1. Übung: Codec: Control Interface (24 P)

Sister, where's that noise comin' from?

Gesamtsystem *Ein Überblick*

Im Rahmen dieser Übung werden Sie ein FSK-Modem auf Basis des DE1-SoC Boards aufbauen. Abb. 1 zeigt die gesamte Übertragungsstrecke. Sie arbeiten dazu in Teams zu zwei (bzw. drei) Personen jeweils an einem FSK-Sender oder einem FSK-Empfänger. Der Sender erzeugt anhand eines 1-Bit-Eingangssignals (serielle Schnittstelle) zwei unterschiedliche Frequenzen. Der Empfänger detektiert diese zwei Frequenzen und decodiert diese wiederum in einen seriellen Bitstrom.

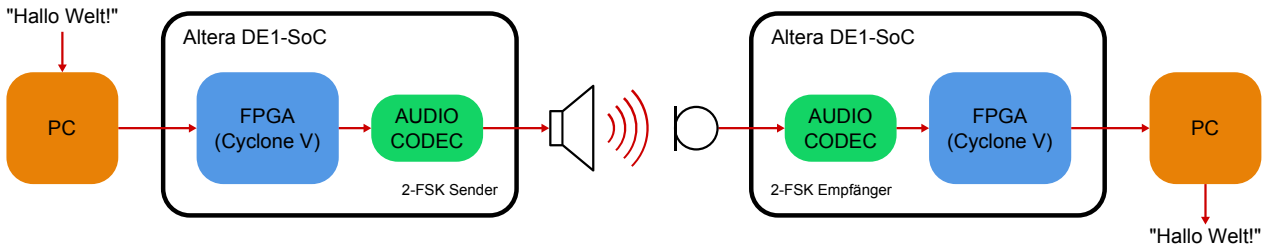

Abbildung 1: Gesamtsystem

1. Aufgabe *Teams, Subversion, fhlow, Material*

Innerhalb einer Übungsgruppe sollten gleich viele Tx- und Rx-Teams gebildet werden. Jedes Team kann zwei Personen aufnehmen. Die Teamaufteilung kann bei Bedarf (z.B. ungleiche Anzahl an Tx- und Rx-Teams) vom Übungsleiter angepasst werden.

a☐ Sprechen Sie die Teameinteilung untereinander ab und tragen Sie sich bis zum Ende der ersten Übungseinheit in der Gruppeneinteilungsliste ein.

Subversion

Der Sourcecode der Übung in SCD5 wird für jedes dieser Teams in einem Subversion-Repository verwaltet. Für die Zusammenarbeit innerhalb der Teams und um Änderungen sinnvoll nachvollziehen zu können ist es unumgänglich den Zwischenstand regelmäßig durch einen ***commit*** im Repository abzuspeichern. Es hat sich als hilfreich erwiesen sprechende und verständliche *Commit-Messages* zu verwenden.

Wichtig! Beachten Sie unbedingt auch die Hinweise in den Übungsrichtlinien!

Da das Repository erst in den nächsten Tagen zur Verfügung stehen wird, bleibt noch etwas Zeit, um eine langfristig ausgerichtete Verzeichnisstruktur zu erstellen. Zwar wäre eine Änderung der Verzeichnisstruktur auch später noch möglich, es sprechen jedoch einige Gründe dagegen, auf eine solche Änderung zu bauen:

- Warum sollte ein Projekt, das chaotisch anfängt aufgeräumt enden?
- Zwar *könnte* man die Verzeichnisstruktur auch später noch ändern, aber keiner *tut* es.
- Je später eine Änderung durchgeführt wird, desto mehr Arbeit ist damit verbunden. Der Wille, die Änderung tatsächlich durchzuführen, sinkt demnach mit der Zeit stark ab. Nach wenigen Wochen ist meist schon der Nullpunkt erreicht.

Für diese Übung ist folgende grundlegende Verzeichnisstruktur zu verwenden:

```
<repository>/
  └ tags/  ..... Verzeichnis für Subversion-Tags (siehe Übungsabgabe)
  └ trunk/  ..... Verzeichnis für aktive Entwicklung der Übung[1]
     └ doc/  ..... Verzeichnis für Übungsangaben und Ausarbeitungen
     └ literature/  ..... Verzeichnis für Literatur
     └ matlab/  ..... (optionales) Verzeichnis für Systemsimulation (Matlab)
     └ prjDsp/  ..... Verzeichnis für fhlow-Projekt
```

b☐1 Erstellen Sie diese Verzeichnisstruktur in Ihrem Subversion-Repository. Anschließend können Sie das Verzeichnis `/trunk` als *Working-Copy* aus Ihrem Repository auschecken[2].

fhlow

Zur Steuerung von Simulation und Synthese in dieser Übung ist die Skriptingumgebung *fhlow*[3] zu verwenden. Das Zip-Archiv für die erste Übung enthält alle Dateien, welche für die Verwendung von *fhlow* nötig sind. Um die Arbeit mit *fhlow* zu erleichtern werden mit den Übungsangaben für viele Units auch bereits fertige Konfigurationsskripts zur Verfügung gestellt.

