Frequenzzähler bis 200 MHz

Dipl.-Ing. MARKUS LEMKE – DL1DSN

Zur Realisierung von Zählern sind diverse logische Baugruppen notwendig, die sich jedoch dank programmierbarer Logik-ICs in einem einzigen Schaltkreis unterbringen lassen.

Technische Daten

Zählfrequenz	bis 200 MHz
Vorteiler	nicht erforderlich
Anzeige	siebenstellig, 100 Hz Auflösung, gut ablesbare LEDs, Zeitmultiplexansteuerung
Zeitbasis	Quarzoszillator, nicht umschaltbar
Stromverbrauch	230 mA bei f_E = 200 MHz
Platine	einseitig für Zähler/Anzeige
Sonstiges	keine Periodendauermessung

Programmierbare Logikbausteine wie CPLDs (Complex Programmable Logic Device) und FPGAs (Field Programmable Gate Arrays) sind die Grundlage für schnelle und zugleich flexible Lösungen bei der Entwicklung komplexer digitaler Schaltungen. Die größten FPGAs weisen derzeit etwa 80 000 Logikelemente, fast 10 Mbit RAM, über 300 Hardwaremultiplizierer, 12 PLLs/DLLs und über 1000 frei programmierbare Ein-/Ausgangsanschlüsse auf.

Aber auch mit wesentlich kleineren Vertretern dieser digitalen ICs lassen sich interessante Schaltungen realisieren, für die früher eine Vielzahl von TTL- oder CMOS-Standard-ICs nötig waren. Der Beitrag zeigt, wie ein siebenstelliger, bis 200 MHz nutzbarer Frequenzzähler mit einigen Tricks in einem preiswerten CPLD-Baustein mit nur 72 Makrozellen realisiert werden kann. Die Programmierung erfolgt mit der Hardware-Beschreibungssprache VHDL und kostenlosen Entwicklungstools des IC-Herstellers.

Warum programmierbare Logik?

Wesentliche Vorteile ergeben sich beim Einsatz programmierbarer Logik-ICs:

– komplexe digitale Funktionen mit einzelnen oder wenigen ICs realisierbar,
– sehr hohe Geschwindigkeit bei der Signalverarbeitung,
– einfaches Layout, da Signale fast beliebig den Pins zugeordnet werden können,
– logische Funktionen jederzeit änderbar,
– Programmierung im System möglich,
– hinzufügen neuer Funktionen möglich, solange Ressourcen im IC frei sind.

Leider hat jede Medaille zwei Seiten. Die Programmierung komplexer FPGAs und CPLDs ist nur noch mit Hardware-Programmiersprachen sinnvoll, da die Eingabe von Schaltplänen fehleranfällig und ineffizient ist.

Auch lassen sich interne Signale nicht ohne Weiteres von außen messen. Eine Simulation der Funktion am PC ist, außer bei ganz simplen Funktionen, notwendig und das Herauslegen von Debug-Signalen zur Inbetriebnahme durchaus sinnvoll. Außerdem sind nur noch wenige dieser Bausteine in handhabbaren Gehäusen, z.B. PLCC, und mit herkömmlichen Betriebsspannungen von 5 oder 3,3 V verfügbar.

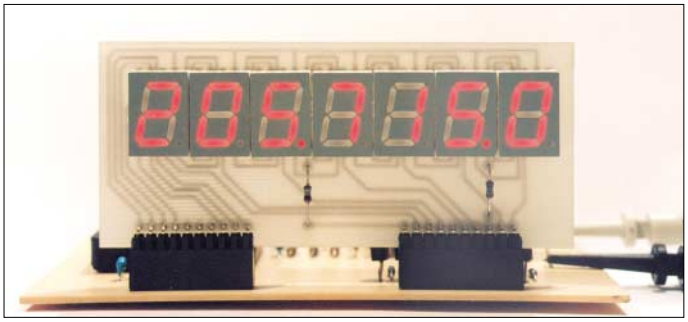

Bild 1: Die LED-Anzeigeplatine wird direkt auf die Leiterplatte des 200-MHz-Zählers gesteckt, wodurch Verdrahtungsarbeiten entfallen.

