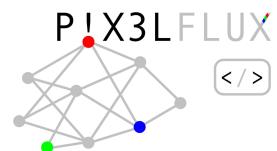




Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES



Fakultät für Elektro- und Informationstechnik
Elektrotechnik und Informationstechnik

LABOR P!X3LFLUX

Laboranleitung: Einfache Echtzeitvideoverarbeitung mit FPGAs

eine Projektarbeit von
Christian Frank, B. Eng.

Stand Sommersemester 2021

Betreuer Professor:
Prof. Dr.-Ing. Jan Bauer

Inhaltsverzeichnis

I Bevor Sie Beginnen	4
1 Einführungsvideos & Tutorial	5
2 MATLAB konfigurieren	6
II Laborversuche	7
3 Teilversuch 1: Skalieren von Intensitätswerten	9
3.1 Teil A: Spielen mit festen Faktoren	9
3.1.1 Getting started	9
3.1.2 Arbeiten mit dem HDL Gain Block	9
3.2 Teil B: A switch to rule them all!	11
3.2.1 Ein erster Entwurf	11
3.2.2 Präskalierung	11
3.2.3 Farbkanäle getrennt skalieren	13
3.3 Teil C: Linear ist langweilig!	15
3.3.1 Einen Sinus erzeugen: Direkte digitale Synthese (DDS)	15
3.3.2 Amplitude und Offset einstellen	17
3.3.3 Gesamtsystem	17
3.4 Optional: Für ganz fleißige	19
4 Teilversuch 2: Vertauschen von Bits	20
4.1 Teil A: Der Lysergsäurediethylamid-Effekt	20
4.1.1 Getting started (again)	20
4.1.2 Bits splitten, vertauschen und mergen	20
4.2 Teil B: Kontrolliertes Chaos	22
4.2.1 Bits per Knopfdruck vertauschen	22
4.2.2 Beispielhaftes Gesamtsystem	23
III Hilfe	25
5 Schnellreferenz Simulink Workbench	26
5.1 Übersicht der Workbench	27
5.1.1 Top-Level Ansicht	27
5.1.2 Hardware-Modell Ansicht	27
5.2 Die Videoschnittstellen des Hardware-Modells	27
5.3 Simulieren	28
5.3.1 Ein Video für die Simulation wählen	28
5.3.2 Die Videoauflösung der Simulation anpassen	28
5.3.3 Schalter und Taster simulieren	28

5.3.4	Blöcke aus der Toolbox verwenden	28
5.4	Den FPGA Programmieren	29
5.4.1	Prinzip	29
5.4.2	HDL Workflow Advisor	30
6	Häufige Fehlerquellen	33
6.1	Probleme bei der Simulation	33
6.1.1	Einige Blöcke in der P!X3LFLUX Workbench fehlen	33
6.1.2	Buttons/Schalter des IO Control Panels bewirken nichts	33
6.2	Probleme bei der FPGA Programmierung	34
6.2.1	HDL Workflow Advisor: Zybo Z7 taucht nicht in der Liste auf	34
6.2.2	HDL Workflow Advisor: Keine Referenzdesigns für das Zybo Z7 gefunden	34
6.2.3	HDL Workflow Advisor: Xilinx Vivado nicht gefunden	34
6.2.4	Sampling-Rate von unendlich (Inf) im Design vorhanden	34
6.2.5	Bitstream generieren: Timing failed	35



Teil I

Bevor Sie Beginnen

Kapitel 1

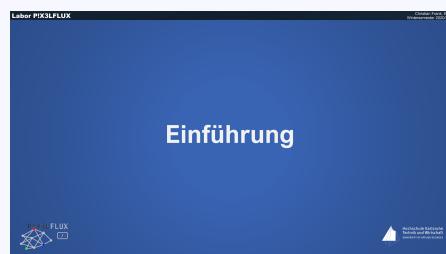
Einführungsvideos & Tutorial

Anstelle einer Versuchseinführung in Schriftform existiert eine kleine Videoserie, die Ihnen den Einstieg so einfach wie möglich machen soll.

Video 1: Einführung

Verschaffen Sie sich mit diesem Video einen Überblick über die Dinge, die Sie im Laborversuch erwarten.

Schauen Sie sich dieses Video bitte noch vor dem Labortermin an, damit Sie am Stichtag vorbereitet sind!



Video 2: Die P!X3LFLUX Simulink Workbench

Eine Vorstellung Ihrer Spielwiese für diesen Laborversuch: Die P!X3LFLUX Simulink Workbench und einige wichtige Konzepte werden hier erklärt.

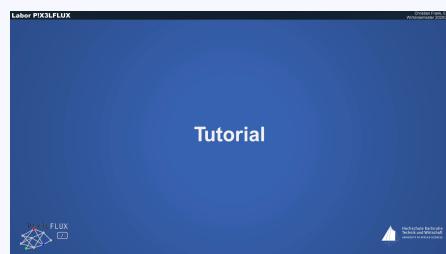
Schauen Sie sich dieses Video bitte noch vor dem Labortermin an, damit Sie am Stichtag vorbereitet sind!



Video 3: Tutorial

Hier werden Sie an einem Beispiel durch den kompletten Workflow geführt, von der Entwicklung des Algorithmus bis zur Programmierung des FPGAs.

Arbeiten Sie dieses Video als kick-off am Labortag durch, um schnell mit dem Workflow warm zu werden.



Kapitel 2

MATLAB konfigurieren

Die folgenden Schritte müssen Sie bei jedem Start von MATLAB erneut abarbeiten:

- Den Pfad zu Xilinx Vivado in MATLAB hinterlegen:** MATLAB HDL Coder verwendet im Hintergrund die Xilinx Vivado Toolchain zum Erzeugen des Bitstreams (Programmierdaten des FPGA-Designs) und für die Programmierung des Chips.

Den Pfad zum Executable von Xilinx Vivado können Sie nach dem Start von MATLAB über das Kommandofenster mit folgendem Befehl deklarieren:

```
hdlsetuptoolpath("ToolName", "Xilinx Vivado", "ToolPath", ...
    "C:\Xilinx\Vivado\2020.1\bin\vivado.bat");
```

Eventuell müssen Sie den Pfad im letzten Argument an Ihre lokalen Verhältnisse anpassen.

- Zybo Z7 Board Definitionen und Referenzdesigns hinzufügen:** Damit MATLAB HDL Coder mit dem Zybo Z7 korrekt arbeiten kann, werden sogenannte Board Definitionen benötigt. Außerdem muss das VHDL-Referenzdesign hinterlegt werden, in das Ihre Algorithmen später eingebettet werden.

Beide Elemente werden im Repository → *zyboZ7_board_definitions* bereitgestellt. Sie müssen diesen Ordner inklusive aller Unterordner zum MATLAB Pfad hinzufügen. Dies erledigen Sie wie in Abbildung 2.1 gezeigt über das Kontextmenü.

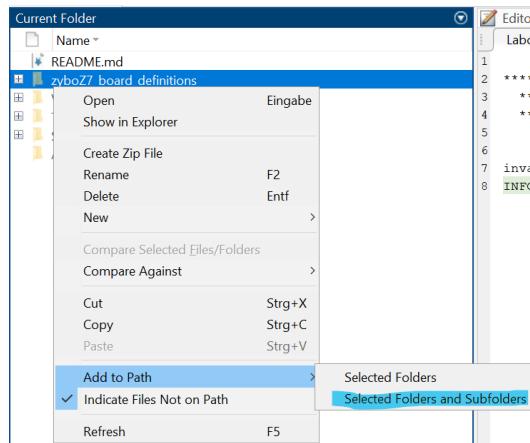


Abbildung 2.1: Board Definitionen und Referenzdesign zu MATLAB hinzufügen

Teil II

Laborversuche



Checkliste vor dem Start

Stop, einen Augenblick noch! Bevor Sie beginnen, müssen Sie folgende Punkte bereits erledigt haben:

- Software installiert und eingerichtet
- Die beiden Einführungsvideos angesehen
- Installationspfad von Xilinx Vivado über `hdlsetuptoolpath` MATLAB bekannt gemacht
- Die Zybo Z7 Board Definitionen dem MATLAB Path hinzugefügt
- Tutorialvideo angesehen und durchgearbeitet

Fertig? Dann los!



Kapitel 3

Teilversuch 1: Skalieren von Intensitätswerten

3.1 Teil A: Spielen mit festen Faktoren

Für den Anfang werden wir uns daran versuchen, unsere drei Farbkanäle mit festen (also im Betrieb unveränderlichen) Faktoren zu multiplizieren. Außerdem machen wir uns mit dem RGB Testbild vertraut, das wir auch im Rest des Teilversuchs verwenden werden.

3.1.1 Getting started

Wichtig: MATLAB konfigurieren

Bevor Sie eines der Simulink-Modell laden, müssen Sie MATLAB wie in Kapitel 2 beschrieben konfigurieren. Ansonsten klappt später das Programmieren des FPGAs nicht!

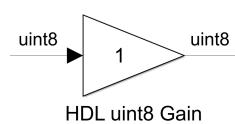
Um loszulegen navigieren Sie zu Ihrem Klon bzw. Download des Git-Repositories. Im Unterordner „Simulink“ finden Sie die vorbereiteten Workbenches zu den Teilversuchen. Gehen Sie folgendermaßen vor:

1. Erzeugen Sie eine Kopie des Modells „intensity_scaling_workbench.slx“ (am besten mit dem Postfix „_TeilA“) und öffnen Sie das Modell in Simulink.
2. Sie sehen die P!X3LFLUX Zybo Z7 Workbench. Öffnen Sie mit einem Doppelklick auf das Hardware-Modell Ihre Spielwiese!
3. Uff... Ganz schön viele Blöcke in der Toolbox. Keine Sorge, für's Erste brauchen wir nur einen davon!

3.1.2 Arbeiten mit dem HDL Gain Block

Der erste für uns interessante Block ist der **HDL uint8 Gain**. Wahrscheinlich ist Ihnen der Simulink Gain Block in Ihrem Studium schon begegnet, bisher mussten Sie sich in der Regel jedoch keine Gedanken über die verwendeten Datentypen machen.

Die Variante aus der Toolbox zeichnet sich in erster Linie dadurch aus, dass sie zur Generierung von HDL Code mit Matlab HDL Coder geeignet ist.



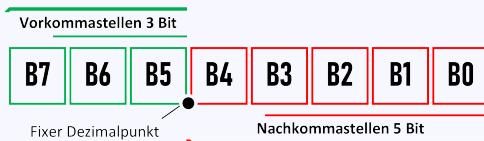
Der Ausgangsdatentyp ist auf uint8 festgelegt, sodass wir den Ausgang direkt mit den Farbkanal-Ausgängen (P!X3LFLUX RGB Merger) verbinden können. Der Gain-Parameter selbst besitzt als voreingestellten Datentyp ufix16_En14¹ und ist auf einen Wertebereich von 0 bis 3.0 limitiert.

