

Versuch 5: Automaten

Einführung

Im Versuch 5 werden mit der PLD-Entwicklungssoftware Altera MAX+plusII auf dem PLD-Board SB2 Automaten realisiert. Der Versuch besteht aus zwei Teilen die in beliebiger Reihenfolge gelöst werden können:

1: 3-Phasen Takt Generator 2: Audio Sequence Player

Für den Teil 2 ist es unbedingt notwendig, einen eigenen Kopf- oder Ohrhörer mit 3.5mm stereo Klinckenstecker (der allgemein übliche Anschluss) mitzubringen.

Das Lösungsblatt zum Versuch erhalten Sie am Praktikumsnachmittag.

Vorbereitung am Praktikumsnachmittag:

Im Praktikumsverzeichnis C:\DigiPrakt ein neues Arbeitsverzeichnis V5 anlegen (falls ein solches von Ihren Vorgängern her bereits existiert, dessen gesamten Inhalt löschen). In dieses die Versuchsfiles als **V5_FilePack.exe** von der DigiPrakt Seite herunterladen und entpacken. Falls Sie Ihren eigenen 7-Segment Decoder (aus Versuch 2) oder Addierer (aus Versuch 3) benützen möchten, so kopieren Sie erst jetzt alle Ihre entsprechenden gdf, tdf und sym Files ins Versuchsdirectory V5 .

Endliche synchrone Automaten

Die meisten sequentiellen Schaltungen (ausser Speichern) werden als endliche synchrone Automaten (Finite State Machines, weiter nur noch als Automaten bezeichnet) realisiert. Die Entwurfsmethoden sind im Prinzip einfach und ausgereift. Die Grundlagen sollen hier kurz repetiert werden.

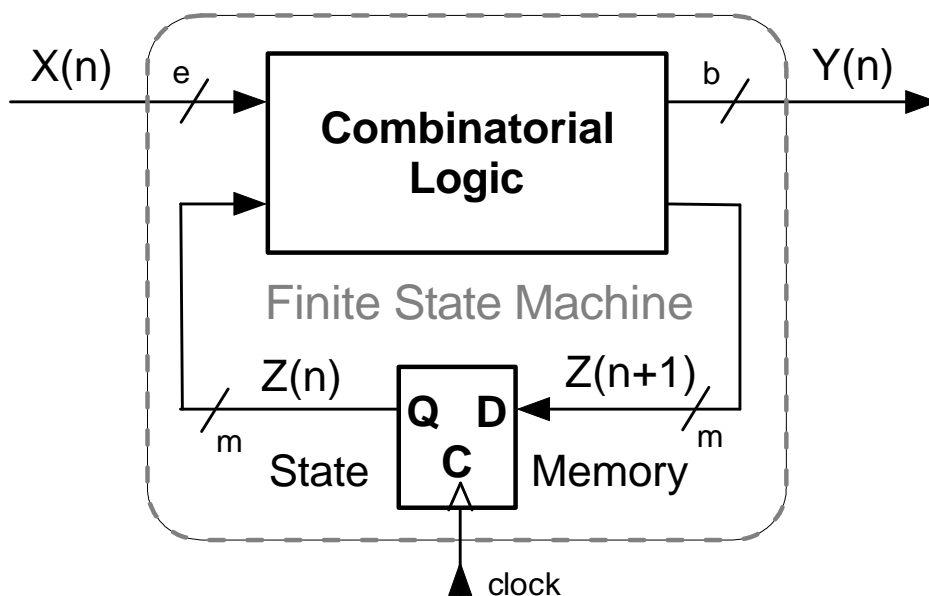


Bild 1.1: Grundsätzlicher Aufbau eines synchronen endlichen Automaten

Das Bild 1.1 zeigt den grundsätzlichen Aufbau eines Automaten. Er besteht aus einer kombinatorischen Logik und einem Speicher (D-Flip-Flops, auch als Register bezeichnet). Im Speicher wird der Zustand (State) des Automaten gespeichert, man spricht deshalb vom Zustandsspeicher (State Memory). Da die Zahl der Flip-Flops in jedem realen Automaten endlich sein muss, ist auch die Anzahl der Zustände die der Automat einnehmen kann endlich, deshalb die Bezeichnung **endlicher Automat**.

