HMT Mini-Stack software

1.0.1

Generated by Doxygen 1.7.5.1

Tue Jan 28 2014 10:20:14

Contents

1 HMT Mini-stack software **1**

1.1 Stack Usage . . . 2

1.1.1 Compilation . . . 2

1.1.2 Hardware configuration for compilation . . . 3

1.1.3 Stack Configuration . . . 3

1.1.4 Sample Application . . . 5

2 Change History **7**

2.1 0.1.10 . . . 7

2.2 0.1.11 . . . 7

2.3 0.1.12 . . . 7

2.4 0.1.13 . . . 7

2.5 1.0.0 . . . 8

2.6 1.0.1 . . . 8

3 Class Index **9**

3.1 Class Hierarchy . . . 9

4 Class Index **11**

4.1 Class List . . . 11

5 File Index **13**

5.1 File List . . . 13

6 Class Documentation **15**

6.1 DebugPin Class Reference . . . 15

6.1.1 Detailed Description . . . 15

6.2 DefaultSsHandler Struct Reference . . . 15
6.2.1 Detailed Description . . . 16
6.3 DemoApp Class Reference . . . 16
6.3.1 Detailed Description . . . 17
6.3.2 Member Function Documentation . . . 17
6.3.2.1 run . . . 17
6.4 IoLink Class Reference . . . 18
6.4.1 Detailed Description . . . 19
6.4.2 Member Enumeration Documentation . . . 19
6.4.2.1 CycleTime . . . 19
6.4.2.2 DeviceDLMode . . . 20
6.4.2.3 DirectParamPage . . . 20
6.4.2.4 MSeqCapability . . . 20
6.4.2.5 ProcessDataIn . . . 20
6.4.2.6 ProcessDataOut . . . 21
6.4.2.7 RevisionId . . . 21
6.4.3 Member Function Documentation . . . 21
6.4.3.1 calculateChecksum . . . 21
6.4.3.2 decodeCycleTime . . . 21
6.4.3.3 encodeCycleTime . . . 22
6.5 PhyDriver< SpiSsHndlr > Class Template Reference . . . 22
6.5.1 Detailed Description . . . 25
6.5.2 Member Enumeration Documentation . . . 25
6.5.2.1 CfgRegister . . . 25
6.5.2.2 LedLevel . . . 25
6.5.3 Member Function Documentation . . . 25
6.5.3.1 registerRead . . . 25
6.5.3.2 registerReadBegin . . . 26
6.5.3.3 registerReadLast . . . 26
6.5.3.4 registerReadNext . . . 27
6.5.3.5 registerReadStatus . . . 27
6.5.3.6 registerReadWriteBegin . . . 28
6.5.3.7 registerReadWriteDone . . . 28
6.5.3.8 registerReadWriteNext . . . 29

6.5.3.9 registerWrite . . . 29
6.5.3.10 registerWriteBegin . . . 29
6.5.3.11 registerWriteBegin . . . 30
6.5.3.12 registerWriteDone . . . 30
6.5.3.13 registerWriteNext . . . 31
6.6 Spi Struct Reference . . . 31
6.6.1 Detailed Description . . . 32
6.6.2 Member Function Documentation . . . 32
6.6.2.1 rx . . . 32
6.6.2.2 tx . . . 32
6.6.2.3 txRx . . . 32
6.7 StackBase< T, PDI, PDO, SpiSsHndlr > Class Template Reference . . . 33
6.7.1 Detailed Description . . . 38
6.7.2 Member Enumeration Documentation . . . 38
6.7.2.1 CfgRegister . . . 39
6.7.2.2 Led . . . 39
6.7.2.3 LedLevel . . . 39
6.7.2.4 SioDriveMode . . . 39
6.7.3 Member Function Documentation . . . 40
6.7.3.1 canRunUserCode . . . 40
6.7.3.2 configurePhy . . . 40
6.7.3.3 getOdOctetCount . . . 40
6.7.3.4 ledLevel . . . 40
6.7.3.5 masterLost . . . 41
6.7.3.6 odRead . . . 41
6.7.3.7 odWrite . . . 41
6.7.3.8 parameterRead . . . 42
6.7.3.9 parameterWrite . . . 42
6.7.3.10 processInputData . . . 42
6.7.3.11 processOutputData . . . 42
6.7.3.12 registerRead . . . 42
6.7.3.13 registerReadBegin . . . 43
6.7.3.14 registerReadLast . . . 43
6.7.3.15 registerReadNext . . . 44

6.7.3.16 registerReadStatus . . . 44
6.7.3.17 registerReadWriteBegin . . . 45
6.7.3.18 registerReadWriteDone . . . 45
6.7.3.19 registerReadWriteNext . . . 46
6.7.3.20 registerWrite . . . 46
6.7.3.21 registerWriteBegin . . . 46
6.7.3.22 registerWriteBegin . . . 47
6.7.3.23 registerWriteDone . . . 47
6.7.3.24 registerWriteNext . . . 48
6.7.3.25 setIoLinkListen . . . 48
6.7.3.26 setLedLevel . . . 48
6.7.3.27 setSioActive . . . 49
6.7.3.28 setSioLevel . . . 49
6.7.3.29 setSioListen . . . 49
6.7.3.30 stack . . . 49
6.7.3.31 stackMode . . . 50
6.7.3.32 startCallbackTimer . . . 50
6.7.3.33 temperature . . . 50
6.7.3.34 validateFrameType . . . 50
6.7.3.35 void::TIMER0_COMPB_vect . . . 51
6.8 StackBase< T, PDI, PDO, SpiSsHndlr >::Parameter Struct Reference . . . 51
6.8.1 Detailed Description . . . 51
6.9 StackBase< T, PDI, PDO, SpiSsHndlr >::ProcessData< SIZE > Struct Template Reference . . . 51
6.9.1 Detailed Description . . . 52
6.10 StackMultiByte Class Reference . . . 52
6.10.1 Detailed Description . . . 57
6.10.2 Member Enumeration Documentation . . . 57
6.10.2.1 CfgRegister . . . 57
6.10.2.2 Led . . . 57
6.10.2.3 LedLevel . . . 58
6.10.2.4 SioDriveMode . . . 58
6.10.3 Member Function Documentation . . . 58
6.10.3.1 canRunUserCode . . . 58
6.10.3.2 configurePhy . . . 59

6.10.3.3 getOdOctetCount . . . 59
6.10.3.4 ledLevel . . . 59
6.10.3.5 masterLost . . . 59
6.10.3.6 odRead . . . 60
6.10.3.7 odWrite . . . 60
6.10.3.8 parameterRead . . . 60
6.10.3.9 parameterWrite . . . 60
6.10.3.10 processInputData . . . 61
6.10.3.11 processOutputData . . . 61
6.10.3.12 registerRead . . . 61
6.10.3.13 registerReadBegin . . . 61
6.10.3.14 registerReadLast . . . 62
6.10.3.15 registerReadNext . . . 62
6.10.3.16 registerReadStatus . . . 63
6.10.3.17 registerReadWriteBegin . . . 63
6.10.3.18 registerReadWriteDone . . . 64
6.10.3.19 registerReadWriteNext . . . 64
6.10.3.20 registerWrite . . . 64
6.10.3.21 registerWriteBegin . . . 65
6.10.3.22 registerWriteBegin . . . 65
6.10.3.23 registerWriteDone . . . 66
6.10.3.24 registerWriteNext . . . 66
6.10.3.25 setIoLinkListen . . . 67
6.10.3.26 setLedLevel . . . 67
6.10.3.27 setSioActive . . . 67
6.10.3.28 setSioLevel . . . 67
6.10.3.29 setSioListen . . . 68
6.10.3.30 stack . . . 68
6.10.3.31 stackMode . . . 68
6.10.3.32 startCallbackTimer . . . 68
6.10.3.33 temperature . . . 68
6.10.3.34 validateFrameType . . . 69
6.10.3.35 void::TIMER0_COMPB_vect . . . 69
6.10.4 Member Data Documentation . . . 69

6.10.4.1 MSEQ_CAPABILITY . . . 69
6.10.4.2 PHY_CFG . . . 69
6.11 StackSingleByte Class Reference . . . 70
6.11.1 Detailed Description . . . 75
6.11.2 Implementation details . . . 75
6.11.2.1 Rx UART operation . . . 75
6.11.2.2 Establish comms . . . 75
6.11.3 Member Enumeration Documentation . . . 76
6.11.3.1 CfgRegister . . . 76
6.11.3.2 Led . . . 76
6.11.3.3 LedLevel . . . 76
6.11.3.4 SioDriveMode . . . 77
6.11.4 Member Function Documentation . . . 77
6.11.4.1 canRunUserCode . . . 77
6.11.4.2 configurePhy . . . 77
6.11.4.3 getOdOctetCount . . . 77
6.11.4.4 ledLevel . . . 78
6.11.4.5 masterLost . . . 78
6.11.4.6 odRead . . . 78
6.11.4.7 odWrite . . . 79
6.11.4.8 parameterRead . . . 79
6.11.4.9 parameterWrite . . . 79
6.11.4.10 processInputData . . . 79
6.11.4.11 processOutputData . . . 80
6.11.4.12 registerRead . . . 80
6.11.4.13 registerReadBegin . . . 80
6.11.4.14 registerReadLast . . . 81
6.11.4.15 registerReadNext . . . 81
6.11.4.16 registerReadStatus . . . 82
6.11.4.17 registerReadWriteBegin . . . 82
6.11.4.18 registerReadWriteDone . . . 82
6.11.4.19 registerReadWriteNext . . . 83
6.11.4.20 registerWrite . . . 83
6.11.4.21 registerWriteBegin . . . 84

6.11.4.22 registerWriteBegin . . . 84
6.11.4.23 registerWriteDone . . . 85
6.11.4.24 registerWriteNext . . . 85
6.11.4.25 setIoLinkListen . . . 85
6.11.4.26 setLedLevel . . . 86
6.11.4.27 setSioActive . . . 86
6.11.4.28 setSioLevel . . . 86
6.11.4.29 setSioListen . . . 86
6.11.4.30 stack . . . 87
6.11.4.31 stackMode . . . 87
6.11.4.32 startCallbackTimer . . . 87
6.11.4.33 temperature . . . 87
6.11.4.34 validateFrameType . . . 88
6.11.4.35 void::TIMER0_COMPB_vect . . . 88
6.11.5 Member Data Documentation . . . 88
6.11.5.1 MSEQ_CAPABILITY . . . 88
6.11.5.2 PHY_CFG . . . 88
6.12 StackTransparent Class Reference . . . 89
6.12.1 Detailed Description . . . 94
6.12.2 Implementation details . . . 94
6.12.2.1 Rx UART operation . . . 94
6.12.2.2 Establish comms . . . 95
6.12.2.3 SPI communication . . . 95
6.12.3 Member Enumeration Documentation . . . 95
6.12.3.1 CfgRegister . . . 95
6.12.3.2 Led . . . 96
6.12.3.3 LedLevel . . . 96
6.12.3.4 SioDriveMode . . . 96
6.12.4 Member Function Documentation . . . 97
6.12.4.1 canRunUserCode . . . 97
6.12.4.2 configurePhy . . . 97
6.12.4.3 getOdOctetCount . . . 97
6.12.4.4 ledLevel . . . 97
6.12.4.5 masterLost . . . 98

6.12.4.6 odRead . . . 98
6.12.4.7 odWrite . . . 98
6.12.4.8 parameterRead . . . 99
6.12.4.9 parameterWrite . . . 99
6.12.4.10 processInputData . . . 99
6.12.4.11 processOutputData . . . 99
6.12.4.12 registerRead . . . 99
6.12.4.13 registerReadBegin . . . 100
6.12.4.14 registerReadLast . . . 100
6.12.4.15 registerReadNext . . . 101
6.12.4.16 registerReadStatus . . . 101
6.12.4.17 registerReadWriteBegin . . . 102
6.12.4.18 registerReadWriteDone . . . 102
6.12.4.19 registerReadWriteNext . . . 103
6.12.4.20 registerWrite . . . 103
6.12.4.21 registerWriteBegin . . . 103
6.12.4.22 registerWriteBegin . . . 104
6.12.4.23 registerWriteDone . . . 104
6.12.4.24 registerWriteNext . . . 105
6.12.4.25 setLedLevel . . . 105
6.12.4.26 setSioLevel . . . 105
6.12.4.27 stack . . . 106
6.12.4.28 stackMode . . . 106
6.12.4.29 startCallbackTimer . . . 106
6.12.4.30 temperature . . . 106
6.12.4.31 validateFrameType . . . 107
6.12.4.32 void::TIMER0_COMPB_vect . . . 107
6.12.5 Member Data Documentation . . . 107
6.12.5.1 MSEQ_CAPABILITY . . . 107
6.12.5.2 PHY_CFG . . . 107
6.13 TransparentModeSsHandler Class Reference . . . 108
6.13.1 Detailed Description . . . 108

7 File Documentation **109**

7.1 debugpin.h File Reference . . . 109
7.1.1 Detailed Description . . . 109
7.2 demoapp.h File Reference . . . 109
7.2.1 Detailed Description . . . 109
7.3 iolink.h File Reference . . . 110
7.3.1 Detailed Description . . . 110
7.3.2 Define Documentation . . . 110
7.3.2.1 DECODE_CYCLE_TIME . . . 110
7.3.2.2 ENCODE_CYCLE_TIME . . . 111
7.3.2.3 ENCODE_PD_BYTES . . . 111
7.4 phydriver.h File Reference . . . 112
7.4.1 Detailed Description . . . 112
7.4.2 Define Documentation . . . 112
7.4.2.1 ENCODE_THERMAL_SHUTDOWN . . . 112
7.4.2.2 PROGMEM_ . . . 113
7.5 spi.h File Reference . . . 113
7.5.1 Detailed Description . . . 113
7.6 stackbase.h File Reference . . . 113
7.6.1 Detailed Description . . . 114
7.6.2 Define Documentation . . . 114
7.6.2.1 BANKBYTE . . . 114
7.6.2.2 HIBYTE . . . 114
7.6.2.3 LOBYTE . . . 115
7.7 stackmultibyte.h File Reference . . . 115
7.7.1 Detailed Description . . . 115
7.8 stacksinglebyte.h File Reference . . . 115
7.8.1 Detailed Description . . . 116
7.9 stacktransparent.h File Reference . . . 116
7.9.1 Detailed Description . . . 116

Chapter 1

HMT Mini-stack software

Author

Roger Bostock at HMT microelectronics
Daniel Gehriger <gehriger@linkcad.com>

Date

28.01.14

Version

1.0.1

The Mini-stack software is a minimal IO-Link protocol stack implemented with the HMT PHY ICs and an Atmel ATMega328P micro controller.

The function of all operating modes of the PHY's (multi-byte, single-byte and transparent), all operating frequencies (COM2 and COM3) and all SIO drive modes (NPN, PNP, Push-pull and Inactive) have been demonstrated with a range of M-sequences with a device tester.

Please refer to the Change History for modifications between the software versions.

This Mini-stack software is primarily designed to demonstrate the function of the HMT7742/HMT7748 PHY's, and to provide a reference design for their use. It *is* intended that users inspect the internals of the code and understand how it functions. The stack is deliberately cut to a minimum number of lines to give users a chance to follow the code. In particular, there is no software support for:

- Events
- ISDU's

The stack is optimised to run efficiently on the microcontroller, and care has been taken to avoid run-time overhead in the interrupt service routines. This demonstrates how the HMT7742/HMT7748 PHY's can be used to reduce the load on the micro-controller (MHz, mA and kBytes)

A demonstration application, see DemoApp, is supplied with the Mini-stack software to facilitate a rapid development start in association with one of the development boards (TM96.1 var. A or B, TM141.0 or TM142.0)

1.1 Stack Usage

Three stack implementations are available, all derived from the templated StackBase base class:

- StackMultiByte: Stack implementation using Multi-Byte mode.
- StackSingleByte: Stack implementation using Single-Byte mode.
- StackTransparent: Stack implementation using Transparent mode.

All implementations share the same API, and a typedef `Stack` is defined as an alias for the selected stack implementation (see Compilation). The stack instance is made available as `Stack::instance`.

The user-provided cyclic code runs inside the application's main loop. It should repeatedly call Stack::instance.canRunUserCode() to determine if it may perform any lengthy calculations or otherwise access the stack. This function will return `true` once per IO-Link communication cycle, **even in the absence of IO-Link communication** when it is set to `true` once per minimum cycle length . When the stack is communicating, this function will run between two sequences, just after the completion of the device response.

Attention

The timer 0 interrupts are important for the stack to track the communication timing, and should not be blocked for more than 50us (max. latency). The timer interrupt routine does not affect the data presented by the stack to the user cyclic code, but may affect the reported operating mode.

Attention

In SIO operation, a PHY data interrupt may arrive at any time and should not be blocked for more than 130us (38.4kBaud operation) or 10us (230.4kBaud operation). The stack is idle in this mode, and will not update or use the process data or write parameter fields returned by Stack::canRunUserCode().

Attention

In established IO-Link operation, the PHY interrupt (only) should be blocked during the operation of the user cyclic code. No data interrupt which requires rapid processing will be received in this time.

Attention

The user cyclic code must complete in the gap between the last device transmission and the end of the master transmission (multi-byte exchange) or between the last device transmission and the start of master transmission (single-byte and transparent exchange)

1.1.1 Compilation

In order to select a stack implementation, include the corresponding stack header file from your application code and compile and link the stack source file.

The code in the stack source files will only be compiled if a corresponding preprocessor symbol has been defined, as shown below. This allows a project file to include all stack implementations and to select the desired stack type by defining the corresponding preprocessor symbol.

- Using the **Multi-Byte** (Io-Link) Mode Stack:
  - compile, and link with, "stack/stackmultibyte.cpp" with STACK_MODE_MULTI_BYTE defined.
  - include the header file "stack/stackmultibyte.h":

```
#include "stack/stackmultibyte.h"
```

- Using the **Single-Byte** Mode Stack:
  - compile, and link with, "stack/stacksinglebyte.cpp" with STACK_MODE_SINGLE_BYTE defined.
  - include the header file "stack/stacksinglebyte.h":

```
#include "stack/stacksinglebyte.h"
```

- Using the **TransparentStack** Mode Stack:
  - compile, and link with, "stack/stacktransparent.cpp" with STACK_MODE_TRANSPARENT defined.
  - include the header file "stack/stacktransparent.h":

```
#include "stack/stacktransparent.h"
```

1.1.2 Hardware configuration for compilation

Multi-byte and single-byte modes are insensitive to the CPU clock frequency, and the internal RC oscillator running at 8MHz is used by default. The define F_CPU=8000000UL should be set to achieve this.

Transparent mode requires an external oscillator running at 18.432MHz and the define F_CPU=18432000UL should be set for this case. The additional define 'USE_EXT_OSC' should be set to enable the correct clock source, which is set by the ATMega fuses.

The correct PHY type, either HMT7742 or HMT7748, must be selected as a preprocessor symbol define. Define either 'USE_HMT7742' to use the HMT7742 PHY, or 'USE_HMT7748' to use the HMT7748 PHY.

The brown-out detector is enabled at the nominal 1.8V level, using the ATMega fuses. The ATmega shows correct behaviour at this level even under conditions where the supply connection has considerable contact bounce.

The ATMega fuse codings are embedded in the source code, and contained in the ".elf" file produced.

