NCF – Node Capability File
LDF – LIN Description File

Segédlet

Összeállította: Tóth Csaba

Csak belső használatra!

Kézirat

Budapest, 2021.

Példa	Node Capability File (NCF) definíció (vázlatosan)
node_capability_file; **LIN_language_version** = "2.1";	**node_capability_file** ; **LIN_language_version =** char_string ; [<node_definition>]
node step_motor {	<node_definition> ::= **node** <node_name> **{** <general_definition> <diagnostic_definition> <frame_definition> <encoding_definition> <status_management> (<free_text_definition>) **}**

general {
LIN_protocol_version = "2.1";
supplier = 0x0005;
function = 0x0020;
variant = 1;
bitrate = **automatic min** 10 **kbps max** 20 **kbps**;
sends_wake_up_signal = "yes";
// A slave képes wake up signalt küldeni.
}
	<general_definition> ::=
	general {
	LIN_protocol_version = <protocol_version> ;
	
	supplier = <supplier_id> ;
	function = <function_id> ;
	variant = <variant_id> ;
	bitrate = <bitrate_definition> ;
	sends_wake_up_signal = "yes" | **"no"** ;
	}
	
	<bitrate_definition> ::=
	automatic (**min** <bitrate>) (**max** <bitrate>) |
	select {<bitrate> [, <bitrate>]**}** |
	<bitrate>
	
	<bitrate> ::= 1 to 20 **kbps**

diagnostic {
NAD = 1 **to** 3;
diagnostic_class = 2; // 1, 2, 3
P2_min = 100 **ms**;
// Request-response timeout: 50 – 500 ms.
ST_min = 40 **ms**; // Default 0 ms
// Két egymás utáni keret közötti minimális idő a slave-ben.
	<diagnostic_definition> ::=
	diagnostic {
	NAD = integer ([, integer]) ; | (integer **to** integer) ;
	diagnostic_class = integer ;
	(**P2_min =** real_or_integer **ms** ;)
	(**ST_min =** real_or_integer **ms** ;)
	(**N_As_timeout =** real_or_integer **ms** ;)
	(**N_Cr_timeout =** real_or_integer **ms** ;)
	(**support_sid {** integer ([, integer]) **}** ;)

```
  support_sid { 0xB0, 0xB2, 0xB7 };
    // Ezeket a szolgálatokat (node configuration,
identification, diagnostic services) ismeri a slave.
Defaults: 0xB2 (Read by Identifier), 0xB7 (Assign frame
identifier range), a többi opcionális vagy elavult (pl.
0xB0=Assign NAD, opcionális).
  }
```

```
  (max_message_length = integer ;) // Csak a diagnosztikai
üzenetekre vonatkozik. Default: 4095.)
  }
```

```
frames {




publish node_status {

   length = 4; // Keretbeli adathossz = 1...8 bájt.
   min_period = 10 ms;
   max_period = 100 ms;



signals {
  state
    {size = 8; init_value = 0; offset = 0;} // bits
  fault_state
    {size = 2; init_value = 0; offset = 9; fault_enc;}
  error_bit
    {size = 1; init_value = 0; offset = 8;}
  angle
    {size = 16; init_value = {0x22, 0x11};
     offset = 16;}
}
}
```

```
<frame_definition> ::=
frames { [<single_frame>] }

<single_frame> ::= <frame_kind> <frame_name> {
<frame_properties> (<signal_definition>) }

<frame_kind> ::= publish | subscribe
<frame_properties> ::=
length = integer ;
(min_period = integer ms ;)
(max_period = integer ms ;)
(event_triggered_frame = identifier;)

<signal_definition> ::=
signals {
[<signal_name> { <signal_properties> }] }

<signal_properties> ::= <init_value> size = integer ;
offset = integer ; (<encoding_name> ;)

<init_value> ::= <init_value_scalar> | init_value_array>
<init_value_scalar> ::= init_value = integer
<init_value_array> ::= init_value = {integer ([, integer
])}
```

```
subscribe control {
  length = 1;
  max_period = 100 ms;

signals {
  command
    {size = 8; init_value = 0; offset = 0; position;}
```

