NCN26010 - Getting Started Basic Configuration, Communication and Exception Handling

AND90155/D

Introduction

The NCN26010 10Base−T1S device has been developed to adhere to the IEEE 802.3cg specifications, as well as the SPI protocol of the Open Alliance (TC6) with various optional additional capabilities.

For correct and reliable operation, there are a few things to keep in mind when configuring NCN26010, especially since the device cannot participate in communication in a multi−drop segment without correct configuration. Also, there is a risk of permanently disrupting the entire segment in the event of improper configuration.

This application note is intended to provide users with a guideline for configuring NCN26010 for their specific application.

Only the basic settings needed are described here, and these are discussed using configuration examples.

Reference Documents

[1]	IEEE802.3cg−2019 “IEEE Standard for Ethernet Amendment 5: Physical Layers Specifications and Management Parameters for 10 Mb/s Operation and Associated Power Delivery over a Single Balanced Pair of Conductors” IEEE Computer Society, ISBN 978−1−5044−6420−8
[2]	OPEN Alliance TC6 − 10BASE−T1x MACPHY Serial Interface Version 1.0 from 14.September 2020

Operating Modes

NCN26010 offers mandatory and optional operating modes as defined in the IEEE 802.3cg Standard as well as some features that offer extended functionality:

- CSMA/CD as the basic operation and fall back
- Physical Layer Collision Avoidance (PLCA)
  - Burst Mode
  - Precedence Mode
- Enhanced Noise Immunity (ENI)

As a MAC−PHY device, the NCN26010 integrates both a 10Base−T1S Physical Layer Device and an IEEE802.3 Clause 4 Compliant Media Access Controller into a single device. This combination offers Ethernet communication to low cost MCUs, that offer a SPI interface capable of running at least 15MHz and having a TCP/IP stack (e.g., FreeRTOS+) implemented in software.

We will highlight the basic functionality of the Open Alliance MACPHY SPI protocol. For in depth details, users are recommended to consult the OPEN Alliance TC6 document in revision 1.0.

To illustrate the basic use of the part, this application note starts with discussing how to read and write configuration registers as well as sending actual ethernet frame data through the SPI interface.

Further down in the text a set of basic configurations and the use of Address Filters and Filter Masks inside the MAC will be explained.

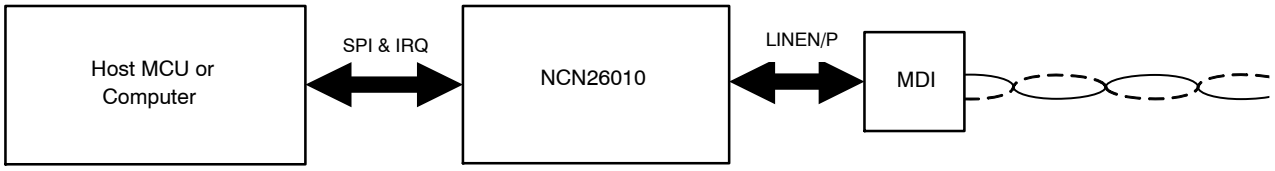

Figure 1. Simplified Block Diagram

APPLICATIONS INFORMATION

SPI Interface

NCN26010 has implemented all mandatory features of the OPEN Alliance TC6 SPI protocol.

The Protocol uses the SPI interface for both Ethernet frame receive and transmit as well as exchange of configuration data through so called control command transactions.

Control and data transactions can be differentiated by looking at the MSB of the 32−bit long communication header.

Control Command Header

A control command is indicated by having bit 31 (DNC = Data – Not Command) set to zero. The format for a control command looks like this:

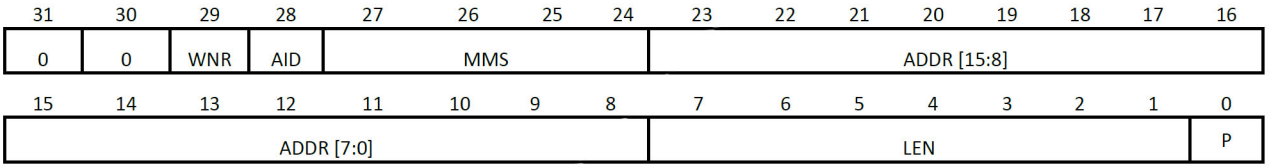

Figure 2. Control Command Format

Data Header

To transfer ethernet data (in both directions) the host will have to send valid data transfer headers followed by the actual Ethernet data. A data transfer header is characterized by bit 31 being set to one. The following shows the fields of a Data Header with respect to use with the NCN26010 device.

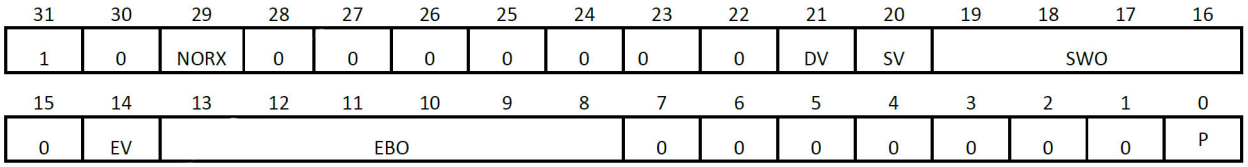

Figure 3. Data Header Fields

Note that in the NCN26010 case only the fields NORX, DV, SV, SWO, EV, EBO and the mandatory (odd) Parity P are used. All other bits are recommended to be always sent containing 0.

For a detailed description of the field names and functions, please consult the Open Alliance TC6 document, rev 1.0.

Writing and Reading Configuration and Status register

To write or read a single register, a total of 12 bytes will have to be sent to the NCN26010 10Base−T1S MACPHY.

