DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart

Entwurf digitaler Systeme

Labor

Ausgabe 0.2, 15.03.2013
Autor: Stephan Rupp

Inhaltsverzeichnis

1. Versuch 1 - Signalgenerator

Ziel des Versuchs ist der Entwurf und die Implementierung eines Funktionsgenerators zur Erzeugung digitaler Signale variabler Signalform und variabler Frequenz. Die Signale werden auf einem FPGA erzeugt und können per DA-Wandler auf einem Messgerät angezeigt werden (Oszilloskop, Spektrumanalysator).

1.1. Funktionsprinzip

Die Funktion der Schaltung beruht auf der direkten Synthese der Signale (Direct Digital Synthesis, DDS). Bei dieser Methode wird die gewünschte Signalform in einem Speicher abgelegt und von dort aus ausgelesen und auf einen Digital-Analogwandler gegeben. Das Auslesen kann hierbei mit unterschiedlichen Schrittweiten geschehen, wodurch sich eine Variation der Frequenz des Signals ergibt. Die folgende Abbildung zeigt das Prinzip für eine harmonische Signalform.

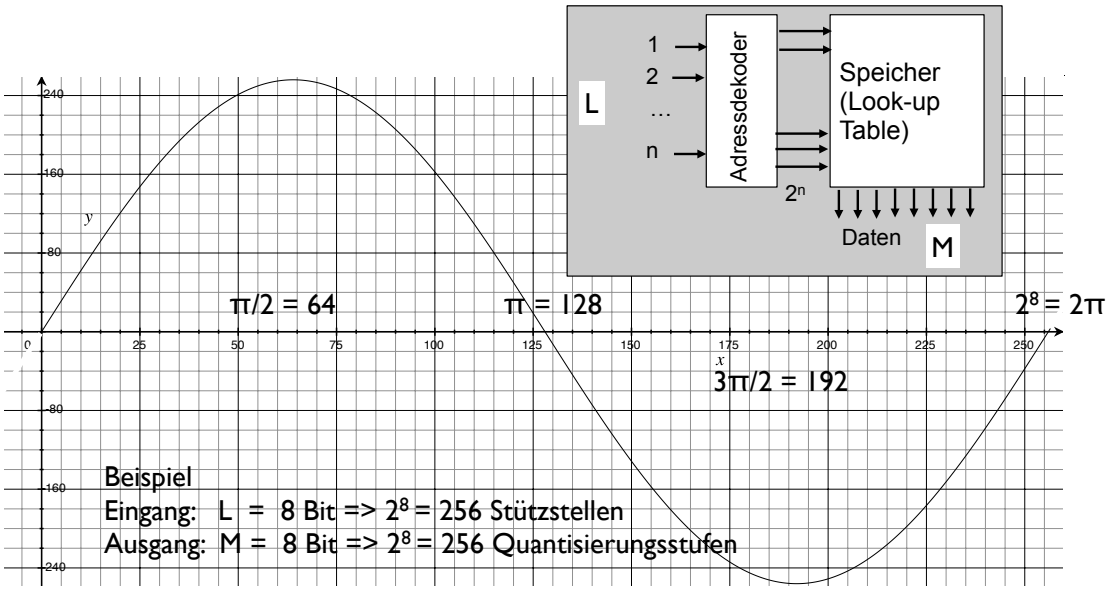

Bild 1.1 Look-up-Table mit Funktionswerten

Im Beispiel wurde das gewünschte Signal (Sinus) über eine Periode mit 256 Stützstellen gespeichert. Wenn man mit Hilfe eines Zählers alle Stützstellen der Reihe nach abfragt, ergibt sich die langsamste mit dieser Anordnung realisierbare Frequenz. Bei einer Taktrate von beispielsweise 20 MHz dauert das Abspielen einer kompletten Periode 256 / 20 MHz = 12.8 µs. Die Frequenz des erzeugten Signals beträgt also 78,125 kHz.

Die mit dieser Methode maximal realisierbare Frequenz ergibt sich, wenn man pro Periode nur die minimal erforderliche Anzahl von 2 Stützstellen ausliest (also z. B. nur die Werte bei π/2 und 3π/2). Diese Grenzfrequenz ergibt sich bei einer Taktrate von 20 MHz also zu 10 MHz.

Somit beträgt in diesem Beispiel der Umfang des realisierbaren Frequenzbereichs 78,125 kHz bis 10 MHz. Wegen der auf 256 begrenzten Anzahl der Stützstellen der Funktion sind Frequenzen in Vielfachen von f_d = 78,125 kHz realisierbar. Ausgehend von der Taktrate f_c bestimmt sich der Frequenzbereich zu $f_d = f_c / 2^L$ und der Grenzfrequenz $f_g = f_c / 2$.

Aufgabe 1.1: Nehmen Sie an, die Eingangswortbreite der zum Speichern der Stützstellen verwendeten LUT (Look-up Table) beträgt 12 bit. Die Taktrate zum Auslesen der Stützstellen wird wiederum mit 20 MHz angenommen. Welchen Frequenzbereich kann die Schaltung abdecken?

Aufgabe 1.2: Nehmen Sie an, die Taktrate des FPGA-Bausteins beträgt 80 MHz, Sie möchten allerdings bei einer Grenzfrequenz von 10 MHz bleiben, und die gleiche Schaltung wie in Aufgabe 1.1 verwenden. Wie gehen Sie vor?

Die Berechnungen zeigen, dass sich Eingangswortbreite L des Speicherelements (LUT) nach der Anzahl der gewünschten Stützstellen richtet. Diese wiederum richten sich nach dem gewünschten Frequenzumfang der Schaltung. Die Ausgangswortbreite M des Speicherelements (LUT) richtet sich nach der gewünschten Güte des Ausgangssignals. Stehen nur 8-Bit D/A Wandler zur Verfügung, genügen auch 8-Bit Ausgangswortbreite. Für anspruchsvollere Aufgaben würde man mit 12-Bit oder 16-Bit AD-Wandlern arbeiten.

Zur Erzeugung der Signale in der beschriebenen Weise würde man den Funktionsspeichers (LUT) in geeigneter Weise ansteuern. Die Ansteuerung soll ein zyklisches Abfragen der Eingänge mit variabler Schrittweite ermöglichen. Ausserdem ermöglicht die Ansteuerung auch das Abfragen von Phasenbezügen.