Foto: DL1DSN

Realisierung

Es wird das klassische Zählerprinzip mit Torschaltung, Zeitbasisteiler und Dezimalzählerkette verwendet, Bild 2. Der Multiplexbetrieb der Anzeige erleichtert das Layout, da so viel weniger Leitungen vom Schaltkreis zur Anzeige zu führen sind. Ich habe mich für ein leicht beschaffbares und preiswertes CPLD von Xilinx mit 5 V Betriebsspannung entschieden. Bevor wir uns einzelne Funktionsblöcke genauer anschauen, soll man zunächst das CPLD an sich näher betrachten.

Ein CPLD enthält sowohl Flipflops als auch kombinatorische Logik. Durch Programmierung lassen sich diese Elemente so verbinden, dass am Ende die gewünschte digitale Funktion realisiert wird. Die Informationen über die internen Verbindungen sind meist in einem Flash gespeichert und bleiben beim Abschalten der Betriebsspannung erhalten. Dadurch ist nach dem Einschalten kein erneutes Laden erforderlich.

Ebenso wird von den meisten CPLDs die Programmierung im System über ein genormtes JTAG-Interface unterstützt. Änderungen an der Funktion lassen sich also direkt auf dem Board mit Hilfe eines Programmierkabels vornehmen.

Welche Möglichkeiten ein programmierbarer Logikbaustein im Detail bietet, ist dem jeweiligen Datenblatt des Herstellers zu entnehmen – diese Darstellung würde den Umfang des Beitrags übersteigen. Die entsprechenden PDF-Dateien von Xilinx findet man unter [1] → Products and Services → Product Datasheets → XC9500.

Wie in [2] beschrieben ist, beinhalten die CPLDs Funktionsblöcke und I/O-Blöcke, die über eine Schaltmatrix verbunden sind. Die Anzahl dieser Blöcke ist vom CPLD-Typ und dessen Gehäuse abhängig.

Jeder Funktionsblock besteht aus 18 Makrozellen, die jeweils ein Flipflop enthalten. Die Steuersignale für das Flipflop lassen sich über eine AND-/OR-Logik aus den Signalen der Schaltmatrix ableiten. Der Ausgang des Flipflops führt zu einem I/O-Block, und kann damit ein Ausgangssignal treiben, sowie zur Schaltmatrix.