Der Automat verfügt über e Eingänge $X = x_1, x_2, \dots, x_e$ und b Ausgänge $Y = y_1, y_2, \dots, y_b$ (Die mit Grossbuchstaben bezeichneten Signale sind Bit-Vektoren, das heisst, mehrere zusammengefasste binäre digitale Signale) sowie einen Takteingang clock. Die Takteingänge aller Flip-Flops sind mit diesem Takt verbunden. Der Inhalt des Zustandsspeichers und damit der Zustand ändern synchron auf jede aktive Taktflanke, deshalb die Bezeichnung **synchroner Automat**. (Das Problem asynchroner Eingangssignale wird hier stillschweigend umgangen, da es in unseren Anwendungen vernachlässigt werden kann.)

Im Taktintervall n wird in der kombinatorischen Logik aus dem aktuellen Zustand $Z(n)$ und den aktuellen Eingängen $X(n)$ der Code des nächsten Zustandes $Z(n+1)$ (den der Automat im Taktintervall $n+1$ einnehmen soll) gebildet.

Zur Bildung der Ausgangssignale im Taktintervall n bestehen zwei verschiedene Konzepte, die nach ihren Vätern Moore und Mealy bezeichnet werden.

Im einfacheren Fall des **Moore-Automaten** werden die Ausgänge in der kombinatorischen Logik nur aus dem aktuellen Zustand $Z(n)$ bestimmt.

Im allgemeineren Fall des **Mealy-Automaten** werden die Ausgänge in der kombinatorischen Logik aus dem aktuellen Zustand $Z(n)$ und zusätzlich den aktuellen Eingängen $X(n)$ bestimmt.

Mealy Automaten kommen zur Lösung des gleichen Problems normalerweise mit weniger Zuständen aus, dafür ist der Entwurf weniger einfach.

Das Verhalten eines endlichen synchronen Automaten wird vollständig durch die folgenden Gleichungen beschrieben (n = Taktintervall Nummer):

$$\text{Zustand:} \quad Z(n) := Z(n+1) \quad Z(n+1) = g(X(n), Z(n))$$

$$\text{Ausgänge:} \quad \text{Moore} \quad Y(n) = f(Z(n)) \quad \text{Mealy} \quad Y(n) = f(X(n), Z(n))$$

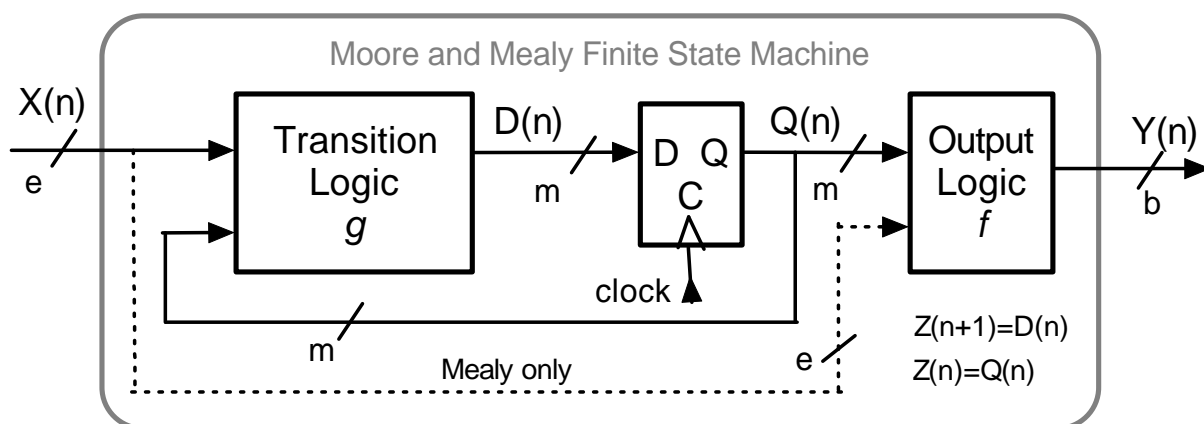


Bild 1.2: Aufgeteilte kombinatorische Logik, Moore – Mealy Modell

Das Bild 1.2 zeigt den Aufbau des Automaten etwas umgeformt mit aufgeteilter kombinatorischer Logik. Die **Transitionlogik** realisiert die Übergangsfunktion f (siehe vorausgehende Gleichungen), die **Outputlogik** die Ausgangsfunktion g unter Ausschluss (Moore) bzw. Einschluss (Mealy) der Eingänge X . An Stelle der Zustandsvektoren $Z(n+1)$ und $Z(n)$ sind im Bild 1.2 die Signalbezeichnungen des Zustandsspeichers $D(n)$ und $Q(n)$ eingezeichnet. Der Spezialfall des Moor-Automaten ohne Ausgangslogik wird als **Medwedjew Automat** bezeichnet und benötigt meistens mehr als die minimale Anzahl Flip-Flops.

