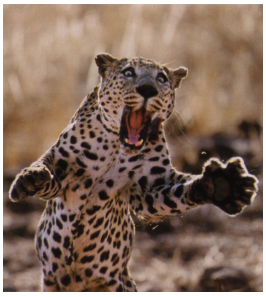
*8. Übung: Modem* (24 P)



*Data!*

In dieser Übung werden die Signalverarbeitungskomponenten zu einem vollständigen Modem zusammengesetzt. Als Ergebnis stehen ein FSK-Sendesystem (Tx-Gruppen) und ein FSK-Empfängersystem (Rx-Gruppen) zur Verfügung (Abb. 1).

Der Sender setzt serielle digitale Signale (z.B. von der seriellen Schnittstelle eines PC) in ein FSK-moduliertes Sendesignal um. Dieses Signal nutzt eine Trägerfrequenz im (hörbaren) Schallbereich und kann mittels eines Lautsprechers (= Wandler von elektrischer Energie in Schallenergie) auf den Übertragungskanal (= Umgebungsluft) gelegt werden.

Die Pegel '0' und '1' des seriellen Datenstroms werden jeweils durch eine Frequenz dargestellt (2-FSK). Dieses Frequenzpaar entspricht einem Kanal. Damit mehrere Modems weitgehend ungestört voneinander über dasselbe Übertragungsmedium kommunizieren können, wird das Frequenzspektrum in meherere Kanäle unterteilt. Abbildung 2 gibt einen Überblick über alle verfügbaren Kanäle.

Auf der Empfängerseite werden die ankommenden Schallwellen mittels eines Mikrofons wieder in elektrische Schwingungen umgewandelt und stehen nach Abtastung durch einen ADC als digitale Signale zur Verfügung. Der Empfänger wertet diese Signale aus und erzeugt aus ihnen wiederum einen seriellen Datenstrom, der demjenigen auf der Sendeseite so weit als möglich entsprechen sollte. Dieser Datenstrom kann an die serielle Schnittstelle eines zweiten PCs übergegeben werden.

Auf diese Weise können zwei PCs über einen Luftübertragungskanal miteinander kommunizieren.

