S/PDIF library

Publication Date: 2023/10/18
Document Number: NoneA

XMOS

IN THIS DOCUMENT

Version	5.0.1
Vendor	XMOS
scope	General Use

1 Summary

A software defined S/PDIF library that allows you to transmit and receive 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 up to sample rates up to 96KHz
- Supports stereo S/PDIF transmit up to 192KHz

1.2 Software Version and Dependencies

The CHANGELOG contains information about the current and previous versions. For a list of direct dependencies, look for DEPENDENT_MODULES in lib_spdif/module_build_info.

1.3 Related Application Notes

The following application notes use this library:

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

2 Required Software (dependencies)

- None

3 Documentation

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

4 Support

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

- `http://www.xmos.com/support`

5 Required Software (dependencies)

- None

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

7 External Signal Description

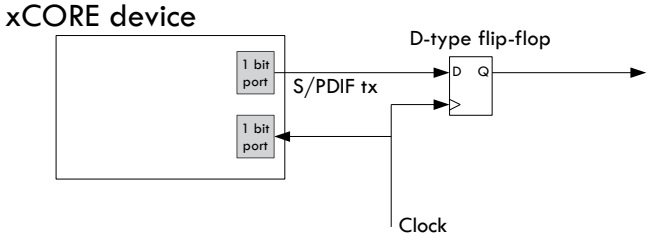
The library implements the S/PDIF (Sony/Philips Digital Interface Format) protocol for carrying uncompressed stereo PCM data of up to 24bits.

7.1 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



The output signal will contain jitter at the level of +-1 core clock (<2ns 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.

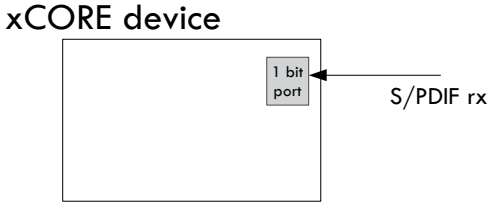
The incoming clock signal is used to drive an internal clock and can be shared with other software functions using the same master clock (e.g. ADAT transmit or I2S).