1.1.3 Stack Configuration

The selected stack is configured by adjusting the public class constants in its header file, as shown below for the multi-byte mode stack:

```
class StackMultiByte : public StackBase<StackMultiByte,
        1, // PD_IN_SIZE
        1  // PD_OUT_SIZE
>
{
public:
    static const uint8_t REVISION_ID =      IoLink::REVISION_ID_1_1;
    static const uint16_t VENDOR_ID =       0x01a6; // HMT
```

```
    static const uint32_t DEVICE_ID =       0x123457;
    static const uint32_t BAUD_RATE =       230400;
    static const uint8_t MIN_CYC_TIME =     30;  // 30x0.1ms
    static const uint8_t MSEQ_CAPABILITY =  IoLink::MSEQCAP_ISDU_NOT_SUPPORTED
      |
                                            IoLink::MSEQCAP_OP_CODE_0 |
                                            IoLink::MSEQCAP_PREOP_CODE_0;
    static const uint8_t PHY_CFG =          CFG_UVT_16_3V | CFG_RF_ABS |
     CFG_S5V_3_3V
                                | (BAUD_RATE == 38400 ? CFG_BD_38400 :
     CFG_BD_230400);
    static const uint8_t PHY_CTL_SCT =      CTL_SCT_190MA;
    static const uint8_t PHY_CTL_MODE =     CTL_IOLINK_MODE;
    static const enum SioDriveMode SIO_DRIVE_MODE = DRIVE_MODE_PUSH_PULL;
    static const uint8_t PHY_THERM_DEG =    175; // ~175 degrees Celsius
// [...]
};
```

Most configuration parameters should be obvious and this documentation mentions the reference to the appropriate standard / data sheet documentation.

The **sequence size** and type is determined by the `PD_IN_SIZE` and `PD_OUT_SIZE` template parameters, as well as the class constant `MSEQ_CAPABILITY`, as described in the *"IO-Link Interface and System Specification"*, V1.1.1, section B.1.5.

The stacks all currently implement sequences TYPE_0, TYPE_2_1 through TYPE_2_5 on a Atmega328p with a 8MHz system clock. Larger sequence sizes are also supported at this frequency when using single-byte mode, but other modes may require a higher clock frequency.

All modes both have been tested for the following sequence types at COM2 (38.4kBaud) and COM3 (230.4kBaud):

- TYPE_0
- TYPE_2_1 through TYPE_2_5

Attention

Transparent mode requires an external oscillator running 18.432 MHz and the connection of jumpers on the GENIE Explorer boards to connect the ATMega hardware UART pins TXD (to MOSI) and RXD (to MISO).

Attention

The PHY permanently drives the MISO line in transparent mode, and this interferes with programming of the microcontroller on the SPI interface. For development, a 470ohm resistor may be inserted between the PHY MISO and micro-controller SPI input, to prevent contention when programming. Alternatively delay in startup (setting the PHY to transparent mode) may be added to provide a window to start programming the device, or the debugWIRE may be used for programming.

Attention

Sequence TYPE_1_1/1_2 (interleaved) is not supported

1.1.4 Sample Application

A typical sample application looks like this:

```
#include "stack/stackmultibyte.h"
#include <avr/sleep.h>

void user_configure();
void user_run(const Stack::Parameter* param);

int main(void)
{
    // configure all software modules
    Stack::instance.configure();

    // configure user code
    user_configure();

    // enable interrupts
    sei();

    // select sleep mode
    set_sleep_mode(SLEEP_MODE_IDLE);

    // enter infinite loop: processing is interrupt controlled from now on
    for (;;)
    {
        // enter sleep until interrupt wakes us up
        sleep_mode();

        // check if it's time to run user code
        const Stack::Parameter* paramWrite;
        if (Stack::instance.canRunUserCode(paramWrite))
        {
            Stack::instance.stopInterrupt();
            user_run(paramWrite);
            Stack::instance.restartInterrupt();
        }
    }
}

void user_run(const Stack::Parameter* param)
{
    // check for write access to direct parameter page
    if (param)
    {
        // handle parameter write access
        // [...]
    }
    else if (Stack::instance.stackMode() == Stack::STACK_MODE_SIO)
    {
         // handle SIO mode
         // [...]
    }

    // update process data
    // [...]
}
```

Chapter 2

Change History

2.1 0.1.10

- Initial release

2.2 0.1.11

- minimised update delay in SIO mode
- initialisation of _parameterWrite address corrected
- multiple process and on-demand octets in all stacks
- implemented non-response to incorrect M-sequence types
- implemented SIO_DRIVE_MODE
- implemented SIO switch to IO-Link listening after delay if HiZ
- implemented masterLoss indication

2.3 0.1.12

- IODD file versions 1.01 (for IO-Link 1.0) are provided

2.4 0.1.13

- transparent mode uses the ATMega hardware internal UART
- updated IODD file versions 1.01 (for IO-Link 1.0) and 1.1 (for IO-Link 1.1) are provided

2.5 1.0.0

Release date: 11.04.13

- testing carried out with IO-Link master for v1.1 and v1.0
- corrections to improve EMC performance
- AVR6 support included

2.6 1.0.1

Release date: 28.01.14

- prefix StackBase:: added to SPI accesses in Stackbase.h, to satisfy modified AVR compiler requirement

Chapter 3

Class Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

DebugPin . . . 15
DefaultSsHandler . . . 15
  TransparentModeSsHandler . . . 108
DemoApp . . . 16
IoLink . . . 18
PhyDriver< SpiSsHndlr > . . . 22
  StackBase< T, PDI, PDO, SpiSsHndlr > . . . 33
    StackMultiByte . . . 52
    StackSingleByte . . . 70
    StackTransparent . . . 89
PhyDriver< DefaultSsHandler > . . . 22
  StackBase< StackMultiByte, 1, 1 > . . . 33
  StackBase< StackSingleByte, 1, 1 > . . . 33
PhyDriver< TransparentModeSsHandler > . . . 22
  StackBase< StackTransparent, 1, 1, TransparentModeSsHandler > . . . 33
StackBase< T, PDI, PDO, SpiSsHndlr >::ProcessData< PDI > . . . 51
StackBase< T, PDI, PDO, SpiSsHndlr >::ProcessData< PDO > . . . 51
Spi . . . 31
StackBase< T, PDI, PDO, SpiSsHndlr >::Parameter . . . 51
StackBase< T, PDI, PDO, SpiSsHndlr >::ProcessData< SIZE > . . . 51

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

DebugPin
: Class to control pins on JP4 extension connector . . . 15

DefaultSsHandler
: Default implementation of class for handling SPI SS/ line . . . 15

DemoApp
: The DemoApp is a demonstration application which uses the Mini-stack software . . . 16

IoLink
: Helper class for supporting IO-Link standard . . . 18

PhyDriver< SpiSsHndlr >
: Static class implementing register access to the PHY . . . 22

Spi
: Helper class for handling SPI communication with the HMT7742 . . . 31

StackBase< T, PDI, PDO, SpiSsHndlr >
: The StackBase is the base class for minimal IO-Link stacks for different IO-Link devices . . . 33

StackBase< T, PDI, PDO, SpiSsHndlr >::Parameter
: Parameter structure . . . 51

StackBase< T, PDI, PDO, SpiSsHndlr >::ProcessData< SIZE >
: Structure for holding process data and associated status flags . . . 51

StackMultiByte
: Stack implementation using multi-byte mode . . . 52

StackSingleByte
: Stack implementation using the PHY single-octet mode . . . 70

StackTransparent
: Stack implementation using the PHY transparent mode . . . 89

TransparentModeSsHandler
: Implementation of class for handling SPI SS/ line with TX save/restore . . . 108

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

debugpin.h
Declares the DebugPin class . . . 109
demoapp.h
Declares the DemoApp class . . . 109
iolink.h
Declares the IO-Link class . . . 110
phydriver.h
Declares the PhyDriver . . . 112
spi.h
Declares the Spi class . . . 113
stackbase.h
Declares the StackBase class . . . 113
stackmultibyte.h
Declares the StackMultiByte class . . . 115
stacksinglebyte.h
Declares the StackSingleByte class . . . 115
stacktransparent.h
Declares the StackTransparent class . . . 116

Chapter 6

Class Documentation

6.1 DebugPin Class Reference

Class to control pins on JP4 extension connector.

`#include <debugpin.h>`

Static Public Member Functions

- static void configure ()
  setup the HW configuration
- static void set (bool level)
  Control debug pin PD4.
- static void toggle (int8_t count)
  Toggle pin.
- static void set2 (bool level)
  Control debug pin PD4.

6.1.1 Detailed Description

Class to control pins on JP4 extension connector.

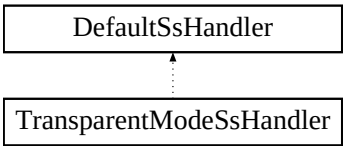
This class is used for debugging purposes only. It controls the pin PD4 (JP6:1) and PD5 (JP6:2)

6.2 DefaultSsHandler Struct Reference

Default implementation of class for handling SPI SS/ line.

`#include <spi.h>`

Inheritance diagram for DefaultSsHandler:



Static Public Member Functions

- static void assert ()

  Assert SS/ to begin SPI communication.
- static void deassert ()

  Deassert SS/ to terminate SPI communication.
- static bool asserted ()

  Test if SS/ asserted.

6.2.1 Detailed Description

Default implementation of class for handling SPI SS/ line.

6.3 DemoApp Class Reference

The DemoApp is a demonstration application which uses the Mini-stack software.

`#include <demoapp.h>`

Public Member Functions

- void run (const Stack::Parameter *param)

  This is the main loop, which is called by the stack after a frame has been received.

Static Public Member Functions

- static void configure ()

  Configure the DemoApp class.

Static Public Attributes

- static DemoApp instance

  The one and only instance of this class.

6.3.1 Detailed Description

The DemoApp is a demonstration application which uses the Mini-stack software.

The demonstration application and its related IODD file is supplied with the Mini-stack software to facilitate a rapid development start in assosiation with one of the development boards (TM96.1 var. A or B, TM141.0 or TM142.0)

The related IODD file for this application can be found in directory IODD, files HMT-Mini_stack_38kB-20120727-IODD1.0.xml for 38.4kBd variants, and HMT-Mini_stack_230kB-20120727-IODD1.0.xml for 230.4kBd variants.

The demonstration application may be compiled with any stack mode, multi-octet, single-octet or transparent.

The board LEDs indicate the status:

- in SIO-mode the red LED alternately brightens and dims
- in IO-Link operation the green LED alternately brightens and dims
- BUT, the red LED lights permanently if the push-button is pressed.

address	direct parameter	comment
0x10	VendorParamMirrorOutput	value read from device
0x11	VendorParamMirrorInput	value sent to device
0x12	VendorParamPidMode	selects value for process data

Direct Parameter Memory map

Basic data exchange on the direct parameter page is demonstrated. A value written in direct parameter VendorParamMirrorOutput will subsequently be read in VendorParamMirrorInput.

The process data can have different contents depending on the value set in direct parameter VendorParamPidMode.

setting in VendorParamPidMode	comment	interpretation
0x00	1 bit representing the push-button state	'1' => pressed
0x01	8 bit octet for the potentiometer setting	
0x02	8 bit internally generated saw-tooth value	

Process data selection

6.3.2 Member Function Documentation

6.3.2.1 void DemoApp::run (const Stack::Parameter * *param*)

This is the main loop, which is called by the stack after a frame has been received.

Parameters

param	Pointer to Parameter structure if the most recent message completed a write access to the direct parameter page. The data is *not* automatically written to the direct parameter page, but needs to be manually committed by calling parameterWrite(). This parameter may be NULL if no write access occurred.

6.4 IoLink Class Reference

Helper class for supporting IO-Link standard.

`#include <iolink.h>`

Public Types

- enum MSeqCtrl { **MC_ADDRESS_MASK** = 0x1f, **MC_CHANNEL_MASK** = 0x60, **MC_RW_MASK** = 0x80, **MC_ADDR_ISDU_COUNT_MASK** = 0x0f, **MC_ADDR_ISDU_START** = 0x10, **MC_ADDR_ISDU_IDLE1** = 0x11, **MC_ADDR_ISDU_IDLE2** = 0x12, **MC_ADDR_ISDU_ABORT** = 0x1f, **MC_CHNL_PROC** = 0 << 5, **MC_CHNL_PAGE** = 1 << 5, **MC_CHNL_DIAG** = 2 << 5, **MC_CHNL_ISDU** = 3 << 5, **MC_WRITE** = 0 << 7, **MC_READ** = 1 << 7 }

  M-sequence control (MC) octet.
- enum MSeqCkt { **CKT_CHECKSUM_MASK** = 0x3f, **CKT_TYPE_MASK** = 0xc0, **CKT_TYPE_0** = 0 << 6, **CKT_TYPE_1** = 1 << 6, **CKT_TYPE_2** = 2 << 6, **CKT_TYPE_NONE** = 3 << 6 }

  Checksum / M-sequence type (CKT) octet.
- enum MSeqCks { **CKS_CHECKSUM_MASK** = 0x3f, **CKS_PD_STATUS_MASK** = 0x40, **CKS_EVENT_FLAG_MASK** = 0x80, **CKS_PD_VALID** = 0 << 6, **CKS_PD_INVALID** = 1 << 6, **CKS_NO_EVENT** = 0 << 7, **CKS_EVENT** = 1 << 7 }

  Checksum / status (CKS) octet.
- enum DirectParamPage { **PAGE_MASTER_CMD** = 0x00, **PAGE_MASTER_CYCLE_TIME** = 0x01, **PAGE_MIN_CYCLE_TIME** = 0x02, **PAGE_MSEQ_CAPABILITY** = 0x03, **PAGE_REVISION_ID** = 0x04, **PAGE_PD_IN** = 0x05, **PAGE_PD_OUT** = 0x06, **PAGE_VENDOR_ID_1** = 0x07, **PAGE_VENDOR_ID_2** = 0x08, **PAGE_DEVICE_ID_1** = 0x09, **PAGE_DEVICE_ID_2** = 0x0a, **PAGE_DEVICE_ID_3** = 0x0b, **PAGE_FUNCTION_ID_1** = 0x0c, **PAGE_FUNCTION_ID_2** = 0x0d, **PAGE_RESERVED** = 0x0e, **PAGE_SYSTEM_CMD** = 0x0f, **PAGE_DEVICE_SPECIFIC_10** = 0x10, **PAGE_DEVICE_SPECIFIC_11** = 0x11, **PAGE_DEVICE_SPECIFIC_12** = 0x12, **PAGE_DEVICE_SPECIFIC_13** = 0x13, **PAGE_DEVICE_SPECIFIC_14** = 0x14, **PAGE_DEVICE_SPECIFIC_15** = 0x15, **PAGE_DEVICE_SPECIFIC_16** = 0x16, **PAGE_DEVICE_SPECIFIC_17** = 0x17, **PAGE_DEVICE_SPECIFIC_18** = 0x18, **PAGE_DEVICE_SPECIFIC_19** = 0x19, **PAGE_DEVICE_SPECIFIC_1A** = 0x1a, **PAGE_DEVICE_SPECIFIC_1B** = 0x1b, **PAGE_DEVICE_SPECIFIC_1C** = 0x1c, **PAGE_DEVICE_SPECIFIC_1D** = 0x1d, **PAGE_DEVICE_SPECIFIC_1E** = 0x1e, **PAGE_DEVICE_SPECIFIC_1F** = 0x1f, **PAGE_NO_PARAMETER** = 0xff }

  Page 1 ranges from 0x00 to 0x0F.
- enum MasterCommand { **MCMD_FALLBACK** = 0x5a, **MCMD_MASTER_IDENT** = 0x95, **MCMD_DEVICE_IDENT** = 0x96, **MCMD_DEVICE_STARTUP** = 0x97, **MCMD_PD_OUT_OPERATE** = 0x98, **MCMD_DEVICE_OPERATE** = 0x99, **MCMD_DEVICE_PREOPERATE** = 0x9a }

  The Master application is able to check the status of a Device or to control its behavior with the help of MasterCommands.
- enum CycleTime { **CYC_MULTIPLIER_MASK** = 0x3f, **CYC_TIME_BASE_MASK** = 0xc0, **CYC_TIME_BASE_0_1_MS** = 0 << 6, **CYC_TIME_BASE_0_4_MS** = 1 << 6, **CYC_TIME_BASE_1_6_MS** = 2 << 6 }

  MasterCycleTime and MinCycleTime.
- enum MSeqCapability { **MSEQCAP_ISDU_MASK** = 0x01, **MSEQCAP_OP_MASK** = 0x0e, **MSEQCAP_PREOP_MASK** = 0x30, **MSEQCAP_ISDU_SUPPORTED** = 1 << 0, **MSEQCAP_ISDU_NOT_SUPPORTED** = 0 << 0, **MSEQCAP_OP_CODE_0** = 0 << 1, **MSEQCAP_OP_CODE_1** = 1 << 1, **MSEQCAP_OP_CODE_4** = 4 << 1, **MSEQCAP_OP_CODE_5** = 5 << 1, **MSEQCAP_OP_CODE_6** = 6 << 1, **MSEQCAP_OP_CODE_7** = 7 << 1, **MSEQCAP_PREOP_CODE_0** = 0 << 4, **MSEQCAP_PREOP_CODE_1** = 1 << 4, **MSEQCAP_PREOP_CODE_2** = 2 << 4, **MSEQCAP_PREOP_CODE_3** = 3 << 4 }

M-sequence Capability.

- enum RevisionId { **REVISION_ID_1_0** = 0x10, **REVISION_ID_1_1** = 0x11 }

  Revision ID (RID)

- enum ProcessDataIn { **PDIN_LENGTH_MASK** = 0x1f, **PDIN_SIO_SUPPORTED** = 1 << 6, **PDIN_SIO_NOT_SUPPORTED** = 0 << 6, **PDIN_BITS** = 0 << 7, **PDIN_BYTES_PLUS_1** = 1 << 7 }

  ProcessDataIn parameter.

- enum ProcessDataOut { **PDOUT_LENGTH_MASK** = 0x1f, **PDOUT_BITS** = 0 << 7, **PDOUT_BYTES_PLUS_1** = 1 << 7 }

  ProcessDataOut parameter.

- enum DeviceDLMode { DDL_MODE_IDLE, DDL_MODE_ESTABLISH_COM, DDL_MODE_STARTUP, DDL_MODE_PREOPERATE, DDL_MODE_OPERATE }

  Device DL-mode.

Public Member Functions

- uint16_t decodeCycleTime (uint8_t cycleTimeParam)

  Decode the MasterCycleTime or MinCycleTime parameter into 0.1ms units.

- uint8_t encodeCycleTime (uint16_t cycleTimeDeciMs)

  Encode desired cycle time into MasterCycleTime or MinCycleTime parameter.

Static Public Member Functions

- static uint8_t calculateChecksum (uint8_t checksum8, uint8_t ckt)

  Calculates CKT / CKS octet.

6.4.1 Detailed Description

Helper class for supporting IO-Link standard.

6.4.2 Member Enumeration Documentation

6.4.2.1 enum IoLink::CycleTime

MasterCycleTime and MinCycleTime.

Recommended cycle times (based on frame type 2.1)

- COM1: 18.0 ms
- COM2: 2.3ms
- COM3: 0.4ms

6.4.2.2 enum IoLink::DeviceDLMode

Device DL-mode.

See also

IO-Link Interface and System Specification V1.1.1, 7.3.2.5

Enumerator:

DDL_MODE_IDLE Stack in SIO mode.

DDL_MODE_ESTABLISH_COM Stack in ESTABLISH_COM mode.

DDL_MODE_STARTUP Stack in STARTUP mode.

