Anleitung bis zu Milestone 1
Synthesizer Projekt

1. Aufgaben für die Codec_Controller Design-Person

1. Entwerfen Sie den VHDL Code für den Codec Controller in einer Datei *codec_controller.vhd* sowie in DTP2-Vorlesung_04 S.21 ff beschrieben.

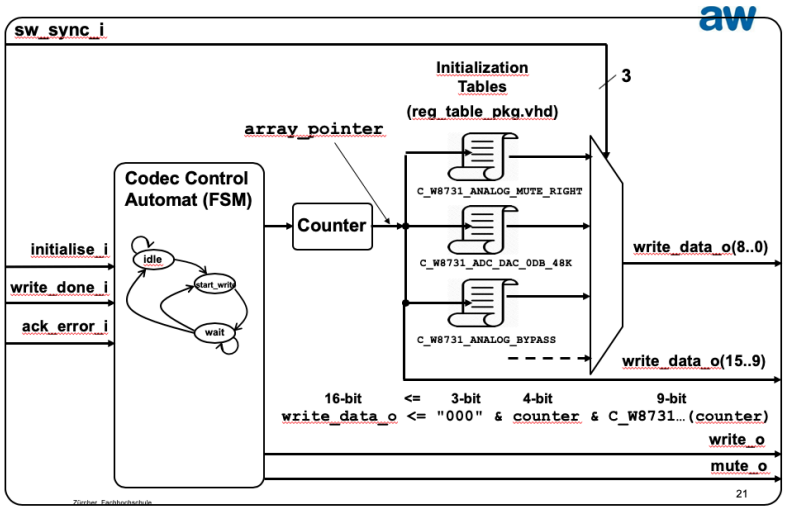

Abbildung 1: Block Diagramm von codec_controller.vhd

2. Die Signale *sw_sync_i* (von Schalter (SW2..0)) sollen die Codec Register entsprechend der folgenden Tabellen initialisieren. Die Initialisierungs-Tabellen sind in *reg_table_pkg.vhd* (herunterladen von OLAT MS1) beschrieben.

SW(2)	SW(1)	SW(0)	Codec Initialisierungs-Tabelle
0	0	1	C_W8731_ANALOG_BYPASS
1	0	1	C_W8731_ANALOG_MUTE_LEFT
0	1	1	C_W8731_ANALOG_MUTE_RIGHT
1	1	1	C_W8731_ANALOG_MUTE_BOTH
x	x	0	C_W8731_ADC_DAC_0DB_48K

3. Synthetisieren und überprüfen Sie *codec_contoller.vhd* mit Quartus bevor Sie den Entwurf an die Top-Level-Design-Person weitergeben. Prüfen Sie das RTL Diagramm auf Latches.

2. Aufgaben der Top-Level-Design-Person

1. Setzen Sie eine neue Projektdatei *synthi_project* auf. Simulations wird von der Testperson populiert. Source enthält sämtliche vhdl Dateien die alle Gruppenmitglieder generieren.

