SPDIF capture

From Mikes Wiki

Capturing S/PDIF is a worthy project - you can listen to CDs on an FPGA, perform real time analysis of the signal, or use it as a handy data source for experimenting with DSP algorithms. It also can be used to provide access to the sub-code information embedded within the bit stream.

Contents

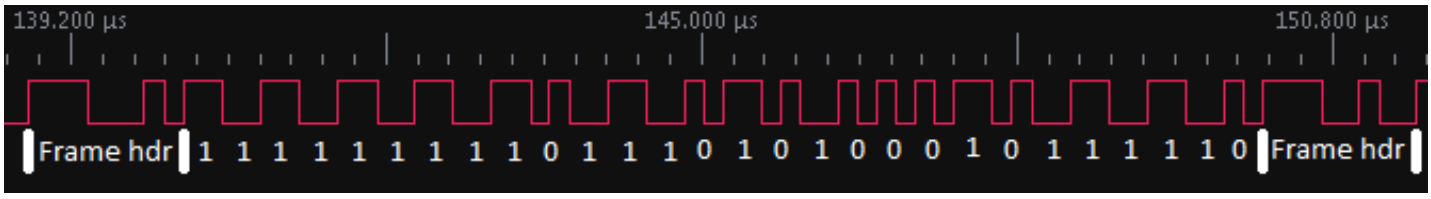
What is S/PDIF?

It is the digital audio output from CD's, PCs and other consumer devices.

In brief, it consists of a stream of subframes, each containing a header (equivilent in length to 4 bits), a 24 bit signed audio sample and 4 bits of subcode data.

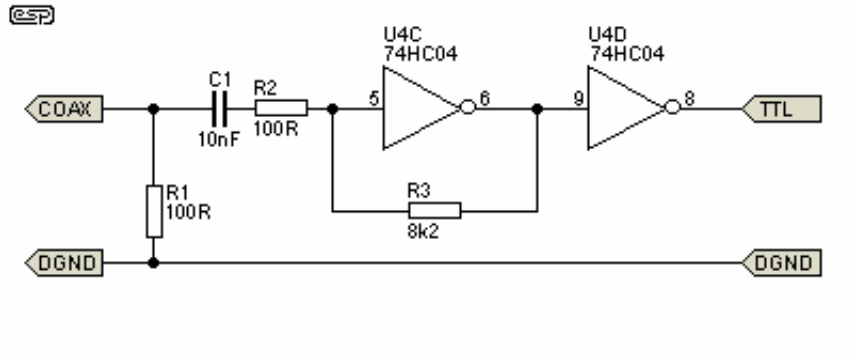
The encoding is such that there each frame is encoded into 64 clock cycles (2 per bit). The signal always 'flips' between each data bit, and it also flips in the middle of a '0' bit. The binary value 11001010 will get encoded as either 11-00-10-10-11-01-00-10 or 00-11-01-01-00-10-11-01. So a 44,200Hz stream will actually consist of 32bit per subframe * 2 clocks per bit * 2 channels * 44,200 samples per second gives a S/PDIF signaling rate of 5,657,600Hz.

To provide syncronisation of subframes, three header patterns are used - 00010111, 00011011, and 00011101 (and their inversions 11101000, 11100100, 11100010). Because these patterns break the usual rules of a signal change every other cycle it can be used to syncronise to the start of a subframe. The three different headers indicate which channel the subframe sample is for, and if the subframe is the start of a frames.



Electrical interface

Over coax, the signal is sent as a 0.5v peak-to-peak signal that needs conversion into LVTTL before it can be processed by an FPGA. I found this schematic at http://sound.westhost.com/project85.htm:



Implemented on a breadboard it looks like:

<picture>

I have tried implementing it using the FPGA's I/O pins, but it wasn't reliable - it needed a ocassional poke of a finger to get it to successfully convert to TTL. I attribute this to the short circuit protection resistors on my FPGA development board, or maybe the Schottky characteristics on the FPGA's ouptuts.

How to capture the signal

First thing is to convert the signal into the FPGA's clock domain. I also use this to detect the flips in the input bitstream:

```
entity resync is
    Port ( clk              : in  STD_LOGIC;
           bitstream        : in  STD_LOGIC;
           flipped    : out STD_LOGIC;
            synced            : out   STD_LOGIC);
end resync;

architecture Behavioral of resync is
     signal ff1,ff2 : std_logic;
begin
     flipped <= ff1 xor ff2;
     synced <= ff2;

     process (clk, pulse, ff1, ff2)
     begin
             if clk'event and clk = '1' then
                     ff2 <= ff1;
                     ff1 <= bitstream;
             end if;
     end process;
 end Behavioral;
```

Failure to reclock caused me much grief.

One way to recover the S/PDIF data is to count the length of the pulses, giving pulses that are either one S/PDIF clock, two clock or three clocks in length. This works well, but needs a finite state machine to work out where the headers are and then to recover the data bits.

