$e^{j\pi} = -1 \iff$ *Wer sich umdreht, blickt in die entgegengesetzte Richtung.*
(frei nach LEONHARD EULER)

Tx-Gruppe

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

Oszillatoren spielen in der Nachrichtentechnik an vielen Stellen eine wichtige Rolle. Auch im Bereich der digitalen Signalverarbeitung werden sie häufig eingesetzt. Eine herausragende Rolle nehmen die Sinusgeneratoren ein, welche meist nach dem Verfahren der *Direct Digital Synthesis*, kurz DDS genannt, arbeiten.

In dieser Übung soll eine Unit erstellt werden, welche dieses Verfahren zur Erzeugung eines möglichst rauscharmen, sinusförmigen Signals einsetzt. Hierzu wird eine Tabelle verwendet, die in RAM-Blöcken im FPGA gespeichert ist. Die vorgeschlagene Implementation erlaubt die Erzeugung von Schwingungen mit weitgehend beliebig wählbaren Frequenzen (lediglich begrenzt durch den Systemtakt bzw. die Samplerate des DAC). Auch die Phase des Ausgangssignals ist mit hoher Genauigkeit steuerbar.

1. Aufgabe *DDS 1.0: einfach, aber flächenaufwändig*

In dieser Aufgabe soll die DDS auf dem einfachsten Weg realisiert werden. Hierzu wird eine *komplette* Periode des Sinussignals in einer ROM-Tabelle abgelegt. Damit gewinnt die Adresse der Daten im ROM die Bedeutung der Phase der Sinusschwingung. Lässt man den Adresszeiger stetig anwachsen, so wächst auch die Phase der Schwingung an. Bei einem Überlauf des Adresszeigers wird mittels einer Modulo-Operation wieder von vorne begonnen (Abb. 1). Die Modulo-Operation wird von den gängigen Synthesewerkzeugen nur für konstante Werte der Form 2^n beherrscht. Ursache ist das implizit in der Codierung vorzeichenloser Binärzahlen inkludierte Überlaufverhalten. Dieser Umstand muss bei der Festlegung der Anzahl der Tabelleneinträge berücksichtigt werden.

Es ist üblich, zur Darstellung der Phase eine höher auflösende Zahl (also ein breiteres Register mit

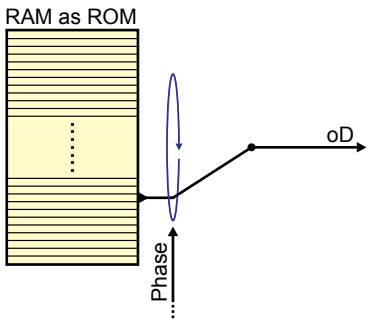

Abbildung 1: DDS-Datenpfad auf Basis einer ROM-Tabelle. Der Datenpfad besteht aus einem ROM, das als Tabelle zur Ablage einer Periode einer Sinusschwingung dient.

niederwertigeren Stellen als dem ursprünglichen LSB) zu verwenden, als es die Adresse eigentlich erfordert. Dies eröffnet die Möglichkeit, den Phaseninkrement genauer zu steuern, als es der Anzahl der Tabelleneinträge entspricht. Die zusätzlichen Bits im Register ermöglichen es, Schwingungen von beinahe beliebiger Frequenz zu erzeugen.

Am Datenausgang des ROMs kann die momentane Amplitude der Schwingung abgenommen werden. Die Block-RAM-Zellen der gängigen FPGA-Familien lassen sich als ROM verwenden, wobei am Ausgang bereits ein Register vorgesehen ist. Dieses kann als Ausgangsregister der Unit verwendet werden. Es ist aber auch möglich ein zusätzliches Register vorzusehen.

