TN1233
Technical note

SENT demo driver for the SPC57xx/SPC58xx microcontroller families

Introduction

The Single Edge Nibble Transmission protocol is targeted for use in those applications where high-resolution data needs to be transmitted from a sensor to an Engine Control Unit (ECU). It is intended as a replacement for the lower resolution methods as conventional sensors providing analog output voltage and PWM and as a simpler low cost alternative to CAN, LIN or PSI5. Applications for throttle position sensing, mass airflow sensing, pressure sensing, temperature sensing, humidity sensing, pedal sensing and so on can be used as examples of automotive applications for SENT-compatible sensor devices.

Contents

List of tables

1 SENT encoding scheme

The SENT encoding scheme is a unidirectional communications scheme from the sensor/transmitting device to the controller/receiving device which does not include a coordination signal from the controller/receiving device. It occurs independently of any action of the receiver module and does not require any synchronization signal from the receiver module. The signal transmitted by the sensor consists of a series of pulses with data encoded as falling to falling edge periods. Data are transmitted in units of 4 bits (1 nibble) for which the interval between two falling edges (single edge) of the modulated signal with a constant amplitude voltage is evaluated and representing values from 0 to 15. The message time of a SENT message depends on the configured tick time, the transmitted data value and the presence of a pause pulse.

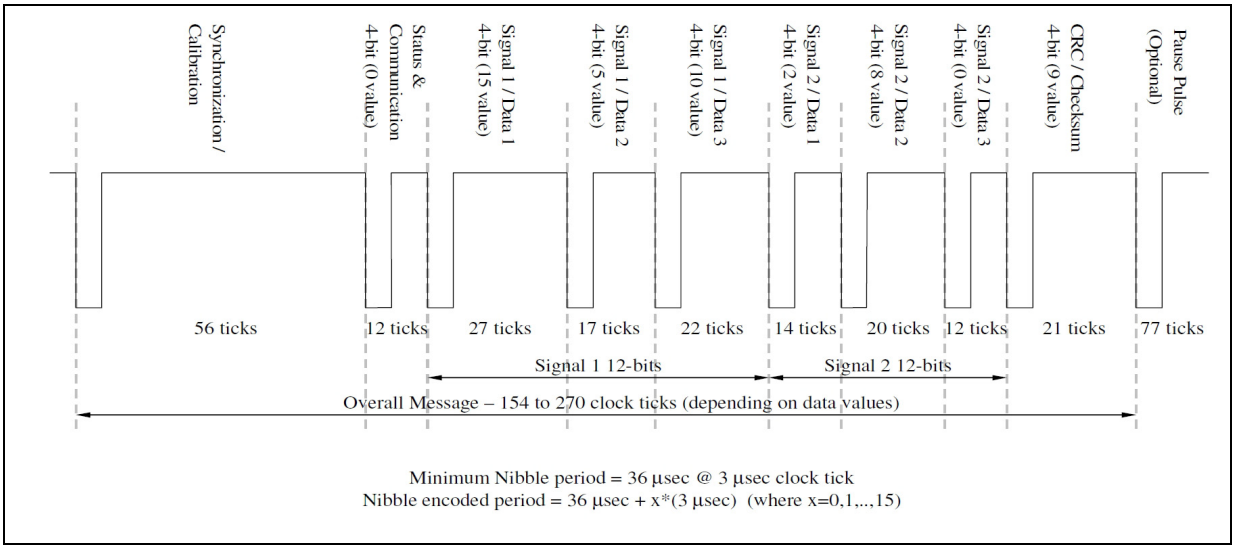
A consecutive SENT transmission starts immediately after the previous transmission ends (the trailing falling edge of the SENT transmission CRC nibble is also the leading falling edge of the consecutive SENT transmission synchronization/calibration pulse).

A transmitter specific nominal clock period used to measure the pulse period (tick time) can be in the range of 3–90 ms, according to the SAE J2716 specification. The maximum allowed clock variation is ± 20% from the nominal tick time, which allows the use of low-cost RC oscillators in the sensor device.

The transmission sequence consists of the following pulses (all times nominal):

1. Synchronization/calibration pulse (56 clock ticks)
2. One 4 bit Status and Serial Communication nibble pulse (12 to 27 clock ticks)
3. A sequence of one up to six data nibble pulses (12 to 27 clock ticks each) representing the values of the signal(s) to be communicated. The number of nibbles will be fixed for each application of the encoding scheme (i.e. throttle position sensors, mass air flow, etc.) but can vary between applications

Figure 1. Example encoding scheme for two 12 bit signals



4. One 4-bit checksum nibble pulse (12 to 27 clock ticks)
5. One optional pause pulse

1.1 Synchronization/calibration pulse

The transmitter clock is not synchronized to the receiver clock. The standard allows deviation from the nominal clock frequency of ± 20%. The data transmission format specifies a 56 tick synchronization/calibration pulse able to provide information on the actual transmitter (sensor) clock ticks period. The time between calibration and synchronization pulse, that precedes the data nibble pulses. Careful measurement of this calibration pulse allows the receiver to reliably normalize and interpret nibble periods as data values. The pulse starts with the falling edge and remains low for more than 4 clock ticks. The remaining clock ticks are driven high.

