LMP_not_accepted;
      fi;

   :: ((security==STAGE1)&&((outstanding==NO)&&(device_type==MASTER)))
->
          out!LMP_encryption_key_size_req;
          outstanding=YES;
          state=s_neg_size;

   :: in?LMP_encryption_key_size_req ->
```

```
      if
      :: (device_type==SLAVE) ->
         if
         :: out!LMP_accepted; security=STAGE2;
         :: out!LMP_encryption_key_size_req; outstanding=YES;
state=s_neg_size;
         fi;
      :: (device_type==MASTER) ->
         if
         :: out!LMP_accepted; security=STAGE2;
         :: out!LMP_not_accepted; security=NO;
         fi;
         outstanding=NO;
         state=NORMAL;
      fi;

   :: ((security==STAGE2)&&((outstanding==NO)&&(device_type==MASTER)))
->
          out!LMP_start_encryption_req;
          outstanding=YES;
          state=s_bg_en;

   :: in?LMP_start_encryption_req ->
          out!LMP_accepted; security=YES;

   :: ((security==YES)&&((outstanding==NO)&&(device_type==MASTER))) ->
          out!LMP_stop_encryption_req;
          outstanding=YES;
          state=s_end_en;

   :: in?LMP_stop_encryption_req ->
          out!LMP_accepted; security=NO;   /* this may be STAGE1 or
STAGE2 */


/*************************************/
/* ACEPTED and NOT_ACCEPTED            */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_neg_en) ->  state=NORMAL; outstanding=NO;
security=STAGE1;
      :: (state==s_neg_size) ->  state=NORMAL; outstanding=NO;
security=STAGE2;
      :: (state==s_bg_en) ->  state=NORMAL; outstanding=NO;
security=YES;
      :: (state==s_end_en) ->  state=NORMAL; outstanding=NO;
security=NO;
      ::
!((state==s_neg_size)||(state==s_neg_en)||(state==s_bg_en)||(state==s_e
nd_en)) ->
            skip;
      fi;


   :: in?LMP_not_accepted ->
      if
```

```
      :: (state==s_neg_en) ->  state=NORMAL; outstanding=NO;
security=NO;
      :: (state==s_neg_size) ->  state=NORMAL; outstanding=NO;
security=NO;
      :: !((state==s_neg_size)||(state==s_neg_en)) -> skip;
      fi;

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
           run device (AB, BA, MASTER);
           run device (BA, AB, SLAVE);
        }
}
```

B.5.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 50, errors: 0
     102 states, stored
      17 states, matched
     119 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 94, state 67, "(1)"
      line 102, state 80, "(1)"
      line 108, state 87, "-end-"
      (3 of 87 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.6 CLOCK OFFSET REQUEST

```
#define LMP_clkoffset_req             5
```

```
#define LMP_clkoffset_res             6

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.6 Clock Offset Request          */
/*************************************/
   :: ((outstanding==NO)&&(device_type==MASTER)) ->
                 out!LMP_clkoffset_req;
                 outstanding=YES;
   :: in?LMP_clkoffset_req -> out!LMP_clkoffset_res;
   :: in?LMP_clkoffset_res -> outstanding=NO;
               /*assert only MASTER can receive */

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.6.2 Spin output

B.6.2.1 Model with channel length of size 1 or 2

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +
```

```
State-vector 36 byte, depth reached 8, errors: 0
       8 states, stored
       2 states, matched
      10 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 32, state 12, "-end-"
      (1 of 12 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.6.2.2 Model with channel length of size 0

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 36 byte, depth reached 8, errors: 0
       6 states, stored
       2 states, matched
       8 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 32, state 12, "-end-"
      (1 of 12 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.7 TIMING ACCURACY INFORMATION REQUEST

```
#define     LMP_not_accepted              4
#define LMP_timing_accuracy_req    47
#define LMP_timing_accuracy_res    48

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
```

```
#define NORMAL 0
#define s_time     7

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL;
   bit outstanding=NO;


BEGIN:
   do
/*****************************************/
/* 3.7 Timing Accuracy Information Request */
/*****************************************/
   :: (outstanding==NO) ->
                out!LMP_timing_accuracy_req;
                outstanding=YES;
                state=s_time;
   :: in?LMP_timing_accuracy_req ->
        if
        :: out!LMP_timing_accuracy_res;
        :: out!LMP_not_accepted;
        fi;
   :: in?LMP_timing_accuracy_res -> state=NORMAL; outstanding=NO;

   :: in?LMP_not_accepted ->
      if
      :: (state==s_time) -> state=NORMAL; outstanding=NO;
      fi;
   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.7.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +
```

```
State-vector 40 byte, depth reached 33, errors: 0
      92 states, stored
      45 states, matched
     137 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 44, state 23, "-end-"
      (1 of 23 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.8 LMP VERSION

```
#define LMP_version_req            37
#define LMP_version_res            38

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.8 LMP Version                   */
/*************************************/
   :: (outstanding==NO) ->
                out!LMP_version_req;
                outstanding=YES;
   :: in?LMP_version_req -> out!LMP_version_res;
   :: in?LMP_version_res -> outstanding=NO;

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
```

```
      atomic {
         run device (AB, BA, MASTER);
         run device (BA, AB, SLAVE);
        }
}
```

B.8.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 36 byte, depth reached 23, errors: 0
      38 states, stored
      24 states, matched
      62 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 31, state 12, "-end-"
      (1 of 12 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.9 SUPPORTED FEATURES

```
#define LMP_features_req          39
#define LMP_features_res          40

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   bit outstanding=NO;


BEGIN:
   do
```

```
/************************************/
/* 3.9 Supported Features           */
/************************************/
   :: (outstanding==NO) ->
                out!LMP_features_req;
                outstanding=YES;
   :: in?LMP_features_req -> out!LMP_features_res;
   :: in?LMP_features_res -> outstanding=NO;

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.9.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 36 byte, depth reached 23, errors: 0
      38 states, stored
      24 states, matched
      62 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 1 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 31, state 12, "-end-"
      (1 of 12 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.10 SWITCH OF MASTER SLAVE ROLE

```
#define     LMP_accepted                  3
#define     LMP_not_accepted              4
#define LMP_switch_req               19
```

```
#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define s_switch  10

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL;
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.10 Switch of Master/Slave Role   */
/*************************************/
   :: (outstanding==NO) ->
                out!LMP_switch_req;
                state=s_switch;
                outstanding=YES;
   :: in?LMP_switch_req ->
      if
      :: out!LMP_accepted;
         if
         :: (device_type==SLAVE) -> device_type=MASTER;
         :: (device_type==MASTER) -> device_type=SLAVE;
         fi;
      :: out!LMP_not_accepted;
      fi;

/*************************************/
/* ACEPTED and NOT_ACCEPTED            */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_switch) ->
         if
         :: (device_type==SLAVE) -> device_type=MASTER;
         :: (device_type==MASTER) -> device_type=SLAVE;
         fi;
         state=NORMAL;
         outstanding=NO;
      fi;

   :: in?LMP_not_accepted ->
      if
      :: (state==s_switch) -> state=NORMAL; outstanding=NO;
      fi;

   :: timeout;
```

```
   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.10.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 85, errors: 0
     279 states, stored
      96 states, matched
     375 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 62, state 38, "-end-"
      (1 of 38 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.11 NAME REQUEST

```
#define LMP_name_req                  1
#define LMP_name_res                  2

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */
```

```
proctype device (chan in, out; bit device_type)
{
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.11 Name request                 */
/*************************************/
   :: (outstanding==NO) ->
                out!LMP_name_req;
                outstanding=YES;
   :: in?LMP_name_req -> out!LMP_name_res;
   :: in?LMP_name_res -> outstanding=NO;

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.11.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 36 byte, depth reached 23, errors: 0
      38 states, stored
      24 states, matched
      62 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 31, state 12, "-end-"
      (1 of 12 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.12 DETACH

```
#define     LMP_detach                    7

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.12 Detach                       */
/*************************************/
   :: (outstanding==NO) -> out!LMP_detach -> outstanding=YES;
   :: in?LMP_detach -> break;

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.12.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 36 byte, depth reached 13, errors: 0
      26 states, stored
```

```
       7 states, matched
      33 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      (0 of 10 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.13 HOLD MODE

B.13.1.1 Model with channel length of size 0

```
#define     LMP_accepted                    3
#define     LMP_not_accepted                4
#define LMP_hold1                     20
#define LMP_hold_req                  21

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0

#define CHAN_LEN  0    /* length of channel (number of messages to be
stored) */

bit outstanding=NO;
bit outstand1=NO, outstand2=NO;
bit holded1 = NO, holded2 = NO;
bit req_send1 = NO, req_send2 = NO;
bit hold_accepted = NO;

proctype device (chan in, out; bit device_type, holded, outstand,
req_send)
{
end:
   do
   :: atomic {
            holded == NO && outstand == NO && outstanding == NO ->
                  outstanding = YES;
                  if
                  ::    if
                        :: device_type == MASTER && hold_accepted ==
YES ->
                              out!LMP_hold1;
                              holded = YES
                        :: device_type == SLAVE ->
                              out!LMP_hold1;
                              holded = YES;
```

```
                              outstand = YES
                        :: else -> skip
                        fi
                  :: out!LMP_hold_req;
                        req_send = YES;
                        outstand = YES
                  fi;
                  outstanding = NO
            }
   :: atomic {
            in?LMP_hold1 ->
                  outstanding = YES;
                  if
                  :: device_type == MASTER ->
                        hold_accepted = YES;
                        out!LMP_hold1
                  :: device_type == SLAVE ->
                        outstand = NO
                  fi;
                  outstanding = NO
            }
   :: atomic {
            in?LMP_hold_req ->
                  outstanding = YES;
                  if
                  :: req_send == YES ->
                        req_send = NO;
                        outstand = NO
                  :: req_send == NO -> skip
                  fi;
                  if
                  :: out!LMP_hold_req;
                        req_send = YES;
                        outstand = YES
                  :: out!LMP_accepted
                  :: out!LMP_not_accepted
                  fi;
                  outstanding = NO
            }
   :: in?LMP_accepted ->
            holded = YES;
            req_send = NO;
            outstand = NO
   :: in?LMP_not_accepted ->
            req_send = NO;
            outstand = NO
   od
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (BA, AB, MASTER, holded1, outstand1, req_send1);
             run device (AB, BA, SLAVE, holded2, outstand2, req_send2)
      }
}
```

B.13.1.2 Spin output with channel length of size 0

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 67, errors: 0
      99 states, stored
      43 states, matched
     142 transitions (= stored+matched)
     227 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 81, state 60, "-end-"
      (1 of 60 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.13.2.1 Model with channel length of size 1

```
#define     LMP_accepted                    3
#define     LMP_not_accepted                4
#define LMP_hold1                      20
#define LMP_hold_req0                   21
#define LMP_hold_req1                  121

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define s_time     7
#define s_switch  10
#define s_hold0    13   /* used for hold */
#define s_hold_r0  131  /* used for hold request master init*/
#define s_hold_r1  113  /* used for hold request slave init*/
#define s_qos     18
#define s_sco_add 19
#define s_sco_rem 191
#define s_slot    20