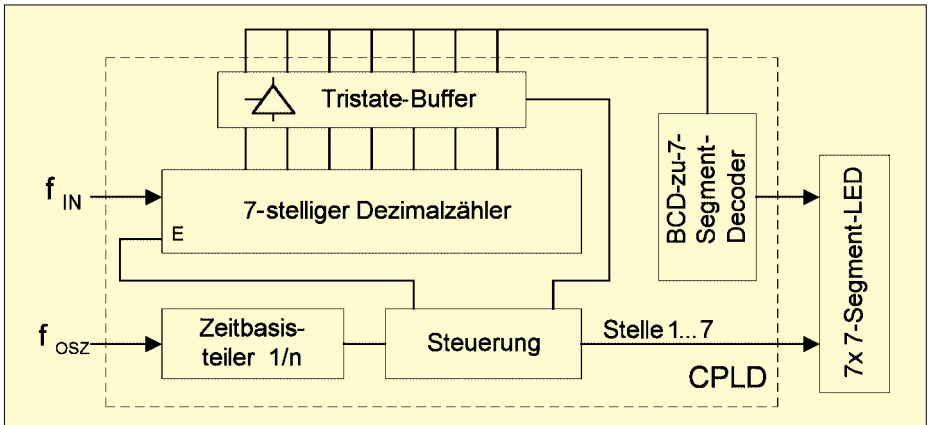

Bild 2: Blockschaltbild des 200-MHz-Frequenzzählers mit LED-Multiplexanzeige

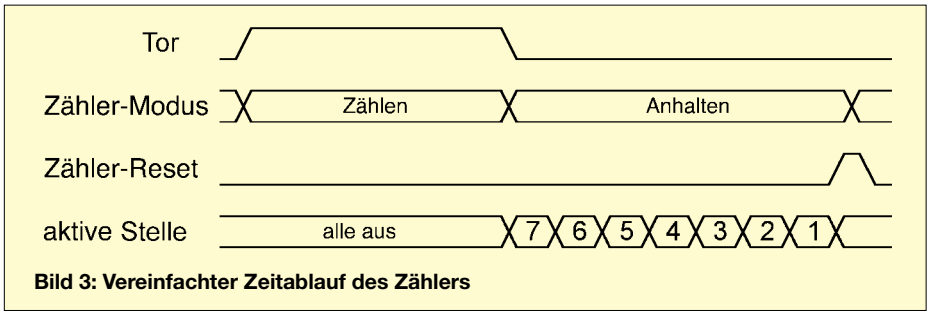

Bild 3: Vereinfachter Zeitablauf des Zählers

Die CPLD-Pins sind größtenteils frei programmierbar. Sie können sowohl als Eingang, Ausgang oder bidirektional konfiguriert werden. Die 9500er CPLDs von Xilinx sind für Ausgangsströme von maximal 24 mA ausgelegt und können daher z.B. LEDs direkt treiben.
Einige Ein- und Ausgänge sind für spezielle Signale vorgesehen, z.B. für globale Taktsignale. Dem Pin GCK (Global Clock) ist ein Buffer zugeordnet, der einen synchronen Takt für alle Flipflops im CPLD bereitstellen kann. Die vier JTAG-Pins TDI, TCK, TMS und TDO dienen der Programmierung. Zusätzlich sind mehrere Anschlüsse für die Betriebsspannung, Ein-/Ausgangsspannung von 5 V bzw. 3,3 V und Masse vorhanden.

■ Auswahl des CPLDs

Für einen siebenstelligen Dezimalzähler benötigt man $7 \cdot 4 = 28$ Flipflops. Die 100-MHz-Stelle unseres Zählers soll höchstens eine Zwei anzeigen können – es genügen also insgesamt 26 Flipflops. Der Speicher für das Zählerergebnis benötigt nochmals 26. Bei einer Quarzzeitbasis von 8,192 MHz und 10 ms Torzeit wird eine Teilerkette mit 17 Bit benötigt. Damit sind ohne Ablaufsteuerung, Anzeigen-Multiplexer und BCD-zu-7-Segment-Decoder schon 69 Makrozellen nötig.
Das größte CPLD der 9500er Serie im PLCC-Gehäuse besitzt 108 Makrozellen, ist aber nicht so leicht beschaffbar. Deshalb habe ich versucht, die gesamte Logik mit einigen Tricks in einem XC9572-PC84-15 unterzubringen.
Die Bezeichnung hat folgende Bedeutung: XC9572 steht für ein Xilinx CPLD der 5-V-Serie mit 72 Makrozellen. PC84 bezieht sich auf das 84-polige PLCC-Gehäuse. Die 15 gibt schließlich die garantierte, maximale Pin-zu-Pin-Laufzeit an, also 15 ns. Doch wie lässt sich der Hardwareaufwand so weit reduzieren, dass der gesamte Frequenzzähler in einen XC9572 passt?