Eine Vorlage für die Verzeichnisstruktur einer *Unit* in *fhlow* finden Sie im Verzeichnis `unitTemplate`, welches wiederum innerhalb der *Group* `grpTemplates` gelegen ist.

c☐2 Machen Sie sich mit der Funktionsweise von *fhlow* vertraut. Ein funktionsfähiges Beispiel finden Sie in der `unitExample1` in der `grpExamples`. Achten Sie insbesondere auf

- den Aufbau der Verzeichnisstruktur von *fhlow*. Das Gesamtprojekt wird in Groups unterteilt. Jede Group fasst ein oder mehrere Units und Packages zusammen.

  Projekt `prjDsp/`

  - `fhlow/` ..... Skriptingumgebung *fhlow* (globale Build-Skripts)
  - `Config.tcl` ..... globale Konfigurationsdatei
  - `TimingConstraints.sdc` ..... globale Timing-Constraints
  - `Paths.bat` bzw. `Paths.config` ..... Skripts zur Ermittlung der Toolchain

    Tipp: Die Programmpfade der Toolchains sollten für jeden Benutzer separat konfiguriert werden. *fhlow* verwendet dazu entsprechende Konfigurationsskripts außerhalb des Projektverzeichnisses:

    – Windows: `%USERPROFILE%\.fhlow\Paths.bat`; diese Datei wird beim ersten Start von *fhlow* automatisch erstellt.

[1] ***Tipp:*** Die Zip-Archive zur Übung sind immer so aufgebaut, dass Sie diese direkt in diesem Verzeichnis extrahieren können.

[2] ***Tipp:*** Es bietet sich an, wirklich nur das Verzeichnis `/trunk` auszuchecken. Würden Sie das gesamte Repository auschecken, ließe das Verzeichnis `/tags` die Größe der *Working-Copy* rasch enorme Ausmaße annhemen.

[3] https://github.com/michaelroland/fhlow

  - Linux: `/.fhlow/Paths.config`; kopieren Sie dazu einfach die gleichnamige Datei aus dem Projektverzeichnis und adaptieren Sie diese.
- Group `grp.../` ..... Groups fassen Units und Packages zusammen
  - `Config.tcl` ..... (optionale) Konfigurationsdatei für alle Module einer Group
  - `TimingConstraints.sdc` ..... (optionale) Timing-Constraints für alle Module einer Group
  - Unit `unit.../` ..... einzelnes VHDL-Modul (Entity mit einer oder mehreren Architectures und einer entsprechenden Testbench)
    - `Config.tcl` ..... Konfigurationsdatei für diese Unit
    - `TimingConstraints.sdc` ..... (optionale) Timing-Constraints für diese Unit
    - `src/` ..... Verzeichnis für Sourcecode-Dateien dieser Unit
    - `flw/`
      - `simQuestasim/` ..... Verzeichnis für Simulationssteuerung
        - `*.bat` ..... Batchfiles für Steuerung unter Windows
        - `Makefile` ..... Makefile für Steuerung unter Linux
        - `Wave.do` do-File zur Konfiguration der Waveform-Aufzeichnung
      - `synlayQuartus/` ..... Verzeichnis für Synthesesteuerung
        - `*.bat` ..... Batchfiles für Steuerung unter Windows
        - `Makefile` ..... Makefile für Steuerung unter Linux
        - `MyAddons.tcl` ..... zusätzliche Einträge für die QSF-Datei
        - `MyPostprocessing.tcl` ... wird nach der Synthese ausgeführt
        - `synlayResults/` . automatisch generiertes Verzeichnis mit dem generierten Quartus-Projekt und den Syntheseergebnissen
  - Package `pkg.../` ..... einzelnes VHDL-Package
    - `src/` ..... Verzeichnis für Sourcecode-Dateien dieses Package