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */
```

```
proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL;
   bit outstanding=NO, previous_h=NO;


BEGIN:
   do
/*************************************/
/* 3.13 Hold Mode                    */
/*************************************/
/* can not send if something is in the transmit queue */
/* can not send if unit has not received a previous hold request */
/* can not send if waiting for a response    */
   ::
((len(out)==0)&&((previous_h==YES)&&((outstanding==NO)&&(device_type==M
ASTER)))) ->
                  out!LMP_hold1;
                  goto HOLD_STATE;

   :: ((len(out)==0)&&((outstanding==NO)&&(device_type==SLAVE))) ->
                  out!LMP_hold1;
                  outstanding=YES;
                  state=s_hold0;

   :: in?LMP_hold1 ->
      if
      :: (device_type==SLAVE) -> goto HOLD_STATE;
      :: (device_type==MASTER) -> out!LMP_hold1; previous_h=YES; goto
HOLD_STATE;
      fi;

   :: ((outstanding==NO)&&(device_type==MASTER)) ->
                  out!LMP_hold_req0;
                  outstanding=YES;
                  state=s_hold_r0;

   :: ((outstanding==NO)&&(device_type==SLAVE)) ->
                  out!LMP_hold_req1;
                  outstanding=YES;
                  state=s_hold_r1;

   :: in?LMP_hold_req0 ->
      if
      :: (device_type==SLAVE) ->
         if
         ::
((outstanding==YES)&&((state==s_hold0)||(state==s_hold_r1))) ->
                        out!LMP_not_accepted;
         :: ((outstanding==YES)&&(state==s_hold_r0)) ->
            if
            :: out!LMP_hold_req0;
            :: out!LMP_accepted; goto HOLD_STATE;
            :: out!LMP_not_accepted; outstanding=NO; state=NORMAL;
            fi;
```

```
/* next line need for combining with models other functions.
Unreachable otherwise. */
         ::
((outstanding==YES)&&!(((state==s_hold_r0)||(state==s_hold0))||(state==
s_hold_r1))) ->
                         out!LMP_not_accepted;
         :: (outstanding==NO) ->
            if
            :: out!LMP_hold_req0; outstanding=YES; state=s_hold_r0;
            :: out!LMP_accepted; goto HOLD_STATE;
            :: out!LMP_not_accepted;
            fi;
         fi;
      :: (device_type==MASTER) ->
         if
/* The next condition cannot be met because new procedures were defined
to not */
/* permit multiple requests (i.e. multiple requests would be denied.
*/
         :: ((outstanding==YES)&&(state==s_hold_r1)) ->
                        out!LMP_not_accepted;
         :: ((outstanding==YES)&&(state==s_hold_r0)) ->
            if
            :: out!LMP_hold_req0;
            :: out!LMP_accepted; goto HOLD_STATE;
            :: out!LMP_not_accepted; outstanding=NO; state=NORMAL;
            fi;
/* condition cannot occur because MASTER cannot be an any state other
than these two. */
/* However, when added to models of other functions, this condition can
be met.  */
         ::
((outstanding==YES)&&!((state==s_hold_r0)||(state==s_hold_r1))) ->
                         out!LMP_not_accepted;
         fi;
      fi;

   :: in?LMP_hold_req1 ->
      if
      :: (device_type==SLAVE) ->
         if
         :: ((outstanding==YES)&&(state==s_hold_r1)) ->
            if
            :: out!LMP_hold_req1;
            :: out!LMP_accepted; goto HOLD_STATE;
            :: out!LMP_not_accepted; outstanding=NO; state=NORMAL;
            fi;
/* condition cannot occur because SLAVE cannot be an any state other
than these two. */
/* However, when added to models of other functions, this condition can
be met.  */
         ::
((outstanding==YES)&&(!(((state==s_hold_r0)||(state==s_hold0))||(state=
=s_hold_r1)))) ->
                         out!LMP_not_accepted;
         fi;
      :: (device_type==MASTER) ->
```

```
         if
         :: ((outstanding==YES)&&(state==s_hold_r0)) ->
                        out!LMP_not_accepted;
         :: ((outstanding==YES)&&(state==s_hold_r1)) ->
            if
            :: out!LMP_hold_req1;
            :: out!LMP_accepted; goto HOLD_STATE;
            :: out!LMP_not_accepted; outstanding=NO; state=NORMAL;
            fi;
/* condition cannot occur because MASTER cannot be an any state other
than these two. */
/* However, when added to models of other functions, this condition can
be met.  */
         ::
((outstanding==YES)&&!((state==s_hold_r0)||(state==s_hold_r1))) ->
                         out!LMP_not_accepted;
         :: (outstanding==NO) ->
            if
            :: out!LMP_hold_req1; outstanding=YES; state=s_hold_r1;
            :: out!LMP_accepted; goto HOLD_STATE;
            :: out!LMP_not_accepted;
            fi;
         fi;
      fi;


/*************************************/
/* ACEPTED and NOT_ACCEPTED          */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_hold_r0) -> goto HOLD_STATE;
      :: (state==s_hold_r1) -> goto HOLD_STATE;
      fi;

   :: in?LMP_not_accepted ->
      if
      :: (state==s_hold_r0) -> state=NORMAL; outstanding=NO;
      :: (state==s_hold_r1) -> state=NORMAL; outstanding=NO;
      fi;

   :: timeout;

   od;

HOLD_STATE:
   printf("Entered HOLD state\n");
   /* empty in & out queues */
   do
   :: in?LMP_hold1;
   :: in?LMP_hold_req0;
   :: in?LMP_hold_req1;
   :: in?LMP_accepted;
   :: in?LMP_not_accepted;
   :: timeout -> break;
   od;
   /* before leaving HOLD_STATE set everything to "go" */
```

```
   state=NORMAL;
   outstanding=NO;
progressA:   goto BEGIN;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.13.2.2 Spin output with channel length of size 1

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 72, errors: 0
     452 states, stored
     106 states, matched
     558 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 78, state 39, "out!4"
      line 91, state 53, "out!4"
      line 101, state 64, "out!4"
      line 118, state 81, "out!4"
      line 133, state 97, "out!4"
      line 178, state 146, "-end-"
      (6 of 146 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.14 SNIFF MODE

B.14.1.1 Model with channel length of size 0

```
#define     LMP_accepted                    3
#define     LMP_not_accepted                4
#define LMP_sniff1                    22
```

```
#define LMP_sniff_req                23
#define LMP_unsniff_req              24

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0

#define CHAN_LEN  0    /* length of channel (number of messages to be
stored) */

bit outstand1= NO, outstand2= NO;
bit sniffed1 = NO, sniffed2 = NO;
bit req_send1 = NO, req_send2 = NO;

/* Assumption: only the SLAVE can enter the sniff mode */

proctype device (chan in, out; bit device_type, outstand, sniffed,
req_send)
{
end:
   do
   :: ((sniffed == NO) && (outstand == NO)) ->
      if
      ::    if
            :: device_type == MASTER ->
                  out!LMP_sniff1;
                  sniffed = YES
            :: device_type == SLAVE -> skip
            fi
      :: out!LMP_sniff_req;
                  req_send = YES;
                  outstand = YES
      fi
   :: atomic {
            ((sniffed == YES) && (outstand == NO)) ->
                  out!LMP_unsniff_req;
                  outstand = YES
            }
   :: in?LMP_sniff1 -> sniffed = YES
   :: in?LMP_unsniff_req ->
            out!LMP_accepted;
            sniffed = NO
   :: in?LMP_sniff_req ->
      if
      :: req_send == YES ->
            req_send = NO;
            outstand = NO
      :: req_send == NO -> skip
      fi;
      if
      :: out!LMP_sniff_req;
            req_send = YES;
            outstand = YES
      :: out!LMP_accepted;
            sniffed = YES
      :: out!LMP_not_accepted
```

```
      fi
   :: in?LMP_accepted ->
      outstand = NO;
      if
      :: sniffed == NO ->
            req_send = NO;
            sniffed = YES
      :: sniffed == YES -> sniffed = NO
      fi
   :: in?LMP_not_accepted ->
            req_send = NO;
            outstand = NO
   od
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
        run device (BA, AB, MASTER, outstand1, sniffed1, req_send1);
        run device (AB, BA, SLAVE, outstand2, sniffed2, req_send2)
      }
}
```

B.14.1.2 Spin output with channel length of size 0

```
 (Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 26, errors: 0
      67 states, stored
      41 states, matched
     108 transitions (= stored+matched)
       3 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 72, state 54, "-end-"
      (1 of 54 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.14.2.1 Model with channel length of size 1

```
#define     LMP_accepted                  3
```

```
#define     LMP_not_accepted               4
#define LMP_sniff1                    22
#define LMP_sniff_req0                23
#define LMP_sniff_req1               123
#define LMP_unsniff_req               24


#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define s_sniff_r0  141  /* used for sniff request master init */
#define s_sniff_r1 114   /* used for sniff request slave init */
#define s_unsniff   140

#define CHAN_LEN  1   /* length of channel (number of messages to be
stored) */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL;
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.14 Sniff Mode                   */
/*************************************/
   :: ((len(out)==0)&&((outstanding==NO)&&(device_type==MASTER))) ->
                  out!LMP_sniff1;
                  goto SNIFF_STATE;

/* only possible for SLAVE */
   :: in?LMP_sniff1 -> goto SNIFF_STATE;

   :: ((outstanding==NO)&&(device_type==MASTER)) ->
                  out!LMP_sniff_req0;
                  outstanding=YES;
                  state=s_sniff_r0;

   :: ((outstanding==NO)&&(device_type==SLAVE)) ->
                  out!LMP_sniff_req1;
                  outstanding=YES;
                  state=s_sniff_r1;

   :: in?LMP_sniff_req0 ->
      if
      :: (device_type==SLAVE) ->
         if
         :: ((outstanding==YES)&&(state==s_sniff_r1)) ->
                        out!LMP_not_accepted;
         :: ((outstanding==YES)&&(state==s_sniff_r0)) ->
            if
            :: out!LMP_sniff_req0;
```

```
            :: out!LMP_accepted; outstanding=NO; state=NORMAL; goto
SNIFF_STATE;
            :: out!LMP_not_accepted; outstanding=NO; state=NORMAL;
            fi;
/* next line needed for combining with models of other functions.
Unreachable otherwise. */
         ::
((outstanding==YES)&&!((state==s_sniff_r0)||(state==s_sniff_r1))) ->
                        out!LMP_not_accepted;
         :: (outstanding==NO) ->
            if
            :: out!LMP_sniff_req0; outstanding=YES; state=s_sniff_r0;
            :: out!LMP_accepted; goto SNIFF_STATE;
            :: out!LMP_not_accepted;
            fi;
         fi;
      :: (device_type==MASTER) ->
            if
            :: out!LMP_sniff_req0;
            :: out!LMP_accepted; outstanding=NO; state=NORMAL; goto
SNIFF_STATE;
            :: out!LMP_not_accepted; outstanding=NO; state=NORMAL;
            fi;
      fi;

