

# AP32123

## TC1796

SENT Receiver (CPU & PCP  
implementation)

Microcontrollers



Never stop thinking

**Edition 2008-08-07**

**Published by  
Infineon Technologies AG  
81726 München, Germany**

**© Infineon Technologies AG 2008.  
All Rights Reserved.**

#### **LEGAL DISCLAIMER**

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

#### **Information**

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office ([www.infineon.com](http://www.infineon.com)).

#### **Warnings**

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

---

**AP32123**

**Revision History:** 2008-08 V1.2

Previous Version: none

Author: Alann Denais

Page	Subjects (major changes since last revision)

**We Listen to Your Comments**

Any information within this document that you feel is wrong, unclear or missing at all?

Your feedback will help us to continuously improve the quality of this document.

Please send your proposal (including a reference to this document) to:

[mcdocu.comments@infineon.com](mailto:mcdocu.comments@infineon.com)



Table of Contents		Page
<b>1</b>	<b>Scope.....</b>	<b>5</b>
<b>2</b>	<b>Needed material .....</b>	<b>6</b>
<b>3</b>	<b>Installation steps .....</b>	<b>7</b>
<b>4</b>	<b>Micro controller resources used .....</b>	<b>8</b>
<b>5</b>	<b>Implementation on the TC1796 (SENT channel 1) .....</b>	<b>9</b>
5.1	Capturing the SENT Rx signal .....	9
5.2	Buffering the nibbles .....	10
5.3	Decoding the frame .....	10
5.4	Synchronization.....	12
5.5	Notification to the application layer .....	13
5.6	Configuration summary .....	13
5.7	Configuration differences for the PCP version.....	14
<b>6</b>	<b>2<sup>nd</sup> SENT channel .....</b>	<b>15</b>
6.1	Configuration summary .....	16
6.2	Configuration differences for the PCP version.....	17
<b>7</b>	<b>CPU / PCP resources .....</b>	<b>18</b>
7.1	Load for the CPU & PCP versions .....	18
7.2	Memory footprint .....	18
<b>8</b>	<b>Source code example .....</b>	<b>19</b>
8.1	Documentation .....	19
8.2	Building the CPU version .....	19
8.3	Building the PCP version .....	19
8.3.1	Generating PCP object files .....	19
8.3.2	Generating the target .elf file.....	19
8.3.3	Patch for the Tasking PCP compiler version 2.5r2 .....	20
<b>9</b>	<b>Possible extension.....</b>	<b>21</b>

## 1 Scope

Dear Reader,

Thanks for using this SENT<sup>1</sup> receiver! This tool will help you to receive and decode data coming from a SENT sensor<sup>2</sup>, on your target microcontroller. This document will give you step by step instruction in order to install and operate the receiver, but also describe in detail the configuration needed to realize such a function with the TC1796 microcontroller.

The aim of this receiver is to decode a SENT compatible signal. For this a TC1796 microcontroller TriBoard is used.

The following features are supported by the receiver:

- Reception and decoding of SENT compliant frames. The decoding is done on the PCP or on the CPU, both version are available as source code.
- Up to 8 data nibbles per frame.
- Serial data decoding.
- Programmable clock rate.
- Error detection: signal loss, synchronization loss, clock drift, invalid CRC, invalid data nibble, serial data error.
- Full source code available<sup>3</sup>.

Have fun with Infineon's SENT receiver!

---

<sup>1</sup> Single Edge Nibble Transmission (SENT) refers to the SAE standard J2716. For more information, please visit [www.sae.org](http://www.sae.org).

<sup>2</sup> The receiver decodes logical SENT frames. Electrical characteristics as defined by the standard are not covered.

<sup>3</sup> The code delivered with this application note is aimed at development and demonstration purpose only. Neither is its quality nor its robustness guaranteed.

## **2 Needed material**

The following HW and SW material is mandatory to operate the tool:

- A PC with local administration rights.
- A functional TC1796 TriBoard (incl. cables, power supplies and adapter board).
- A debugger supporting TC1796.
- An oscilloscope.
- The Receiver source code and executable (included in this package).
- Optional: Tasking VX-toolset for TriCore, v2.5.
- Optional: Tasking VX-toolset for TriCore & PCP, v2.5
- A SENT compatible sensor or the SENT simulator (see appnote AP32118)

It is strongly recommended to have the Tasking toolchain installed and running on the PC where the receiver is running, so that the default parameters of the receiver can be changed (recompilation needed). For more information about the toolchain, please visit [www.tasking.com](http://www.tasking.com).

The Receiver software delivered with this application note is made of:

- Source code
- Object file for the PCP version
- Executable files (.elf and .hex) for each PCP and CPU version
- Two tasking project files for the toolchain v2.5, one for each PCP and CPU version.