Control command starts with a command control header followed by:

- 32 bits of register content and 32−bit dummy data for a register write control transaction (register write)
- 64 bits of dummy data for read control transaction (register read)

SPI, being bidirectional full duplex, needs one SCK transition per transferred/received bit. To receive the appropriate number of bytes for a register read or write transactions, dummy bytes need to be sent by the MCU host on the MOSI line to the NCN26010 device.

The connected MCU must be able to transfer a complete transaction without de−asserting the MACPHY's SPI chip select.

It appears easiest to prepare two 12−byte long arrays for register read and write. Fill the array with correct data (header and write data) and then send it in a 12−byte bulk transfer over SPI, while receiving the result into the 12−byte receive buffer.

Register Read Example:

An application wants to **read** the MACPHY's SPI Identification register located at MMS0 Address 0x0001.

First, we need to generate an appropriate header: Follow this theme to determine the header:

Field	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WNR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MMS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADDRESS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
LENGTH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bitwise OR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
odd parity	1 if number of ones is even, else 0																															0
Header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
hex	**0**				**0**				**0**				**0**				**0**				**1**				**0**				**0**			

Figure 4.

Once the header is determined, fill the array with zero byte for a read operation.

This will provide a byte array to be sent on the MOSI and received on the MISO with the following content:

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
	HEADER				DATA							
MOSI	00	00	01	00	00	00	00	00	00	00	00	00
MISO	DONT CARE				00	00	01	00	READ DATA			
					ECHOED HEADER							

Figure 5.

A program routine that would read this register would have to look at the last two 32−bit words in the data stream.

The penultimate 32−bit word (echoed header, Bytes 4 to 7) should be equal to the command header sent on the MOSI line to the MACPHY, while the last 32−bit word contains the actual data of the register.

Note that register reads always return 32−bit words of register content, even if the underlying register only contains 16 bits of usable content.

When there is a parity error on the header due to either malforming of the header by using an invalid parity bit calculation or a transmit error on the SPI, the echoed control header and all subsequent 32−bit words will be 0x40000000, indicating a "HEADER BAD" condition. As a result, the MACPHY will ignore the command, a connected host can use that information to detect a Control Command error and trigger a retransmission of a correct control command.

Register Write Example:

Assume you want to write the MAC Control 0 register inside the NCN26010 to enable TX and RX transfers between the integrated MAC to and from the integrated PHY, while also having the MACPHY calculate the Ethernet Frame's Frame Check Sequence (FCS) to off−load the MCU from having to perform that calculation.

Consult the Datasheet of NCN26010 to find the MAC CONTROL0 Register. It is in MMS 1 (Memory Map Selection Group 1) at address 0x0000.

To enable TX and RX as well as having the MAC calculate and auto−append the FCS (Frame Check Sequence) to every Ethernet frame it will send, the bits 8 (FCSA), 1 (TXEN) and 0 (RXEN) will have to be set in the register. This results in a data word to be written which contains the value 0x00000103.

Similar to the above example for register read, we need to first determine the correct control command header for a single register write. Following the above scheme, we would get:

Field	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WNR	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AID	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MMS	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADDRESS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LENGTH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bitwise OR	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
odd parity	1 if number of ones is even, else 0																															0
Header	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
hex	**3**				**1**				**0**				**0**				**0**				**0**				**0**				**0**			

Figure 6.

Once we have determined the header, we fill the array with the data we want to write into the register.

This will give us a byte array to be sent on the MOSI and received on the MISO with the following content:

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
	HEADER				DATA				Don't Care			
MOSI	31	00	00	01	00	00	01	03	00	00	00	00
MISO	DONT CARE				31	00	00	01	00	00	01	03
					ECHOED HEADER				ECHOED WRITE DATA			

Figure 7.

Note that the Parity checking works the same way as in the register read example. A false Parity bit in the header results in subsequent 32−bit words to be set to 0x40000000. In such case the MACPHY would not alter the content of the register. A Connected MCU would have to handle this situation accordingly by re−issuing the register write transaction with an intact header. Also, it is important that data is sent MSB first.

The following code snippet should give an example implementation of a function generating the command transaction header:

```
struct TC6_command_header {
         unsigned int WNR :1;
         unsigned int AID :1;
         unsigned int MMS :4;
         unsigned int ADDR: 16;
         unsigned int LEN:7;
         unsigned int P:1;
         };
unsigned int CalcCmdHeader(struct
TC6_command_header header) {
              unsigned int result = 0;
             result |= header.WNR<<29;
             result |= header.AID<<28;
             result |= header.MMS<<24;
             result |= header.ADDR<<8;
             result |= header.LEN <<1;
      result |= !__builtin_parity(result);
                  return result;
}
```

Figure 8.

The above examples illustrate how to generate a valid control transaction allowing single register read and write.

Readers of this application note are encouraged to check the Open Alliance TC6 document for details on how to read consecutive register addresses in one SPI transfer.

Sending and Receiving Ethernet Frames

The OA−TC6 Protocol is designed not only to allow device configuration through the SPI interface, but also to utilize the same interface to transport Ethernet frames in both directions, either in a half−duplex or full−duplex fashion, depending on the capabilities of the software running on the host system. Please note, however, that the actual Ethernet traffic on the mixing segment is *always* half−duplex.

Ethernet frames are transferred in both direction in so−called chunks.

The Concept of Data Chunks

The NCN26010, implementing the OA SPI protocol, transmits Ethernet data to/from a connected host device or system using “data chunks”.

A data chunk is a fragment of an Ethernet frame or on short frames an entire Ethernet frame. Data chunks in the transmit direction have a data header followed by a predefined number of pay load data. NCN26010 can be configured to 8, 16, 32 or 64 bytes of pay load per chunk, with the default being 64 bytes. In the receive direction, when a host is receiving data from the NCN26010 MACPHY, the pay load is sent first and is followed by a 32−bit data footer. The footer indicates to the host:

- If the sent data contains valid Ethernet frame data.
- If there were errors seen on the last transmission.
- The available number of data chunks in the NCN26010’s receive buffer ready for reading.
- The available number of empty transmit chunks that can currently be used for TX data transfers.