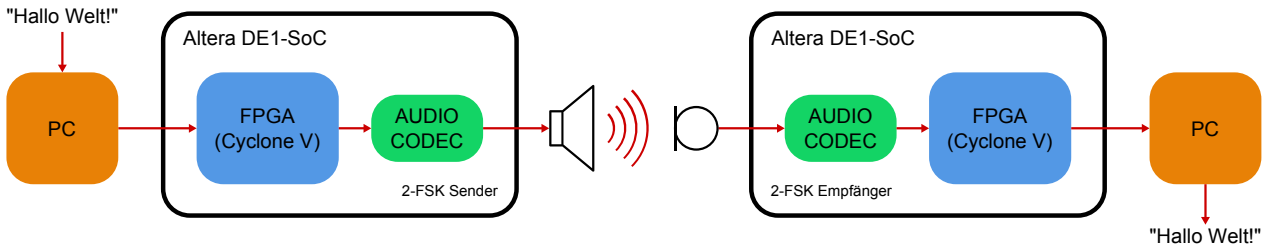



Abbildung 1: Überblick über Modemsystem

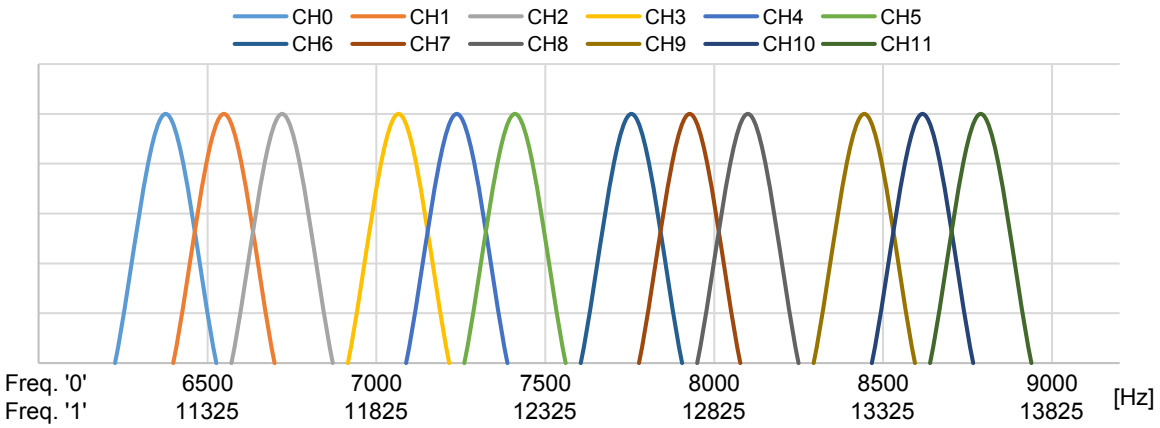



Abbildung 2: Kanalübersicht

## Tx-Gruppe

***Hinweis:*** Dieser Abschnitt muss nur von *Tx*-Gruppen bearbeitet werden.

**1. Aufgabe** *FSK-Sender*

a☐5 Entwickeln Sie eine Unit `TxFsk` (in der Group `ComFsk`) mit folgender Schnittstelle:

```
entity TxFsk is
  generic (
    gClkFrequency     : natural := cDefaultClkFrequency;
    gAudioBitWidth    : natural := cDefaultAudioBitWidth;
    gSampleRate       : natural := cDefaultSampleRate;
    gChannels         : aSetOfTxChannels := (
        0 => (Frequency0 =>  800.0,   -- Hz
              Frequency1 => 1300.0),  -- Hz
        1 => (Frequency0 => 2800.0,   -- Hz
              Frequency1 => 3300.0))  -- Hz
  );
  port (
    inResetAsync : in std_ulogic;
    iClk         : in std_ulogic;
    -- Channel selection signal
    iChannelSelect : in natural range gChannels'range;
    -- Input data signal (must of course be synchronous!)
    iD : in std_ulogic;
    -- Sample sync signal for the DAC
    iSampleStrobe : in std_ulogic;
    -- Parallel digital audio data protocol
    oVal : out std_ulogic;
    oD   : out aAudioData(0 downto -(gAudioBitWidth-1))
  );
end entity TxFsk;
```

In der Architecture dieser Unit werden lediglich die Unit `DspDds` instantiiert und die durch diese erzeugte Frequenz je nach anliegendem Datensignal `iD` und anliegender Kanalauswahl `iChannelSelect` eingestellt.

Ihre Unit soll grundsätzlich eine beliebige Anzahl an Frequenzpaaren (Kanälen) unterstützen. Diese werden mittels der Generic `gChannels` definiert. Die Definition des Datentyps `aSetOfTxChannels` finden Sie im Package `DefinitionsFsk` (in der Group `ComFsk`):

```
  type aTxFrequencySet is record
    Frequency0 : real;
    Frequency1 : real;
  end record;

  type aSetOfTxChannels is array (natural range <>) of aTxFrequencySet;
```

Diese Frequenzen werden in der Unit anhand der Abtastrate und der Auflösung des Phasenzählers in die entsprechenden Phaseninkrementwerte umgerechnet. Zur Speicherung der Phaseninkrementwerte können Sie die Datentypen `aSetOfPhaseIncrements` und `aPhaseIncrementSet` zuhilfe nehmen.

b☐3 Erstellen Sie eine Testbench und weisen Sie die korrekte Funktion der Unit nach. Den seriellen Datenstrom können Sie beispielsweise in einer Konstanten ablegen.

c☐3 Nun benötigen Sie noch ein Testbed (Unit `TbdTxFskBasic`), welches den Codec parametriert und

einen seriellen Datenstrom entgegen nimmt. Vorläufig soll dieses Signal von einem Schalter kommen. Ihrer Gruppe wurden zwei Kanäle zugeteilt. Einen Kanal haben Sie mit Ihrer zugeteilten Partner-Rx-Gruppe gemeinsam, den anderen haben alle Gruppen gemeinsam. Auch der verwendete Kanal soll über einen Schalter eingestellt werden.