### 3 Installation steps

It is assumed that the user knows how to operate the different tools mentioned above (Compiler, Debugger, TriBoard, etc.). The following steps may be followed to install operate the receiver:

- Install all the needed SW tools (Tasking toolchain, debugger).
- Connect a TriBoard to the debugger.
- Connect the SENT Rx signal to the pin 2.8, Connect the SENT ground signal to the TriBoard digital ground
- Connect an scope to the pin 2.9 (DMA trigger)
- Connect a scope to the pin 2.10 (Decoding ISR execution time)
- Build the project.
- Start a debugger session.
- Run the program, and look for the global variable `System`. The member structure `System.SentStatus` holds the received information. Further more the DMA trigger signal is made available on the pin 2.9 on the scope.
- This example implements a second SENT channel<sup>4</sup>:
  - SENT Rx signal: pin 3.0
  - DMA trigger pin 3.1

---

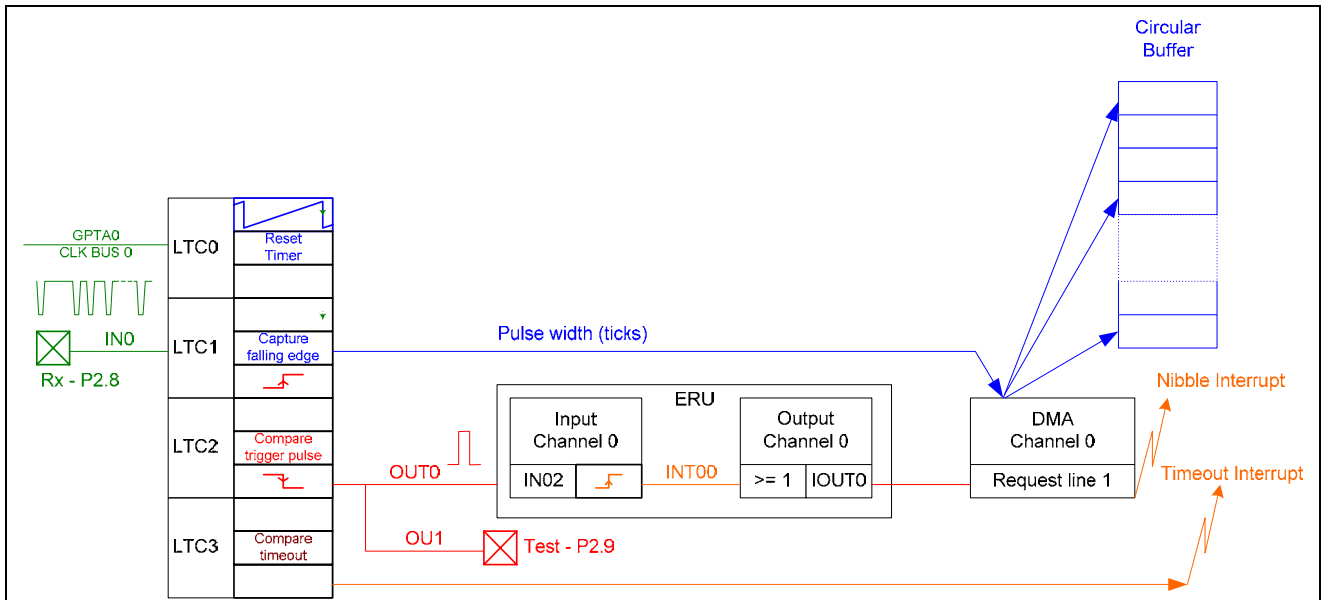
<sup>4</sup> The 2<sup>nd</sup> SENT channel is disabled by default. It can be enabled with the switch `SENT_ENABLE_CHANNEL_1`

#### **4        Micro controller resources used**

- GPTA: Event capture, Signal generation
- ERU: Trigger routing between GPTA LTC cell and DMA
- DMA: Data buffering
- PCP/CPU: Data processing



## 5 Implementation on the TC1796 (SENT channel 1)



**Figure 1** SENT channel resources usage

The Figure 1 presents the implementation of the SENT driver on the TC1796.

The decoding of the SENT frames can be seen as 4 steps:

- Capturing and buffering the nibbles (GPTA & DMA)
- Checking for signal timeout (GPTA)
- Decoding the frame (PCP or CPU according to the implemented version)
- Synchronization on the frame start/end (PCP or CPU according to the implemented version)

### 5.1 Capturing the SENT Rx signal

The Rx signal present on the P2.8 pin is connected to the GPTA input signal IN0. The input multiplexer is setup to connect the signal IN0 to the LTC0.

The SENT nibble start is defined by the falling edge of the SENT signal and the nibble value by the time between two consecutive falling edges. In order to acquire each nibble, the GPTA is setup to continuously capture the time between two falling edges.