Diesen Block wollen wir nun verwenden, um die Eingangsintensitätswerte zu manipulieren. Es ist Zeit für Ihre erste Simulation!

Erklärung: Datentypen mit fixer Präzision

Bestimmt kennen Sie Fließkommatypen wie z.B. double. Deren Realisierung auf einem FPGA verbraucht aber recht viele Ressourcen!

Stattdessen werden wir stets Datentypen mit fixer Präzision verwenden. Das bedeutet, die Anzahl der Bits für die Vor- und Nachkommastellen wird fest eingestellt, woraus eine feste Quantisierung der Nachkommastellen resultiert.



Eine vorzeichenlose 8 Bit fixed-point Zahl mit 5 Bit Nachkommapräzision.

Tipp: Modell aktualisieren

Mit der Tastenkombination STRG+D können Sie das Modell aktualisieren, um mögliche Fehler aufzudecken und die angezeigten Datentypen zu aktualisieren.

Tipp: Blöcke aus der Toolbox schnell kopieren

Blöcke lassen sich mit der Maus schnell kopieren: Klicken und halten Sie mit der rechten Maustaste auf einen Block und verschieben Sie dann den Mauszeiger.

Simulationen zu Teil A

Legen wir also los! Für die nachfolgenden Simulationen wählen Sie bitte in der Videoquelle das Video „rgb_testbild.mp4“ aus dem Testdatenordner.

1. Verwenden Sie den **HDL uint8 Gain** Block aus der Toolbox, um jeden der drei Farbkanäle mit festen Werten zwischen 0 und 3 zu multiplizieren. Probieren Sie ruhig ein paar verschiedene Werte aus.
 - (a) Was beobachten Sie im Testbild, wenn Sie mit Werten größer als 1 multiplizieren? Warum tritt dieser Effekt auf?
 - (b) Stellen Sie als Verstärkungswerte ein paar der Zahlen aus der linken Skala des Testbilds (1.25, 1.5, 2.0,...) ein. Wozu könnte diese Skala gut sein?
2. In den Parametern der HDL Gain Blöcke (Doppelklick, Signal Attributes) finden Sie die Option „Saturate on Integer overflow“. Aktivieren Sie diese Option für alle drei Gains.
 - (a) Wie hat sich das Ergebnis für Werte größer als 1 verändert? Was bewirkt die Option offensichtlich?

¹ufix16_En14 bedeutet: Unsigned, Wortbreite 16 Bit, davon 14 Bit für Nachkommastellen. Also haben wir eine Quantisierung von $\frac{1}{2^{14}} \approx 6,1 \cdot 10^{-5}$ bei den Nachkommastellen

3.2 Teil B: A switch to rule them all!

Einfach nur mit festen Werten zu multiplizieren ist schön und gut. Aber auch ein wenig langweilig, nicht wahr? Deshalb opfern wir für diese ersten Gehversuche aus Teil A nicht extra die Zeit, den FPGA zu programmieren.

Stattdessen werden wir in Teil B den Skalierungsfaktor variabel gestalten - und zwar separat für jeden Farbkanal! Hierfür bieten sich natürlich die Schiebeschalter an, die Ihnen auf der Hardware (und natürlich in der Simulation) zur Verfügung stehen.

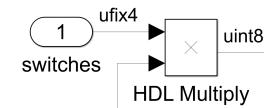
Wie in Teil A legen Sie am besten auch hier eine Kopie der Simulink P!X3LFLUX Workbench an!

3.2.1 Ein erster Entwurf

In einem ersten Ansatz verwenden wir den Input der Schiebeschalter direkt als Multiplikationsfaktor. Der Import **switches** des Modells repräsentiert die 4 Bits, welche die Schiebeschalter codieren.

Da wir nicht mehr einfach nur einen festen Faktor einsetzen, benötigen wir eine Alternative zum HDL Gain Block. In der Toolbox finden Sie dafür den Block **HDL Multiply**.

Dieser Block liefert am Ausgang das Produkt aus den beiden Eingangssignalen. Der Datentyp des Ausgangs ist auf uint8 voreingestellt. Die beiden Eingänge können unterschiedliche Datentypen aufweisen.



Teil B-1: Simulationen zum ersten Entwurf

Wählen Sie bitte auch hier für die Simulation in der Videoquelle das Video „rgb_testbild.mp4“ aus dem Testdatenordner.

1. Schnappen Sie sich drei der **HDL Multiply** Blöcke aus der Toolbox. Bauen Sie damit eine Schaltung auf, welche getrennt für jeden Farbkanal die Eingangsintensität mit dem Wert der Switches multipliziert. Das Produkt soll dann als Ausgangsintensitätswert dienen. Denken Sie daran, dass die Schiebeschalter über das IO Control Panel der P!X3LFLUX Workbench simuliert werden können.
 - (a) Wie ist der Wertebereich des Faktors, mit dem wir auf diese Weise multiplizieren können?
 - (b) Was ist ein offensichtlicher Nachteil dieser einfachen Methode?
2. Auch die HDL Multiply Blöcke verfügen über die Option „*Saturate on Integer overflow*“. Aktivieren Sie die Option und schauen Sie sich das Ergebnis noch einmal an.

3.2.2 Präskalierung

Jetzt können wir also die Intensitätsskalierung im Betrieb steuern. Momentan sind wir aber nur in der Lage, mit ganzzahligen Faktoren zu multiplizieren. Nun wollen wir versuchen, auch rationale Zahlenwerte im Bereich 0 bis 3.0 einzustellen.

Natürlich möchten wir weiterhin unsere Schiebeschalter verwenden. Um unseren gewünschten Wertebereich zu erhalten, müssen wir das switches-Signal jedoch vorskalieren. Hierfür können wir uns erneut des HDL Gain Blocks bedienen, diesmal nehmen wir allerdings den **HDL ufix16 Gain** Block aus der Toolbox.

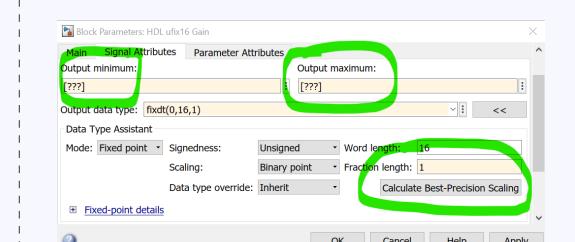


Dieser Block ist so eingestellt, dass sowohl der Ausgang wie auch der Gain Parameter als 16 Bit fixed-point Zahl dargestellt werden. Die Einstellung der Nachkommastellen für das Ausgangssignal dürfen (müssen) Sie diesmal allerdings selbst übernehmen!

Tipp: Fixed-Point Datentyp anpassen

In den Block-Einstellungen (Doppelklick) lässt sich mit dem *Data Type Assistant* leicht die optimale Konfiguration eines fixed-point Datentyps bestimmen. Klicken Sie auf den „>“-Button, um das Tool anzuzeigen.

Wenn Sie als Minimum und Maximum den gewünschten Wertebereich angeben, lässt sich mit dem Button „*Calculate Best-Precision Scaling*“ die optimale Fraction length (Nachkommastellen) automatisch übernehmen.



Exemplarische Einstellungen zum Ausgangsdatentyp.

Teil B-2: Simulation zur Präskalierung

Wählen Sie bitte wie bisher für die Simulation in der Videoquelle das Video „rgb_testbild.mp4“ aus dem Testdatenordner.

1. Erweitern Sie Ihre bestehende Schaltung um die Präskalierung des switches-Signals. Stellen Sie den Gain so ein, dass der Wertebereich des 4 Bit switches-Import genau auf den Wertebereich [0, 3] abgebildet wird. Stellen Sie Minimum & Maximum für den Ausgangsdatentyp sinnvoll ein übernehmen Sie die optimale Fraction length.
 - (a) Wie müssen Sie den Gain wählen, um den gewünschten Wertebereich zu erhalten?
 - (b) Welche Quantisierung bekommen Sie für die Nachkommastellen am Ausgang?
2. Jetzt sollten Sie in der Simulation auch rationale Werte auswählen können!

Teil B-2: Den FPGA konfigurieren

Genug mit den schwerfällig laufenden Simulationen, es wird echt Zeit für Echtzeit!

Wichtig: Der Import **buttons** dürfte aktuell mit nichts verbunden sein. Damit Sie später keine Probleme bei der HDL Generierung bekommen, müssen Sie diesen Port vorübergehend löschen!

1. Entscheiden Sie sich, ob Sie die die Option „*Saturate on Integer overflow*“ in den HDL Multiply Blöcken aktivieren möchten oder nicht.
2. Gehen Sie wie im Tutorial gezeigt vor, um Ihre Schaltung auf den FPGA zu bringen. Denken Sie daran, den switches Outport dem gleichnamigen Signal im Referenzdesign zuzuweisen.
3. Sie können Ihren Laptop oder den Labor-PC mit dem HDMI-Rx Port des Zybo Z7 verbinden (720p Auflösung). Zum Vergleich mit der Simulation bietet es sich an, „rgb_testbild.mp4“ mit dem Medioplayer Ihrer Wahl zu öffnen. Viel Spaß beim Spielen!

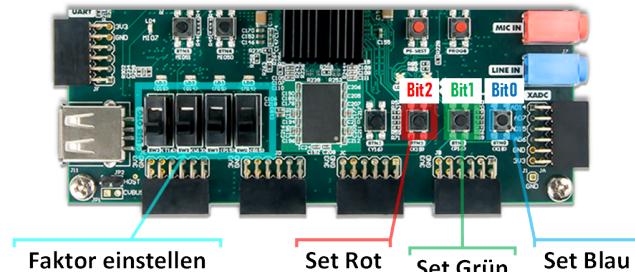
3.2.3 Farbkanäle getrennt skalieren

Mit den Schiebeschaltern zu spielen macht Spaß, aber auf dem Board gibt's ja auch noch ein paar Knöpfe. Da geht noch was!

Unser neues Ziel soll es sein, für jeden Farbkanal separat den Faktor für die Intensitätsskalierung auswählen zu können.

Dafür weisen wir den Schiebeschaltern sowie drei der Buttons (wie rechts im Bild gezeigt) folgende Funktionalitäten zu:

- Mit den Schiebeschaltern sollen wir weiterhin den Faktor für die Skalierung der Intensitätswerte definieren können.
- Mit den Buttons 0, 1 und 2 soll sich der eingestellte Faktor jeweils für den blauen, grünen oder roten Kanal übernehmen lassen.

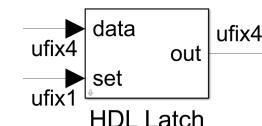


Offensichtlich benötigen wir eine Art Speicherglied, mit dem wir den gewählten Faktor für jeden Kanal auf Knopfdruck speichern können.