   :: in?LMP_sniff_req1 ->
      if
      :: (device_type==SLAVE) ->
         if
         :: ((outstanding==YES)&&(state==s_sniff_r1)) ->
            if
            :: out!LMP_sniff_req1;
            :: out!LMP_accepted; outstanding=NO; state=NORMAL; goto
SNIFF_STATE;
            :: out!LMP_not_accepted; outstanding=NO; state=NORMAL;
            fi;
         fi;
      :: (device_type==MASTER) ->
         if
         :: ((outstanding==YES)&&(state==s_sniff_r0)) ->
                        out!LMP_not_accepted;
         :: ((outstanding==YES)&&(state==s_sniff_r1)) ->
            if
            :: out!LMP_sniff_req1;
            :: out!LMP_accepted; outstanding=NO; state=NORMAL; goto
SNIFF_STATE;
            :: out!LMP_not_accepted; outstanding=NO; state=NORMAL;
            fi;
/* condition cannot occur because MASTER cannot be an any state other
than these two. */
/* However, when added to models of other functions, this condition can
be met.  */
         ::
((outstanding==YES)&&!((state==s_sniff_r0)||(state==s_sniff_r1))) ->
                        out!LMP_not_accepted;
         :: (outstanding==NO) ->
            if
```

```
            :: out!LMP_sniff_req1; outstanding=YES; state=s_sniff_r1;
            :: out!LMP_accepted; goto SNIFF_STATE;
            :: out!LMP_not_accepted;
            fi;
         fi;
      fi;


/*************************************/
/* ACEPTED and NOT_ACCEPTED            */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_sniff_r0) -> outstanding=NO; state=NORMAL; goto
SNIFF_STATE;
      :: (state==s_sniff_r1) -> outstanding=NO; state=NORMAL; goto
SNIFF_STATE;
      fi;

   :: in?LMP_not_accepted ->
      if
      :: (state==s_sniff_r0) -> state=NORMAL; outstanding=NO;
      :: (state==s_sniff_r1) -> state=NORMAL; outstanding=NO;
      fi;

   :: timeout;

   od;

SNIFF_STATE:
   printf("Entered SNIFF state\n");
   /* empty in & out queues */
   do
   :: in?LMP_sniff1;
   :: in?LMP_sniff_req0;
   :: in?LMP_sniff_req1;
   :: in?LMP_accepted;
      if
      :: (state==s_unsniff) -> state=NORMAL; outstanding=NO; goto
BEGIN;
/* condition not possible using this model alone, but possible when
combined with models */
/* of other functions. */
      :: (state!=s_unsniff) -> skip;
      fi;
   :: in?LMP_not_accepted;
   :: ((outstanding==NO)&&(len(in)==0))&&(len(out)==0) ->
        out!LMP_unsniff_req;
        outstanding=YES;
        state=s_unsniff;
   :: in?LMP_unsniff_req ->
      if
        :: (state==s_unsniff) -> out!LMP_accepted;
        :: (state!=s_unsniff) -> out!LMP_accepted; goto BEGIN;
      fi;
   :: timeout;
   od;
```

```
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.14.2.2 Spin output with channel length of size 1

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 88, errors: 0
     272 states, stored
     105 states, matched
     377 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 63, state 30, "out!4"
      line 103, state 85, "out!4"
      line 145, state 133, "(1)"
      line 159, state 153, "-end-"
      (4 of 153 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.15 PARK MODE

B.15.1.1 Model with channel length of size 0

B.15.1.2 Spin output with channel length of size 0

B.15.2.1 Model with channel length of size 1

```
#define     LMP_accepted                    3
#define     LMP_not_accepted                4
#define LMP_park_req                  25
#define LMP_park1                     26
#define LMP_set_broadcast_scan_window 27
#define LMP_modify_beacon             28
#define LMP_unpark_BD_ADDR_req        29
#define LMP_unpark_PM_ADDR_req        30

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define s_park    15
#define s_unpark  151

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL;
   bit outstanding=NO, parked=NO;


BEGIN:
   do
/*************************************/
/* 3.15 Park Mode                    */
/*************************************/
   :: ((parked==NO)&&(outstanding==NO)&&(device_type==MASTER)) ->
                  out!LMP_park1;
                  parked=YES;

   :: in?LMP_park1 ->
      if
      :: (state==s_park) -> outstanding=NO; state=NORMAL; goto
PARK_STATE;
      :: (state!=s_park) -> goto PARK_STATE;
      fi;

   :: ((parked==NO)&&(outstanding==NO)&&(device_type==MASTER)) ->
                  out!LMP_park_req;
                  outstanding=YES;
                  state=s_park;

   :: ((parked==NO)&&(outstanding==NO)&&(device_type==SLAVE)) ->
                  out!LMP_park_req;
                  outstanding=YES;
                  state=s_park;

   :: in?LMP_park_req ->
      if
```

```
      :: (device_type==SLAVE) ->
         if
         :: out!LMP_accepted;
         :: out!LMP_not_accepted;
         fi;
      :: (device_type==MASTER) ->
         if
         :: (parked==YES) -> skip;
         :: (parked==NO) ->
            if
            :: out!LMP_park1; parked=YES;
            :: out!LMP_not_accepted;
            fi;
         fi;
      fi;

   ::
((parked==YES)&&((len(out)==0)&&(outstanding==NO)&&(device_type==MASTER
))) ->
                     out!LMP_set_broadcast_scan_window;

   ::
((parked==YES)&&((len(out)==0)&&(outstanding==NO)&&(device_type==MASTER
))) ->
                     out!LMP_modify_beacon;

   :: ((parked==YES)&&((outstanding==NO)&&(device_type==MASTER))) ->
                     out!LMP_unpark_BD_ADDR_req;
                     outstanding=YES;
                     state=s_unpark;

   :: ((parked==YES)&&((outstanding==NO)&&(device_type==MASTER))) ->
                     out!LMP_unpark_PM_ADDR_req;
                     outstanding=YES;
                     state=s_unpark;


/*************************************/
/* ACEPTED and NOT_ACCEPTED          */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_park) ->
         if
         :: (device_type==MASTER) ->
            if
            :: (parked==YES) -> skip;
            :: (parked==NO) -> parked=YES; out!LMP_park1;
            fi;
         :: (device_type==SLAVE) ->  skip;
         fi;
      :: (state==s_unpark) -> parked=NO; outstanding=NO; state=NORMAL;
      fi;

   :: in?LMP_not_accepted ->
      if
      :: (state==s_park) -> state=NORMAL; outstanding=NO;
```

```
      fi;

   od;

PARK_STATE:
   /* while in park state only broadcast messages are accepted */
   do
   :: in?LMP_set_broadcast_scan_window;
   :: in?LMP_modify_beacon;
   :: in?LMP_unpark_BD_ADDR_req ->
          out!LMP_accepted; outstanding=NO; state=NORMAL; goto BEGIN;
   :: in?LMP_unpark_PM_ADDR_req ->
          out!LMP_accepted; outstanding=NO; state=NORMAL; goto BEGIN;
/* receive leftover PDUs from queue and do nothing */
   :: in?LMP_accepted;
   :: in?LMP_not_accepted;
   :: in?LMP_park1;
   :: in?LMP_park_req;
   :: timeout;
   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.15.2.2 Spin output with channel length off size 1

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 40, errors: 0
     277 states, stored
      88 states, matched
     365 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 99, state 63, "(1)"
      line 127, state 101, "-end-"
      (2 of 101 states)
```

```
unreached in proctype :init:
      (0 of 4 states)
```

B.16 POWER CONTROL

```
#define LMP_incr_power_req            31
#define LMP_decr_power_req            32
#define LMP_max_power              33
#define LMP_min_power              34

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define MAX    1
#define NORMAL 0
#define MIN    -1

#define CHAN_LEN  1


proctype device (chan in, out; bit device_type)
{
   byte power=NORMAL;
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.16 Power Control                */
/*************************************/
   :: ((len(out)==0)&&((outstanding==NO)&&(power!=MAX))) ->
            out!LMP_incr_power_req;
            power=NORMAL;
   :: in?LMP_incr_power_req ->
      if
      :: skip               /* increment power, not at maximum */
      :: out!LMP_max_power;  /* increment power causes maximum power */
      fi;
   :: ((len(out)==0)&&((outstanding==NO)&&(power!=MIN))) ->
            out!LMP_decr_power_req;
            power=NORMAL;
   :: in?LMP_decr_power_req ->
      if
      :: skip;              /* decrement power, not at mimimum */
      :: out!LMP_min_power;  /* decrement power causes minimum power */
      fi;
   :: in?LMP_max_power -> power=MAX;
   :: in?LMP_min_power -> power=MIN;

   :: timeout;

   od;
}
```

```
init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.16.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 774, errors: 0
    2660 states, stored
    3085 states, matched
    5745 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 67 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 50, state 25, "-end-"
      (1 of 25 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.17 CHANNEL QUALITY DRIVEN CHANGE BETWEEN DM AND DH

```
#define LMP_auto_rate             35
#define LMP_preferred_rate        36

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define DH     1
#define DM     0

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type, tx_channel)
{
```

```
   bit p_type=DM, outstanding=NO, rx_channel=NO;


BEGIN:
   do
/*************************************/
/* 3.17 Channel Quality Driven change */
/*************************************/
   :: ((tx_channel==YES)&&((len(out)==0)&&(outstanding==NO))) ->
              out!LMP_auto_rate;
   :: in?LMP_auto_rate -> rx_channel=YES;
   :: ((rx_channel==YES)&&((len(out)==0)&&(outstanding==NO))) ->
              out!LMP_preferred_rate;
   :: in?LMP_preferred_rate ->
      if
      :: (p_type==DM) -> p_type=DH;
      :: (p_type==DH) -> p_type=DM;
      fi;

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER,YES);
         run device (BA, AB, SLAVE,YES);
        }
}
```

B.17.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 148, errors: 0
     405 states, stored
     375 states, matched
     780 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 5 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 38, state 18, "-end-"
```

```
      (1 of 18 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.18 QUALITY OF SERVICE (QoS)

```
#define     LMP_accepted                    3
#define     LMP_not_accepted                4
#define LMP_quality_of_service     41
#define LMP_quality_of_service_req 42

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define NORMAL 0
#define s_qos     18

#define CHAN_LEN  1    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL;
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.18 Quality of Service (QOS)      */
/*************************************/
   :: ((len(out)==0)&&((outstanding==NO)&&(device_type==MASTER))) ->
         out!LMP_quality_of_service;
   :: in?LMP_quality_of_service -> skip;
/* assert device_type==maseter and receive LMP_quality_of_service */
   :: (outstanding==NO) ->
         out!LMP_quality_of_service_req;
         state=s_qos;
         outstanding=YES;
   :: in?LMP_quality_of_service_req ->
      if
      :: out!LMP_accepted;
      :: out!LMP_not_accepted;
      fi;