- die Namensgebung für Dateien.
  - Package: *`Package`*`-p.vhd`
  - Entity: *`Unit`*`-e.vhd`
  - Architecture: *`Unit-Arch`*`-a.vhd`
  - Kombination aus Entity und Architecture: *`Unit-Arch`*`-ea.vhd`
  - Testbench: `tb`*`Unit-Arch`*`-ea.vhd`
- die Konfigurationsdateien und deren Aufbau.
  - auf Benutzerebene: `Paths.[bat|config]`
  - auf Projekt-, Group- und Unit-Ebene: `Config.tcl`, `TimingConstraints.sdc`
  - im Simulationsverzeichnis: `Wave.do`
  - im Syntheseverzeichnis: `MyAddons.tcl`, `MyPostprocessing.tcl`
- die Batch-Dateien (für Windows) bzw. das Makefile (für Linux). Die gebräuchlichsten Befehle sind dabei
  - `CompSimQuestasimGui`, zum Starten des Kompilieren und der Simulation mit Questasim/Modelsim, und
  - `SynLayQuartusGui`, zum Starten der vollständigen Synthese mit Quartus.
- Stellen Sie sicher, dass die richtigen Toolchain-Pfade für Ihr System eingestellt sind.

Material

Neben einem DE1-SoC Rapid Prototyping Board brauchen Sie für die Durchführung der Übungen innerhalb jedes Teams noch

- eine Micro-SD-Karte,
- einen USB-Soundadapter,
- ein Verbindungskabel mit beidseitig 3,5 mm-Klinkenstecker,
- zwei Verbindungskabel mit einseitig 3,5 mm-Klinkenstecker und einseitig Cinch-Steckern,
- Mikrofon mit 3,5 mm-Klinkenstecker,
- Stereokopfhörer mit 3,5 mm-Klinkenstecker, und
- ein DE1-SoC Board.

d☐ Die USB-Soundadapter, die Verbindungskabel, das Mikrofon, und die Micro-SD-Karte werden vom Studiengang zur Verfügung gestellt. Holen Sie sich diese Komponenten bitte *am Ende* der Übungseinheit beim Übungsleiter ab.

Wichtig! Es wird von Ihnen erwartet, dass Sie pro Team stets einen Satz dieser Ausrüstung in *jeder* SCD5-Übungseinheit bereitstehen haben.

2. Aufgabe *I^2C-Schnittstelle zur Parametrisierung des Codec: FSMD-Modellierung*

Ein Audio-*Codec* (von „*cod*er“ / „*dec*oder“) stellt die Kombination aus einem Analog-Digital-Umsetzer (ADC) und einem Digital-Analog-Umsetzer (DAC) in *einem* Gehäuse dar. Solche Bausteine werden beispielsweise in Smartphones und MP3-Playern zur Ein- und Ausgabe von Audiosignalen eingesetzt. Häufig sind auf dem Codec-IC auch analoge Zusatzfunktionen, wie Kopfhörer- und Mikrofonverstärker, untergebracht.

Die Funktionen, welche von solchen Bausteinen angeboten werden, können meist digital eingestellt (parametriert) werden. Beispiele sind passende Filtereinstellungen für eine vorgegebene *Sampling Rate* von ADC und DAC oder die Lautstärkeeinstellung des Kopfhörerverstärkers. Alle Parameter werden über ein *Control-Interface* auch eingestellt. Um Gehäuseanschlüsse einzusparen und das Platinenlayout zu vereinfachen, ist dieses Control-Interface zumeist eine serielle Schnittstelle. Im vorliegenden Fall folgt diese Schnittstelle dem weit verbreiteten I^2C-Protokoll (detaillierte Informationen zum I^2C-Bus finden Sie im Verzeichnis `literature/Philips/`).

Es soll nun ein synthesefähiges FSMD-Modell erstellt werden, welches einen vorgegebenen Parametersatz über die I^2C-Schnittstelle in einem Audio-Codec vom Typ *Wolfson WM8731* ablegt. Das verwendete I^2C-Protokoll ist in Abb. 2 dargestellt.

Das Datenblatt[4] des Codec findet sich im Verzeichnis `literature/Wolfson/`. Für die Implementierung sind insbesondere die Abschnitte „SOFTWARE CONTROL INTERFACE“ (Seite 46) und „MPU INTERFACE TIMING“ (Seite 19) interessant. Beachten Sie, dass der Codec zwei verschiedene Protokolle für das Control-Interface unterstützen würde, wobei wir uns auf das I^2C-Protokoll („2-WIRE SERIAL CONTROL MODE“) beschränken.

