



S/PDIF library

Publication Date: 2023/10/18

Document Number: NoneA

IN THIS DOCUMENT

- ▶ Summary
 - ▶ External Signal Description
 - ▶ Usage
 - ▶ API
 - ▶ Known Issues
-

Version	5.0.1
Vendor	XMOS
Scope	General Use

1 Summary

A software defined S/PDIF library that allows transmission and reception of S/PDIF data via xCORE ports. S/PDIF is a digital data streaming interface. The components in the library are controlled via C using the XMOS multicore extensions (xC) and provides both a S/PDIF receiver and transmitter.

1.1 Features

- ▶ Supports stereo S/PDIF receive for sample rates up to 192KHz
- ▶ Supports stereo S/PDIF transmit for sample rates up to 192KHz

1.2 Related Application Notes

The following application notes use this library:

- ▶ AN00231 - SPDIF Receive to I2S output using Asynchronous Sample Rate Conversion

1.3 Documentation

You can find the documentation for this software in the /doc directory of the package.

1.4 Support

This package is supported by XMOS Ltd. Issues can be raised against the software at:

- ▶ <http://www.xmos.com/support>

1.5 Required Software (dependencies)

- ▶ None

1.6 Typical Resource Usage

This following table shows typical resource usage in some different configurations. Exact resource usage will depend on the particular use of the library by the application.

Configuration	Pins	Ports	Clocks	Ram	Logical cores
Transmit	1	1 (1-bit)	1	~3.3K	1
Receive	1	1 (1-bit)	1	~3.5K	1



2 External Signal Description

The library implements the S/PDIF (Sony/Philips Digital Interface Format) protocol for transporting uncompressed stereo PCM data of up to 24bits.

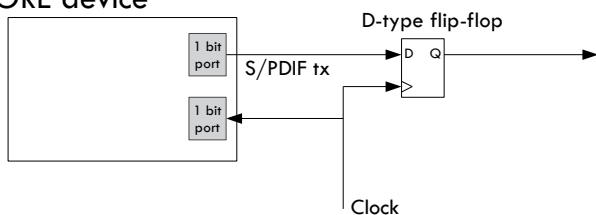
2.1 Connecting to the xCORE as Transmitter

The precise transmission frequencies supported depend on the availability of an external clock (e.g. a PLL or a crystal oscillator) that runs at a frequency of $channels * sampleRate * 64$ or a power-of-2 multiple. For example, for 2 channels at 192 KHz the external clock has to run at a frequency of 24.576 MHz. This same frequency also supports 2 channels at 48 KHz (which requires a minimum frequency of 6.144 MHz). If both 44,1 and 48 KHz frequencies are to be supported, both a 24.576 MHz and a 22.579 MHz master clock is required.

The connection of an S/PDIF transmit line to the xCORE is shown in Figure 1.

Figure 1: Connecting S/PDIF transmit

xCORE device



The output signal will contain jitter at the level of ± 1 core clock (< 2 ns for a 500 MHz xcore) this is typically inconsequential but if lower jitter levels are desired the signal can be re-clocked by the external master clock to reduce the jitter to that of the external master clock. A simple D-type flip flop can be used for this purpose.

The incoming clock signal is used to drive an internal clock and can be shared with other software functions using the same master clock (e.g. ADAT transmit or I2S).

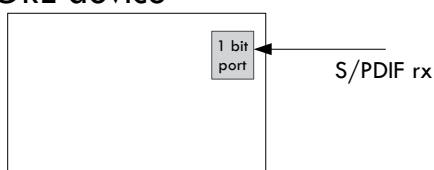
2.2 Connecting to the xCORE as Receiver

The receiver can receive stereo PCM signals up to 192 KHz.

The connection of an S/PDIF receiver line to the xCORE is shown in Figure 2.

Figure 2: Connecting S/PDIF receiver

xCORE device



Only a single wire is connected - the clock is recovered from the incoming data stream.

3 Usage

All S/PDIF functions can be accessed via the `spdif.h` header:

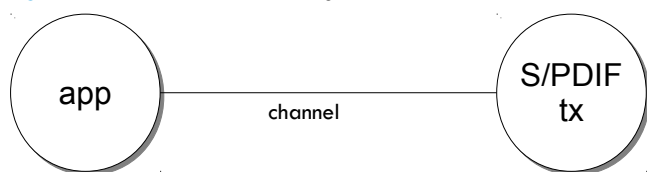
```
#include <spdif.h>
```

`lib_spdif` should also be added to the `USED_MODULES` field of the application Makefile.