Nachfolgend finden Sie die Definition der Entity für die Unit `DspDds`. Bitte ignorieren Sie zunächst die Generics `gNrOfPhaseDitherBits` und `gWaveTableBitWidth`. Diese werden erst in einer verfeinerten Version der DDS benötigt.

```
1  entity DspDds is
2    generic (
3      gClkFrequency            : natural       := cDefaultClkFrequency;
4      gAudioBitWidth           : natural       := cDefaultAudioBitWidth;
5      -- How many bits does the phase register have?
6      gPhaseResolution         : natural       := 20;
7      -- How many bits of the phase register will be dithered?
8      gNrOfPhaseDitherBits     : natural       := 10;
9      -- How many bits do the waveform ROM table entries have?
10     gWaveTableBitWidth       : natural       := 14;
11     -- Precalculated waveform ROM table entries
12     gWaveTable               : aSetOfFactors := (
13       0.00076699, 0.002300969, 0.003834943, 0.005368907, ... ));
14   port (
15     iClk         : in std_ulogic;
16     inResetAsync : in std_ulogic;
17     -- Phase steps determining the frequency to be generated
18     iPhaseIncrement : in natural range 0 to 2**(gPhaseResolution-1)-1;
19     -- Sample rate sync strobe signal
20     iSampleStrobe : in std_ulogic;
21     -- Parallel digital audio data protocol
22     oVal : out std_ulogic;
23     oD   : out aAudioData(0 downto -(gAudioBitWidth-1)));
24 end DspDds;
```

a□4 Die in der dargestellten Entity fast vollständig fehlende Sinustabelle können Sie sich beispielsweise mit Matlab oder einer Tabellenkalkulation erstellen. Bitte beachten Sie, dass die Adressierung der Tabelle mittels Abschneiden der unteren Bits der Phase dazu führt, dass ein systematischer Rundungsfehler der Adresse auftritt. Es wird immer der Tabellenwert ausgewählt, der zum kleineren Phasenwert gehört. Es wird also stets ab- aber nie aufgerundet. Diesen Fehler kann man leicht ausgleichen, indem in der Tabelle die um den Phasenwinkel $\frac{2\pi}{2N}$ rad (N ist die Anzahl der Tabelleneinträge) versetzten Einträge abgelegt werden. Der erste Tabelleneintrag besitzt also nicht den Wert 0, sondern bei einer Tabelle mit beispielsweise 4096 Einträgen den Wert 0.00076699. Diesen Wert würde man bei einer echten Rundung mittig zwischen dem ersten und dem zweiten Tabelleneintrag finden.

Es wäre übrigens durchaus denkbar, die Sinuswerte mit Hilfe des Packages `math_real` (IEEE) zu berechnen, aber leider kommen nur wenige Synthesewerkzeuge hiermit zurecht. Nachteilig würde sich aber in diesem Fall möglicherweise die geringe garantierte Genauigkeit des Typs `real` in einigen VHDL-Versionen bemerkbar machen. Die Berechnung von Sinuswerten erfolgt in diesem Package in einer Vielzahl von Schritten unter Verwendung des CORDIC-Algorithmus. Jeder dieser Schritte ist mit einem Rundungsfehler behaftet. All diese Rundungsfehler summieren sich zu einer möglicherweise inakzeptablen Ungenauigkeit auf.

b☐3 Wie hoch kann die **Auflösung der Werte in der Sinustabelle** maximal gewählt werden und **wie viele Werte** können Sie dort maximal ablegen?

Versuchen Sie die Anzahl der Einträge für Ihre ROM-Tabelle so zu wählen, dassdie Bitbreite der ROM-Adresse in etwa gleich oder ein bis zwei Bit unter der Breite der abgelegten Sinuswerte liegt.

c☐6 Erstellen Sie nun eine Architecture `RtlSimple` für die Unit `DspDds`. Achten Sie schon früh darauf, dass vom Synthesewerkzeug wirklich **Block-RAM-Zellen** in der von Ihnen **erwarteten Anzahl** vorgesehen werden.

d☐1 Werden die theoretischen Werte aus dem letzten Aufgabenpunkt in der Praxis erreicht oder kann das Synthesewerkzeug die RAM-Zellen in der Praxis womöglich nicht in der von Ihnen angenommenen Weise nutzen?

e☐3 Binden Sie Ihre Beschreibung in eine Testbench ein, welche die erzeugte Schwingung in einer Wave-Datei abspeichert. Hierzu können Sie die Testbench einer bereits vorhandenen DSP-Unit abwandeln.

