**FH-OÖ Hagenberg/ESD**
**Hardware / Software Codesign, SS 2020**
Florian Eibensteiner © 2020 (R 5003)

*7. Übung: Cordic alive! – bring it into hardware*

Name(n):

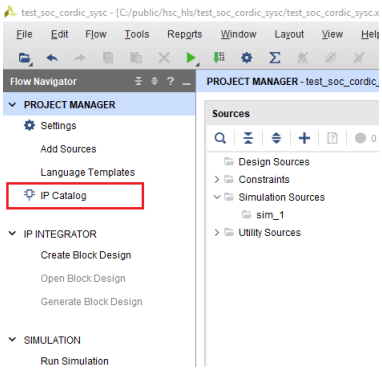
Im letzten Schritt soll nun der erzeugte IP-Core aus der 7. Übung gemeinsam mit dem Zynq-7 Prozessor des XC7Z007S in ein Block-Design integriert werden. In weiterer Folge soll die Firmware vom VP in das SDK von Xilinx portiert und auf dem ARM Cortex A9 des Zynq-7 exekutiert werden. Die notwendigen Schritte um ein Block-Design zu erstellen, ein Bitfile zu generieren und darauf aufbauend im Xilinx SDK eine Plattform und Board-Support-Package aufzusetzen, sollten aus der 4. Übung bereits bekannt sein. Nähere Informationen zu dazu findet sich auch in [Xil19b, CH.10], [Xil19a, Ch.1, Ch.2], bzw. [Xil18, Ch.5,6], Informationen zur Konfiguration des Zynq-7 [Xil19c, Ch.3]

# 1 Hardware Synthese – SoC Design

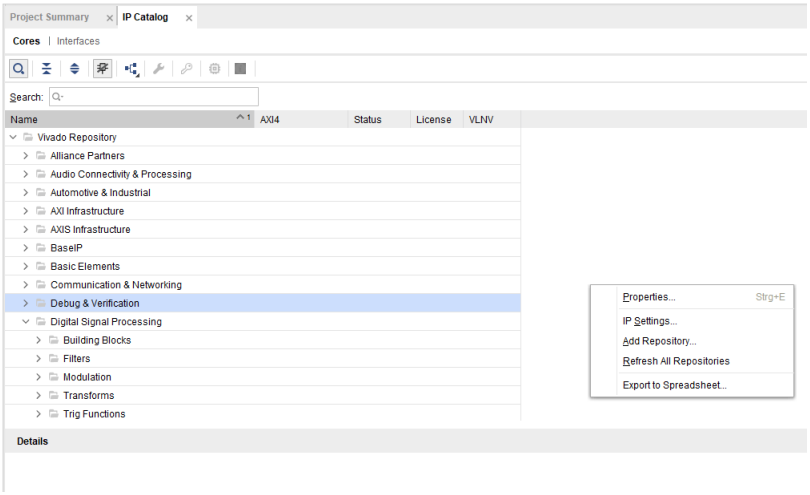
## 1.1 Integration in IP Catalog von Vivado

Zuerst soll der in Übung 7 bereits exportierte IP-Core in den IP-Catalog von Vivado integriert werden. Erstellen Sie dazu ein neues Projekt in Vivado und wählen Sie als Plattform das Ziel-Board MiniZed (xc7z007sclg225-1) aus.

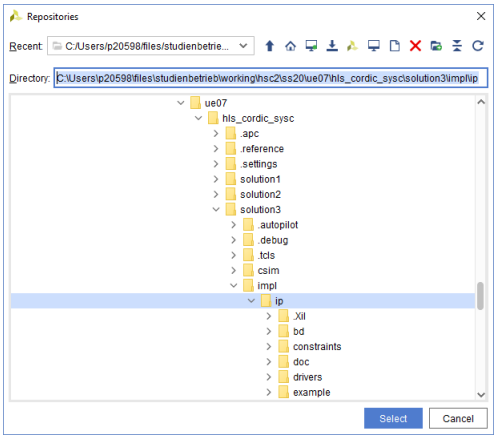
Öffnen Sie nun im *Flow Navigator* den *IP Catalog* und fügen Sie das Repository hinzu welches Ihre IP enthält.