The acquisition is implemented using two LTCs, the 1<sup>st</sup> one as a “reset timer” (LTC0) that counts up and is reset when an event on the next cell occurs. The 2<sup>nd</sup> one is configured as a “capture” (LTC1) that captures the timer value on each falling edge and reset the timer LTC0. The captured value represents the nibble value in GPTA ticks.

In order to process the incoming nibbles a trigger event is generated using the following configuration:

- LTC1 set the trigger signal on falling edge of Rx (cell action: Set)
- LTC2 reset the trigger signal after a time  $T_{inhibit}$  (cell action: copy or reset)

## Implementation on the TC1796 (SENT channel 1)

- The trigger signal is routed to the GPTA OUT0<sup>5</sup>

The inhibit time  $T_{\text{inhibit}}$  avoids unnecessary triggering due to high frequency noise that could be seen on the Rx pin. This also acts as a filter<sup>6</sup> and strengthens the signal reception quality.

In order to check for the presence of the SENT signal, a 4<sup>th</sup> LTC cell is used (LTC3). This cell compares the timer value with the maximal nibble time<sup>7</sup> and generates a timeout event to the processor in case a nibble is missing. See the `Sent_TimeoutLsr_0()` function.

### 5.2 Buffering the nibbles

In order to offload the processor that decodes the SENT frame, the nibbles are buffered by the DMA engine until a complete frame has been received<sup>8</sup>. Then the DMA raise an interrupt to the processor which decode the full frame. See the `Sent_NibbleLsr_0()` function.

The trigger defined previously is used to trigger a DMA move on each captured nibble. For that purpose the ERU is configured to route the signal GPTA OUT0 to the request line 1 of the DMA channel 0. The ERU is configured as follow:

- OUT0 is connected to the ERU input channel 0, input IN02
- The ERU input channel 0 generates a pulse on INT00, on the rising edge of OUT0 signal
- The ERU output channel 0 forward the INT00 signal to IOUT0
- The IOUT0 is connected to the request line 1 of the DMA channel 0

The DMA channel 0 is configured so that on each trigger event, the captured nibble value is read and copied to a circular buffer. The DMA destination pointer is incremented on each DMA move. Once the buffer end is reached, the destination pointer is set to the beginning of the buffer, and so on.

This configuration implies some restriction on the buffer base address and buffer size:

- The buffer size should buffer at least 2 full frames, so that 1 frame can be processed while the next one is being buffered.
- The move size must be set according to the memory access modes, and be at least 16 bit.
- The DMA circular buffer wrap around mechanism only allow a buffer base address aligned on multiple of the buffer size itself.

In the current implementation, the frame size is 9 nibbles (24 data bits). Therefore the buffer should be at least 18 nibbles long. The next power of two, in order to be able to use the circular buffer is then 32 nibbles.

For the CPU version, the buffer is located in the LDRAM which can be accessed 16 bit wise; therefore the buffer size is set to 64 bytes. The base address of the buffer should have the 6 least significant bits set to 0.

For the PCP version, the buffer is located in the PCP RAM which can only be accessed 32 bit wise; therefore the buffer size is set to 128 bytes. The base address of the buffer should have the 7 least significant bits set to 0.

### 5.3 Decoding the frame

As described previously, the SENT nibbles are buffered by the DMA engine and an interrupt is generated to the processor once a full frame is supposed to be buffered.

The interrupt will then check the incoming nibbles for:

<sup>5</sup> The trigger signal is also routed to the GPTA OUT1 for debugging purpose (oscilloscope probe)

<sup>6</sup> Note that glitches that appear during the OUT0 high state will still be captured by the LTC, but ignored by the nibble decoding process because no DMA transfer is triggered.

<sup>7</sup> The maximal nibble time limit is defined by the calibration pulse:  $56 * (T_{\text{TickNominal}} * 1.25)$ .

<sup>8</sup> A complete frame is supposed to be received when all frame's nibbles have been received, for example `FrameSize = 9` nibbles for a frame with 24 bits data (1x calibration pulse + 1x status and communication + 6x data nibbles + 1x CRC).

Implementation on the TC1796 (SENT channel 1)

- A valid calibration pulse
- Valid nibbles
- A Valid Frame CRC, and valid serial CRC if a serial frame has been received

Then save the received data, and optionally generate an interrupt to the CPU. The Figure 2 presents in details the decoding of the SENT frame.