■ Design-Optimierung

Die Auflösung von 100 Hz bedingt eine Torzeit von 10 ms. Bei dieser kurzen Zeit kann der Zwischenspeicher für das Zählergebnis eingespart werden. Die Ausgänge der sieben Zähler werden nach Ablauf der Torzeit über einen Multiplexer direkt zur Anzeige des Ergebnisses verwendet. Eine Wiederholfrequenz von 50 Hz sollte für das Auge gerade noch akzeptabel sein.
Während des Zählvorgangs muss die Anzeige selbstverständlich dunkel getastet bleiben. Die restlichen 10 ms bleiben für die Darstellung des siebenstelligen Zählergebnisses übrig, siehe Bild 3.
Um alle sieben Stellen des Zählers nacheinander anzuzeigen, ist ein 4 Bit breiter 7-zu-1-Multiplexer notwendig. Dieser lässt sich auch ohne die Kombinatorik der Funktionsblöcke realisieren, und zwar mit I/O-Blöcken. In jedem dieser Blöcke gibt es einen Tristate-Buffer. Verbindet man die Ausgänge mehrerer Tristate-Buffer, d.h. hier die CPLD-Pins, dann erhält man einen Multiplexer. Die Steuerung muss nur garantieren, dass jeweils nur ein Tristate-Buffer treibt und alle übrigen hochohmig sind.
Das PLCC84-Gehäuse bietet genügend freie Anschlüsse für unseren Multiplexer. Das Ergebnis muss über einen weiteren Pin in den Chip zurückgeführt werden. Insgesamt benötigen wir 30 Pins. Damit beim Layout nicht das absolute Chaos entsteht, wählt man selbstverständlich für die zu verbindenden Signale benachbarte Pins. Nur dadurch ist überhaupt ein einseitiges Layout möglich.
Um die Lesbarkeit der Anzeige zu verbessern, sollen führende Nullen unterdrückt werden. Das ist mit minimalem Aufwand möglich, da ohnehin alle Zählerstellen nacheinander den Multiplexer durchlaufen. Wenn man bei der 100-MHz-Stelle beginnt, dann muss nur ein Flag gesetzt werden, sobald der Multiplexerausgang einen Wert größer als Null hat. Bis dahin bleibt die Anzeige dunkel. Ab der 1-kHz-Stelle wird das Flag immer gesetzt, damit ohne Eingangssignal 0.0 angezeigt wird.
Die höchsten Anforderungen in puncto Geschwindigkeit werden an den ersten Zähler gestellt, die 100-Hz-Stelle. Für optimales Timing müssen alle vier Flipflops dieses Zählers im selben Funktionsblock liegen – es gibt vier solcher Blöcke im XC9572. Nur dann können die schnellen lokalen Rückführungen genutzt werden. Ansonsten erfolgt das Routing über die langsamere Schaltmatrix.
Leider passt nicht der gesamte Zähler in einen Funktionsblock, sondern maximal 18 Bit. Es müssen ja auch nicht alle Flipflops mit der höchstmöglichen Frequenz laufen. Wir teilen den Zähler deshalb in drei Stufen ein. Das höchstwertige Bit der ersten Zählerstelle ist gleichzeitig das Taktsignal für die zweite Zählerstufe. Das höchstwertige Bit der zweiten Zählerstufe ist wiederum der Takteingang für die restlichen fünf Zählerstellen, die als Synchronzähler arbeiten. Damit werden bei 5 V Betriebsspannung und normaler Umgebungstemperatur über 200 MHz erreicht.
Es soll nicht verschwiegen werden, dass die CPLDs der XC9500er Serie – obwohl in einem CMOS-Prozess gefertigt – reichlich Strom konsumieren. Würden alle 72 Makrozellen mit 100 MHz laufen, käme man auf rund 160 mA allein für das CPLD. Es gibt aber einen Stromsparmodus, den man für langsame Makrozellen unbedingt nutzen sollte: *Low Power Mode* statt *High Performance Mode*. Letzterer wird beim Frequenzzähler nur für die ersten beiden Zählerstufen und die Torsteuerung benötigt. Das spart rund 40 mA.
Die Einstellungen für den Stromsparmodus erfolgen, wie auch die Anschlussbelegung, im *User Constraints File counter.ucf*, siehe [4].

Kasten 1: Tristate-Buffer

```
entity tribuf is
   port (
      din: in std_logic;      -- Daten-Eingang
      t: in std_logic;        -- Tristate-Eingang
      dout: out std_logic  -- Daten-Ausgang
   );
end;

architecture behave of tribuf is
begin
   dout <= not din when t='0' else 'Z';
end;
```

Kasten 2: Zeitbasisteiler

```
architecture behave of counter is
…
   constant ZEITBASIS : integer := 3999;
   signal basis_cnt_q : integer range 0 to
   ZEITBASIS;
…
begin
…
   process(rst_n, clk)
   begin
      if rst_n='0' then
         basis_cnt_q <= ZEITBASIS;
      elsif rising_edge(clk) then
         if basis_cnt_q = 0 then
            basis_cnt_q <= ZEITBASIS;
         else
            basis_cnt_q <= basis_cnt_q -1;
         end if;
      end if;
   end process;
end;
```

Schaltung

Bild 4 zeigt die Schaltung des Frequenzzählers. Für das CPLD wird eine Betriebsspannung von 5 V benötigt. Ein Pierce-Oszillator erzeugt den Takt für die Zeitbasis. Ich habe 4 MHz gewählt – prinzipiell sind Werte zwischen 1 und 8,192 MHz möglich, wenn die Konstante *Zeitbasis* in der Datei *counter.vhd* entsprechend geändert und ein neues Programmierfile erzeugt wird.