1.2 Status and communication nibble pulse

The status nibble contains 4-bit. The least significant bit 0 and 1are reserved to enable the sensor to transmit miscellaneous information such as part numbers or error code information. Bits 2 and 3 define a transmitter optional serial message channel which can be implemented either as a short or an enhanced serial message format. The complete 16-bit short serial message is then transmitted in 16 consecutive SENT transmissions whereas the complete 36-bit enhanced serial message is then transmitted in 18 consecutive SENT transmissions. The width of the status nibble pulse is dependent on the nibble value. The status nibble pulse and data nibble pulse formats are identical.

1.3 Data nibble pulse

A single data nibble pulse carries 4-bit sensor data. A maximum of six data nibbles can be transmitted in one SENT transmission. The total number of data nibbles depends on the size of the data provided by the sensor, and this will be fixed for each application of the encoding scheme. The width of the data nibble pulse is dependent on the nibble value.

1.4 Checksum nibble pulse

The checksum nibble contains a 4-bit CRC. The checksum is calculated using the x4+x3+x2+1 polynomial with the seed value of 5 (0b0101), and is calculated over all data nibbles. The status and communication nibble is not included in CRC calculation.

The CRC allows detection of the following errors:

1. All single bit errors.
2. All odd number of errors.
3. All single burst errors of length δ 4.
4. 87.5% of single burst errors of length = 5.
5. 93.75% of single burst errors of length > 5.

1.5 Pause pulse

The time for an individual message is a function of the data in the message and the transmitter clock rate only. If the nibble values are small, the message period will be short. Conversely, when the values are large, the message period will be long. This is a disadvantage for applications that desire constantly spaced data. To create a SENT

transmission with a constant number of clock ticks, the SENT protocol includes an optional pause pulse which acts as filler between the checksum nibble and the next calibration pulse. If implemented, the pause pulse has the following properties:

- Minimum Length 12 ticks (equivalent to a nibble with 0 value)
- Maximum Length 768 ticks (3 * 256)

2 SENT software demo driver

The demo driver is provided as example code only, and in the form of source code optimized for the Green Hills compiler. The demo driver supports up to two independent SENT instances and received data can be managed using eDMA or Interrupts Service Routines.

2.1 Utilized peripherals

The demo driver utilizes the following peripherals:

- System Integration Unit Lite (SIUL) — 1 pin for a single SENT channel
- Enhanced Direct Memory Address engine (eDMA) — a single channel for a single SENT channel
- Interrupt Controller INTC

2.2 Demo driver configuration

There are six pre-processor macros (accessible in the sent.h header file) that need to be properly defined before the final application is built. *Table 1* lists a description of all macros.

Table 1. Pre-compile time parameters

Macro	Description
TICK_LENGHT	Length of the receiver clock tick period in microseconds (SENT messages bit length)
SENT_CLOCK	High frequency receiver clock (SENT instance clock)
SENT_FAST_MESSAGES SENT_SLOW_MESSAGES	Define what kind of messages will be received by SENT cell; fast messages, slow messages, or both
SENT_DMA SENT_INTERRUPT	Define method for storing received messages: – Direct Memory Access (DMA) – Interrupt Service Routine (ISR)
SENT_INST_0 SENT_INST_1	Define which SENT instance will be configured to receive message; – SENT_0 – SENT_1 – SENT_0 and SENT_1

2.3 SENT channel configuration structure

The demo driver can configure all the channels for each SENT instance. Each SENT instance has its own static configuration structure in sent.c file (SENT_0_Static_Config and SENT_1_Static_Config) which needs to be initialized before the demo driver can be initialized, using the appropriate macros defined into sent.h header file. *Table 2* lists all members of the structure fields to be properly initialized before calling the configuration function.