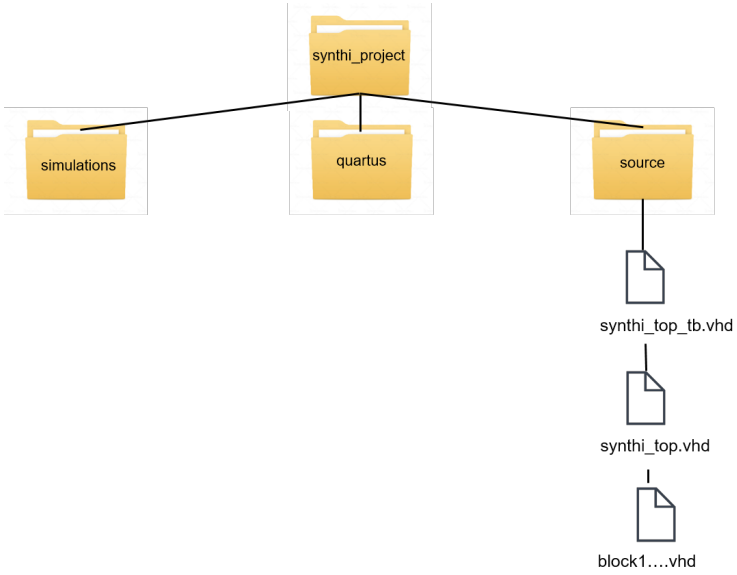

Abbildung 2: Verzeichnisse im Projekt

2. Erstellen Sie den Infrastructure-Block wie unten gezeigt.

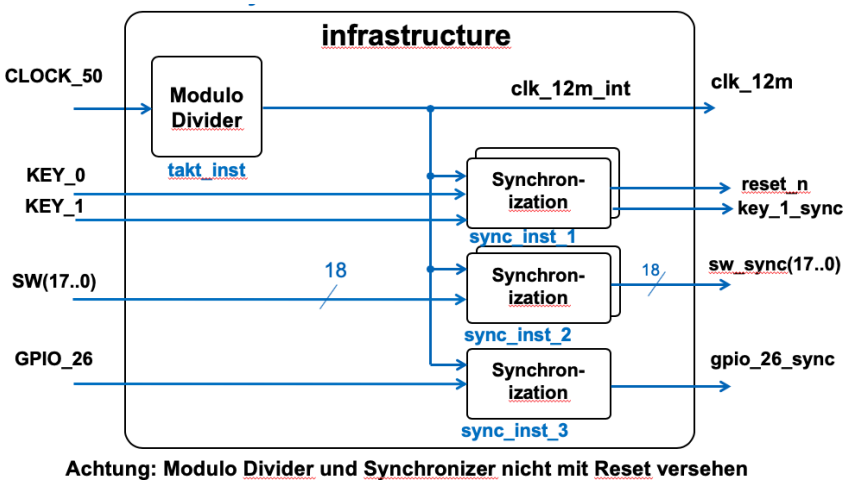

Abbildung 3: Infrastrukture Block

3. Benutzen Sie als Basis für die Synchronisierung *synchronize.vhd* (im OLAT). Erweitern Sie den Infrastructure-Block mit Synchronisierern für SW(17..0), KEY_1 und einem Modulo-Divider, der den Takt von 50Mhz auf 12.5 MHz herunter teilt.

   Hinweis: Der Synchronizer lässt sich mit einem *Generic* skalieren, so dass Sie für sw_sync(17..0) den Block nur einmal einbauen müssen.
4. Einen *modulo_divider.vhd* finden Sie in OLAT/Laborübungen/Milestone 1 zum Herunterladen. Benutzen Sie "Generic-Map" in *infrastructure.vhd*, um die Ausgangsfrequenz des Modulo-Dividers zu bestimmen.
5. Kopieren Sie sich aus OLAT die Datei *synthi_top.vhd*. Die "Architecture" ist noch leer. Integrieren Sie *infrastructure.vhd* in *synthi_top.vhd* mit Hilfe von emacs, wie in Lab1.
6. Erstellen Sie ein neues Quartus Projekt *„synthi_top"* im Verzeichnis *quartus.* Laden Sie *DE2_115_pin.csv* aus *Downloads* herunter und importieren Sie die Pin-Assignments in das Quartus Projekt.
7. Synthetisieren sie das Projekt, bis es keine Fehler mehr gibt. Prüfen Sie mit dem RTL Viewer ob der Infrastucture-Block ohne Latches implementiert wurde.
8. Sobald der Codec-Controller fertig ist, integrieren Sie diesen zusammen mit *i2c_master.vhd* ebenfalls in *synthi_top.vhd*