DDL_MODE_PREOPERATE Stack in PREOPERATE mode.

DDL_MODE_OPERATE Stack in OPERATE mode.

6.4.2.3 enum IoLink::DirectParamPage

Page 1 ranges from 0x00 to 0x0F.

It comprises the following categories of parameters:

- Communication control
- Identification parameter
- Application control

6.4.2.4 enum IoLink::MSeqCapability

M-sequence Capability.

See also

IO-Link Interface and System Specification, V1.1.1, B.1.5.

6.4.2.5 enum IoLink::ProcessDataIn

ProcessDataIn parameter.

See also

IO-Link Interface and System Specification, V1.1.1, B.1.7

6.4.2.6 enum IoLink::ProcessDataOut

ProcessDataOut parameter.

See also

IO-Link Interface and System Specification, V1.1.1, B.1.8

6.4.2.7 enum IoLink::RevisionId

Revision ID (RID)

The RevisionID parameter is the two-digit version number of the SDCI protocol implemented within the Device.

Note

The RevisionID can be overwritten (see 10.6.3). An accepted different RevisionID shall be volatile.

6.4.3 Member Function Documentation

6.4.3.1 static uint8_t IoLink::calculateChecksum (uint8_t *checksum8,* uint8_t *ckt*) [inline, static]

Calculates CKT / CKS octet.

Takes the XOR'ed octets of a message **excluding the CKT or CKS itself** and returns the corresponding CKT / CKS octet.

Warning

Do NOT include the seed value in the `checksum8`!

Parameters

checksum8	Result of XORing all message octets, except CKT/CKS
ckt	Input CKT / CKS octet (only bits 6 and 7 are used)

Returns

CKT / CKS octet with Checksum bits updated

See also

IOL Interface Specs V1.1.1, A.1.6

6.4.3.2 uint16_t IoLink::decodeCycleTime (uint8_t *cycleTimeParam*) [inline]

Decode the MasterCycleTime or MinCycleTime parameter into 0.1ms units.

Parameters

cycleTimeParam	Value of MasterCycleTime or MinCycleTime parameter

Returns

Cycle time in 1/10 of ms, or 0 if parameter invalid

6.4.3.3 uint8_t IoLink::encodeCycleTime (uint16_t *cycleTimeDeciMs*) [inline]

Encode desired cycle time into MasterCycleTime or MinCycleTime parameter.

The resulting parameter may not correspond exactly to the desired input value. Use `decodeCycleTime` to obtain the exact value.

See also

`ENCODE_CYCLE_TIME()`

Parameters

cycleTimeDeciMs	Cycle time in 1/10 of ms

Returns

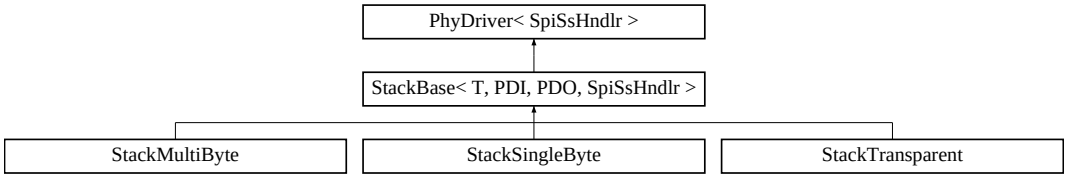
Parameter value, or 0 if cycle time out of valid range

6.5 PhyDriver< SpiSsHndlr > Class Template Reference

Static class implementing register access to the PHY.

`#include <phydriver.h>`

Inheritance diagram for PhyDriver< SpiSsHndlr >:



Public Types

- enum LedLevel { LED_LEVEL_OFF, LED_LEVEL_1, LED_LEVEL_2, LED_LEVEL_3, LED_LEVEL_4, LED_LEVEL_5, LED_LEVEL_6, LED_LEVEL_7, **LED_LEVEL_MAX** = LED_LEVEL_7, LED_LEVEL_INVALID }

  LED currents.

Protected Types

- enum MseqRegister { **MSEQ_M2CNT_SHIFT** = 2, **MSEQ_OD_1** = 0 << 0, **MSEQ_OD_2** = 1 << 0, **MSEQ_OD_8** = 2 << 0 }

  PHY MSEQ register flags.
- enum CfgRegister { **CFG_NONE** = 0, **CFG_UVT_18_0V** = 0 << 5, **CFG_UVT_16_3V** = 1 << 5, **CFG_UVT_15_0V** = 2 << 5, **CFG_UVT_13_9V** = 3 << 5, **CFG_UVT_12_0V** = 4 << 5, **CFG_UVT_10_0V** = 5 << 5, **CFG_UVT_8_6V** = 6 << 5, **CFG_UVT_7_2V** = 7 << 5, CFG_BD_38400 = 0 << 4, CFG_BD_230400 = 1 << 4, **CFG_RF_ABS** = 0 << 3, **CFG_RF_REL** = 1 << 3, **CFG_S5V_SS** = 0 << 0, **CFG_S5V_3_3V** = 2 << 0, **CFG_S5V_5_0V** = 3 << 0 }

  PHY CFG register flags.
- enum CtlRegister { **CTL_NONE** = 0, **CTL_TRNS_MODE** = 1 << 7, **CTL_SCT_190MA** = 0 << 4, **CTL_SCT_210MA** = 1 << 4, **CTL_SCT_230MA** = 2 << 4, **CTL_SCT_250MA** = 3 << 4, **CTL_SCT_110MA** = 4 << 4, **CTL_SCT_130MA** = 5 << 4, **CTL_SCT_150MA** = 6 << 4, **CTL_SCT_170MA** = 7 << 4, **CTL_SGL_MODE** = 1 << 3, **CTL_IEN_MODE** = 1 << 3, **CTL_IOLINK_MODE** = 0 << 2, **CTL_DIO** = 1 << 2, **CTL_JOIN** = 0 << 2, **CTL_SIO_MODE** = 1 << 2, **CTL_HS** = 1 << 1, **CTL_LS** = 1 << 0 }

  PHY CTL register flags.
- enum LinkRegister { **LINK_NONE** = 0, **LINK_CNT_MASK** = 0x3C, **LINK_CNT_SHIFT** = 2, **LINK_END** = 1 << 1, **LINK_SND** = 1 << 0 }

  PHY LINK register flags.
- enum StatusRegister { **STATUS_NONE** = 0, **STATUS_RST** = 1 << 7, **STATUS_INT** = 1 << 6, **STATUS_UV** = 1 << 5, **STATUS_DINT** = 1 << 4, **STATUS_CHK** = 1 << 3, **STATUS_DAT** = 1 << 2, **STATUS_SSC** = 1 << 1, **STATUS_SOT** = 1 << 0 }

  PHY STATUS register flags.
- enum TempRegister { **TEMP_NONE** = 0 }

  PHY TEMP register flags.
- enum DcDcRegister { **DCDC_NONE** = 0, **DCDC_DIS** = 1 << 7, **DCDC_BYP** = 1 << 6, **DCDC_FSET_500kHz** = 4 << 3, **DCDC_FSET_625kHz** = 5 << 3, **DCDC_FSET_710kHz** = 6 << 3, **DCDC_FSET_830kHz** = 7 << 3, **DCDC_FSET_1000kHz** = 0 << 3, **DCDC_FSET_1250kHz** = 1 << 3, **DCDC_FSET_1670kHz** = 2 << 3, **DCDC_FSET_2000kHz** = 3 << 3, **DCDC_VSET_4V2** = 4, **DCDC_VSET_4V5** = 5, **DCDC_VSET_4V9** = 6, **DCDC_VSET_5V4** = 7, **DCDC_VSET_6V0** = 0, **DCDC_VSET_6V7** = 1, **DCDC_VSET_7V8** = 2, **DCDC_VSET_9V5** = 3 }

  HMT7748 DCDC register flags.
- enum DstatRegister { **DSTAT_NONE** = 0, **DSTAT_LVL** = 1 << 2, **DSTAT_SSC** = 1 << 1 }

  HMT7748 DSTAT register flags.

Protected Member Functions

- PhyDriver ()

  Default constructor.

Static Protected Member Functions

- static void configure ()

  Configures the hardware resources for the ISR.

- static void stopInterrupt ()

  Temporarily disable the ISR.
- static void restartInterrupt ()

  Restart the ISR.
- static uint8_t registerReadBegin (Registers address)

  Start reading from PHY registers.
- static uint8_t registerReadNext ()

  Read next PHY register value.
- static uint8_t registerReadLast ()

  Read final PHY register value.
- static uint8_t registerRead (Registers address)

  Read a single byte from a PHY register.
- static uint8_t registerReadStatus ()

  Read the status register.
- static void registerWriteBegin (Registers address)

  Start write operation to PHY registers.
- static uint8_t registerWriteBegin (Registers address, uint8_t data)

  Start write operation to PHY registers.
- static void registerWriteNext (uint8_t data)

  Write next PHY register value.
- static void registerWriteDone ()

  Finish write access.
- static uint8_t registerWrite (Registers address, uint8_t data)

  Write a single byte to a PHY register.
- static uint8_t registerReadWriteBegin (Registers address, uint8_t data)

  Start write/read operation to PHY registers.
- static uint8_t registerReadWriteNext (uint8_t data)

  Write/read next PHY register value.
- static uint8_t registerReadWriteDone ()

  Finish write access and return final PHY register value.
- static void registerAbortAccess ()

  Abort register access.

Static Protected Attributes

- static SpiSsHndlr _ssHndlr

  Handler functor for SS/ line.

6.5.1 Detailed Description

template<class SpiSsHndlr = DefaultSsHandler>class PhyDriver< SpiSsHndlr >

Static class implementing register access to the PHY.

Parameters

SpiSsHndlr	Class for asserting / deasserting SPI SS/ line

6.5.2 Member Enumeration Documentation

6.5.2.1 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::CfgRegister [protected]

PHY CFG register flags.

Enumerator:

CFG_BD_38400 COM2.

CFG_BD_230400 COM3.

6.5.2.2 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::LedLevel

LED currents.

Enumerator:

LED_LEVEL_OFF LED off.

LED_LEVEL_1 ∼0.5mA

LED_LEVEL_2 ∼1.0mA

LED_LEVEL_3 ∼1.5mA

LED_LEVEL_4 ∼2.0mA

LED_LEVEL_5 ∼2.5mA

LED_LEVEL_6 ∼3.0mA

LED_LEVEL_7 ∼3.5mA

LED_LEVEL_INVALID not a LED level

6.5.3 Member Function Documentation

6.5.3.1 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerRead (Registers *address*) [inline, static, protected]

Read a single byte from a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY register value

See also

registerReadBegin
registerReadNext
registerReadLast

6.5.3.2 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadBegin (Registers *address*) [static, protected]

Start reading from PHY registers.

Asserts SS/ and starts reading from PHY register at specified address. This call must be followed by zero or more calls to `registerReadNext()` and a final call to registerReadLast().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY status register value

See also

registerReadNext
registerReadLast

6.5.3.3 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadLast ( ) [static, protected]

Read final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadNext

6.5.3.4 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadNext ( ) [static, protected]

Read next PHY register value.

This function automatically request the following register value. Use `registerReadLast` when reading the last required PHY register value.

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadLast

6.5.3.5 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadStatus ( ) [static, protected]

Read the status register.

Returns

PHY status register value

6.5.3.6 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteBegin (Registers *address*, uint8_t *data*) [static, protected]

Start write/read operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.5.3.7 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteDone ( ) [static, protected]

Finish write access and return final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteNext

6.5.3.8 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteNext (uint8_t *data*) [static, protected]

Write/read next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteDone

6.5.3.9 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWrite (Registers *address*, uint8_t *data*) [inline, static, protected]

Write a single byte to a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	Value to write

Returns

PHY status register

6.5.3.10 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*) [static, protected]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by one or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

See also

registerWriteNext
registerWriteDone

6.5.3.11 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*, uint8_t *data*) [static, protected]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.5.3.12 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteDone ( ) [static, protected]

Finish write access.

Waits for SPI communication to complete and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

See also

registerWriteBegin
registerWriteNext

6.5.3.13 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteNext (uint8_t *data*) [static, protected]

Write next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

See also

registerWriteBegin
registerWriteDone

6.6 Spi Struct Reference

Helper class for handling SPI communication with the HMT7742.

`#include <spi.h>`

Static Public Member Functions

- static void configure ()
  Configures the hardware resources for SPI communication with the PHY.
- static void enable ()
  Enable SPI.
- static void disable ()
  Disable SPI.
- static void wait ()
  Waits until SPI transmission / reception complete.

- static void tx (uint8_t data)

  Send a byte to SPI.
- static uint8_t rx ()

  Read a byte from SPI.
- static uint8_t txRx (uint8_t data)

  Send and receive a byte by SPI.

6.6.1 Detailed Description

Helper class for handling SPI communication with the HMT7742.

6.6.2 Member Function Documentation

6.6.2.1 static uint8_t Spi::rx ( ) [inline, static]

Read a byte from SPI.

Note

Automatically calls `wait()`

Returns

Received byte

6.6.2.2 static void Spi::tx (uint8_t *data*) [inline, static]

Send a byte to SPI.

Note

Call spiRx to read the byte received by the slave.

Parameters

data	Byte to send

6.6.2.3 static uint8_t Spi::txRx (uint8_t *data*) [inline, static]

Send and receive a byte by SPI.

Parameters

data	Byte to send

Returns

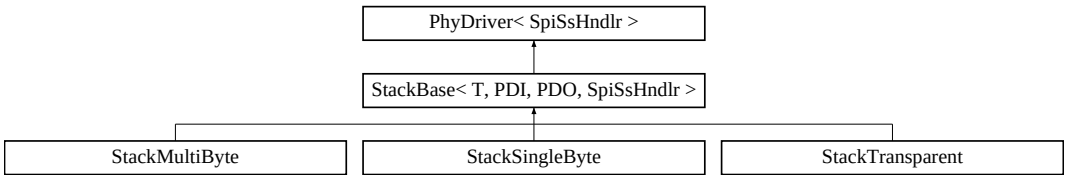
Byte received

6.7 StackBase< T, PDI, PDO, SpiSsHndlr > Class Template Reference

The StackBase is the base class for minimal IO-Link stacks for different IO-Link devices.

`#include <stackbase.h>`

Inheritance diagram for StackBase< T, PDI, PDO, SpiSsHndlr >:



Classes

- struct Parameter

  Parameter structure.
- struct ProcessData

  Structure for holding process data and associated status flags.

Public Types

- enum Led { LED_1, LED_2 }

  LEDs.
- enum StackMode { **STACK_MODE_SIO**, **STACK_MODE_IOLINK** }

  Stack mode.
- enum SioDriveMode { DRIVE_MODE_PUSH_PULL = 0, DRIVE_MODE_NPN, DRIVE_MODE_PNP, DRIVE_MODE_INACTIVE }

  Drive capability in SIO mode.
- typedef PhyDriver< SpiSsHndlr > Phy

  Convenience typedef for PhyDriver.
- typedef T StackT

  Specific derived stack type.
- typedef StackBase< T, PDI, PDO, SpiSsHndlr > BaseT

  Convenience typedef for this class.

- typedef ProcessData< PDI > ProcessDataIn

  Input process data.
- typedef ProcessData< PDO > ProcessDataOut

  Output process data.
- enum LedLevel { LED_LEVEL_OFF, LED_LEVEL_1, LED_LEVEL_2, LED_LEVEL_3, LED_LEVEL_4, LED_LEVEL_5, LED_LEVEL_6, LED_LEVEL_7, **LED_LEVEL_MAX** = LED_LEVEL_7, LED_LEVEL_INVALID }

  LED currents.

Public Member Functions

- void configure ()

  Configure sets up the hardware resources on the uC, and initializes the stack.
- bool canRunUserCode (const Parameter *&lastWrittenParameter)
- StackMode stackMode () const

  Get current stack mode.
- bool masterLost () const

  Test if connection to master has been lost.
- bool flag () const

  Debugging flag.
- void setSioLevel (bool active)

  Specify the level of the CQ line in SIO mode (STACK_MODE_SIO) state.
- uint8_t parameterRead (uint8_t address) const

  Read value from direct parameter page.
- void parameterWrite (uint8_t address, uint8_t value)

  Write value to direct parameter page.
- ProcessDataIn & processInputData ()

  Get buffer for returning process input data from slave to master.
- const ProcessDataOut & processOutputData () const

  Get buffer for process output data received from master.
- void setLedLevel (Led led, typename Phy::LedLevel level)

  Set LED level.
- Phy::LedLevel ledLevel (Led led) const

  Get LED level.
- uint8_t temperature () const

  Get current measured temperature value.

Static Public Member Functions

- static void stopInterrupt ()

  Temporarily disable the ISR.
- static void restartInterrupt ()

  Restart the ISR.

Static Public Attributes

- static const uint8_t PD_IN_SIZE = PDI

  Amount of input process data (in octets)
- static const uint8_t PD_OUT_SIZE = PDO

  Amount of output process data (in octets)

Protected Types

- enum HandlerResult { **ResultSuccess** = 0, **ResultNoData**, **ResultChecksumError**, **ResultIllegalMessageType**, **ResultPhyReset** }

  Result codes for ISR sub-handlers.
- enum MseqRegister { **MSEQ_M2CNT_SHIFT** = 2, **MSEQ_OD_1** = 0 << 0, **MSEQ_OD_2** = 1 << 0, **MSEQ_OD_8** = 2 << 0 }

  PHY MSEQ register flags.
- enum CfgRegister { **CFG_NONE** = 0, **CFG_UVT_18_0V** = 0 << 5, **CFG_UVT_16_3V** = 1 << 5, **CFG_UVT_15_0V** = 2 << 5, **CFG_UVT_13_9V** = 3 << 5, **CFG_UVT_12_0V** = 4 << 5, **CFG_UVT_10_0V** = 5 << 5, **CFG_UVT_8_6V** = 6 << 5, **CFG_UVT_7_2V** = 7 << 5, CFG_BD_38400 = 0 << 4, CFG_BD_230400 = 1 << 4, **CFG_RF_ABS** = 0 << 3, **CFG_RF_REL** = 1 << 3, **CFG_S5V_SS** = 0 << 0, **CFG_S5V_3_3V** = 2 << 0, **CFG_S5V_5_0V** = 3 << 0 }

  PHY CFG register flags.
- enum CtlRegister { **CTL_NONE** = 0, **CTL_TRNS_MODE** = 1 << 7, **CTL_SCT_190MA** = 0 << 4, **CTL_SCT_210MA** = 1 << 4, **CTL_SCT_230MA** = 2 << 4, **CTL_SCT_250MA** = 3 << 4, **CTL_SCT_110MA** = 4 << 4, **CTL_SCT_130MA** = 5 << 4, **CTL_SCT_150MA** = 6 << 4, **CTL_SCT_170MA** = 7 << 4, **CTL_SGL_MODE** = 1 << 3, **CTL_IEN_MODE** = 1 << 3, **CTL_IOLINK_MODE** = 0 << 2, **CTL_DIO** = 1 << 2, **CTL_JOIN** = 0 << 2, **CTL_SIO_MODE** = 1 << 2, **CTL_HS** = 1 << 1, **CTL_LS** = 1 << 0 }

  PHY CTL register flags.
- enum LinkRegister { **LINK_NONE** = 0, **LINK_CNT_MASK** = 0x3C, **LINK_CNT_SHIFT** = 2, **LINK_END** = 1 << 1, **LINK_SND** = 1 << 0 }

  PHY LINK register flags.
- enum StatusRegister { **STATUS_NONE** = 0, **STATUS_RST** = 1 << 7, **STATUS_INT** = 1 << 6, **STATUS_UV** = 1 << 5, **STATUS_DINT** = 1 << 4, **STATUS_CHK** = 1 << 3, **STATUS_DAT** = 1 << 2, **STATUS_SSC** = 1 << 1, **STATUS_SOT** = 1 << 0 }

  PHY STATUS register flags.
- enum TempRegister { **TEMP_NONE** = 0 }

  PHY TEMP register flags.
- enum DcDcRegister { **DCDC_NONE** = 0, **DCDC_DIS** = 1 << 7, **DCDC_BYP** = 1 << 6, **DCDC_FSET_500kHz** = 4 << 3, **DCDC_FSET_625kHz** = 5 << 3, **DCDC_FSET_710kHz** = 6 << 3, **DCDC_FSET_830kHz** = 7 << 3, **DCDC_FSET_1000kHz** = 0 << 3, **DCDC_FSET_1250kHz** = 1 << 3, **DCDC_FSET_1670kHz** = 2 << 3, **DCDC_FSET_2000kHz** = 3 << 3, **DCDC_VSET_4V2** = 4, **DCDC_VSET_4V5** = 5, **DCDC_VSET_4V9** = 6, **DCDC_VSET_5V4** = 7, **DCDC_VSET_6V0** = 0, **DCDC_VSET_6V7** = 1, **DCDC_VSET_7V8** = 2, **DCDC_VSET_9V5** = 3 }

  HMT7748 DCDC register flags.
- enum DstatRegister { **DSTAT_NONE** = 0, **DSTAT_LVL** = 1 << 2, **DSTAT_SSC** = 1 << 1 }

  HMT7748 DSTAT register flags.