/*************************************/
/* ACEPTED and NOT_ACCEPTED            */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_qos) -> state=NORMAL; outstanding=NO;
      fi;
```

```
   :: in?LMP_not_accepted ->
      if
      :: (state==s_qos) -> state=NORMAL; outstanding=NO;
      fi;

   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.18.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 65, errors: 0
     133 states, stored
      86 states, matched
     219 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 57, state 30, "-end-"
      (1 of 30 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.19 SCO LINKS

```
#define     LMP_accepted_m                  3
#define     LMP_not_accepted_m              4
#define LMP_SCO_link_req_m          43
#define LMP_remove_SCO_link_req    44
#define     LMP_accepted_s                  103  /* new messages to
fake 0/1 bit*/
```

```
#define     LMP_not_accepted_s               104  /* that distinguishes
initiated*/
#define LMP_SCO_link_req_s               143  /* transaction (see 2,
PART C) */

#define SLAVE  1    /* dual purpose: distinguish transaction id */
#define MASTER 0    /*               device function            */
#define YES    1
#define NO     0
#define MAX    1
#define NORMAL 0
#define MIN    -1
#define DH     1
#define DM     0
#define num_sco_links 3        /* for use in sco links */
#define s_sco_add 19
#define s_sco_add0 190
#define s_sco_rem 191

#define CHAN_LEN 1


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL, SCO=0;
   bit state2=NORMAL, outstanding=NO, outstanding2=NO;


BEGIN:
   do
/*************************************/
/* 3.19 SCO Links (add & remove)      */
/*************************************/
   :: ((outstanding==NO)&&(SCO<num_sco_links)) ->
                  if
                  :: (device_type==SLAVE) ->
                                 out!LMP_SCO_link_req_s;
                  :: (device_type==MASTER) ->
                                 out!LMP_SCO_link_req_m;
                  fi;
                  state=s_sco_add;
                  outstanding=YES;
   :: in?LMP_SCO_link_req_m ->
            if
            :: out!LMP_accepted_m; SCO=SCO+1;
            :: out!LMP_not_accepted_m;
            fi;
   :: in?LMP_SCO_link_req_s ->
            if
            :: (device_type==SLAVE) ->
               if
               :: out!LMP_accepted_s; SCO=SCO+1;
               :: out!LMP_not_accepted_s;
               fi;
               state=NORMAL;
               outstanding=NO;
            :: (device_type==MASTER) ->
```

```
               if
               :: out!LMP_SCO_link_req_s;
                  if
                  :: (outstanding==YES) -> outstanding2=YES;
                  :: (outstanding==NO) -> outstanding=YES;
                  fi;
                  state2=MAX;
               :: out!LMP_not_accepted_s;
               fi;
            fi;

   :: ((outstanding==NO)&&(SCO>0)) ->
                  out!LMP_remove_SCO_link_req;
                  state=s_sco_rem;
                  outstanding=YES;
   :: in?LMP_remove_SCO_link_req -> out!LMP_accepted_m; SCO=SCO-1;

/*************************************/
/* ACEPTED and NOT_ACCEPTED          */
/*************************************/
   :: in?LMP_accepted_m ->
      if
      :: (state==s_sco_add) -> SCO=SCO+1;
      :: (state==s_sco_rem) -> SCO=SCO-1;
      fi;
      state=NORMAL;
      if
      :: (outstanding2==YES) -> outstanding2=NO; /*leave outstanding
unchanged*/
      :: (outstanding2==NO) -> outstanding=NO;
      fi;

   :: in?LMP_not_accepted_m ->
      if
      :: (state==s_sco_add) -> state=NORMAL;
      fi;
      if
      :: (outstanding2==YES) -> outstanding2=NO; /*leave outstanding
unchanged*/
      :: (outstanding2==NO) -> outstanding=NO;
      fi;

   :: in?LMP_accepted_s ->
      if
      :: (state2==MAX) -> state2=NORMAL; SCO=SCO+1; outstanding=NO;
      fi;

   :: in?LMP_not_accepted_s ->
      if
      :: (state2==MAX) -> state2=NORMAL; outstanding=NO;
      fi;


   :: timeout;

   od;
}
```

```
init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.19.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 3605, errors: 0
   18533 states, stored
    3134 states, matched
   21667 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 42 (resolved)
(max size 2^19 states)

Stats on memory usage (in Megabytes):
0.890       equivalent memory usage for states (stored*(State-vector +
overhead))
0.717       actual memory usage for states (compression: 80.58%)
      State-vector as stored = 31 byte + 8 byte overhead
2.097       memory used for hash-table (-w19)
0.240       memory used for DFS stack (-m10000)
3.156       total actual memory usage

unreached in proctype device
      line 115, state 88, "-end-"
      (1 of 88 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.20 CONTROL OF MULTI-SLOT PACKETS

```
#define     LMP_accepted                  3
#define     LMP_not_accepted              4
#define LMP_max_slot1             45    /* name changed due to Promela
*/
#define LMP_max_slot_req          46

#define SLAVE  1
#define MASTER 0
```

```
#define YES    1
#define NO     0
#define MAX    1
#define NORMAL 0
#define MIN    -1
#define s_slot    20

#define CHAN_LEN  1   /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL, SCO=1;
   bit outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.20 Control of Multi-Slot packets */
/*************************************/
   ::
((len(out)==0)&&((outstanding==NO)&&((device_type==MASTER)&&(SCO>0))))
->
               out!LMP_max_slot1;
   :: in?LMP_max_slot1 -> skip;
/* assert that MASTER cannot receive LMP_max_slot1 */
   :: ((outstanding==NO)&&((device_type==SLAVE)&&(SCO>0))) ->
                  out!LMP_max_slot_req;
                  state=s_slot;
                  outstanding=YES;
   :: in?LMP_max_slot_req ->
      if
      :: out!LMP_accepted;
      :: out!LMP_not_accepted;
      fi;

/*************************************/
/* ACEPTED and NOT_ACCEPTED           */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_slot) -> state=NORMAL; outstanding=NO;
      fi;

   :: in?LMP_not_accepted ->
      if
      :: (state==s_slot) -> state=NORMAL; outstanding=NO;
      fi;

   :: timeout;

   od;
}

init
```

```
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

B.20.2 Spin output

```
(Spin Version 3.2.3 -- 1 August 1998)
      + Partial Order Reduction

Full statespace search for:
      never-claim             - (not selected)
      assertion violations    - (disabled by -A flag)
      cycle checks            - (disabled by -DSAFETY)
      invalid endstates +

State-vector 40 byte, depth reached 26, errors: 0
      37 states, stored
      30 states, matched
      67 transitions (= stored+matched)
       1 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)

2.542       memory usage (Mbyte)

unreached in proctype device
      line 59, state 30, "-end-"
      (1 of 30 states)
unreached in proctype :init:
      (0 of 4 states)
```

B.21 Single asynchronous model (functions 1-4)

```
#define LMP_name_req                  1
#define LMP_name_res                  2
#define     LMP_accepted                  3
#define     LMP_not_accepted              4
#define LMP_clkoffset_req             5
#define LMP_clkoffset_res             6
#define     LMP_detach                    7
#define LMP_in_rand                   8
#define LMP_comb_key                  9
#define LMP_unit_key                 10
#define LMP_au_rand                  11
#define     LMP_sres                     12
#define LMP_temp_rand                13
#define LMP_temp_key                 14
#define LMP_encryption_mode_req      15
#define LMP_encryption_key_size_req  16
```

```
#define LMP_start_encryption_req      17
#define LMP_stop_encryption_req       18
#define LMP_switch_req                19
#define LMP_hold1                     20
#define LMP_hold_req                  21
#define LMP_sniff1                    22
#define LMP_sniff_req                 23
#define LMP_unsniff_req               24
#define LMP_park_req                  25
#define LMP_park1                     26
#define LMP_set_broadcast_scan_window 27
#define LMP_modify_beacon             28
#define LMP_unpark_BD_ADDR_req        29
#define LMP_unpark_PM_ADDR_req        30
#define LMP_incr_power_req            31
#define LMP_decr_power_req            32
#define LMP_max_power              33
#define LMP_min_power              34
#define LMP_auto_rate              35
#define LMP_preferred_rate         36
#define LMP_version_req            37
#define LMP_version_res            38
#define LMP_features_req           39
#define LMP_features_res           40
#define LMP_quality_of_service     41
#define LMP_quality_of_service_req 42
#define LMP_SCO_link_req           43
#define LMP_remove_SCO_link_req    44
#define LMP_max_slot1              45
#define LMP_max_slot_req           46
#define LMP_timing_accuracy_req    47
#define LMP_timing_accuracy_res    48
#define LMP_setup_complete         49
#define LMP_use_semi_permanent_key 50
#define LMP_host_connection_req    51

#define SLAVE  1
#define MASTER 0
#define YES    1
#define NO     0
#define MAX    1
#define NORMAL 0
#define MIN    -1
#define TEMP1   7
/* use of previous indicates linkey==YES */
#define PREVIOUS 2
/* security (0=NO, 1=YES, 2=STAGE1, 3=STAGE2) YES=ENCRYPTION ON*/
#define STAGE1   2
#define STAGE2   3
#define DH     1
#define DM     0
#define s_auth     1
#define s_ukey     2
#define s_ckey     3
#define s_semi     4
#define s_neg_en   50  /* negotiation of encryption */
#define s_neg_size 51  /* negotiation of key size */
```

```
#define s_bg_en    52  /* begin encryption */
#define s_end_en   53  /* end encryption */
#define s_time     7
#define s_switch  10
#define s_detach  12
#define s_qos     18
#define s_sco_add 19
#define s_sco_rem 191
#define s_slot    20

#define CHAN_LEN  2  /* length of channel (number of messages to be
stored) */


proctype device (chan in, out; bit device_type, linkkey)
{
   byte state=NORMAL, power=NORMAL, SCO=0, linkey=linkkey;
   bit p_type, outstanding=NO, auth_done=NO;


BEGIN:
   do
/*************************************/
/* 3.2 Pairing                       */
/*************************************/
   :: (((outstanding==NO)&&(auth_done==NO))&&(state==NORMAL)) ->
      if
      :: (linkey==NO) -> out!LMP_in_rand; linkey=TEMP1;
      :: !(linkey==NO) -> skip;
      fi;
      out!LMP_au_rand;
      outstanding=YES;
      state=s_auth;

   :: in?LMP_in_rand ->
      if
      :: (state==s_detach) -> skip;
      :: !(state==s_detach) -> linkey=TEMP1;
      fi;

   :: in?LMP_au_rand ->
      if
      :: (state==s_detach) -> skip;
      :: !(state==s_detach) ->
         if
         :: (linkey==YES) -> out!LMP_sres;
         :: (linkey==TEMP1) -> out!LMP_sres;
         :: (linkey==NO) -> out!LMP_not_accepted;
         :: !((linkey==NO)||((linkey==YES)||(linkey==TEMP1))) -> skip;
         fi;
      fi;

   :: in?LMP_sres ->
      if
      :: (state==s_detach) -> skip;
      :: (state==s_auth) ->
         if
```

```
         :: auth_done=YES ->
            if
            :: (linkey==TEMP1) ->
               if
               :: out!LMP_unit_key; outstanding=YES; state=s_ukey;
               :: out!LMP_comb_key; outstanding=YES; state=s_ckey;
               fi;
            :: !(linkey==TEMP1) -> outstanding=NO; state=NORMAL;
            fi;
         :: auth_done=NO; outstanding=NO; state=NORMAL;
         :: out!LMP_detach; outstanding=YES; state=s_detach;
         fi;
      fi;

   :: in?LMP_unit_key ->
      if
      :: (state==s_detach) -> skip;
      :: (state==s_ukey) -> outstanding=NO; state=NORMAL;
      :: (state==s_ckey) -> outstanding=NO; state=NORMAL;
      :: !((state==s_ukey)||(state==s_ckey)||(state==s_detach)) ->
         if
         :: out!LMP_unit_key;
         :: out!LMP_comb_key;
         fi;
      fi;
      linkey=YES;

   :: in?LMP_comb_key ->
      if
      :: (state==s_detach) -> skip;
      :: (state==s_ukey) -> outstanding=NO; state=NORMAL;
      :: (state==s_ckey) -> outstanding=NO; state=NORMAL;
      :: !((state==s_ukey)||(state==s_ckey)||(state==s_detach)) ->
         if
         :: out!LMP_unit_key; printf("refuse to change link key\n");
         :: out!LMP_comb_key; printf("accept change of link key\n");
         fi;
      fi;
      linkey=YES;

/*************************************/
/* 3.3 Change Link Key               */
/*************************************/
   :: ((outstanding==NO)&&(auth_done==YES)) ->
         out!LMP_comb_key; outstanding=YES; state=s_ckey;

/*************************************/
/* 3.4 Change Current Link Key       */
/*************************************/
   :: ((len(out)==0)&&((outstanding==NO)&&(device_type==MASTER))) ->
         out!LMP_temp_rand;
         out!LMP_temp_key;

   :: in?LMP_temp_rand -> skip;

   :: in?LMP_temp_key ->
         linkey=TEMP1;
```

```
          printf("link key changed to temporary key\n");

   :: ((len(out)==0)&&((outstanding==NO)&&(device_type==MASTER))) ->
          out!LMP_use_semi_permanent_key;
          outstanding=YES;
          state=s_semi;

   :: in?LMP_use_semi_permanent_key ->
          out!LMP_accepted;
          linkey=PREVIOUS;


/*************************************/
/* ACEPTED and NOT_ACCEPTED          */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_semi) ->  state=NORMAL; outstanding=NO;
linkey=PREVIOUS;
      :: !(state==s_semi) -> skip;
      fi;

   :: in?LMP_not_accepted ->
      if
      :: (state==s_detach) -> skip;
      :: (state==s_auth) -> state=NORMAL; outstanding=NO;
      fi;

   :: in?LMP_detach -> break;

   :: timeout ->
      if
      :: (state==s_detach)&&(len(out)==0) -> break;
      :: (state==s_detach)&&!(len(out)==0) -> skip;
      :: !(state==s_detach) ->
          if
          :: (auth_done==YES) -> skip;
          :: (auth_done==NO) -> skip;
          fi;
      fi;
   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
           run device (AB, BA, MASTER,YES);
           run device (BA, AB, SLAVE,YES);
        }
}
```