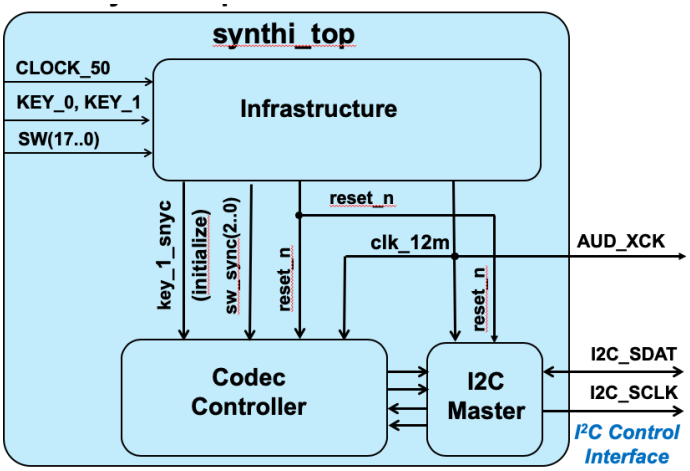

Abbildung 4: synthi_top bis MS1

3. Aufgaben der Testperson

1. Laden Sie das Verzeichnis *simulations* aus OLAT herunter, entpacken Sie es und platzieren Sie *simulations* im Verzeichnis *synthi_project (siehe* Abbildung 2*).*

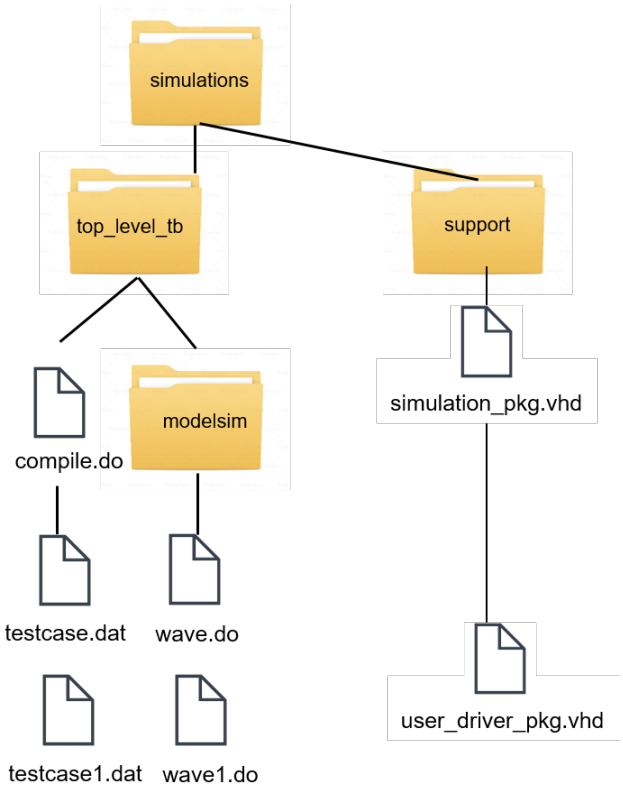

Abbildung 5: Aufbau des Simulationsverzeichnisses

2. Der Verzeichnisname der Testbench wie er im Download enthalten ist, soll in top_level_tb umbenannt werden.

   Vervollständigen Sie *compile.do* mit allen VHDL Dateien, die im Verzeichnis *source* vorhanden sind. Achten Sie auf den richtigen relativen Pfad zu den VHDL Dateien. Da VHDL hierarchisch aufgebaut ist, müssen zuerst die Libraries, dann die Source VHDL Dateien kompiliert werden. Gehen Sie vor wie in Lab2.