3.2.3.1 Der HDL Latch Block

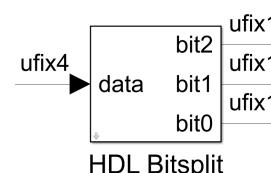
Genau für den Zweck der Speicherung auf Knopfdruck finden Sie in der Toolbox den **HDL Latch** Block. Dieser verfügt über einen Daten- (data) sowie einen Steuereingang (set). Die Funktionalität ist folgendermaßen definiert:

1. Wird der set-Eingang 1 (oder true) geschaltet, so wird mit dem nächsten Pixeltakt der Wert am data-Eingang gespeichert und am out-Ausgang ausgegeben.
2. Ist der set-Eingang 0 (oder false), so wird der letzte gespeicherte Wert am Ausgang beibehalten. Initial wird 0 ausgegeben.



3.2.3.2 Der HDL Bitsplit Block

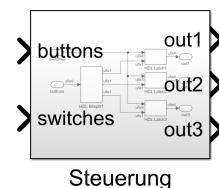
Der HDL Latch Block erwartet am set-Eingang lediglich ein einzelnes Bit (ufix1) bzw. einen Logikwert (boolean). Das Signal unserer Buttons ist jedoch vom Typ ufix4, also 4 Bit breit. Daher müssen wir dieses ufix4-Signal in einzelne ufix1-Singale aufspalten, um so auf die einzelnen Bits verwenden zu können.



Hierzu existiert der **HDL Bitsplit** Block. In dessen Einstellungen können Sie die Anzahl der Bits wählen, die extrahiert werden sollen. Da wir nur drei Buttons benötigen, reicht für uns ebendiese Anzahl an Bits.

3.2.3.3 Putting it all together

Damit kennen Sie nun alle Elemente, die Sie zum bauen einer kleinen „Steuerung“ benötigen. Diese sollte drei HDL Latch Blöcke zum Speichern eines Faktors für jeden Farbkanal enthalten.



Es ist empfehlenswert, Ihre Steuerung in einem eigenen Subsystem zusammenzufassen (wie rechts gezeigt), da wir im zweiten Teilversuch eine identische Steuerung benötigen werden. So können Sie den Subsystem-Block einfach kopieren und einfügen.

Tipp: Blöcke als Subsystem zusammenfassen

Wenn Sie mehrere Blöcke in Simulink markiert haben, können Sie diese durch STRG+G (oder über das Kontextmenü) zu einem Subsystem zusammenfassen. Das hilft etwas dabei, den Überblick zu behalten.

Tipp: Debugging mit dem Logic Analyzer

Eine gute Möglichkeit für das Debugging ihrer Schaltung ist der Matlab Logic Analyzer. Diesen können Sie in der Simulink GUI unter *Simulation→Review Results* aufrufen.

Sie können Signale zum Logging mit dem Logic Analyzer hinzufügen, indem Sie eine Signalleitung anklicken und mit der Maus über das „...“-Symbol fahren. Es erscheint ein Menü, in dem Sie den Punkt „*Enable Data Logging*“ auswählen können.

Teil B-3: Simulationen mit Steuerung

Wählen Sie bitte auch hier für die Simulation in der Videoquelle das Video „rgb_testbild.mp4“ aus dem Testdatenordner.

1. Basteln Sie sich Ihre Steuerung aus den HDL Latch und HDL Bitsplit Blöcken zusammen. Bauen Sie eine Schaltung auf, mit der Sie für jeden Farbkanal einen Faktor im Wertebereich [0, 3] einstellen können.
2. Spielen Sie mit den Kombinationen! Sie können beispielsweise nur einen oder zwei Farbkanäle „einschalten“ oder versuchen, eine bestimmte Farbe mit der Graustufenskala zu mischen (Sie haben 4096 Permutationen zur Auswahl).

Teil B-3: Den FPGA konfigurieren

Es ist wieder an der Zeit, unsere Schaltung in Hardware zu manifestieren!

1. Entscheiden Sie sich, ob Sie die die Option „*Saturate on Integer overflow*“ in den HDL Multiply Blöcken aktivieren möchten oder nicht.
2. Gehen Sie wie im Tutorial gezeigt vor, um Ihre Schaltung auf den FPGA zu bringen. Diesmal müssen Sie sowohl dem switches-Port wie auch dem buttons-Port das jeweilige Gegenstück im Referenzdesign zuweisen.
3. Sie können Ihren Laptop oder den Labor-PC mit dem HDMI-Rx Port des Zybo Z7 verbinden (720p Auflösung). Zum Vergleich mit der Simulation bietet es sich an, „rgb_testbild.mp4“ mit dem Mediaplayer Ihrer Wahl zu öffnen. Viel Spaß beim Spielen!

3.3 Teil C: Linear ist langweilig!

Bis jetzt haben wir unsere Intensitätswerte ganz simpel mit einem linear wählbaren Faktor gewichtet. Das wird als „Spielzeug“ schnell langweilig. Darum werden wir im Folgenden einen neuen Ansatz wählen: Wir lassen unseren Skalierungsfaktor einem Sinusverlauf folgen, und zwar mit einer zeitlichen Abhängigkeit!

Dabei soll es auch möglich sein, die Frequenz, Amplitude und den Offset im Betrieb einzustellen.

Ähnlich wie in Teil B verwenden wir die Schalter und Knöpfe folgendermaßen:

- Mit den Schiebeschaltern lässt sich ein 4 Bit Zahlenwert einstellen.
- Die Buttons 2, 1 und 0 erlauben es, den eingestellten Zahlenwert für die Frequenz, Amplitude oder den Offset des Sinus zu übernehmen.

Denken Sie daran, wieder eine Kopie der Simulink P!X3LFLUX Workbench anzulegen!

3.3.1 Einen Sinus erzeugen: Direkte digitale Synthese (DDS)

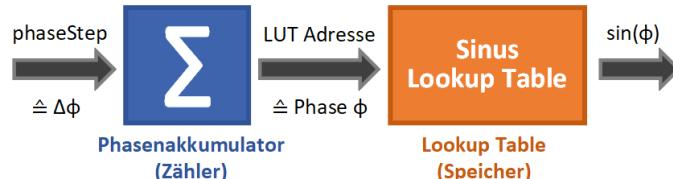


Abbildung 3.1: Veranschaulichung der DDS Struktur

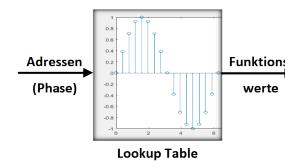
Wir benötigen also eine günstige Möglichkeit, einen in der Frequenz veränderlichen Sinus in unserem FPGA zu erzeugen.

Eine gängige Methode zur Lösung dieser Aufgabe ist die sogenannte *direkte digitale Synthese (DDS)*. Keine Sorge, das Konzept ist recht simpel! Schauen wir uns die in Grafik 3.1 gezeigten Blöcke einmal an:

3.3.1.1 Lookup Table (LUT)

Das Herzstück der DDS ist die *Lookup Table*. Dabei handelt es um einen „tabellarischen“ Speicher, welcher vorberechnete Abtastwerte einer Sinusperiode² beinhaltet. Die Abtastwerte werden während der Konfiguration des FPGAs eingespeichert.

Der Vorteil liegt darin, dass zur Laufzeit keine Funktionswerte des Sinus berechnet werden müssen. Stattdessen lässt sich der gewünschte Funktionswert einfach aus dem Speicher auslesen! Dabei entspricht die gewählte Speicheradresse direkt einem Phasenwert Φ der Sinusfunktion $\sin(\Phi)$.



²In der Praxis reicht tatsächlich eine Viertelperiode aus, da die Symmetrien der Sinusfunktion ausnutzt werden.

3.3.1.2 Phasenakkumulator

Die Adressen (Phasenwerte) für unsere LUT müssen wir aber auch noch irgendwie generieren. Hierzu dient der sogenannte *Phasenakkumulator*. Hinter dem Namen verbirgt sich ein einfacher digitaler Zähler!

Dieser Zähler wird mit einer einstellbaren Schrittweite (entspricht einem Phasenschritt $\Delta\Phi$) inkrementiert. Ein Zählerstand von 0 entspricht der Phase $\Phi = 0$, der maximale Stand entspricht $\Phi = 2\pi$. Durch die endliche Bitbreite kommt es periodisch zum Überlauf, wodurch das Ausgangssignal des Phasenakkumulators einer Sägezahnfunktion gleicht. In Abbildung 3.2 sehen Sie zur Verdeutlichung die simulierten Zeitverläufe.

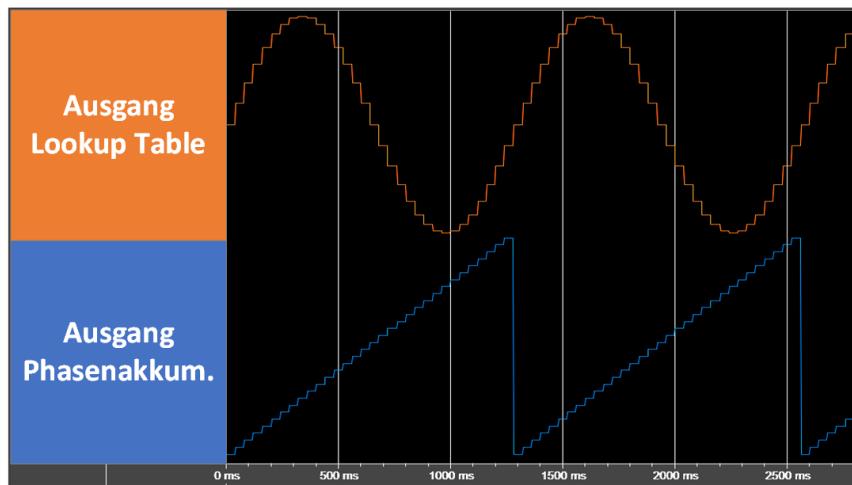


Abbildung 3.2: In Simulink simulierte DDS Zeitverläufe

Mit dem Phaseninkrement $\Delta\Phi$ lässt sich die Frequenz des Sinussignals steuern: Je größer wir $\Delta\Phi$ wählen, desto schneller wird die Lookup Table durchlaufen.

3.3.1.3 DDS in Matlab

Die direkte digitale Synthese müssen Sie nicht selbst implementieren (Puh...!). Stattdessen finden Sie in der Toolbox mit dem **DDS** Block eine fertige Musterimplementierung.