3.1 S/PDIF Transmitter

S/PDIF components are instantiated as parallel tasks that run in a `par` statement. The application can connect via a channel connection.

Figure 3: S/PDIF transmit task diagram



For example, the following code instantiates an S/PDIF transmitter component and connects to it:

```
buffered out port:32 p_spdif_tx = XS1_PORT_1K;
in port p_mclk_in = XS1_PORT_1L;
clock clk_audio = XS1_CLKBLK_1;

int main(void)
{
    chanend c_spdif;
    par
    {
        on tile[0]:
        {
            spdif_tx_port_config(p_spdif_tx, clk_audio, p_mclk_in, 0);
            spdif_tx(p_spdif_tx, c_spdif);
        }

        on tile[0]: my_application(c_spdif);
    }
    return 0;
}
```

The helper function `spdif_tx_port_config()` clocks the clock-block from the master clock port and, in turn, clocks the S/PDIF transmit port from this clock-block.

The application can communicate with the components via API functions that take the channel end as arguments e.g.:

```
void my_application(chanend c_spdif)
{
    int32_t sample = 0;
    spdif_tx_reconfigure_sample_rate(c, 96000, 12288000);
    while (1)
    {
        sample++;
        spdif_tx_output(c_spdif, sample, sample + 1);
    }
}
```

3.1.1 Configuring the Underlying Clock

When using the transmit component, the internal clock needs to be configured to run off the incoming signal e.g.:

```
spdif_tx_port_config(p_spdif_tx, clk_audio, p_mclk_in, 7);
```

This function needs to be called before the `spdif_tx()` function in the programs `par` statement.

In this function the `configure_clock_src()` is used configure a clock to run off an incoming port - see the XMOS Programming Guide (see [XM-004440-PC](#)) for more information.

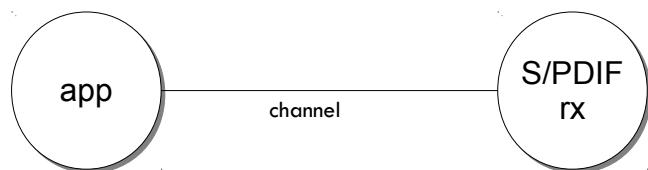
The last parameter is used with the `set_clock_fall_delay()` function to configure an internal delay from the incoming clock signal to the internal clock's falling edge. This is done to allow for the correct alignment of outgoing data with the master clock at the external D-type flip-flop.

Note, the delay value shown above is a typical example and may need to be tuned for the specific hardware being used.

3.2 S/PDIF Receiver

S/PDIF components are instantiated as parallel tasks that run in a `par` statement. The application can connect via a channel connection.

Figure 4: S/PDIF receiver task diagram



For example, the following code instantiates an S/PDIF receiver component and connects to it:

```
port p_spdif_rx = XS1_PORT_1F;
clock audio_clk = XS1_CLKBLK_1;

int main(void)
{
    streaming chan c;
    par
    {
        spdif_rx(c, p_spdif_rx, audio_clk, 96000);
        handle_samples(c);
    }
    return 0;
}
```

The application can communicate with the components via API functions that take the channel end as arguments e.g.:

```
void my_application(streaming chanend c)
{
    int32_t sample;
    size_t index;
    size_t left_count, right_count;
    while(1)
    {
        select
        {
            case spdif_receive_sample(c, sample, index):
                // sample contains the 24bit data
                // You can process the audio data here
                if (index == 0)
                    left_count++;
                else
                    right_count++;
                break;
        }
        ...
    }
}
```

Note that a program can react to incoming samples using a `select` statement. More information on using `par` and `select` statements can be found in the XMOS Programming Guide (see [XM-004440-PC](#)).

4 API

4.1 Creating an S/PDIF Receiver Instance

Function	<code>spdif_rx</code>	
Description	S/PDIF receive function. This function provides an S/PDIF receiver component. It is capable of receiving 44100, 48000, 88200, 96000, 176400 and 192000 Hz sample rates. The receiver will modify the divider of the clock-block to lock to the incoming sample rate.	
Type	<pre>void spdif_rx(streaming_chanend c, in port p, clock clk, unsigned sample_freq_estimate)</pre>	
Parameters	<code>p</code>	S/PDIF input port.
	<code>c</code>	Channel to connect to the application.
	<code>clk</code>	A clock block used internally to clock data.
	<code>sample_freq_estimate</code>	The initial expected sample rate (in Hz).