Der Codec speichert die Parameter in 10 Registern von je 9 Bit Breite. Die Register innerhalb des Codec werden mittels der Adressen 0 bis 9 angesprochen. In VHDL lässt sich der Parametersatz mittels Records auf einem hohen Abstraktionslevel darstellen. Im Package `ParamCodec` (in der Group `Parameterize`) sind bereits einige fertige Parameterkonfigurationen definiert. Das Package `DefinitionsCodec` (in der Group `Wm8731`) gibt dazu eine entsprechenden Struktur aus Records sowie Methoden, um diese abstrakten Parametersätze in für den Codec verständliche Registerwerte zu konvertieren, vor.

Die Registerwerte selbst können in VHDL mithilfe einer *Constant* (`cRegBitMap`) in tabellenartiger Form (Array aus `std_ulogic_vector`) modelliert werden. Die zu entwickelnde FSMD soll den ge-

[4] Teilen Sie sich die Arbeit innerhalb des Teams auf. Arbeiten brauchen nicht doppelt erledigt zu werden. Es ist daher sinnvoll, wenn sich nur ein Miglied des Teams intensiv mit dem Datenblatt auseinandersetzt.

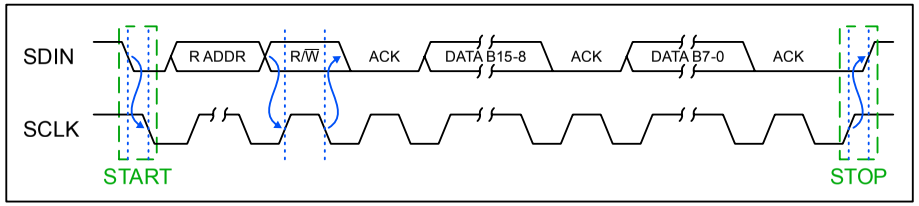

Abbildung 2: I²C-Timing laut Datenblatt des WM8731. `Sdin` wird immer mit der fallenden Flanke von `Sclk` geschrieben und mit der steigenden Flanke von `Sclk` gelesen.

