Plug and Play: Node detection and node ID assignment with the LSS (Layer Setting Services) Fastscan service

Olaf Pfeiffer, Embedded Systems Academy GmbH

This paper introduces the new LSS Fastscan method that implements a generic and efficient scan cycle for non-configured devices. The LSS Fastscan was conceived for the CiA Application Profile 447 "car add-on devices" to simplify plug-and-play in the environment of taxis and other special vehicles. It was added to the CiA Document 305 "Layer Setting Services" and can thus be used by any CANopen implementation. Furthermore, the method is generic enough to also be used in CAN based systems using another higher-layer protocol.

Introduction and background

The automatic assignment of CANopen Node IDs is a requirement for applications requiring a high level of plug-and-play support. There have been various solution in the past, often optimized towards a specific application and not easily transferable to all applications. Software solutions often require a long scan time, it could take multiple minutes to complete a detection and assignment cycle.

One of the existing solutions in CANopen are Layer Setting Services (LSS) which are used for basic node configuration such as assigning a node ID and the CAN bit rate used. As previously defined, LSS has a few drawbacks like long and limited scan cycles for detecting nodes. A typical requirement was that the Vendor IDs of the installed devices needed to be known. Another drawback for microcontrollers with limited performance is that the existing LSS services use back-to-back messages which cannot be received by CANopen nodes with limited performance.

The LSS Fastscan service introduced by the engineers of the Embedded Systems Academy is an optimized variation of LSS allowing more efficient and flexible scan cycles. No back-to-back messages are required, allowing the implementation to be made also on embedded systems with limited performance.

The existing LSS Scan Cycle, a brief description

Each LSS Slave has a unique LSS ID, a 128bit combination of vendor ID, product code, revision number and serial number. A simplified scan for devices consists of multiple messages by the LSS Master that ask questions such as "Is there a node present whose LSS ID is between X and Y?" All LSS Slave nodes that match the criteria send a single acknowledgement response CAN message (no data). So as long as the LSS Master receives this ACK message it can continue with narrowing down X and Y. If no acknowledge is received within a timeout other ranges can be tried. Using a binary search pattern the Master can narrow X and Y until they are the same and one unique device ID is selected. The LSS Master then sends further messages in the style of "You, you with the LSS ID XYZ, take this new configuration!"

The significant delay factors in this cycle are the length of the LSS ID and the timeouts used. With a long 128bit LSS ID and multiple messages needed for each request (testing for lower and upper bound), the longer it will take for the LSS Master to narrow down the values of X and Y. Secondly the LSS Master needs to know how long to wait for the ACK message. The negative acknowledge cannot be implemented by a message, it is implemented by a timeout in the LSS Master when not receiving the ACK message. So with the timeout the LSS Master knows that there are no nodes present in the current range from X to Y and has to try another range. The length of this timeout significantly changes the length of the scan cycle.

As the timeouts are not standardized and are application specific true plug-and-play was never achieved with the existing mechanisms. Some applications using this mechanism required multiple minutes to complete an auto-detection and node ID assignment cycle.

Requirements for CiA447, the car add-on devices

When the interest group for car add-on devices was founded one of the first requirements was a reliable and efficient way to automatically assign node IDs. The scenario used as an example was that of a taxi with a printer connected to the CANopen network. At a taxi stand a typical situation could be that one driver says to the other: “My printer doesn’t work, may I quickly try yours?” So a hot swap of devices during operation needed to be supported and a scan cycle of even a minute would not be acceptable. For the taxi drivers there needs to be an almost instantaneous feedback if the newly attached printer does or does not work.

Optimizing the scan cycle step by step

So far the LSS scan cycle was not really standardized and not very efficient. However, any add-on would need to be backwards compatible and not interfere with existing implementations. For the Fastscan service a new command byte was introduced. The command byte is the first byte in the LSS Master message. Existing implementations would simply recognize that this new command is unknown and ignore the message. This ensures full backwards compatibility.

The next issue was that of multiple back-to-back LSS Master messages. Any new mechanism would need to use one message only for each iteration step of the scan cycle, allowing lower performance devices to participate in the cycle.

Lastly, the mechanism would need to be so simple, that even low performance devices can achieve a short response time allowing for short timeouts. A node that has not yet a node ID assigned is still not configured and in this mode typically has not much other tasks to do, so even a lower performance implementation should be able to handle response times in the area of single digit milliseconds.