} } }	

encoding {
position
{**physical_value** 0, 199, 1.8, 0, "deg";}
// min, max, scale, offset, text
// physical_value = (scale * raw_value) + offset
 	
fault_enc
{**logical_value**, 0, "no result";
logical_value, 1, "failed";
logical_value, 2, "passed";}
}
	<encoding_definition> ::=
	encoding { [<encoding_name> **{** [<logical_value> |
	<physical_range> | <bcd_value> | <ascii_value>] **}**]
	}
	
	<logical_value> ::=
	logical_value, <signal_value> (, <text_info>) ;
	<physical_range> ::=
	physical_value, <min_value>, <max_value>, <scale>,
	<offset> (, <text_info>) ;
	<bcd_value> ::= **bcd_value** ;
	<ascii_value> ::= **ascii_value** ;
	<signal_value> ::= integer // 0...65535
	<min_value> ::= integer // 0...65535
	<max_value> ::= integer // 0...65535
	<scale> ::= real_or_integer
	<offset> ::= real_or_integer
	<text_info> ::= char_string

status_management {
response_error = error_bit;
fault_state_signals = fault_state; }
 	
free_text
{ "step_motor signal values outside 0 - 199 are ignored" }
}
	<status_management> ::=
	status_management {
	response_error = identifier ;
	(**fault_state_signals =** identifier ([, identifier]) ;)
	}
	<free_text_definition> ::= **free_text {** char_string **}**

Példa	LIN Description File (LDF) definíció (vázlatosan)

LIN_description_file;
LIN_protocol_version = "2.1";
LIN_language_version = "2.1";
LIN_speed = 19.2 kbps;
Channel_name = "DB";
// Postfix for all named objects in the LDF.
// Cluster azonosító, több cluster is lehet.
Nodes {
Master: CEM, 5 **ms**, 0.1 **ms**;
// node_name, time_base, jitter
Slaves: LSM, RSM;
// node_names
}
 	
Node_attributes {
LSM {
LIN_protocol = "2.1";
configured_NAD = 0x20;
initial_NAD = 0x01;
product_id = 0x4A4F, 0x4841;
response_error = LSMerror;
fault_state_signals = IntTest;
P2_min = 150 ms;
ST_min = 50 ms;
configurable_frames { // 2.1 verzió szerint
CEM_Frm1;
LSM_Frm1;
LSM_Frm2;}
}
 	
RSM {
LIN_protocol = "2.0"; // Korábbi változat
configured_NAD = 0x20;
product_id = 0x4E4E, 0x4553, 1;
response_error = RSMerror;
P2_min = 150 ms;
ST_min = 50 ms;
configurable_frames { // 2.0 verzió szerint
CEM_Frm1 = 0x0001;
LSM_Frm1 = 0x0002;
LSM_Frm2 = 0x0003;} }
	<LIN_description_file> ::=
	LIN_description_file ;
	<LIN_protocol_version_def>
	<LIN_language_version_def>
	<LIN_speed_def>
	(<Channel_name_def>)
	
	<node_def> ::=
	Nodes {
	Master: <node_name>, <time_base> **ms**, <jitter> **ms** ;
	Slaves: <node_name>([, <node_name>]) ;
	}
	
	<node_attributes_def> ::=
	Node_attributes {
	[<node_name> **{**
	LIN_protocol = <protocol_version> ;
	configured_NAD = <diag_address> ;
	(**initial_NAD =** <diag_address> ;)
	<attributes_def> ;
	}]
	}
	