Aufgabe 1.3: Beschreiben Sie, wie Sie ein Signal mit halber Anzahl der Stützstellen (also doppelter minimaler Frequenz f_d) erzeugen? Welche Schrittweite ist hierfür erforderlich? Wie können Sie die Schrittweite variable gestalten?

Aufgabe 1.4: Beschreiben Sie, wie Sie mit Hilfe des Funktionsspeichers ein Kosinus-Signal erzeugen können. Wie lässt sich eine variable Phasenbeziehung bei vorgegebener Frequenz einstellen?

Folgendes Blockschaltbild beschreibt die komplette Signalkette, bestehend aus:

- einem Register für das Phaseninkrement (PIR, Phase Increment Register). Das Phaseninkrement beschreibt die Schrittweite, in der die Stützstellen der Funktion ausgelesen werden sollen.
- einer Schaltung zur Adressierung des Funktionsspeichers (LUT) mit der vorgegebenen Schrittweite, bestehend aus einem Addierer und einem Phasenregister
- dem Funktionsspeicher (LUT)
- dem Digital-Analogwandler.

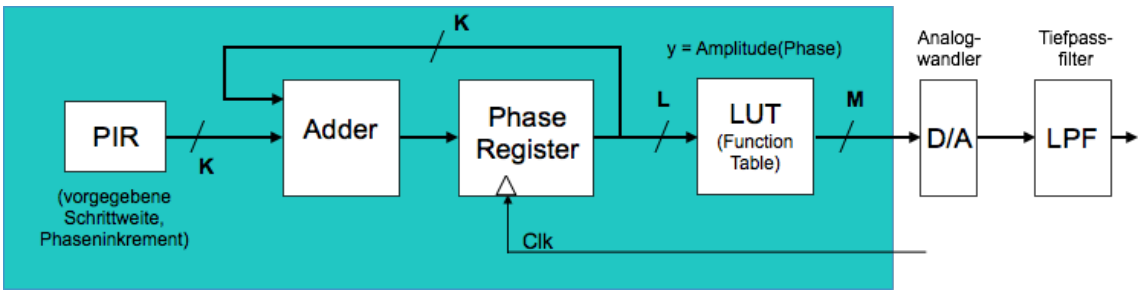

Bild 1.2 Blockschaltbild des Signalgenerators mit variabler Frequenz

Aufgabe 1.5: Erläutern Sie die Funktion der Schaltung. Hinweis: Nehmen Sie folgende Wortbreiten an: K = L = 12 Bit, M = 8 Bit. Geben Sie eine Schrittweite vor. Wie wird die zyklische Ansteuerung des Funktionsspeichers erreicht (Überlauf am Ende der Tabelle)?

Aufgabe 1.6: Die Wortbreite des Phasenregisters wird auf K = 24 Bit erhöht, wobei zur Adressierung des Funktionsspeichers weiterhin L = 12 Bits verwendet werden. Welche Änderung ergibt sich hierdurch?

Aufgabe 1.7: In Ergänzung zu Aufgabe 1.6: Würde es in der gegebenen Anordnung Sinn machen, die Wortbreite des Funktionsspeichers ebenfalls auf L = 24 Bit zu erhöhen? Begründen Sie Ihre Aussage.

Bemerkung: Mathematisch betrachtet nutzt man folgende Beziehung zwischen Frequenz und Phase: f = dϕ /dt. Bei konstanter Frequenz schreitet die Phase linear fort. Je größer die Phasenänderung pro Zeiteinheit, desto höher ist also die Frequenz. Somit lässt sich durch das Phaseninkrement also die Frequenz definieren. Dieser Zusammenhang ist natürlich auch an der Funktionstabelle unmittelbar erkennbar.

Die in Abbildung 1.2 wiedergegebene Schaltung ermöglicht zwar die Einstellung der Frequenz (= Phaseninkrement) mit Hilfe des Phaseninkrement-Register (PIR). Die Schaltung ermöglicht jedoch nicht nicht die Einstellung einer Phasenverschiebung, wie z.B. $\phi_0 = 90^o$. Um bei vorgegebener Frequenz eine Phasenverschiebung (engl. phase offset) einzustellen, wird daher ein weiteres Register ergänzt. Dieses Register (Phase Offset Register, POS) erlaubt die Einstellung einer konstanten Phasenverschiebung unabhängig von der Frequenz.

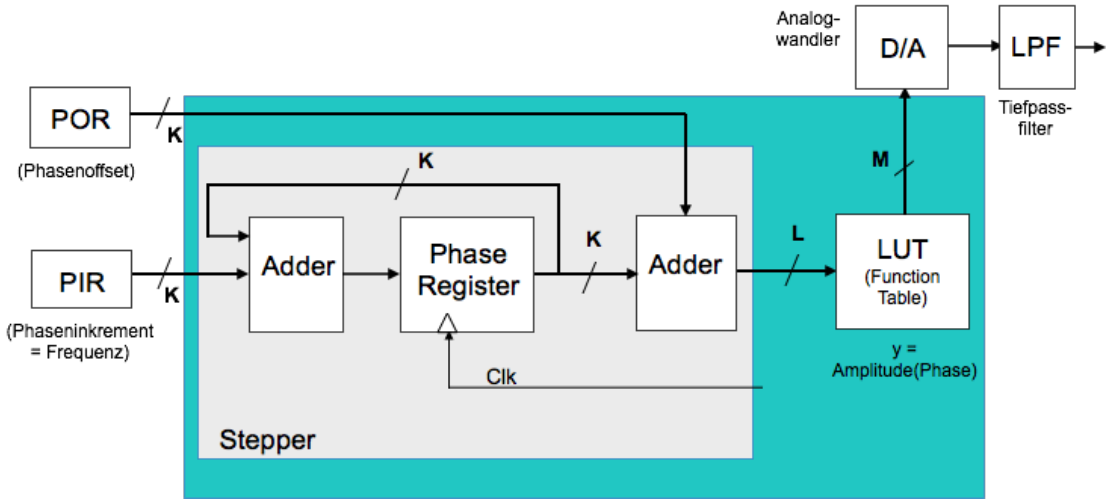

Bild 1.3 Blockschaltbild des Signalgenerators

Die Schaltung enthält somit folgende Funktionsblöcke: (1) ein Schaltwerk für Phaseninkrement und Phasenoffset (als „Stepper“ bezeichnet), (2) den gespeicherten Signalverlauf (Funktionstabelle, LUT). Bemerkung: Sofern nicht wirklich eine flexible Phase erwünscht ist, sondern beispielsweise nur zueinander orthogonale Signale (mit +/- 90° Phasenverschiebung), kann man auch mit zwei Speichern (LUT) für ein sinusförmigen und kosinusförmigen Signalverlauf arbeiten. Diese Signale addiert man dann in Abhängigkeit des zu modulierenden Eingangssignals.