Parameters of the LSS Fastscan Identify Master message

The 8 bytes (0 to 7) of the LSS Master Message are used as follows with the Fastscan service:

Byte 0, command byte:

Set to 81h, identifies this as a Fastscan Identify message

Bytes 1-4, IDNumber:

32bits that are currently checked versus the Vendor ID, product code, revision number or serial number.

Byte 5, BitChecked:

Defines how many of the bits in IDNumber are currently checked. This is a value in the range of 0 to 31. 31 means that only bit 31 is checked, 30 means that bits 31 and 30 are checked, 29 means that bits 31, 30 and 29 are checked and so on. 0 means that all 32bits are checked.

A value of 80h is an exception and indicates the start of a new scan cycle, all nodes supporting Fastscan reset their internal state machines and respond.

Byte 6, LSSSub:

Defines which part of the 128bit LSS ID is currently checked in the 32bit IDNumber. This is a value from 0 to 3 representing the Vendor ID, product code, revision number or serial number.

Byte 7, LSSNext:

Defines which part of the 128bit LSS ID will be checked towards the 32bit IDNumber in the next cycle. This is a value from 0 to 3 standing for the Vendor ID, product code, revision number or serial number.

The most simple best case: LSS ID known

If the LSS Master knows parts of the LSS ID of the nodes on the network, it can significantly shorten the scan cycle. The most simple case would be if an entire LSS ID is known. One can look at this scenario to get a first impression on how the Fastscan cycle works. The steps taken and initiated by the LSS Master would be:

1. Start new Fastscan cycle

The first LSS Master message would use a BitChecked value of 80h, indicating that all participating nodes should reset their internal state machines. All participating LSS slave nodes send an acknowledge message.

2. Verify Vendor ID

The LSS Master puts the known vendor ID into IDNumber, sets BitChecked to 0 (all 32 bits checked), sets LSSSub to 0 (current check is Vendor ID) and LSSNext to 1 (next check is product code).

Nodes that receive this message and have a match of the Vendor ID set their internal state machine to “I have a match, now waiting for product code” and send the acknowledge message. Nodes that do not have a match of the Vendor ID go back to waiting for the start of a new Fastscan cycle.

3. Verify product code

The LSS Master puts the known product code into IDNumber, sets BitChecked to 0 (all 32 bits checked), sets LSSSub to 1 (current check is product code) and LSSNext to 2 (next check is revision number).

Nodes that receive this message and have a match of the product code set their internal state machine to “I have a match, now waiting for revision number” and send the acknowledge message. Nodes that do not have a match of the product code go back to waiting for the start of a new Fastscan cycle.

3. Verify revision number

The LSS Master uses the same method as previously.

4. Verify serial number

The LSS Master uses the same method as previously. Note that LSSNext is set to 4, indicating that this is the last check.

The node that receives this message and has a match of the serial number sets its internal state machine to “I have a complete 128bit LSS ID match, now going directly into LSS configuration mode” and sends the acknowledge message.

5. Node ID Assignment

The node can now be assigned a node ID number using the existing LSS services. These are all single LSS Master messages (no back to back), so the entire cycle was now completed without the usage of back-to-back LSS master messages.

The second most simple case: a single bit of the LSS ID is unknown

To get a step by step understanding of the LSS Fastscan service it is best to slowly increase the complexity. After looking at the most simplistic case of the entire LSS ID being known to the LSS Master we now increase the complexity by one bit: lets assume the LSS Master does not know the highest bit (31) of the serial number (but knows all other bits of the LSS ID). The LSS Fastscan cycle would now look a little bit different. Steps 1 to 3 would be the same as in the previous example and step 4 would now be:

4. Determine bit 0 of serial number

The LSS Master sets IDNumber to zero, sets BitChecked to 31 (only bit 31 is checked), sets LSSSub to 3 (current check is serial number) and LSSNext to 3 (next check will still be for serial number).

NOTE: At this point the LSS Master got this far in the Fastscan cycle because it previously received acknowledgement messages. So there must be at least one node on the network that has all previous bits of the LSS ID (Vendor ID, product code and revision number)

Case A: a node responds with an acknowledge message because in its own serial number bit 31 is set to zero. The LSS master receives the acknowledge message and knows that there is at least one node which has bit 31 of the serial number set to 0.