Zum Entwurf eines Automaten empfiehlt es sich, in folgenden Schritten vorzugehen:

1. Aufgabe studieren und verstehen (meistens der schwierigste Teil).
2. Zustände festlegen (den verschiedenen Phasen des Automaten zuordnen, Anzahl bestimmen, meistens der zweitschwierigste Teil).
3. Anzahl Flip-Flops und die Methode der Zustandskodierung festlegen (die minimale Anzahl führt nicht immer zur einfachsten oder besten Realisierung).
4. Zustandsdiagramm aufzeichnen, zuerst die Hauptfunktionen, danach die Nebenfunktionen zufügen.
5. Jedem Zustand einen Zustandscode zuordnen (eine geschickte Zuordnung kann die kombinatorische Logik vereinfachen).
6. Wahrheitstabellen für die kombinatorische Logik aufstellen (sollte jetzt nur noch eine Fleissarbeit sein).
7. Implementation des Automaten (in unserem Fall mit MAX+plus kombinatorische Logik durch Tabellen in AHDL festlegen, alles mit dem Graphikeditor zusammenstellen).
8. Test und bis alles ok wieder von vorne beginnen.

1: 3-Phasen Takt Generator

Einführung

Als Beispiel eines einfachen Automaten soll ein 3-Phasen Taktgenerator entwickelt werden. Dieser könnte etwa zur Ansteuerung eines Mikromotors mit links-rechts Lauf, Stop und Freilauf verwendet werden. Das folgende Bild 1.3 zeigt den Generator als Black-Box mit dem verlangten Verlauf der Ausgangssignale.

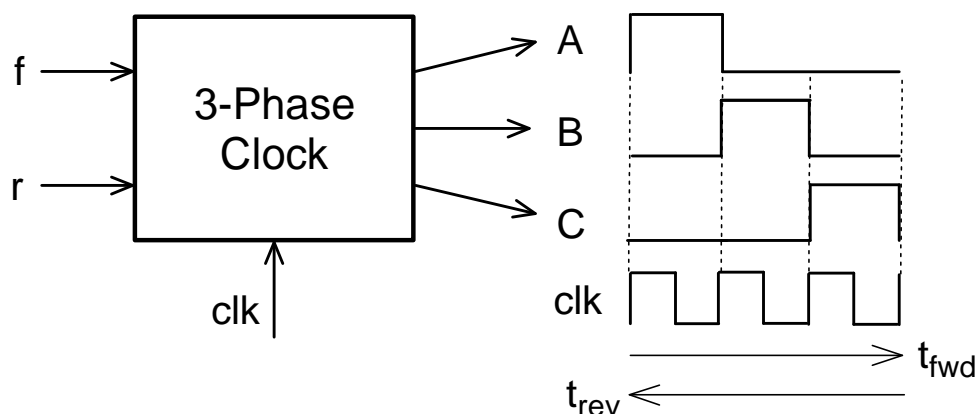


Bild 1.3: 3-Phasen Taktgenerator

Mit den beiden Eingangssignalen *f* (forward) und *r* (reverse) wird die Betriebsart vorgegeben. Neben den beiden Hauptbetriebsarten *vorwärts* (forward, fwd) und *rückwärts* (reverse, rev) sind *aus* (off, alle 3 Ausgänge 0) und *stop* (stp, Stillstand in der aktuellen Phase) gemäss der folgenden Tabelle möglich.

| f | r | Mode | Short | Funktion |
|---|---|---------|-------|------------------------------|
| 0 | 0 | off | off | A, B, C = 0 |
| 0 | 1 | reverse | rev | C> B> A> C> B> A> C> B |
| 1 | 0 | forward | fwd | A> B> C> A> B> C> A> B |
| 1 | 1 | stop | stp | X (= aktuelle Phase A, B, C) |

Betriebsarten des 3-Phasen Taktgenerators

Aufgaben

Wir wollen den 3-Phasen Taktgenerator als Moore-Automaten (Ausgänge nur abhängig vom Zustand) mit der minimal möglichen Anzahl von Zuständen und Flip-Flops und mit getrennter Transition- und Output-Logik gemäss Bild 1.2 entwickeln (dies entspricht der einfachsten Realisierungsart).