Die Vorgabe des Phaseninkrements und des Phasenoffsets erlauben viele Möglichkeiten zur Erzeugung digitaler Signale, beispielsweise phasenmodulierte Signale durch definierte Phasen-

sprünge, bzw. frequenzmodulierte Signale, Frequenzsprungverfahren, bzw. Signaldurchläufe durch den gesamten Frequenzbereich (signal sweeps). Solche Spezialfälle lassen sich durch eine komplexere Beschaltung des Phaseninkrements und Phasenoffsets realisieren, wobei diese dann zeitlich veränderliche Größen werden.

Natürlich lassen sich mit sehr geringem Aufwand mit der in Abbildung 1.3 gezeigten Anordnung auch andere Signalformen als harmonische Signale erzeugen. Durch alternative Belegung des Funktionsspeichers sind beispielsweise Dreieck-Signale, Sägezahn, Rechteck und sonstige Signalverläufe realisierbar.

1.2. Schaltungsentwurf

Für den Signalgenerator soll folgendes Blockschaltbild verwendet werden. Für den Signalausgang stehen 8-Bit Analogwandler zur Verfügung. Die Schaltung wird mit 20 MHz Taktfrequenz betrieben. Eine Periode wird bzgl. der Vorgaben für die Frequenz (= Phaseninkrement) und den Phasenoffset in 2^{20} Stützstellen unterteilt, d.h. die Wortbreite der Register und des Schaltwerks wird mit 20 Bit gewählt. Die Schaltung soll ausserdem über einen Eingang für das Taktsignal (Clk), sowie über einen Eingang zum Rücksetzen (Reset) verfügen.

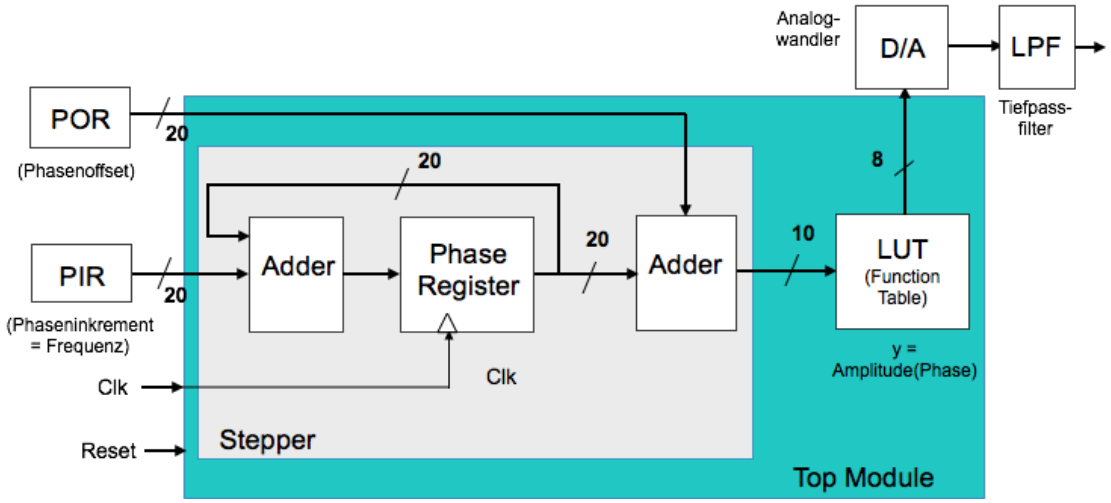

Bild 1.4 Blockschaltbild des Signalgenerators

Aufgabe 1.8: Wir groß ist der Frequenzumfang der Schaltung? Reicht die Schaltung in den Audiobereich? Welche Frequenzschritte sind möglich, ausgehend von der tiefsten Frequenz?

Aufgabe 1.9: Für den Eingang der Funktionstabelle (LUT) wird nur eine Wortbreite von 10 Bit verwendet (die oberen 10 Bits des Schaltwerks). Welche Einschränkungen ergeben sich hierdurch?

Für den Schaltungsentwurf ergeben sich aus dem Blockschaltbild folgende Module:

- das Schaltwerk für den Stepper (grau unterlegter Block)
- die Funktionstabelle (LUT)
- ein übergeordnetes Modul (Top Modul), das beide Module verbindet (grün unterlegter Block).

Diese drei Module sind nun als Schaltung zu entwerfen, d.h. in VHDL zu beschreiben. Für den Entwurf werden Muster vorgegeben, die Sie als Basis verwenden können. Hinweis: Wenn Sie in zu zweit bzw. in Gruppen organisiert sind, können Sie die Aufgaben folgendermassen aufteilen und parallel starten: (1) Schaltungsentwurf und funktionale Verifikation: machen Sie sich mit der Funktion der Schaltung und der Simulation der Schaltung vertraut, damit Sie auf Tests und mögliche Fehler bei der Implementierung vorbereitet sind. (2) Schaltungssynthese: Machen Sie sich mit den Synthesewerkzeugen vertraut. Verwenden Sie hierzu die vorgegebenen Muster als Dateivorlagen für den Umgang mit den Werkzeugen zur Synthese und Implementierung.

Aufgabe 1.10: Entwerfen Sie das Top Modul.