A full ethernet frame that is longer than the amount of data that a single data chunk can transport will need to be cut into pieces that fit inside data chunks. This concept allows interrupts of an Ethernet frame exchange between NCN26010 and host device without losing data, as long as there is enough space available in the 4 kbyte send and receive buffers inside the NCN26010 device.

Depending on configuration, the Ethernet frame data may or may not contain the 4−byte FCS (frame check sequence). When configured to calculate and auto append the FCS, transmit data is not expected to have the FCS appended when it is sent to the NCN26010 by the host device. Similarly, when FCS checking inside the NCN26010 device is enabled, incoming frames do not have to provide an FCS since invalid frames (frames that have an invalid FCS) will not be stored in the RX buffer (i.e. they will be dropped).

NCN26010 can be configured to either operate in “Store and Forward” mode or a “Cut Though” mode of operation.

In Store and Forward mode, entire frames will be stored inside the device’s buffer before they are either sent to the single twisted pair Ethernet segment or received from the ethernet medium.

Cut Though mode has more stringent requirements on the latency and throughput performance of the software running on the host, as the host will always have to keep up with the speed of the incoming and outgoing ethernet data. Slowing down communication, caused by long interrupt service routines or unfavorable task switching when using multitasking operating systems, will cause data to be lost due to TX buffer underruns or RX buffer overruns.

This application note will not address data exchange using the NCN26010’s Cut Though mode but rather focus on the Store and Forward mode of operation.

When running in Store and Forward mode, the host will always have to keep an eye on the buffer fill levels, especially for the RX side (or downstream). For TX upstream traffic, the software running on the host can easily defer traffic until the required amount of data chunks is available (i.e., the amount that would fit the current Ethernet frame scheduled to be transferred).

As the host does not have the RX (downstream) data flow under control, precedence should be given to RX traffic if the device exchanges data in a half−duplex fashion.

The selection of the chunk size, among other configuration options, imposes a minimum speed on the SPI interface that connects NCN26010 to the Host MCU or System.

When using a simple configuration, where new Ethernet frames are only allowed to start at the beginning of a chunk, the required SPI speed depends on the length of the Ethernet

frame as well as the chunk size: The following graph illustrates this relationship.

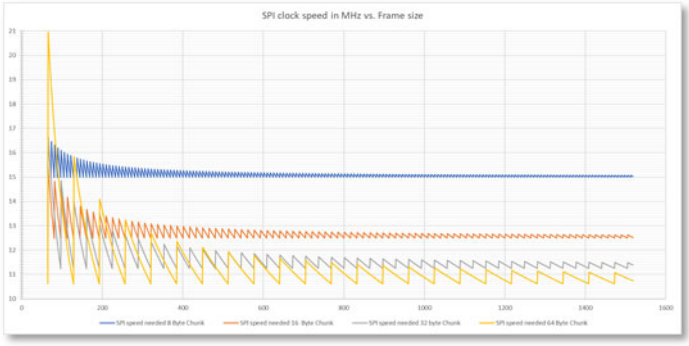

Figure 9.

From this graph, the limitation of this approach becomes obvious. When the application primarily uses short frames, a smaller chunk size might give a higher throughput, depending on how the NCN26010 is configured. In this simple configuration, the extreme would be 65−byte long frames with a chunk size of 64 bytes. This will require a SPI speed of 21 MHz.

NCN26010 is designed to support SPI speeds of up to 25 MHz on the SCLK, so even under this condition, the MACPHY will be fast enough to transport the required data; however, the application program or driver will have to handle both RX and TX simultaneously. As the Ethernet frames get larger in size, the default chunk size of 64 bytes will require lower clock speeds on the SPI interface.

In a multi−drop environment, we could assume that most of the traffic is in the RX direction. Due to the 10Base−T1S half−duplex nature, a 25 MHz SCLK should provide stable dataflow without loss of data in most of the cases, even when handling the SPI TX and RX traffic in a half−duplex fashion.

When NCN26010 should transmit data on the single pair Ethernet segment a host will have to send a series of valid data chunks containing the complete ethernet frame data.

A new Ethernet frame will start with the data header's SV flag (Start Valid) set to one. In the simple case where the frame is only allowed to start at the beginning of a chunk right after a CS (chip select) being asserted, the Start Word Offset will always be 0. If the ethernet frame fits inside a single data chunk, the EV (End Valid) flag will also have to be set as well as the position of the last valid byte of payload data. Note that when the Ethernet frame data to be transmitted is shorter than the minimum required Ethernet frame length of 64 bytes (including source/destination address, length, field/ether−type, and FCS), the MAC inside NCN26010 will add the padding bytes required to extend the frame to its minimum required length. Padding bytes as well as FCS will only have to be provided by the host, if NCN26010 is configured to not auto−append the FCS.

Analogue to the calculation of a control transaction header, the data transaction header can be generated as shown in the example code below:

```
struct TC6_data_header {
unsigned int NORX :1;
unsigned int DV :1;
unsigned int SV: 1;
unsigned int SWO:4;
unsigned int EV: 1;
unsigned int EBO: 6;
unsigned int P:1;
};
unsigned int CalcDataHeader(struct TC6_data_header
header) {
unsigned int result = 0x80000000; //DNC is always one
for a data transaction
result |= header.NORX<<29;
result |= header.DV<<21;
result |= header.SV<<20;
result |= header.EV<<14;
result |= header.EBO <<8;
result |= header.SWO <<16;
result |= !__builtin_parity(result);
          return result;
 }
```

Figure 10.