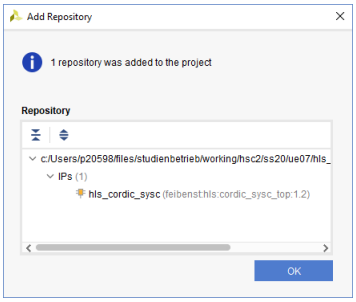



Im Bereich des *Project Manager* öffnet sich der Reiter *IP Catalog*. Klicken Sie hier mit der rechten Maustaste in die leeren Fläche, um das Kontextmenü zu öffnen und wählen Sie die Option *Add Repository . . .* aus.




In dem Fenster das sich nun öffnet muss der Pfad zur exportierten IP eingestellt werden. Wie in unten stehender Grafik links dargestellt liegt der IP-Core standardmäßig im Pfad des Vivado HLS Projektes, in der zuletzt aktiven Solution (z.B.: `<vivado_hls_prj>\solution1\impl\ip`). Wenn der eingestellte Pfad einen IP Core im entsprechenden Format enthält, erscheint ein Fenster in dem das Hinzufügen bestätigt wird.




## 1.2 Block-Design

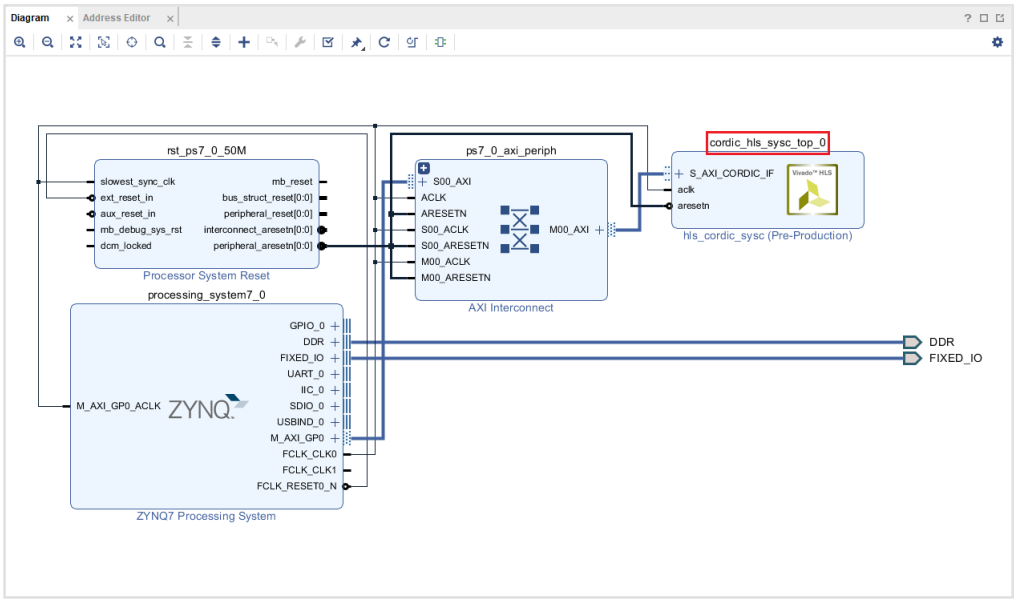
Erstellen Sie nun basierend auf dem MiniZed (xc7z007sclg225-1) ein System-on-a-Chip, dass folgende Komponenten enthält und verbinden Sie die Blöcke mit einem AXI-Bus:

- Zynq7 Processing System (aktives M AXI GP0 Interface)
- Cordic IP

Die Verbindungen zwischen den IP-Blöcke können mit den Design-Assistenten *Run Connection Automation* hinzugefügt werden. Beachten Sie dass der Design Assistent die Ports erst dann automatisch verbinden kann, wenn diese in der jeweiligen IP aktiviert wurden – AXI GP Master Interface

des Zynq ist standardmäßig deaktiviert. Generelle Hinweise zur Erstellung einer Hardwareplattform basierend auf das Board Minized mit Vivado, bzw. generelles zur Verwendung des Tools Vivado findet sich in [Avn18, Xil19a] oder [Xil19b, Ch.10].