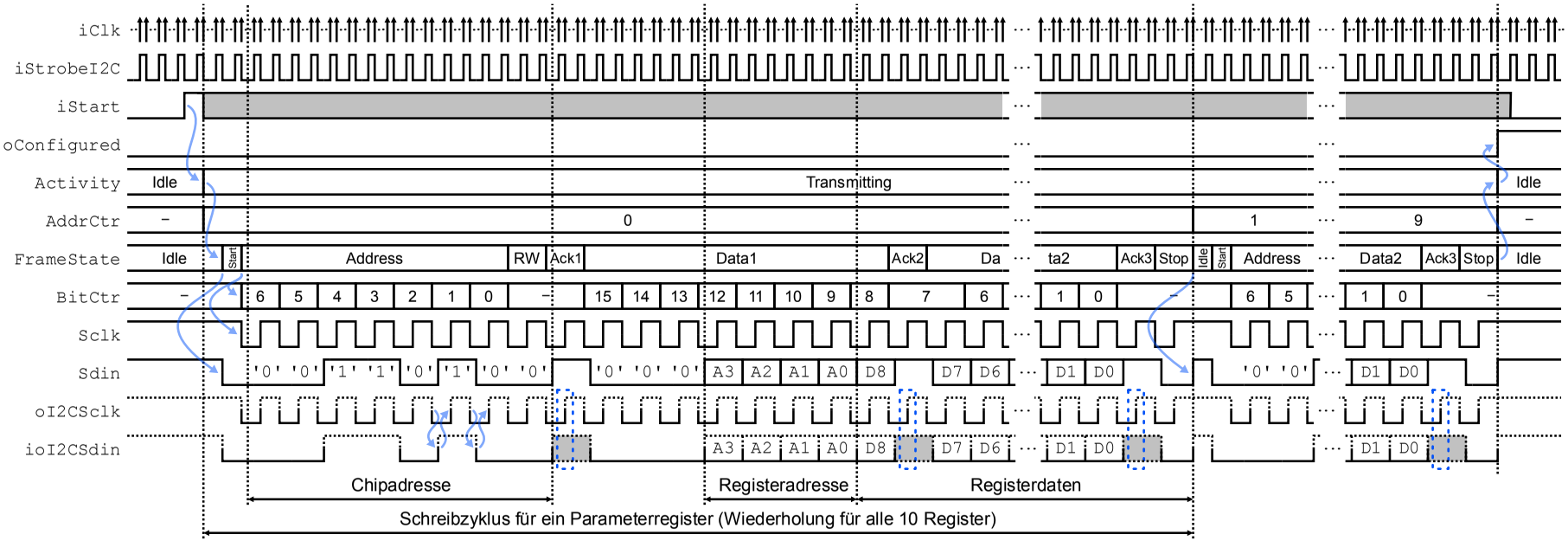

Abbildung 3: I²C-Timing zum Schreiben in die Parameterregister des *Codecs* WM8731. Ein vorangestelltes `i` kennzeichnet Eingangssignale, ein `o` Ausgangssignale und ein `io` bidirektionale Signale. `iClk` hat eine Frequenz von 48 MHz. Das Signal `iStrobeI2C` liefert einen regelmäßigen *Strobe*-Impuls der Länge einer Periode von `iClk` und einer Frequenz von 750 kHz. Daraus wird ein Signal `Sclk` mit einer Frequenz von 375 kHz erzeugt. Beachten Sie, dass die schreibende *Unit* das Datensignal `Sdin` nur während der *Low*-Phase von `Sclk` ändern darf. Während der gesamten *High*-Phase von `Sclk` muss `Sdin` stabil anliegen. Während der ACK-Phasen zieht *das Codec* das Signal `ioI2CSdin` auf *Low*. Ihre *Unit* kann dieses Signal bei der steigenden Flanke von `Sclk` detektieren.

gebenen Parametersatz mittels des I²C-Protokolls in die entsprechenden Register des Codec schreiben.

Um die Modellierung als FSMD zu erleichtern, sind in Abb. 3 bereits einige Signals zur Darstellung des Zustands und der Ausgabewerte zu finden: `Activity`, `AddrCtr`, `FrameState`, `BitCtr`, `Sclk`, `Sdin` und `oConfigured`. Der Ablauf des Schreibvorgangs für *ein* Register ist hervorgehoben. Für jedes Register werden 3 mal 8 Bit übertragen. Der Empfang wird jeweils durch ein Acknowledge-Bit vom WM8731 bestätigt. Mit den ersten 8 Bit (***Address***) wird der Chip adressiert. Die Chipadresse ist für jedes Register `'0011010'` gefolgt vom R/$\overline{\text{W}}$-Bit `'0'`. Die nächsten 8 Bit (***Data1***) bestehen aus drei Füllbits `'000'`, der Registeradresse `A(3 downto 0)` und dem höchsten Bit `D(8)` der Registerdaten. Die letzten 8 Bit (***Data2***) sind die übrigen Datenbits `D(7 downto 0)`.

a ☐ 10 Erstellen Sie eine synthesefähige Unit `ConfigureCodecViaI2c(Rtl)` mit folgender Schnittstelle:

```
generic (
  gChoosenParamSet : aParamSetName := MicroOverSidetone;
  gI2cAddress      : std_ulogic_vector(6 downto 0) := "0011010");
port (
  iClk          : in    std_ulogic;
  inResetAsync  : in    std_ulogic;
  iStrobeI2C    : in    std_ulogic;
  iStart        : in    std_ulogic;
  oConfigured   : out   std_ulogic;
  oAckError     : out   std_ulogic;
  oI2cSclk      : out   std_ulogic;
  ioI2cSdin     : inout std_logic);
```

Ein entsprechendes Grundgerüst findet sich bereits in der Unit `ConfigureCodecViaI2c` (in der Group `Wm8731`). Dort ist lediglich der kombinatorische Teil der FSMD zu ergänzen.

Das Strobesignal `iStrobeI2C` gibt den Takt von `Sclk` vor. Die Zustandsübergänge der FSMD werden daher typischerweise immer bei diesem Strobe, d.h. bei `(iStrobeI2C = cActivated)`, erfol-
b ☐ 3 gen. Warum wurde die Frequenz für das Strobesignal `iStrobeI2C` mit 750 kHz festgelegt? Welche Frequenz und welches Tastverhältnis darf `Sclk` maximal haben?

Beachten Sie, dass es sich bei I²C um ein Bussystem handelt, bei dem die Taktleitung `oI2cSclk` und die Datenleitung `ioI2cSdin` von mehreren Teilnehmern lesend und schreibend verwendet werden können. Der „Trick“ dabei ist, dass die beiden Leitungen mit einem externen Pull-up-Widerstand auf dem High-Pegel gehalten werden. Schreibende Busteilnehmer sind über einen Tri-State-Buffer an die Busleitungen angeschlossen. Sie können nun einen Low-Pegel am Ausgang anlegen um einen Low-Pegel (logisch `'0'`) auf der Signalleitung zu erhalten oder den Ausgang hochohmig halten um einen High-Pegel (logisch `'1'`) auf der Signalleitung zu erhalten.