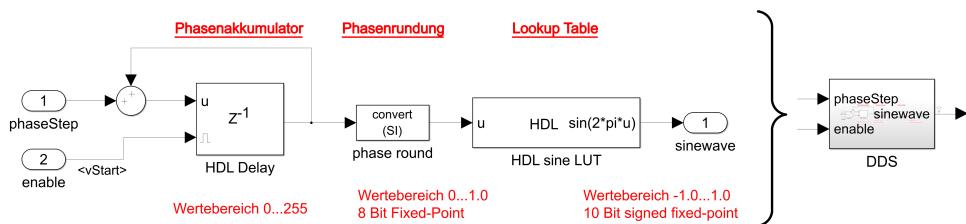


Abbildung 3.3: DDS Struktur in Simulink aufgebaut

Diese Implementierung arbeitet mit einem 8 Bit Phasenakkumulator, es können also 256 verschiedene Abtastwerte in der LUT adressiert werden³.

³Die LUT der HDL Coder Library erfordert einen Eingangswertebereich von 0 bis 1.0. Daher wird die 8 Bit Ganzzahl aus dem Phasenakkumulator vor der LUT in eine Fixed-Point Zahl konvertiert

3.3.1.4 Den DDS Takt verlangsamen

Bei einem zu Grunde liegenden Pixeltakt von 74,25 MHz (1280x720p bei 60 Bildern pro Sekunde) wäre die kleinste Ausgangsfrequenz unserer DDS $\frac{74.25 \text{ MHz}}{256} \approx 290 \text{ kHz}$. Das ist natürlich viel zu schnell, um die Intensitätsschwankung mit bloßem Auge zu erkennen!

Daher werden wir stattdessen die Taktung der DDS auf die Bildrate (60 Hz) verlangsamen. Damit wird die kleinste mögliche Frequenz zu gemütlichen $\frac{60 \text{ Hz}}{256} = 0.23 \text{ Hz}$.

Die Frequenzverlangsamung wird möglich durch den *enable*-Eingang des DDS-Blocks. Diesen müssen wir pro Frame genau für einen Pixeltakt auf logisch 1 bzw. true schalten.

Günstigerweise enthält der **pixelcontrol**-Bus unseres Video-Eingangs das Signal **vStart**: Dieses schaltet genau für das erste Pixel in einem Frame auf true. In der Toolbox finden Sie den Block **vStart select**, welcher zum Abgriff des vStart-Signals für den DDS Enable-Eingang dient.

3.3.2 Amplitude und Offset einstellen

Jetzt sind wir also ohne viel Aufwand dazu in der Lage, einen in der Frequenz veränderlichen Sinus zu erzeugen. Klasse Sache! Allerdings möchten wir ja auch noch dazu in der Lage sein, die Amplitude und den Offset des Sinussignals einzustellen.

Zu diesem Zweck finden Sie in der Toolbox den fertigen Block **WaveModify**. Für dessen Ausgang gilt folgende Gleichung:

$$\text{out} = 0.2 \cdot \text{amplitude} \cdot \text{sinewave} + \underbrace{0.2 \cdot (\text{amplitude} + \text{offset})}_{\text{Amplitude}=\text{Minimaloffset}}$$

Um nicht mit negativen Werten zu multiplizieren, wird die Sinusfunktion (unabhängig vom eingesetzten Offset) um die gewählte Amplitude nach oben verschoben.

3.3.3 Gesamtsystem

Zusammen mit unserer bewährten Steuerung haben wir jetzt alles nötige zur Hand, um ein funktionales Gesamtsystem zusammenzubasteln.

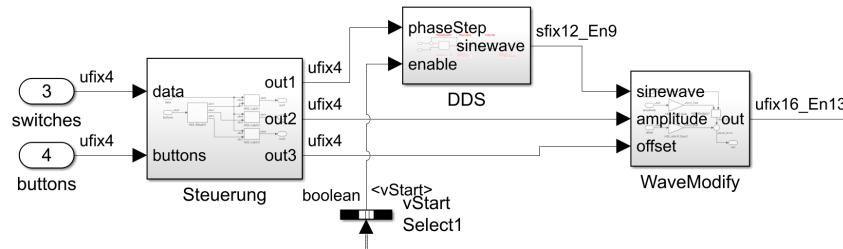


Abbildung 3.5: Gesamtsystem für die sinusförmige Intensitätsskalierung

Dieser Aufbau erlaubt es...

- ... die Frequenz in Schritten von 0.23 Hz zwischen 0 Hz und 3.45 Hz zu variieren.
- ... die Amplitude und den Offset in Schritten von 0.2 zwischen 0 und 3 zu variieren.

Genug Theorie! Jetzt können wir uns endlich an die Simulationen und die Konfiguration des FPGAs machen.

Hinweis: Abweichende DDS Frequenz in der Simulation

Unsere DDS taktet mit dem Frame-Takt. Auf der Hardware ist dieser typischerweise $f_{framerate} = 60$ Hz. In der Simulation verwenden wir jedoch Videos mit einem Frametakt von $f_{framerate} = 25$ Hz. Daher beläuft sich die kleinste DDS-Frequenz in der Simulation auf $\frac{25 \text{ Hz}}{256} \approx 0.1$ Hz

Tipp: Simulationsgeschwindigkeit

Die Simulation der DDS dürfte deutlich langsamer laufen als alle bisherigen Simulationen. Damit Sie schneller ganze Perioden sehen (besonders beim Aufzeichnen mit dem Logic Analyzer), sollten Sie eine möglichst hohe DDS-Frequenz einstellen (z.B. Faktor 10).

Teil C: Simulationen zur DDS

Für die nachfolgenden Simulationen wählen Sie bitte in der Videoquelle das Video „rgb_testbild.mp4“ aus dem Testdatenordner (wie überraschend!).

1. Bauen Sie das oben gezeigt Gesamtsystem in Ihrer Simulink Workbench zusammen. Verifizieren Sie durch eine Simulation, dass sich die Frequenz, Amplitude und Offset wie gewünscht einstellen lassen.
 - (a) Zeichnen Sie mit dem Logic Analyzer den Ausgang des Phasenakkumulators und des DDS-Blocks auf. Verhalten sich die Signale, wie Sie das erwarten würden?

Teil C: Den FPGA konfigurieren

Jetzt wird es wieder Zeit, mit unserer Schaltung in Echtzeit zu spielen!

1. Diesmal sollten Sie die Option „Saturate on Integer overflow“ in den HDL Multiply Blöcke auf jeden Fall aktivieren. Das Ergebnis lässt sich so viel leichter sehen.
2. Gehen Sie wie im Tutorial gezeigt vor, um Ihre Schaltung auf den FPGA zu bringen. Weisen Sie wie immer den switches und buttons die korrekten Signale zu.
3. Sie können Ihren Laptop oder den Labor-PC mit dem HDMI-Rx Port des Zybo Z7 verbinden (720p Auflösung). Zum Vergleich mit der Simulation bietet es sich an, „rgb_testbild.mp4“ mit dem Medioplayer Ihrer Wahl zu öffnen. Viel Spaß beim Spielen!

3.4 Optional: Für ganz fleißige

Falls Sie bis jetzt gut in der Zeit liegen (oder am Ende des Labors noch Zeit und Lust übrig ist), können Sie sich daran versuchen, die Lösung aus Teil C so zu erweitern, dass die Frequenz für jeden Farbkanal unterschiedlich gewählt werden kann.

Das lässt sich mit den Blöcken aus der Toolbox anstellen, viel Spaß beim Tüfteln!

Kapitel 4

Teilversuch 2: Vertauschen von Bits

4.1 Teil A: Der Lysergsäurediethylamid-Effekt

Inzwischen haben Sie bestimmt die Schnauze voll von Intensitätsskalierung! Deshalb werden wir uns im Folgenden einem Thema widmen, bei dem man ohne großen Aufwand interessante Effekte erhält: Wir basteln für unseren FPGA einen Lysergsäurediethylamid-Modus¹!

Was das sein soll? Sie werden bald sehen, woher die Bezeichnung röhrt...

In diesem zweiten Teilversuch sind Sie eingeladen, sich bei der Gestaltung Ihrer Schaltung frei auszutoben. Es gibt keine „falschen“ oder „richtigen“ Lösungen für die folgenden Aufgaben.

4.1.1 Getting started (again)

Die Workbench zu diesem Versuch finden Sie im Repository unter *Simulink → 2_Laborversuch_Bit_Switching → bit_switching_worbench.slx*.

Auch hier sollten Sie wieder für jeden Teil eine Kopie anlegen, damit sie stets Zugriff auf einen sauberen Ausgangszustand haben. Außerdem ist es sinnvoll, Zwischenstände separat abzuspeichern, wenn Sie etwas tolles gebastelt haben.

4.1.2 Bits splitten, vertauschen und mergen

Unseren LSD-Effekt erhalten wir durch das Vertauschen von Bits in den Farbkanälen.

Dafür benötigen wir einen Weg, die jeweils 8 Bits unserer drei uint8-Farbsignale zu separieren. Hierzu bedienen wir uns wieder dem **HDL Bitsplit** Block, den Sie bereits im ersten Teilversuch kennengelernt haben. Die Variante aus der Toolbox ist diesmal direkt passend auf eine Wortbreite von 8 Bits eingestellt.

Außerdem müssen wir aus den separierten Bits auch wieder ein uint8-Signal für den Ausgang erzeugen können. Hierzu dient der **HDL Bit Concat** Block aus der Toolbox. Dieser vereint die einzelnen Bits ans seinen Eingängen zu einem 8 Bit Signal am Ausgang. Dabei wird das unterste Bit als LSB und das oberste als MSB gewertet.

In Abbildung 4.1 sehen Sie beispielhaft, wie die Bits 7 und 2 miteinander vertauscht werden können.

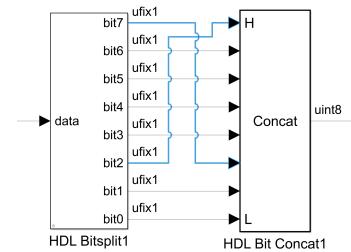


Abbildung 4.1: Vertauschen der Bits 7 und 2

¹Sie kennen diesen nicht ganz einfach auszusprechenden Begriff vielleicht besser unter der Abkürzung LSD.

Tipp: Einen guten Effekt erzielen

Den größten Einfluss haben die Bits im oberen Nibble. Aber Vorsicht, wenn Sie zu viele Bits gleichzeitig vertauschen werden Sie feststellen, dass man schnell überhaupt nichts mehr im Bild erkennt!

Teil A: Simulationen

Diesmal dürfen Sie sich frei der Videos aus dem Testdatenordner bedienen. Wählen Sie die kleineren Auflösungen, um die Simulation etwas zu beschleunigen.