Protected Member Functions

- StackBase ()

  Default constructor.
- StackT & stack ()

  Helper function returning derived stack specialization.
- void configureStackBase ()

  Configure the stack base class.
- void configureStack ()

  Empty default implementation of derived stack configuration.
- void configurePhy ()

  Configure the PHY.
- uint8_t setSioActive ()

  Put PHY in SIO-Active state.
- void odWrite (uint8_t channel, uint8_t address, uint8_t data)

  Write received on-demand data.
- uint8_t odRead (uint8_t channel, uint8_t address)

  Read requested on-demand data.
- void updateCyclePeriod ()

  Calculate cycle period from MasterCycleTime.
- void startCallbackTimer (uint8_t delay=0)

  Start / synchronize user-callback timer.
- void onTimer0CompBInterrupt ()

  ISR handler as member function.
- friend void::TIMER0_COMPB_vect ()

  The ISR function can access the stack state, and is declared here as a friend.

Static Protected Member Functions

- static uint8_t setSioListen ()

  Put PHY in SIO-Listen mode.
- static uint8_t setIoLinkListen ()

  Put PHY in IO-Link-Listen mode.
- template<IoLink::DeviceDLMode DDL_MODE>
  static int8_t getOdOctetCount ()

  Get number of OD octets.
- static bool validateControlOctet (uint8_t mc)

  Validate control octet.
- template<IoLink::DeviceDLMode DDL_MODE>
  static bool validateFrameType (uint8_t ckt)

  Validate frame type.
- static uint8_t registerReadBegin (Registers address)

  Start reading from PHY registers.

- static uint8_t registerReadNext ()

  Read next PHY register value.
- static uint8_t registerReadLast ()

  Read final PHY register value.
- static uint8_t registerRead (Registers address)

  Read a single byte from a PHY register.
- static uint8_t registerReadStatus ()

  Read the status register.
- static void registerWriteBegin (Registers address)

  Start write operation to PHY registers.
- static uint8_t registerWriteBegin (Registers address, uint8_t data)

  Start write operation to PHY registers.
- static void registerWriteNext (uint8_t data)

  Write next PHY register value.
- static void registerWriteDone ()

  Finish write access.
- static uint8_t registerWrite (Registers address, uint8_t data)

  Write a single byte to a PHY register.
- static uint8_t registerReadWriteBegin (Registers address, uint8_t data)

  Start write/read operation to PHY registers.
- static uint8_t registerReadWriteNext (uint8_t data)

  Write/read next PHY register value.
- static uint8_t registerReadWriteDone ()

  Finish write access and return final PHY register value.
- static void registerAbortAccess ()

  Abort register access.

Protected Attributes

- IoLink::DeviceDLMode _ddlMode

  Device DL-mode.
- ProcessDataIn _processDataIn

  process input data buffers
- ProcessDataOut _processDataOut

  process output data buffer
- uint8_t _deadCycleCtr

  Count of cycles from last master exchange.
- int8_t _hiZCounter

  Cycle counter for listening to the CQ.
- bool _sioLevel: 1

  Level of CQ line during SIOActive state.

Static Protected Attributes

- static SpiSsHndlr _ssHndlr

  Handler functor for SS/ line.

6.7.1 Detailed Description

template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler>class StackBase< T, PDI, PDO, SpiSsHndlr >

The StackBase is the base class for minimal IO-Link stacks for different IO-Link devices.

It is expected that a derived version of the MiniStack is used, with member functions redefined according to the application.

One stack derivative is instantiated in the application.

The stack is configured before operation, using configure(). This first configures the uC hardware resources to communicate with the PHY, including the interrupt service routine. The stack is reset, defaulting to SIO mode.

The PHY itself is configured on demand. If the PHY sees a reset, and is ready for operation, this is indicated in the status bits read by the MiniStack. The configurePhy() function is called, automatically and the stack is placed into SIO mode.

Operation of the stack is primarily interrupt driven. When a PHY event occurs, either because the master has sent an M-sequence, or due to a local PHY event, the interrupt service routine (ISR) is called. The ISR maintains the stack state (STACK_MODE_SIO or STACK_MODE_IOLINK), and automatically passes process data in or out.

The parameter data page is maintained within the stack. Where the master reads from the page, the stack provides the information without informing the application. Where the master writes to the page, the attempted operation to the page is reported in the stack state, and must be passed back to the stack using setParameterData() to have an effect.

Attention

ISDU's are not supported.
Frame TYPE_1_1/1_2 (interleaved) is not supported

The MiniStack code is closely linked to the interrupt service routine, ISR(PCINT1_vect), which performs much of the stack handling, and makes reference to the global stack.

Parameters

T	Type of specific derived stack class, will be typedef'ed as `StackT`
PDI	Amount of input process data (in octets), will be assigned to `PD_IN_SIZE`
PDO	Amount of output process data (in octets), will be assigned to `PD_OUT_SIZE`
SpiSsHndlr	Optional class handling switching between SPI and UART communication with the PHY when using transparent mode.

6.7.2 Member Enumeration Documentation

6.7.2.1 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::CfgRegister [protected, inherited]

PHY CFG register flags.

Enumerator:

CFG_BD_38400 COM2.
CFG_BD_230400 COM3.

6.7.2.2 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> enum StackBase::Led

LEDs.

Enumerator:

LED_1 LED 1.
LED_2 LED 2.

6.7.2.3 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::LedLevel [inherited]

LED currents.

Enumerator:

LED_LEVEL_OFF LED off.
LED_LEVEL_1 ∼0.5mA
LED_LEVEL_2 ∼1.0mA
LED_LEVEL_3 ∼1.5mA
LED_LEVEL_4 ∼2.0mA
LED_LEVEL_5 ∼2.5mA
LED_LEVEL_6 ∼3.0mA
LED_LEVEL_7 ∼3.5mA
LED_LEVEL_INVALID not a LED level

6.7.2.4 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> enum StackBase::SioDriveMode

Drive capability in SIO mode.

Enumerator:

DRIVE_MODE_PUSH_PULL Push-pull, HS and LS active.
DRIVE_MODE_NPN LS only used.
DRIVE_MODE_PNP HS only used.
DRIVE_MODE_INACTIVE neither switch used (typical for an actuator)

6.7.3 Member Function Documentation

6.7.3.1 template<class T , int PDI, int PDO, class SpiSsHndlr > bool StackBase< T, PDI, PDO, SpiSsHndlr >::canRunUserCode (const Parameter *& *lastWrittenParameter*)

Test if cyclic user code may run

Call this function immediately after being woken up in the application's main() loop, or, if no sleep mode is being used, at least every 0.1ms.

Parameters

lastWrittenParameter	Pointer reference in which the function returns a Parameter structure. If the returned pointer is not NULL then the most recent message completed a write access to the direct parameter page. The data is *not* automatically written to the direct parameter page, but needs to be manually committed by calling parameterWrite(). The returned pointer may be NULL if no write access occurred.

Returns

true: user code may run now; false otherwise

6.7.3.2 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::configurePhy ( ) [protected]

Configure the PHY.

This function is called whenever the stack detects that the PHY has been reset

6.7.3.3 template<class T , int PDI, int PDO, class SpiSsHndlr > template<IoLink::DeviceDLMode DDL_MODE> int8_t StackBase< T, PDI, PDO, SpiSsHndlr >::getOdOctetCount ( ) [static, protected]

Get number of OD octets.

Template Parameters

DDL_MODE	Applicable DDL mode (see IoLink::DeviceDLMode)

Returns

Expected OD octet count

6.7.3.4 template<class T , int PDI, int PDO, class SpiSsHndlr > PhyDriver< SpiSsHndlr >::LedLevel StackBase< T, PDI, PDO, SpiSsHndlr >::ledLevel (Led *led*) const

Get LED level.

Parameters

led	Selected LED

Returns

Current LED level

6.7.3.5 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> bool StackBase< T, PDI, PDO, SpiSsHndlr >::masterLost ( ) const [inline]

Test if connection to master has been lost.

Returns

`True` if no communication exchange took place for at least four cycles.

6.7.3.6 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::odRead (uint8_t *channel,* uint8_t *address*) [protected]

Read requested on-demand data.

Parameters

channel	Message channel (IoLink::MC_CHNL_*)
address	Address within selected channel

Returns

Data octet at address

6.7.3.7 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::odWrite (uint8_t *channel,* uint8_t *address,* uint8_t *data*) [protected]

Write received on-demand data.

Parameters

channel	Message channel (IoLink::MC_CHNL_*)
address	Address within selected channel
data	Data octet to write to address

6.7.3.8 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::parameterRead (uint8_t *address*) const [inline]

Read value from direct parameter page.

Parameters

address	Parameter index

Returns

Read value

6.7.3.9 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::parameterWrite (uint8_t *address*, uint8_t *value*)

Write value to direct parameter page.

Parameters

address	Parameter index
value	Value to write

6.7.3.10 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> ProcessDataIn& StackBase< T, PDI, PDO, SpiSsHndlr >::processInputData ( ) [inline]

Get buffer for returning process input data from slave to master.

Returns

Process input data buffer (read/writable)

6.7.3.11 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> const ProcessDataOut& StackBase< T, PDI, PDO, SpiSsHndlr >::processOutputData ( ) const [inline]

Get buffer for process output data received from master.

Returns

Process output data buffer (read-only)

6.7.3.12 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerRead (Registers *address*) [inline, static, protected, inherited]

Read a single byte from a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY register value

See also

registerReadBegin
registerReadNext
registerReadLast

6.7.3.13 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadBegin (Registers *address*) [static, protected, inherited]

Start reading from PHY registers.

Asserts SS/ and starts reading from PHY register at specified address. This call must be followed by zero or more calls to `registerReadNext()` and a final call to registerReadLast().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY status register value

See also

registerReadNext
registerReadLast

6.7.3.14 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadLast ( ) [static, protected, inherited]

Read final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadNext

6.7.3.15 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadNext ( ) [static, protected, inherited]

Read next PHY register value.

This function automatically request the following register value. Use `registerReadLast` when reading the last required PHY register value.

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadLast

6.7.3.16 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadStatus ( ) [static, protected, inherited]

Read the status register.

Returns

PHY status register value

6.7.3.17 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteBegin (Registers *address*, uint8_t *data*) [static, protected, inherited]

Start write/read operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.7.3.18 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteDone ( ) [static, protected, inherited]

Finish write access and return final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteNext

6.7.3.19 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteNext (uint8_t *data*) [static, protected, inherited]

Write/read next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteDone

6.7.3.20 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWrite (Registers *address*, uint8_t *data*) [inline, static, protected, inherited]

Write a single byte to a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	Value to write

Returns

PHY status register

6.7.3.21 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*) [static, protected, inherited]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by one or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

See also

registerWriteNext
registerWriteDone

6.7.3.22 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*, uint8_t *data*) [static, protected, inherited]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.7.3.23 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteDone ( ) [static, protected, inherited]

Finish write access.

Waits for SPI communication to complete and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

See also

registerWriteBegin
registerWriteNext

6.7.3.24 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteNext (uint8_t *data*) [static, protected, inherited]

Write next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

See also

registerWriteBegin
registerWriteDone

6.7.3.25 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setIoLinkListen ( ) [static, protected]

Put PHY in IO-Link-Listen mode.

Returns

PHY status register

Reimplemented in StackTransparent.

6.7.3.26 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::setLedLevel (Led *led,* typename Phy::LedLevel *level*)

Set LED level.

Parameters

led	Selected LED
level	Desired level

6.7.3.27 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setSioActive ( ) [protected]

Put PHY in SIO-Active state.

The CQ line is driven according to the setting of the HS and LS bits, as specified by Stack::SIO_DRIVE_MODE.

Returns

PHY status register

Reimplemented in StackTransparent.

6.7.3.28 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::setSioLevel (bool *active*)

Specify the level of the CQ line in SIO mode (STACK_MODE_SIO) state.

Parameters

active	If Stack::SIO_DRIVE_MODE is Stack::DRIVE_MODE_PNP or Stack::DRIVE_MODE_NPN the relevant switch is activated if `active` == true. In Stack::DRIVE_MODE_PUSH_PULL (push-pull) CQ is driven high (`active` == true) or low (`active` == false).

6.7.3.29 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setSioListen ( ) [static, protected]

Put PHY in SIO-Listen mode.

Returns

PHY status register

Reimplemented in StackTransparent.

6.7.3.30 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackT& StackBase< T, PDI, PDO, SpiSsHndlr >::stack ( ) [inline, protected]

Helper function returning derived stack specialization.

Returns

Derived stack instance

6.7.3.31 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackMode StackBase< T, PDI, PDO, SpiSsHndlr >::stackMode ( ) const [inline]

Get current stack mode.

Returns

STACK_MODE_SIO or STACK_MODE_IOLINK

6.7.3.32 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::startCallbackTimer (uint8_t *delay* = 0) [protected]

Start / synchronize user-callback timer.

Starts a timer which ensures that the user call-back will be called on a regular basis, even if no master message has been received

Parameters

delay	Delay until first call to user callback (in 1/10ms units)

6.7.3.33 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::temperature ( ) const

Get current measured temperature value.

In order to convert the returned value to a Celsius reading, the following formula should be applied:

Temperature [Celsius] = (80 - temp) $*$ 2.70

Returns

Current temperature value

6.7.3.34 template<class T , int PDI, int PDO, class SpiSsHndlr > template<IoLink::DeviceDLMode DDL_MODE> bool StackBase< T, PDI, PDO, SpiSsHndlr >::validateFrameType (uint8_t *ckt*) [static, protected]

Validate frame type.

Template Parameters

DDL_MODE	Applicable DDL mode (see IoLink::DeviceDLMode)

Parameters

ckt	Received CKT frame octet.

Returns

True if frame type valid, false otherwise

6.7.3.35 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackBase< T, PDI, PDO, SpiSsHndlr >::void::TIMER0_COMPB_vect ( ) [protected]

The ISR function can access the stack state, and is declared here as a friend.

6.8 StackBase< T, PDI, PDO, SpiSsHndlr >::Parameter Struct Reference

Parameter structure.

`#include <stackbase.h>`

Public Attributes

- uint8_t **address**
- uint8_t **value**

6.8.1 Detailed Description

template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler>struct StackBase< T, PDI, PDO, SpiSsHndlr >::Parameter

Parameter structure.

6.9 StackBase< T, PDI, PDO, SpiSsHndlr >::ProcessData< SIZE > Struct Template Reference

Structure for holding process data and associated status flags.

`#include <stackbase.h>`

Public Attributes

- uint8_t buffer [SIZE]

  Process data buffer.
- bool isValid

  Indicates if buffer data is valid.

6.9.1 Detailed Description

template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler>template<int SIZE>struct StackBase< T, PDI, PDO, SpiSsHndlr >::ProcessData< SIZE >

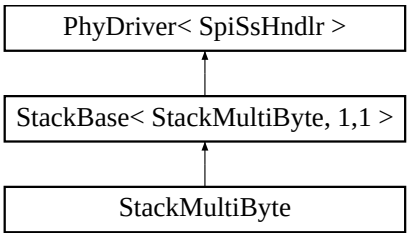
Structure for holding process data and associated status flags.

6.10 StackMultiByte Class Reference

Stack implementation using multi-byte mode.

`#include <stackmultibyte.h>`

Inheritance diagram for StackMultiByte:



Public Types

- enum Led { LED_1, LED_2 }
  LEDs.
- enum StackMode { **STACK_MODE_SIO**, **STACK_MODE_IOLINK** }
  Stack mode.
- enum SioDriveMode { DRIVE_MODE_PUSH_PULL = 0, DRIVE_MODE_NPN, DRIVE_MODE_PNP, DRIVE_MODE_INACTIVE }
  Drive capability in SIO mode.
- typedef PhyDriver< SpiSsHndlr > Phy
  Convenience typedef for PhyDriver.
- typedef T StackT
  Specific derived stack type.
- typedef StackBase< T, PDI, PDO, SpiSsHndlr > BaseT
  Convenience typedef for this class.
- typedef ProcessData< PDI > ProcessDataIn
  Input process data.
- typedef ProcessData< PDO > ProcessDataOut
  Output process data.
- enum LedLevel { LED_LEVEL_OFF, LED_LEVEL_1, LED_LEVEL_2, LED_LEVEL_3, LED_LEVEL_4, LED_LEVEL_5, LED_LEVEL_6, LED_LEVEL_7, **LED_LEVEL_MAX** = LED_LEVEL_7, LED_LEVEL_INVALID }
  LED currents.

Public Member Functions

- void configure ()

  Configure sets up the hardware resources on the uC, and initializes the stack.
- bool canRunUserCode (const Parameter *&lastWrittenParameter)
- StackMode stackMode () const

  Get current stack mode.
- bool masterLost () const

  Test if connection to master has been lost.
- bool flag () const

  Debugging flag.
- void setSioLevel (bool active)

  Specify the level of the CQ line in SIO mode (STACK_MODE_SIO) state.
- uint8_t parameterRead (uint8_t address) const

  Read value from direct parameter page.
- void parameterWrite (uint8_t address, uint8_t value)

  Write value to direct parameter page.
- ProcessDataIn & processInputData ()

  Get buffer for returning process input data from slave to master.
- const ProcessDataOut & processOutputData () const

  Get buffer for process output data received from master.
- void setLedLevel (Led led, typename Phy::LedLevel level)

  Set LED level.
- Phy::LedLevel ledLevel (Led led) const

  Get LED level.
- uint8_t temperature () const

  Get current measured temperature value.

Static Public Member Functions

- static void stopInterrupt ()

  Temporarily disable the ISR.
- static void restartInterrupt ()

  Restart the ISR.