Sending Ethernet Frames

Here we only describe sending TX frames as referred to as “TX Frame ends at Chunk Boundary” as well as “Every TX frame fits into Chunk”. Please see [2] “Figure 8: Transmit Data Chunk Cases” for details.

We can distinguish three cases of frame data that need to be handled differently:

1. Frame fits entirely into a single data chunk
2. The Ethernet frame fits into two data chunks
3. The Ethernet frame need more than two data chunks

In the first case users will send a data header followed by the actual frame data. the data header will have these bits set:

- SV = 1
- DV = 1
- EV = 1

EBO (End Byte Offset) points to the last byte of valid data. Note that even if the Ethernet data does not fill the entire chunk (i.e. an ARP broadcast frame has a length of 46 bytes, so a default 64 byte chunk would not be fully utilized) the application will always have to send full length data chunks.

In the case of a 64−byte chunk and a 46−byte long ARP (Address Resolution Protocol) broadcast to be sent, the EBO will point to byte 45 as the first byte in the chunk has the index 0.

In the second case, where two chunks are needed to fit the payload, the first header would have:

- SV = 1
- DV = 1
- EV = 0
- EBO = 0
- SWO = 0

While the second and last data header in the frame will have:

- SV = 0
- DV = 1
- EV = 1
- EBO = position of last valid pay load byte
- SWO = 0

In the case where more than two chunks are needed to transfer the payload data to the NCN26010 MACPHY, the first header will look like the one in the previous case, with:

- SV = 1
- DV = 1
- EV = 0
- EBO = 0
- SWO = 0

All headers following the first and preceding the last chunk will have:

- SV = 0
- DV = 1
- EV = 0
- EBO = 0
- SWO = 0

Or 0xA0200000.

The last header will have:

- SV = 0
- DV = 1
- EV = 1
- EBO = position of last valid pay load byte
- SWO = 0

Because of the header preceded data transfer, an application that sends a data chunk to the MACPHY will always have to send 4 Bytes more than the chunk’s payload.

For an 8−byte chunk this will be 12 bytes, for 16−byte chunk 20 bytes need to be provided, a 32−byte chunk needs 36 bytes total, while a 64−byte chunk requires 68 bytes.

Note that these are fixed length even if a data chunk is not fully used. Data sent after the EBO at a EV=1 chunk is “don’t care” and can be any random data. For the ease of use it is recommended to fill the chunk with bytes containing all zeros.

Like sending control transactions, a data transaction simultaneously works in both directions.

When sending data to NCN26010 via the SPI interface, the host will receive the same number of bytes back provided by the MACPHY via the MISO (Master In Slave Out) pin of NCN26010. Typically, data is sent and received from a buffer large enough to hold all the data chunk needed to send frame data. Incoming data would go into a second (i.e., a receive) buffer.

Receiving Ethernet Data

While data sent to the MACPHY is preceded by a per chunk 32−bit header, data chunks sent by the MACPHY have a 32−bit footer following the actual data.

The footer contains information that the host can use to direct further communication, regardless of it sending or receiving more data. The fields of the footer are explained in [2] section 7.3.7. NCN26010 does not provide the optional time stamping feature defined by the Open Alliance, hence the fields RTSA (bit 7) and RTSP (bit 6) will not be used and always contain 0.

In the context of the NCN26010 device the data footer looks like this:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EXST	HDRB	SYNC	RCA					DONT CARE		DV	SV	SWO			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FD	EV	EBO						DONT CARE		TXC					P

Figure 11.

The fields can be summarized to have the following functions:

EXST	External Status. When this flag is one then the host program will have to examine the content of the SPI Protocol STATUS0 register (MMS: 0, Address: 0x0008) and handle errors accordingly. See “Error Handling” section in this Application Note
HDRB	Header Bad. This indicates that the MACPHY has receive a control transaction or data chunk whose parity bit was not correct. Hence it indicates loss of data. A host can opt to resend the last transaction/ chunk or handle that situation in another way.
SYNC	This is a copy from the SPICFG0 register bit 7. It will normally be set by the host after it finishes configuring the MACPHY. A MACPHY that has its SYNCH bit set to zero will not send or receive Ethernet data. Once set, SYNCH can only be cleared by resetting the device
DV	Data Valid. The function is the same as for sending data. When it is one, the chunk contains valid Ethernet data.
SV	Start valid. When set to one, the chunk contains the start of a new Ethernet frame.
SWO	Start Word Offset. Points to the location inside the data chunk where a new Ethernet frame starts. In the basic configuration, where new frame data can only start at the beginning of a data chunk directly followed by the assertion of the Chips Select (CSn) this field should always be zero
FD	Frame Drop. This can only be one in cut through mode. It indicates to the host that it should discard the entire Ethernet frame it received, mainly due to a wrong FCS being received.
EV	End Valid. A chunk with a footer that has EV set will contain the end of an Ethernet frame. A connected host will typically read data chunks until is sees the EV flag in the footer. It can then pass a complete Ethernet frame to its TCP/IP stack or any other application software or hardware handling the OSI Layers 3 and above.
EBO	End Byte Offset. Same as in transmit header, points to the last byte of valid frame data when the EV flag is set.
RCA	Receive chunk available. Tells the host how many more chunks of valid data are available in NCN26010’s receive buffer for the host to read. Note that these chunks can belong to more than one complete Ethernet frame. To avoid data loss due to receive buffer overflows, the host should always try to keep the RCA number as low as possible.
TXC	Transmit chunks available. Tells the host how many transmit chunk are available for writing inside the TX buffer of NCN26010. Both TXC and RCA could also be polled by reading the buffer status register (MMS: 0, Address: 0x000B)

The simplest, yet slowest way of exchanging Ethernet data is in a half duplex fashion. When sending frames, with the NORX flag set in the Data Header, NCN26010 is instructed to not send Ethernet data to the host while receiving TX frame data from the host.