1. Versuchen Sie für den Anfang, lediglich für einen der drei Farbkanäle (am besten eine recht dominante Farbe im gewählten Testvideo) Bits zu tauschen. Dafür benötigen Sie einen HDL Bitsplit und einen HDL Bit Concat Block. Die anderen beiden Farbkanäle können Sie einfach direkt vom Eingang auf den Ausgang schalten.
 - (a) Jetzt verstehen Sie, weshalb wir vom LSD-Effekt sprechen (so viele schöne Farben...!).
 - (b) Können Sie sich erklären, warum das Vertauschen der Bits eine so dramatische Wirkung auf das Bild hat?
2. Jetzt wird es unübersichtlich: Vertauschen Sie in allen drei Farbkanäle einige Bits miteinander!
 - (a) Seien Sie kreativ, es gibt viele Möglichkeiten!
 - (b) Haben Sie zum Beispiel schon versucht, einzelne Bits zwischen zwei Farbkanälen zu tauschen?

Teil A: Den FPGA konfigurieren

Bringen Sie Ihre Schaltung auf die Hardware, sobald Sie mit Ihrem Effekt zufrieden sind.

Wichtig: Die Imports **buttons** und **switches** dürften aktuell mit nichts verbunden sein. Damit Sie später keine Probleme bei der HDL Generierung bekommen, müssen Sie diese Ports löschen!

1. Gehen Sie wie im Tutorial gezeigt vor, um Ihre Schaltung auf den FPGA zu bringen. Diesmal sollten Sie keine weiteren Einstellungen in der Signalzuweisung vornehmen müssen.
2. Sie können Ihren Laptop oder den Labor-PC mit dem HDMI-Rx Port des Zybo Z7 verbinden (720p Auflösung). Vielleicht findet sich im Labor ja sogar eine Kamera mit HDMI Ausgang. Viel Spaß beim Spielen!

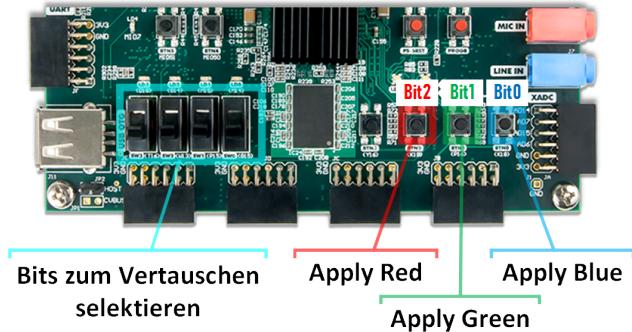
4.2 Teil B: Kontrolliertes Chaos

Der Effekt, den wir durch das Vertauschen erzielen können, ist schon ziemlich cool! Noch spannender wird es aber, wenn wir mit Hilfe der Buttons und Schiebeschalter in Echtzeit festlegen können, welche Bits miteinander vertauscht werden sollen!

Hier ein Vorschlag für eine mögliche Belegung der Elemente auf dem Zybo Z7 Board:

- Mit den Schiebeschalter werden die zu vertauschenden Bits selektiert.
- Die Buttons erlauben eine individuelle Konfiguration pro Farbkanal.

Denken Sie wie immer daran, eine Kopie der Workbench anzulegen.



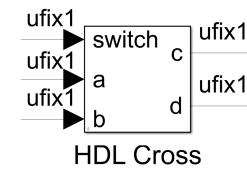
4.2.1 Bits per Knopfdruck vertauschen

Schauen wir uns im Folgenden einmal an, wie sich die Bitvertauscherei auf Knopfdruck bewerkstelligen lässt!

4.2.1.1 Der HDL Cross Block

Damit Sie besonders einfach zur Laufzeit Bits vertauschen können steht Ihnen in der Toolbox der **HDL Cross** Block zur Verfügung. Dieser arbeitet folgendermaßen:

- Liegt am switch-Eingang eine logische 0, so wird a auf c und b auf d durchgeschaltet.
- Liegt am switch-Eingang eine logische 1, so wird a auf d und b auf c durchgeschaltet.



4.2.1.2 Beispiel zum HDL Cross Block

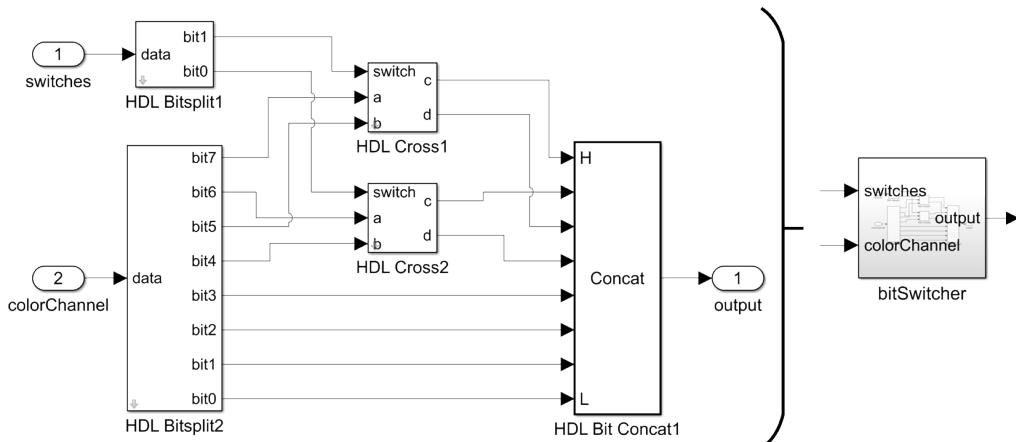


Abbildung 4.2: Beispiel zum gesteuerten Vertauschen mit zwei HDL Cross Blöcken

In Abbildung 4.2 sehen Sie eine beispielhafte Beschaltung zweier HDL Cross Blöcke:

- Wie zuvor werden die 8 Bits des Farbkanals durch einen HDL Bitsplit Block aufgetrennt und später durch einen HDL Bit Concat Block wieder zusammengefügt.
- Mit den beiden HDL Cross Blöcken lassen sich jeweils die Bits {7, 5} und {6, 4} vertauschen.
- Um die *switch*-Eingänge mit den Schiebeschaltern ansteuern zu können, wird deren 4 Bit breites ufix4-Signal in 1 Bit breite ufix1-Signale aufgetrennt. Auch hier kommt der HDL Bitsplit Block zum Einsatz.

An diesem Beispiel können Sie sich beim Zusammenstöpseln Ihrer eigenen Schaltung orientieren. Natürlich können Sie mit dem ufix4 Signal der Schiebeschalter insgesamt vier HDL Cross Block einzeln schalten.

4.2.2 Beispielhaftes Gesamtsystem

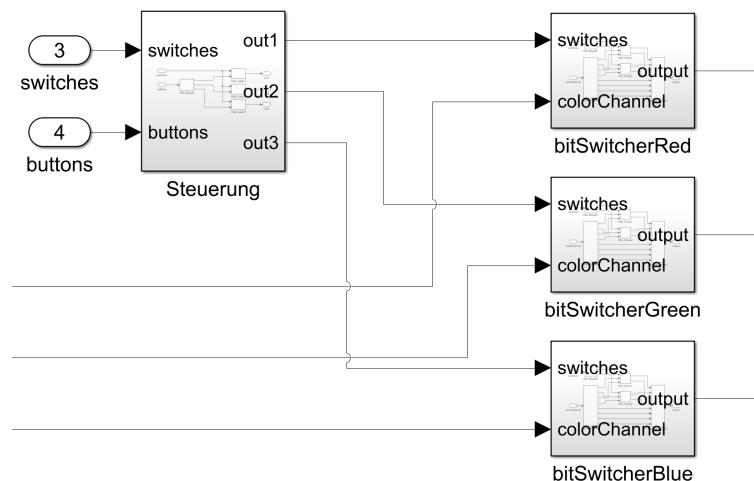


Abbildung 4.3: Eine mögliche Implementierung mit der Steuerung aus Teilversuch 1

Abbildung 4.3 zeigt eine mögliche Implementierung, bei der für jeden Farbkanal ein System nach Abbildung 4.2 eingesetzt wird. Um die Vertauschung für jeden Kanal einzeln zu wählen, kommt hier die bereits aus dem ersten Teilversuch bekannte Steuerung zum Einsatz. Kopieren Sie diesen Block also aus einer Ihrer älteren Workbenches.

Das Beispiel dient natürlich nur zur Orientierung, vielleicht schwert Ihnen ja eine kreativere Idee vor!

Tipp: Für Ordnung sorgen

Damit Sie nicht so leicht den Überblick verlieren, sollten Sie hier Ihre Teilschaltungen unbedingt in Subsysteme verpacken! Bauen Sie nicht alles auf einmal zusammen. Verifizieren Sie mit Simulationen stattdessen zwischendurch, ob Ihre Teilschaltungen das gewünschte Verhalten zeigen.

Teil B: Simulationen

Suchen Sie sich auch hier wieder ein Testvideo aus, das Ihnen gut gefällt!

1. Bauen Sie ein System auf, mit dem Sie für einen Farbkanal möglichst viele Vertauschungsmöglichkeiten über die Schiebeschalter einstellen können. Bestimmt haben Sie in Teil A bereits einige hübsche Kombinationen entdeckt!
 - (a) Testen Sie mit den Schaltern und dem Simulink Logic Analyzer, ob die Vertauschung wie gewünscht funktioniert.
2. Nun können Sie die Steuerung hinzuziehen, um die Farbkanäle getrennt zu bedienen. Bauen Sie sich ein Gesamtsystem, das Sie auf den FPGA bringen möchten.

Teil B: Den FPGA konfigurieren

Wenn Sie mit Ihrer Schaltung zufrieden sind, dann wird es Zeit, zum letzten Mal die Mühlen des HDL Workflow Advisors zu durchlaufen!

1. Gehen Sie wie im Tutorial gezeigt vor, um Ihre Schaltung auf den FPGA zu bringen. Diesmal sollten Sie keine weiteren Einstellungen in der Signalzuweisung vornehmen müssen.
2. Sie können Ihren Laptop oder den Labor-PC mit dem HDMI-Rx Port des Zybo Z7 verbinden (720p Auflösung). Vielleicht findet sich im Labor ja sogar eine Kamera mit HDMI Ausgang. Viel Spaß beim Spielen!