***Wichtig!*** Bitte beachten Sie bei der Konstruktion des Testbeds, dass diese Signale (Schalter) asynchron zum Takt des FPGA sind und daher innerhalb des Testbeds einsynchronisiert werden müssen!

d☐ Prüfen Sie die korrekte Funktion des Testbeds auf dem Rapid Prototyping Board.

e☐$_3$ Messen Sie die erzeugten Frequenzen mit dem Audiotester (***Screenshots!***).

f☐$_1$ Wie groß ist die belegte Fläche im FPGA?

g☐$_1$ Wie groß ist die maximale Taktfrequenz für das FPGA?

h☐$_1$ Welche maximale Frequenz könnten Sie mit Ihrem Entwurf (unter der Annahme, dass ein entsprechend schneller DAC-Baustein angeschlossen wäre) demnach erzeugen?

i☐$_2$ Zeichnen Sie ein ***Blockschaltbild*** des Datenpfads (d.h. Schalter, Frequenzauswahl, ..., Lautsprecher) ihres FSK-Senders.

## Rx-Gruppe

***Hinweis:*** Dieser Abschnitt muss nur von *Rx*-Gruppen bearbeitet werden.

**2. Aufgabe** *FSK-Empfänger*

a☐8 Entwickeln Sie eine Unit `RxFsk` (in der Group `ComFsk`) mit folgender Schnittstelle:

```
entity RxFsk is
  generic (
    gClkFrequency  : natural    := cDefaultClkFrequency;
    gAudioBitWidth : natural    := cDefaultAudioBitWidth;
    gSampleRate    : natural    := cDefaultSampleRate;
    gCoefWidth     : natural    := cFskFilterCoefWidth;
    -- Filter coefficients (The number of these coefficients determines
    -- the number of taps of the filter.)
    gChannelBandpasses : aSetOfRxBandpasses := (
        0 => (Bandpass0 => ( ... ),     -- band pass for Channel 0/freq. 0
              Bandpass1 => ( ... )),    -- band pass for Channel 0/freq. 1
        1 => (Bandpass0 => ( ... ),     -- band pass for Channel 1/freq. 0
              Bandpass1 => ( ... ));    -- band pass for Channel 1/freq. 1
    gLowpass : aSetOfFactors := ( ... ) -- low pass
  );
  port (
    inResetAsync : in std_ulogic;
    iClk         : in std_ulogic;
    -- Channel selection signal
    iChannelSelect : in natural range gChannelBandpasses'range;
    -- Parallel digital audio data protocol
    iVal : in std_ulogic;
    iD   : in aAudioData(0 downto -(gAudioBitWidth-1));
    -- Output data signal
    oD : out std_ulogic
  );
end entity RxFsk;
```

Die zugehörige Architecture soll jene Struktur verwirklichen, welche im Konzeptstadium entwickelt wurde. Diese besteht neben den Filtern aus einer Absolutwertbildung, einer Subtraktion, und einer Schwellenwertentscheidung. Die Funktion `abs()` wird vom Package `fixed_pkg` bereits fertig angeboten.

Beachten Sie bitte, dass möglicherweise (Pipeline-)Register in den Ablauf der Datenverarbeitung eingefügt werden müssen, da sonst die notwendige Systemtaktfrequenz u.U. nicht erreicht wird.

