S/PDIF library

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

Features

- Supports stereo S/PDIF receive up to sample rates up to 96KHz
- Supports stereo S/PDIF transmit up to 192KHz

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	~3.8K	1
Receive	1	1 (1-bit)	1	~3.9K	1

Software version and dependencies

This document pertains to version 2.0.2 of this library. It is known to work on version 14.2.1 of the xTIMEcomposer tools suite, it may work on other versions.

The library does not have any dependencies (i.e. it does not rely on any other libraries).

Related application notes

The following application notes use this library:

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

1 External signal description

The library implements the S/PDIF (Sony/Philips Digital Interface Format) protocol for carrying uncompressed 24-bit stereo PCM data.

The precise transmission frequencies supported depend on the availability of an external clock (eg, 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 receiver can receive stereo PCM signals up to 96 Khz.

1.1 Connecting to the xCORE as transmitter

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

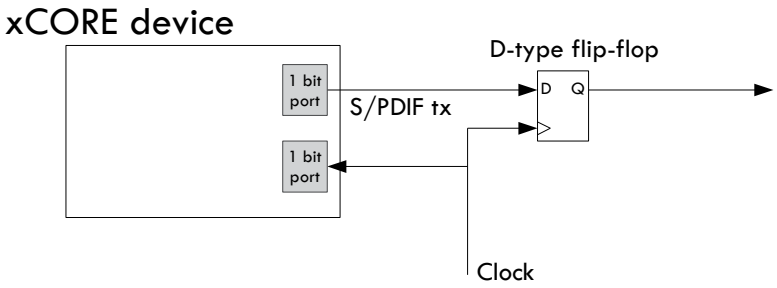

Figure 1: Connecting S/PDIF transmit

The outgoing signal should be resynchronized to the external clock using a D-type flip-flop. The incoming clock signal is used to drive an internal clock and can be shared with other software functions using the clock (e.g. S/PDIF receive or I2S).

For the best jitter tolerances on output it is recommended that a 500 Mhz part is used.

1.2 Connecting to the xCORE as receiver

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

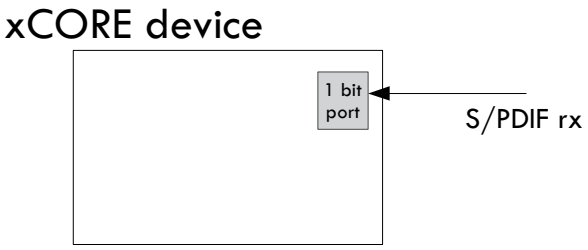

Figure 2: Connecting S/PDIF receiver

Only a single wire is connected. The clock is recovered from the incoming data signal.

2 Usage

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

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

You will also have to add `lib_spdif` to the `USED_MODULES` field of your application Makefile.

2.1 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.

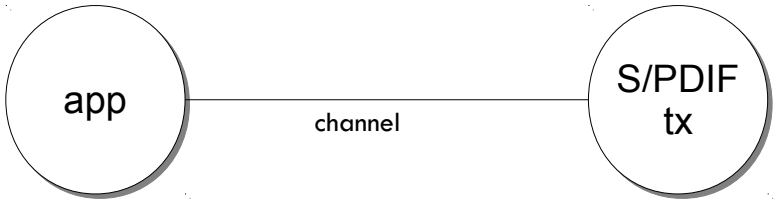

Figure 3: S/PDIF transmit task diagram

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

```
out port 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]: {
       configure_clock_src(clk_audio, p_mclk_in);
       spdif_tx_set_clock_delay(clk_audio);
       start_clock(clk_audio);
       spdif_tx(c_spdif, p_spdif_tx, clk_audio);
      }

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

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);
  }
}
```

2.2 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.

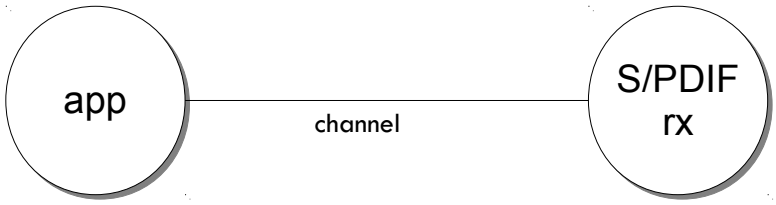

Figure 4: S/PDIF receiver task diagram

For example, the following code instantiates an S/PDIF transmitter 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 your 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).

2.3 Configuring the underlying clock

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

```
configure_clock_src(clk_audio, p_mclk_in);
spdif_tx_set_clock_delay(clk_audio);
start_clock(clk_audio);
```

These functions needs to be called before the `spdif_tx` function in the programs `par` statement.

The `configure_clock_src` will configure a clock to run off an incoming port (see the XMOS tools user guide for more information). The `spdif_tx_set_clock_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 stillbe used with the same clock after setting this delay.

3 API

3.1 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 11025, 12000, 22050, 24000, 44100, 48000, 88200 and 96000 Hz sample rates. When the decoder encounters a long series of zeros it will lower its inernal divider; when it encounters a short series of 0-1 transitions it will increase its internal divider. This means that is will lock to the incoming sample rate.

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

Parameters	`p_spdif` 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).

3.2 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).

3.3 Creating an S/PDIF transmitter instance

Function	**spdif_tx_set_clock_delay**
Description	Set the delay of a clock to enable correct timing of S/PDIF transmit. This function needs to be called 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 of the same clock but ensures S/PDIF functions correctly.
Type	`void spdif_tx_set_clock_delay(clock clk)`
Parameters	`clk` the clock that the S/PDIF component will use.

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(chanend c, out port p_spdif, const clock mclk)`
Parameters	`c` chanend to connect to the application `p_spdif` The output port to transmit to `mclk` The clock connected to the master clock frequency. Usually this should be configured to be driven by an incoming master system clock.

3.4 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,`
	`  int32_t lsample,`
	`  int32_t 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

No known issues.

APPENDIX B - SPDIF library change log

B.1 2.0.2

- Fixed exception when running on xCORE-200 targets

B.2 2.0.1

- Update to source code license and copyright

B.3 2.0.0

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

B.4 1.3.1

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

B.5 1.3.0

- Added this file
- Removed xcommon dep