Static Public Attributes

- static const uint8_t REVISION_ID = IoLink::REVISION_ID_1_1

  RevisionID of protocol implemented (Direct Parameter 0x04)
- static const uint16_t VENDOR_ID = 0x01a6

  VendorID (Direct Parameters 0x07 and 0x08)
- static const uint32_t DEVICE_ID = 0x123456

  DeviceID (Direct Parameters 0x09 - 0x0b)

- static const uint32_t BAUD_RATE = 38400

  Communication speed (must be either 38400 or 230400)
- static const uint8_t MIN_CYC_TIME = 30

  MinCycleTime in 0.1ms units.
- static const uint8_t MSEQ_CAPABILITY

  M-sequence Capability (Direct Parameter 0x03)
- static const uint8_t PHY_CFG

  PHY configuration.
- static const uint8_t PHY_CTL_SCT = CTL_SCT_190MA

  PHY short-circuit threshold.
- static const uint8_t PHY_CTL_MODE = CTL_IOLINK_MODE

  PHY mode (must be CTL_IOLINK_MODE)
- static enum SioDriveMode SIO_DRIVE_MODE = DRIVE_MODE_PUSH_PULL

  PHY drive mode to use in SIO mode.
- static const uint8_t PHY_THERM_DEG = 175

  PHY thermal shutdown temperature (in degrees centigrade)
- static StackMultiByte instance

  The one and only stack instance.
- static const uint8_t PD_IN_SIZE = PDI

  Amount of input process data (in octets)
- static const uint8_t PD_OUT_SIZE = PDO

  Amount of output process data (in octets)

Protected Types

- enum HandlerResult { **ResultSuccess** = 0, **ResultNoData**, **ResultChecksumError**, **ResultIllegalMessageType**, **ResultPhyReset** }

  Result codes for ISR sub-handlers.
- enum MseqRegister { **MSEQ_M2CNT_SHIFT** = 2, **MSEQ_OD_1** = 0 << 0, **MSEQ_OD_2** = 1 << 0, **MSEQ_OD_8** = 2 << 0 }

  PHY MSEQ register flags.
- enum CfgRegister { **CFG_NONE** = 0, **CFG_UVT_18_0V** = 0 << 5, **CFG_UVT_16_3V** = 1 << 5, **CFG_UVT_15_0V** = 2 << 5, **CFG_UVT_13_9V** = 3 << 5, **CFG_UVT_12_0V** = 4 << 5, **CFG_UVT_10_0V** = 5 << 5, **CFG_UVT_8_6V** = 6 << 5, **CFG_UVT_7_2V** = 7 << 5, CFG_BD_38400 = 0 << 4, CFG_BD_230400 = 1 << 4, **CFG_RF_ABS** = 0 << 3, **CFG_RF_REL** = 1 << 3, **CFG_S5V_SS** = 0 << 0, **CFG_S5V_3_3V** = 2 << 0, **CFG_S5V_5_0V** = 3 << 0 }

  PHY CFG register flags.
- enum CtlRegister { **CTL_NONE** = 0, **CTL_TRNS_MODE** = 1 << 7, **CTL_SCT_190MA** = 0 << 4, **CTL_SCT_210MA** = 1 << 4, **CTL_SCT_230MA** = 2 << 4, **CTL_SCT_250MA** = 3 << 4, **CTL_SCT_110MA** = 4 << 4, **CTL_SCT_130MA** = 5 << 4, **CTL_SCT_150MA** = 6 << 4, **CTL_SCT_170MA** = 7 << 4, **CTL_SGL_MODE** = 1 << 3, **CTL_IEN_MODE** = 1 << 3, **CTL_IOLINK_MODE** = 0 << 2, **CTL_DIO** = 1 << 2, **CTL_JOIN** = 0 << 2, **CTL_SIO_MODE** = 1 << 2, **CTL_HS** = 1 << 1, **CTL_LS** = 1 << 0 }

  PHY CTL register flags.

- enum LinkRegister { **LINK_NONE** = 0, **LINK_CNT_MASK** = 0x3C, **LINK_CNT_SHIFT** = 2, **LINK_END** = 1 << 1, **LINK_SND** = 1 << 0 }

  PHY LINK register flags.
- enum StatusRegister { **STATUS_NONE** = 0, **STATUS_RST** = 1 << 7, **STATUS_INT** = 1 << 6, **STATUS_UV** = 1 << 5, **STATUS_DINT** = 1 << 4, **STATUS_CHK** = 1 << 3, **STATUS_DAT** = 1 << 2, **STATUS_SSC** = 1 << 1, **STATUS_SOT** = 1 << 0 }

  PHY STATUS register flags.
- enum TempRegister { **TEMP_NONE** = 0 }

  PHY TEMP register flags.
- enum DcDcRegister { **DCDC_NONE** = 0, **DCDC_DIS** = 1 << 7, **DCDC_BYP** = 1 << 6, **DCDC_FSET_500kHz** = 4 << 3, **DCDC_FSET_625kHz** = 5 << 3, **DCDC_FSET_710kHz** = 6 << 3, **DCDC_FSET_830kHz** = 7 << 3, **DCDC_FSET_1000kHz** = 0 << 3, **DCDC_FSET_1250kHz** = 1 << 3, **DCDC_FSET_1670kHz** = 2 << 3, **DCDC_FSET_2000kHz** = 3 << 3, **DCDC_VSET_4V2** = 4, **DCDC_VSET_4V5** = 5, **DCDC_VSET_4V9** = 6, **DCDC_VSET_5V4** = 7, **DCDC_VSET_6V0** = 0, **DCDC_VSET_6V7** = 1, **DCDC_VSET_7V8** = 2, **DCDC_VSET_9V5** = 3 }

  HMT7748 DCDC register flags.
- enum DstatRegister { **DSTAT_NONE** = 0, **DSTAT_LVL** = 1 << 2, **DSTAT_SSC** = 1 << 1 }

  HMT7748 DSTAT register flags.

Protected Member Functions

- StackT & stack ()

  Helper function returning derived stack specialization.
- void configureStackBase ()

  Configure the stack base class.
- void configureStack ()

  Empty default implementation of derived stack configuration.
- void configurePhy ()

  Configure the PHY.
- uint8_t setSioActive ()

  Put PHY in SIO-Active state.
- void odWrite (uint8_t channel, uint8_t address, uint8_t data)

  Write received on-demand data.
- uint8_t odRead (uint8_t channel, uint8_t address)

  Read requested on-demand data.
- void updateCyclePeriod ()

  Calculate cycle period from MasterCycleTime.
- void startCallbackTimer (uint8_t delay=0)

  Start / synchronize user-callback timer.
- void onTimer0CompBInterrupt ()

  ISR handler as member function.
- friend void::TIMER0_COMPB_vect ()

  The ISR function can access the stack state, and is declared here as a friend.

Static Protected Member Functions

- static uint8_t setSioListen ()

  Put PHY in SIO-Listen mode.
- static uint8_t setIoLinkListen ()

  Put PHY in IO-Link-Listen mode.
- template<IoLink::DeviceDLMode DDL_MODE>
  static int8_t getOdOctetCount ()

  Get number of OD octets.
- static bool validateControlOctet (uint8_t mc)

  Validate control octet.
- template<IoLink::DeviceDLMode DDL_MODE>
  static bool validateFrameType (uint8_t ckt)

  Validate frame type.
- static uint8_t registerReadBegin (Registers address)

  Start reading from PHY registers.
- static uint8_t registerReadNext ()

  Read next PHY register value.
- static uint8_t registerReadLast ()

  Read final PHY register value.
- static uint8_t registerRead (Registers address)

  Read a single byte from a PHY register.
- static uint8_t registerReadStatus ()

  Read the status register.
- static void registerWriteBegin (Registers address)

  Start write operation to PHY registers.
- static uint8_t registerWriteBegin (Registers address, uint8_t data)

  Start write operation to PHY registers.
- static void registerWriteNext (uint8_t data)

  Write next PHY register value.
- static void registerWriteDone ()

  Finish write access.
- static uint8_t registerWrite (Registers address, uint8_t data)

  Write a single byte to a PHY register.
- static uint8_t registerReadWriteBegin (Registers address, uint8_t data)

  Start write/read operation to PHY registers.
- static uint8_t registerReadWriteNext (uint8_t data)

  Write/read next PHY register value.
- static uint8_t registerReadWriteDone ()

  Finish write access and return final PHY register value.
- static void registerAbortAccess ()

  Abort register access.

Protected Attributes

- IoLink::DeviceDLMode _ddlMode
  Device DL-mode.
- ProcessDataIn _processDataIn
  process input data buffers
- ProcessDataOut _processDataOut
  process output data buffer
- uint8_t _deadCycleCtr
  Count of cycles from last master exchange.
- int8_t _hiZCounter
  Cycle counter for listening to the CQ.
- bool _sioLevel: 1
  Level of CQ line during SIOActive state.

Static Protected Attributes

- static SpiSsHndlr _ssHndlr
  Handler functor for SS/ line.

6.10.1 Detailed Description

Stack implementation using multi-byte mode.

6.10.2 Member Enumeration Documentation

6.10.2.1 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::CfgRegister [protected, inherited]

PHY CFG register flags.

Enumerator:

CFG_BD_38400 COM2.

CFG_BD_230400 COM3.

6.10.2.2 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> enum StackBase::Led [inherited]

LEDs.

Enumerator:

LED_1 LED 1.

LED_2 LED 2.

6.10.2.3 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::LedLevel [inherited]

LED currents.

Enumerator:

LED_LEVEL_OFF LED off.

LED_LEVEL_1 ~0.5mA

LED_LEVEL_2 ~1.0mA

LED_LEVEL_3 ~1.5mA

LED_LEVEL_4 ~2.0mA

LED_LEVEL_5 ~2.5mA

LED_LEVEL_6 ~3.0mA

LED_LEVEL_7 ~3.5mA

LED_LEVEL_INVALID not a LED level

6.10.2.4 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> enum StackBase::SioDriveMode [inherited]

Drive capability in SIO mode.

Enumerator:

DRIVE_MODE_PUSH_PULL Push-pull, HS and LS active.

DRIVE_MODE_NPN LS only used.

DRIVE_MODE_PNP HS only used.

DRIVE_MODE_INACTIVE neither switch used (typical for an actuator)

6.10.3 Member Function Documentation

6.10.3.1 template<class T , int PDI, int PDO, class SpiSsHndlr > bool StackBase< T, PDI, PDO, SpiSsHndlr >::canRunUserCode (const Parameter *& *lastWrittenParameter*) [inherited]

Test if cyclic user code may run

Call this function immediately after being woken up in the application's main() loop, or, if no sleep mode is being used, at least every 0.1ms.

Parameters

lastWritten-Parameter	Pointer reference in which the function returns a Parameter structure. If the returned pointer is not NULL then the most recent message completed a write access to the direct parameter page. The data is *not* automatically written to the direct parameter page, but needs to be manually committed by calling parameterWrite(). The returned pointer may be NULL if no write access occurred.

Returns

true: user code may run now; false otherwise

6.10.3.2 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::configurePhy ( ) [protected, inherited]

Configure the PHY.

This function is called whenever the stack detects that the PHY has been reset

6.10.3.3 template<class T , int PDI, int PDO, class SpiSsHndlr > template<IoLink::DeviceDLMode DDL_MODE> int8_t StackBase< T, PDI, PDO, SpiSsHndlr >::getOdOctetCount ( ) [static, protected, inherited]

Get number of OD octets.

Template Parameters

DDL_MODE	Applicable DDL mode (see IoLink::DeviceDLMode)

Returns

Expected OD octet count

6.10.3.4 template<class T , int PDI, int PDO, class SpiSsHndlr > PhyDriver< SpiSsHndlr >::LedLevel StackBase< T, PDI, PDO, SpiSsHndlr >::ledLevel (Led *led*) const [inherited]

Get LED level.

Parameters

led	Selected LED

Returns

Current LED level

6.10.3.5 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> bool StackBase< T, PDI, PDO, SpiSsHndlr >::masterLost ( ) const [inline, inherited]

Test if connection to master has been lost.

Returns

`True` if no communication exchange took place for at least four cycles.

6.10.3.6 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::odRead (uint8_t *channel*, uint8_t *address*) [protected, inherited]

Read requested on-demand data.

Parameters

channel	Message channel (IoLink::MC_CHNL_*)
address	Address within selected channel

Returns

Data octet at address

6.10.3.7 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::odWrite (uint8_t *channel*, uint8_t *address*, uint8_t *data*) [protected, inherited]

Write received on-demand data.

Parameters

channel	Message channel (IoLink::MC_CHNL_*)
address	Address within selected channel
data	Data octet to write to address

6.10.3.8 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::parameterRead (uint8_t *address*) const [inline, inherited]

Read value from direct parameter page.

Parameters

address	Parameter index

Returns

Read value

6.10.3.9 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::parameterWrite (uint8_t *address*, uint8_t *value*) [inherited]

Write value to direct parameter page.

Parameters

address	Parameter index
value	Value to write

6.10.3.10 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> ProcessDataIn& StackBase< T, PDI, PDO, SpiSsHndlr >::processInputData ( ) [inline, inherited]

Get buffer for returning process input data from slave to master.

Returns

Process input data buffer (read/writable)

6.10.3.11 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> const ProcessDataOut& StackBase< T, PDI, PDO, SpiSsHndlr >::processOutputData ( ) const [inline, inherited]

Get buffer for process output data received from master.

Returns

Process output data buffer (read-only)

6.10.3.12 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerRead (Registers *address*) [inline, static, protected, inherited]

Read a single byte from a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY register value

See also

registerReadBegin
registerReadNext
registerReadLast

6.10.3.13 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadBegin (Registers *address*) [static, protected, inherited]

Start reading from PHY registers.

Asserts SS/ and starts reading from PHY register at specified address. This call must be followed by zero or more calls to `registerReadNext()` and a final call to registerReadLast().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY status register value

See also

registerReadNext
registerReadLast

6.10.3.14 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadLast ( ) [static, protected, inherited]

Read final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadNext

6.10.3.15 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadNext ( ) [static, protected, inherited]

Read next PHY register value.

This function automatically request the following register value. Use `registerReadLast` when reading the last required PHY register value.

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadLast

6.10.3.16 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadStatus ( ) [static, protected, inherited]

Read the status register.

Returns

PHY status register value

6.10.3.17 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteBegin (Registers *address*, uint8_t *data*) [static, protected, inherited]

Start write/read operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.10.3.18 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteDone ( ) [static, protected, inherited]

Finish write access and return final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteNext

6.10.3.19 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteNext (uint8_t *data*) [static, protected, inherited]

Write/read next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteDone

6.10.3.20 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWrite (Registers *address*, uint8_t *data*) [inline, static, protected, inherited]

Write a single byte to a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	Value to write

Returns

PHY status register

6.10.3.21 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*) [static, protected, inherited]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by one or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

See also

registerWriteNext
registerWriteDone

6.10.3.22 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*, uint8_t *data*) [static, protected, inherited]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.10.3.23 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteDone ( ) [static, protected, inherited]

Finish write access.

Waits for SPI communication to complete and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

See also

registerWriteBegin
registerWriteNext

6.10.3.24 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteNext (uint8_t *data*) [static, protected, inherited]

Write next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

See also

registerWriteBegin
registerWriteDone

6.10.3.25 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setIoLinkListen ( ) [static, protected, inherited]

Put PHY in IO-Link-Listen mode.

Returns

PHY status register

Reimplemented in StackTransparent.

6.10.3.26 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::setLedLevel (Led *led,* typename Phy::LedLevel *level*) [inherited]

Set LED level.

Parameters

led	Selected LED
level	Desired level

6.10.3.27 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setSioActive ( ) [protected, inherited]

Put PHY in SIO-Active state.

The CQ line is driven according to the setting of the HS and LS bits, as specified by Stack::SIO_DRIVE_MODE.

Returns

PHY status register

Reimplemented in StackTransparent.

6.10.3.28 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::setSioLevel (bool *active*) [inherited]

Specify the level of the CQ line in SIO mode (STACK_MODE_SIO) state.

Parameters

active	If Stack::SIO_DRIVE_MODE is Stack::DRIVE_MODE_PNP or Stack::DRIVE_MODE_NPN the relevant switch is activated if `active` == true. In Stack::DRIVE_MODE_PUSH_PULL (push-pull) CQ is driven high (`active` == true) or low (`active` == false).

6.10.3.29 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setSioListen ( ) [static, protected, inherited]

Put PHY in SIO-Listen mode.

Returns

PHY status register

Reimplemented in StackTransparent.

6.10.3.30 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackT& StackBase< T, PDI, PDO, SpiSsHndlr >::stack ( ) [inline, protected, inherited]

Helper function returning derived stack specialization.

Returns

Derived stack instance

6.10.3.31 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackMode StackBase< T, PDI, PDO, SpiSsHndlr >::stackMode ( ) const [inline, inherited]

Get current stack mode.

Returns

STACK_MODE_SIO or STACK_MODE_IOLINK

6.10.3.32 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::startCallbackTimer (uint8_t *delay* = 0) [protected, inherited]

Start / synchronize user-callback timer.

Starts a timer which ensures that the user call-back will be called on a regular basis, even if no master message has been received

Parameters

delay	Delay until first call to user callback (in 1/10ms units)

6.10.3.33 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::temperature ( ) const [inherited]

Get current measured temperature value.

In order to convert the returned value to a Celsius reading, the following formula should be applied:

Temperature [Celsius] = (80 - temp) ∗ 2.70

Returns

Current temperature value

6.10.3.34 template<class T , int PDI, int PDO, class SpiSsHndlr > template<IoLink::DeviceDLMode DDL_MODE> bool StackBase< T, PDI, PDO, SpiSsHndlr >::validateFrameType (uint8_t *ckt*) [static, protected, inherited]

Validate frame type.

Template Parameters

DDL_MODE	Applicable DDL mode (see IoLink::DeviceDLMode)

Parameters

ckt	Received CKT frame octet.

Returns

True if frame type valid, false otherwise

6.10.3.35 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackBase< T, PDI, PDO, SpiSsHndlr >::void::TIMER0_COMPB_vect ( ) [protected, inherited]

The ISR function can access the stack state, and is declared here as a friend.

6.10.4 Member Data Documentation

6.10.4.1 const uint8_t StackMultiByte::MSEQ_CAPABILITY [static]

Initial value:

```
IoLink::MSEQCAP_ISDU_NOT_SUPPORTED |
                                        IoLink::MSEQCAP_OP_CODE_0 |
                                        IoLink::MSEQCAP_PREOP_CODE_0
```

M-sequence Capability (Direct Parameter 0x03)

6.10.4.2 const uint8_t StackMultiByte::PHY_CFG [static]

Initial value:

```
        CFG_UVT_16_3V | CFG_RF_ABS | CFG_S5V_3_3V
                                | (BAUD_RATE == 38400 ? CFG_BD_38400 :
    CFG_BD_230400)
```

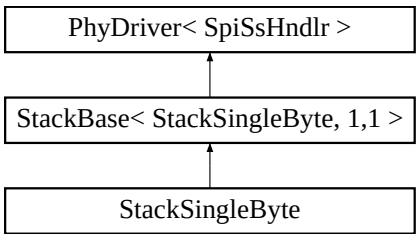
PHY configuration.

6.11 StackSingleByte Class Reference

Stack implementation using the PHY single-octet mode.

