FPGA DVB-S Encoder

Die Idee ist, einen DVB-S-Encoder in VHDL zu realisieren.

Links / Referenzen

- ETSI-Standard DVB-S
- drmpeg gr-dvbs
- TS-Erzeugung CBR

Schnittstellen

- Schnittstelle zum PC: Ethernet (UDP)
- Schnittstelle zum I/Q-Modulator: 2xDAC

Komponenten

Die geplante Komponentenstruktur wurde in KiCAD erstellt, was bei der Planung ungemein hilft:

- Download Blockschaltbild
- KiCAD-Projekt und Schaltplanfiles

Designfragen:

- Können die FrameSync-Eingänge einfach durch den Reset ersetzt werden / sind sie notwendig?

Bis nach dem Interleaver ist die Struktur byteweise, danach arbeitet sie bit-seriell. Die Pipeline muss vor dem RS-Encoder aller 188 Byte angehalten werden können, damit der RS-Encoder seine sechs Paritätsbytes einschieben kann.

Controller

Aufgabe:

- Datenstrom überwachen (Frame-Syncronität)
- Steuersignale für die einzelnen Komponenten erzeugen
- noch keine Modulspec!

Netzwerk-RX

Aufgabe:

- Empfang von UDP-Paketen (Sanity-Check)
- Weiterreichen der Nutzdaten an FIFO

- keine Modulspec hier (implementiert Stefan)

Netzwerk-TX

Aufgabe:

- Auswerten der FIFO-Signale und Erzeugung von UDP-Nachichten zur Datenflusskontrolle
- Wenn FIFO fast leer: „Mach schneller“ senden
- Wenn FIFO fast voll: „Mach langsamer“ senden
- keine Modulspec hier (implementiert Stefan)

FIFO

Aufgabe:

- MPEG-Datenstrom von Ethernet entgegennehmen und an Encoder weitergeben
- Signalisierung der noch vorhandenen Daten (zu viel / zu wenig)
- keine Modulspec hier (schon vorhanden)

Scrambler

Aufgabe:

- Scrambling der Daten entsprechend Spec - Seite 9
  - http://outputlogic.com/?page_id=205 verwenden möglich (Bytewise Scrambler)
  - LFSR Reset aller 8 Pakete
  - Erstes Byte jedes Pakets (Syncword) nicht gescramblet, Scrambler läuft aber weiter
  - Erstes Byte des ersten von acht Paketen invertieren.
  - Ein Ausgangsbyte pro Eingangsbyte (Clock Enable verwenden)

RS-Encoder

Aufgabe:

- Verkürzten RS-Code auf jeweils einen MPEG-Frame anwenden (Spec Seite 10)
- Paritätsbytes einfügen
- Sync-Signal signalisiert das erste Byte des Pakets
- Bei jedem Clock Enable ein Ausgangsbyte erzeugen!
  - 188x Clock Enable pro Eingangsbyte
  - 6x Clock Enable für die Paritätsbytes (keine neuen Eingangsdaten anliegend).

Interleaver

Aufgabe:

- Vertauschen der Byte-Reihenfolge (Spec Seite 10)

- Implementierung mit Dual-Port-RAM - Berechnung der Indizes
- Bei jedem Clock-Enable ein Eingangsbyte lesen und ein Ausgangsbyte erzeugen
- Sync-Eingang verwenden um Scrambler auf Pfad 0 (keine Verzögerung) zu setzen, siehe Seite 12

P/S-Converter

Aufgabe:

- Byteweisen Datenstrom in Bitweisen Datenstrom wandeln
- Bei jedem Clock-Enable-Puls ein neues Ausgangsbit erzeugen
- „Start“ signalisiert das Anliegen eines neuen Bytes → serielle Ausgabe dieses Bytes, MSB first

Convolutional Coder

Aufgabe:

- Faltungskode auf serielle Daten anwenden (Seite 12)
- Bei jedem clk_en ein neues Bit reinschieben und die X/Y-Ausgänge des Kodieres erzeugen

Mapping

Aufgabe:

- Mapping auf I/Q-Symbole (Seite 13)
- Eingang: Bits, Ausgang: Signed x-bit-Vektoren (entsprechend Spec)
- optional: Puncturing - wird erstmal weggelassen
- bei jedem Clock-Enable ein Symbol mappen

Interpolator

Aufgabe:

- Einfügen von Nullen in den Datenstrom
- bei jedem Clock-Enable einen Ausgangswert produzieren
- länge parametrisierbar
- Ausgang = 0, wenn nur clock-enable
- Ausgang = Eingangswert, wenn clock-enable und d_valid