Als Vorlage können Sie die folgende HDL-Beschreibung verwenden. Analysieren Sie die Vorlage und ändern Sie gegebenenfalls nach Bedarf. Speichern Sie die Datei auf einem Verzeichnis zusammen mit den folgenden HDL-Beschreibungen im VHDL-Format (.vhd) so, dass der HDL-Simulator später direkt darauf zugreifen kann.

```
--- Top Module for DDS Generator (Direct Digital Synthesis)
--- VHDL

library ieee;
use ieee.std_logic_1164.all;

entity TOP is

        -- define K, L and M as constants (natural = positive integer)
        generic ( STEPPER_BITS : natural := 20; -- stepper is K=20 bits
                    LUT_BITS  : natural := 10; -- LUT is L=10 bits
                    DAC_BITS  : natural :=  8);-- DAC is M=8 bits

        -- declare external ports of top module
        port ( Clk, RS : in std_logic;          -- clock, reset
                   PIR : in std_logic_vector (STEPPER_BITS-1 downto 0);
                   POR : in std_logic_vector (STEPPER_BITS-1 downto 0);
                   DAC : out std_logic_vector(DAC_BITS-1 downto 0));
        end TOP;

architecture RTL of TOP is

        component Stepper is
              generic (STEPPER_BITS : natural;
                          LUT_BITS : natural);
              port (Clk, RS : in std_logic;
              PIR : in std_logic_vector (STEPPER_BITS-1 downto 0);
              POR : in std_logic_vector (STEPPER_BITS-1 downto 0);
              LUT : out std_logic_vector(LUT_BITS-1 downto 0));
        end component Stepper;

        component LookUpTable is
              generic (LUT_BITS : natural;
```

```
                        DAC_BITS : natural);
                port ( LUT : in  std_logic_vector(LUT_BITS-1 downto 0);
                       DAC : out std_logic_vector(DAC_BITS-1 downto 0));
        end component LookUpTable;

-- declare internal signal of Top module (from Stepper to LookUpTable)
   signal stepperOut :  std_logic_vector(LUT_BITS-1 downto 0);

-- connect Stepper to LookUpTable
 begin

 STEP: Stepper generic map(STEPPER_BITS=>STEPPER_BITS,
                           LUT_BITS=>LUT_BITS)
    port map(Clk=>Clk, RS=>RS, PIR=>PIR, POR=>POR, LUT=>stepperOut);

 LUTB: LookUpTable generic map(LUT_BITS=>LUT_BITS, DAC_BITS=>DAC_BITS)
                   port map(LUT=>stepperOut, DAC=>DAC);

 end RTL;
```

Aufgabe 1.11: Entwerfen Sie das Schaltwerk (Stepper).

Folgende HDL-Beschreibung können Sie als Vorlage verwenden. Analysieren Sie die Vorlage und ändern Sie gegebenenfalls. Speichern Sie die Datei zusammen mit den anderen zur Aufgabe gehörigen HDL-Beschreibungen auf einem Verzeichnis ab.

```
--- Stepper Module for DDS Generator (Direct Digital Synthesis)
--- VHDL

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- IEEE 1076.3 standard

entity Stepper is

        generic (STEPPER_BITS : natural;  -- K bits wide
                  LUT_BITS : natural); -- L bits wide
        port (Clk, RS : in std_logic;
              PIR : in std_logic_vector (STEPPER_BITS-1 downto 0);
              POR : in std_logic_vector (STEPPER_BITS-1 downto 0);
              LUT : out std_logic_vector(LUT_BITS-1 downto 0));
              -- internal port to Top
end Stepper;

architecture RTL of Stepper is

-- store accumulated phase in Phase Register
signal phaseReg : unsigned ((STEPPER_BITS -1) downto 0);

-- phase output to LUT
```

```
    signal phaseOut : unsigned ((STEPPER_BITS -1) downto 0);

    begin
           process (Clk, RS)
           begin
                 if RS ='1' then   -- reset to initial state
                       phaseReg <= (others => '0');
                       phaseOut <= (others => '0');
                 elsif rising_edge(Clk) then
                       -- increment phase
                       phaseReg <= phaseReg + unsigned(PIR);
                       -- add offset
                       phaseOut <= phaseReg + unsigned(POR);
                 end if;

           end process;

           -- use top K-M bits of phaseOut to address loop up table
           LUT <= std_logic_vector(phaseOut((STEPPER_BITS-1) downto
                                        (STEPPER_BITS - LUT_BITS)));

    end RTL;
```

Aufgabe 1.12: Entwerfen Sie die Funktionstabelle (LUT).

Als Vorlage können Sie wiederum folgende HDL-Beschreibung verwenden. Analysieren Sie die Vorlage und ändern Sie sie gegebenenfalls. In der Vorlage fehlen noch die Funktionswerte für den Funktionsspeicher (LUT). Erzeugen Sie diese Werte mit der gewünschten Auflösung (8-Bit Werte) und mit der gewünschten Anzahl Stützstellen. Stellen Sie den Schaltungsentwurf fertig.

```
    --- LookUpTable Module for DDS Generator (Direct Digital Synthesis)
    --- VHDL

    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;

    entity LookUpTable is

           generic (LUT_BITS : natural;   -- L bits wide
                    DAC_BITS : natural);  -- M bits wide

           port ( LUT : in  std_logic_vector(LUT_BITS-1 downto 0);
                  DAC : out std_logic_vector(DAC_BITS-1 downto 0));

    end LookUpTable;

    architecture RTL of LookUpTable is

        -- function table of 2**L samples, each M bits wide
```

```
        type lut_array is array(0 to 2**LUT_BITS - 1) of
               std_logic_vector(DAC_BITS - 1 downto 0);

        -- fill function table with 2**M sample values
        signal ltb : lut_array := ( x"80",
                    -- fill in sample values here ...
                    );

        -- look-up function in table: y = amplitude(phase)
        begin
        DAC <= ltb(conv_integer(LUT));

end RTL;
```

1.3. Funktionale Verifikation

Übernehmen Sie die Schaltungsentwürfe für den Generator bestehen aus den drei Modulen aus Abschnitt 1.2 in den HDL-Simulator und übersetzen Sie den Schaltungsentwurf. Übersetzen Sie die Dateien zunächst einzeln. Arbeiten Sie bitte direkt auf dem Verzeichnis, auf dem Sie die HDL-Dateien abgelegt haben, so dass bei der Fehlerkorrektur dort der fehlerfreie Quellcode abgelegt bleibt.

Aufgabe 1.13: Übersetzen Sie den Schaltungsentwurf und beheben Sie alle ggf. auftretenden Fehler, die der Compiler meldet.

Überlegen Sie sich eine Methode, um den Schaltungsentwurf zu verifizieren. Hierzu benötigen Sie eine Testumgebung, die alle Eingangssignale des Top Moduls stimuliert und die Ausgangssignale aufnimmt.

Aufgabe 1.14: Entwerfen Sie ein Testumgebung für das Top-Modul.