7.2 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



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

8 Usage

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

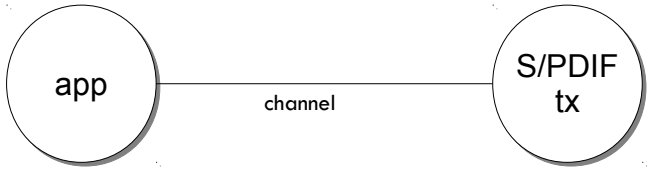
```
#include <spdif.h>
```

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

8.1 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);
  }
}
```

8.2 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.

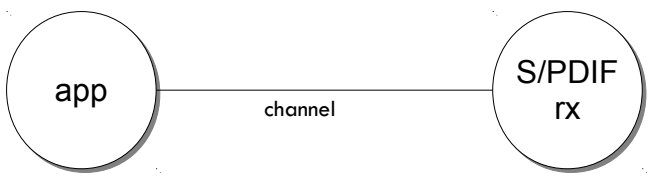
In this function the `configure_clock_src` will configure a clock to run off an incoming port (see the XMOS tools user guide for more information). The `set_clock_fall_delay` function configures an internal delay from the incoming clock signal to the internal clock. This will enable the correct alignment of outgoing data with the clock. Other components such as I2S can still be used with the same clock after setting this delay.

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

8.3 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).

9 API

9.1 Creating an S/PDIF Receiver Instance

Function	**spdif_rx**
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 modifiy the divider of the clock-block to lock to the incoming sample rate.

Type	`void spdif_rx(streaming chanend c,`
	`in port p,`
	`clock clk,`
	`unsigned sample_freq_estimate)`

Parameters	`p` S/PDIF input port. `c` Channel to connect to the application. `clk` A clock block used internally to clock data. `sample_freq_estimate` The initial expected sample rate (in Hz).

9.2 S/PDIF Receiver API

Function	spdif_receive_sample

Description	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.
	`int32_t sample;`
	`size_t index;`
	`select {`
	`  case spdif_receive_sample(c, sample, index):`
	`  // use sample and index here...`
	`  ...`
	`  break;`
	`...`
	The case in this select will fire when the S/PDIF component has data ready.

Type	`void`
	`spdif_receive_sample(streaming chanend c,`
	`  int32_t &sample,`
	`  size_t &index)`

Parameters	`c` chanend connected to the S/PDIF receiver component `sample` This reference parameter gets set with the incoming sample data `index` This is the index of the same in the current frame (i.e. 0 for left channel and 1 for right channel).

Function	spdif_receive_shutdown
Description	Shutdown the S/PDIF component. This function shuts down the SPDIF RX component causing the call to spdif_rx() to return.
Type	`void` `spdif_receive_shutdown(streaming chanend c)`
Parameters	`c` chanend connected to the S/PDIF receiver component

9.3 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	`void`
	`spdif_tx_port_config(out buffered port:32 p,`
	`clock clk,`
	`in port p_mclk,`
	`unsigned delay)`	

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 spdif_tx_reconfigure_sample_rate() function).	
Type	`void` `spdif_tx(buffered out port:32 p_spdif, chanend c)`	
Parameters	`p_spdif`	The output port to transmit to
	`c`	chanend to connect to the application

9.4 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	`void`
	`spdif_tx_reconfigure_sample_rate(chanend c_spdif_tx,`
	`  unsigned sample_frequency,`
	`  unsigned master_clock_frequency)`

Parameters	`c_spdif_tx` chanend connected to the S/PDIF transmitter `sample_frequency` The required new sample frequency in Hz. `master_clock_frequency` The master_clock_frequency that the S/PDIF transmitter is using

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	`void spdif_tx_output(chanend c_spdif_tx,`
	`  unsigned lsample,`
	`  unsigned rsample)`

Parameters	`c_spdif_tx` chanend connected to the S/PDIF transmitter `lsample` left sample to transmit `rsample` right sample to transmit

Appendix A Known Issues

- None

Appendix B lib_spdif Change Log

Appendix B.1 HEAD

- CHANGED: Receiver rearchitected for improved performance and jitter tolerance

Appendix B.2 5.0.1

- FIXED: Reinstated graceful handling of bad sample-rate/master-clock pair

Appendix B.3 5.0.0

- CHANGED: Updated examples for new XK-AUDIO-316-MC board
- CHANGED: Updated transmit to simplified implementation (note, no longer supports XS1 based devices)
- CHANGED: Removed headers SpdifReceive.h and SpdifTransmit.h. Users should include spdif.h

Appendix B.4 4.2.1

- CHANGED: Documentation updates

Appendix B.5 4.2.0

- ADDED: Added shutdown function for S/PDIF Receiver
- CHANGED: spdif_tx_example updated to use XK-AUDIO-216-MC

Appendix B.6 4.1.0

- CHANGED: Use XMOS Public Licence Version 1
- CHANGED: Rearrange documentation files

Appendix B.7 4.0.1

- REMOVED: Unrequired cpanfile

Appendix B.8 4.0.0

- CHANGED: Build files updated to support new “xcommon” behaviour in xwaf.

Appendix B.9 3.1.0

- Add library wscript to enable applications built using xwaf

Appendix B.10 3.0.0

- spdif_tx() no longer configures port. Additional function spdif_tx_port_config() provided. Allows sharing of clockblock with other tasks

Appendix B.11 2.0.2

- Fixed exception when running on xCORE-200 targets

Appendix B.12 2.0.1

- Update to source code license and copyright

Appendix B.13 2.0.0

- Move to library format. New documentation and helper functions.

Appendix B.14 1.3.1

- Added .type and .size directives to SpdifReceive. This is required for the function to show up in xTIMEcomposer binary viewer

Appendix B.15 1.3.0

- Added this file
- Removed xcommon dep