Case B: no node responds with an acknowledge message and the LSS

Master internally generates a timeout. The LSS Master now knows (as in previous steps acknowledgement messages were received) that there must be a node which has bit 31 in the serial number set to 1.

In both cases the LSS Master can now add the newly gained knowledge to the existing knowledge about the LSS ID and complete the Fastscan cycle as in the previous example.

The worst case: LSS ID is entirely unknown to the LSS Master

Once a system is in place to determine a single unknown bit, this can easily be expanded to all 128 bits of the LSS ID. Using the BitChecked parameter the LSS Master can determine unknown bits step by step.

Optimization Options

The knowledge the LSS Master has about the nodes connected has a direct influence on the scan time. If mostly a few selected Vendor IDs or product codes are used in a system, then the LSS Master can scan for these first, before initiating a bit by bit cycle – so a scan for such nodes will complete significantly faster.

A method used in CiA447 and generally recommended is that the LSS Master internally keeps a list of all nodes previously found in non-volatile memory. A system restart without any changes (same nodes used as previously) then only consists of best cases and each node can typically be configured within 50 to 100 milliseconds.

Just adding/changing a single node then only requires one complete bit by bit check loop which typically completes in less then a second.

Timing Analysis and Test Implementation

The timeout used is by the LSS Master to determine if an acknowledge message was received or not is the biggest factor when it comes to the entire time needed to complete a scan cycle.

Unfortunately the timeout is also directly responsible for the level of plug-and-play support. If the LSS Master uses a timeout that is shorter than the response time of the LSS Slaves, the Fastscan cycle will fail.

As the LSS Master and Slave messages used have a very low CAN priority the LSS Master must be furthermore intelligent enough to dynamically extend the timeout in times of heavy CAN traffic load.

First implementations by Embedded Systems Academy have shown that LSS Fastscan Slaves based on embedded microcontrollers can start transmitting their responses to the LSS Fastscan Master in less than 5 milliseconds. On an otherwise idle network a LSS Master timeout of 7 milliseconds was used, resulting in a complete 128bit identification of a single node in less than a second.

The LSS Fastscan mechanism was implemented into Embedded Systems Academy MicroCANopen source code which is available at no charge for educational purposes.

Outlook

The LSS Fastscan service is currently in the process of being integrated into CiA305.

To ensure some level of guaranteed plug-and-play service we suggest introducing two classes of LSS Slaves, class I are nodes that can start transmitting their responses within 5 milliseconds of the reception of the LSS Fastscan Master message. Class II devices are those that can start transmitting their responses within 20 milliseconds of the reception of the LSS Fastscan Master message.

The responsibility for selecting a timeout value lies with the LSS Fastscan Master. Preferably the Master always supports an adaptive timeout that is automatically extended in cases of high CAN background traffic or in case of failures in the scan cycle.

The LSS Fastscan mechanism can also be used on other higher layer protocols. All that is required for the implementation is an unused CAN message identifier pair for the LSS Master request and the LSS

Slave response. Per default, these are 7E4hand 7E5h.

Pseudo Code for LSS Slave Responder

This section shows the pseudo code executed with the reception of the LSS Fastscan Master message. The parameters IDNumber, BitChecked, LSSSub and LSSNext need to be extracted from that message.

Found: local boolean variable

Mask: local unsigned 32bit variable

MyState: global unsigned 8bit variable

LSS_ID[4]: const array with four 32bit unsigned values containing 128bit LSS ID

```
Found = 0
IF BitChecked == 80
{ // Reset, re-init
  Found = 1
  MyState = 0
}
ELSE IF LSSSub == MyState
{ // Slave is in the state requested
  Mask = FFFFFFFFh << BitChecked
  IF (LSS_ID[LSSSub] & Mask) == (IDNumber & Mask)
  { // All bits requested match
    Found = 1;
    // Update own state as commanded by Master
    MyState = LSSNext
    IF BitChecked == 0
    { // all 32bit match
      IF LSSSub == 3
      { // Complete match, scan completed, NODE IDENTIFIED
        // Switch node to configuration mode now!
        PerformWhatIsNeededToSwitchToLSSConfigMode()
      } // end of LSSSub == 3
    } // end of BitChecked == 0
  } // end of bits match
} // end of LSSSub == MyState
IF Found == 1
{ // Send an acknowledge response as long as Found
  SendAcknowledgeMesaage()
}
```