1. Wie viele Zustände muss der Automat mindestens haben und wie viele Flip-Flops werden mindestens für den Zustandsspeicher benötigt? → **B**
2. Zeichnen Sie das Zustandsdiagramm für den Automaten auf. Zeichnen Sie darin alle eindeutig durch die Aufgabenstellung gegebenen Übergänge zwischen den Zuständen ein (die Ergänzung mit den nicht bestimmten Übergängen folgt als nächste Aufgabe). → **B**

In der Aufgabe ist nicht beschrieben, wie das Verlassen der Betriebsart *aus* (off) erfolgen muss, sobald ein Wechsel verlangt wird. Nicht vollständige Vorgaben sind typisch für jede praktische Aufgabe. Die Ingenieurin und der Ingenieur muss selbst sinnvolle Annahmen treffen oder beim Auftraggeber nachfragen.

3. Ergänzen Sie das Zustandsdiagramm mit sinnvollen Übergängen aus der Betriebsart *aus* in die anderen Betriebsarten. → **B**
4. Ordnen Sie jedem Zustand einen Zustandscode zu und stellen Sie die Wahrheitstabelle für die kombinatorische Transition-Logik auf (die Output-Logik ist Gegenstand der nächsten Aufgabe).

Tip: im Ein- und Ausgangsteil der Wahrheitstabelle je eine Hilfsspalte einfügen in der decodiert (=Zustandsnummer) im Eingangsteil der aktuelle Zustand (CS = Current State) und im Ausgangsteil der nächste Zustand (NS = Next State) eingetragen wird. Die Darstellung wird damit sehr viel übersichtlicher. Das folgende Beispiel zeigt den Anfang der Wahrheitstabelle für die Transition Logik eines Automaten mit 4 Zuständen und einem Eingang. Daran anschliessend folgt die entsprechende Tabelle in AHDL-Schreibweise als Subdesign Example_Tab.

Achtung: Die Hilfsspalten je als Kommentar (in % - Zeichen eingeschlossen) einfügen. Kommentare können in AHDL nicht verschachtelt werden, nicht balancierte %-Zeichen führen zu irren Fehlern!

| Q1 | Q0 | in | CS | NS | D1 | D0 |
|-------|-------|-------|----------|----------|-------|-------|
| 0 | 0 | 0 | 0 | 3 | 1 | 1 |
| 0 | 0 | 1 | | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 2 | 1 | 0 |
| 0 | 1 | 1 | | 0 | 0 | 0 |
| | | | | | | |

Beispiel für die Darstellung der WT einer Transition-Logik (1)

```

SUBDESIGN Example_tab
(  Q1, Q0, in  : INPUT;
   D1, D0      : OUTPUT; )
BEGIN
TABLE % transition logic %
% CS = Current State,  NS = Next State %
  Q1, Q0, in  % CS % => % NS %  D1, D0 ;
%=====
   0,  0,  0  %  0 % => %  3 %   1,  1 ;
   0,  0,  1  %  0 % => %  1 %   0,  1 ;
   0,  1,  0  %  1 % => %  2 %   1,  0 ;
   0,  1,  1  %  1 % => %  0 %   0,  0 ;
%
%          ..... usw ..... %
END TABLE;
END;

```

Beispiel WT als AHDL-Subdesign (Text-Design-File .Example_Tab.tdf) (1)

(1) Achtung: diese Beispiele entsprechen nicht dem Versuch!

5. Stellen Sie die Wahrheitstabelle für die kombinatorische Ausgangslogik auf.

6. Realisieren Sie den 3-Phasen Taktgenerator auf dem PLD-Board. → **B**

Transition und Output- Logik je mit einer AHDL-Wahrheitstabelle beschreiben (vorausgehendes Beispiel oder die Musterlösung 7-Segment Decoder aus dem V5-Filepaket als Vorlage für die AHDL Tabelle verwenden). Entweder beide Tabellen gemeinsam in einem File (als ein Subdesign, womit ein Symbol für die gesamte komb.Logik entsprechend Bild 1.1 entsteht), oder jede Tabelle in einem eigenen File (wodurch 2 Symbole entsprechend Bild 1.2 entstehen).

Mit dem Graphikeditor den ganzen Automaten zusammensetzen (als Zustandspeicher D-Flip-Flops "dff" aus der prim- Bibliothek einsetzen, nicht eigenes aus Versuch 4!).

Als Taktquelle (clock) den langsamen Taktgenerator auf dem Board verwenden (Signalname GCLK1).