Likewise, frame data can be received by the host without providing TX data, when the data header’s DV flag is set to zero.

When operating in this fashion, the connected host should give precedence to reading frames over transmitting frames or do a “fair−share” of alternating transmit and receive. It is advised to always try to empty the receive buffer as quickly as possible.

Basic Configuration

This section discusses the minimum configuration settings required to use the NCN26010 as well as some of its optional features. It starts with the simple case of CSMA/CD without address filtering and then discusses how to enable additional features:

- Address filtering
- PLCA
  - Leader mode
  - Follower mode
  - Burst mode
- FCS frame check
- FCS calculations off−load
- Enhanced Noise Immunity

Configuring NCN26010 for the simplest form of operation means setting it to strictly work in CSMA/CD (carrier sense multiple access / collision detection) mode, with none of the address filtering options being activated.

First, we will have to set the SPI protocol format to the desired form. In this application note we chose to select the following SPI protocol transfer options:

- Receive frames will be aligned to start at the beginning of a receive chunk payload (byte 0) directly following CSn assertion by the host
- Transmit and Receive in “Store & Forward” mode
- Control Data Read/Write protection disabled (see [2] section 7.4 for details)
- Default payload size of 64 bytes per data chunk.

To avoid configuration glitches with stations inadvertently occupying the Single Pair Ethernet medium, it is recommended to configure the device in the following order:

1. Device reset
   A device reset can be forced in three different ways: a) power cycling (power−on Reset);
   b) asserting the physical RSTn pin on the device; or
   c) writing configuration registers.
   Since sometimes error handling requires a device reset, we chose the latter method of resetting the device through a register write.
   NCN26010 has a dedicated register allowing it to be soft reset.
   Setting bit 0 in the RESET register located at MMS 0, Address 0x0003 will trigger a soft reset.
   Assume you have a routine to write a configuration register called T1SRegWrite (MMS, ADDR, DATA). To issue a device reset, you would call:
   T1SRegWrite(0,0x0003,0x00000001).
2. PHY related configurations
   a.) PLCA
   b.) ENI
   In this simple CSMA/CD configuration, no action is required, other than “activating the link” by setting bit 12 in the PHYCRTL register (MMS0, Address 0xFF00) to one:
   T1SRegWrite(0, 0xFF00, 0x00001000)
3. MAC related configurations:
   a.) FSC filtering and calculation
   b.) Address filtering
   c.) Broadcast and Multicast filtering
   For basic operation we chose the device to auto−append the FCS, allow Broadcast and Multicast messages, and not filter any MAC addresses. This is effectively running the device in promiscuous mode. To achieve this, we set bit 8 (FCS append), bit 1 (TX Enable) and bit 0 (RX Enable) in the MACCTRL0 (MMS 1, address 0x0000):
   T1SRegWrite(1,0x0000,0x00000103)
4. SPI configuration
   a.) Data alignment
   b.) CutThrough or Store and Forward
   c.) Chunk Payload size
   d.) TXC Threshold
   e.) Finally set the SYNC bit to enable the communication
   We want the device to align all new frames with the CS assertion starting at byte 0 of the new data chunk. The part should operate in “S&F” mode for both TX and RX and the TXC Threshold we want to be 16.
   All settings are done in the SPICFG0 register (MMS 0, Address 0x0004).
   We need to set SYNC (bit 15), CSARFE (bit 13), TXCTRHESH = 0x3 (bits 11 and 10), CPS = 0x6 (bits 2:0). The resulting register content is: 0x0000AC06.

In summary, basic operation can be achieved by using the configuration shown in the table below, when writing the registers in the order shown (we refer to this as the configuration “card”).

MINIMUM CSMA/CD CONFIGURATION

order	MMS	ADDR	DATA	comment
1	0	0x0003	0x00000001	Reset
2	0	0xFF00	0x00001000	Activate Link
3	1	0x0000	0x00000103	Auto FCS + MAC enable
4	0	0x0004	0x0000AC06	Config SPI and set SYNC

Note that the above lists show the bare minimum that needs to be sent to the device after reset to allow the MACPHY to participate in communication over the single pair Ethernet medium and with the host Computer or MCU.

Adding PLCA

When we want the device to be part of a PLCA enabled network, we need to configure and enable the functions of the PLCA Reconciliation Sublayer (RS) inside the NCN26010 MAC PHY. Please consult [1] Clause 148 for a detailed description of Physical Layer Collision Avoidance (PLCA). In a PLCA enabled collision domain, there needs to be one so−called Leader. This is typically the device that is assigned the PLCA ID=0. The Leader is the station that starts PLCA cycles by submitting the BEACON signal on the line.

The leader will also need to be configured with a max node number that needs to be equal or higher than the highest ID assigned to any station participating in the local collision domain.

When we want the part to be a Follower node, we must assign it a local ID and enable PLCA.

This is done by setting the appropriate node ID in the PLCACTRL1 register (bits 7:0) and enable PLCA (bit 15) in the PLCACTRL0 register.

As an example, assume we chose ID=7 for the node we configure. The configuration card from the previous example must be extended and will now look like this (lines in red show the added configuration items):

MINIMUM PLCA FOLLOWER NODE CONFIGURATION

order	MMS	ADDR	DATA	comment
1	0	0x0003	0x00000001	Reset
2	0	0xFF00	0x00001000	Activate Link
3	1	0x0000	0x00000103	Auto FCS + MAC enable
4	4	0xCA02	0x00000007	Set PLCA ID to 7
	4	0xCA01	0x00008000	Enable PLCA
5	0	0x0004	0x0000AC06	Config SPI and set SYNC

When instead we want to configure the node to be the PLCA leader, the local ID is set to 0 and the maximum node count needs to be set to a reasonable number (8 or larger).