`#include <stacksinglebyte.h>`

Inheritance diagram for StackSingleByte:



Public Types

- enum Led { LED_1, LED_2 }

  LEDs.
- enum StackMode { **STACK_MODE_SIO**, **STACK_MODE_IOLINK** }

  Stack mode.
- enum SioDriveMode { DRIVE_MODE_PUSH_PULL = 0, DRIVE_MODE_NPN, DRIVE_MODE_PNP, DRIVE_MODE_INACTIVE }

  Drive capability in SIO mode.
- typedef PhyDriver< SpiSsHndlr > Phy

  Convenience typedef for PhyDriver.
- typedef T StackT

  Specific derived stack type.
- typedef StackBase< T, PDI, PDO, SpiSsHndlr > BaseT

  Convenience typedef for this class.
- typedef ProcessData< PDI > ProcessDataIn

  Input process data.
- typedef ProcessData< PDO > ProcessDataOut

  Output process data.
- enum LedLevel { LED_LEVEL_OFF, LED_LEVEL_1, LED_LEVEL_2, LED_LEVEL_3, LED_LEVEL_4, LED_LEVEL_5, LED_LEVEL_6, LED_LEVEL_7, **LED_LEVEL_MAX** = LED_LEVEL_7, LED_LEVEL_INVALID }

  LED currents.

Public Member Functions

- void configure ()

  Configure sets up the hardware resources on the uC, and initializes the stack.
- bool canRunUserCode (const Parameter *&lastWrittenParameter)
- StackMode stackMode () const

  Get current stack mode.
- bool masterLost () const

  Test if connection to master has been lost.
- bool flag () const

  Debugging flag.
- void setSioLevel (bool active)

  Specify the level of the CQ line in SIO mode (STACK_MODE_SIO) state.
- uint8_t parameterRead (uint8_t address) const

  Read value from direct parameter page.
- void parameterWrite (uint8_t address, uint8_t value)

  Write value to direct parameter page.
- ProcessDataIn & processInputData ()

  Get buffer for returning process input data from slave to master.
- const ProcessDataOut & processOutputData () const

  Get buffer for process output data received from master.
- void setLedLevel (Led led, typename Phy::LedLevel level)

  Set LED level.
- Phy::LedLevel ledLevel (Led led) const

  Get LED level.
- uint8_t temperature () const

  Get current measured temperature value.

Static Public Member Functions

- static void stopInterrupt ()

  Temporarily disable the ISR.
- static void restartInterrupt ()

  Restart the ISR.

Static Public Attributes

- static const uint8_t REVISION_ID = IoLink::REVISION_ID_1_1

  RevisionID of protocol implemented (Direct Parameter 0x04)
- static const uint16_t VENDOR_ID = 0x01a6

  VendorID (Direct Parameters 0x07 and 0x08)
- static const uint32_t DEVICE_ID = 0x123456

  DeviceID (Direct Parameters 0x09 - 0x0b)

- static const uint32_t BAUD_RATE = 38400

  Communication speed (must be either 38400 or 230400)
- static const uint8_t MIN_CYC_TIME = 30

  MinCycleTime in 0.1ms units.
- static const uint8_t MSEQ_CAPABILITY

  M-sequence Capability (Direct Parameter 0x03)
- static const uint8_t PHY_CFG

  PHY configuration.
- static const uint8_t PHY_CTL_SCT = CTL_SCT_190MA

  PHY short-circuit threshold.
- static const uint8_t PHY_CTL_MODE = CTL_SGL_MODE

  PHY mode (must be CTL_SGL_MODE)
- static enum SioDriveMode SIO_DRIVE_MODE = DRIVE_MODE_PUSH_PULL

  PHY drive mode to use in SIO mode.
- static const uint8_t PHY_THERM_DEG = 175

  PHY thermal shutdown temperature (in degrees centigrade)
- static StackSingleByte instance

  The one and only stack instance.
- static const uint8_t PD_IN_SIZE = PDI

  Amount of input process data (in octets)
- static const uint8_t PD_OUT_SIZE = PDO

  Amount of output process data (in octets)

Protected Types

- enum HandlerResult { **ResultSuccess** = 0, **ResultNoData**, **ResultChecksumError**, **ResultIllegalMessageType**, **ResultPhyReset** }

  Result codes for ISR sub-handlers.
- enum MseqRegister { **MSEQ_M2CNT_SHIFT** = 2, **MSEQ_OD_1** = 0 << 0, **MSEQ_OD_2** = 1 << 0, **MSEQ_OD_8** = 2 << 0 }

  PHY MSEQ register flags.
- enum CfgRegister { **CFG_NONE** = 0, **CFG_UVT_18_0V** = 0 << 5, **CFG_UVT_16_3V** = 1 << 5, **CFG_UVT_15_0V** = 2 << 5, **CFG_UVT_13_9V** = 3 << 5, **CFG_UVT_12_0V** = 4 << 5, **CFG_UVT_10_0V** = 5 << 5, **CFG_UVT_8_6V** = 6 << 5, **CFG_UVT_7_2V** = 7 << 5, CFG_BD_38400 = 0 << 4, CFG_BD_230400 = 1 << 4, **CFG_RF_ABS** = 0 << 3, **CFG_RF_REL** = 1 << 3, **CFG_S5V_SS** = 0 << 0, **CFG_S5V_3_3V** = 2 << 0, **CFG_S5V_5_0V** = 3 << 0 }

  PHY CFG register flags.
- enum CtlRegister { **CTL_NONE** = 0, **CTL_TRNS_MODE** = 1 << 7, **CTL_SCT_190MA** = 0 << 4, **CTL_SCT_210MA** = 1 << 4, **CTL_SCT_230MA** = 2 << 4, **CTL_SCT_250MA** = 3 << 4, **CTL_SCT_110MA** = 4 << 4, **CTL_SCT_130MA** = 5 << 4, **CTL_SCT_150MA** = 6 << 4, **CTL_SCT_170MA** = 7 << 4, **CTL_SGL_MODE** = 1 << 3, **CTL_IEN_MODE** = 1 << 3, **CTL_IOLINK_MODE** = 0 << 2, **CTL_DIO** = 1 << 2, **CTL_JOIN** = 0 << 2, **CTL_SIO_MODE** = 1 << 2, **CTL_HS** = 1 << 1, **CTL_LS** = 1 << 0 }

  PHY CTL register flags.

- enum LinkRegister { **LINK_NONE** = 0, **LINK_CNT_MASK** = 0x3C, **LINK_CNT_SHIFT** = 2, **LINK_END** = 1 << 1, **LINK_SND** = 1 << 0 }

  PHY LINK register flags.
- enum StatusRegister { **STATUS_NONE** = 0, **STATUS_RST** = 1 << 7, **STATUS_INT** = 1 << 6, **STATUS_UV** = 1 << 5, **STATUS_DINT** = 1 << 4, **STATUS_CHK** = 1 << 3, **STATUS_DAT** = 1 << 2, **STATUS_SSC** = 1 << 1, **STATUS_SOT** = 1 << 0 }

  PHY STATUS register flags.
- enum TempRegister { **TEMP_NONE** = 0 }

  PHY TEMP register flags.
- enum DcDcRegister { **DCDC_NONE** = 0, **DCDC_DIS** = 1 << 7, **DCDC_BYP** = 1 << 6, **DCDC_FSET_500kHz** = 4 << 3, **DCDC_FSET_625kHz** = 5 << 3, **DCDC_FSET_710kHz** = 6 << 3, **DCDC_FSET_830kHz** = 7 << 3, **DCDC_FSET_1000kHz** = 0 << 3, **DCDC_FSET_1250kHz** = 1 << 3, **DCDC_FSET_1670kHz** = 2 << 3, **DCDC_FSET_2000kHz** = 3 << 3, **DCDC_VSET_4V2** = 4, **DCDC_VSET_4V5** = 5, **DCDC_VSET_4V9** = 6, **DCDC_VSET_5V4** = 7, **DCDC_VSET_6V0** = 0, **DCDC_VSET_6V7** = 1, **DCDC_VSET_7V8** = 2, **DCDC_VSET_9V5** = 3 }

  HMT7748 DCDC register flags.
- enum DstatRegister { **DSTAT_NONE** = 0, **DSTAT_LVL** = 1 << 2, **DSTAT_SSC** = 1 << 1 }

  HMT7748 DSTAT register flags.

Protected Member Functions

- StackT & stack ()

  Helper function returning derived stack specialization.
- void configureStackBase ()

  Configure the stack base class.
- void configureStack ()

  Empty default implementation of derived stack configuration.
- void configurePhy ()

  Configure the PHY.
- uint8_t setSioActive ()

  Put PHY in SIO-Active state.
- void odWrite (uint8_t channel, uint8_t address, uint8_t data)

  Write received on-demand data.
- uint8_t odRead (uint8_t channel, uint8_t address)

  Read requested on-demand data.
- void updateCyclePeriod ()

  Calculate cycle period from MasterCycleTime.
- void startCallbackTimer (uint8_t delay=0)

  Start / synchronize user-callback timer.
- void onTimer0CompBInterrupt ()

  ISR handler as member function.
- friend void::TIMER0_COMPB_vect ()

  The ISR function can access the stack state, and is declared here as a friend.

Static Protected Member Functions

- static uint8_t setSioListen ()

  Put PHY in SIO-Listen mode.
- static uint8_t setIoLinkListen ()

  Put PHY in IO-Link-Listen mode.
- template<IoLink::DeviceDLMode DDL_MODE>
  static int8_t getOdOctetCount ()

  Get number of OD octets.
- static bool validateControlOctet (uint8_t mc)

  Validate control octet.
- template<IoLink::DeviceDLMode DDL_MODE>
  static bool validateFrameType (uint8_t ckt)

  Validate frame type.
- static uint8_t registerReadBegin (Registers address)

  Start reading from PHY registers.
- static uint8_t registerReadNext ()

  Read next PHY register value.
- static uint8_t registerReadLast ()

  Read final PHY register value.
- static uint8_t registerRead (Registers address)

  Read a single byte from a PHY register.
- static uint8_t registerReadStatus ()

  Read the status register.
- static void registerWriteBegin (Registers address)

  Start write operation to PHY registers.
- static uint8_t registerWriteBegin (Registers address, uint8_t data)

  Start write operation to PHY registers.
- static void registerWriteNext (uint8_t data)

  Write next PHY register value.
- static void registerWriteDone ()

  Finish write access.
- static uint8_t registerWrite (Registers address, uint8_t data)

  Write a single byte to a PHY register.
- static uint8_t registerReadWriteBegin (Registers address, uint8_t data)

  Start write/read operation to PHY registers.
- static uint8_t registerReadWriteNext (uint8_t data)

  Write/read next PHY register value.
- static uint8_t registerReadWriteDone ()

  Finish write access and return final PHY register value.
- static void registerAbortAccess ()

  Abort register access.

Protected Attributes

- IoLink::DeviceDLMode _ddlMode

  Device DL-mode.
- ProcessDataIn _processDataIn

  process input data buffers
- ProcessDataOut _processDataOut

  process output data buffer
- uint8_t _deadCycleCtr

  Count of cycles from last master exchange.
- int8_t _hiZCounter

  Cycle counter for listening to the CQ.
- bool _sioLevel: 1

  Level of CQ line during SIOActive state.

Static Protected Attributes

- static SpiSsHndlr _ssHndlr

  Handler functor for SS/ line.

6.11.1 Detailed Description

Stack implementation using the PHY single-octet mode.

The PHY UART is used to handle UART frames singly.

The general operation is the same as for the multi-octet stack implementation, but now the checksum checks must be carried out by the stack.

The _ddlMode member defines the state of the data-link layer. In DDL_MODE_IDLE, the device drives the CQ line according to the sioLevel set, and the drive mode defined . The other states support IO-Link communication.

6.11.2 Implementation details

6.11.2.1 Rx UART operation

Following receipt of a UART frame, a timer is started in the PHY. If more frames are expected (no transmit is received) and the next frame is not received within the expected time, then the PHY reports a communication error and any partially received sequence is discarded. Following a parity or checksum error, any partially received sequence is similarly discarded.

6.11.2.2 Establish comms

If a short-circuit is detected by the PHY in SIO mode, (possible WURQ), updates of the SIO are temporarily silenced by introducing a delay before the next user code call. If an exchange does not complete in this time, then the stack will automatically switch back to SIO operation after ca. 100ms.

Attention

The single octet mode handshake requires the device to be in IO-Link mode. The device **must** be placed into IoLinkListen mode before the first LINK::END command is returned following the reception of the firs UART frame.

The PHY is switched to IoLinkListen mode after a delay once the line is in at high impedance in SIO operation. This allows for the reception of UART frames despite the lack of a wake-up request. This reception may be interrupted if the device drives the CQ line, and a retry is required.

6.11.3 Member Enumeration Documentation

6.11.3.1 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::CfgRegister [protected, inherited]

PHY CFG register flags.

Enumerator:

CFG_BD_38400 COM2.

CFG_BD_230400 COM3.

6.11.3.2 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> enum StackBase::Led [inherited]

LEDs.

Enumerator:

LED_1 LED 1.

LED_2 LED 2.

6.11.3.3 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::LedLevel [inherited]

LED currents.

Enumerator:

LED_LEVEL_OFF LED off.

LED_LEVEL_1 ∼0.5mA

LED_LEVEL_2 ∼1.0mA

LED_LEVEL_3 ∼1.5mA

LED_LEVEL_4 ∼2.0mA

LED_LEVEL_5 ∼2.5mA

LED_LEVEL_6 ∼3.0mA

LED_LEVEL_7 ∼3.5mA

LED_LEVEL_INVALID not a LED level

6.11.3.4 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> enum StackBase::SioDriveMode [inherited]

Drive capability in SIO mode.

Enumerator:

DRIVE_MODE_PUSH_PULL Push-pull, HS and LS active.

DRIVE_MODE_NPN LS only used.

DRIVE_MODE_PNP HS only used.

DRIVE_MODE_INACTIVE neither switch used (typical for an actuator)

6.11.4 Member Function Documentation

6.11.4.1 template<class T , int PDI, int PDO, class SpiSsHndlr > bool StackBase< T, PDI, PDO, SpiSsHndlr >::canRunUserCode (const Parameter *& *lastWrittenParameter*) [inherited]

Test if cyclic user code may run

Call this function immediately after being woken up in the application's main() loop, or, if no sleep mode is being used, at least every 0.1ms.

Parameters

lastWritten-Parameter	Pointer reference in which the function returns a Parameter structure. If the returned pointer is not NULL then the most recent message completed a write access to the direct parameter page. The data is *not* automatically written to the direct parameter page, but needs to be manually committed by calling parameterWrite(). The returned pointer may be NULL if no write access occurred.

Returns

true: user code may run now; false otherwise

6.11.4.2 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::configurePhy ( ) [protected, inherited]

Configure the PHY.

This function is called whenever the stack detects that the PHY has been reset

6.11.4.3 template<class T , int PDI, int PDO, class SpiSsHndlr > template<IoLink::DeviceDLMode DDL_MODE> int8_t StackBase< T, PDI, PDO, SpiSsHndlr >::getOdOctetCount ( ) [static, protected, inherited]

Get number of OD octets.

Template Parameters

DDL_MODE	Applicable DDL mode (see IoLink::DeviceDLMode)

Returns

Expected OD octet count

6.11.4.4 template<class T , int PDI, int PDO, class SpiSsHndlr > PhyDriver< SpiSsHndlr >::LedLevel StackBase< T, PDI, PDO, SpiSsHndlr >::ledLevel (Led *led*) const [inherited]

Get LED level.

Parameters

led	Selected LED

Returns

Current LED level

6.11.4.5 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> bool StackBase< T, PDI, PDO, SpiSsHndlr >::masterLost ( ) const [inline, inherited]

Test if connection to master has been lost.

Returns

`True` if no communication exchange took place for at least four cycles.

6.11.4.6 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::odRead (uint8_t *channel,* uint8_t *address*) [protected, inherited]

Read requested on-demand data.

Parameters

channel	Message channel (IoLink::MC_CHNL_*)
address	Address within selected channel

Returns

Data octet at address

6.11.4.7 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::odWrite (uint8_t *channel,* uint8_t *address,* uint8_t *data*) [protected, inherited]

Write received on-demand data.

Parameters

channel	Message channel (IoLink::MC_CHNL_*)
address	Address within selected channel
data	Data octet to write to address

6.11.4.8 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::parameterRead (uint8_t *address*) const [inline, inherited]

Read value from direct parameter page.

Parameters

address	Parameter index

Returns

Read value

6.11.4.9 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::parameterWrite (uint8_t *address,* uint8_t *value*) [inherited]

Write value to direct parameter page.

Parameters

address	Parameter index
value	Value to write

6.11.4.10 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> ProcessDataIn& StackBase< T, PDI, PDO, SpiSsHndlr >::processInputData ( ) [inline, inherited]

Get buffer for returning process input data from slave to master.

Returns

Process input data buffer (read/writable)

6.11.4.11 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> const ProcessDataOut& StackBase< T, PDI, PDO, SpiSsHndlr >::processOutputData ( ) const [inline, inherited]

Get buffer for process output data received from master.

Returns

Process output data buffer (read-only)

6.11.4.12 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerRead (Registers *address*) [inline, static, protected, inherited]

Read a single byte from a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY register value

See also

registerReadBegin
registerReadNext
registerReadLast

6.11.4.13 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadBegin (Registers *address*) [static, protected, inherited]

Start reading from PHY registers.

Asserts SS/ and starts reading from PHY register at specified address. This call must be followed by zero or more calls to `registerReadNext()` and a final call to registerReadLast().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY status register value

See also

registerReadNext
registerReadLast

6.11.4.14 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadLast ( ) [static, protected, inherited]

Read final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadNext

6.11.4.15 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadNext ( ) [static, protected, inherited]

Read next PHY register value.

This function automatically request the following register value. Use `registerReadLast` when reading the last required PHY register value.

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadLast

6.11.4.16 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadStatus ( ) [static, protected, inherited]

Read the status register.

Returns

PHY status register value

6.11.4.17 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteBegin (Registers *address*, uint8_t *data*) [static, protected, inherited]

Start write/read operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.11.4.18 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteDone ( ) [static, protected, inherited]

Finish write access and return final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteNext

6.11.4.19 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteNext (uint8_t *data*) [static, protected, inherited]

Write/read next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteDone

6.11.4.20 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWrite (Registers *address*, uint8_t *data*) [inline, static, protected, inherited]

Write a single byte to a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	Value to write

Returns

PHY status register

6.11.4.21 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*) [static, protected, inherited]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by one or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

See also

registerWriteNext
registerWriteDone

6.11.4.22 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*, uint8_t *data*) [static, protected, inherited]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.11.4.23 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteDone ( ) [static, protected, inherited]

Finish write access.

Waits for SPI communication to complete and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

See also

registerWriteBegin
registerWriteNext

6.11.4.24 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteNext (uint8_t *data*) [static, protected, inherited]

Write next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

See also

registerWriteBegin
registerWriteDone

6.11.4.25 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setIoLinkListen ( ) [static, protected, inherited]

Put PHY in IO-Link-Listen mode.

Returns

PHY status register

Reimplemented in StackTransparent.

6.11.4.26 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::setLedLevel (Led *led*, typename Phy::LedLevel *level*) [inherited]

Set LED level.

Parameters

led	Selected LED
level	Desired level

6.11.4.27 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setSioActive ( ) [protected, inherited]

Put PHY in SIO-Active state.

The CQ line is driven according to the setting of the HS and LS bits, as specified by Stack::SIO_DRIVE_MODE.

Returns

PHY status register

Reimplemented in StackTransparent.

6.11.4.28 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::setSioLevel (bool *active*) [inherited]

Specify the level of the CQ line in SIO mode (STACK_MODE_SIO) state.

Parameters

active	If Stack::SIO_DRIVE_MODE is Stack::DRIVE_MODE_PNP or Stack::DRIVE_MODE_NPN the relevant switch is activated if `active` == true. In Stack::DRIVE_MODE_PUSH_PULL (push-pull) CQ is driven high (`active` == true) or low (`active` == false).