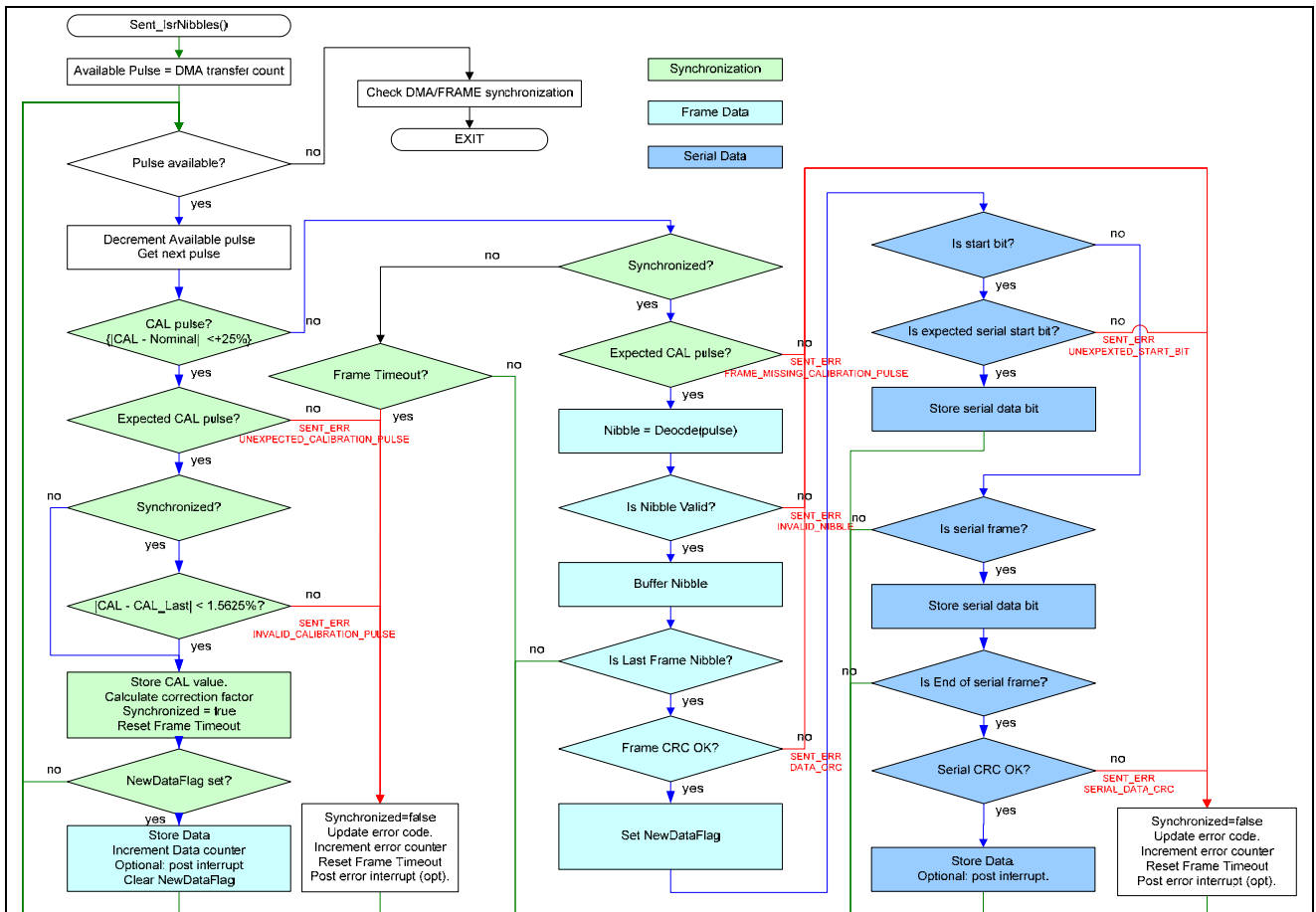


Figure 2 IsrNibbles()

The Figure 3 presents the action on nibble timeout.