B.22 Single synchronous model (functions 1-5)

```
/* Because there may be so many LMP_accepted or LMP_not_accepted messages
       received at roughly same time, currently there is no way for the protocol
       to differentiate them, the temporary solution is to make assumption,
       meaning that one time only sends one message until getting back the
```

```
       response for this message. But what about for other messages, for example,
       there is no response necessary for LMP_temp_rand and LMP_temp_key.
*/
/* authentication perform contains pairing perform */

#define LMP_name_req                                1
#define LMP_name_res                                2
#define LMP_accepted                                3
#define LMP_not_accepted                            4
#define LMP_clkoffset_req                           5
#define LMP_clkoffset_res                           6
#define LMP_detach                                          7
#define LMP_in_rand                                         8
#define LMP_comb_key                                9
#define LMP_unit_key                                10
#define LMP_au_rand                                         11
#define LMP_sres                                            12
#define LMP_temp_rand                               13
#define LMP_temp_key                                14
#define LMP_encryption_mode_req                     15
#define LMP_encryption_key_size_req          16
#define LMP_start_encryption_req             17
#define LMP_stop_encryption_req                     18
#define LMP_switch_req                              19
#define LMP_hold                                            20
#define LMP_hold_req                                21
#define LMP_sniff                                           22
#define LMP_sniff_req                               23
#define LMP_unsniff_req                                     24
#define LMP_park_req                                25
#define LMP_park                                            26
#define LMP_set_broadcast_scan_window 27
#define LMP_modify_beacon                           28
#define LMP_unpark_BD_ADDR_req                      29
#define LMP_unpark_PM_ADDR_req                      30
#define LMP_incr_power_req                          31
#define LMP_decr_power_req                          32
#define LMP_max_power                               33
#define LMP_min_power                               34
#define LMP_auto_rate                               35
#define LMP_preferred_rate                          36
#define LMP_version_req                                     37
#define LMP_version_res                                     38
#define LMP_features_req                            39
#define LMP_features_res                            40
#define LMP_quality_of_service                      41
#define LMP_quality_of_service_req           42
#define LMP_SCO_link_req                            43
#define LMP_remove_SCO_link_req                     44
#define LMP_max_slot                                45
#define LMP_max_slot_req                            46
#define LMP_timing_accuracy_req                     47
#define LMP_timing_accuracy_res                     48
#define LMP_setup_complete                          49
#define LMP_use_semi_permanent_key           50
#define LMP_host_connection_req                     51

#define yes                                         1
#define no                                          0
#define master                               1
#define slave                                0
#define combb_key                            1
#define unitt_key                            0
#define max_attempts                  2

byte auth_attempt1 = 0, auth_attempt2 = 0;
byte pair_attempt1 = 0, pair_attempt2 = 0;
byte key_send1 = 0, key_send2 = 0;
byte encryption_flag = 0;

bit outstanding = no;  /* run one after finishing one */
```

```
bit encrypted_broadcast = yes;
bit undone_flag = yes; /* make the semi-permanent key the current key */
bit change_any = yes;  /* change encryption mode, key or random number */

bit outstand1 = no, outstand2 = no;
bit auth_done1 = no, auth_done2 = no;
bit pair_done1 = yes, pair_done2 = yes;
bit result_key1 = combb_key, result_key2 = combb_key;
bit encryption1 = yes, encryption2 = yes;
bit temp_key1 = no, temp_key2 = no;
bit send_mode1 = no, send_mode2 = no;
bit agree_encryption1 = no, agree_encryption2 = no;
bit break_flag1 = no, break_flag2 = no;
bit change_key1 = no, change_key2 = no;

proctype device (chan in, out; byte auth_attempt, pair_attempt,     key_send;
        bit device_type, auth_done, pair_done, result_key, temp_key, encryption,
        send_mode, agree_encryption, change_key, outstand, break_flag)
{
        do
        :: atomic {
                auth_done == no && auth_attempt == 0
                && outstand == no && outstanding == no && break_flag == no ->
                        outstanding = yes;
                        outstand = yes;
                        out!LMP_au_rand;
                        auth_attempt = auth_attempt + 1;
                        outstanding = no
                }
        :: atomic {
                auth_done == yes
                && outstand == no && outstanding == no && break_flag == no ->
progress:       skip;
                        outstanding = yes;
                        if /* encryption */
                        :: device_type == master && encrypted_broadcast == yes ->
                                encrypted_broadcast = no;     /* change to temporary key */
                                temp_key = yes;
label1:                 encryption = yes;
                                out!LMP_temp_rand;
                                out!LMP_temp_key
                        :: temp_key == yes && encryption == yes && agree_encryption == no -
>
                                encryption = no;
                                outstand = yes;
                                out!LMP_encryption_mode_req;  /* encryption mode */
                                send_mode = yes
                        :: device_type == master && temp_key == yes && encryption_flag == 3
->
                                outstand = yes;
                                out!LMP_stop_encryption_req
                        /* change encryption mode, key or random number */
                        :: device_type == master && temp_key == yes
                                && change_any == yes && encryption_flag > 0 ->
                                change_any = no;
                                encryption_flag = 3;
                                outstand = yes;
                                out!LMP_stop_encryption_req
                        :: device_type == master && temp_key == yes
                                && change_any == no && encryption_flag == 0 ->
                                change_any = yes;
                                goto label1
                        /* make the semi-permanent key the current key */
                        :: device_type == master && temp_key == yes && undone_flag == yes -
>
                                outstand = yes;
                                if
                                :: encryption_flag > 0 ->
                                        encryption_flag = 3;
                                        out!LMP_stop_encryption_req
                                :: encryption_flag == 0 ->
```

```
                                        undone_flag = no;
                                        out!LMP_use_semi_permanent_key
                                fi
                        :: temp_key == no && change_key == no ->
                                change_key = yes;  /* change link key */
                                outstand = yes;
                                if
                                :: result_key == combb_key ->
                                        out!LMP_comb_key;
                                        key_send = 1
                                :: result_key == unitt_key ->
                                        out!LMP_in_rand;
                                        out!LMP_au_rand;
                                        pair_attempt = pair_attempt + 1
                                fi
                        :: else -> skip
                        fi;
                        outstanding = no
                }
        /* reset so as to restart the program */
        :: atomic {
                break_flag == yes ->
                        break_flag = no;
                        auth_attempt = 0;
                        pair_attempt = 0;
                        auth_done = no;
                        pair_done = yes;
                        result_key = combb_key
                }
        :: in?LMP_accepted ->
                outstand = no;
                if
                :: auth_done == yes && device_type == master && temp_key == yes ->
                        if
                        :: encryption_flag == 0 ->
                                if
                                /* master win, based on accepted for master encryption mode
request */
                                :: undone_flag == yes ->
                                        encryption_flag = 1;
                                        outstand = yes;
                                        out!LMP_encryption_key_size_req
                                :: undone_flag == no -> temp_key = no
                                fi
                        :: encryption_flag == 1 ->
                                encryption_flag = 2;
                                outstand = yes;
                                out!LMP_start_encryption_req
                        :: encryption_flag == 2 -> encryption_flag = 3
                        :: encryption_flag == 3 -> encryption_flag = 0
                        fi
                :: else -> skip
                fi
        :: atomic {
                in?LMP_not_accepted ->
                        outstand = no;
                        if
                        :: auth_done == no && pair_done == yes ->
                                outstanding = yes;
                                pair_done = no;
                                outstand = yes;
                                out!LMP_in_rand;
                                out!LMP_au_rand;
                                pair_attempt = pair_attempt + 1;
                                outstanding = no
                        :: auth_done == no && pair_done == no ->
                                break_flag = yes       /* pairing not allowed */
                        :: auth_done == yes -> change_key = no
                                /* give consideration to two or more things:
                                        change link key and encryption */
```

```
                        fi
                }
        :: in?LMP_detach -> break_flag = yes  /* authentication failure */
        :: in?LMP_in_rand -> skip
        :: in?LMP_comb_key ->
progress1:      skip;
                if
                :: key_send > 0 ->
                                outstand = no;
                                auth_done = yes;
                                pair_done = yes;
                                if
                                :: key_send == 1 -> result_key = combb_key
                                :: key_send > 1 -> result_key = unitt_key
                                fi;
                                change_key = no;  /* change link key for one time */
                                key_send = 0
                :: key_send == 0 ->
                        if
                        :: out!LMP_comb_key
                        :: out!LMP_unit_key
                        fi
                fi
        :: in?LMP_unit_key ->
progress2:      skip;
                if
                :: key_send > 0 ->
                                key_send = 0;
                                outstand = no;
                                auth_done = yes;
                                pair_done = yes;
                                result_key = unitt_key;
                                change_key = no
                :: key_send == 0 ->
                        if
                        :: out!LMP_comb_key
                        :: out!LMP_unit_key
                        fi
                fi
        :: in?LMP_au_rand ->
                if
                :: out!LMP_sres
                :: out!LMP_not_accepted
                fi
        :: atomic {
                in?LMP_sres ->
                        outstand = no;
                        outstanding = yes;
                        if
                        ::      if
                                :: auth_done == no && pair_done == yes ->
                                        auth_done = yes
                                :: auth_done == no && pair_done == no ->
                                        outstand = yes;
                                        if
                                        :: out!LMP_comb_key; key_send = 1
                                        :: out!LMP_unit_key; key_send = 2
                                        fi
                                :: auth_done == yes ->
                                        outstand = yes;
                                        out!LMP_comb_key;
                                        key_send = 1
                                fi
                        ::      if
                                :: auth_done == no && pair_done == yes ->
                                        if
                                        :: auth_attempt <= max_attempts ->
                                                outstand = yes;
                                                out!LMP_au_rand;
                                                auth_attempt = auth_attempt + 1
                                        :: auth_attempt > max_attempts ->
```

```
                                        break_flag = yes;
                                        out!LMP_detach
                                fi
                        :: (auth_done == no && pair_done == no) || auth_done == yes
->
                                /* also repeat for changing link key */
                                if
                                :: pair_attempt <= max_attempts ->
                                        outstand = yes;
                                        out!LMP_in_rand;
                                        out!LMP_au_rand;
                                        pair_attempt = pair_attempt + 1
                                :: pair_attempt > max_attempts ->
                                        break_flag = yes;
                                        out!LMP_detach
                                fi
                        fi
                fi;
                outstanding = no
        }
    :: in?LMP_temp_rand -> skip
    :: in?LMP_temp_key ->
        temp_key = yes;
        encryption = yes
    :: in?LMP_encryption_mode_req ->
        if
        :: out!LMP_accepted;
                agree_encryption = yes;
                if
                :: device_type == master && send_mode == no ->
                        encryption_flag = 1;
                        outstand = yes;
                        out!LMP_encryption_key_size_req
                :: else -> skip
                fi
        :: out!LMP_not_accepted
        fi
    :: in?LMP_encryption_key_size_req ->
        if
        :: device_type == master ->
                outstand = no;
                if
                :: out!LMP_accepted;
                        encryption_flag = 2;
                        out!LMP_start_encryption_req
                :: out!LMP_not_accepted;
                        encryption_flag = 0    /* unsuccessful negotiation */
                fi
        :: device_type == slave ->
                if
                :: out!LMP_accepted
                :: outstand = yes;
                        out!LMP_encryption_key_size_req
                fi
        fi
    :: in?LMP_start_encryption_req -> out!LMP_accepted
    :: in?LMP_stop_encryption_req -> out!LMP_accepted
    :: in?LMP_use_semi_permanent_key ->
        out!LMP_accepted;
        temp_key = no
    od
}

init
{       chan AB = [0] of {byte};
    chan BA = [0] of {byte};
    run device (BA, AB, auth_attempt1, pair_attempt1, key_send1, master,
            auth_done1, pair_done1, result_key1, temp_key1, encryption1,
            send_mode1, agree_encryption1, change_key1, outstand1, break_flag1);
        run device (AB, BA, auth_attempt2, pair_attempt2, key_send2, slave,
            auth_done2, pair_done2, result_key2, temp_key2, encryption2,
```

```
            send_mode2, agree_encryption2, change_key2, outstand2, break_flag2)
}
```