6.11.4.29 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::setSioListen ( ) [static, protected, inherited]

Put PHY in SIO-Listen mode.

Returns

PHY status register

Reimplemented in StackTransparent.

6.11.4.30 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackT& StackBase< T, PDI, PDO, SpiSsHndlr >::stack ( ) [inline, protected, inherited]

Helper function returning derived stack specialization.

Returns

Derived stack instance

6.11.4.31 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackMode StackBase< T, PDI, PDO, SpiSsHndlr >::stackMode ( ) const [inline, inherited]

Get current stack mode.

Returns

STACK_MODE_SIO or STACK_MODE_IOLINK

6.11.4.32 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::startCallbackTimer (uint8_t *delay* = 0) [protected, inherited]

Start / synchronize user-callback timer.

Starts a timer which ensures that the user call-back will be called on a regular basis, even if no master message has been received

Parameters

delay	Delay until first call to user callback (in 1/10ms units)

6.11.4.33 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::temperature ( ) const [inherited]

Get current measured temperature value.

In order to convert the returned value to a Celsius reading, the following formula should be applied:

Temperature [Celsius] = (80 - temp) $*$ 2.70

Returns

Current temperature value

6.11.4.34 template<class T , int PDI, int PDO, class SpiSsHndlr > template<IoLink::DeviceDLMode DDL_MODE> bool StackBase< T, PDI, PDO, SpiSsHndlr >::validateFrameType (uint8_t *ckt*) [static, protected, inherited]

Validate frame type.

Template Parameters

DDL_MODE	Applicable DDL mode (see IoLink::DeviceDLMode)

Parameters

ckt	Received CKT frame octet.

Returns

True if frame type valid, false otherwise

6.11.4.35 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackBase< T, PDI, PDO, SpiSsHndlr >::void::TIMER0_COMPB_vect ( ) [protected, inherited]

The ISR function can access the stack state, and is declared here as a friend.

6.11.5 Member Data Documentation

6.11.5.1 const uint8_t StackSingleByte::MSEQ_CAPABILITY [static]

Initial value:

```
IoLink::MSEQCAP_ISDU_NOT_SUPPORTED |
                                        IoLink::MSEQCAP_OP_CODE_0 |
                                        IoLink::MSEQCAP_PREOP_CODE_0
```

M-sequence Capability (Direct Parameter 0x03)

6.11.5.2 const uint8_t StackSingleByte::PHY_CFG [static]

Initial value:

```
    CFG_UVT_16_3V | CFG_RF_ABS | CFG_S5V_3_3V
                          | (BAUD_RATE == 38400 ? CFG_BD_38400 :
CFG_BD_230400)
```

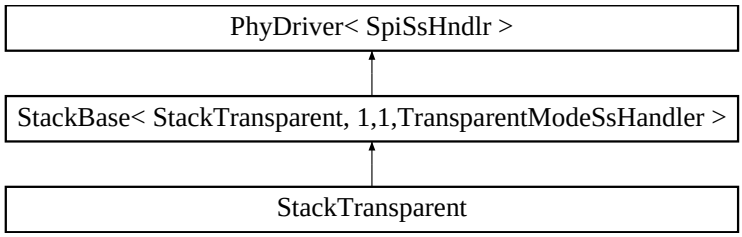
PHY configuration.

6.12 StackTransparent Class Reference

Stack implementation using the PHY transparent mode.

`#include <stacktransparent.h>`

Inheritance diagram for StackTransparent:



Public Types

- enum Led { LED_1, LED_2 }

  LEDs.
- enum StackMode { **STACK_MODE_SIO**, **STACK_MODE_IOLINK** }

  Stack mode.
- enum SioDriveMode { DRIVE_MODE_PUSH_PULL = 0, DRIVE_MODE_NPN, DRIVE_MODE_PNP, DRIVE_MODE_INACTIVE }

  Drive capability in SIO mode.
- typedef PhyDriver< SpiSsHndlr > Phy

  Convenience typedef for PhyDriver.
- typedef T StackT

  Specific derived stack type.
- typedef StackBase< T, PDI, PDO, SpiSsHndlr > BaseT

  Convenience typedef for this class.
- typedef ProcessData< PDI > ProcessDataIn

  Input process data.
- typedef ProcessData< PDO > ProcessDataOut

  Output process data.
- enum LedLevel { LED_LEVEL_OFF, LED_LEVEL_1, LED_LEVEL_2, LED_LEVEL_3, LED_LEVEL_4, LED_LEVEL_5, LED_LEVEL_6, LED_LEVEL_7, **LED_LEVEL_MAX** = LED_LEVEL_7, LED_LEVEL_INVALID }

  LED currents.

Public Member Functions

- void configureStack ()

  Configure the specific stack.
- uint8_t setSioActive ()

reimplementation

- uint8_t setSioListen ()

  reimplementation

- uint8_t setIoLinkListen ()

  reimplementation

- void configure ()

  Configure sets up the hardware resources on the uC, and initializes the stack.

- bool canRunUserCode (const Parameter *&lastWrittenParameter)
- StackMode stackMode () const

  Get current stack mode.

- bool masterLost () const

  Test if connection to master has been lost.

- bool flag () const

  Debugging flag.

- void setSioLevel (bool active)

  Specify the level of the CQ line in SIO mode (STACK_MODE_SIO) state.

- uint8_t parameterRead (uint8_t address) const

  Read value from direct parameter page.

- void parameterWrite (uint8_t address, uint8_t value)

  Write value to direct parameter page.

- ProcessDataIn & processInputData ()

  Get buffer for returning process input data from slave to master.

- const ProcessDataOut & processOutputData () const

  Get buffer for process output data received from master.

- void setLedLevel (Led led, typename Phy::LedLevel level)

  Set LED level.

- Phy::LedLevel ledLevel (Led led) const

  Get LED level.

- uint8_t temperature () const

  Get current measured temperature value.

Static Public Member Functions

- static void stopInterrupt ()

  Temporarily disable the ISR.

- static void restartInterrupt ()

  Restart the ISR.

Static Public Attributes

- static const uint8_t REVISION_ID = IoLink::REVISION_ID_1_1

  RevisionID of protocol implemented (Direct Parameter 0x04)
- static const uint16_t VENDOR_ID = 0x01a6

  VendorID (Direct Parameters 0x07 and 0x08)
- static const uint32_t DEVICE_ID = 0x123456

  DeviceID (Direct Parameters 0x09 - 0x0b)
- static const uint32_t BAUD_RATE = 38400

  Communication speed (must be either 38400 or 230400)
- static const uint8_t MIN_CYC_TIME = 30

  MinCycleTime in 0.1ms units.
- static const uint8_t MSEQ_CAPABILITY

  M-sequence Capability (Direct Parameter 0x03)
- static const uint8_t PHY_CFG

  PHY configuration.
- static const uint8_t PHY_CTL_SCT = CTL_SCT_190MA

  PHY short-circuit threshold.
- static const uint8_t PHY_CTL_MODE = CTL_TRNS_MODE

  PHY mode (must be CTL_TRNS_MODE)
- static enum SioDriveMode SIO_DRIVE_MODE = DRIVE_MODE_PUSH_PULL

  PHY drive mode to use in SIO mode.
- static const uint8_t PHY_THERM_DEG = 175

  PHY thermal shutdown temperature (in degrees centigrade)
- static StackTransparent instance

  The one and only stack instance.
- static const uint8_t PD_IN_SIZE = PDI

  Amount of input process data (in octets)
- static const uint8_t PD_OUT_SIZE = PDO

  Amount of output process data (in octets)

Protected Types

- enum HandlerResult { **ResultSuccess** = 0, **ResultNoData**, **ResultChecksumError**, **ResultIllegalMessageType**, **ResultPhyReset** }

  Result codes for ISR sub-handlers.
- enum MseqRegister { **MSEQ_M2CNT_SHIFT** = 2, **MSEQ_OD_1** = 0 << 0, **MSEQ_OD_2** = 1 << 0, **MSEQ_OD_8** = 2 << 0 }

  PHY MSEQ register flags.
- enum CfgRegister { **CFG_NONE** = 0, **CFG_UVT_18_0V** = 0 << 5, **CFG_UVT_16_3V** = 1 << 5, **CFG_UVT_15_0V** = 2 << 5, **CFG_UVT_13_9V** = 3 << 5, **CFG_UVT_12_0V** = 4 << 5, **CFG_UVT_10_0V** = 5 << 5, **CFG_UVT_8_6V** = 6 << 5, **CFG_UVT_7_2V** = 7 << 5, CFG_BD_38400 = 0 << 4, CFG_BD_230400 = 1 << 4, **CFG_RF_ABS** = 0 << 3, **CFG_RF_REL** = 1 << 3, **CFG_S5V_SS** = 0 << 0, **CFG_S5V_3_3V** = 2 << 0, **CFG_S5V_5_0V** = 3 << 0 }

PHY CFG register flags.

- enum CtlRegister { **CTL_NONE** = 0, **CTL_TRNS_MODE** = 1 << 7, **CTL_SCT_190MA** = 0 << 4, **CTL_SCT_210MA** = 1 << 4, **CTL_SCT_230MA** = 2 << 4, **CTL_SCT_250MA** = 3 << 4, **CTL_SCT_110MA** = 4 << 4, **CTL_SCT_130MA** = 5 << 4, **CTL_SCT_150MA** = 6 << 4, **CTL_SCT_170MA** = 7 << 4, **CTL_SGL_MODE** = 1 << 3, **CTL_IEN_MODE** = 1 << 3, **CTL_IOLINK_MODE** = 0 << 2, **CTL_DIO** = 1 << 2, **CTL_JOIN** = 0 << 2, **CTL_SIO_MODE** = 1 << 2, **CTL_HS** = 1 << 1, **CTL_LS** = 1 << 0 }

  PHY CTL register flags.
- enum LinkRegister { **LINK_NONE** = 0, **LINK_CNT_MASK** = 0x3C, **LINK_CNT_SHIFT** = 2, **LINK_END** = 1 << 1, **LINK_SND** = 1 << 0 }

  PHY LINK register flags.
- enum StatusRegister { **STATUS_NONE** = 0, **STATUS_RST** = 1 << 7, **STATUS_INT** = 1 << 6, **STATUS_UV** = 1 << 5, **STATUS_DINT** = 1 << 4, **STATUS_CHK** = 1 << 3, **STATUS_DAT** = 1 << 2, **STATUS_SSC** = 1 << 1, **STATUS_SOT** = 1 << 0 }

  PHY STATUS register flags.
- enum TempRegister { **TEMP_NONE** = 0 }

  PHY TEMP register flags.
- enum DcDcRegister { **DCDC_NONE** = 0, **DCDC_DIS** = 1 << 7, **DCDC_BYP** = 1 << 6, **DCDC_FSET_500kHz** = 4 << 3, **DCDC_FSET_625kHz** = 5 << 3, **DCDC_FSET_710kHz** = 6 << 3, **DCDC_FSET_830kHz** = 7 << 3, **DCDC_FSET_1000kHz** = 0 << 3, **DCDC_FSET_1250kHz** = 1 << 3, **DCDC_FSET_1670kHz** = 2 << 3, **DCDC_FSET_2000kHz** = 3 << 3, **DCDC_VSET_4V2** = 4, **DCDC_VSET_4V5** = 5, **DCDC_VSET_4V9** = 6, **DCDC_VSET_5V4** = 7, **DCDC_VSET_6V0** = 0, **DCDC_VSET_6V7** = 1, **DCDC_VSET_7V8** = 2, **DCDC_VSET_9V5** = 3 }

  HMT7748 DCDC register flags.
- enum DstatRegister { **DSTAT_NONE** = 0, **DSTAT_LVL** = 1 << 2, **DSTAT_SSC** = 1 << 1 }

  HMT7748 DSTAT register flags.

Protected Member Functions

- StackT & stack ()

  Helper function returning derived stack specialization.
- void configureStackBase ()

  Configure the stack base class.
- void configurePhy ()

  Configure the PHY.
- void odWrite (uint8_t channel, uint8_t address, uint8_t data)

  Write received on-demand data.
- uint8_t odRead (uint8_t channel, uint8_t address)

  Read requested on-demand data.
- void updateCyclePeriod ()

  Calculate cycle period from MasterCycleTime.
- void startCallbackTimer (uint8_t delay=0)

  Start / synchronize user-callback timer.
- void onTimer0CompBInterrupt ()

  ISR handler as member function.
- friend void::TIMER0_COMPB_vect ()

  The ISR function can access the stack state, and is declared here as a friend.

Static Protected Member Functions

- template<IoLink::DeviceDLMode DDL_MODE>
  static int8_t getOdOctetCount ()

  Get number of OD octets.
- static bool validateControlOctet (uint8_t mc)

  Validate control octet.
- template<IoLink::DeviceDLMode DDL_MODE>
  static bool validateFrameType (uint8_t ckt)

  Validate frame type.
- static uint8_t registerReadBegin (Registers address)

  Start reading from PHY registers.
- static uint8_t registerReadNext ()

  Read next PHY register value.
- static uint8_t registerReadLast ()

  Read final PHY register value.
- static uint8_t registerRead (Registers address)

  Read a single byte from a PHY register.
- static uint8_t registerReadStatus ()

  Read the status register.
- static void registerWriteBegin (Registers address)

  Start write operation to PHY registers.
- static uint8_t registerWriteBegin (Registers address, uint8_t data)

  Start write operation to PHY registers.
- static void registerWriteNext (uint8_t data)

  Write next PHY register value.
- static void registerWriteDone ()

  Finish write access.
- static uint8_t registerWrite (Registers address, uint8_t data)

  Write a single byte to a PHY register.
- static uint8_t registerReadWriteBegin (Registers address, uint8_t data)

  Start write/read operation to PHY registers.
- static uint8_t registerReadWriteNext (uint8_t data)

  Write/read next PHY register value.
- static uint8_t registerReadWriteDone ()

  Finish write access and return final PHY register value.
- static void registerAbortAccess ()

  Abort register access.

Protected Attributes

- IoLink::DeviceDLMode _ddlMode
  Device DL-mode.
- ProcessDataIn _processDataIn
  process input data buffers
- ProcessDataOut _processDataOut
  process output data buffer
- uint8_t _deadCycleCtr
  Count of cycles from last master exchange.
- int8_t _hiZCounter
  Cycle counter for listening to the CQ.
- bool _sioLevel: 1
  Level of CQ line during SIOActive state.

Static Protected Attributes

- static SpiSsHndlr _ssHndlr
  Handler functor for SS/ line.

6.12.1 Detailed Description

Stack implementation using the PHY transparent mode.

The internal UART of the AVR is used, which requires the connection of an external crystal or oscillator to give the required frequency stability. A frequency of 18.432MHz is used as standard, which is internally divided for 38.4kBaud and 230.4kBaud operation.

The general operation is the same as for the multi-octet stack implementation, but now all of the low-level checks and most of the timing must be carried out by the AVR.

The _ddlMode member defines the state of the data-link layer. In DDL_MODE_IDLE, the device drives the CQ line according to the sioLevel set, and the drive mode defined . The other states support IO-Link communication.

6.12.2 Implementation details

6.12.2.1 Rx UART operation

The UART is enabled, except under the following conditions:

- in DDL_MODE_IDLE if CQ is driven
- in DDL_MODE_IDLE for a period (dead cycles) after the CQ line was driven
- when transmitting using the AVR UART

Following receipt of a UART frame, a timer is started. If more frames are expected and the next frame is not received within the expected time, then any partially received sequence is discarded. Following a parity or checksum error, any partially received sequence is similarly discarded.

6.12.2.2 Establish comms

If a short-circuit is detected by the PHY in DDL_MODE_IDLE, then the stack switches immediately DDL_MODE_ESTABLISH_COM, and starts listening for UART frames. Typically the end of the wake-up pulse and any communications at higher frequencies than the operational frequency of the device are rejected as invalid frames. In DDL_MODE_ESTABLISH_COM the CQ line is not driven by the device.

If a complete, correct, master sequence is received in either mode DDL_MODE_ESTABLISH_COM (following a wake-up request, WURQ) or in DDL_MODE_IDLE (when the line is not driven), then the _ddlMode is changed to DDL_MODE_STARTUP. The device will now only leave IO-Link operation if a FALLBACK command is sent by the master.

If the PHY reports that the short-circuit condition no longer exists while we are in DDL_MODE_ESTABLISH_COM, then we return to SIO operation and DDL_MODE_IDLE. The PHY reports a short-circuit condition for ca. 100ms after a short-circuit condition is detected on the line as part of the output self-protection.

6.12.2.3 SPI communication

The PHY MOSI and MISO lines are used to transmit both SPI data and to transmit and receive data from the AVR UART. The AVR UART RX and TX pins must be connected to the AVR MISO and MOSI pins respectively, and suitable jumper resistor footprints are provided on the GENIE Explorer boards.

Once IO-Link communication is established, the predictable timing of the IO-Link exchanges means that a conflict on the lines can be avoided. SPI communication is carried out immediately after sending on the UART TX line.

Following a recognised wake-up request (WURQ) the call-back timer is stopped, suspending SPI communication while waiting for the master.

In DDL_MODE_IDLE it is possible that an SPI transmission may conflict with an incoming master sequence, which will corrupt the sequence. (Note, it is also possible that the device may start to drive the CQ line in normal SIO operation, similarly corrupting the sequence. An inverted line state in a high impedance condition is not considered sufficient evidence for presence of a master to block SIO operation.)

For the StackTransparent, the PHY MOSI line is normally driven by the UART TX pin. This operation is only interrupted for actual SPI communications. It would be possible to use the AVR UART as an SPI driver and so avoid any switching, but this has not been implemented here.

The PORTD1 retains the state of the SIO output during SPI communication and ensures that the correct level is reestablished before the SS line is released (set to '1'). PORTD1 is also set to '1' before enabling transmission through the PHY, so that the line output level on CQ is correctly driven low while the output is enabled before and after the UART itself is sending.

In transparent mode, the PHY permanently drives the MISO line. This is incompatible with the PHY sharing the bus with other SPI slaves, and also interferes with programming the AVR over the SPI lines. (Programming via the reset pin is still possible). For stack development purposes we have inserted a 270ohm resistor between the PHY and the AVR to allow programming over the SPI lines.

6.12.3 Member Enumeration Documentation

6.12.3.1 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::CfgRegister [protected, inherited]

PHY CFG register flags.

Enumerator:

CFG_BD_38400 COM2.

CFG_BD_230400 COM3.

6.12.3.2 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> enum StackBase::Led [inherited]

LEDs.

Enumerator:

LED_1 LED 1.

LED_2 LED 2.

6.12.3.3 template<class SpiSsHndlr = DefaultSsHandler> enum PhyDriver::LedLevel [inherited]

LED currents.

Enumerator:

LED_LEVEL_OFF LED off.

LED_LEVEL_1 ∼0.5mA

LED_LEVEL_2 ∼1.0mA

LED_LEVEL_3 ∼1.5mA

LED_LEVEL_4 ∼2.0mA

LED_LEVEL_5 ∼2.5mA

LED_LEVEL_6 ∼3.0mA

LED_LEVEL_7 ∼3.5mA

LED_LEVEL_INVALID not a LED level

6.12.3.4 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> enum StackBase::SioDriveMode [inherited]

Drive capability in SIO mode.

Enumerator:

DRIVE_MODE_PUSH_PULL Push-pull, HS and LS active.