Baseband Filter

Aufgabe:

- Spectral Shaping mit RRC-Filter
- FIR-Filter, Koeffizienten berechnen (gnuradio firdes.root_raised_cosine, Matlab …)
- Datenlänge parametrisierbar

- Filterordnung parametrisierbar?
- Bei jedem Clock-Enable ein neues Eingangsbyte lesen und ein Ausgangsbyte schreiben

Berechnung Bitrate des MPEG2-TS

- gegeben: Symbolrate 4,5 MSym/s
- QPSK, also 2 Bit pro Symbol
  - aber: aus Faltungskodierer kommen 2 Bit pro Datenbit
  - d.h.: 2×4,5Mbit/s Datenstrom am Ausgang
- Durch Puncturing: Weglassen von Datenbits, damit geringere Bitrate
  - z.B. 2/3
    - 3 Ausgangsbits pro 2 Datenbits
    - Redundanz bedeutet Faktor 2
    - also: pro Datenbit 0,75 Ausgangsbits
    - 4,5 Mbit/s / 0,75 = 6 MBit/s
- RS erzeugt aus 188 Byte immer 204 Byte
  - Geringere Nutzdatenrate, Faktor 188/204 = 0,921…
  - 6 Mbit/s * 0,921 = 5,529 Mbit/s
- Also Gesamtechnung: Sendebitrate / Bit pro Symbol / Puncturing factor * RS-Faktor

Bei Weglassen des Puncturing (Code Rate 1/2) ist die Sendesymbolrate (=Bitrate nach RS) ein ganzzahliger Teiler der Systemfrequenz. Die Bitrate des TS errechnet sich nur durch Multiplikation mit dem Reed-Solomon-Overhead-Faktor. Folgende Tabelle fasst erreichbare Datenraten bei 50MHz Systemtakt zusammen.

Clock-Divider	Brutto-Datenrate	Netto-(TS)-Datenrate
1	50 MSym/s	46,08 MBit/s
2	25 MSym/s	23,04 MBit/s
3	16,67 MSym/s	15,36 MBit/s
4	12,5 MSym/s	11,51 MBit/s
5	10 MSym/s	9,22 MBit/s
6	8,33 MSym/s	7,68 MBit/s
7	7,14 MSym/s	6,58 MBit/s
8	6,25 MSym/s	6,76 MBit/s

Netzwerkprotokoll

Eine aufwendige Stack-Implementierung soll vermieden werden - diese Aufgabe könnte später mal ein IP-Stack übernehmen. Im Moment ist es wichtig, eine stabile, Datenflusskontrollierte Verbindung in den FPGA aufzubauen. Es wird daher eine einfache Zwei-Wege-Kommunikation definiert.

Allgemeiner Paketaufbau:

Ethernet-Header	IP-Header	UDP-Header	UDP-Daten	Ethernet-Footer

Folgende Anforderungen werden spezifiziert:

- Im Ethernet-Header sollen beliebige Quellen- und Ziel-MAC-Adresse stehen dürfen
- Im IP-Header sollen beliebige Ziel- und Quell-Adressen stehen dürfen - kein Feld wird überprüft

- Im UDP-Header wird der Ziel-Port überprüft. Die Länge kann optional überprüft werden
- Das Längen-Feld aller Protokollebenen wird ignoriert
- Zum Senden werden feste (vorher definierte) Pakete mit vorher berechneten Checksummen genutzt.

Pakete PC -> FPGA

- UDP-Pakete, Ziel-Port 40000
- Payload: 7*188 Byte Daten (7 MPEG-Frames)
    - optional: variable Länge, kann vom FPGA überprüft werden

Pakete FPGA -> PC

- UDP-Pakete, Ziel-Port 40001
- Payload: 1 Byte
    - 0x00: FIFO hat unteren Schwellwert erreicht - Datenrate erhöhen
    - 0x01: FIFO hat oberen Schwellwert erreicht - Datenrate erniedrigen
    - 0x02: (optional) FIFO ist leer, einmalig
    - 0x03: (optional) FIFO ist voll, einmalig

Pakete werden nicht wiederholt, bevor die entsprechende Bedingung nicht verlassen wurde (FIFO wieder auf normalen Füllzustand zurückgekehrt).

From:
https://www.loetlabor-jena.de/ - **Lötlabor Jena**

Permanent link:
https://www.loetlabor-jena.de/doku.php?id=projekte:das:dvbs&rev=1422820038

Last update: **2015/02/01 19:47**