Die Eingangssignale r und f mit den Tasten KA und KB eingeben.

Die Ausgangssignale A, B, C je mit einem Segment auf der Anzeige DB anzeigen (diese so wählen, dass der vorwärts-rückwärts Lauf visualisiert wird, z.B A auf Segment 7, B Segment 2 und C Segment 3, Rest off).

7. **Fakultative Aufgabe:** Wie viele Flip-Flops müssen bei einer Realisierung als **Medwedjew-Automat** verwendet werden und warum braucht diese Realisierung trotzdem weniger Ressourcen (Anzahl PAL-Blöcke, Struktur des PLDs in der Dokumentation zum Board beachten) im PLD?. → **B**

2: Audio Sequence Player

Einführung

Das Endziel des Praktikums ist es einen einfachen Audio-Synthesizer zur Erzeugung kurzer Melodien (System Handy-Ruf) auf dem PLD-Board zu realisieren. Dies werden wir im Versuch 6 machen, wo Sie Melodien nach Noten programmieren und/oder mit einem digitalen pseudo Zufallsgenerator automatisch erzeugen können. Im Versuch 5 begnügen wir uns damit, Tonsequenzen (Tonleitern) zu generieren. Die Blockschaltung des Sequenz Generators zeigt das Bild 2.2. Auf dieses bezieht sich auch die folgende Beschreibung der verschiedenen Komponenten.

Die Bezeichnung Synthesizer für unser Projekt ist etwas hoch gegriffen, unsere Tonleitern und Melodien bleiben monophon und die Töne bleiben auf den unverkennbaren Digiprakt-Rechteck-Sound beschränkt. Trotzdem, lassen wir uns den Spass nicht verderben!

Ton Generator:

Das Kernstück des Synthesizers (siehe Bild 2.2) ist der Tongenerator der eine Reihe von programmierbaren Frequenzen (Tonhöhen) erzeugen kann. Dazu eignet sich ein programmierbarer Zähler. Die Tonhöhe ergibt sich aus der Taktfrequenz dividiert durch die programmierte Länge des Zählers. Mit einem 8-Bit Zähler können für unsere Anwendung genügend genau 32 Töne in temperierter Halbtonstimmung (Intervall $2^{-12} = 1.059463$) erzeugt werden. Dies entspricht ca. 2.5 Oktaven. Für die Auswahl der Halbtöne genügen 5 Bits. Die Umsetzung von Ton (Halbton-nummer) auf 8-Bit Zählerlänge erfolgt mit einer Tabelle. Diese bestimmt die Stimmung des Synthesizers und könnte leicht an andere Stimmungen angepasst werden. Den vollständigen Tongenerator finden sie als *Tone_Gen.gdf* im V5_FilePack. Das folgende Bild 2.1 zeigt den gewählten Aufbau.

Hinweis: Obwohl der Zähler synchron arbeitet, ist das Ganze wegen dem asynchronen Ladevorgang (LDN-Eingang) kein reiner synchroner Automat.