4.2 S/PDIF Receiver API

Function	spdif_receive_sample
Description	<p>Receive a sample from the S/PDIF component. This function receives a sample from the S/PDIF component. It is a “select handler” so can be used within a select e.g.</p> <pre>int32_t sample; size_t index; select { case spdif_receive_sample(c, sample, index): // use sample and index here... ... break; ... }</pre> <p>The case in this select will fire when the S/PDIF component has data ready.</p>
Type	<pre>void spdif_receive_sample(streaming_chanend c, int32_t &sample, size_t &index)</pre>
Parameters	<p>c chanend connected to the S/PDIF receiver component</p> <p>sample This reference parameter gets set with the incoming sample data</p> <p>index This is the index of the same in the current frame (i.e. 0 for left channel and 1 for right channel).</p>

Function	spdif_receive_shutdown
Description	<p>Shutdown the S/PDIF component. This function shuts down the SPDIF RX component causing the call to <code>spdif_rx()</code> to return.</p>
Type	<pre>void spdif_receive_shutdown(streaming_chanend c)</pre>
Parameters	<p>c chanend connected to the S/PDIF receiver component</p>



4.3 Creating an S/PDIF Transmitter Instance

Function	spdif_tx_port_config	
Description	S/PDIF transmit configure port function. This function configures a port to be used by the SPDIF transmit function. This function takes a delay for the clock that is to be passed into the S/PDIF transmitter component. It sets the clock such that output data is slightly delayed. This will work if I2S is clocked off the same clock but ensures S/PDIF functions correctly.	
Type	<pre>void spdif_tx_port_config(out buffered port:32 p, clock clk, in port p_mclk, unsigned delay)</pre>	
Parameters	p	the port that the S/PDIF component will use
	clk	the clock that the S/PDIF component will use
	p_mclk	The clock connected to the master clock frequency. Usually this should be configured to be driven by an incoming master system clock.
	delay	delay to uses to sync the SPDIF signal at the external flip-flop

Function	spdif_tx	
Description	S/PDIF transmit function. This function provides an S/PDIF transmit component. It is capable of 11025, 12000, 22050, 24000, 44100, 48000, 88200, 96000, and 192000 Hz sample rates. The sample rate can be dynamically changes during the operation of the component. Note that the first API call to this component should be to reconfigure the sample rate (using the <code>spdif_tx_reconfigure_sample_rate()</code> function).	
Type	<pre>void spdif_tx(buffered out port:32 p_spdif, chanend c)</pre>	
Parameters	p_spdif	The output port to transmit to
	c	chanend to connect to the application

4.4 S/PDIF Transmitter API

Function	spdif_tx_reconfigure_sample_rate
Description	Reconfigure the S/PDIF tx component to a new sample rate. This function instructs the S/PDIF transmitter component to change sample rate.
Type	<pre>void spdif_tx_reconfigure_sample_rate(chanend c_spdif_tx, unsigned sample_frequency, unsigned master_clock_frequency)</pre>
Parameters	<p><code>c_spdif_tx</code> chanend connected to the S/PDIF transmitter</p> <p><code>sample_frequency</code> The required new sample frequency in Hz.</p> <p><code>master_clock_frequency</code> The master_clock_frequency that the S/PDIF transmitter is using</p>

Function	spdif_tx_output
Description	Output a sample pair to the S/PDIF transmitter component. This function will output a left channel and right channel sample to the S/PDIF transmitter.
Type	<pre>void spdif_tx_output(chanend c_spdif_tx, unsigned lsample, unsigned rsample)</pre>
Parameters	<p><code>c_spdif_tx</code> chanend connected to the S/PDIF transmitter</p> <p><code>lsample</code> left sample to transmit</p> <p><code>rsample</code> right sample to transmit</p>

Appendix A Known Issues

- None



Copyright © 2023, All Rights Reserved.

Xmos Ltd. is the owner or licensee of this design, code, or Information (collectively, the "Information") and is providing it to you "AS IS" with no warranty of any kind, express or implied and shall have no liability in relation to its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.

XMOS, xCore, xcore.ai, and the XMOS logo are registered trademarks of XMOS Ltd in the United Kingdom and other countries and may not be used without written permission. Company and product names mentioned in this document are the trademarks or registered trademarks of their respective owners.