f☐4 Untersuchen Sie das **Spektrum** der erzeugten Schwingung für drei Frequenzen:

a) Verwenden Sie ein Phaseninkrement, bei dem stets *genau* ein **Tabelleneintrag getroffen** wird und

b) zwei weitere Phaseninkremente, bei denen dies häufig nicht der Fall sein wird. Nun soll also häufig ein Wert vom Phasenregister gewünscht werden, der irgendwo **zwischen zwei Tabelleneinträgen** liegt. Begründen Sie die Auswahl Ihrer Werte hinsichtlich der Erfüllung dieser Bedingung. Durch das Abschneiden von niederwertigeren Bits des Phasenregisters wird in diesem Fall derjenige Tabelleneintrag ausgewählt, der dem kleineren Phasenwinkel entspricht.

g☐3 Erstellen Sie für Ihre Beschreibung der DDS ein Testbed, das auf dem Rapid Prototyping Board eine Sinusschwingung ausgibt, deren Frequenz zyklisch den Bereich von 0 bis 20 kHz durchläuft („*Sweep*“). Sehen Sie für einen Durchlauf eine Zeit von mindestens 60 s vor. Beobachten Sie mit Hilfe des Audiotesters das Spektrum der Schwingung.

Tipp: Ein „Sweep“ lässt sich durch langsames, stetiges Erhöhen des Phaseninkrements erreichen, z.B. durch einen langsam ansteigenden Zähler. Der Maximalwert bzw. Überlaufwert des Zählers ergibt sich aus der maximal auszugebenden Frequenz.

Wichtig: Wenn Sie den Frequenzbereich zu schnell durchlaufen, dann können wegen der raschen Frequenzwechsel Nebenkeulen im Spektrum auftreten. Halten Sie daher unbedingt eine Sweep-Zeit **von mindestens 60 Sekunden** ein!

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-06` mit dem Endstand Ihrer Übungsausarbeitung.

$e^{j\pi} = -1 \iff$ *Wer sich umdreht, blickt in die entgegengesetzte Richtung.*
(frei nach LEONHARD EULER)

Rx-Gruppe

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

Wichtig: Für die Rx-Gruppen umfasst dieser Angabezettel die Übungen 6 und 7. Den Rx-Gruppen steht daher eine zusätzliche Woche zur Bearbeitung zur Verfügung (Abgabetermin „Übung 7“).

Tipp: Heben Sie sich die Bearbeitung dieses Übungszettels dennoch nicht bis zum letztmöglichen Zeitpunkt auf! Sie sollten bis zum Abgabetermit von Übung 6 zumindest Aufgabenpunkte im Ausmaß von ca. 20 Punkten fertigstellen um ausreichend Zeit für ev. auftretende Fragen zu haben.

FIR-Filter stellen eine zuverlässige Struktur zur Realisierung digitaler Filter dar. In dieser Übung soll ein solches Filter realisiert werden.

2. Aufgabe *FIR-Filter: einfache Struktur, riesige Hardware*

Die einfachste Art der Realisierung eines FIR-Filters ist die direkte Umsetzung einer geeigneten Struktur in der Hardware. Jedes Strukturelement wird unmittelbar durch einen entsprechenden Hardwareblock ersetzt. Eine einfache Struktur zur Implementation eines FIR-Filters ist die sogenannte Direktform oder Transversalfilterstruktur, wie sie in Abb. 2 zu sehen ist. Die Koeffizienten für diese Form lassen sich so skalieren, dass Fraktionalzahlen ($|b_i| < 1$) verwendbar sind. Werkzeuge zur Skalierung stellen beispielsweise die entsprechenden Matlab-Module zur Verfügung. Es finden sich jedoch auch freie Programme, mit welchen die Skalierung vorgenommen werden kann.

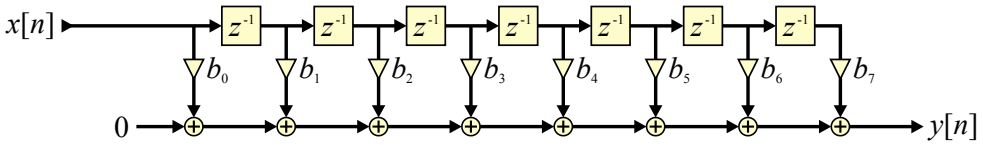

Abbildung 2: FIR-Filter in Direktform. Die Koeffizienten lassen sich skalieren: $|b_i| < 1$.