DRIVE_MODE_NPN LS only used.

DRIVE_MODE_PNP HS only used.

DRIVE_MODE_INACTIVE neither switch used (typical for an actuator)

6.12.4 Member Function Documentation

6.12.4.1 template<class T , int PDI, int PDO, class SpiSsHndlr > bool StackBase< T, PDI, PDO, SpiSsHndlr >::canRunUserCode (const Parameter *& *lastWrittenParameter*) [inherited]

Test if cyclic user code may run

Call this function immediately after being woken up in the application's main() loop, or, if no sleep mode is being used, at least every 0.1ms.

Parameters

lastWrittenParameter	Pointer reference in which the function returns a Parameter structure. If the returned pointer is not NULL then the most recent message completed a write access to the direct parameter page. The data is *not* automatically written to the direct parameter page, but needs to be manually committed by calling parameterWrite(). The returned pointer may be NULL if no write access occurred.

Returns

true: user code may run now; false otherwise

6.12.4.2 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::configurePhy ( ) [protected, inherited]

Configure the PHY.

This function is called whenever the stack detects that the PHY has been reset

6.12.4.3 template<class T , int PDI, int PDO, class SpiSsHndlr > template<IoLink::DeviceDLMode DDL_MODE> int8_t StackBase< T, PDI, PDO, SpiSsHndlr >::getOdOctetCount ( ) [static, protected, inherited]

Get number of OD octets.

Template Parameters

DDL_MODE	Applicable DDL mode (see IoLink::DeviceDLMode)

Returns

Expected OD octet count

6.12.4.4 template<class T , int PDI, int PDO, class SpiSsHndlr > PhyDriver< SpiSsHndlr >::LedLevel StackBase< T, PDI, PDO, SpiSsHndlr >::ledLevel (Led *led*) const [inherited]

Get LED level.

Parameters

led	Selected LED

Returns

Current LED level

6.12.4.5 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> bool StackBase< T, PDI, PDO, SpiSsHndlr >::masterLost ( ) const [inline, inherited]

Test if connection to master has been lost.

Returns

`True` if no communication exchange took place for at least four cycles.

6.12.4.6 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::odRead (uint8_t *channel,* uint8_t *address*) [protected, inherited]

Read requested on-demand data.

Parameters

channel	Message channel (IoLink::MC_CHNL_∗)
address	Address within selected channel

Returns

Data octet at address

6.12.4.7 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::odWrite (uint8_t *channel,* uint8_t *address,* uint8_t *data*) [protected, inherited]

Write received on-demand data.

Parameters

channel	Message channel (IoLink::MC_CHNL_∗)
address	Address within selected channel
data	Data octet to write to address

6.12.4.8 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::parameterRead (uint8_t *address*) const [inline, inherited]

Read value from direct parameter page.

Parameters

address	Parameter index

Returns

Read value

6.12.4.9 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::parameterWrite (uint8_t *address,* uint8_t *value*) [inherited]

Write value to direct parameter page.

Parameters

address	Parameter index
value	Value to write

6.12.4.10 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> ProcessDataIn& StackBase< T, PDI, PDO, SpiSsHndlr >::processInputData ( ) [inline, inherited]

Get buffer for returning process input data from slave to master.

Returns

Process input data buffer (read/writable)

6.12.4.11 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> const ProcessDataOut& StackBase< T, PDI, PDO, SpiSsHndlr >::processOutputData ( ) const [inline, inherited]

Get buffer for process output data received from master.

Returns

Process output data buffer (read-only)

6.12.4.12 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerRead (Registers *address*) [inline, static, protected, inherited]

Read a single byte from a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY register value

See also

registerReadBegin
registerReadNext
registerReadLast

6.12.4.13 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadBegin (Registers *address*) [static, protected, inherited]

Start reading from PHY registers.

Asserts SS/ and starts reading from PHY register at specified address. This call must be followed by zero or more calls to `registerReadNext()` and a final call to registerReadLast().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

Returns

PHY status register value

See also

registerReadNext
registerReadLast

6.12.4.14 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadLast ( ) [static, protected, inherited]

Read final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadNext

6.12.4.15 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadNext ( ) [static, protected, inherited]

Read next PHY register value.

This function automatically request the following register value. Use `registerReadLast` when reading the last required PHY register value.

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value

See also

registerReadBegin
registerReadLast

6.12.4.16 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadStatus ( ) [static, protected, inherited]

Read the status register.

Returns

PHY status register value

6.12.4.17 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteBegin (Registers *address*, uint8_t *data*) [static, protected, inherited]

Start write/read operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.12.4.18 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteDone ( ) [static, protected, inherited]

Finish write access and return final PHY register value.

Reads final PHY register value and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteNext

6.12.4.19 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerReadWriteNext (uint8_t *data*) [static, protected, inherited]

Write/read next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

Returns

PHY register value at previous address

See also

registerWriteBegin
registerWriteDone

6.12.4.20 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWrite (Registers *address*, uint8_t *data*) [inline, static, protected, inherited]

Write a single byte to a PHY register.

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	Value to write

Returns

PHY status register

6.12.4.21 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*) [static, protected, inherited]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by one or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register

See also

registerWriteNext
registerWriteDone

6.12.4.22 template<class SpiSsHndlr > uint8_t PhyDriver< SpiSsHndlr >::registerWriteBegin (Registers *address*, uint8_t *data*) [static, protected, inherited]

Start write operation to PHY registers.

Asserts SS/ and starts writing to PHY register at specified address. This call must be followed by zero or more calls to `registerWriteNext()` and a final call to registerWriteDone().

Warning

Ensure interrupts disabled before calling!

Parameters

address	PHY register
data	PHY register value to write at address

Returns

PHY status register value

See also

registerWriteNext
registerWriteDone

6.12.4.23 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteDone ( ) [static, protected, inherited]

Finish write access.

Waits for SPI communication to complete and de-asserts SS/

Warning

Ensure interrupts disabled before calling!

See also

registerWriteBegin
registerWriteNext

6.12.4.24 template<class SpiSsHndlr > void PhyDriver< SpiSsHndlr >::registerWriteNext (uint8_t *data*) [static, protected, inherited]

Write next PHY register value.

Use `registerWriteDone` to finish the write operation.

Warning

Ensure interrupts disabled before calling!

Parameters

data	PHY register value to write at next address

See also

registerWriteBegin
registerWriteDone

6.12.4.25 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::setLedLevel (Led *led,* typename Phy::LedLevel *level*) [inherited]

Set LED level.

Parameters

led	Selected LED
level	Desired level

6.12.4.26 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::setSioLevel (bool *active*) [inherited]

Specify the level of the CQ line in SIO mode (STACK_MODE_SIO) state.

Parameters

active	If Stack::SIO_DRIVE_MODE is Stack::DRIVE_MODE_PNP or Stack::DRIVE_MODE_NPN the relevant switch is activated if `active` == true. In Stack::DRIVE_MODE_PUSH_PULL (push-pull) CQ is driven high (`active` == true) or low (`active` == false).

6.12.4.27 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackT& StackBase< T, PDI, PDO, SpiSsHndlr >::stack ( ) [inline, protected, inherited]

Helper function returning derived stack specialization.

Returns

Derived stack instance

6.12.4.28 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackMode StackBase< T, PDI, PDO, SpiSsHndlr >::stackMode ( ) const [inline, inherited]

Get current stack mode.

Returns

STACK_MODE_SIO or STACK_MODE_IOLINK

6.12.4.29 template<class T , int PDI, int PDO, class SpiSsHndlr > void StackBase< T, PDI, PDO, SpiSsHndlr >::startCallbackTimer (uint8_t *delay* = 0) [protected, inherited]

Start / synchronize user-callback timer.

Starts a timer which ensures that the user call-back will be called on a regular basis, even if no master message has been received

Parameters

delay	Delay until first call to user callback (in 1/10ms units)

6.12.4.30 template<class T , int PDI, int PDO, class SpiSsHndlr > uint8_t StackBase< T, PDI, PDO, SpiSsHndlr >::temperature ( ) const [inherited]

Get current measured temperature value.

In order to convert the returned value to a Celsius reading, the following formula should be applied:

Temperature [Celsius] = (80 - temp) $*$ 2.70

Returns

Current temperature value

6.12.4.31 template<class T , int PDI, int PDO, class SpiSsHndlr > template<IoLink::DeviceDLMode DDL_MODE> bool StackBase< T, PDI, PDO, SpiSsHndlr >::validateFrameType (uint8_t *ckt*) [static, protected, inherited]

Validate frame type.

Template Parameters

DDL_MODE	Applicable DDL mode (see IoLink::DeviceDLMode)

Parameters

ckt	Received CKT frame octet.

Returns

True if frame type valid, false otherwise

6.12.4.32 template<class T, int PDI, int PDO, class SpiSsHndlr = DefaultSsHandler> StackBase< T, PDI, PDO, SpiSsHndlr >::void::TIMER0_COMPB_vect ( ) [protected, inherited]

The ISR function can access the stack state, and is declared here as a friend.

6.12.5 Member Data Documentation

6.12.5.1 const uint8_t StackTransparent::MSEQ_CAPABILITY [static]

Initial value:

```
IoLink::MSEQCAP_ISDU_NOT_SUPPORTED |
                                        IoLink::MSEQCAP_OP_CODE_0 |
                                        IoLink::MSEQCAP_PREOP_CODE_0
```

M-sequence Capability (Direct Parameter 0x03)

6.12.5.2 const uint8_t StackTransparent::PHY_CFG [static]

Initial value:

```
        CFG_UVT_16_3V | CFG_RF_ABS | CFG_S5V_3_3V
                              | (BAUD_RATE == 38400 ? CFG_BD_38400 :
    CFG_BD_230400)
```

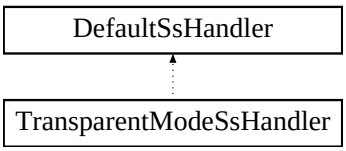
PHY configuration.

6.13 TransparentModeSsHandler Class Reference

Implementation of class for handling SPI SS/ line with TX save/restore.

`#include <stacktransparent.h>`

Inheritance diagram for TransparentModeSsHandler:



Public Member Functions

- void configure ()

  Reimplementation so that PORTB3/MOSI is not active by default.
- bool isReading () const

  Test if reading.

Static Public Member Functions

- static void assert ()

  Assert SS/ to begin SPI communication.
- static void deassert ()

  Deassert SS/ to terminate SPI communication.
- static void setReading (bool set=true)

  Set read/write mode.

Static Private Member Functions

- static bool asserted ()

  Test if SS/ asserted.

6.13.1 Detailed Description

Implementation of class for handling SPI SS/ line with TX save/restore.

Caching of the TX line is done by the port register (PORTB) itself. This class simply enables / disables SPI to toggle between SPI and PORTB register value.

Chapter 7

File Documentation

7.1 debugpin.h File Reference

Declares the DebugPin class.

```
#include <avr/io.h>
```

Classes

- class DebugPin

  Class to control pins on JP4 extension connector.

7.1.1 Detailed Description

Declares the DebugPin class.

7.2 demoapp.h File Reference

Declares the DemoApp class.

Classes

- class DemoApp

  The DemoApp is a demonstration application which uses the Mini-stack software.

7.2.1 Detailed Description

Declares the DemoApp class.

7.3 iolink.h File Reference

Declares the IO-Link class.

```
#include <avr/pgmspace.h>
```

Classes

- class IoLink

  Helper class for supporting IO-Link standard.

Defines

- #define ENCODE_PD_BYTES(PDBYTES)

  Encodes desired number of process input data into ProcessDataIn / ProcessDataOut direct parameter.
- #define ENCODE_CYCLE_TIME(DECIMS)

  Encode desired cycle time into MasterCycleTime or MinCycleTime parameter.
- #define DECODE_CYCLE_TIME(CYC)

  Decode MasterCycleTime or MinCycleTime parameter into cycle time in 1/10 of ms.

7.3.1 Detailed Description

Declares the IO-Link class.

7.3.2 Define Documentation

7.3.2.1 #define DECODE_CYCLE_TIME(*CYC*)

Value:

```
((CYC) < 4          ? (uint16_t)0 :
        \
    (CYC) < 64         ? (uint16_t)(CYC) :
           \
    ((CYC) & IoLink::CYC_TIME_BASE_MASK) == IoLink::CYC_TIME_BASE_0_4_MS ?
           \
        ((uint16_t)((CYC) & IoLink::CYC_MULTIPLIER_MASK) << 2) + 64 :
           \
    ((CYC) & IoLink::CYC_TIME_BASE_MASK) == IoLink::CYC_TIME_BASE_1_6_MS ?
           \
        ((uint16_t)((CYC) & IoLink::CYC_MULTIPLIER_MASK) << 4) + 320 :
           \
    0)
```

Decode MasterCycleTime or MinCycleTime parameter into cycle time in 1/10 of ms.

Parameters

CYC	Parameter value

Returns

Cycle time in 1/10 of ms, or 0 if parameter `CYC` out of valid range

7.3.2.2 #define ENCODE_CYCLE_TIME(*DECIMS*)

Value:

```
((DECIMS) < 4       ? 0 :
         \
     (DECIMS) < 64      ? ((DECIMS) | IoLink::CYC_TIME_BASE_0_1_MS) :
              \
     (DECIMS) < 320     ? ((((DECIMS) - 64) >> 2) | IoLink::CYC_TIME_BASE_0_4_MS)
        :    \
     (DECIMS) < 1328    ? ((((DECIMS) - 320) >> 4) | IoLink::CYC_TIME_BASE_1_6_MS
       ) :   \
     0)
```

Encode desired cycle time into MasterCycleTime or MinCycleTime parameter.

The resulting parameter may not correspond exactly to the desired input value. Use `decodeCycleTime` to obtain the exact value.

Parameters

DECIMS	Cycle time in 1/10 of ms

Returns

Parameter value, or 0 if parameter `DECIMS` out of valid range

7.3.2.3 #define ENCODE_PD_BYTES(*PDBYTES*)

Value:

```
((PDBYTES) == 0   ? (IoLink::PDOUT_BITS) :                                \
    (PDBYTES) == 1   ? (IoLink::PDOUT_BITS | 8) :                           \
    (PDBYTES) == 2   ? (IoLink::PDOUT_BITS | 16) :                          \
    (PDBYTES) <= 32  ? (IoLink::PDOUT_BYTES_PLUS_1 | ((PDBYTES) - 1)) :     \
    0)
```

Encodes desired number of process input data into ProcessDataIn / ProcessDataOut direct parameter.

Parameters

PDBYTES	Number of bytes in process data (0 .. 32)

Returns

Value for ProcessDataIn parameter, or 0 if PDBYTES out of valid range

7.4 phydriver.h File Reference

Declares the PhyDriver.

```
#include <avr/io.h> #include <avr/interrupt.h> #include <stdint.h> #include
"spi.h"
```

Classes

- class PhyDriver< SpiSsHndlr >

  Static class implementing register access to the PHY.

Defines

- #define ENCODE_THERMAL_SHUTDOWN(TEMPC)

  Calculates set-point value of PHY THERM:TH[4:0] register.
- #define PROGMEM_ __attribute__((section(".progmem.data")))

  Alternative to PROGMEM storage class.

Functions

- **ISR** (PCINT1_vect)

7.4.1 Detailed Description

Declares the PhyDriver.

7.4.2 Define Documentation

7.4.2.1 #define ENCODE_THERMAL_SHUTDOWN(*TEMPC*)

Value:

```
((TEMPC) > 200 ? 0 :                                                    \
     (TEMPC) < -40 ? 23 :                                                   \
    (((8000 - 37 * (TEMPC)) / 400) & 0x1f))
```

Calculates set-point value of PHY THERM:TH[4:0] register.

The next higher available set-point temperature will be selected.

Parameters

TEMPC	Set-point temperature in degree Celsius in range [-40 °C .. +200 °C]

Returns

THERM:TH[4:0] register value

7.4.2.2 #define PROGMEM_ __attribute__((section(".progmem.data")))

Alternative to PROGMEM storage class.

Same effect as PROGMEM storage class, but avoiding erroneous warning by GCC.

See also

http://gcc.gnu.org/bugzilla/show_bug.cgi?id=34734

7.5 spi.h File Reference

Declares the Spi class.

Classes

- struct Spi
  Helper class for handling SPI communication with the HMT7742.
- struct DefaultSsHandler
  Default implementation of class for handling SPI SS/ line.

7.5.1 Detailed Description

Declares the Spi class.

7.6 stackbase.h File Reference

Declares the StackBase class.

```
#include "debugpin.h" #include "phydriver.h" #include "iolink.h" #include <stddef.-
h>
```

Classes

- class StackBase< T, PDI, PDO, SpiSsHndlr >
  The StackBase is the base class for minimal IO-Link stacks for different IO-Link devices.
- struct StackBase< T, PDI, PDO, SpiSsHndlr >::Parameter
  Parameter structure.
- struct StackBase< T, PDI, PDO, SpiSsHndlr >::ProcessData< SIZE >
  Structure for holding process data and associated status flags.

Defines

- #define LOBYTE(w) ((uint8_t)(((w)) & 0xff))

  Returns LSB of argument.
- #define HIBYTE(w) ((uint8_t)((((w)) >> 8) & 0xff))

  Returns MSB of argument.
- #define BANKBYTE(w) ((uint8_t)((((w)) >> 16) & 0xff))

  Returns bank byte of argument.

Functions

- **ISR** (TIMER0_COMPA_vect)
- **ISR** (TIMER0_COMPB_vect)
- **ISR** (PCINT0_vect)

Variables

- uint8_t directParameter [32]

  direct parameter page
- uint8_t eventPage [8]

  event buffer

7.6.1 Detailed Description

Declares the StackBase class.

7.6.2 Define Documentation

7.6.2.1 #define BANKBYTE(w) ((uint8_t)((((w)) >> 16) & 0xff))

Returns bank byte of argument.

Parameters

w	Argument

Returns

Bank byte (bits 16..23) of argument

7.6.2.2 #define HIBYTE(w) ((uint8_t)((((w)) >> 8) & 0xff))

Returns MSB of argument.

Parameters

w	Argument

Returns

MSB (bits 8..15) of argument

7.6.2.3 #define LOBYTE(w) ((uint8_t)(((w)) & 0xff))

Returns LSB of argument.

Parameters

w	Argument

Returns

LSB (bits 0..7) of argument

7.7 stackmultibyte.h File Reference

Declares the StackMultiByte class.

```
#include "stackbase.h"
```

Classes

- class StackMultiByte
  Stack implementation using multi-byte mode.

Typedefs

- typedef StackMultiByte Stack
  Alias for selected stack type.

7.7.1 Detailed Description

Declares the StackMultiByte class.

7.8 stacksinglebyte.h File Reference

Declares the StackSingleByte class.

```
#include "stackbase.h"
```

Classes

- class StackSingleByte

  Stack implementation using the PHY single-octet mode.

Typedefs

- typedef StackSingleByte Stack

  Alias for selected stack type.

7.8.1 Detailed Description

Declares the StackSingleByte class.

7.9 stacktransparent.h File Reference

Declares the StackTransparent class.

```
#include "stackbase.h"
```

Classes

- class StackTransparent

  Stack implementation using the PHY transparent mode.
- class TransparentModeSsHandler

  Implementation of class for handling SPI SS/ line with TX save/restore.

Typedefs

- typedef StackTransparent Stack

  Alias for selected stack type.

Functions

- **ISR** (USART_TX_vect)
- **ISR** (USART_RX_vect)

7.9.1 Detailed Description

Declares the StackTransparent class.