B.23 Single synchronous model (functions 6-11)

```
#define LMP_name_req                  1
#define LMP_name_res                  2
#define     LMP_accepted                  3
#define     LMP_not_accepted              4
#define LMP_clkoffset_req             5
#define LMP_clkoffset_res             6
#define     LMP_detach                    7
#define LMP_in_rand                   8
#define LMP_comb_key                  9
#define LMP_unit_key                 10
#define LMP_au_rand                  11
#define     LMP_sres                     12
#define LMP_temp_rand                13
#define LMP_temp_key                 14
#define LMP_encryption_mode_req      15
#define LMP_encryption_key_size_req  16
#define LMP_start_encryption_req     17
#define LMP_stop_encryption_req      18
#define LMP_switch_req               19
#define LMP_hold                     20
#define LMP_hold_req                 21
#define LMP_sniff                    22
#define LMP_sniff_req                23
#define LMP_unsniff_req              24
#define LMP_park_req                 25
#define LMP_park                     26
#define LMP_set_broadcast_scan_window 27
#define LMP_modify_beacon            28
#define LMP_unpark_BD_ADDR_req       29
#define LMP_unpark_PM_ADDR_req       30
#define LMP_incr_power_req           31
#define LMP_decr_power_req           32
#define LMP_max_power              33
#define LMP_min_power              34
#define LMP_auto_rate              35
#define LMP_preferred_rate         36
#define LMP_version_req            37
#define LMP_version_res            38
#define LMP_features_req           39
#define LMP_features_res           40
#define LMP_quality_of_service     41
#define LMP_quality_of_service_req 42
#define LMP_SCO_link_req           43
#define LMP_remove_SCO_link_req    44
#define LMP_max_slot               45
#define LMP_max_slot_req           46
#define LMP_timing_accuracy_req    47
#define LMP_timing_accuracy_res    48

#define SLAVE  1
#define MASTER 0
```

```
#define YES    1
#define NO     0
#define MAX    1
#define NORMAL 0
#define MIN    -1
#define DH     1
#define DM     0
#define s_time     7
#define s_switch  10
#define s_qos     18
#define s_sco_add 19
#define s_sco_rem 191
#define s_slot    20

#define CHAN_LEN  2    /* length of channel (number of messages to be
stored */


proctype device (chan in, out; bit device_type)
{
   byte state=NORMAL, power=NORMAL, SCO=0;
   bit p_type, outstanding=NO;


BEGIN:
   do
/*************************************/
/* 3.6 Clock Offset Request          */
/*************************************/
   :: ((outstanding==NO)&&(device_type==MASTER)) ->
                 out!LMP_clkoffset_req;
                 outstanding=YES;
   :: in?LMP_clkoffset_req -> out!LMP_clkoffset_res;
   :: in?LMP_clkoffset_res -> outstanding=NO;
               /*assert only MASTER can receive */

/*****************************************/
/* 3.7 Timing Accuracy Information Request */
/*****************************************/
   :: (outstanding==NO) ->
                out!LMP_timing_accuracy_req;
                outstanding=YES;
                state=s_time;
   :: in?LMP_timing_accuracy_req ->
        if
        :: out!LMP_timing_accuracy_res;
        :: out!LMP_not_accepted;
        fi;
   :: in?LMP_timing_accuracy_res -> outstanding=NO;

/*************************************/
/* 3.8 LMP Version                   */
/*************************************/
   :: (outstanding==NO) ->
                out!LMP_version_req;
                outstanding=YES;
   :: in?LMP_version_req -> out!LMP_version_res;
```

```
   :: in?LMP_version_res -> outstanding=NO;

/*************************************/
/* 3.9 Supported Features            */
/*************************************/
   :: (outstanding==NO) ->
                out!LMP_features_req;
                outstanding=YES;
   :: in?LMP_features_req -> out!LMP_features_res;
   :: in?LMP_features_res -> outstanding=NO;

/*************************************/
/* 3.10 Switch of Master/Slave Role  */
/*************************************/
   :: (outstanding==NO) ->
                out!LMP_switch_req;
                state=s_switch;
                outstanding=YES;
   :: in?LMP_switch_req ->
      if
      :: out!LMP_accepted;
         if
         :: (device_type==SLAVE) -> device_type=MASTER;
         :: (device_type==MASTER) -> device_type=SLAVE;
         fi;
      :: out!LMP_not_accepted;
      fi;

/*************************************/
/* 3.11 Name request                 */
/*************************************/
   :: (outstanding==NO) ->
                out!LMP_name_req;
                outstanding=YES;
   :: in?LMP_name_req -> out!LMP_name_res;
   :: in?LMP_name_res -> outstanding=NO;

/*************************************/
/* ACEPTED and NOT_ACCEPTED          */
/*************************************/
   :: in?LMP_accepted ->
      if
      :: (state==s_switch) ->
         if
         :: (device_type==SLAVE) -> device_type=MASTER;
         :: (device_type==MASTER) -> device_type=SLAVE;
         fi;
         state=NORMAL;
         outstanding=NO;
      fi;

   :: in?LMP_not_accepted ->
      if
      :: (state==s_time) -> state=NORMAL; outstanding=NO;
      :: (state==s_switch) -> state=NORMAL; outstanding=NO;
      fi;
```

```
   :: timeout;

   od;
}

init
{     chan AB = [CHAN_LEN] of {byte};
      chan BA = [CHAN_LEN] of {byte};
      atomic {
         run device (AB, BA, MASTER);
       run device (BA, AB, SLAVE);
        }
}
```

C ANNEX C
Comments

This annex includes the copies of the comments submitted to the Bluetooth members webpage.

C.1 Comments on Version 0.7

Comments on Bluetooth Project Version 0.7, Part C: Link Manager Protocol
Submitted: January, 1999

Modeling Assumptions

1. Since 1.1, last paragraph states that there is no need to explicitly acknowledge the messages in LMP, we will assume no loss or out-of-sequence packets.