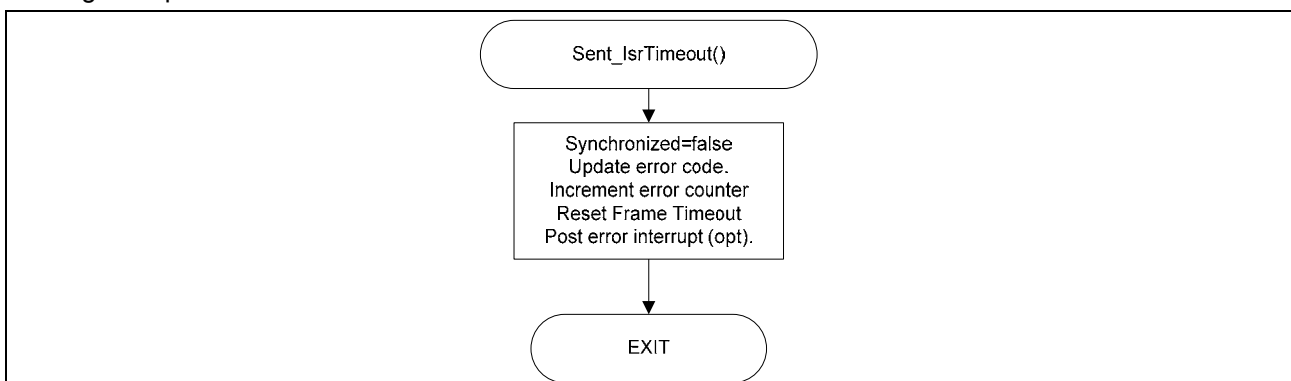
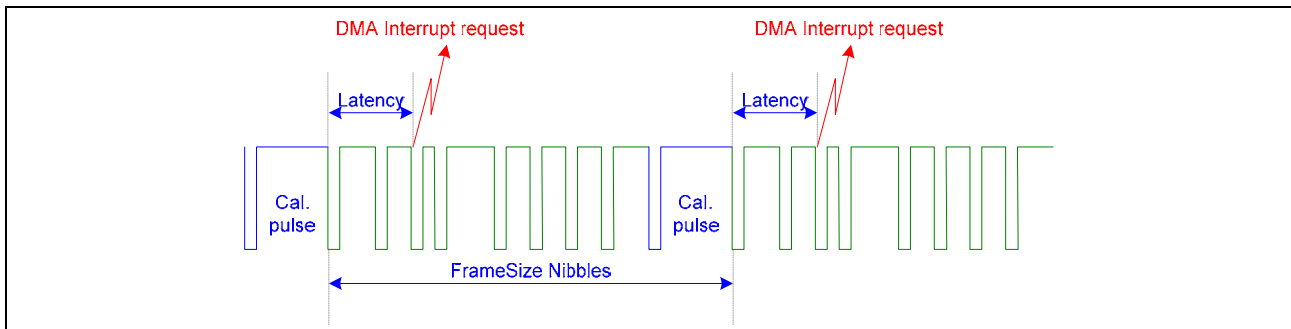


Figure 3 IsrTimeout()

## 5.4 Synchronization

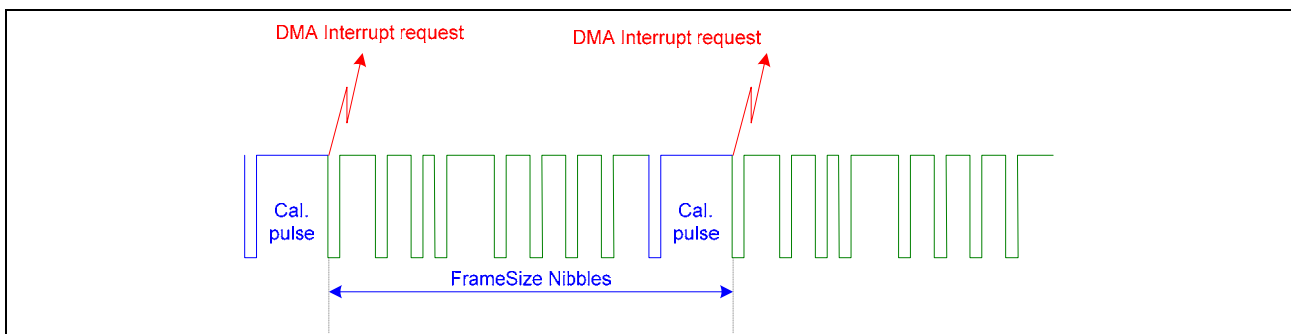
As soon as enabled, the driver starts buffering the incoming nibbles. Each time a full frame nibble count has been received (9 nibbles for the current implementation); an interrupt is raised to the processor. In most of the cases the interrupt is not synchronized on the end of the SENT frame, and the latency between the last frame nibble and the notification to the processor may last up to (FrameSize-1) nibbles in the worst case. See figure below:



**Figure 4 Not synchronized frame state**

In order to reduce the latency between the time a full frame has been received and the start of the frame processing, the DMA interrupt is synchronized on the SENT frame last nibble. In this case the latency is reduced to the time needed by the processor to handle the received frame.

The synchronization is checked each time a complete frame has been received. In case the DMA is not synchronized, the next DMA interrupt is delayed so that it is raised exactly at the end of the frame as shown in Figure 5.



**Figure 5 Synchronized frame state**

In order for the synchronization to be valid, it must be ensured that the reconfiguration of the DMA transfer count value is done between the falling edge of the SENT signal that triggered the interrupt and the next SENT signal falling edge. In the case this condition can not be fulfilled, the synchronization might not be done properly and the SENT frame decoding will not be valid (unknown behavior). The reason for not being able to fulfill this condition is a too high processor load. In such case the synchronization should be disabled (undefined the switch `SENT_ENABLE_FRAME_SYNC`).

## 5.5 Notification to the application layer

When a SENT frame has been received, a sticky status Flag is set which can be reset with the function `Sent_ResetStatus()`. When an error has been detected, a sticky status Flag is set which can be reset with the function `Sent_ResetErrors()`.