Ein Vorverstärker ist nicht vorhanden. Wer ihn benötigt, findet dafür ausreichend Beispiele in der Literatur, z.B. [5]. Für niedrige Frequenzen sollte auf jeden Fall ein Schmitt-Trigger, z.B. der 74HC14, vorgeschaltet werden. Für VT8 und VT9 sind auch andere Typen verwendbar.

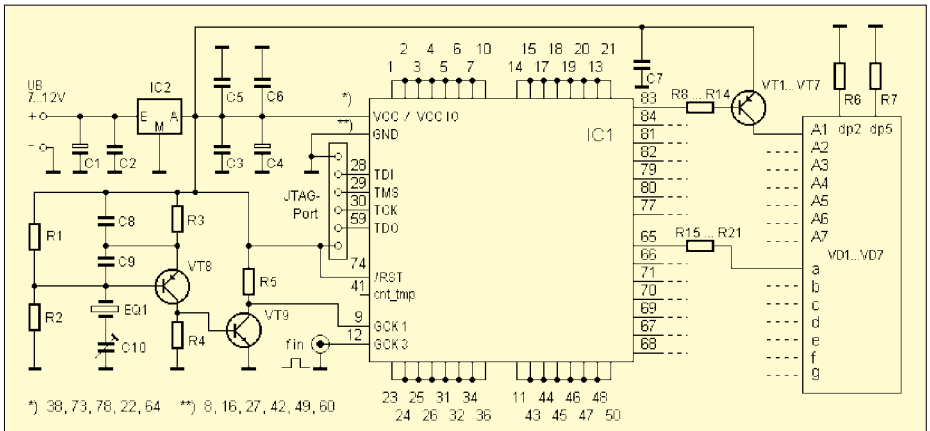

Bild 4: Stromlaufplan des 200-MHz-Frequenzzählers mit LED-Anzeige

Hardware-Beschreibung mit VHDL

VHDL ist eine Hardware-Beschreibungssprache für digitale Schaltungen, mit der Designs und Simulationen sehr effizient möglich sind. Es gibt zwar reichlich Literatur über VHDL, aber eine kurze, praktische Einführung ist schwer zu finden. Meist wird der Anfänger mit einer Unmenge an Theorie und Konstrukten überhäuft, die für einen Einstieg völlig unnötig sind. Einige ausgewählte Links finden sich in [3].

Für den reinen Elektroniker dürfte der Einstieg ebenso ein Umdenken erfordern, wie z.B. für einen Programmierer von Mikrocontrollern. Es ist sicher gewöhnungsbedürftig, Hardware mittels Software zu beschreiben. Man kann aber schon mit wenigen Grundelementen in VHDL erste Erfolge erzielen.

Ein VHDL-Modul besteht prinzipiell aus einer Schnittstellenbeschreibung (*Entity*) und einer Funktionsbeschreibung (*Architecture*). In der *Entity* werden die Ein- und Ausgänge, also das Interface der Schaltung, beschrieben. Es gibt verschiedene Signaltypen, z.B. für einzelne Leitungen oder Busse. In der *Architecture* wird das Verhalten der Schaltung beschrieben, also wann ein Flipflop gesetzt werden oder ein bestimmtes Bitmuster am Ausgang erscheinen soll.

Das Beispiel im Kasten 1 zeigt einen invertierenden Tristate-Buffer in VHDL. Der Ausgang *dout* bekommt den invertierten Wert von *din* zugewiesen, solange t = 0 ist. Bei t = 1 wird *dout* Z zugewiesen, was den Compiler veranlasst, einen Tristate-Buffer zu implementieren. Diese Beispielanweisung ist reine Kombinatorik, benötigt also keine Register. Wie lässt sich nun ein Flipflop oder ein ganzer Zähler realisieren? Dafür wird ein neues VHDL-Element benötigt, und zwar ein *Prozess*. In einem *Prozess* können Register und Kombinatorik beschrieben werden. Als Beispiel dient der Zeitbasisteiler des Frequenzzählers, Kasten 2.