Ihre Unit soll grundsätzlich eine beliebige Anzahl an Frequenzpaaren (Kanälen) unterstützen. Die Kanalauswahl erfolgt anhand des Eingangssignals `iChannelSelect`. Die Koeffizienten der Bandpassfilter für jeden Kanal werden mittels der Generic `gChannelBandpasses` definiert. Die Definition des Datentyps `aSetOfRxBandpasses` finden Sie im Package `DefinitionsFsk` (in der Group `ComFsk`):

```
  constant cFskBandpassOrder   : natural := ...;

  type aRxBandpassSet is record
    Bandpass0 : aSetOfFactors(0 to cFskBandpassOrder);
    Bandpass1 : aSetOfFactors(0 to cFskBandpassOrder);
  end record;

  type aSetOfRxBandpasses is array (natural range <>) of aRxBandpassSet;
```

Sie können einfach für jeden Kanal ein eigenes Paar Bandpassfilter vorsehen. Wird Ihre Unit beispielsweise für 2 Kanäle konfiguriert, würden Sie somit 4 Bandpassfilter instanzieren. Die Auswahl des zu empfangenden Kanals könnte beispielsweise direkt zwischen den Bandpässen und der Absolutwertbildung (oder auch zwischen der Subtraktion und dem gemeinsamen Tiefpassfilter) erfolgen. Damit brauchen Sie Ihre Bandpassfilter nicht dynamisch zu rekonfigurieren. Beachten Sie, dass alle Ihre Bandpassfilter dieselbe Anzahl an Koeffizienten haben müssen. Sie müssen die Konstante `cFskBandpassOrder` demnach an Ihre Implementierung anpassen.

b☐4 Erstellen Sie eine Testbench und weisen Sie die korrekte Funktion der Unit nach.

c☐2 Nun benötigen Sie noch ein Testbed (Unit `TbdRxFskBasic`), welches den Codec parametriert und einen seriellen Datenstrom ausgibt. Vorläufig soll dieses Signal auf eine LED ausgegeben werden. Ihrer Gruppe wurden zwei Kanäle zugeteilt. Einen Kanal haben Sie mir Ihrer zugeteilten Partner-Tx-Gruppe gemeinsam, den anderen haben alle Gruppen gemeinsam. Der verwendete Kanal soll über einen Schalter eingestellt werden.

***Wichtig!*** Bitte beachten Sie bei der Konstruktion des Testbeds, dass dieses Signal (Schalter) asynchron zum Takt des FPGA ist und daher innerhalb des Testbeds einsynchronisiert werden muss!

d☐ Prüfen Sie die korrekte Funktion des Testbeds auf dem Rapid Prototyping Board.

e☐1 Wie groß ist die belegte Fläche im FPGA?

f☐1 Wie groß ist die maximale Taktfrequenz für das FPGA?

g☐1 Welche maximale Frequenz könnten Sie mit Ihrem Entwurf (unter der Annahme, dass ein entsprechend schneller ADC-Baustein angeschlossen wäre) demnach empfangen?

h☐2 Zeichnen Sie ein ***Blockschaltbild*** des Datenpfads (d.h. Mikrofon, Codec, Schalter, Filterblöcke, Absolutwertbildung, Subtraktion, Kanalauswahl, Entscheidung, LED, ...) ihres FSK-Empfängers.

## Tx- und Rx-Gruppe

***Hinweis:*** Dieser Abschnitt muss von beiden Gruppen bearbeitet werden.

**3. Aufgabe** *Serielle Schnittstelle*

Auf Ihrem Rapid Prototyping Board (DE1-SoC) steht eine serielle Schnittstelle in Form eines UART-to-USB-Adapters zur Verfügung. Diesen können Sie über den Mini-USB-Anschluss J4 mit einem PC verbinden. Der Adapterchip FTDI FT232R übernimmt dabei die Codierung und Decodierung des UART-Framing-Protokolls. Sie können daher die TXD-Leitung des FT232R direkt als seriellen Bitstrom am Eingang des FSK-Senders verwenden. Es ist nicht notwendig eigene UART-Transceiver zu implementieren.