Technical comments

1. Section 3.1.4(Repeated Attempts): a maximum value is hinted to, but no value is specified in this section. One should be defined. The period always increases after a failure. No mention here about how to decrease this period. {Part B 14.4.1 made mention of decreasing it, but gave no specifics.}
2. Section 3.5.2, sequence 15; shows an unsuccessful negotiation of the encryption key by sending a LNP_not_accepted, there is no reason given and a reason is required by that message. What should it be? #6: not supported?
3. Section 3.6; talks about receiving a FHS packet, upon which the slave responds. Yet the Sequence 18 shows sending and receiving LMP_clkoffset_req and LMP_clkoffset_res. [Investigate further on the inclusion of these PDUs in FHS packets. {Part B 4.4.1.5: The FHS has no payload for this LMP PDU. Therefore, it must be sent using another packet type (i.e. DM1)}]. Is it to be assumed that there is a one-to-one mapping with the generation of LMP PDU with FHS packet?
4. Section 3.7, Timing accuracy information request, Sequence 20; Show that the requested device does not support timing accuracy information. What does this mean? Since these PDUs are optional, this device may not recognize this PDU because it does not implement or support it and respond with LMP_not_accepted (#4 unknown PDU) as per section 3.21. Yet the sequence appears to show the case where if requested for such information, that is the requested device recognizes the PDU, it must then respond with LMP_not_accepted (#6 not supported). Even though the PDU is the same the reason is not. This should be clarified.
5. Section 3.9 (Supported Features); It should be possible to know which features are supported and therefore should be tested. However, there are no procedures defined for any mismatch, but text now states that this cannot happen. Information about which are the supported services was not found in reference [1] as stated.
6. Section 3.10; states that if a device wants to join an existing piconet, it must respond to the LMP_switch_req with a LMP_accepted. However if the device does not support the master-slave switch it will send a LMP_not_accepted. How can this be possible? If the device wants to join an existing piconet, it knows it must respond

positively to the LMP_switch_req. [Where is the indication that the new device wants to join the existing piconet? {Part B}]

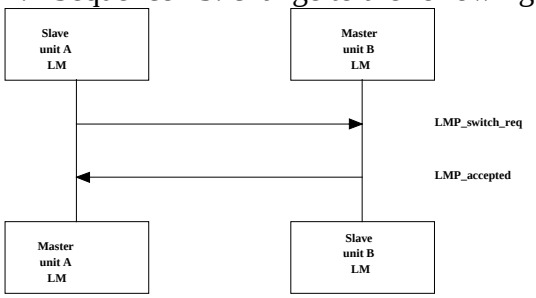
7. Section 3.13.2; contains no text describing the behavior, only the Sequence 28 is shown. Text should indicate that whether or not the LMP_hold exchange must occur because just reading the other sections on Hold Mode does not indicate that the Master must accept and acknowledge the Slave's LMP_hold.
8. Section 3.14; What are the two options hinted to for the calculation of the first sniff slot? {Part B 10.8.2}
9. Section 3.14.3; The title indicates that the slave is being moved from sniff mode to active mode, yet the text and sequence 32 indicates that initiating (either master or slave) can do this. Also Table 5.1 supports the latter, so most likely the title must be replaced with "Moving a device from sniff mode to active mode".
10. Section 3.15, third paragraph; The sentence, "When a slave is placed in park mode it is assigned a unique PM_ADDR, which can be used by the master to unpark slaves", is confusing. Is the PM_ADDR unique? If so then the sentence should end "… by the master to unpark that slave". If the PM_ADDR can be used to refer to a group of slaves, then the word, unique, should be removed. If both are true, which seems to be what section 3.15.6 indicates, then better text needs to be written to clarify this.
11. Section 3.15.2; There is no text to explain the sequence 34, which appears to be misleading. Is the sequence 34 correct? If so, what is the purpose of sending the LMP_park_req, when sending the LMP_park will do? OR the sequence 34 is wrong and the LMP_park should be removed. However in this case the parameters are not passed, which causes more errors. What is the purpose of the master allowing the slave to accept or reject being park? If the slave rejects being placed in park, can the master force it? It seems like LMP_park_req PDU is really only for the slave.
12. Section 3.15.6, second paragraph; the text and related messages do not appear to align. Also Table 5.1 adds to the confusion (e.g. which PM or BD is in which message and how many).
13. Section 3.17 (Channel quality driven change between DM and DH); If this is an optional feature, then how can sequence 46 be accomplished? If the left-hand device does not understand the message, then what?
14. Table 3.18: need to add M/O column with both PDUs marked as mandatory, since text indicates that the slave cannot reject the notification, which is could do if the PDUs were optional.
15. Table 5.1, page 205, modify and unpark PDUs are variable. How is the PDU to be parsed? For the LMP_unpark_BD_ADDR_req and LMP_unpark_PM_ADDR_req, how does one know how many AM_ADDR are present?
16. For the LMP_modify_beacon, it appears that the M_{access} and the access scheme share one byte, but Table 5.2 states that M_{access} is 4bits and access scheme is 8bits. Therefore this will not work.
17. For the LMP_unpark_BD_ADDR_req and LMP_unpark_PM_ADDR_req, it appears that two AM_ADDR can share one byte, but Table 5.2 states that AM_ADDR is 8bits. Therefore this will not work. {PART B states that AM_ADDR is 3bits}
18. For the LMP_unpark_PM_ADDR_req, it appears that the BD_ADDR uses only one byte, but Table 5.2 shows 6 bytes. This could be related to another mistake in that the

BD_ADDR should be PM_ADDR. {Part B states that the PM_ADDR is 8 bits and BD_ADDR is 48 bits}

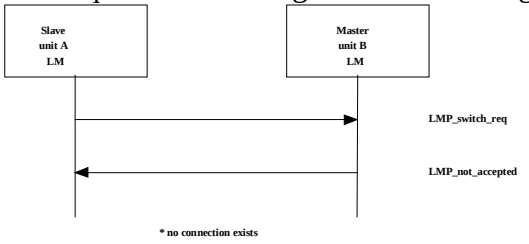
19. Table 5.2, Name=reason row; in the detailed column the values, 0 (no error) and 5 (repeated attempts) are not used in Part C and should be removed or have the missing procedures added which could use this value. The 0 (no error) is not consistent with how the reason value is used throughout this section. The reason value is only included in the LMP_not_accepted PDU and gives the reason for not accepting. Therefore 0 (no error) is right out.
20. Table 5.3; lists coding features.
    I) The optional timing accuracy information request (3.7) does not have a code point.
    II) Hold mode is not optional, so why does it have a code point?
    III) The optional channel quality (3.17) does not have a code point.
    IV) According to Table 5.4; the SCO packet's default value is HV3, so why does it have a code point?
    V) According to Table 5.4; the air mode's default value is CVSD, so why does it have a code point?
21. Table 6.1; needs the M/O column added with both LMP PDUs marked as M (Mandatory), since the text states that the link manager must be able to receive this message anytime.

Editorials

1. Section 3.2.1: change first "but" to "and"
2. Section 3.2.3, first paragraph: change "has to" to "must"
3. Section 3.8: change "confirm" to "conform"
4. Sequence 23: change to the following for clarification.



5. Sequence 24: change to the following for clarification.



6. Section 3.13.1, sequence 27; add caption, "Master forces slave into hold mode"
7. Section 3.13.1, sequence 28; add caption, "Slave forces master into hold mode"
8. Section 3.14.1, sequence 30; add caption, "Master forces slave into sniff mode"

9. Section 3.14.3, sequence 32, add caption, “TBD based on outcome of technical comment”
10. Section 3.19.2; add colon: “…, but the parameters**:** timing control flags and D_{SCO} are …”
11. Section 3.19.2, sequences 53 and 54 have the devices master (right) and slave (left) which is the opposite of all previous ones. Suggest switching them to be consistent.
12. Section 3.19.2, sequence 53; add caption, “Master rejects slave’s request for a SCO link”
13. Section 3.19.2, sequence 54; add caption, “Master accepts slave’s request for a SCO link”

Date: 1/11/99

Alignments

The following need to be corrected one way or the other.

1. Section 3.18, Table 3.18, LMP_quality_of_service_req: has only one parameter, poll interval, however, the Table 5.1 (page 206) show an additional parameter, N_{BC}.
2. Table 5.2
   i. hold time: Length column is 1 byte, while Type column is u_int16. Table 5.1 indicates two (2) bytes.
   ii. broadcast scan window: table indicates 1 byte, while Table 5.1 indicates two (2) bytes.

Editorials

1. Section 3.3: change "Kinit" to "K_{init}"
2. Section 3.4.2: change "… key been changed …" to "… key has been changed …"
3. Section 3.9; 4th line: change "…send any other than ID, …" to "… send any packets other than ID, …
4. Section 3.16, 1st paragraph, last line: change "effects" to "effect"
5. Section 4, last paragraph; LMP_setup_complete PDU is mentioned, but no Table is provided stating its mandatory or optional status. It is assumed that it is mandatory, since the text states that a unit will send this PDU.
6. Section 4, last paragraph, 2nd line: change "request" to "requests"

January 14, 1999

Technical

3.17 Channel Quality Driven Change Between DM and DH: This is a continuation of the technical comments #13 and #20 III above. This section describes optional procedures for changing between DM and DH. What exactly does this mean? The first sentence states that a device is configured to always use DM packets or to always use DH packets or to automatically adjust its packet type to the quality of the channel. Based on this information I get the following.

Assumption: A) Only a device configured to automatically change mandatorily supports these procedures. Devices that support only DM or only DH do not support these procedures.

1) Since there is no way of knowing (i.e. through Bluetooth features) whether a device supports this feature or not, a device that does support this feature can send either message to an unknown device.
    i) The LMP_auto_rate is sent only to notify the other device that it supports these features. (In this case technical comment #20 III is void)
        a) If the receiving device does not support this feature, it
            I) does nothing (meaning the other side has learnt nothing about whether the other device supports this feature)
            II) sends back a LMP_error (meaning that it does not support this message. No change is possible)
        b) If the receiving device supports this feature, it
            I) does nothing (meaning the sender did not discover whether the other device supports this feature, but receiver knows the senders capability.)
            II) sends back a LMP_accepted (meaning that this device understands the message and supports the feature)
            III) sends back an LMP_auto_rate (meaning that the receiving device also supports this feature. Both devices know that the other supports this feature.)
            III) sends back a LMP_preferred_rate (meaning that the device supports the feature and wants to change the current packet)
                1. What does the data rate value of Medium or High mean in this returned PDU, since it is an indication to switch between DM and DH? Does Medium equal DM or DH? Why not change data rate values to DM or DH, instead of Medium or High?
    ii) The LMP_auto_rate is sent to tell the other device that it will be switching between DM and DH packets (i.e. a toggle message)
        a) If the receiving device does not support this feature, it

I) does nothing (meaning the other side continues to send current packet type and will be receiving a different packet type, which is may not be able to process)

II) sends back a LMP_error (meaning that it does not support this message. No change is possible. Sender shall not switch packet type.)

b) If the receiving device supports this feature, it

I) does nothing (meaning

1. it accepts the toggle
2. it has determined that there is no need to change (i.e. not send an LMP_preferred_rate).

II) can send back an LMP_preferred_rate (meaning that the device supports the feature and wants to change current packet type)

1. If LMP_auto_rate is a toggle, what is the need for this PDU?

iii) The LMP_auto_rate is sent to suggest to the other device to do a test to change packet type.

a) If the receiving device does not support this feature, it

I) does nothing (unknown meaning)

II) sends an LMP_error

III) sends an LMP_not_accepted [not supported]

b) If the receiving device supports this feature, it

I) does nothing (meaning it does not want to change or it has determined that there is no need to change)

II) sends an LMP_preferred_rate

iv) The LMP_preferred_rate is sent to order the other device to switch between DM and DH packets (i.e. a toggle message)

a) If the receiving device does not support this feature, it

I) does nothing (meaning it ignores the order to switch from current packet type)

II) sends back a LMP_error (meaning that it does not support this message. No change is possible. Neither sender or receiver shall switch packet type.)