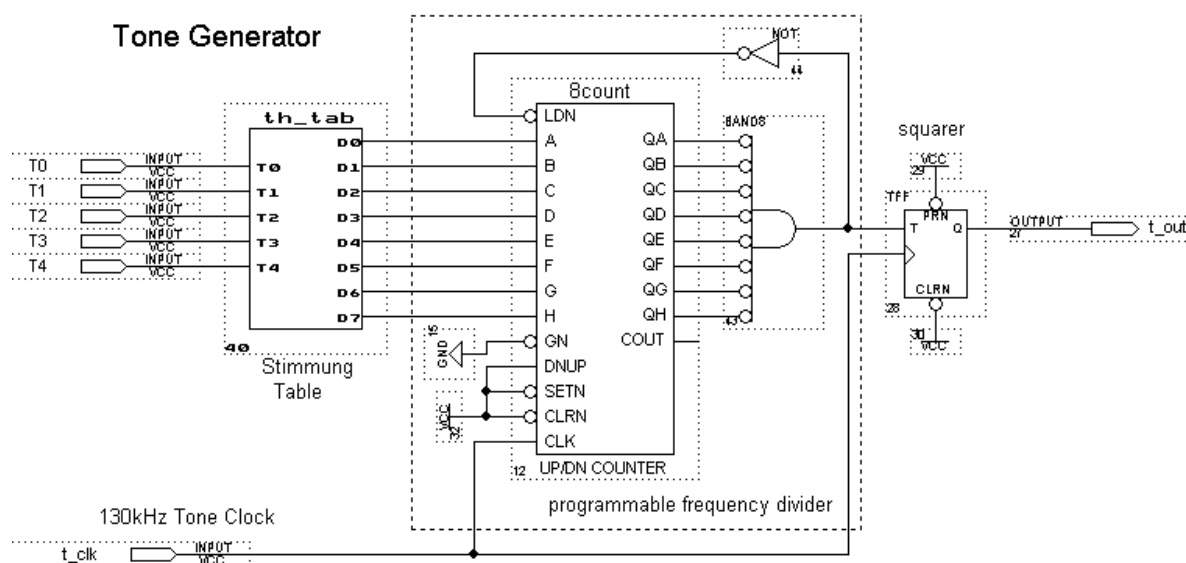


Bild 2.1: Ton Generator (ton_gen.gdf)

Noten Generator:

Der Synthesizer soll Folgen von Tönen erzeugen können. Im Versuch 5 begnügen wir uns mit Tonleitern auf-abwärts in einstellbaren Tonschritten. Diese Aufgabe übernimmt der Noten Generator (note_gen). Um vom aktuellen Ton auf den nächsten zu kommen, soll zum aktuellen ein vorgegebener Schritt (Tone step) addiert werden. Im Rythmus des Notentaktes wird der mit einem Addierer gebildete nächste Ton zum aktuellen. Der Noten Generator ist ein synchroner Automat.

Sequenz Generator:

Das Schema des Sequenz Generators zeigt das Bild 2.2. Zusätzlich zum Ton- und Noten- Generator ist ein Volumen (Lautstärke) Kontroll-Block (vol_ctrl) vorhanden, mit dem die drei Tri-State Tonausgänge (f_out, m_out, p_out) über die Schalter SA3 und SA4 (Tone volume) einzeln ein- oder alle ausgeschaltet werden können. Bereits eingezeichnet sind die zwei Schnittstellen (A, B, Interface to V6 Melody Generator) über welche im Versuch 6 der Ausbau zum Melodie Generator erfolgen wird.

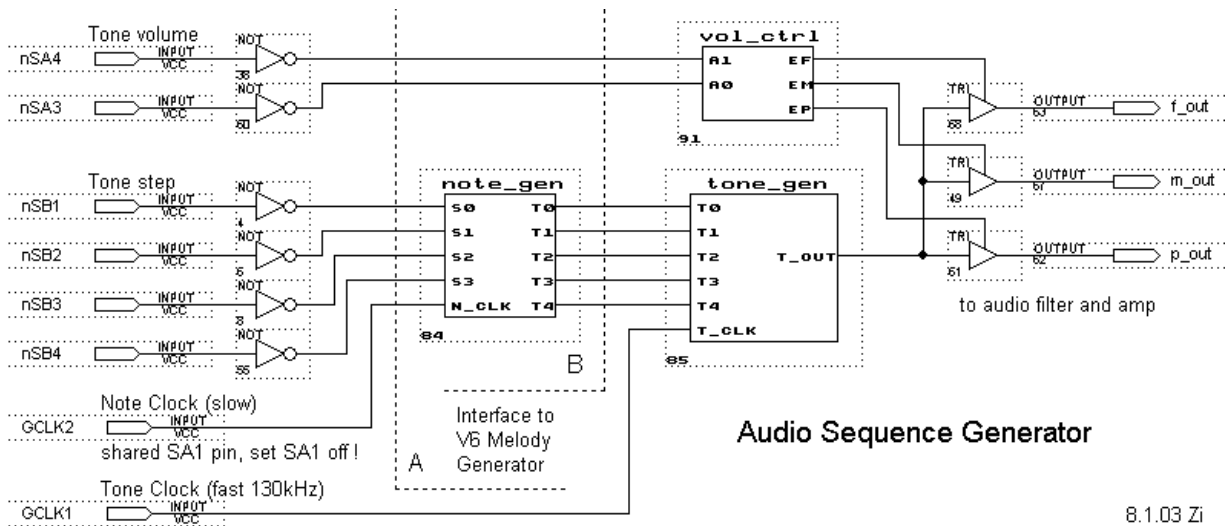


Bild 2.2: Sequenz Generator.