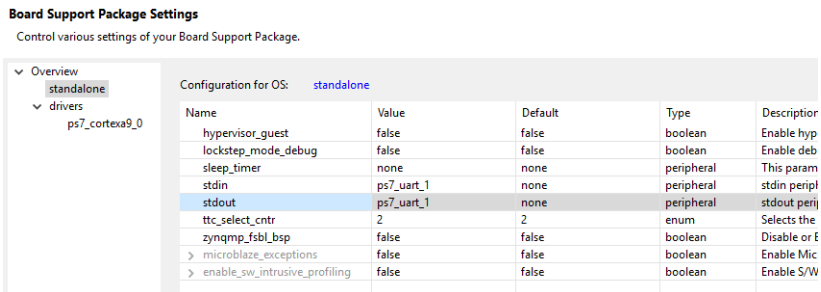
Nach dem Einfügen und Verbinden der beiden Blöcke sollte das Block-Design wie unten dargestellt aussehen. Stellen Sie sicher, dass der Name des Cordic IP-Cores hier ebenfalls das Postfix *top* aufweist. Ansonsten kann es, wie in Übung 7 auf S.6 beschrieben, zu Problemen bei der Einbindung der Treiber im Board-Support-Package kommen. Generieren Sie nun den Bit-Stream und exportieren Sie die Plattform für die Verwendung im Xilinx-SDK.



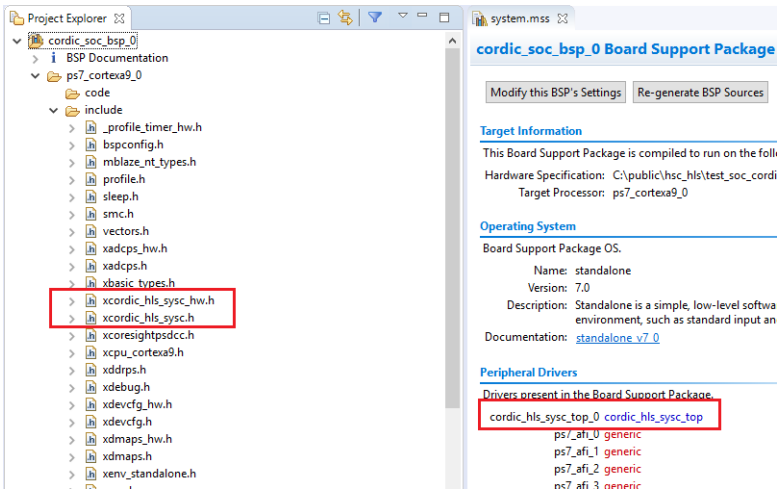

# 2 Firmware und Applikation

Im Xilinx SDK kann nun basierend auf den Daten die aus Vivado exportiert wurden eine *Hardware Platform Specification* und auf diesem aufbauend ein *Board-Support-Package* generiert werden, welches die Treiber der Peripherie enthält.

**Anmerkung:** Bei der Erstellung des *BSP* muss darauf geachtet werden als Peripheral für *stdin* und *stdout* auf *ps7_uart_1* umzustellen (Default-Einstellung ist ps7_uart_0).



Bei erfolgreicher Integration der Treiber und wenn die Namen des Peripherals im Block-Design und in den Treiberinformationen übereinstimmen, sollten die Source-Files im BSP enthalten sein.




Die Device ID und die Basisadresse des Cordic-Core sind in der Datei `xparameters.h` enthalten und sollten beispielsweise wie folgt aussehen.

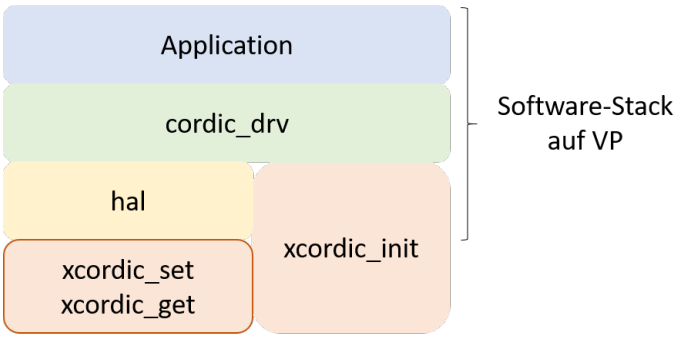
Listing 1: Eintrag der Device-ID und Basisadresse in der Datei xparameters.h

```
/******************************************************************/
/* Definitions for driver CORDIC_HLS_SYSC_TOP */
#define XPAR_XCORDIC_HLS_SYSC_NUM_INSTANCES 1

/* Definitions for peripheral CORDIC_HLS_SYSC_TOP_0 */
#define XPAR_CORDIC_HLS_SYSC_TOP_0_DEVICE_ID 0
#define XPAR_CORDIC_HLS_SYSC_TOP_0_S_AXI_CORDIC_IF_BASEADDR 0x43C00000
#define XPAR_CORDIC_HLS_SYSC_TOP_0_S_AXI_CORDIC_IF_HIGHADDR 0x43C0FFFF
```