III) sends back a LMP_not_accepted (meaning that device does not want to change.)

b) If the receiving device supports this feature, it

I) sends back a LMP_accepted (meaning it accepts the change.)

II) Sends nothing (meaning that it accepts the change.)

III) sends back a LMP_preferred_rate with opposite data value. (meaning that the device supports the feature, but does not want to change.)

January 19, 1999

Technical

1) Section 3.3 (Change Link Key): This section states that this procedure is the same as 3.2.3, except for the current link key value. However this is not true. If the current key is a derived key from combination keys, then sending of a LMP_unit_key is not possible, since it had initially sent a LMP_comb_key. Otherwise this procedure would not apply. Assuming that the previous statement is wrong. When the unit receives the LMP_unit_key, it will change its link key to unit key because only one unit sent an LMP_unit_key, as per 3.2.3.
2) Section 3.15 (Park Mode) State information must be maintained for sequence 34. Otherwise the receipt of the LMP_accepted cannot invoke an LMP_park.

C.2 Comments on Version 0.8

Comments on Bluetooth Project Version 0.8, Part C: Link Manager Protocol
Submitted: March 3, 1999

Date: February 9, 1999
Comments on Bluetooth Project Version 0.8, Part C: Link Manager Protocol

Technical comments

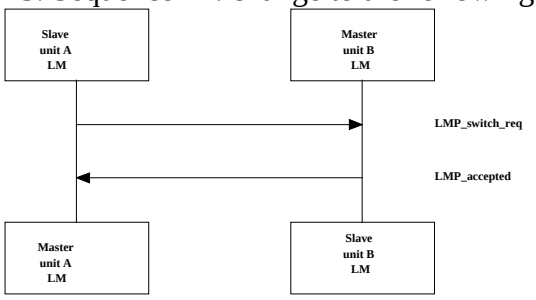
22. Section 3.1.3(Repeated Attempts): a maximum value is hinted to, but no value is specified in this section. One should be defined. The period always increases after a failure. No mention here about how to decrease this period. {related Part B 14.4.1}
23. Section 3.3 (Change Link Key): This section states that this procedure is the same as 3.2.3, except for the current link key value is used instead of K_{init}. However this is not true. If the current key is a derived key from combination keys, then sending of a LMP_unit_key is not possible, since it had initially sent a LMP_comb_key. Otherwise this procedure would not apply. Assuming that the previous statement is wrong. When the unit receives the LMP_unit_key, it will change its link key to unit key because only one unit sent an LMP_unit_key, as per 3.2.3. Therefore Sequence 7 does not indicate that change of link key is not possible, but instead indicates that the key has changed to the value in the LMP_unit_key. Clarification is needed.
24. Section 3.5.2, sequence 13; shows an unsuccessful negotiation of the encrytion key by sending a LMP_not_accepted, there is no reason given and a reason is required by that message. What should it be? #6: not supported?
25. Section 3.9 (Supported Features); It should be possible to know which features are supported and therefore should be tested. However, there are no procedures defined for any mismatch, but text now states that this cannot happen. Information about which are the supported services was not found in reference [1] as stated. Reference should be removed or information should be provided at said reference point.
26. Section 3.13.2; contains no text describing the behavior, only the Sequence 26 is shown. Text should indicate that whether or not the LMP_hold exchange must occur because just reading the other sections on Hold Mode does not indicate that the Master must accept and acknowledge the Slave's LMP_hold.
27. Section 3.15 2; State information must be maintained for sequence 32. Otherwise the receipt of the LMP_accepted cannot invoke an LMP_park.
28. Section 3.17 (Channel quality driven change between DM and DH); If this is an optional feature, then how can sequence 44 be accomplished? If the left-hand device does not understand the message, then what? Can the device send LMP_not_accepted (reason: unknown PDU)?
29. Table 5.1, LMP PDU=LMP_in_rand, possible direction is only M->S, however, the change from v0.7 to 0.8 made this M<->S.
30. Table 5.2, Name=reason row; in the detailed column the values, 0 (no error) and 5 (repeated attempts) are not used in Part C and should be removed or have the missing procedures added which could use this value. The 0 (no error) is not consistent with how the reason value is used throughout this section. The reason value is only

included in the LMP_not_accepted PDU and gives the reason for not accepting. Therefore 0 (no error) is right out.

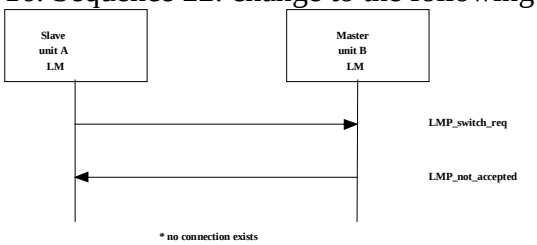
31. Table 5.3; lists coding features.
    - VI) The optional timing accuracy information request (3.7) does not have a code point.
    - VII) Hold mode is not optional, so why does it have a code point?
    - VIII) The optional channel quality (3.17) does not have a code point.
    - IX) According to Table 5.4; the SCO packet's default value is HV3, so why does it have a code point?
    - X) According to Table 5.4; the air mode's default value is CVSD, so why does it have a code point?
32. Table 6.1; needs the M/O column added with both LMP PDUs marked as M (Mandatory), since the text states that the link manager must be able to receive these messages anytime.

Editorials

14. Sequence 1; has an extraneous arrow, which should be deleted from the verifier.
15. Sequence 21: change to the following for clarification.



16. Sequence 22: change to the following for clarification.



17. Section 3.11, sequence 23; add caption, "Device's name requested and it response"
18. Section 3.12, sequence 24; add caption, "Connection closed by sending an LMP_detach"
19. Section 3.13.1, sequence 25; add caption, "Master forces slave into hold mode"
20. Section 3.13.1, sequence 26; add caption, "Slave forces master into hold mode"
21. Section 3.14.1, sequence 28; add caption, "Master forces slave into sniff mode"
22. Section 3.14.3, sequence 30, add caption, "Slave moved from sniff mode to active mode"
23. Section 3.15.1, sequence 31, add caption, "Slave forced into park mode
24. Section 3.15.4, sequence 36, add caption, "Master notifies all slaves of increase in broadcast capacity"

25. Section 3.19.1, sequence 50; add caption, “Master requests an SCO link”
26. Section 3.19.2, sequences 51 and 52 have the devices master (right) and slave (left) which is the opposite of all previous ones. Suggest switching them to be consistent.
27. Section 3.19.2, sequence 51; add caption, “Master rejects slave’s request for a SCO link”
28. Section 3.19.2, sequence 52; add caption, “Master accepts slave’s request for a SCO link”
29. Section 3.19.5, sequence 53; add caption, “SCO link removed”
30. Both the LMP_accepted and LMP_not_accepted were deleted from the Table in version 0.7, now there is no table indicating their M/O or contents. Create a new table and section with these two PDUs stating that they are mandatory and their content. The accompanying text should state that these PDUs are used as responses to other PDUs as stated elsewhere in the document.

Technical

3.17 Channel Quality Driven Change Between DM and DH: This is a continuation of the technical comments #7 and #10 III above. This section describes optional procedures for changing between DM and DH. What exactly does this mean? The first sentence states that a device is configured to always use DM packets or to always use DH packets or to automatically adjust its packet type to the quality of the channel. Based on this information I get the following.

Assumption: A) Only a device configured to automatically change mandatorily supports these procedures. Devices that support only DM or only DH do not support these procedures.

2) Since there is no way of knowing (i.e. through Bluetooth features) whether a device supports this feature or not, a device that does support this feature can send either message to an unknown device.
    - v) The LMP_auto_rate is sent only to notify the other device that it supports these features. (In this case technical comment #10 III is void)
        - c) If the receiving device does not support this feature, it
            - IV) does nothing (meaning the other side has learnt nothing about whether the other device supports this feature)
            - V) sends back an LMP_not_accepted (meaning that it does not support this message. No change is possible)
        - d) If the receiving device supports this feature, it
            - IV) does nothing (meaning the sender did not discover whether the other device supports this feature, but receiver knows the sender’s capability.)
            - V) sends back a LMP_accepted (meaning that this device understands the message and supports the feature), however this is not stated as possible behavior in the specification.

VI) sends back an LMP_auto_rate (meaning that the receiving device also supports this feature. Not directly specified, but a possible outcome. Both devices know that the other supports this feature.)

VI) sends back a LMP_preferred_rate (meaning that the device supports the feature and wants to change the current packet)

2. What does the data rate value of Medium or High mean in this returned PDU, since it is an indication to switch between DM and DH? Does Medium equal DM or DH? Why not change data rate values to DM or DH, instead of Medium or High?

vi) The LMP_auto_rate is sent to tell the other device that it will be switching between DM and DH packets (i.e. a toggle message)

c) If the receiving device does not support this feature, it

III) does nothing (meaning the other side continues to send current packet type and will be receiving a different packet type, which it may not be able to process)

IV) sends back a LMP_not_accepted (meaning that it does not support this message. No change is possible. Sender shall not switch packet type.)

d) If the receiving device supports this feature, it

III) does nothing (meaning

3. it accepts the toggle
4. it has determined that there is no need to change (i.e. not send an LMP_preferred_rate).

IV) can send back an LMP_preferred_rate (meaning that the device supports the feature and wants to change current packet type)

2. If LMP_auto_rate is a toggle, what is the need for this PDU?

vii) The LMP_auto_rate is sent to suggest to the other device to do a test to change packet type.

c) If the receiving device does not support this feature, it

IV) does nothing (unknown meaning)

V) sends an LMP_not_accepted [not supported]

d) If the receiving device supports this feature, it

III) does nothing (meaning it does not want to change or it has determined that there is no need to change)

IV) sends an LMP_preferred_rate

viii) The LMP_preferred_rate is sent to order the other device to switch between DM and DH packets (i.e. a toggle message)

c) If the receiving device does not support this feature, it

IV) does nothing (meaning it ignores the order to switch from current packet type)

V) sends back a LMP_not_accepted (meaning that device does not want to change.)

d) If the receiving device supports this feature, it

IV) sends back a LMP_accepted (meaning it accepts the change.)

V) Sends nothing (meaning that it accepts the change.)

VI) sends back a LMP_preferred_rate with opposite data value. (meaning that the device supports the feature, but does not want to change.)

Assuming no error retransmissions, would the LMP_auto_rate PDU be sent more than once on a link by a supporting/implementing device? Based on the above outcomes, I think the answer is no. The LMP_auto_rate is sent at most once per link (between the master and a slave) by a supporting/implementing device. The LMP_preferred_rate can be sent only after receiving an LMP_auto_rate. After receiving an LMP_auto_rate, the LMP_preferred_rate can be sent any time a change is determined to be advantageous.