   Passen Sie ein *testcase.dat an.* Die erste Zeile sollte *rst_sim* sein, gefolgt vom Befehl *ini_cod arg1.* arg1 repräsentiert das *SW(2..0)* Signal, durch welches die Initialisierungstabelle bestimmt wird.
   Ein Template *testcase.dat* für die Test-Befehle befindet sich ebenfalls im OLAT (Vorsicht, die i2c_ch Funktionen zunächst mit # auskommentieren, den sie funktionieren erst, nachdem Sie die i2c_chk Funktionen eingebaut haben (später in Punkt 9).
3. Der Top Level Designer sollte inzwischen *synthi_top.vhd* erzeugt haben, falls nicht, laden Sie von OLAT *synthi_top.vhd* in das *source* Verzeichnis und erzeugen Sie eine Testbench mit Hilfe von emacs, wie in Lab2 beschrieben.
4. Suchen Sie in der Testbench (emacs hat diese *synthi_top_tb.vhd* genannt) den Teil, an dem die Test Procedures aufgerufen werden. Fügen Sie nach dem Aufruf der Procedure *rst_sim* die Prozedur *ini_cod* ein, welche KEY_1 und SW(2..0) stimuliert.

```
elsif cmd = string'("ini_cod") then
          ini_cod(tv, SW(2..0), KEY_1);
```

5. Integrieren Sie in die Testbench (Abbildung 6) noch das Bus Functional Model (bfm), welches die Funktion des Codecs emuliert. Dazu wird *i2c_slave_bfm.vhd* als Komponente in die Testbench eingebunden (Achtung: i2c_slave_bfm nicht in den synthi_top einbinden).

6. Starten Sie die Kompilation im Transcript Fenster mit *do ../compile.do*. Falls *compile.do* nicht gefunden wird, könnte es sein, dass der Simulator im falschen Verzeichnis läuft. Vorher sollte die Person, die am Top-Level arbeitet, den Top Level korrekt synthetisiert haben.
7. Nachdem alle Dateien erfolgreich kompiliert wurden, startet der Simulator. Wählen Sie jetzt die Signale, die Sie betrachten wollen und speichern Sie den *wave.do* ab. Prüfen Sie zunächst nur im Zeitverlaufsdiagramm, ob Ihre Schaltung wie geplant funktioniert.
8. Wenn Sie mit der Funktionalität vertraut sind, bauen Sie in die Testbench die *gpo_chk* Procedures für jeden *i2c_slave_bfm* Registerausgang ein. Fügen Sie danach die automatischen Checks im testcase.dat *i2c_chk* hinzu (s. DTP2_Vorlsung_04, Folie 28). Studieren Sie *reg_table_pkg.vhd* (herunterladen von OLAT MS1) und finden Sie heraus, was die zu erwartenden Ergebnisse für die Registerinitialisierung sein sollten.

   Testen Sie den Codec Controller mit den verschiedenen Tabellen, die Sie mit SW(2..0) umschalten.
9. Starten Sie die Simulation und bringen Sie den Codec-Controller zum Laufen. Der MS1 ist dann erreicht, wenn alle Initialisierungs-Tabellen selbstcheckend passieren.

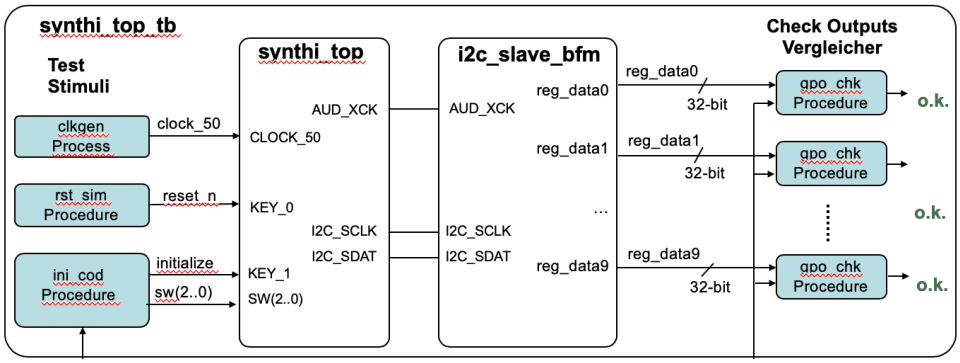

Abbildung 6: Aufbau der Testbench mit i2c_slave_bfm