Im Grundgerüst für die Unit `ConfigureCodecViaI2c(Rtl)` wurden diese Tri-State-Buffer bereits mit folgenden Concurrent-Statements eingebaut:

```
oI2cSclk  <= '0' when (R.Sclk = cInactivated) else 'Z';
ioI2cSdin <= '0' when (R.Sdin = cInactivated) else 'Z';
```

In der Simulation müssen daher auch entsprechende Pull-up-Widerstände auf den I²C-Leitungen modelliert werden. In der Testbench lässt sich dies einfach mit folgenden Concurrent-Statements erreichen:

```
I2cSclk <= 'H';
I2cSdin <= 'H';
```

Weil somit mehrere Treiber (einerseits die Unit `ConfigureCodecViaI2c` und andererseits das Pull-Up-Modell) auf diese Signals schreiben, ist es notwendig, dass für diese ein Resolved-Type (z.B. `std_logic`) verwendet wird.

c☐1 Zur Verifikation Ihrer Beschreibung steht eine fertige Testbench zur Verfügung. Weisen Sie die korrekte Funktion Ihrer FSMD-Beschreibung mithilfe dieser Testbench nach. Testen Sie Ihre Unit `ConfigureCodecViaI2c(Rtl)` ausführlich. Neben der Simulation kann die Synthese bei der Kontrolle der Beschreibung bereits sehr hilfreich sein.

Tipp: Passen Sie bei Bedarf die Datei `grp.../unit.../Config.tcl` und das Skript `grp.../unit.../flw/simQuestasim/Wave.do` zur automatisierten Simulationssteuerung der Unit an.

Übrigens können Sie das Format einer Waveform direkt aus QuestaSim heraus als .do-Skript abspeichern: Wählen Sie im Waveform-Fenster den Menüpunkt *File* → *Save Format...* (bzw. klicken Sie einfach auf das Diskettensymbol in der Toolbar).

3. Aufgabe *Codec-Test*

Bevor Sie Ihre eigene Konfigurationsunit auf dem DE1-SoC Board ausprobieren, sollten Sie sicherstellen, dass der Audio-Codec auf Ihrem Board auch einwandfrei funktioniert. Im Verzeichnis `prjDsp/testAlteraDE1SoCAudio` dieser Übung findet sich ein Design, welches Sie zum Testen des Codec verwenden können. Die Designs `Mic*_AlteraDE1SoC_V12.sof` übertragen die Signale vom Mikrofon (Anschluss MIC IN) auf den Kopfhörerausgang (Anschluss LINE OUT).

a☐ Schließen Sie einen Kopfhörer und ein Mikrofon Ihr DE1-SoC Board an, konfigurieren Sie das FPGA mit einer der Dateien `Mic*_AlteraDE1SoC_V12.sof`, und testen Sie Ihr Board indem Sie z.B. über das Mikrofon streichen. Das entstehende Kratzgeräusch ist im Kopfhörer deutlich(!) wahrnehmbar.

4. Aufgabe *Testbed für ConfigureCodecViaI2c*

Wichtig! Stellen Sie zunächst sicher, dass Ihre Beschreibung von `ConfigureCodecViaI2c(Rtl)` in der Simulation einwandfrei funktioniert. Erst wenn Sie sich dessen wirklich sicher sind, sollten Siemit dieser Aufgabe beginnen. Bedenken Sie, dass *sämtliche* folgenden Beschreibungen die Unit `ConfigureCodecViaI2c` einbinden werden und daher von deren Funktion abhängig sind. Fehlfunktionen dieser Unit machen sich zum Teil erst sehr spät bemerkbar!

Für den Test auf dem Rapid Prototyping Board steht ein fertiges Testbed zur Verfügung. Die Grundstruktur dieses Testbeds ist in Abb. 4 dargestellt. In der Abbildung fehlen jedoch noch die wesentlichen Steuersignale zwischen den einzelnen Komponenten des Testbeds.

a☐3 Machen Sie sich mit der Implementierung des Testbeds `TbdConfigureCodecViaI2c` vertraut und ergänzen Sie die Abbildung um die entsprechenden Steuersignale[5] .