	<attributes_def> ::=
	product_id = <supplier_id>, <function_id> (, <variant>) ;
	response_error = <signal_name> ;
	(**fault_state_signals =** <signal_name>([, <signal_name>]) ;)
	(**P2_min =** real_or_interger **ms** ;)
	(**ST_min =** real_or_interger **ms** ;)
	(**N_As_timeout =** real_or_integer **ms** ;)
	(**N_Cr_timeout =** real_or_integer **ms** ;)
	<configurable_frames_20_def> | <configurable_frames_21_def>

```
  }
Signals {
  IntLightsReq:          2, 0, CEM, LSM, RSM;
  RightIntLightsSwitch: 8, 0, RSM, CEM;
  LeftIntlLightsSwitch: 8, 0, LSM, CEM;
  LSMerror,              1, 0, LSM, CEM;
  RSMerror,              1, 0, RSM, CEM;
  IntTest,               2, 0, LSM, CEM;
}
// name, size, init_value, published_by, subscribed_by
```

```
<Signal_def>
(<Diag_signal_def>)
<Frame_def>
(<Sporadic_frame_def>)
(<Event_triggered_frame_def>)
(<Diag_frame_def>)
<Node_attributes_def>
<Schedule_table_def>
(<Signal_groups_def>)
(<Signal_encoding_type_def>)
(<Signal_representation_def>)

<signal_def> ::=
Signals {
  [<signal_name>: <signal_size>, <init_value>, <published_by>
  [, <subscribed_by>] ;]
  }
```

```
Frames {
  CEM_Frm1: 0x01, CEM, 1 { InternalLightsRequest, 0; }
  LSM_Frm1: 0x02, LSM, 2 { LeftIntLightsSwitch,   0; }
  LSM_Frm2: 0x03, LSM, 1 { LSMerror, 0; IntError, 1; }
  RSM_Frm1: 0x04, RSM, 2 { RightIntLightsSwitch,  0; }
  RSM_Frm2: 0x05, RSM, 1 { RSMerror,              0; }
}
// name,PID,published_by,size,signal_name,signal_offset
//     0...59          byte                bit

Event_triggered_frames {
  Node_Status_Event :
    Collision_resolver, 0x06, RSM_Frm1, LSM_Frm1;
//                      PID
}

Schedule_tables {
  Configuration_Schedule {
    AssignNAD          {LSM}           delay 15 ms;
    AssignFrameIdRange {LSM, 0}        delay 15 ms;
    AssignFrameId      {RSM, CEM_Frm1} delay 15 ms;
    AssignFrameId      {RSM, RSM_Frm1} delay 15 ms;
    AssignFrameId      {RSM, RSM_Frm2} delay 15 ms;
  }
```

```
<frame_def> ::=
Frames {
  [<frame_name>: <frame_id>, <published_by>, <frame_size> {
  [<signal_name>, <signal_offset> ;]
  }]
  }

<frame_size> ::= integer     // 1..8 byte
<signal_offset> ::= integer  // 0 to (8 * frame_size - 1)

<event_triggered_frame_def> ::=
Event_triggered_frames {
[<event_trig_frm_name>:
<collision_resolving_schedule_table>,
<frame_id>
[, <frame_name>] ;]
}

<schedule_table_def> ::=
Schedule_tables {
[<schedule_table_name> {
[<command> delay <frame_time> ms ;]
}]
}
```

```
  Normal_Schedule {
    CEM_Frm1          delay 15 ms;
    LSM_Frm2          delay 15 ms;
    RSM_Frm2          delay 15 ms;
    Node_Status_Event delay 10 ms;
  }

  MRF_schedule { MasterReq delay 10 ms; }

  SRF_schedule { SlaveResp delay 10 ms; }

  Collision_resolver
   { // Keep timing of other frames if collision
    CEM_Frm1 delay 15 ms;
    LSM_Frm2 delay 15 ms;
    RSM_Frm2 delay 15 ms;

    RSM_Frm1 delay 10 ms; // Poll the RSM node
    CEM_Frm1 delay 15 ms;
    LSM_Frm2 delay 15 ms;
    RSM_Frm2 delay 15 ms;
    LSM_Frm1 delay 10 ms; // Poll the LSM node
  }
}

Signal_encoding_types {
  Dig2Bit {
    logical_value, 0, "off";
    logical_value, 1, "on";
    logical_value, 2, "error";
    logical_value, 3, "void";
  }

  ErrorEncoding {
    logical_value, 0, "OK";
    logical_value, 1, "error";
  }

  FaultStateEncoding {
    logical_value, 0, "No test result";
    logical_value, 1, "failed";
    logical_value, 2, "passed";
```

```
<signal_encoding_type_def> ::=
Signal_encoding_types {
[<signal_encoding_type_name> {
[<logical_value> |
<physical_range> |
<bcd_value> |
<ascii_value>]
}]
}
```

```
    logical_value, 3, "not used";
  }

  LightEncoding {
    logical_value,  0, "Off";
    physcial_value, 1, 254, 1, 100, "lux";
    logical_value, 255, "error";
  }
}

Signal_representation {
  Dig2Bit:             InternalLightsRequest;
  ErrorEncoding:       RSMerror, LSMerror;
  FaultStateEncoding: IntError;
  LightEncoding:       RightIntLightsSwitch,
                       LefttIntLightsSwitch;
}
```

—•—