Trace recording of LSS Fastscan execution

CAN	Msg Type	Details	Comment	Raw Message (hex)
0x7E5	LSS Request	LSSFS: Vendor ID - Ignore all bits	find non confg nodes	81 00 00 00 00 80 00 00
0x7E4	LSS Response	Identify	start vendor ID	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 31 - 31	start vendor ID	81 00 00 00 00 1F 00 00
0x7E4	LSS Response	Identify	response: bit 31 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 30 - 31		81 00 00 00 00 1E 00 00
0x7E4	LSS Response	Identify	response: bit 30 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 29 - 31		81 00 00 00 00 1D 00 00
0x7E4	LSS Response	Identify	response: bit 29 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 28 - 31		81 00 00 00 00 1C 00 00
0x7E4	LSS Response	Identify	response: bit 28 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 27 - 31		81 00 00 00 00 1B 00 00
0x7E4	LSS Response	Identify	response: bit 27 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 26 - 31		81 00 00 00 00 1A 00 00
0x7E4	LSS Response	Identify	response: bit 26 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 25 - 31		81 00 00 00 00 19 00 00
0x7E4	LSS Response	Identify	response: bit 25 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 24 - 31		81 00 00 00 00 18 00 00
			no response: bit 24 is one	
0x7E5	LSS Request	LSSFS: Vendor ID - bits 23 - 31		81 00 00 00 01 17 00 00
0x7E4	LSS Response	Identify	response: bit 23 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 22 - 31		81 00 00 00 01 16 00 00
0x7E4	LSS Response	Identify	response: bit 22 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 21 - 31		81 00 00 00 01 15 00 00
			no response: bit 21 is one	
0x7E5	LSS Request	LSSFS: Vendor ID - bits 20 - 31		81 00 00 20 01 14 00 00
			no response: bit 20 is one	
0x7E5	LSS Request	LSSFS: Vendor ID - bits 19 - 31		81 00 00 30 01 13 00 00
0x7E4	LSS Response	Identify	response: bit 19 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 18 - 31		81 00 00 30 01 12 00 00
			no response: bit 18 is one	
0x7E5	LSS Request	LSSFS: Vendor ID - bits 17 - 31		81 00 00 34 01 11 00 00
0x7E4	LSS Response	Identify	response: bit 17 is zero	4F 00 00 00 00 00 00 00
0x7E5	LSS Request	LSSFS: Vendor ID - bits 16 - 31		81 00 00 34 01 10 00 00
			no response: bit 16 is one	
			continuation	
0x7E5	LSS Request	LSSFS: Vendor ID - bits 0 - 31	verify entire vendor ID	81 41 53 35 01 00 00 01
0x7E4	LSS Response	Identify	ID 0x01355341 confirmed	4F 00 00 00 00 00 00 00
			continuation	
0x7E5	LSS Request	LSSFS: Serial Number bits 0 - 31	serial number 0x00000000	81 00 00 00 00 00 03 04
0x7E4	LSS Response	Identify		4F 00 00 00 00 00 00 00
0x7E5	LSS Request	Configure Node ID - NID: 0x02	assign node ID 2	11 02 00 00 00 00 00 00
0x7E4	LSS Response	Configure Node ID - Success		11 00 00 00 00 00 00 00
0x7E5	LSS Request	Store Configuration		17 00 00 00 00 00 00 00
0x7E4	LSS Response	Store Configuration - Success		17 00 00 00 00 00 00 00
0x7E5	LSS Request	Switch Mode Global - Operation	start node	04 00 00 00 00 00 00 00
0x702	Bootup		node 2 boots up	0

The trace recording above is a summary of a LSS Fastscan execution. The columns show the CAN message identifier seen on the network, the message interpretation with type and details, a comment and the raw message contents.

Olaf Pfeiffer
Embedded Systems Academy GmbH
Bahnhofstr. 17
D-30890 Barsinghausen
opfeiffer@esacademy.com
www.esacademy.com

References

[1] CiA DS 301, CANopen application layer and communication profile
[2] CiA WD 447, Car add-on devices
[3] CiA WD 305, Layer Setting Services (LSS) and protocols
[4] MicroCANopen source code, www.microcanopen.com