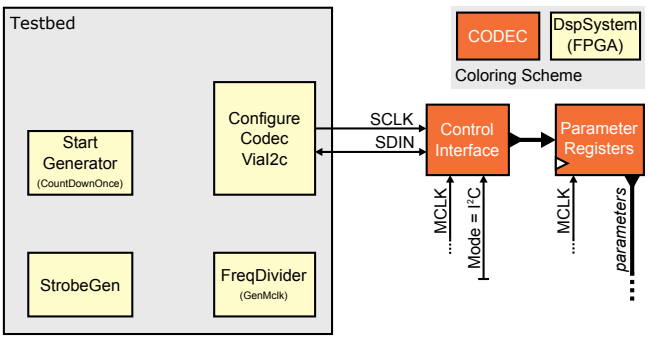

Abbildung 4: Testbed `TbdConfigureCodecViaI2c`

Die Architecture des Testbeds besteht im Wesentlichen aus Instantiations der Konfigurationsunit, eines Strobe-Generators und eines Frequency-Dividers. Letzterer versorgt (vorerst) den Codec über den Ausgang `oMclk` mit einem symmetrischen Taktsignal mit einer Frequenz von 12 MHz.

[5] ***Tipp:*** Zeichnen Sie das Testbed auch wirklich in der Abbildung ein um ausreichend für die Übungsbesprechung vorbereitet zu sein!

Zudem muss das Startsignal `iStart` innerhalb des Testbed erzeugt werden. Der Start-Generator (Process `CountDownOnce`) wartet dazu nach dem Reset einfach einige Impulse des I^2C-Strobe-Signals ab, bevor mit der Konfiguration des Audio-Codecs begonnen wird.

Das FPGA auf dem DE1-SoC Board wird von einer externen Taktquelle mit Taktsignalen versorgt. Alle verfügbaren Taktsignale haben eine Frequenz von 50 MHz (Pins `AF14`, `AA16`, `Y26` und `K14`). Sie möchten Ihr Design jedoch mit 48 MHz betreiben. Dieser Takt ist notwendig um später den Codec mit einem sauberen, symmetrischen 12 MHz Takt zu versorgen.

Das FPGA besitzt sechs PLLs, die aus dem externen Grundtaktsignal in weiten Grenzen frei wählbare Frequenzen als Systemtakt für Ihr Design erzeugen können. Die PLLs können Sie in Form der *Altera PLL Megafunction* über den *IP Catalog* von Quartus Prime in Ihr Design einbinden.

Sie müssen daher den 50 MHz Takt von Pin `AF14` mit Hilfe einer PLL auf 48 MHz verringern. Eine fertig vorkonfigurierte PLL finden Sie in der Unit `PLL50to48` (in der Group `AlteraCycloneV`). Um die Simulation einfacher zu gestalten wurde diese PLL in einer Unit gekapselt, welche die PLL für die Simulation ausblendet.

b☐3 Machen Sie sich mit der Funktionsweise der Unit `PLL50to48` vertraut und fügen Sie diese in das Testbed mit ein. Die Unit verwendet den 50 MHz Takt vom Eingang `iClk` des Testbeds und erzeugt am Ausgang `oClk48MHz` einen Takt mit 48 MHz. Diesen Takt verwenden Sie für ***alle*** anderen Units und Register die in Ihrem Testbed eingebettet sind.

Wichtig! Verwenden Sie das Taktsignal `iClk` *nur* um die PLL zu betreiben. Alle anderen Komponenten Ihres Designs müssen unbedingt mit dem Ausgangssignal der PLL (`oClk48MHz`) getaktet sein. Andernfalls führt die Verwendung mehrerer Clock-Domains ohne Synchronisation zu Timing-Problemen!

Tipp: Die mit dem *MegaWizard* generierte Include-Datei für den Altera PLL IP Core finden Sie unter `grpAlteraCycloneV/unitPLL50to48/megafunction/AlteraPLL50to48.qip`. In fhlow können Sie diese mit folgenden Zeilen in Ihrer `Config.tcl` einbinden:

```
append ForeignUnits {
    {AlteraCycloneV PLL50to48 megafunction/AlteraPLL50to48.qip 0 QIP}
}
```

Die Wrapper-Unit `PLL50to48` können Sie mit folgenden Zeilen in Ihre fhlow-Umgebung einbinden:

```
append Units {
    {AlteraCycloneV PLL50to48 Inst}
}
```

c☐1 Testen Sie Ihren Entwurf auf der Hardware. Als Einstellungen für den Codec verwenden Sie den Parametersatz `MicroOverSidetone`. Dieser stellt den Codec folgendermaßen ein:

- Der Mikrofoneingang wird aktiviert und als *Sidetone* mit einer Dämpfung von -6 dB auf den Ausgang geführt.
- Die Kopfhörerausgänge werden mit einer Lautstärkeeinstellung von jeweils 0 dB betrieben.
- ADC und DAC, sowie deren Signalein- und Signalausgänge, sind deaktiviert.
- Die *Power-Down*-Einstellungen sind so eingestellt, dass nur der Signalpfad vom Mikrofon über den Sidetone zu den Kopfhörern eingeschaltet ist.
- Der Codec wird im *Slave Mode* betrieben (siehe Register *Digital Audio Interface Format*).

