technische universität
dortmund

fakultät für informatik
informatik 12

BFP Entwurf Eingebetteter Systeme „Elektronische Musik“

Wintersemester 2011/12

„Ein Klavier“ oder „Wie klingt ein Rechteck?“

Michael Engel
Informatik 12
TU Dortmund

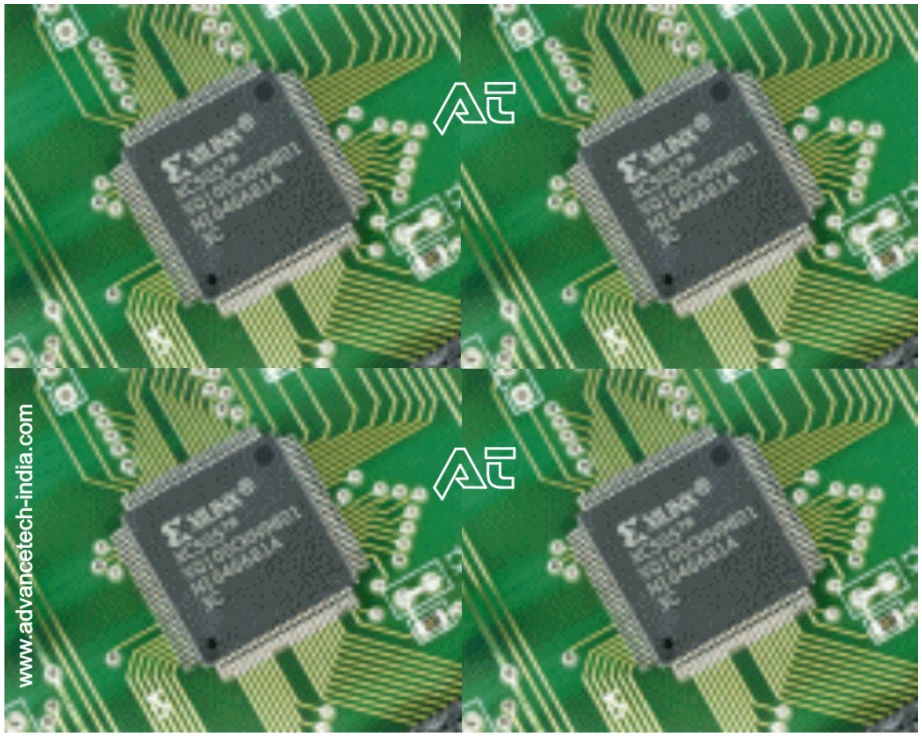
2011/10/26

Überblick

- Entwurfsprinzipien
- Beispielcode
- Ergebnis und Erweiterungen
- Ausblick: Wie klingen Rechtecke besser?