The application can use the polling method or react on an interrupt being set by the driver. This interrupt is generated on the following events:

- A valid frame has been received
- A valid serial frame has been received
- An error has been detected

To enable the interrupt generation, the following switches must be defined:

- `SENT_ENABLE_DATA_INTERRUPT`
- `SENT_ENABLE_SDATA_INTERRUPT`
- `SENT_ENABLE_ERROR_INTERRUPT`

## 5.6 Configuration summary

P2.8	<ul style="list-style-type: none"> <li>• Direction : Input</li> <li>• Pull-up / pull-down : No</li> </ul>
LTC0 (Timer)	<ul style="list-style-type: none"> <li>• Mode : Timer with reset from next cell</li> <li>• Input : GPTA clock bus 0</li> <li>• Input mode : Level sensitive (high level)</li> <li>• Initial value : 0x0000</li> <li>• Interrupt : No</li> <li>• Action on event : Hold</li> <li>• Output : none</li> </ul>
LTC1 (Nibble capture)	<ul style="list-style-type: none"> <li>• Mode : Capture last timer</li> <li>• Input : Pin P2.8</li> <li>• Input mode : Edge sensitive (falling edge)</li> <li>• Interrupt : No</li> <li>• Action on event : Set</li> <li>• Output : none</li> </ul>
LTC2 (HF filter)	<ul style="list-style-type: none"> <li>• Mode : Compare with last timer</li> <li>• Compare value : <math>5 * T_{Tick} * 0.75</math></li> <li>• Interrupt : No</li> <li>• Action on event : Copy or reset</li> <li>• Output : OUT0</li> </ul>
LTC3 (Timeout)	<ul style="list-style-type: none"> <li>• Mode : Compare with last timer</li> <li>• Compare value : <math>56 * T_{Tick} * 1.25</math></li> <li>• Interrupt : Yes (timeout)</li> <li>• Action on event : Hold</li> <li>• Output : none</li> </ul>
ERU input channel 0	<ul style="list-style-type: none"> <li>• Input : IN02 (GPTA OUT0)</li> <li>• Edge detection : Rising edge</li> <li>• Auto reset flag : Enabled</li> <li>• Output : INT00</li> </ul>
ERU output channel 0	<ul style="list-style-type: none"> <li>• ERU INTF0 : Disabled</li> <li>• ERU INTF1 : Disabled</li> <li>• ERU INTF2 : Disabled</li> <li>• ERU INTF3 : Disabled</li> <li>• Pattern trigger : Disabled</li> <li>• Interrupt gating : Always</li> <li>• Output : IOUT0</li> </ul>
DMA channel 0, Engine 0	<ul style="list-style-type: none"> <li>• DMA move size : 16 bit</li> <li>• DMA transfer : 1 DMA move</li> <li>• DMA transaction : FrameSize or TransactionCount</li> <li>• Transfer mode : Single transfer, continuous</li> </ul>

**Implementation on the TC1796 (SENT channel 1)**

	<ul style="list-style-type: none"> <li>• Source : No increment</li> <li>• Destination : Positive increment by 16 bit, wrap around at 64 bytes</li> <li>• Source address : LTC1 value</li> <li>• Destination address : Channel buffer address</li> <li>• Trigger : Request line 1 (IOUT0)</li> <li>• Channel interrupt : Interrupt node pointer 1, Interrupt on Count=0. Interrupt generated after each transaction.</li> <li>• Region enabled : GPTA0, GPTA1,LTCA2, DMI image</li> <li>• Transaction lost error : Disabled</li> <li>• Source error : Disabled</li> <li>• Destination error : Disabled</li> <li>• DMA priority : Low</li> <li>• DMA channel priority : Low</li> <li>• Shadow : No</li> </ul>
Timeout interrupt	<ul style="list-style-type: none"> <li>• Source : LTC3</li> <li>• Priority : 8</li> </ul>
Nibble interrupt	<ul style="list-style-type: none"> <li>• Source : DMA channel 0</li> <li>• Priority : 9</li> </ul>
SENT channel interrupt	<ul style="list-style-type: none"> <li>• Source : software, on SEND frame received &amp; error</li> <li>• Priority : 1</li> </ul>

**5.7 Configuration differences for the PCP version**

DMA channel 0, Engine 0	<ul style="list-style-type: none"> <li>• DMA move size : 32 bit</li> <li>• Destination : Positive increment by 32 bit, wrap around at 128 bytes</li> </ul>
-------------------------	--

## 6 2<sup>nd</sup> SENT channel

A second SENT channel has been implemented in a similar way than the 1<sup>st</sup> channel. The specific channel configuration is described in the figure below. Note that the channel is by default disabled and can be enabled with the switch `SENT_ENABLE_CHANNEL1_1`.