Audio Ausgang:

Die drei Rechteck- Tonausgänge werden mit einer invertierenden Operationsverstärker Schaltung summiert, etwas gefiltert (2 Tiefpässe 1.Ordnung mit RC Gliedern) und mit genügender Leistung auf die Ausgangsbuchse geführt. Der eingesetzte OpAmp ist ein CMOS-RRIO (klein-) Leistungstyp (RRIO = Rail to Rail Input und Output, Ein- und Ausgangssignale können von GND bis zur positiven Speisung angesteuert werden).

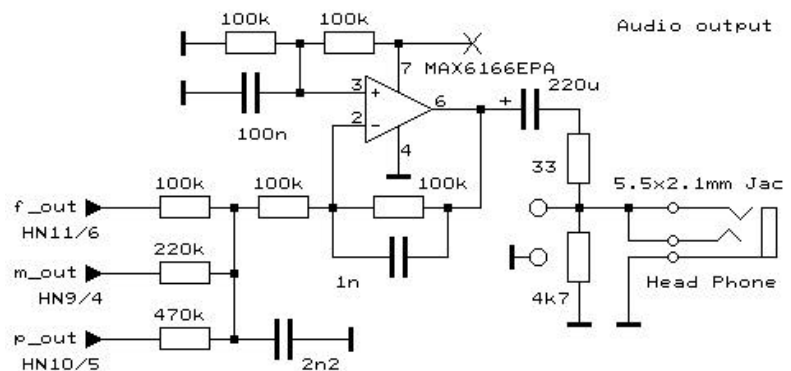


Bild 2.3: Analoge Audio Ausgangsschaltung

Vorbereitung am Praktikumsnachmittag:

Bevor Sie den 2. Teil der Aufgabe beginnen müssen Sie auf dem PLD-Board folgende Einstellungen vornehmen (siehe auch SB2_Info auf Arbeitsplatz oder Digiprakt Seite):

Den *Clock select* Jumper (oberhalb der beiden Takttrimmer) auf high (linke Position) stecken.

Den schnellen Taktgenerator mit dem linken Takttrimmer auf 130kHz einstellen. Entspricht etwa der Mittelstellung des Trimmers, genaue Einstellung durch Messung mit dem KO am *Clock in* Pin (ist jedoch nicht notwendig).

Den Schalter SA1 auf OFF schalten. SA1 ist am PLD-Pin 2 angeschlossen, dieser wird jedoch für die Eingabe des Notentakts (*N_clk*) vom langsamen Taktgenerator als Global Clock 2 (*GCLK2*) verwendet. Deshalb muss SA1 unbedingt auf OFF gestellt werden und in dieser Position bleiben!).

Den langsamen Taktgenerator über das bereitliegende Jumperkabel mit HN7 verbinden (vom freien 3.Pin rechts auf der Takt select Stiffliste auf die 2-reihige Stiffliste oberhalb des PLDs, obere Reihe Pin in der Mitte = 4.Pin von aussen).

Aufgaben

Der Sequenz Generator ist gemäss Bild 2.2 bereits als Schema (*Seq_gen_2_vt.gdf*) im *V5_FilePack* mit einer Tonanzeige auf DA und DB erweitert vorhanden. Gehen sie für alle Aufgaben davon aus.

8. Wie funktioniert der programmierbare Frequenzteiler (Zähler) im Ton Generator (Bild 2.1) und welche Aufgabe hat das T-Flip-Flop am Ausgang? → **B**
9. Entwerfen Sie das Schema des Noten Generators (*note_gen.gdf*) welcher nur als leeres Symbol vorhanden ist nach der Beschreibung in der Einführung. → **B**

Den Addierer aus 5 1-Bit Stufen (*add1.gdf* aus dem *V5_FilePack* oder aus dem eigenen vom Versuch 2) aufbauen. Als D-Flip-Flops unbedingt *dff* aus der prim – Bibliothek verwenden, nicht eigene aus dem Versuch 4!

Man überlege sorgfältig, Wie das fünfte (höchstwertige) Bit des Addierereingangs beschaltet werden muss, damit der Schritt richtig als 4-Bit 2-er Komplement Wert (-8 bis +7) zur aktuellen 5-Bit Note addiert wird.

Compilieren und programmieren Sie den Sequenz Generator. Test auf dem PLD-Board, dazu den mitgebrachten Kopf- oder Ohrhörer an der 3.5mm Audio-Klinkenbuchse anschliessen (geht relativ streng).

Achtung: Im *V5_FilePack* ist bereits das *Seq_Gen_2_vt.acf* File mit Pinlocking enthalten. Pin-Lock-Tabelle nicht mehr aus *SB2_Pin-Lock-List.txt* kopieren!.