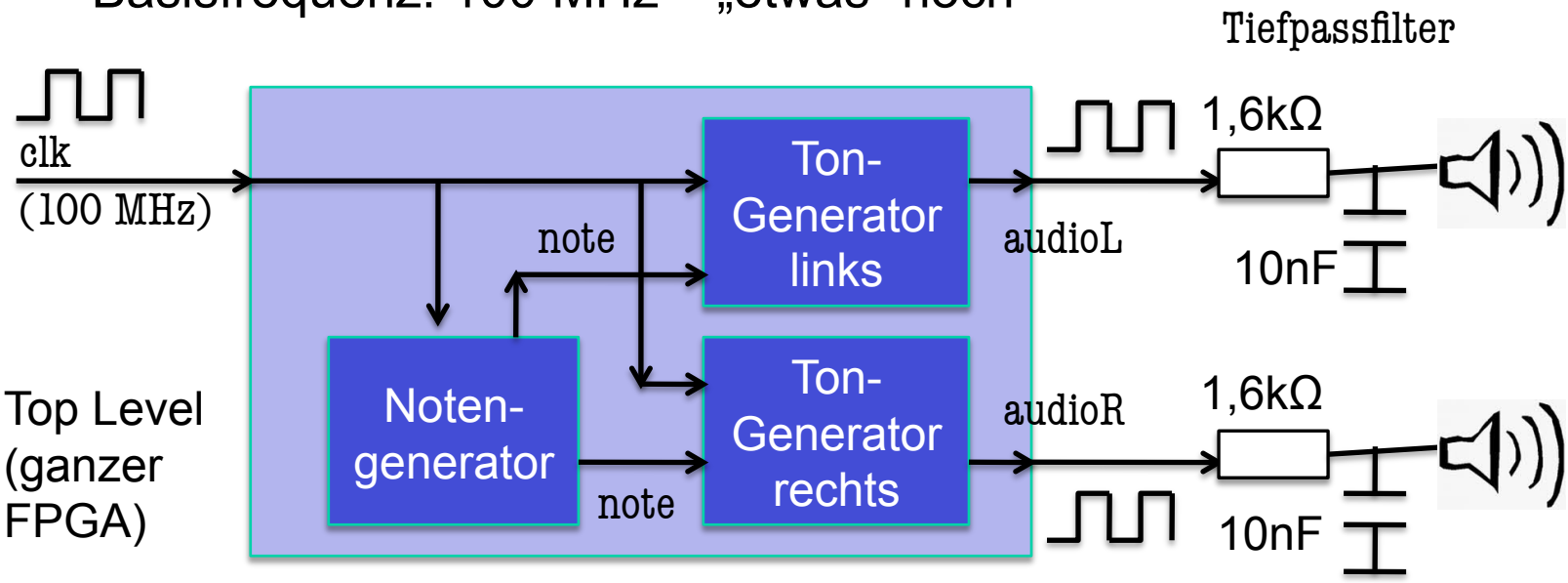
Entwurfsprinzipien

- Schaltungsentwurf
  - Einzelne Komponenten (ICs) mit fest definierten Schnittstellen (Signalen/Bussen an Pins)
  - Verbunden durch Leiterbahnen
- Hierarchien
  - Chips -> Platinen -> Systeme...



Entwurf des „Klaviers“

- Komponenten:
  - Frequenzgenerator für einzelne Noten
  - Zeitsteuerung
  - Melodie spielen
- Grundprinzip: Frequenzteiler
  - Basisfrequenz: 100 MHz – „etwas“ hoch



Frequenzen teilen

- Grundprinzip: Frequenzteiler
  - Basisfrequenz: 100 MHz – „etwas“ hoch
  - Wie sehen Frequenzen für einzelne Noten aus?
  - Kammerton „a“ = 440 Hz
- f = 440 Hz = 1/t => Periodendauer 1/440s = 227,272 ms
- Ein 100 MHz-Zähler muss also
  - 100*10^6 / 440 = **227272**-mal
    in den 227,272 ms Periodendauer für den Ton a „ticken“
- Für alle 12 Halbtöne:

Ton	C	C#	D	D#	E	F	F#	G	G#	A	A#	H
Hz	261	277	293	311	329	349	370	392	415	440	466	493
Div	382219	360773	340529	321409	303370	286344	270277	255102	240789	**227272**	214518	202478

VHDL-Frequenzteiler

- Der Tongenerator – Deklaration:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity PlayNote is
   Port (   clk : in  STD_LOGIC;
            note : in integer;
            audio : out  STD_LOGIC
            );
end PlayNote;
```

VHDL-Frequenzteiler (2)

- Der Tongenerator – Implementierung:

```
architecture Behavioral of PlayNote is

signal c : integer range 0 to 24999999 := 0; – 0,25s bei 100MHz f_osc
signal x : std_logic:= '0';

type note_type is array(0 to 12)of integer;
signal notes: note_type;

begin
        – 12 half tones starting from C0
        notes <= ( 382219, 360773, 340529, 321409, 303370, 286344,
                   270277, 255102, 240789, 227272, 214518, 202478, 191113 );
```

VHDL-Frequenzteiler (3)

- Der Tongenerator – Implementierung:

```
  process begin
    wait until rising_edge(clk); – warten bis zum nächsten Takt
    if (c<notes(note)/2) then   – 100 MHz-Zähler bis zum Max.-wert für Note laufen lassen
       c <= c+1;                – wenn kleiner: weiterzählen
    else                        – wenn Zählerende erreicht:
       c <= 0;                  – Zähler zurücksetzen
       x <= not x;              – und Signal x togglen -> Rechtecksignal
    end if;
  end process;
  audio <= x;                   – Signal x ausgeben
end Behavioral;
```

Melodie spielen

- Melodie ist Abfolge von Tonwerten, als Halbton angegeben:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity GenerateNotes is
   Port (     clk : in  STD_LOGIC;              -- Eingabe: 100 MHz-Takt
              note : out integer                -- Ausgabe: aktuell zu spielende Note
   );
end GenerateNotes;

architecture Behavioral of GenerateNotes is

type melody_type is array(0 to 15) of integer;
signal melody: melody_type;
signal c : integer range 0 to 15 := 0; -- Noten C, C#, D, D#, E, ...H
signal count : integer range 0 to 199999999 := 0;
```