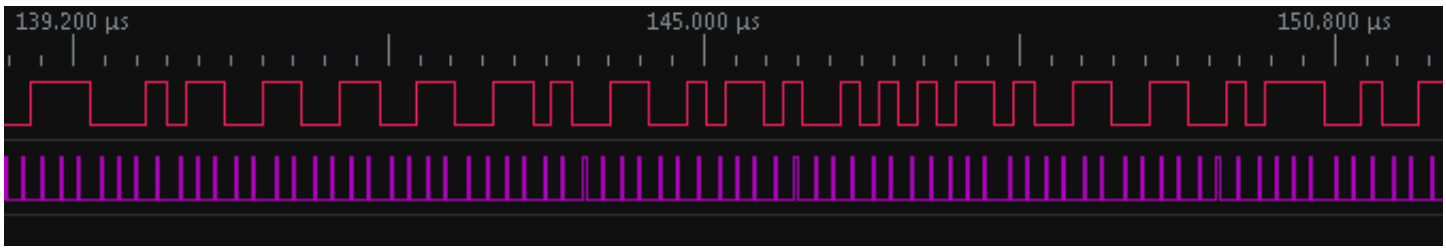
I chose to recover something close to the the sender's original clock, and use this to sample the signal into a 64 bit shift register the size of the frame. The highest 8 bits can be checked for a frame header, and the bits can be recovered by comparing even and odd positions in the shift register. Here's how the frame is assembled:

```
entity frameCapture is
    Port ( clk         : in  STD_LOGIC;
           bitstream   : in  STD_LOGIC;
           takeSample : in  STD_LOGIC;
           data        : out STD_LOGIC_VECTOR (23 downto 0);
           channelA    : out STD_LOGIC;
            dataValid  : out std_logic);
end frameCapture;

architecture Behavioral of frameCapture is
     signal frame : STD_LOGIC_VECTOR (63 downto 0) := x"0000000000000000";
begin
     process(clk,bitstream)
     begin
             if clk'event and clk='1' and takeSample = '1' then
                     frame <= frame(62 downto 0) & bitstream;
             end if;
     end process;

     process(frame)
     begin
             -- checking for a subframe header
             dataValid <= '0';
             channelA  <= '0';
             if frame(63 downto 56) = "00010111" or
                     frame(63 downto 56) = "11101000" then
                     dataValid <= '1';
                     channelA  <= '1';
             end if;

             if frame(63 downto 56) = "00011101" or
                     frame(63 downto 56) = "11100010" then
                     dataValid <= '1';
                     channelA  <= '1';
             end if;

             if frame(63 downto 56) = "00011011" or
                     frame(63 downto 56) = "11100100" then
                     dataValid <= '1';
                     channelA  <= '0';
             end if;
     end process;

     -- Recovery of data bits
     data( 0) <= not frame(55) xor frame(54);
     data( 1) <= not frame(53) xor frame(52);
     ...
     data(21)  <= not frame(13) xor frame(12);
     data(22)  <= not frame(11) xor frame(10);
     data(23)  <= not frame( 9) xor frame( 8);
end Behavioral;
```

So, how to regenerate something approaching the sender's clock? I chose to find the length of the shortest pulse, and then sample at 0.5x, 1.5x and 2.5x the minimum pulse length from a flip of the input signal. If the signal does not flip within four times the minimum sample time it indicates that minimum pulse length is incorrect, or the signal is no longer present.

```
architecture Behavioral of reclock is
      ...
      type reclock_reg is record
              count                                   : STD_LOGIC_VECTOR(9 downto 0);
              takeSample                      : STD_LOGIC;
              resetInputCounter : STD_LOGIC;
      end record;
      signal r : reclock_reg := ("0000000000",'0','0');
      signal n : reclock_reg;
 begin
      ...
      process(flipped, r, oneAndAHalfPulse, twoAndAHalfPulse, fourPulse)
      begin
              n.count             <= r.count+1;
              n.takeSample        <= '0';
              n.resetInputCounter <= '0';

              if n.count >= fourPulse then
                      n.resetInputCounter <= '1';
              end if;

              if n.count  = halfPulse then
                      n.takeSample <= '1';
              elsif n.count  = twoAndAHalfPulse then
                      n.takeSample <= '1';
              elsif n.count = oneAndAHalfPulse then
                      n.takeSample <= '1';
              end if;

              if flipped = '1' then
                      n.count <= "0000000001";
              end if;
      end process;

      -- Assign next State
      process (clk, n)
      begin
          if clk'event and clk = '1' then
              r <= n;
          end if;
      end process;
 end Behavioral;
```

Here is the original bitstream, and a second trace of the trigger used for sampling:



This is sub-optimal - if the minimum pulse is just under 5 FPGA cycles 2.5 x 4 cycles = 10 cycles - close enough that a sampling error can occur. Myabe sampling at (minimum pulse len-1), (2*minimum pulse len-1), (3*minimum pulse len-1) would be better when the FPGA clock rate is not many times that of the SPDIF signaling rate.

And that is pretty much it

Converting samples back to audio

Once you have the data, it's pretty simple to send it into a two generic 1bit DACs and listen to the sound. Just remeber to convert the signed integer sample into an unsigned value for the DAC by inverting bit 15:

```
entity dac16 is
   Port ( clk : in  STD_LOGIC;
       data : in  STD_LOGIC_VECTOR (15 downto 0);
       dac_out : out  STD_LOGIC);
end dac16;

architecture Behavioral of dac16 is
     signal sum : STD_LOGIC_VECTOR (16 downto 0) := "01000000000000000";
begin
     dac_out <= sum(16);
     process (Clk, sum)
     begin
             if Clk'Event and Clk = '1' then
                     -- Don't forget to flip data(15) to convert it to an unsinged int value
                     sum <= ("0" & sum(15 downto 0)) + ("0" & (not data(15)) & data(14 downto
             end if;
     end process;
end Behavioral;
```

Retrieved from "http://192.168.56.10/index.php/SPDIF_capture"

- This page was last modified on 3 April 2011, at 21:42.