10. Was für ein Automatentyp ist der Notengenerator (*note_gen*)? → **B**
11. **Fakultative Aufgabe:** Entwerfen Sie an Stelle des Lautstärke-Kontrollblocks (*vol_ctrl*) einen **Amplituden Kontrollblock** (*amp_ctrl*) mit welchem neben der bisherigen Einstellung der Lautstärke bei gedrückter Drucktaste KA Staccato Töne erzeugt werden können.

Der Amplituden-Kontrollblock muss zusätzliche Eingänge KA für das Tastensignal und *n_clk* für den Notentakt (*GCLK2*) erhalten.

Bei einem Staccato-Ton muss die Amplitude in der ersten Phase (Signal = 0) des Notentakts null sein, in der zweiten (Signal = 1) den durch die eingestellte Lautstärke gegebenen Wert erhalten.

Die Beschreibung ist mit Schema oder Tabelle möglich. Die Verwendung einer Tabelle wird recht einfach, wenn man berücksichtigt, dass AHDL auch dont cares (x) im Eingangsteil (linker Teil) der Tabelle zulässt!

Bauen Sie die Erweiterung ins Top-Level Schema ein und benennen Sie dieses von Seq_Gen_2_vt.gdf in Seq_gen.gdf um (File > Save As).

Wenn das Schema bei der Umbenennung Projekt ist, so wird automatisch das acf File mit umbenannt und der Chipnamen darin nachgeführt.

Compilieren, programmieren und austesten. → **B**

Schluss

Der Sequenzgenerator wird im Versuch 6 zum Melodiegenerator ausgebaut werden. Sichern Sie deshalb Ihre Arbeit (alle tdf, gdf und sym Files sowie das Top-Level acf File). Sonst können Sie für den Versuch 6 die Musterlösung herunterladen.

Bevor Sie den PC herunterfahren und alles ausschalten unbedingt sämtliche Files und Directories des Versuches löschen sowie auch noch den Papierkorb leeren.

Wie immer: vergessen Sie bitte nicht, im Feedback dem Versuch eine Note zu erteilen und Ihren Kommentar zum Versuch zu notieren.

In eigener Sache

Viele von Ihnen haben mit dem heutigen Versuch 5 wohl die Testatbedingung (5 von 6 Versuchen gemacht) erfüllt. Manche werden deshalb den Versuch 6 nicht mehr machen. Das ist sehr schade, aber ich verstehe sehr wohl, dass für Sie das DigiPrakt auch nicht die Welt bedeutet. Deshalb möchte ich mich schon jetzt bei allen für die Teilnahme am Praktikum herzlich bedanken und schöne Ferien und ein gutes weiteres Studium wünschen!. Ich hoffe auch sehr, einige nächstes Jahr als Hilfsassistenten wieder im DigiPrakt anzutreffen.

Ihr Praktikumsleiter
Dr. Rolf Zinniker

Babylon ist überall

Sie haben es sicher bereits lebhaft erfahren und sich drüber geärgert, dass gleiche Dinge je nach Geschmack oder Lust und Laune auf verschiedenste Arten bezeichnet und dargestellt werden: Logikschaltungen mit internationalen (amerikanischen) oder deutschen (in Deutschland genormten, ursprünglich eingeführt damit man sie auch mit der Schreibmaschine produzieren kann, - kein Witz!) Symbolen gezeichnet, Latches als Flip-Flops, Eingänge mit E1, E2, S, R, C, T, Ausgänge mit A, Q, X, Y, Z, Antivalenz mit XOR, EXOR, EOR, $\bar{}$, =1, Speisespannungen mit Vcc, Ucc, Vdd, Udd, Masse mit GND, Vee, Vss, bezeichnet werden, und so weiter und so fort.

Auch in den Anleitungen zu unserem Praktikum können Sie sicher genügend Beispiele dafür finden. Oft lässt es sich einfach nicht vermeiden, oft ist es nur Unaufmerksamkeit oder schlechte Gewohnheit, gelegentlich ein undurchschaubares höheres Konzept und manchmal auch einfach nur Bequemlichkeit um nicht zum x-ten mal alle Zeichnungen und Schemas ändern zu müssen. Ich möchte Sie dafür um Nachsicht und Entschuldigung bitten.

Als Trost bleibt die Zuversicht, dass Ingenieurinnen und Ingenieure schlussendlich genügend flexibel sind, um sich trotz allem in diesem babylonischen Wirrwarr zurecht zu finden.