Teil III

Hilfe

Kapitel 5

Schnellreferenz Simulink Workbench

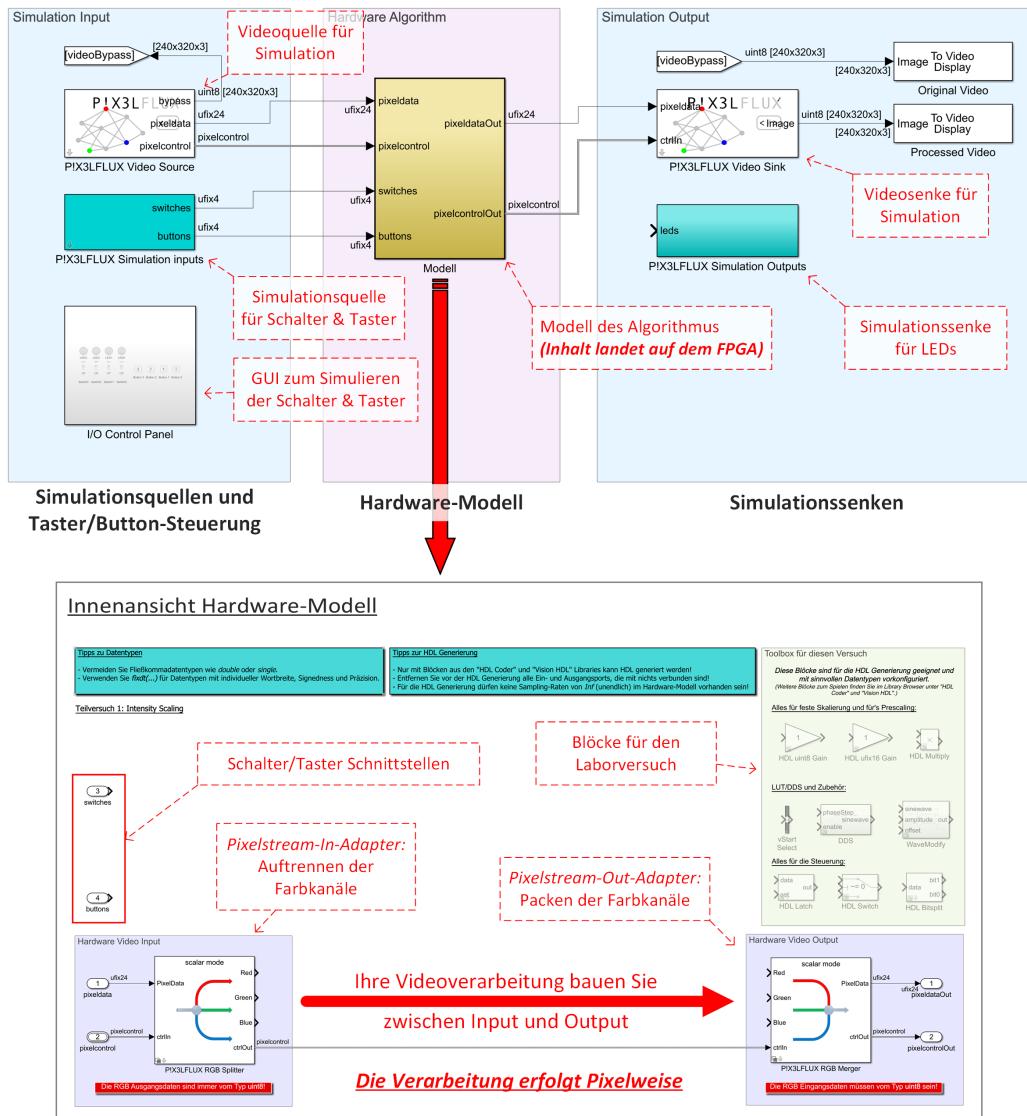


Abbildung 5.1: Übersicht der P!X3LFLUX Zybo Z7 Workbench

5.1 Übersicht der Workbench

5.1.1 Top-Level Ansicht

Die Top-Level Ansicht (Abbildung 5.1 oben) der Workbench ist in drei Bereiche unterteilt:

Hardware Algorithm Dies ist das Herzstück der Workbench! Der **Modell** Block repräsentiert den Videoverarbeitungsalgorithmus, welcher auf der Hardware implementiert werden soll. Die Imports und Outports des Modells entsprechen dabei den realen Schnittstellen, mit denen der Algorithmus an das VHDL-Referenzdesign angebunden wird.

Simulation Input Hier werden die Simulationssignale für alle Input-Schnittstellen des Hardware-Modells generiert. Die **P!X3LFLUX Video Source** simuliert den Pixelstream, welcher auf dem FPGA dem HDMI Input entspringt. Über das **I/O Control Panel** und die **P!X3LFLUX Simulation Inputs** lassen sich die Schalter und Taster des Zybo Z7 simulieren.

Simulation Output In diesem Bereich werden die Output-Schnittstellen des Hardware-Modells für die Simulation angezapft. Die **P!X3LFLUX Video Sink** wandelt den verarbeiteten Pixelstream aus dem Algorithmus in einzelne Frames. Sowohl das originale wie auch das verarbeitete Video werden zum Vergleich während der Simulation in zwei Ausgabefenstern angezeigt.

5.1.2 Hardware-Modell Ansicht

Über einen Doppelklick auf das Modell erreichen Sie die Innenansicht des Hardware-Modells (Abbildung 5.1 unten). Dies ist der Platz, an dem Sie Ihren Videoverarbeitungsalgorithmus realisieren! Der Inhalt des Modells landet später auf dem FPGA.

Auch hier sehen Sie einige vorgegebene Strukturen:

P!X3LFLUX RGB Splitter Dieser Block spaltet den eingehenden Pixelstream in drei getrennte uint8 Farbkanäle (Rot, Grün, Blau) auf. Der RGB Splitter bildet die Schnittstelle zum eingehenden Videosignal auf dem Referenzdesign, Ihr Arbeitsbereich beginnt daher mit den Ausgängen dieses Blocks. Außerdem verfügt der Block über implizites Pipelining, um das Timing bei der FPGA-Implementierung zu stabilisieren (HDL-Konfiguration des Blocks).

P!X3LFLUX RGB Merger Dies ist der Gegenspieler zum RGB Splitter: Die drei Farbkanäle des Pixelstreams werden wieder in ein gemeinsames Signal gepackt. Ihr Arbeitsbereich endet an den Eingängen dieses Blocks. Wie beim RGB Splitter existiert auch hier implizites Pipelining, um das Timing bei der FPGA-Implementierung zu stabilisieren (HDL-Konfiguration des Blocks).

Toolbox Die Toolbox enthält alle Blöcke, die Sie zum bearbeiten des jeweiligen Teilversuchs benötigen. Die Blöcke sind auskommentiert bzw. deaktiviert.

Buttons/Switches Imports Diese Simulink Imports repräsentieren die Schnittstelle zu den Schiebeschaltern und Buttons auf dem Zybo Z7 Board. Beide besitzen eine Wortbreite von 4 Bits.

Wichtig zu verstehen ist, dass der implizite Takt des Algorithmus der Pixeltakt des eingehenden Videosignals ist. Bei einer Auflösung von 1280x720p @60 FPS (wie sie vom Referenzdesign im Labor verwendet wird) entspricht dies $f_{720p} = 74.25$ MHz.

5.2 Die Videoschnittstellen des Hardware-Modells

Das Hardware-Modell verfügt über je einen eingehenden und ausgehenden Pixelstream. Eine Videoschnittstelle besteht dabei aus jeweils zwei Signalen:

pixeldata Dieses 24 Bit breite Signal repräsentiert die RGB-Grauwerte eines einzelnen Pixels. Die Farbkanäle haben jeweils eine Auflösung von 8 Bit.

pixelcontrol Hierbei handelt es sich um einen Bus vom Typ *pixelcontrol*. Das Signal enthält fünf boolean Flags (siehe Tabelle 5.1), welche Informationen zur Bildposition des aktuellen Pixels liefern.

Flag	Bedeutung
vStart	Ist für das erste Pixel im Frame (erste Zeile, erste Spalte) <i>true</i> .
vEnd	Ist für das letzte Pixel im Frame (letzte Zeile, letzte Spalte) <i>true</i> .
hStart	Ist für das erste Pixel eines Frames <i>true</i> .
hEnd	Ist für das letzte Pixel eines Frames <i>true</i> .
valid	Ist für jedes Pixel im sichtbaren (aktiven) Bereich <i>true</i> .

Tabelle 5.1: Flags im pixelcontrol-Bus

Die Videoschnittstellen arbeiten also *pixelweise*, d.h. der Algorithmus erhält pro Taktflanke ein neues Pixel zur Verarbeitung.

Die Schnittstellen folgen dabei dem Konzept des MATLAB „Streaming Pixel Interface“. Für nähere Informationen lohnt sich ein Blick in die MATLAB Dokumentation [hier \(klick!\)](#) und [hier \(klick!\)](#).

5.3 Simulieren

5.3.1 Ein Video für die Simulation wählen

In den Einstellungen des Blocks **P!X3LFLUX Video Source** (Top-Level Ansicht) können Sie den Pfad zu einer Videodatei angeben, die Sie für Ihre Simulation verwenden möchten. Hierzu stehen Ihnen die mitgelieferten Testvideos zur Verfügung.

5.3.2 Die Videoauflösung der Simulation anpassen

Die Blöcke **P!X3LFLUX Video Source** und **P!X3LFLUX Video Sink** (Top-Level Ansicht) bieten Ihnen in den Einstellungen verschiedene Auflösungen an, mit denen Sie das Testvideo für die Simulation skalieren können. Eine kleine Auflösung führt dabei zu einer performanteren Simulation.

Wichtig: Die Auflösung muss in beiden Blöcken auf den selben Wert eingestellt sein.

5.3.3 Schalter und Taster simulieren

Über das **IO Control Panel** (Top-Level Ansicht) können Sie die Schalter und Taster des Zybo Z7 Boards simulieren. Bedienen Sie hierzu einfach die entsprechenden Elemente bei laufender Simulation.

Die eigentlichen Signale werden im **P!X3LFLUX Simulation Inputs** Block generiert und an das Hardware Modell geleitet.

Tipp: Mit Rechtsklick→*Open in New Window* können Sie das IO Control Panel in einem eigenen Fenster öffnen. Damit lässt es sich leichter arbeiten!

5.3.4 Blöcke aus der Toolbox verwenden

Die Toolbox enthält alle für den Versuch notwendigen Blöcke. Diese sind standardmäßig auskommentiert. Um einen Blöcke zu benutzen sollten Sie ihn zunächst entweder durch Ziehen bei gedrückter rechter Maustaste oder mittels Copy-Paste vervielfältigen. Anschließend können Sie über das Kontextmenü→*Uncomment* den Block aktivieren.

5.4 Den FPGA Programmieren

5.4.1 Prinzip

Der MATLAB HDL Coder erlaubt es, aus dem Simulink Hardware-Modell der P!X3LFLUX Workbench VHDL Code zu generieren. Bei dem für dieses Labor relevanten „IP Core Generation“ Workflow entsteht dabei ein sogenannter IP-Core¹; dabei handelt es sich um eine abgeschlossene Funktionseinheit mit definierten Schnittstellen, welche die VHDL Implementierung des Algorithmus aus Simulink enthält².