Assume we want to set up a leader that supports 8 nodes in the collision domain (the multi−drop sequence). The configuration card will change to the one shown below:

MINIMUM PLCA HEAD NODE (NODE COUNT 8)

order	MMS	ADDR	DATA	comment
1	0	0x0003	0x00000001	Reset
2	0	0xFF00	0x00001000	Activate Link
3	1	0x0000	0x00000103	Auto FCS + MAC enable + Address
4	4	0xCA02	0x00000800	PLCA ID = 0 Node Count = 8
	4	0xCA01	0x00008000	Enable PLCA
5	0	0x0004	0x0000AC06	Config SPI and set SYNC

We can add more configuration options following this principle. For instance, PLCA has additional options that can be enabled to deal with specific situations in application that have an uneven spread of sent data from different stations. For example, if a particular station needs to send twice the amount of data than all others, it could be allowed to send two or more frames per PLCA transmit opportunity instead of only one. This is supported enabling the **PLCA burst mode** (see [1] Clause 148.4.4.2). Burst mode is enabled by setting the maximum burst count (the number of additional Ethernet frames that may be transmitted during a single transmit opportunity) bits 15:8 to the number of additional frames a station can send per transmit opportunity. When we want to allow the station to send two frames per TO (Transmit Opportunity), we need to set the MAXBC inside the PLCABURST register to 1. Hence the configuration card will change to:

PLCA HEAD NODE (NODE COUNT 8) WITH BURST ENABLED

order	MMS	ADDR	DATA	comment
1	0	0x0003	0x00000001	Reset
2	0	0xFF00	0x00001000	Activate Link
3	1	0x0000	0x00000103	Auto FCS + MAC enable
4	4	0xCA02	0x00000800	PLCA ID = 0 Node Count = 8
	4	0xCA05	0x00010080	Allow one extra frame per TO
	4	0xCA01	0x00008000	Enable PLCA
5	0	0x0004	0x0000AC06	Config SPI and set SYNC

Enhanced Noise Immunity Mode

In applications that are exposed to an elevated level of noise on the Single Pair Ethernet line (i.e., close to motors or AC lines, or multiple SPE connections bundled in a single multi twisted pair cable) NCN26010 provides superior performance by offering a non−standard feature called Enhanced Noise Immunity (ENI). ENI works without side effects in PLCA enabled network segments. When activating ENI in CSMA/CD operation, the station ability to

detect collisions will slightly degrade, however ENI will not break communication on the network.

Enabling ENI can be done by simply setting bit 7 in the PHYCFG1 register (MMS 4, address 0x8001). Adding this to the previous example will make the configuration card look like this:

PLCA HEAD NODE (NODE COUNT 8) WITH BURST ENABLED AND ENI

order	MMS	ADDR	DATA	comment
1	0	0x0003	0x00000001	Reset
2	0	0xFF00	0x00001000	Activate Link
3	1	0x0000	0x00000103	Auto FCS + MAC enable
4	4	0xCA02	0x00000800	PLCA ID = 0 Node Count = 8
	4	CA05	0x00010080	Allow one extra frame per TO
	4	0xCA01	0x00008000	Enable PLCA
	4	0x8001	0x00000083	Enable ENI
6	0	0x0004	0x0000AC06	Config SPI and set SYNC

Address Filtering

Next, we can look at enabling address filtering. NCN26010 has four configurable MAC address filters that can be set with wildcards to match a wide range of destination MAC address or just single dedicated ones.

Address filtering is enabled by first setting the ADRF field (bit 16) in the MAC Control0 register (MMS 1, Address 0x0000). Then a filter rule can be set in one of the ADDRFLT registers and corresponding filter masks must be defined in the ADDRMASK registers.

If the device is to filter a specific address, all bits of the corresponding ADDRMASK must be set to 1. ADDRFLT and ADDRMASK are both 48 bits long, corresponding to the length of standard Ethernet MAC addresses. Only bits with the MASK bit set to 1 are considered for filtering.

Example: the address filter should only store incoming frames that match the MAC Address 60:c0:bf:01:02:03

We use ADDRFLT0H/L to input that address into the filter:

ADDRFILT0H=0x800060c0

ADDRFILT0L=0xbf010203

Note that bit 31 in the ADDRFILT0H enables the filter rule when set to 1.

As we want the filter to look at the entire MAC address, we need to set the ADDRMASK0H/L accordingly.

ADDRMAKS0H = 0x0000FFFF

ADDRMASK0L = 0xFFFFFFFF

To set the filter and the mask right, it is important for users of this feature to understand the principal of operation:

The filter takes effect if the bitwise AND operation of the destination address of the incoming Ethernet packet is equal to the content of the filter register.

So,

Destination MAC address & ADDRMASK = Filter

Must be true for a frame to pass the filter.