Die Implementierung des FIR-Blocks soll auch im Nachhinein mit einem beliebigen Satz aus einer beliebigen Anzahl von Koeffizienten parametrisiert werden können. Die Koeffizienten werden daher durch eine Generic-Constant in das Modell eingespeist:

```
1 entity DspFir is
2   generic (
3     gAudioBitWidth : natural       := cDefaultAudioBitWidth;
4     gCoefWidth     : natural       := 16;
5     -- The values used as filter coefficients. The number of those
6     -- coefficients determines the number of taps of the filter.
7     gB             : aSetOfFactors := ( ... ));
```

```
8   port (
9     iClk         : in std_ulogic;
10    inResetAsync : in std_ulogic;
11    -- Input audio channel
12    iDdry   : in aAudioData(0 downto -(gAudioBitWidth-1));
13    iValDry : in std_ulogic;
14    -- Output audio channel
15    oDwet   : out aAudioData(0 downto -(gAudioBitWidth-1));
16    oValWet : out std_ulogic);
17 end entity DspFir;
```

Nachdem die Umsetzung einer Struktur in einer VHDL-Beschreibung eher schlecht lesbar ist, sollte man sich zunächst mit einer Zeichnung die Orientierung erleichtern. Abb. 3 dient als Grundlage.

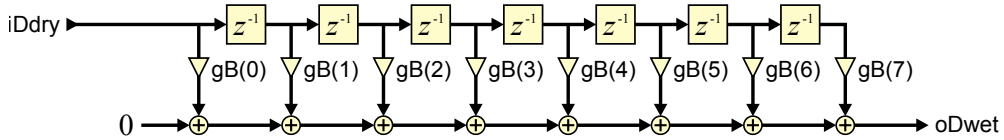

Abbildung 3: FIR-Filter in Direktform mit Namensvorschlägen für die VHDL-Implementierung

Weil die Multiplikation in Verbindung mit der Addition mehrerer Werte möglicherweise nicht mehr innerhalb eines Taktzyklus durchführbar ist, jedenfalls aber den kritischen Pfad der Schaltung bilden wird, sollten in diese Struktur Registerstufen zwischen der Betragskorrektur[1] und der Addition eingefügt werden, so wie dies in Abb. 4 dargestellt ist.

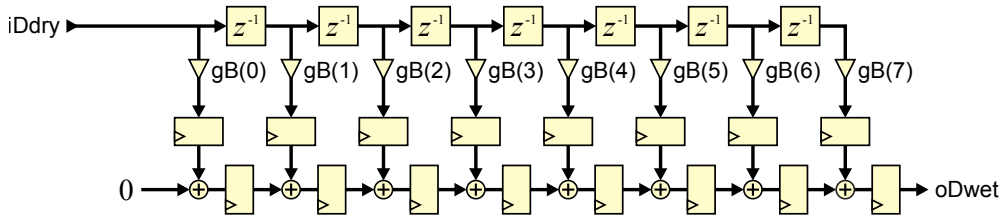

Abbildung 4: FIR-Filter in Direktform mit Registerstufen

Der kritische Pfad in einer Schaltung beginnt typischerweise an einem Register und endet an einem zweiten. Durch Einfügen eines weiteren Registers, wird der ursprüngliche Pfad in zwei Teile aufgespaltet. Jeder Teilpfad weist dann vermutlich eine erheblich kürzere Durchlaufzeit auf als der komplette Pfad. Die maximal mögliche Taktfrequenz rückt (hoffentlich) wieder in den Bereich der geforderten (beispielsweise 48 MHz). In der industriellen Praxis wird häufig eine bestimmte Taktfrequenz gefordert, weil Komponenten vorhanden sind, welche diese Frequenz zum Betrieb benötigen (z.B. USB, PCIe, SATA).

Damit für nachfolgende Verarbeitungsstufen eine einfache Timing-Analyse möglich wird, sollte das Endergebnis (`oWet`) über ein Register geführt werden.

a☐2 Die eingefügten Registerstufen stellen Pipeline-Stufen im Datenpfad des FIR-Filters dar. Wie viele Pipeline-Stufen ergeben sich aus Abb. 4 bei n Koeffizienten?

b☐1 Neben dem Datenpfad ist nun auch ein Steuerwerk notwendig, welches die Gültigkeit der Daten auf der Leitung `oDwet` anzeigt. Beschreiben Sie die Funktion dieses Steuerwerks kurz in Worten.

c☐1 Ist es möglich, in jedem Takt einen neuen Eingabewert in den Datenpfad zu schicken und erhält man pro Takt einen neuen Ausgabewert?