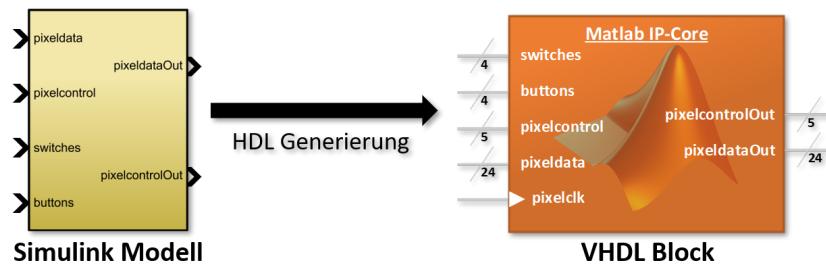


Abbildung 5.2: Generierung eines HDL IP-Cores aus dem Simulink Modell

Der so entstandene IP-Core wäre allerdings auf dem FPGA alleine nicht funktional. Daher steht ein speziell für das Labor erstelltes VHDL-Referenzdesign (bzw. Template) zur Verfügung, welches in erster Linie die Ansteuerung der HDMI Ports übernimmt und Andockpunkte für die Schnittstellen des IP-Cores bereitstellt.

In dieses Referenzdesign wird der IP-Core eingefügt. Aus dem entstandenen Gesamtsystem wird über die Xilinx Vivado Toolchain ein Bitstream generiert, mit welchem anschließend der FPGA programmiert werden kann.

All diese Schritte laufen in der Praxis größtenteils automatisiert über den MATLAB HDL Workflow Advisor ab.

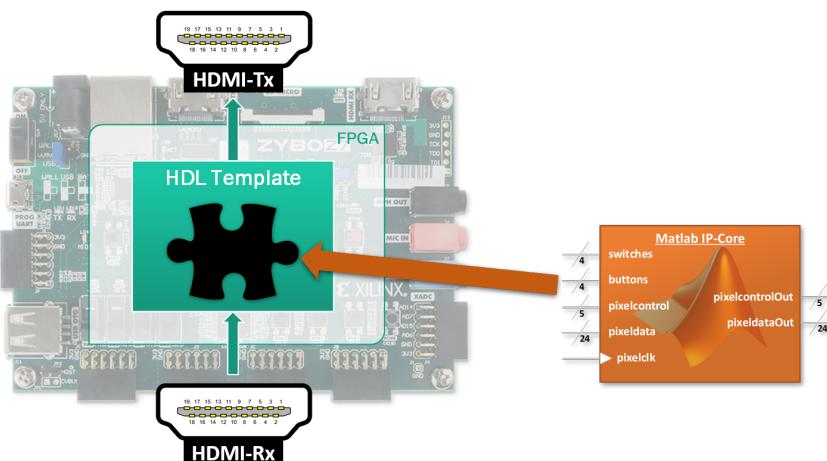


Abbildung 5.3: Einbetten des IP-Cores in ein HDL-Referenzdesign

¹IP steht hierbei für „Intellectual Property“.

²Außerdem fügt MATLAB einige weitere Strukturen hinzu, unter anderem Adapterstrukturen für die Videoschnittstellen.

5.4.2 HDL Workflow Advisor

Der HDL Workflow Advisor ist ein grafisches Tool, mit dem Sie Ihr Simulink Modell komfortabel auf der Hardware implementieren können. Nachfolgend werden die einzelnen Schritte knapp kommentiert. Ein vollständiges Beispiel finden Sie im Tutorial Video.

5.4.2.1 Den HDL Workflow Advisor öffnen

Mit einem Rechtsklick auf das Hardware-Modell finden Sie im Kontextmenü den Punkt *HDL Code* → *HDL Workflow Advisor*. Hiermit lässt sich (wie in Abbildung 5.4 gezeigt) das Tool öffnen.

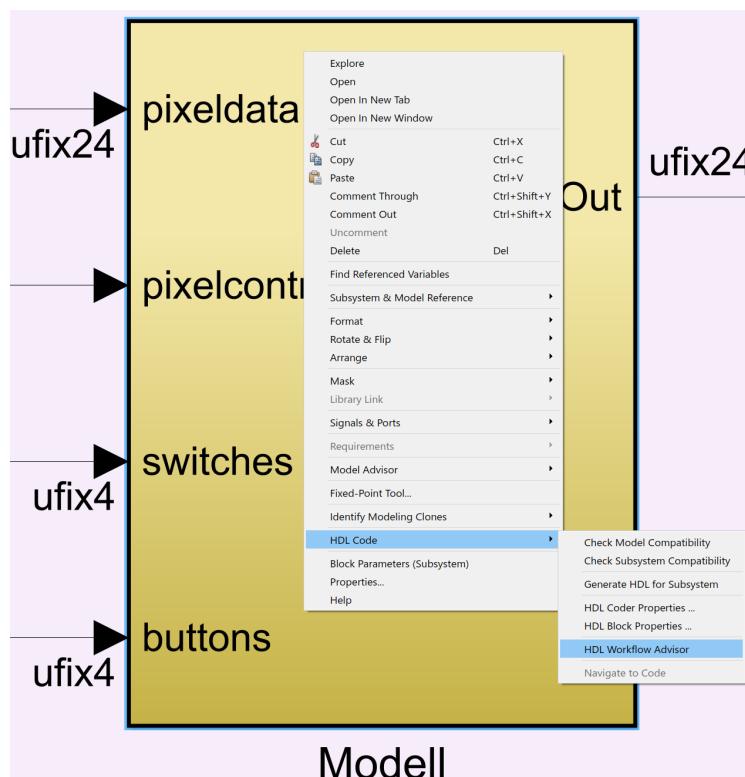


Abbildung 5.4: HDL Workflow Advisor öffnen

5.4.2.2 Schritt 1.1 Set Target Device

In diesem Schritt werden der Workflow, das Zielgerät (Zybo Z7-20), das Synthese Tool (Xilinx Vivado) und ein Pfad für die HDL Coder Projektdateien spezifiziert. Die korrekten Einstellungen können Sie Abbildung 5.5 entnehmen.

Wichtig: Projektpfad

Xilinx Vivado ist leider etwas eingeschränkt bezüglich der Pfadauswahl. Es dürfen keine Leerzeichen oder Umlaute im Pfad enthalten sein. Idealerweise legen Sie - wie im Screenshot gezeigt - einen Ordner im Stammverzeichnis Ihres Rechners an. Diesen sollten Sie (sofern Sie einen der Labor-PCs benutzen) am Ende des Labors wieder löschen.

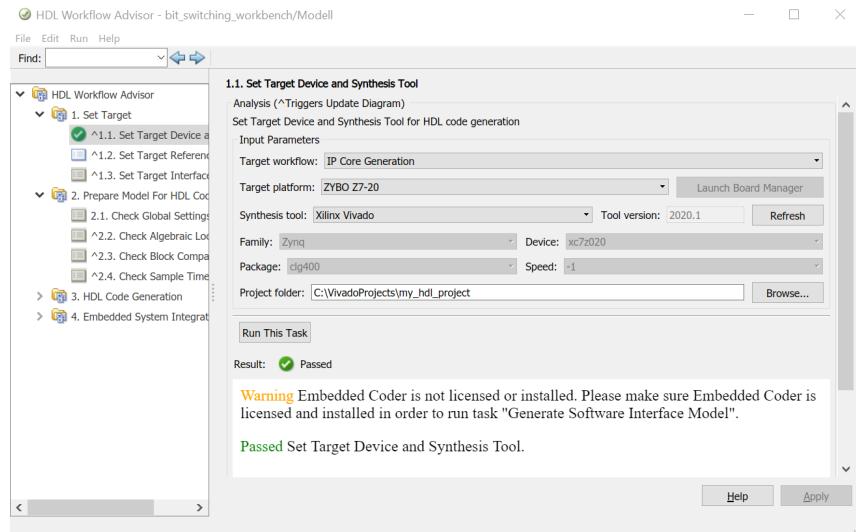


Abbildung 5.5: Einstellungen zu Schritt 1.1

5.4.2.3 Schritt 1.2 Set Target Reference Design

In diesem Schritt selektieren Sie (wie in Abbildung 5.6 gezeigt) als Referenzdesign „Einfaches AXI4s-Video (RGB)“.

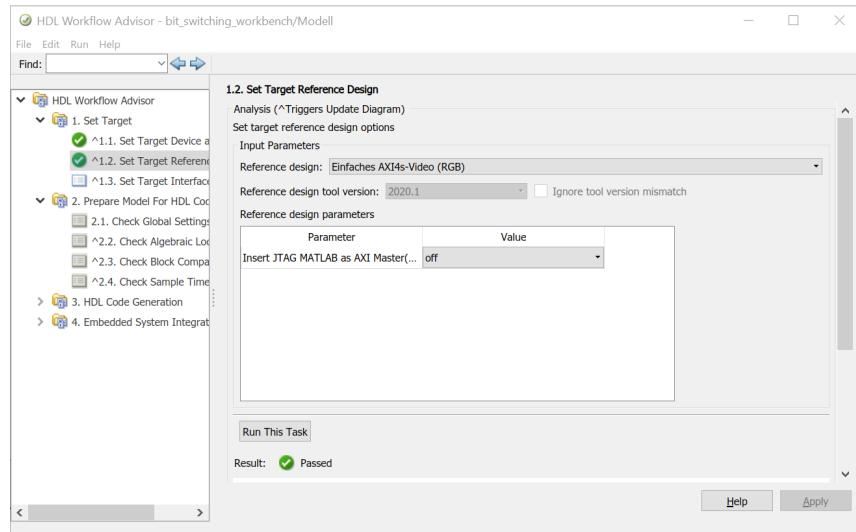


Abbildung 5.6: Einstellungen zu Schritt 1.2

5.4.2.4 Schritt 1.3 Set Target Interface

Hier weisen Sie den Schnittstellen Ihres Hardwaremodells entsprechende Signale im Referenzdesign zu. Die korrekten Einstellungen sind in Abbildung 5.7 zu sehen.