Allerdings sind die TXD- und RXD-Pins des FT232R an Pins des im Cyclone V integrierten Hard Processing Systems (HPS) angeschlossen. Diese Pins können Sie also nicht direkt für Ihr FPGA-Design nutzen. Stattdessen muss das HPS so konfiguriert werden, dass diese Pins an den FPGA-Teil weitergereicht werden („Loan-IO“). Abbildung 3 gibt einen Überblick über dieses System.

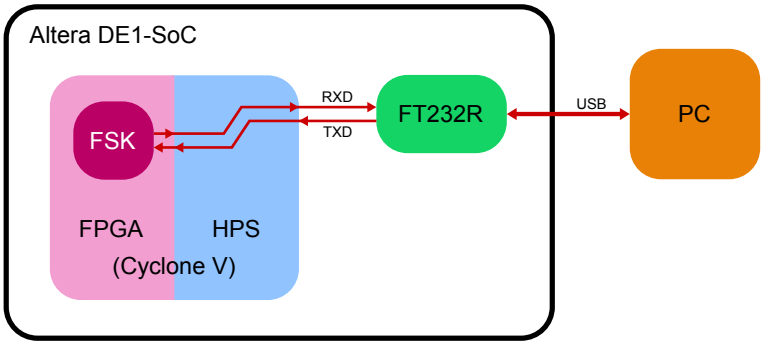



Abbildung 3: Anbindung des Cyclone V FPGA-Teil an den UART-to-USB-Adapter über das HPS.

a☐3 Die Unit `HPSComputerMin` in der Group `AlteraCycloneV` liefert Ihnen ein entsprechend vorkonfiguriertes HPS in einer einfach instanzierbaren Wrapper-Unit. Erstellen Sie ein weiteres Testbed `TbdTxFsk` bzw. `TbdRxFsk`, welches das HPS integriert und den seriellen Datenstrom über den FT232R überträgt. Sie können die Datei `DE1_SoC_Computer.vhd` (direkt im Ordner der Unit `HPSComputerMin`) als Anhaltspunkt für die Einbindung der Wrapper-Unit in Ihr Design heranziehen.

Die Entity dieser Unit stellt verschiedene Ports bereits, die Sie in Ihr Design integrieren müssen:

- Der Port `CLOCK_50` muss mit dem **50-MHz**-Takt (vor Ihrer PLL!) betrieben werden.
- Das Signal des Ports `FPGA_RESET_N` ermöglicht einen Reset des FPGA-Teils vom HPS aus und soll Ihren bestehenden Reset-Button ersetzen.

```
    -- RESET (for FPGA, generated by HPS)
    FPGA_RESET_N     : out   std_ulogic;
```

- Die beiden Ports `FPGA_UART_TX` bzw. `FPGA_UART_RX` entsprechen der FPGA-Seite des UART-to-USB-Adapters.

```
    -- UART (HPS LOAN I/O -> FPGA)
    FPGA_UART_TX     : in    std_ulogic;
    FPGA_UART_RX     : out   std_ulogic;
```

Als Tx-Gruppe können Sie das Signal `FPGA_UART_`**`RX`** als zu sendenden Bitstrom verwenden. Beachten Sie jedoch dass dieses Signal vom HPS kommt und daher asynchron zu Ihrem Design

ist. Es muss daher innerhalb des Testbeds auf Ihren **48-MHz**-Takt einsynchronisiert werden. Den Port `FPGA_UART_`**`TX`** sollten Sie konstant mit `'1'` treiben.

Als Rx-Gruppe können Sie das Signal `FPGA_UART_`**`RX`** ignorieren (`open`). Den Port `FPGA_UART_`**`TX`** können Sie mit dem Ausgang Ihres FSK-Empfängers treiben. Beachten Sie jedoch, dass die Wrapper-Unit an diesem Port ein Signal erwartet, dass synchron zu `CLOCK_50` ist. Sie müssen den Ausgang Ihres FSK-Empfängers daher zunächst innerhalb Ihres Testbeds auf den **50-MHz**-Takt einsynchronisieren.