## 2.1 Migration der Firmware

Mithilfe des VP wurde bereits ein Software-Stack bestehend aus Applikation, Treiber und HAL entwickelt. Zusätzlich wurden bereits die Initialisierungsfunktionen und auch das Interrupt-Handling an die API von Xilinx angepasst und entsprechende Funktionen am VP nachempfunden. In nachfolgender Abbildung ist eine Übersicht des Firmware-Stacks, wie er aktuell am VP vorhanden sein sollte, dargestellt.




Aktuell wird am VP in der HAL über TLM auf die Register des Cordic-Cores zugegriffen. Integrieren Sie nun in die HAL die *Get*- und *Set*-Methoden der Low-Level-Treiber die im BSP enthalten sind.

**Anmerkung:** Die Namen der Funktionen und Datenstrukturen der Low-Level-Treiber im BSP hängen auch von den Namen ab die bei der High-Level-Synthese verwendet wurden und müssen bei der Migration vom VP evtl. angepasst werden.

Wie sich bereits in Übung 7 bei der Recherche zur Interface-Synthese mit SystemC herausstellte, wird hier in der AXI4-Lite Schnittstelle kein Interrupt zur Verfügung gestellt. Interrupt-Modus ist deshalb auch **nicht** gefordert.

Nach dem auch das Start-Flag nicht wie angenommen automatisch in der Adresslogik der Bus-Schnittstelle erzeugt wird musste auch hier in der 7. Übung Abhilfe geschaffen werden. Passen Sie deshalb die Steuerung des Cordic-Core durch die Software an Ihre Implementierung aus der 7. Übung an.

Der Funktionsumfang der Applikation, berechnen von Cosinus und Sinus für Winkel von $0°$ bis $360°$, kann direkt vom VP übernommen werden. Überprüfen Sie auch ob die Ergebnisse stimmen und bei der HLS keine Fehler aufgetreten sind.

# 3 Diskussion Design-Flow

Im Rahmen der Übung zur LVA HSC haben wir uns mit *virtual Prototpying* beschäftigt und dazu *SystemC* und *TLM* verwendet, um anhand eines sehr abstrakten Simulationsmodells Hardware und Software gleichermaßen zu entwickeln. Schlussendlich haben wir den Versuch gemacht das SystemC-Modell des Cordic-Core aus dem VP für die HLS zu verwendet, um direkt aus dem SystemC Code eine RTL-Beschreibung und einen IP-Core (inkl. Treiber) zu erzeugen. Zuletzt wurde Hardware wie auch Software auf ein echtes System migriert. Diskutieren Sie im Team die Aspekte, Schwierigkeiten, Pro und Kons der einzelnen Stationen des Entwicklungsablaufs:

- Erstellung abstrakter VP
- Verfeinerung des Hardware-Modells für HLS
- Erweiterung der Firmware um Teile der API der Zielplattform und asynchrone Interrupts
- High-Level-Synthese mit SystemC, Schnittstellen-Synthese und erzeugen eines IP-Cores

- Migration auf die Zielplattform

# 4 Abgabe

- IP Core:
  - zip-Archiv im Ordner `<vivado_hls_prj>\solution\impl\ip`
- SoC-Design:
  - Vivado Projekt als Archiv (Project → Archive...)
- Applikation:
  - SDK Projekte
    - Hardware Platform Specification
    - Board Support Package
    - Application (inkl. aller Treiberfiles die nicht im BSP enthalten sind)
- Diskussion:
  - zusammengefasst in einem PDF.

# Literatur


[Avn18] Avnet. *MiniZed: Creating a Zynq Hardware Platform in Vivado*, June 2018.

[Xil18] Xilinx. *UltraFast Embedded Design Methodology Guide*, April 2018. v2.3.

[Xil19a] Xilinx. *Vivado Design Suite Tutorial – Embedded Processor Hardware Design*, June 2019. v2019.1.

[Xil19b] Xilinx. *Vivado Design Suite Tutorial – High-Level Synthesis*, May 2019. v2019.1.

[Xil19c] Xilinx. *Vivado Design User Guide – Embedded Processor Hardware Design*, June 2019. v2019.1.