Damit haben wir einen kompletten Teiler 1:4000 recht elegant beschrieben. Soll eine andere Quarzfrequenz zum Einsatz kommen, muss nur die Konstante *Zeitbasis* geändert werden, nicht jedoch der Zähler selbst. Der Compiler ermittelt selbst die Anzahl benötigter Flipflops, darum müssen wir uns nicht kümmern. Die Anweisung *if rising_edge(clk)* sagt dem Compiler, dass hier Register implementiert werden müssen, die auf die steigende Flanke von *clk* reagieren.

Der Zähler *basis_cnt_q* wird mit jedem Takt dekrementiert. Wenn er den Wert 0 erreicht hat, wird mit der nächsten Taktflanke wieder der Startwert geladen. Das Signal *rst_n* ist ein asynchroner Zähler-Reset und für den Zeitbasisteiler eigentlich nicht zwingend erforderlich. Er ermöglicht aber z.B. bei der Simulation einen definierten Startwert für den Zähler.

In [3] sind weitere VHDL-Beispiele zu finden. Mit Hilfe der zahlreichen Kommentare und zusätzlicher Literatur sollte ein VHDL-Einstieg keine allzu großen Probleme bereiten.

Stückliste

Bauteil	Wert
C1	220 µF
C2, C3, C5…C7	100 nF
C4	470 µF
C8	180 pF
C9	390 pF
C10	10...40 pF, Trimmer
R1	33 kΩ
R2	22 kΩ
R3	1,8 kΩ
R4, R8...R14	680 Ω
R5	1,2 kΩ
R6, R7	180 Ω
R15...R21	100 Ω
VD1…VD7	SA 52-11, 7-Segment-LED, rot
VT1…V.T7	BC327-40
VT8	BC559C
VT9	2SC2026
EQ1	z.B. 4 MHz, siehe Text
IC1	XC9572-PC84-15
IC2	µA7805, TO220-Gehäuse

Wie kommt die Schaltung ins CPLD?

Die CPLD-Programmierung kann in vier Stufen unterteilt werden. Zuerst erfolgt die Eingabe der Schaltung, z.B. in Form eines Schaltplans. Effektiver ist die Verwendung einer Hardware-Beschreibungssprache. Im zweiten Schritt wird daraus vom Compiler eine Netzliste erstellt. Solche Compiler können sehr teuer sein, werden aber z.B. auch von den IC-Herstellern in einfachen Versionen kostenlos angeboten. Danach sorgt ein so genannter Fitter für die Implementierung der Funktion in die Makrozellen des CPLDs. Schließlich benötigt man noch eine Download-Software nebst entsprechendem Kabel, um z.B. über ein JTAG-Interface das IC zu programmieren.

Mit Hilfe der Xilinx-Software *WebFitter* kann man aus dem VHDL-Sourcecode in *counter.vhd* und dem *User Constraints File counter.ucf* ein JEDEC-Programmierfile erstellen: *counter.jed*. *WebFitter* lässt sich über einen Web-Browser aufrufen.

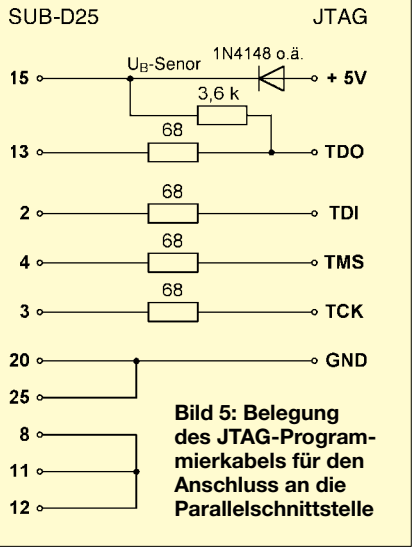

Bild 5: Belegung des JTAG-Programmierkabels für den Anschluss an die Parallelschnittstelle