Streicht man mit dem Finger über das Mikrofon, so ruft dies ein (recht lautes) Kratzen im Kopfhörer hervor.

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-01` mit dem Endstand Ihrer Übungsausarbeitung. Gehen Sie dabei folgendermaßen vor:

a) Fügen Sie alle wesentlichen Dateien zu Ihrer Working-Copy hinzu (***add***).

b) Checken Sie den Endstand Ihrer Working-Copy (d.h. des Verzeichnisses `/trunk`) im Repository ein (***commit***). Wenn Sie *TortoiseSVN* nutzen, können Sie dazu den Menüpunkt „SVN Commit...“ im Kontextmenü des Windows Explorer verwenden. Alternativ können Sie den Befehl

```
svn commit -m "Eine Commit-Message"
```

verwenden.

c) Erstellen Sie einen Tag mit dem Namen `/tags/ue-01` von ihrer Working-Copy. Wenn Sie *TortoiseSVN* nutzen, können Sie dazu den Menüpunkt „TortoiseSVN → Branch/tag...“ im Kontextmenü des Windows Explorer verwenden (siehe Abb. 5). Alternativ können Sie den Befehl

```
svn copy https://svn01.fh-hagenberg.at/hsd/SCD5G1AbcDef/trunk \
         https://svn01.fh-hagenberg.at/hsd/SCD5G1AbcDef/tags/ue-01 \
         -m "Endstand der 1. Übung"
```

verwenden.

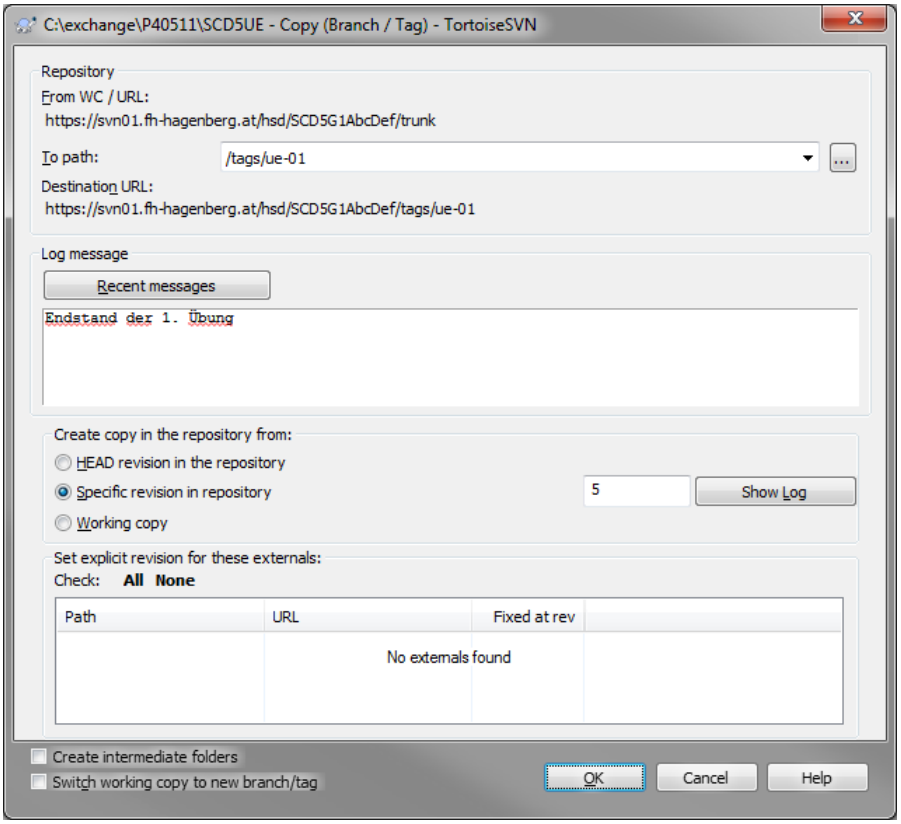

Abbildung 5: Subversion: Erstellen eines Tags

Sollten Sie nach dem Anlegen eines Tags feststellen, dass Sie noch Fehler in Ihrer Abgabe ausbessern müssen, dann können Sie einen weiteren Tag mit einem Namen der Form „ue-<Nr>-<Zahl>“ anlegen. In diesem Fall wird nur jener Tag mit der größten <Zahl> berücksichtigt.