With that, if we wanted the filter to match on only the OUI (60:C0:BF is **onsemi**'s OUI)

We could set the filter mask to

ADDRMAKS0H = 0x0000FFFF

ADDRMASK0L = 0xFF000000

And the filter to:

ADDRFLT0H = 0x00060C0

ADDRFLT0L = 0xBF000000

If we do the test for incoming destination address of 60:C0:BF:01:01:15 (let's use binary notation here)

0110 0000 1100 0000 1011 1111 0000 0001 0000 0001 0001 0101

&

1111 1111 1111 1111 1111 1111 0000 0000 0000 0000 0000 0000

=

0110 0000 1100 0000 1011 1111 0000 0000 0000 0000 0000 0000

6 0 C 0 B F 0 0 0 0 0 0

we see that the MAC address filter accepts all frame whose destination address starts with 60:C0:BF

If we extend the configuration card for this MAC address filtering, it will look like this:

PLCA HEAD NODE (NODE COUNT 8) WITH BURST ENABLED AND ENI, MAC FILTER ON 60:C0:BF:01:02:03

order	MMS	ADDR	DATA	comment
1	0	0x0003	0x00000001	Reset
2	0	0xFF00	0x00001000	Activate Link
3	1	0x0000	0x00010103	Auto FCS + MAC enable+ Filter Enable
4	4	0xCA02	0x00000800	PLCA ID = 0 Node Count = 8
	4	CA05	0x00010080	Allow one extra frame per TO
	4	0xCA01	0x00008000	Enable PLCA
	4	0x8001	0x00008003	Enable ENI
	1	0x0020	0xFF000000	ADDRMASK0L
	1	0x0021	0x0000FFFF	ADDRMAKS0H
	1	0x0010	0xbf000000	ADDRFLT0L
	1	0x0011	0x800060C0	ADDRFLT0H + enable bit set
6	0	0x0004	0x0000AC06	Config SPI and set SYNC

All of these examples demonstrate the use of configuration registers, as described in the datasheet of NCN26010, to set the part into simple and advanced modes of operation. Users are invited to explore more of the part's capabilities by studying the NCN26010 datasheet.

Handling Exceptions

The following is a guide on potential actions to take when the NCN26010 signals errors or exceptions to the host. The content of this section should be regarded as recommendations, as there will be a multitude of different possible action that could result from errors or exceptions reported by NCN26010 depending on the application's needs.

Whenever NCN26010 needs to notify the host of occurring events, it will assert the IRQn pin to trigger an interrupt event in the software running on the host.

In the normal case, where there are no errors or abnormal events, the interrupt is triggered to notify the host of new ethernet frames having been received and stored in NCN26010's RX buffer ready to be picked up by the host.

Whenever an interrupt is triggered, the Open Alliance recommends in [2] to have the host get a valid footer to determine if any action, other than receiving data must be taken.

If the host finds the EXT flag set in the footer, it will have to determine the kind of error or exception it needs to handle. To ascertain the specific issue reported the host can then read the SPISTATUS0 register.

From this, the host can extract 10 different error conditions that we will discuss below.

The status register can identify the following errors and exceptions:

Symbol	Description
CDPE	Control Data Protection Error Only if the SPI protocol is configured to operate in control command Protection mode (see [2]), this would indicate that the last control command was corrupted.
TXFCSE	Transmit FCS Error. Only if MACPHY is configured to expect the FCS from the host, then this error indicates a corrupt Ethernet frame has been received from the host via SPI
PHYINT	Exception detected by the integrated PHY inside NCN26010 A host will have to collect more data to determine the root cause
HDRE	Header Error. Indicates a parity check error in the last data or control header sent by the host.
RESETC	Not a real error. RESETC is asserted every time the NCN26010 has completed a reset. This is an indication for the host to check and resend the configuration data to NCN26010
LOFE	LOSS of framing. Indicates that a transaction did not complete due to CS being de−asserted before the transaction (either data or control) was completely transferred
RXBOE	Receive buffer overflow Error. the 4kbyte receive buffer inside the NCN26010 is full. This means data loss.
TXBUE	Transmit buffer underflow Error. Only happens in "cut through" mode, when the host sends data at a slower pace than required by the MACPHY to maintain a steady dataflow while transmitting an Ethernet frame.
TXBOE	Transmit buffer overflow error. Host was trying to send transmit data to the NCN26010 MACPHY although it's TX buffer is full.
TXPE	SPI Transmit protocol Error TX Data Chunk protocol Error in header. Either of: DV=1 without previous SV=1 Repeated SV=1 without EV in between. SWO or EBO point to a location that is beyond the length of the data chunk.

Handling Control Data Protection Error

When the SPI Status 0 register (MMS 0, address 0x0008) has bit 12 set, it indicates that a Data protection error occurred during the SPI transfer of a control command transaction. This can only be reported when running in the control command protection mode (see [2] for details), where each 32−bit data word is followed by a copy of its 1's complement.

When this error occurs, the NCN26010 ignores the control command. The Host should re−send the control command in this case. If these errors happen repeatedly or frequently, it is an indication that, either:

- The SPI speed is too fast
- The load on the SPI pins is too high
- The distance between the SPI master (host) and the Slave (NCN26010) is too long for the selected SPI speed.
- Excessive noise on the PCB.

In such case, users should verify their PCB design and make modifications accordingly or set the SPI speed to a lower frequency.

Handling TXFCS Error

When the NCN26010 is configured to expect an FCS from the host along with the Ethernet frame data (FCSA bit of MACCONTROL0 register set to 0), it will compare the incoming FCS (frame check sequence) with the one it internally calculated. If it finds a mismatch, it issues the TXFCS Error, informing the host that the last Ethernet

Frame it received from the SPI interface is corrupted and will not be sent on the single pair line to other stations. (discarded frame).
The host should resend the corrupt frame.

This error, when occurring frequently, could be an indication that either:

- The SPI speed is too high, or
- The load on the SPI pins it too high, or
- The SPI pins are too noisy.

Look for signal integrity issues on the SPI interface and verify the timing on the SPI interface.

Handling Header Error

A header error indicates a wrong Parity bit or any other bit in the header of a control or data transfer over SPI to have inadvertently transferred inverted.

If this happens in a control data transmission, then the host could simply resend the data. In control data transactions, this will be detectable before the actual interrupt is triggered, because the echoed control header will be receiving 0x400000000 in this case. When the host software can react on the echoed control header and detect the error that way, it can ignore the interrupt.

When a Header error occurs, the MACPHY ignores all further data until the CSn pin is de−asserted.