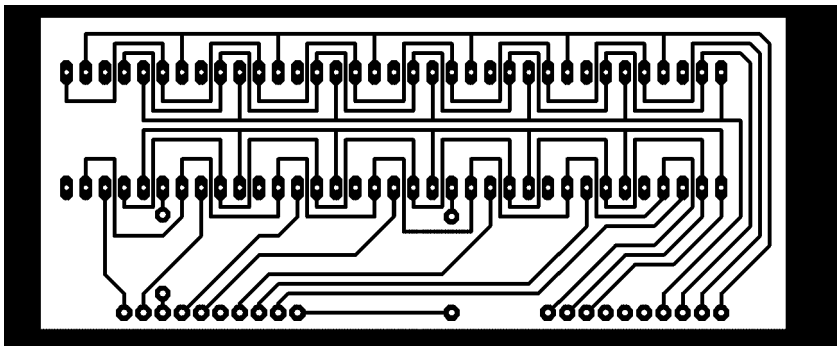

Bild 6: Layout der Anzeigeplatine

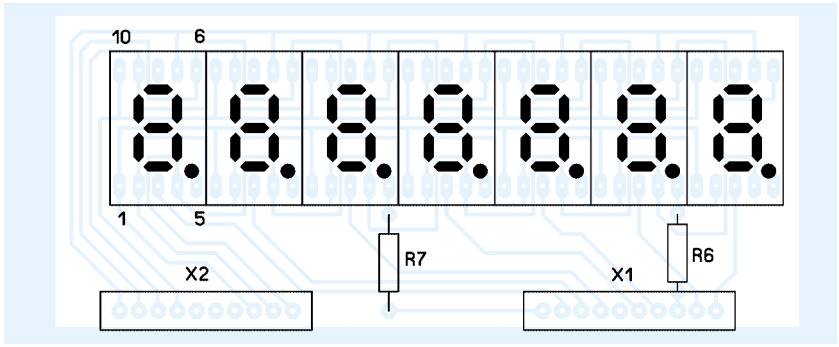

Bild 7: Bestückung der Anzeigeplatine

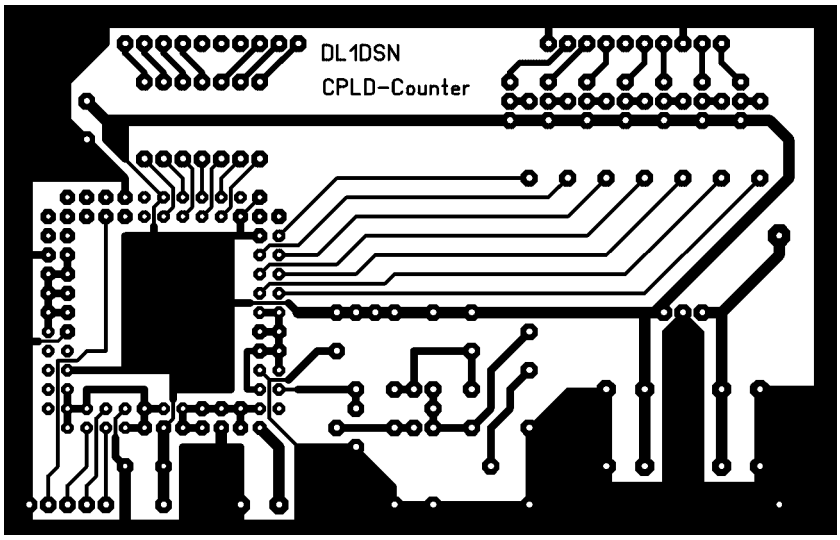

Bild 8: Layout der Zählerplatine

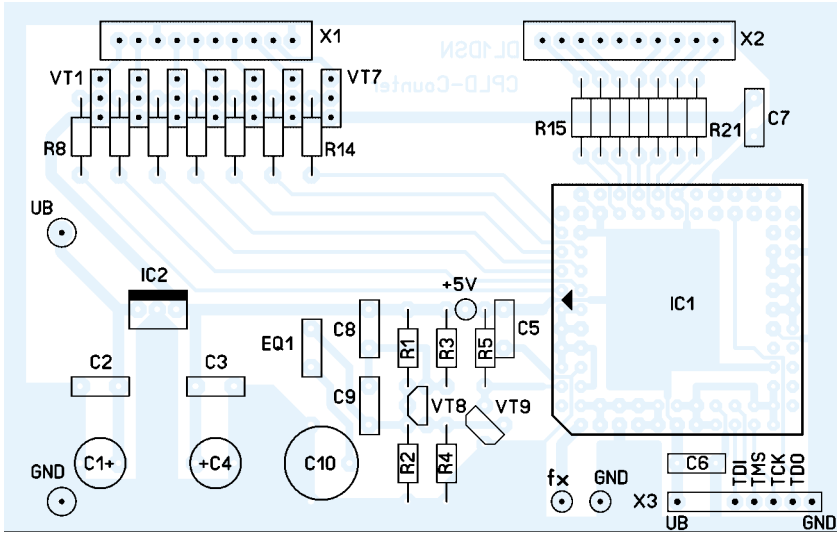

Bild 9: Bestückung der Zählerplatine

Wer die entsprechenden Downloadmöglichkeiten hat, kann auch das komplette *ISE WebPack* herunterladen und das Design selbst compilieren. Das fertige Programmierfile *counter.jed* kann jedoch auch von [4] geladen werden. Jetzt fehlt noch ein JTAG-Programmier-Tool nebst Downloadkabel.

Um das CPLD mit der Datei *counter.jed* programmieren zu können, ist eine entsprechende Software nötig. Ich empfehle die Verwendung des *Xilinx-JTAG-Programmers*, der Bestandteil des *WebPACK* und unter [1] → Support → Software → WebPACK ISE → Download ISE WEBPACK verfügbar ist. Nach dem Registrieren kann man sich die verschiedenen Programmpakete herunterladen.

Da das aktuelle JTAG-Tool vom Umfang her eher etwas für DSL-Nutzer ist, wählt man besser unter *Related Features* den Punkt *Earlier ISE WebPACK releases*. Die älteste Version 3.3WP8.1 ist völlig ausreichend. Wir müssen nun das Programm *9500 Family Programmer* mit 6,3 MB Umfang herunterladen. Es war leider kein kleineres Programm im Internet zu finden, aber man bekommt dafür immerhin ein Windowsprogramm für alle CPLDs der 9500er Serie.