Folgende HDL-Beschreibung können Sie als Vorlage verwenden. Analysieren Sie die Vorlage und ändern Sie gegebenenfalls. Speichern Sie die Datei zusammen mit den anderen zur Aufgabe gehörigen HDL-Beschreibungen auf einem Verzeichnis ab.

```
--- Testbench for Top Module of DDS Generator (VHDL)

library ieee;
use ieee.std_logic_1164.all;

entity test_TOP is

        -- define K, L and M as constants
        generic ( STEPPER_BITS : natural := 20;  -- stepper is K=20
                    LUT_BITS   : natural := 10   -- LUT is L=10 bits
                    DAC_BITS   : natural :=  8); -- DAC is M=8 bits

-- no external signals in test bench
end test_TOP;
```

```
architecture Behavioural of test_TOP is

        component TOP is
              generic ( STEPPER_BITS : natural;
                              LUT_BITS  : natural;
                              DAC_BITS  : natural);
              port ( Clk, RS : in std_logic;      -- clock, reset
              PIR : in std_logic_vector (STEPPER_BITS-1 downto 0);
              POR : in std_logic_vector (STEPPER_BITS-1 downto 0);
              DAC : out std_logic_vector(DAC_BITS-1 downto 0));
        end component TOP;

        -- define test signals
        signal T_Clk : std_logic := '0';
        signal T_RS  : std_logic := '1';
        signal T_PIR : std_logic_vector (STEPPER_BITS-1 downto 0)
                                        := (others => '0');
        signal T_POR : std_logic_vector (STEPPER_BITS-1 downto 0)
                                        := (others => '0');
        signal T_DAC : std_logic_vector(DAC_BITS-1 downto 0);

        -- connect DUT to testbench
        begin
        DUT: TOP generic map(STEPPER_BITS => STEPPER_BITS,
                    LUT_BITS => LUT_BITS, DAC_BITS => DAC_BITS)
                  port map(Clk => T_Clk, RS => T_RS, PIR => T_PIR,
                    POR => T_POR, DAC => T_DAC);

        -- run tests
        test: process
        begin
              wait for 50 ns;
              T_RS <= '0';

              T_PIR <= b"0000_0000_0100_0000_0000";
              T_POR <= b"0000_0000_0000_0000_0000";

              for j in 1 to 10000 loop
                   T_Clk <= '0';
                   wait for 25 ns;
                   T_Clk <= '1';
                   wait for 25 ns;
              end loop;

              wait;
     end process;
 end Behavioural;
```

Aufgabe 1.15: Testen Sie das Modul im Simulator. Geben Sie hierzu sinnvolle Werte für das Phaseninkrement und den Phasenoffset vor. Analysieren Sie die Ausgangssignale des Prüflings (DUT). Prüfen Sie die Ergebnisse auf Plausibilität.

Aufgabe 1.16: Dokumentieren Sie die Ergebnisse Ihrer Tests. Legen Sie diese Dokumentation Ihrem Laborbericht bei.

1.4. Schaltungssynthese und Implementierung

Setzen Sie sich mit den Werkzeugen zur Synthese und Implementierung auseinander. Stellen Sie den Zusammenhang mit dem in der Abbildung gezeigten grundsätzlichen Ablauf her.

Aufgabe 1.17: Erläutern Sie den in der Abbildung gezeigten Ablauf.

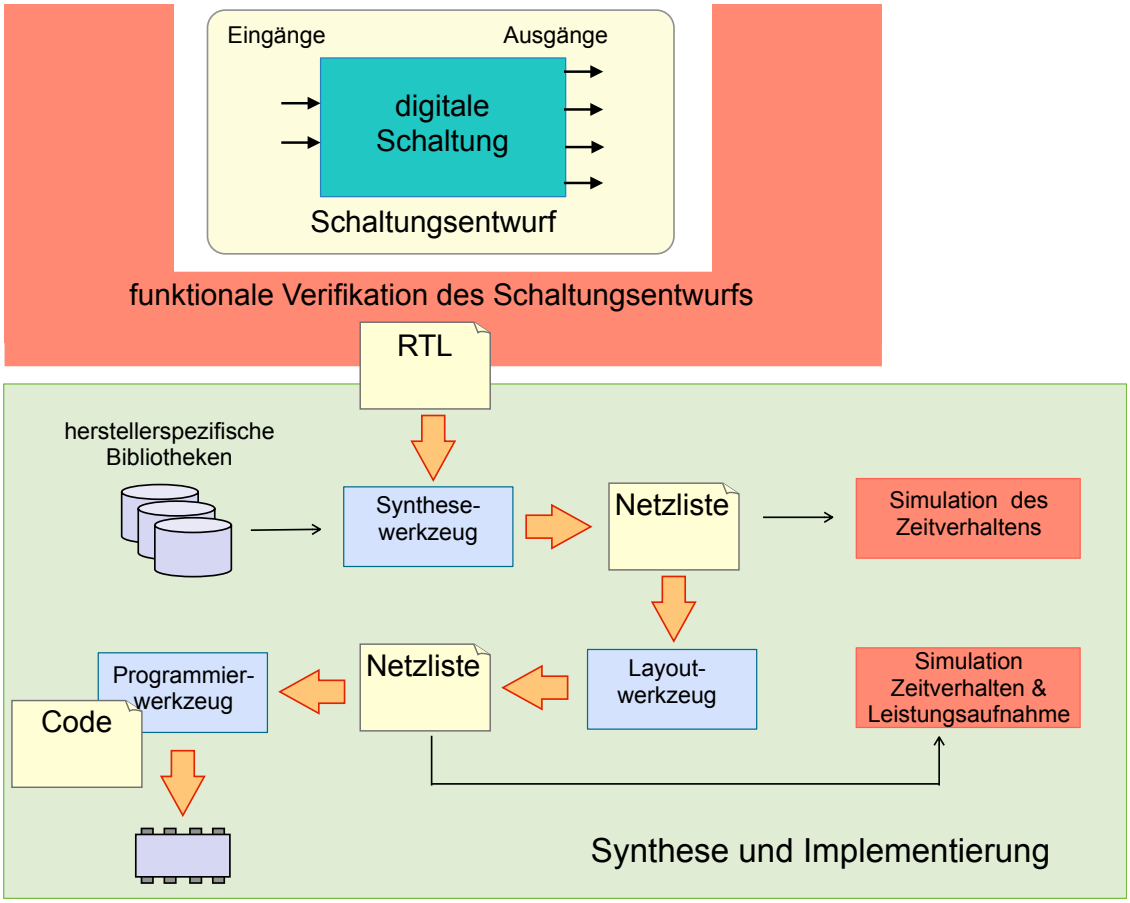

Bild 1.5 Synthese und Implementierung des Schaltungsentwurfs

Aufgabe 1.18: Machen Sie sich mit den Synthesewerkzeugen vertraut. Verwenden Sie hierzu die vorgegebenen Muster als Dateivorlagen für den Umgang mit den Werkzeugen zur Synthese und Implementierung.