Melodie spielen (2)

- Zähler verwendet selbes Prinzip wie Notengenerator:

```
begin
                -              C  D E  F  G G  A A  A A  G  A A  A A  G
                - melody <= (0, 2, 4, 5, 7, 7, 9, 9, 9, 9, 7, 9, 9, 9, 9, 7);
                melody <= (12, 10, 12, 7, 3, 7, 0, 0, 12, 10, 12, 7, 3, 7, 0, 0);

  process begin
    wait until rising_edge(clk);      - warten bis zum nächsten Takt
    if (count<25000000) then          - 250000000 = 1/4 Sekunde bei 100MHz
       count <= count+1;              - wenn kleiner: weiterzählen
    else                              - wenn Zählerende erreicht:
       count <= 0;                    - Zähler zurücksetzen
       c <= c + 1;                    - und Notenindex für "melody" hochzählen
    end if;
  end process;
  note <= melody(c);                   - aktuellen Notenwert ausgeben
end Behavioral;
```

Zusammenbauen

- Top-Level-Modul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sound_top is                    – Schnittstelle nach “aussen”: externe FPGA-Anschlüsse
   Port (     clk : in  STD_LOGIC;                  – clk (100MHz) kommt rein
              audioL : out  STD_LOGIC;              – Stereo Audio kommt raus...
              audioR : out  STD_LOGIC);
end sound_top;
```

Zusammenbauen (2)

- Deklaration der Komponentenschnittstellen (redundant!):

```
architecture Behavioral of sound_top is

   COMPONENT PlayNote
   PORT(
      clk : IN  std_logic;
      note : IN  integer;
      audio : OUT  std_logic
     );
   END COMPONENT;

   COMPONENT GenerateNotes
   PORT(
      clk : IN  std_logic;
      note : OUT  integer
      );
   END COMPONENT;
```

Zusammenbauen (3)

- Instanziieren und „Zusammenstecken“ der Komponenten:

```
signal note: integer;      – “Leitung”, um generierte Note an Tongenerator weiterzuleiten