- Den Port `MIRROR_HPS_LED` sollten Sie direkt mit einer der 10 LEDs (`LEDRx`) verbinden. Bei korrekter Konfiguration des HPS sollten Sie auf dieser LED dasselbe Signal sehen, das auch auf der HPS USER LED ausgegeben wird.

```
-- MIRRORING THE SIGNAL OUTPUT ON THE HPS LED
MIRROR_HPS_LED    : out    std_ulogic;
```

- Die Pins des HPS nach außen sind über die Ports `HPS_*` ausgeführt. Diese müssen daher 1:1 durch Ihr Testbed durchgeschleust werden und in die Pin-Konfiguration (`Config.tcl`) eingetragen werden. Die entsprechenden Einträge sollten bereits in der `Config.tcl`-Datei vorhanden sein.

```
-- HPS I/O PINS
HPS_DDR3_...
HPS_DDR3_RZQ      : in     std_logic;
HPS_ENET_...
HPS_FLASH_...
...
HPS_UART_TX       : inout  std_logic;
HPS_UART_RX       : inout  std_logic;
HPS_KEY           : inout  std_logic;
HPS_LED           : inout  std_logic;
...
```

Die in der Unit `HPSComputerMin` integrierte HPS-Konfiguration wurde mit dem Werkzeug Qsys erstellt. Mehr dazu erfahren Sie in der 9. Übung. In der Übungsangabe sind bereits alle notwendigen Dateien vorhanden. Sie müssen das Qsys-Projekt (zusätzlich zur Unit `HPSComputerMin` selbst) in Ihr Design als Foreign-Unit einbinden:

```
append ForeignUnits {
  {AlteraCycloneV HPSComputerMin Computer_System/synthesis/Computer_System.qip 0 QIP}
}
```

***Wichtig:*** Mit Quartus unter Windows kann es zu Problemen durch zu lange Pfadnamen kommen. Verwenden Sie das Skript `MapPathToDrive.bat` um das Verzeichnis `prjDsp` auf einen eigenen Laufwerksbuchstaben zu mappen.

Für das FSK-Modem wird nur die Loan-IO-Funktionalität des HPS benötigt um den UART-to-USB-Adapter nutzen zu können. Es ist daher lediglich ein Preloader notwendig, der das HPS entsprechend konfiguriert. Anschließend bleibt das HPS ungenutzt. Dazu ist in der Unit `HPSComputer-Min` ein vereinfachter Preloader inkludiert. Dieser konfiguriert das HPS so, dass die Loan-IO-Pins mit dem FPGA verknüpft werden, und bleibt anschließend in einer Endlosschleife stecken. Der Preloader wird nicht auf einer MicroSD-Karte gespeichert, sondern direkt in einem On-Chip-Memory (im FPGA) abgelegt. Dies geschieht mithilfe der Datei `Computer_System/synthesis/submodules/Computer_System_FPGA_Boot_SRAM.hex`, die ein FPGA-basiertes On-Chip-Memory initialisiert. Normalerweise würden Sie den Preloader mithilfe des Werkzeugs `bsp-editor` (dieses erreichen Sie über die SoC EDS Command Shell) erstellen und auf eine MicroSD-Karte spielen. Auch dazu erfahren Sie mehr in der 9. Übung.