Aufgabe 1.19: Überprüfen Sie die Ergebnisse der Verifikation des Schaltungsentwurfs (Dokumentation zu Aufgabe 1.16). Synthetisieren und implementieren Sie die den Schaltungsentwurf und testen Sie seine Funktionen.

Aufgabe 1.20: Dokumentieren Sie die Ergebnisse. Legen Sie diese Dokumentation Ihrem Laborbericht bei.

2. Versuch 2 - freies Thema

Als weiterer Versuch ist Ihnen ein Thema freigestellt. Ausgehend von Versuch 1 (Funktionsgenerator), kämen beispielsweise folgende Erweiterungen in Frage: (1) eine Beschaltung zur Bedienung des Funktionsgenerators, (2) digitaler Modulator mit PSK/4-QAM, (3) digitaler Modulator mit 16-QAM, (4) Empfängerseite für digitale Modulation mit PSK und QAM, (5) digitale Filter. Ihr Thema kann aber auch in eine völlig andere Richtung gehen.

Die Art des Themas richtet sich nach Ihren eigenen Interessen, sowie nach den Möglichkeiten der verwendeten Entwicklungsumgebung und des Zielsystems. Beispielsweise können sich sich nach frei verfügbaren Bausteinen (IP-Cores umschauen), bzw. ein komplett eigenes Design realisieren.

Gehen Sie bitte wie folgt vor:

- Wenn möglich, wählen Sie sich einen Sparringspartner in Ihrem Kurs, mit dem Sie Ihr Thema gemeinsam realisieren. Auf diese Weise können Sie Arbeiten aufteilen und den Fortschritt und das weitere Vorgehen miteinander diskutieren. Da die Synthesewerkzeuge recht komplex sind, ist beispielsweise eine Arbeitsaufteilung in (1) Entwurf und Test der Schaltung, sowie (2) Synthese und Implementierung denkbar.
- Besprechen Sie Ihren Themenvorschlag mit Ihrem Betreuer. Bei dieser Gelegenheit ist auch zu klären, ob Sie zusätzliche Hardware benötigen und anschaffen möchten.
- Realisieren Sie Ihren Systementwurf in den in den folgenden Kapiteln angegebenen Schritten.
- Dokumentieren Sie die wichtigsten Ergebnisse in jedem der Schritte.
- Präsentieren Sie Ihren Systementwurf in der Abschlusspräsentation vor den anderen Teilnehmern des Kurses.

Das Testat für das Labor erhalten Sie nach erfolgreicher Abschlusspräsentation und Abgabe der Dokumentation bei Ihrem Betreuer.

2.1. Funktionale Beschreibung

Beschreiben Sie Ihr Vorhaben und das Funktionsprinzip. Als Muster für die Dokumentation können Sie die Pflichtaufgabe verwenden, bzw. auf Literatur oder Fundstellen im Web verweisen. Beschreiben Sie Ihre geplante Vorgehensweise und ggf. die Aufteilung der Aufgaben.

2.2. Schaltungsentwurf

Entwerfen Sie Ihre Schaltung mit VHDL. Dokumentieren Sie Ihre Ergebnisse.

2.3. Funktionale Verifikation

Schreiben Sie eine Testumgebung zu Ihrem Schaltungsentwurf und testen Sie die Schaltung. Dokumentieren Sie Ihre Ergebnisse.

2.4. Schaltungssynthese

Implementieren Sie Ihren Schaltungsentwurf auf der Ziel-Hardware mit Hilfe der Entwicklungsumgebung. Dokumentiren Sie Ihre Ergebnisse.

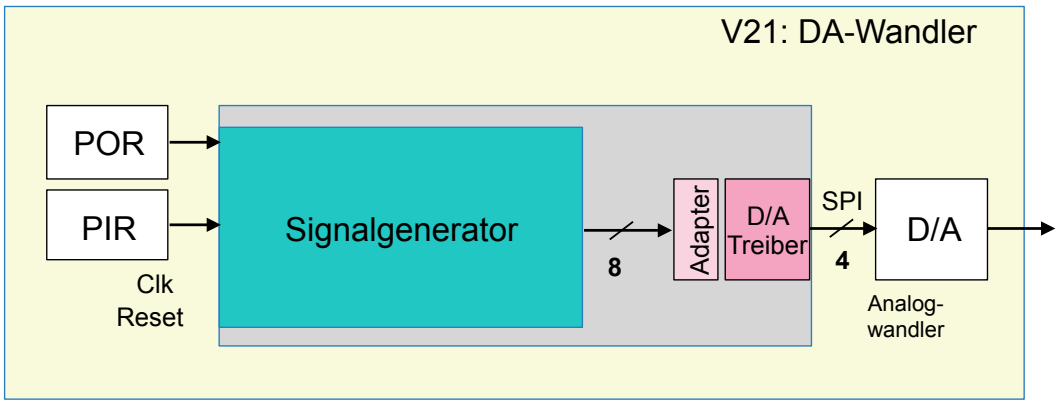

Bild 2.1 Ausgabe über D/A-Wandler

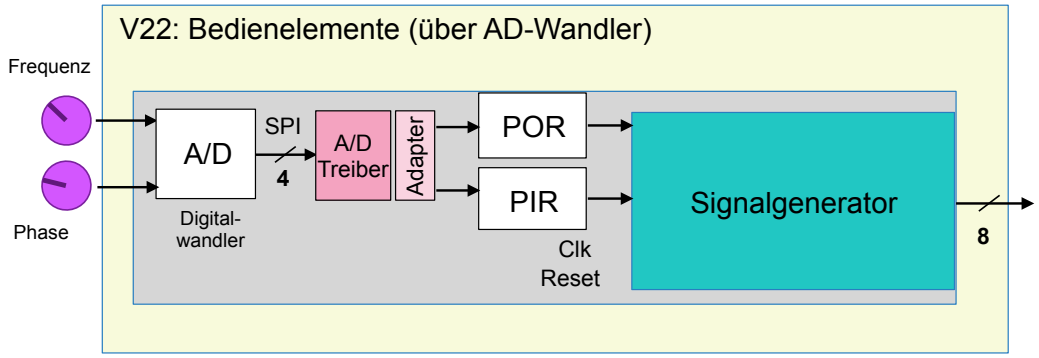

Bild 2.2 Bedienelemente (über A(D-Wandler)