Table 2. Parameters of the SENT demo driver configuration structure

Structure member	Description	Range
COMP_EN	Compensation Enable. To enable compensation logic to adjust the receiver clock against variation in Tx clock on selected channel	ENABLE, DISABLE
BUS_IDLE_CNT	Bus Idle Count: This value defines the maximum allowable idle period on the sensor interface of selected channel.	BUS_IDLE_CNT_DISABLED BUS_IDLE_CNT_1 BUS_IDLE_CNT_2 BUS_IDLE_CNT_4 BUS_IDLE_CNT_8
IE_CAL_RESYNC	Successive Calibration Check Resynchronized Interrupt Enable	ENABLE, DISABLE
IE_CAL_20_25	Calibration Variation 20 - 25% Interrupt Enable.	ENABLE, DISABLE
IE_SMSG_OFLW	Slow Serial Message Overflow Interrupt Enable	ENABLE, DISABLE
IE_FMSG_OFLW	Fast Message Overflow Interrupt Enable	ENABLE, DISABLE
FCRC_CHK_OFF	Fast Message CRC Check Off: This bit can be used to switch off CRC check in Fast Message	ENABLE, DISABLE
IE_PP_DIAG_ERR	Ratio of calibration pulse length to message length varies by more than ±1.5625% between two frames. Interrupt enable bit	ENABLE, DISABLE
IE_CAL_LEN_ERR	Calibration pulse is wider than 56 ticks ±25%. Interrupt enable bit	ENABLE, DISABLE
IE_CAL_DIAG_ERR	Successive Calibration pulses differ by more than ±1.56%. Interrupt enable bit	ENABLE, DISABLE
IE_NIB_VAL_ERR	Any nibble data value <0 or >15. Interrupt enable	ENABLE, DISABLE
IE_SMSG_CRC_ERR	Checksum error in Slow Serial Message. Interrupt enable	ENABLE, DISABLE
IE_FMSG_CRC_ERR	Checksum error in Fast Message. Interrupt enable	ENABLE, DISABLE
IE_NUM_EDGES_ERR	Not the expected number of negative edges between calibration pulse. Interrupt enable	ENABLE, DISABLE
DCHNG_INT	Enable for Interrupt on Reception of Fast Message with Changed Data.	ENABLE, DISABLE

Table 2. Parameters of the SENT demo driver configuration structure (continued)

Structure member	Description	Range
CAL_RNG	Valid Calibration Pulse Range Selection: This bit is used to control whether 20% variation or 25% variation in Calibration Pulse will be tolerated	TWENTY_VARIATION TWENTY-FIVE_VARIATION
PP_CHKSEL	Pause Pulse Diagnostic Check Selection	CALIB_AND_PAUSE_PULSES PAUSE_PULSE
FCRC_TYPE	Fast Message CRC Type	XOR_BASED, LEGACY_LUT
FCRC_SC_EN	Fast Message CRC Status and Communication Nibble Enable. This bit enables the inclusion of Status and Communication Nibble when CRC is calculated for Fast Messages.	ENABLE, DISABLE
SCRC_TYPE	Slow Serial Message CRC Type	XOR_BASED, LEGACY_LUT
PAUSE_EN	Pause Pulse Enable. Enables the receiver to detect a pause pulse.	ENABLE, DISABLE
SUCC_CAL_CHK	Successive Calibration Pulse Check Method	OPTION_2, OPTION_1
FIL_CNT	Input Filter Sample Count	NO_FILTERING FILTERING_ENABLE
TSPRSC	Time Stamp Prescaler Value	PRSC_BYPASS, ONE, TWO, THREE, FOUR, FIVE. (Admitted values are in the range [1, 255]. Only values listed above have been defined. To use different prescaler, please define another macro in sent.h file)
FMDUIE	Fast Message DMA Underflow Interrupt Enable	ENABLE, DISABLE
SMDUIE	Slow Serial Message DMA Underflow Interrupt Enable	ENABLE, DISABLE
FAST_CLR	Fast Clearing Enable bit. Enables clearing of ready status (for both fast and slow) bit automatically when corresponding message is read.	ENABLE, DISABLE
DBG_FRZ	Debug Freeze. This bit will enable the debug mode support. SENT Module will freeze in debug mode if this bit is set.	NO_EFFECT, FREEZE

2.4 Files containing the demo driver implementation

To guarantee the correct behavior of the demo driver into the microcontroller projects, where SENT receiver is available, it is recommended to include the following files:

- sent.c
- sent.h
- dma.c
- dma.h
- config.c
- config.h
- interrupt_routines.c

2.4.1 How to import demo driver into different microcontroller platform.

This chapter provides information about procedure that user should follow to import demo driver in a new project:

- Include demo driver files into the platform project.
- Update the microcontroller specific header file inclusion in files ‘sent.c’, ‘dma.c’, config.c.
- Enable interrupts
- Provide clock to the SENT peripherals.
- Update the input SENT pads in Pads_Init() function and the interrupts source number into Interrupts_Setup() function.
- For reception via DMA, update the DMA source and DMA TCD in DMA_Setup() function.
- For reception via interrupt, copy the interrupt routines defined into interrupt_routine.c file into the interrupt routines file of the used platform and update the interrupt table properly.