5.4.2.5 Schritt 2 Prepare Model For HDL Code Generation

Die Subschritte von Schritt 2 können Sie automatisch über den Button „Run All“ ausführen. Sollte es hier zu Problemen kommen wird Ihnen der HDL Workflow Advisor einen Button „Modify All“

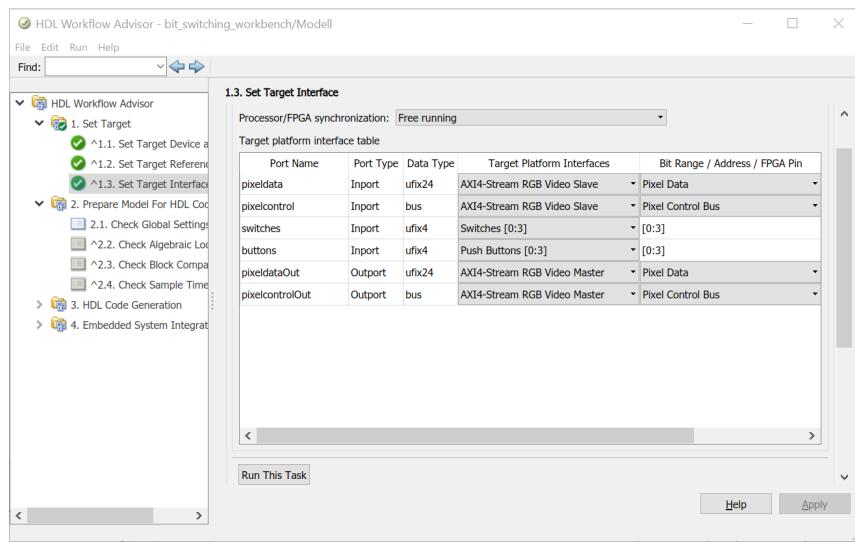


Abbildung 5.7: Einstellungen zu Schritt 1.3

anbieten, mit dem die notwendigen Einstellungen automatisch korrigiert werden.

5.4.2.6 Schritt 3 HDL Code Generation

Auch Schritt 3 kann vollautomatisch mit dem „Run All“-Button abgeschlossen werden.

5.4.2.7 Schritt 4.1 Create Project

In diesem Schritt müssen sie keine Einstellungen vornehmen.

5.4.2.8 Schritt 4.2 Generate Software Interface Model

Diesen Schritt müssen Sie durch Setzen der Checkbox „Skip this task“ überspringen.

5.4.2.9 Schritt 4.3 Build FPGA Bitstream

Hier starten Sie die Synthese und anschließende Bitstream-Generierung durch Xilinx Vivado. Aktivieren Sie hierzu die Checkbox „Run build process externally“. Es wird sich ein Kommandofenster öffnen, in dem Sie die Ausgaben von Vivado sehen. Der Prozess kann einige Zeit dauern. In einem finalen Print werden Sie über den Erfolg des Vorgangs informiert und aufgefordert, das Kommandofenster zu schließen.

5.4.2.10 Schritt 4.4 Program Target Device

Schließen Sie Ihr Zybo Z7 über ein USB Kabel an Ihren Rechner an. Wählen Sie als Programmiermethode „JTAG“. Der Programmievorgang kann einige Sekunden dauern.

Kapitel 6

Häufige Fehlerquellen

6.1 Probleme bei der Simulation

6.1.1 Einige Blöcke in der P!X3LFLUX Workbench fehlen

Werden in den vorgegebenen Strukturen der P!X3LFLUX Workbench Blöcke als „not installed“ angezeigt werden, so fehlt Ihnen mindestens eine MATLAB Abhängigkeit. Für das Labor müssen folgende Toolboxen installiert sein:

- Simulink
- Fixed-Point Designer
- Image Processing Toolbox
- Computer Vision Toolbox
- HDL Coder
- HDL Verifier
- Vision HDL Toolbox

Außerdem benötigen Sie folgende Add-Ons:

- HDL Coder Support Package for Xilinx Zynq Platform
- Computer Vision Toolbox Support Package for Xilinx Zynq-Based Hardware

6.1.2 Buttons/Schalter des IO Control Panels bewirken nichts

In den Laborversuchen arbeiten Sie mit mehreren Kopien der P!X3LFLUX Workbench. Dabei kann es leicht passieren, dass ein altes „IO Control Panel“-Fenster versehentlich nicht geschlossen wird. In diesem Fall werden die Schalter und Buttons nichts bewirken, da sie zu einem anderen Simulink Modell gehören.

Best Practice: Stellen Sie beim Wechsel zwischen P!X3LFLUX Workbenches sicher, dass Sie alle Simulink Fenster schließen. Vorsicht: Wenn Sie auch MATLAB schließen müssen Sie daran denken, die Zybo Z7 Board Definitionen erneut in den Pfad aufzunehmen.

Eine weitere Ursache kann sein, dass die jeweiligen Imports des Hardwaremodells nicht mit den Simulationsquellen aus der Top-Level Ansicht verbunden wurden.

6.2 Probleme bei der FPGA Programmierung

6.2.1 HDL Workflow Advisor: Zybo Z7 taucht nicht in der Liste auf

Taucht das Zybo Z7 im Dropdownmenü unter Schritt 1.1 des HDL Workflow Advisors nicht auf, so haben Sie wahrscheinlich vor dem Start von Simulink vergessen, die Zybo Z7 Board Definitionen zum MATLAB-Pfad hinzuzufügen.

Gehen Sie folgendermaßen vor:

1. Speichern Sie Ihre Arbeit und schließen Sie MATLAB/Simulink vollständig.
2. Öffnen Sie MATLAB, ohne Simulink (bzw. die P!X3LFLUX Workbench) zu starten.
3. Navigieren Sie im MATLAB Dateiexplorer zu Ihrem Klon/Download des Git-Repositories und fügen Sie über das Kontextmenü die Board Definitionen zum MATLAB-Pfad hinzu. Verwenden Sie hierzu die Option „Add to Path → Selected Folders and Subfolders“.
4. Das Zybo Z7 und die zugehörigen Referenzdesigns sollten nun verfügbar sein.

6.2.2 HDL Workflow Advisor: Keine Referenzdesigns für das Zybo Z7 gefunden

Ist das Zybo Z7 im Schritt 1.1 des HDL Workflow Advisors zwar verfügbar, dessen Auswahl liefert jedoch eine Fehlermeldung bzgl. fehlender Referenzdesigns, so haben Sie wahrscheinlich beim Hinzufügen der Zybo Z7 Board Definitionen zum MATLAB-Pfad einen Fehler gemacht: Sie müssen auch alle Unterordner einbeziehen.

Gehen Sie folgendermaßen vor:

1. Speichern Sie Ihre Arbeit und schließen Sie MATLAB/Simulink vollständig.
2. Öffnen Sie MATLAB, ohne Simulink (bzw. die P!X3LFLUX Workbench) zu starten.
3. Navigieren Sie im MATLAB Dateiexplorer zu Ihrem Klon/Download des Git-Repositories und fügen Sie über das Kontextmenü die Board Definitionen zum MATLAB-Pfad hinzu. Verwenden Sie hierzu die Option „Add to Path → Selected Folders and Subfolders“.
4. Das Zybo Z7 und die zugehörigen Referenzdesigns sollten nun verfügbar sein.

6.2.3 HDL Workflow Advisor: Xilinx Vivado nicht gefunden

Für die Erzeugung des Bitstreams - also die Daten, mit denen der FPGA Programmiert wird - ist das Tool Xilinx Vivado erforderlich. Sie müssen MATLAB den Pfad zu diesem Tool mitteilen.

Geben Sie hierzu den folgenden Befehl in das MATLAB Kommandofenster ein: `hdlsetuptoolpath('ToolName', 'Xilinx Vivado', ... 'ToolPath', 'C:\Vivado\2020.1\bin\vivado.bat')`

6.2.4 Sampling-Rate von unendlich (Inf) im Design vorhanden

Sollten Sie vom HDL Workflow Advisor eine Fehlermeldung bzgl. unendlicher Sampling-Raten in Ihrem Design erhalten, so liegt dies meist an In- oder Outports des Hardwaremodells, welche mit nichts verbunden wurden.

Solche Ports erhalten leider standardmäßig eine automatisch zugewiesene Sampling-Rate von unendlich (inf). Sollten Sie die Ports nicht benötigen können Sie diese einfach entfernen.

Wird Ihr Problem damit nicht gelöst können Sie die vorhandenen Sampling-Raten im Modell anzeigen lassen. Aktivieren Sie dazu über das Kontextmenü die Option „Sample Time Display → All“. So sollten Sie die störenden Blöcke leicht identifizieren können.

6.2.5 Bitstream generieren: Timing failed

Dieses Problem tritt auf, wenn die Implementierung Ihres Algorithmus' auf dem FPGA die Timing-Bedingungen (Setup und Hold Time) verletzt.

Das ist etwa dann der Fall, wenn ein Signal zwischen zwei Taktflanken viele Logikgatter (durch Addition, Multiplikation, etc.) durchlaufen muss: Jedes Logikgatter erzeugt eine gewisse Verzögerung. Diese Verzögerungen summieren sich zu einer Pfadlatenz auf. Ab einer gewissen Latenz kann nicht mehr garantiert werden, dass die Schaltung korrekt funktioniert, es kann zu sogenannten Metastabilitäten kommen.

In den Versuchsaufgaben sollte dies nicht so schnell passieren. Überprüfen Sie bitte zuerst, ob Ihre Lösung nicht vielleicht effizienter gestaltet werden kann!

Leider lässt sich dieses Problem nicht immer nach einem „Kochrezept“ lösen. Eine gute und für unsere Anwendungen oft ausreichende Strategie ist jedoch das sogenannte „Pipelining“. Die Idee ist, den Signalpfad in mehrere Etappen aufzubrechen, indem Verzögerungs- bzw. Halteglieder eingefügt werden. Hierdurch muss ein Signal innerhalb eines Taktzyklus einen kürzeren (d.h. weniger Latenz durch Gatter) Pfad durchlaufen. Der Preis ist natürlich, dass eine Verzögerung des Signalfusses in Kauf genommen werden muss.

Ein Beispiel sehen Sie in den Abbildungen 6.1 und 6.2. Im oberen Fall muss das Signal in einem Taktzyklus den vollständigen Pfad zwischen den beiden Additionen durchlaufen. Ist die Gatterverzögerung der Blöcke zu groß kommt es zur Timing-Verletzung. Im unteren Block wird als Gegenmaßnahme Pipelining eingesetzt. Das getaktete Verzögerungsglied trennt den Pfad in zwei Teile auf, deren Pfadlatenz kleiner ist als zuvor.

Tipp: Delay Block

Einen zur HDL-Generierung tauglichen Delay Block finden Sie in der Simulink Block Library unter „HDL Coder → Commonly Used Blocks → Delay“.



Abbildung 6.1: Das Signal muss zwei Additionen durchlaufen

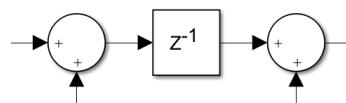


Abbildung 6.2: Auf trennen des Pfades durch Pipelining

Wichtig: Asynchronität

Bitte bedenken Sie, dass durch die Verzögerungsglieder eine Asynchronität zwischen den Ein- und Ausgängen eines Blocks entstehen kann. Üblicherweise müssen Sie alle Eingangs- bzw. Ausgangssignale um den selben Wert verzögern.

Wichtig: Verzögerung der Farbkanäle

Die Signale der drei Farbkanäle sowie der pixelcontrol-Bus müssen immer um den selben Wert verzögert werden!