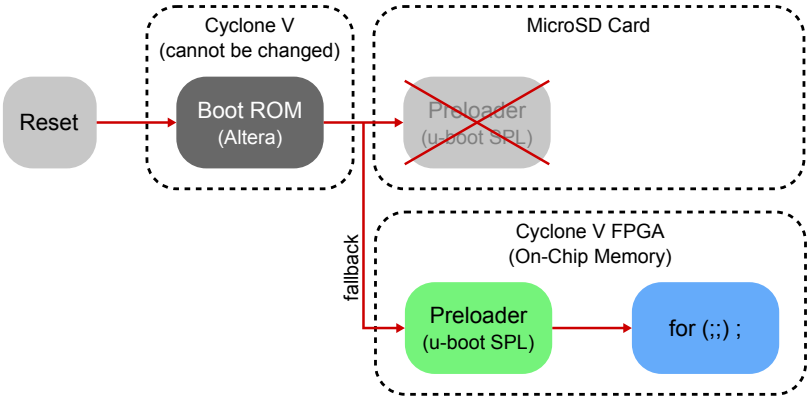



Abbildung 4: Bootprozess für die Unit `HPSComputerMin`.

Abbildung 4 zeigt den Bootvorgang für dieses System. Der Startup-Code im Boot-ROM sucht zunächst den Preloader auf einer MicroSD-Karte. Ist keine Karte eingelegt, dann bleibt das Boot-ROM in einer Endlosschleife stecken. Ist jedoch eine MicroSD-Karte eingelegt und ist darauf kein Preloader vorhanden, dann lädt das Boot-ROM den Preloader aus dem On-Chip-Memory im FPGA-Teil.

b☐1 Prüfen Sie die korrekte Funktion des Testbeds auf dem Rapid Prototyping Board. Arbeiten Sie dabei mit Ihrer Partner-Rx-Gruppe bzw. Partner-Tx-Gruppe zusammen[1]. Als Terminal-Programm für die serielle Schnittstelle kann beispielsweise HTerm[2] verwendet werden. Stellen Sie eine Datenrate von mindestens 190 Baud ein (geringer Datenraten werden ev. nicht unterstützt).

***Wichtig:*** Beachten Sie, dass das HPS nur dann konfiguriert wird, wenn eine MicroSD-Karte ***eingelegt*** ist. Diese MicroSD-Karte muss ***leer*** sein, damit auch wirklich der Preloader im On-Chip-Memory für den Bootvorgang herangezogen wird. Es empfiehlt sich daher die MicroSD-Karte vor der ersten Verwendung zu löschen[3].

***Tipp:*** Wenn HPS und FPGA korrekt konfiguriert wurden, dann müssen die HPS USER LED (LEDG7) ***und*** die mit dem Port `MIRROR_HPS_LED` verbundene LED (LEDRx) identisch mit folgendem Muster blinken: 700 ms aus, 300 ms an, 1500 ms aus, 300 ms an, 700 ms aus, 500 ms an, 900 ms aus, 100 ms an, 500 ms aus, 100 ms an, 900 ms aus, 500 ms an. Außerdem gehen die LEDs an, solange der Taster HPS USER BUTTON (KEY8) gedrückt ist.

c☐1 Wie haben sich die belegte Fläche und die maximale Taktfrequenz für das FPGA verändert?

**Abgabe des Sourcecodes**

Neben den regulären ***Commits*** im Laufe der Arbeit an jedem Übungszettel ist der Endstand jedes Übungszettels als Tag im Repository abzulegen. Jeder Tag hat dabei einen Namen der Form „ue-<Nr>“ zu tragen, wobei <Nr> die Nummer des jeweiligen Übungszettels ist.

a☐ Erstellen Sie bis spätestens ***23:59 Uhr am Vortag des Abgabetags*** (siehe *Semesterübersicht*) einen Tag `/tags/ue-08` mit dem Endstand Ihrer Übungsausarbeitung.

[1] Sie können zudem auch das unter `prjDsp/testComFsk/` zur Verfügung gestellte Referenzmodem für Ihre Tests heranziehen.

[2] Sie finden es in der .zip-Datei zu dieser Übung im Verzeichnis `tools/HTerm`.

[3] Sie können dazu beispielsweise den *SD Memory Card Formatter* (siehe Abschnitt *Tools für SD-Karten* im E-Learning-Kurs) verwenden oder das bereitgestellte Image einer leeren SD-Karte mit dem *Win32 Disk Imager* auf Ihre SD-Karte spielen.