Die Installation mit dem *Install Shield Wizard* geht problemlos. Anschließend setzt man noch die Variable *Xilinx* auf *root* und fügt zu *Path root\bin\nt* hinzu, wobei *root* für den *WebPack*-Installationspfad steht.

Ein passendes Downloadkabel für die Parallelschnittstelle des PCs ist schnell hergestellt, Bild 5. Die Widerstände sollten auf keinen Fall größer gewählt werden. Alle Bauteile passen in das Gehäuse eines Sub-D25-Steckers.

Nach dem Start des *JTAG-Programmers* wählt man *Output → Cable Auto Connect*. Anschließend lädt man mit Edit → Add Device die Datei *counter.jed*. Unter *Operations → Chain Operations* wird *counter.jed* angeklickt und schließlich unter *Selected Device Operation* der letzte Punkt *Program* ausgewählt.

Mit *Execute* wird der Programmiervorgang gestartet, der einige Sekunden dauert – vorher Betriebsspannung einschalten.

Der Zähler sollte jetzt 0.0 anzeigen und ist damit betriebsbereit. Das CPLD kann jederzeit neu programmiert werden, falls eine Funktionsänderung gewünscht wird.

dl1dsn@gmx.de

Literatur

[1] Xilinx: Homepage. *www.xilinx.com*
[2] Xilinx: XC9572 In-System Programmable CPLD. *www.xilinx.com/bvdocs/publications/9572.pdf*
[3] Lemke, M., DL1DSN: VHDL-Beispiele und Links. *www.dl1dsn.de/projects/counter/vhdl_bsp.htm*
[4] Lemke, M., DL1DSN: 200-MHz-Frequenzzähler. *www.dl1dsn.de/projects/counter/counter.htm*
[5] Zech, R., DG0VE: Dezimalzähler – nutzbar bis 12,5 GHz. FUNKAMATEUR 53 (2004) H. 4, S. 366–367