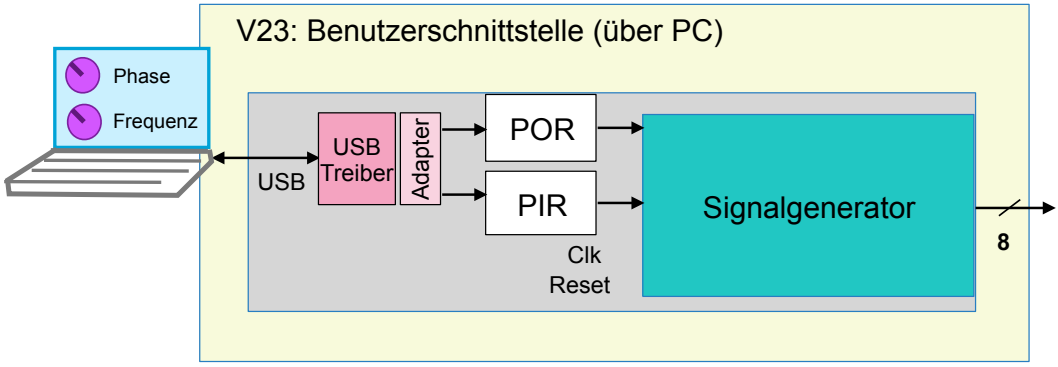

Bild 2.3 Benutzerschnittstelle (über PC)

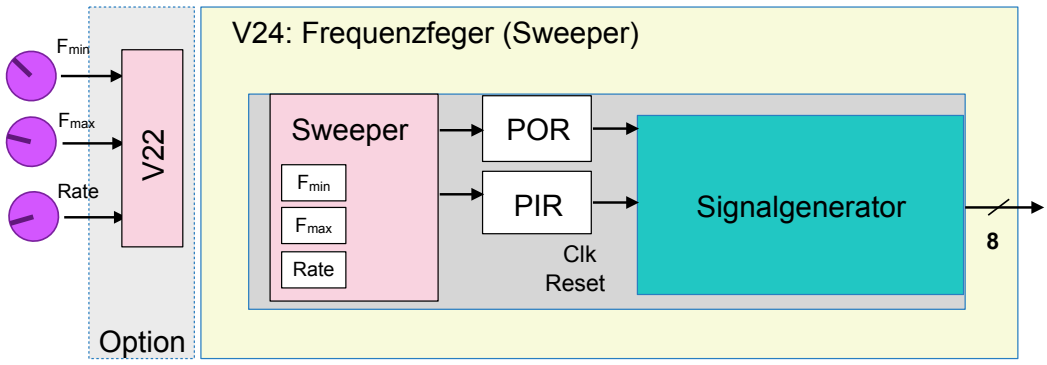

Bild 2.4 Frequenzfeger (Sweeper)