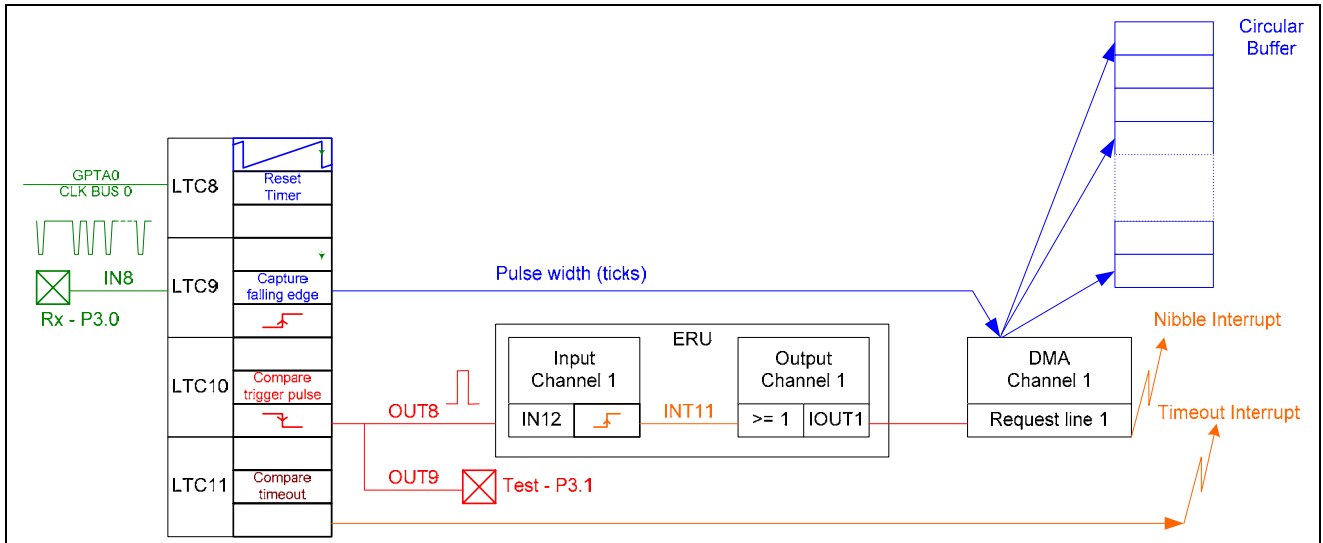


Figure 6 SENT channel 2 resources usage

## 6.1 Configuration summary

P3.0	<ul style="list-style-type: none"> <li>• Direction : Input</li> <li>• Pull-up / pull-down : No</li> </ul>
LTC8 (Timer)	<ul style="list-style-type: none"> <li>• Mode : Timer with reset from next cell</li> <li>• Input : GPTA clock bus 0</li> <li>• Input mode : Level sensitive (high level)</li> <li>• Initial value : 0x0000</li> <li>• Interrupt : No</li> <li>• Action on event : Hold</li> <li>• Output : none</li> </ul>
LTC9 (Nibble capture)	<ul style="list-style-type: none"> <li>• Mode : Capture last timer</li> <li>• Input : Pin P3.0</li> <li>• Input mode : Edge sensitive (falling edge)</li> <li>• Interrupt : No</li> <li>• Action on event : Set</li> <li>• Output : none</li> </ul>
LTC10 (HF filter)	<ul style="list-style-type: none"> <li>• Mode : Compare with last timer</li> <li>• Compare value : <math>5 * T_{\text{Tick}} * 0.75</math></li> <li>• Interrupt : No</li> <li>• Action on event : Copy or reset</li> <li>• Output : OUT8</li> </ul>
LTC11 (Timeout)	<ul style="list-style-type: none"> <li>• Mode : Compare with last timer</li> <li>• Compare value : <math>56 * T_{\text{Tick}} * 1.25</math></li> <li>• Interrupt : Yes (timeout)</li> <li>• Action on event : Hold</li> <li>• Output : none</li> </ul>
ERU input channel 0	<ul style="list-style-type: none"> <li>• Input : IN12 (GPTA OUT8)</li> <li>• Edge detection : Rising edge</li> <li>• Auto reset flag : Enabled</li> <li>• Output : INT11</li> </ul>
ERU output channel 0	<ul style="list-style-type: none"> <li>• ERU INTF0 : Disabled</li> <li>• ERU INTF1 : Disabled</li> <li>• ERU INTF2 : Disabled</li> <li>• ERU INTF3 : Disabled</li> <li>• Pattern trigger : Disabled</li> <li>• Interrupt gating : Always</li> <li>• Output : IOUT1</li> </ul>
DMA channel 0, Engine 0	<ul style="list-style-type: none"> <li>• DMA move size : 16 bit</li> <li>• DMA transfer : 1 DMA move</li> <li>• DMA transaction : FrameSize or TransactionCount</li> <li>• Transfer mode : Single transfer, continuous</li> <li>• Source : No increment</li> <li>• Destination : Positive increment by 16 bit, wrap around at 64 bytes</li> <li>• Source address : LTC9 value</li> <li>• Destination address : Channel buffer address</li> <li>• Trigger : Request line 1 (IOUT1)</li> <li>• Channel interrupt : Interrupt node pointer 1, Interrupt on Count=0. Interrupt generated after each transaction.</li> <li>• Region enabled : GPTA0, GPTA1,LTCA2, DMI image</li> <li>• Transaction lost error : Disabled</li> <li>• Source error : Disabled</li> <li>• Destination error : Disabled</li> <li>• DMA priority : Low</li> <li>• DMA channel priority : Low</li> <li>• Shadow : No</li> </ul>
Timeout interrupt	<ul style="list-style-type: none"> <li>• Source : LTC11</li> <li>• Priority : 10</li> </ul>
Nibble interrupt	<ul style="list-style-type: none"> <li>• Source : DMA channel 0</li> <li>• Priority : 11</li> </ul>
SENT channel interrupt	<ul style="list-style-type: none"> <li>• Source : software, on SEND frame received &amp; error</li> <li>• Priority : 1</li> </ul>