begin
  SoundGen: PlayNote PORT MAP (              – 1. Instanziierung des Tongenerators
        clk => clk,                          – “clk” von “aussen”
        note => note,                        – “note” vom “note-Signal
        audio => audioL                      – “audio” nach “aussen” an linken Kanal
      );
  SoundGen2: PlayNote PORT MAP (
        clk => clk,
        note => note,
        audio => audioR                      – “audio” nach “aussen” an rechten Kanal
      );
 GenNote: GenerateNotes PORT MAP (
        clk => clk,
        note => note                         – aktuelle Note mit signal “note” verbinden
      );
end Behavioral;
```

Ergebnis

- Wieviel Platz braucht sowas im FPGA
  - Für Nexys3 Spartan 6 SLX 16
- Ergebnis der Synthese:
  - Number of Slice LUTs: 93 of 9112 = 1%
- Wie klingt das Rechteck nun?
  - Demo!
- Erweiterungen?
  - Größerer Tonumfang! (aktuell nur eine Oktave)
  - MIDI Input?

MIDI

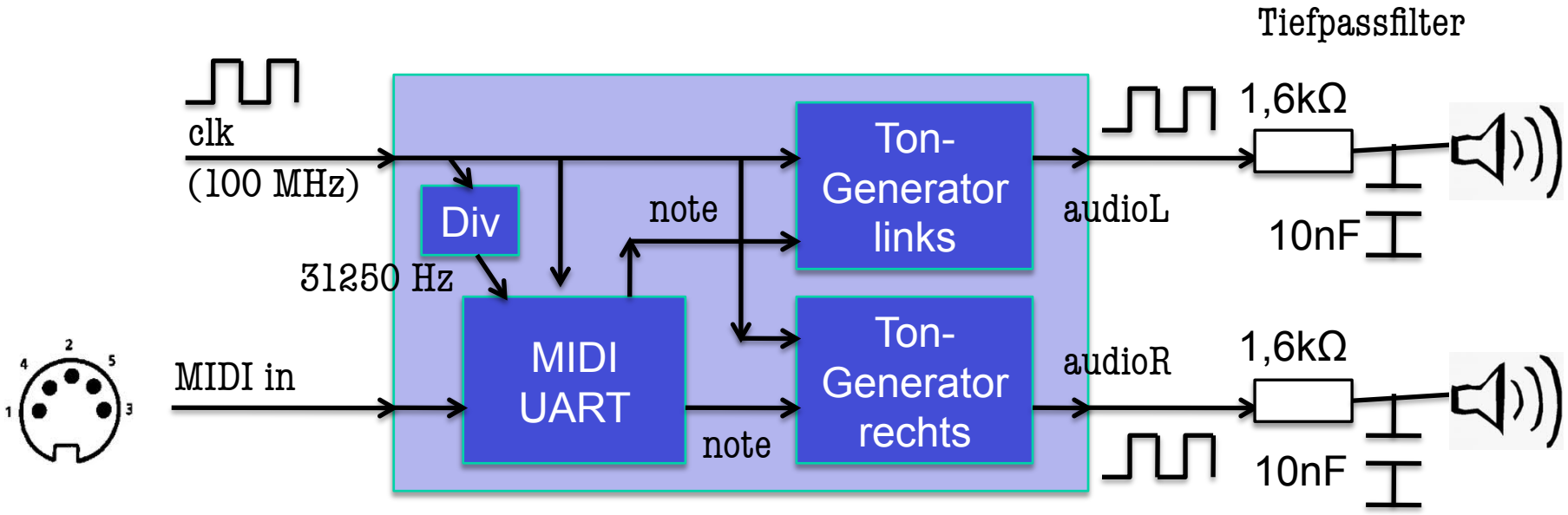
- Bitserielles Protokoll zum Austausch von Noteninformationen zwischen Instrumenten
- 1–3 Byte lange Datenpakete (+ 1 variabel langer Pakettyp)
- Einfach über serielle Schnittstelle (UART) realisierbar
- Einfache Befehlsstrukturen, z.B.:

Status D7----D0	**Data Byte(s) D7----D0**	**Description**
Channel Voice Messages [nnnn = 0-15 (MIDI Channel Number 1-16)]		
1000nnnn	0kkkkkkk 0vvvvvvv	Note Off event. This message is sent when a note is released (ended). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity.
1001nnnn	0kkkkkkk 0vvvvvvv	Note On event. This message is sent when a note is depressed (start). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity.
1010nnnn	0kkkkkkk 0vvvvvvv	Polyphonic Key Pressure (Aftertouch). This message is most often sent by pressing down on the key after it "bottoms out". (kkkkkkk) is the key (note) number. (vvvvvvv) is the pressure value.

http://www.midi.org/techspecs/midimessages.php

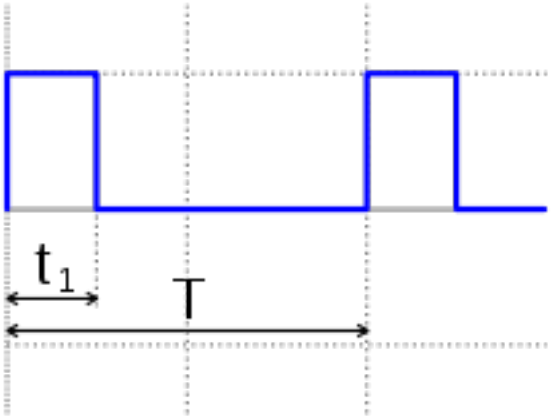
MIDI Input

- Erweiterung des Rechteck-Synthesizers um MIDI in
  - Austausch des Notengenerators durch UART
  - UART empfängt MIDI-Nachrichten (Note on/off)
  - Setzt „key“ 0kkkkkkk-Parameter in „note“ um
  - Ignoriert „velocity“-Parameter



Besser klingende Rechtecke

- Problem des Rechtecksynthesizers:
  - „harte“ Klänge, sehr künstlich
- Wie lässt sich mit 1 Bit Auflösung besserer Klang erzeugen?
- *Pulsweitenmodulation* (Pulse Width Modulation, PWM)



Tastverhältnis t1/T = 25%
Wenn f genügend hoch (also T klein genug), dann erzeugt ein Tiefpass 25% der max. Amplitude