2.5 API

The demo driver API consists of the following function:

1. SENT_Config()

2.5.1 SENT_Config

Syntax: SENT_Config(uint8_t, uint8_t, int);

Reentrancy: Non-reentrant.

Parameters:

- SENT instance number
- SENT channel number
- SENT prescaler

Return: VOID

Description: The function initializes the SENT channel receiver in order to receive messages from connected sensors. At the end of the function execution the SENT channel

receiver is enabled and ready to receive. Received data, timestamp and CRC values are stored into data structures called “Readings_0” for Fast Messages and “Readings_1” for Slow Messages. Received data can be stored by ISR or automatically by DMA depending on data management selected.

2.5.2 Default demo driver configuration

The default SENT demo driver configuration includes the following enabled features:

Pre-compiled time parameters

- #define TICK_LENGHT 3
- #define SENT_FAST_MESSAGES1
- #define SENT_SLOW_MESSAGES1
- #define SENT_DMA
- #define SENT_INST_00

SENT channel configuration structure:

- Clock compensation
- Interrupt on Reception of Fast Message with Changed Data
- 20% variation is acceptable
- Pause Pulse and Successive Calibration pulse Diagnostic Check
- Fast and Slow Message CRC Type = XOR based implementation
- Pause Pulse enabled
- Successive Calibration Pulse Check Method= option 1
- Input Filter Sample Count
- Timestamp prescaler = 2
- Fast Clearing
- Debug Freeze

2.5.3 Functional description

The demo driver initialization is done by the SENT_Config() API function. Before running it, initialize properly the configuration structure. The SENT demo driver is based on the following functions:

Main function

The Main function encloses all functions used to initialize the SENT instance and to start the reception from the SENT receiver. Following the list of functions performed step by step inside the main function:

```
// Rx pads initialization
Pads_Init ();
// Interrupts configuration for Rx data management via interrupts
#ifdef SENT_INTERRUPT
Interrups_Setup();
#endif
// DMA configuration for DMA Rx data management via DMA
#ifdef SENT_DMA
```

```
DMA_Setup ();
#endif
// SENT instance initialization
#ifdef SENT_INST_0
SENT_Config(SENT_INST_0, SENT_channel_0 , prescaler);
#endif
#ifdef SENT_INST_1
SENT_Config(SENT_INST_1, SENT_channel_0 , prescaler);
#endif
while(1){}
```

Other demo driver's functions

Other functions used into the demo driver are listed in *Table 3* along with features of each one and file name where the function is embedded.

Table 3. Other functions

Function Name	Functionality	File
DMA_Config	Configure DMA to transfer received data into defined structures (Readings_0 and Readings_1)	dma.c
Interrupts_Setup	Initialize interrupt controller to generate combined interrupts on messages reception by SENT0 and SENT1 peripherals.	config.c
Pads_Init	Initialize pads for SENT_0 and SENT_1	config.c
DMA_Setup	Call the DMA_Config function using appropriate parameters in order to configure the DMA for transfers of Fast messages and Slow messages coming from SENT_0 and SENT_1	config.c
SENT_0_Channel0_FastMsg_ISR	Interrupt Service Routine to serve combined Interrupts generated by reception of Fast messages on SENT0 instance. (interrupt number 558)	interrupt_routines.c
SENT_0_Channel0_SlowMsg_ISR	Interrupt Service Routine to serve combined Interrupts generated by reception of Slow messages on SENT0 instance (interrupt number 559).	interrupt_routines.c

Table 3. Other functions (continued)

Function Name	Functionality	File
SENT_1_Channel0_FastMsg_ISR	Interrupt Service Routine to serve combined Interrupts generated by reception of Fast messages on SENT1 instance (interrupt number 561).	interrupt_routines.c
SENT_1_Channel0_SlowMsg_ISR	Interrupt Service Routine to serve combined Interrupts generated by reception of Slow messages on SENT1 instance (interrupt number 562).	interrupt_routines.c

3 Conclusion

This technical note describes the SENT protocol, providing also into the text a detailed list of utilized peripherals, configuration description, the API calling sequence and the functional description of the SENT demo driver for the SPC57x and SPC58x families of microcontroller.

The software demo driver provides full communication with SENT sensors via DMA or via Interrupt.

4 Revision history

Table 4. Document revision history

Date	Revision	Changes
23-Feb-2016	1	Initial release.
08-Jun-2016	2	Updated document title.