## 6.2 Configuration differences for the PCP version

DMA channel 0, Engine 0	<ul style="list-style-type: none"><li>• DMA move size : 32 bit</li><li>• Destination : Positive increment by 32 bit, wrap around at 128 bytes</li></ul>
----------------------------	---

## 7 CPU / PCP resources

### 7.1 Load for the CPU & PCP versions

Frame decoding <sup>9</sup>	Frame time (us)	CPU Version (CPU @150Mhz)	PCP version (PCP @75Mhz)
Time for full frame decoding		6.1us	44.4us
<b>Average case<sup>10</sup></b>	<b>642.0</b>	<b>1.0%</b>	<b>6.9%</b>
Worst case <sup>11</sup>	364.8	1.7%	12.2%
Best case <sup>12</sup>	993.6	0.6%	4.5%

### 7.2 Memory footprint

		Code (bytes)	Data (bytes)
CPU version	CPU	11140	7788
PCP version	CPU	14310	7192
	PCP	3452	1776

<sup>9</sup> Frame length of 9 nibbles for 24 data bits. Serial frame decoding disabled

<sup>10</sup> Having a nominal frame tick of 3us, average frame length 214 ticks

<sup>11</sup> Having a nominal frame tick of 3us - 20% = 2.4us, min frame length 152 ticks

<sup>12</sup> Having a nominal frame tick of 3us + 20% = 3.6us, max frame length 276 ticks

## 8 Source code example

### 8.1 Documentation

The source code documentation can be found in HTML format, see the file\Doc\html\index..

The documentation structure is as follow:

Tab	Description
Main page	General information
Module	Structured source code documentation.
Data structures	Information on data structures
Files	File description

Note that the default configuration can be modified in the file "configuration.h".

### 8.2 Building the CPU version

Load and compile the tasking project AP32123\_SENTR(TC1796\_RAM).pjt.

Note that a DAVE initialization version is also provided. See the project AP32123\_SENTR(TC1796\_RAM)\_DAvE.pjt.

### 8.3 Building the PCP version

The PCP version is built in two steps:

- Generate the object file for the using the PCP compiler
- Compile the whole project using the Tricore compiler

#### 8.3.1 Generating PCP object files

Run the batch file "BuildPcp.bat". This will launch mkpcp using the default make file "makefile".

The environment variable PATH should contain the path to mkpcp.exe. The default path is "C:\Program Files\TASKING\Tricore v2.5r1\cpcp\bin".

The following object files will be generated:

- Sent\_core.c → pcp\_sent\_core.o
- Sent\_pcp.c → pcp\_sent\_pcp.o
- Bsp.c → pcp\_bsp.o

#### 8.3.2 Generating the target .elf file

Load and compile the tasking project AP32123\_SENTR(TC1796\_PCP\_RAM).pjt.

### 8.3.3 Patch for the Tasking PCP compiler version 2.5r2

The Tasking Vx toolset for Tricore and PCP v2.5r2, disable the channel by clearing the flag CEN of register R7. This produces the error "Disable channel request" (DCR flag).

In order to avoid such errors, the function `_sti()` and `_ldi()` have been re-written, and overwrite the ones defined in the tasking library. Furthermore, all instruction that set the register R7 (`LDL.L R7, #const`) are patched so that the bit CEN is set.

The patch is enabled when the constant `PATCH_CEN` is defined to 1 in the file `CompilerFix.h`.

## **9 Possible extension**

It is possible to use the FPC to filter the SENT signal against glitches. This is currently not part of the implementation.

<http://www.infineon.com>