A host could either try to resend the frame or rely on protocols above layer 2 to eventually re−send the frame (TCP connection would do this when they do not get a frame acknowledged) or entirely ignore this situation and accept the loss of data.

Like with the previous error types, users are encouraged to check their software and hardware implementation with respect to the SPI interface. Root cause could be too tight interface timing, too fast interface speed, too high loads on the SPI interface, or other various issues.

Handling RESET Complete Event

When the host extracts a RESETC condition from the STAUTS0 register, it does not necessarily point toward an error in communication.

RESET Complete gets triggers every time the device has completed a reset and is ready for operation.

This happens, for instance, when the device is powered−up or when the RSTn is asserted by the host.

This could also happen when NCN26010 is operated on an instable power supply and the operating voltage dropped below the brown−out threshold, causing the MACPHY to reset.

In such a case, the host should reconfigure the MACHPY. All traffic that was in transit as well at the data stored in the RX and TX buffers is lost.

Handling Loss of Framing Error (LOFE)

A loss of framing error can point to two issues:

1. A signal integrity issue on the SPI interface between the host and NCN26010
2. Software running on the host is not respecting the OA−TC6 SPI protocol.

In both cases the device may lose data. LOFE happens when the CSn (or SS, slave select) gets de−asserted before the end of a transaction on the SPI interface.

To debug this issue, users should monitor the CS line to see that there are no glitches.

Also, the software routines should be checked for adherence to the OA−SPI specification (see [2]).

When detecting a LOFE the host could also try to resend the data.

Handling RXBO Error

As implied by its name, this flag gets set when the RX buffer of NCN26010 overflows.

In such case, there will be RX Ethernet frame data lost.

To prevent this from happening, the host needs to run the SPI fast enough to be able to cope with 10Mbit of constant data traffic. Please see the graph in the “Concept of Data Chunks” sections for guidance.
Also, the host should handle RX traffic with priority.

Note that NCN26010 has no mechanism to flush buffers in case of an overflow situation.

If the host sees the overflow, it is good practice to stop TX traffic and empty the RX buffer as quickly as possible.

Handling TXBO Errors

Transmit buffer overflow errors happen when the host connected to NCN26010 is sending TX data faster than what NCN26010 can send on the line. Remember that the 10Base−T1S is a multidrop segment technology that has multiple—in some situations up to 40—potential transmitters on a single bus.

The simplest way to avoid this error from happening is to check the status of the transmit buffer’s number of chunks that are available for writing. If the host has an Ethernet frame that is longer than the number of available chunks in the TX−buffer, then it should wait for the number of buffers needed to store such frame to become available before trying to send the frame to the NCN26010 MACPHY.

On occurrence of this error, the part has already lost data. In most of the cases that will be the last Ethernet frame sent, so when this situation occurs, the host could vote for waiting until the buffer has enough space available and resend the last Ethernet frame or rely on upper layer mechanism to handle re−transmission.

Hosts can easily keep track of the available buffers by looking at the TXC field inside the data footer that is received with every data chunk sent or read in the buffer status register.

In a good implementation, TX Buffer overflows should never happen.

Handling TX Buffer Underrun (TXBU) Errors

A TXBU error gets flagged when the NCN26010 is set to operate the TX in cut though mode.

Cut though mode can help with the latency of sent Ethernet frames but the host needs to be fast enough to send TX data at 10Mbit/sec or faster to not risk the TX buffer be empty before the full frame has been sent to the Ethernet medium.

When TXBU errors occur, at least the last Ethernet frame (basically the ongoing transfer) will be lost.

TXBU can be completely avoided when NCN26010 is set to store and forward mode.

When the application relies on cut though operation, the host needs to be able to send Ethernet frame data fast enough at any time. The required SPI frequency heavily depends on the average size of the Ethernet frame being sent.

Handling TXPE Errors

Transmit Protocol Errors are another class of error that users of the NCN26010 are likely to see during software development only. In an application ready to be deployed, TXPE errors should never appear, except a data chunk was rejected (due to a parity error) and not resent.

TXPE gets triggered in case of violation of the basic TC6 protocol specifications. They typically occur when:

- NCN26010 sees a start valid without a previous end valid flag in the header (only exception is the first data transfer after setting the SYNC bit)
- DV = 1 without a previous SV=1
- EV= 1 without a previous SV=1
- SWO or EBO pointing to locations beyond the length of a data chunk

When these errors happen, customers should look at their software implementation and correct the flow of data to be in line with the OA−SPI Protocol (see [2])

Handling PHYINT Error

Additional steps need to be taken, when the MACPHY signals a PHYINT event.

To determine the actual cause of the event, the host, besides reading the SPISTATUS0 register, will also have to read the MIIM IRQ STATUS Register (MMS12, Address 0x0011).

Bits 0 to 5 report events that trigger PHYINT events. These are:

- Physical Collision
- PLCA Recovery
- Remote Jabber
- Local Jabber
- PLCA status change
- Link Status Change.

There is no definite recommendation on how to handle these events as most are informational and could point to misconfigured networks or issues with excessive noise.

Please consult the NCN26010 datasheet MMS12 register description for details of the MIIM interrupt conditions.

Summary

In this application note, we have outlined the basic use and configuration of the IEEE802.3cg 10Base−T1S Multi−Drop Ethernet MACPHY NCN26010. We also give guidance on required SPI speed, how to setup optional and proprietary features as well as some guidance on how to handle / avoid errors and exceptions.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Email Requests to: orderlit@onsemi.com

onsemi Website: www.onsemi.com

TECHNICAL SUPPORT
North American Technical Support:
Voice Mail: 1 800−282−9855 Toll Free USA/Canada
Phone: 011 421 33 790 2910

Europe, Middle East and Africa Technical Support:
Phone: 00421 33 790 2910
For additional information, please contact your local Sales Representative

◊