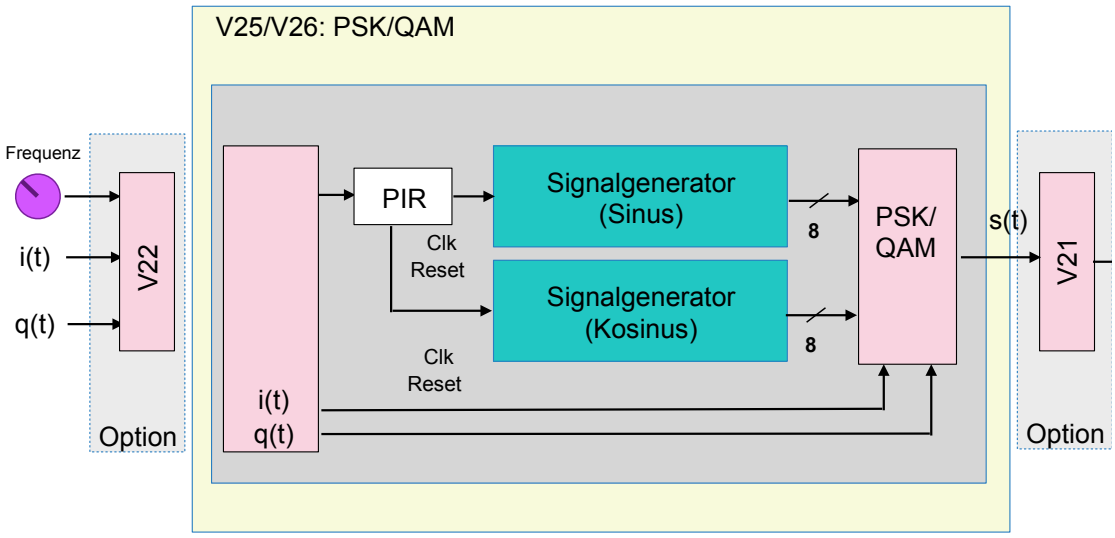

Bild 2.5 PSK/QAM

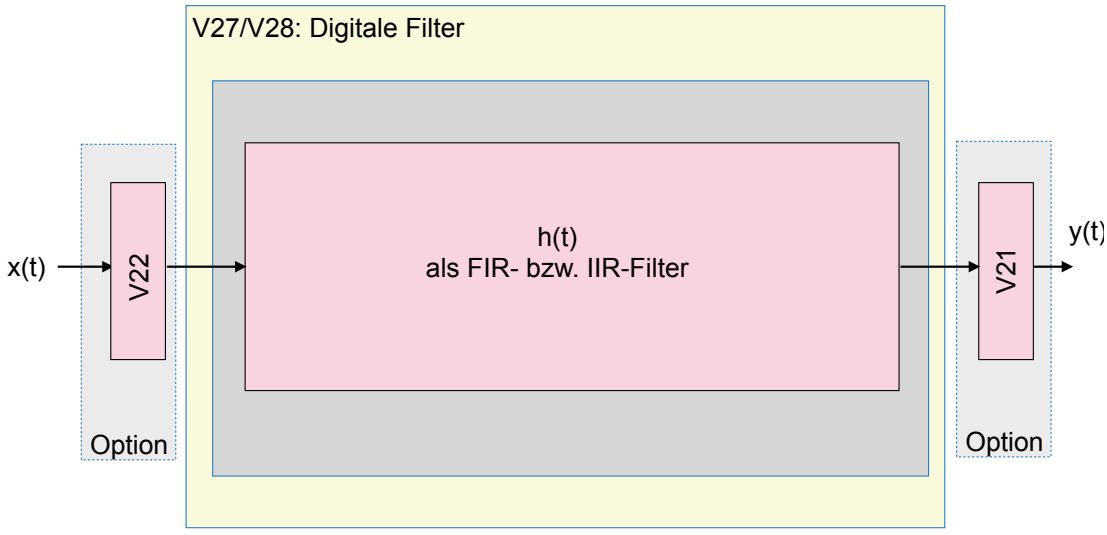

Bild 2.6 Digitale Filter

3. Zielsystem

Das Zielsystem hat folgendes Blockdiagramm (siehe auch Unterlagen zum Zielsystem):

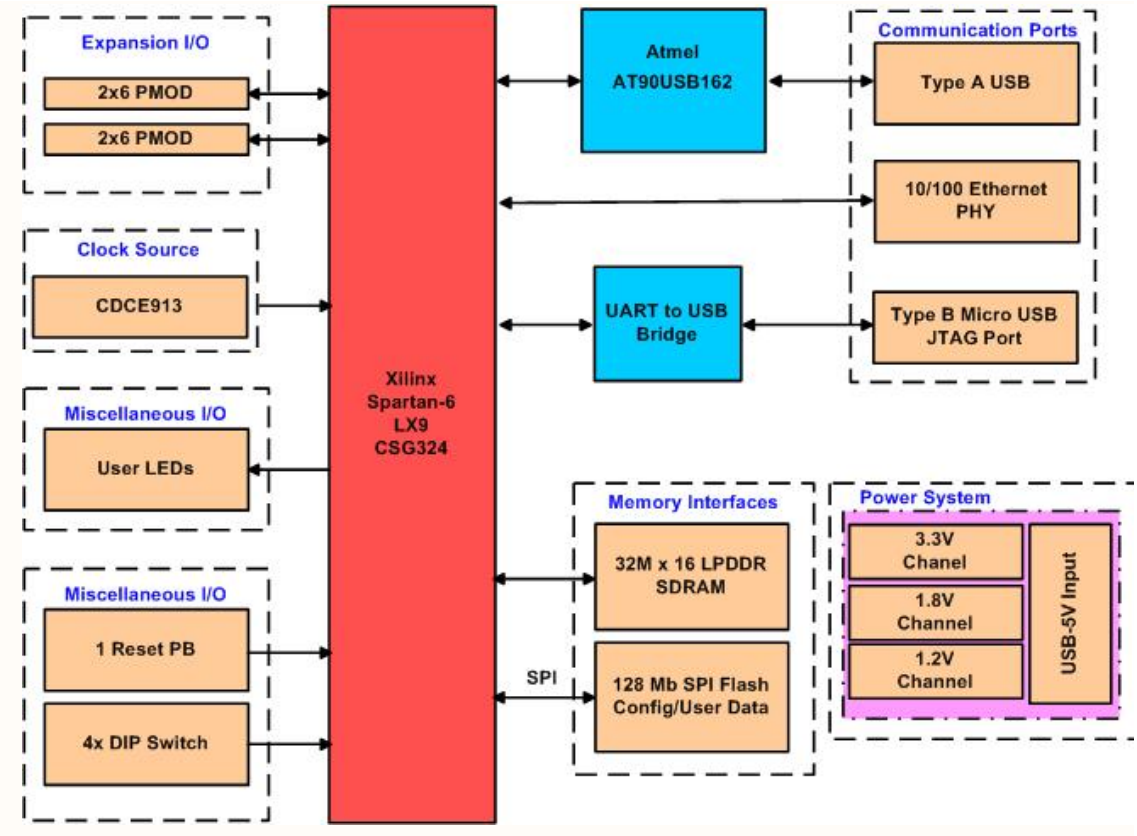

Bild 3.1 Blockschaltbild des Zielsystems

Ausser dem FPGA-Baustein (Xilinx Spartan-6) und der zugehörigen fest eingebauten Peripherie (Arbeitsspeicher, Flash-Speicher, LED-Anzeigen, Mikroschalter und Taktgenerator) sind folgende Schnittstellen zur Erweiterung bzw. zur Kommunikation vorhanden:

- Steckplätze für zwei I/O Module (Expansion I/O für Peripherie-Module PMOD)
- USB-Port (Typ A, auch als JTAG port nutzbar)
- Ethernet (10/100 Mbps) über RJ45
- USB-Port (Typ B).

Als Erweiterungsmodule lassen sich beispielsweise die für den Versuch benötigten DA-Wandler verwenden. Weitere Erweiterungsmöglichkeiten für Versuche finden sich im Literaturverzeichnis unter (3) und (4). Die USB-Ports dienen der Kommunikation mit einem Rechner. Hierunter fällt auch die Programmierschnittstelle für die Entwicklungswerkzeuge. Ebenfalls über einen der USB-Ports erfolgt die Stromversorgung. Für Netzwerkverbindungen steht ein Ethernet-Port zur Verfügung.

Abkürzungen

DDS	Direct Digital Synthesis
PIR	Phase Increment Register
POR	Phase Offset Register
PSK	Phase Shift Keying (Phasensprung-Verfahren zur Modulation)
QAM	Quadratur-Amplituden-Modulation

...

Literatur

(1) Entwurf digitaler Systeme, S. Rupp, Vorlesungsmanuskript, DHBW-Stuttgart

(2) Entwicklungswerkzeuge: Xilinx ISE Design Suite (inkl. WebPACK), wird inklusive Unterlagen passend zum Zielsystem zur Verfügung gestellt (Teil der LX9 Microboard Kits)

(3) Zielsystem: Xilinx Spartan-6 FPGA auf LX9 Microboard, siehe http://www.em.avnet.com/en-us/design/drc/Pages/Xilinx-Spartan-6-FPGA-LX9-MicroBoard.aspx

(4) Peripherie Module zum Zielsystem (LX9 Microboard), siehe z.B. http://www.digilentinc.com/Products/Catalog.cfm?NavPath=2,401&Cat=9&CFID=65844&CFTOKEN=41957016