[1] Die Betragskorrektur folgt unmittelbar auf die Multiplikation. In einer der letzten Übungen haben Sie dafür die Funktion `ResizeTruncAbsVal` in Ihrem Package `Global` entwickelt.

Leider ist die vorgeschlagene Struktur für die Realisierung von Filtern nicht ohne weiteres einsetzbar, da pro Koeffizient b_i ein Multiplizierer benötigt wird. Zwar wird mit dem hierdurch notwendigen Hardwareaufwand eine enorme Verarbeitungsleistung zur Verfügung gestellt, aber eine derart hohe Anzahl an Multiplizierern und Addierern kann im FPGA auf dem Rapid Prototyping Board nicht zur Verfügung gestellt werden. Selbst die größten derzeit verfügbaren FPGA-Bausteine, bieten „nur wenige“ 100 Multiplizierer der notwendigen Bitbreite (Standard auf FPGAs der letzten Generation sind 18×18 Bit-Multiplizierer).

3. Aufgabe *Fläche gegen Zeit*

Die in der vorigen Aufgabe vorgestellte Struktur erfordert zwar eine enorme Fläche, bietet aber zugleich eine überragende Verarbeitungsgeschwindigkeit. Wie in vielen anderen Fällen, kann Fläche gegen Zeit eingetauscht werden. Hierzu wird ein Datenpfad entworfen, welcher alle notwendigen Berechnungen ermöglicht, aber diese nicht gleichzeitig durchführt. Im Datenpfad sind daher Register zum Ablegen von Zwischenergebnissen vorgesehen und an den Eingängen der Operatoren (hier sind dies „$*$“, „$+$“ und die Betragskorrektur) können verschiedene Werte aufgeschaltet werden.

Die Berechnung, welche alles in allem durchgeführt werden soll, wird in Teilschritte untergliedert. Jeder Teilschritt hat die Dauer eines (oder mehrerer) Takte. Der Ablauf der Teilschritte wird von einem Steuerwerk vorgegeben. Unter Umständen beeinflussen Berechnungsergebnisse diesen Ablauf über so genannte Statussignale (etwa „Ergebnis ist negativ“). Das Steuerwerk kann in Form einer (hierarchischen) FSM oder FSMD implementiert werden.

In Abb. 5 ist ein Vorschlag für den Datenpfad eines FIR-Filters nach Abb. 3 zu sehen. Der Ablauf der Datenverarbeitung mittels dieses Datenpfades ist folgender:

a) Die Register `b(0)` bis `b(7)` werden bei einem Reset mit den entsprechenden Konstanten (Filterkoeffizienten) initialisiert. Alle anderen Register werden mit dem Wert 0 initialisiert.

b) Wenn ein Wert (`iDdry`) am Eingang gültig wird (`iValDry`), dann wird dieser in das Schieberegister `x` hineingeschoben. `SelX0` muss entsprechend eingestellt sein. Die Werte, welche sich in diesem Register befanden, sind nun um ein Abtastintervall älter geworden, was durch das Verschieben um eine Stelle berücksichtigt wird. Der älteste Wert wird aus dem letzten Register hinausgeschoben und wird fortan nicht mehr benötigt.

c) Nun wird der Multiplexer `SelX0` so eingestellt, dass sich die Werte `x(i)` in der Registerkette `x` der Reihe nach an die Stelle 7 befördern lassen. Anfangs stehen auf diese Weise `x(7)` und aus dem Ringpuffer `b` der Koeffizient `b(7)` am Eingang des Multiplizierers an. Mit jedem weiteren Takt kann die jeweils folgende Multiplikation (`x(i)* b(i)`) durchgeführt werden.

d) Das Ergebnis der Multiplikation wird unmittelbar in einem Register zwischengespeichert. Dies geschieht, damit der Multiplikationsvorgang einen möglichst großen Anteil des zwischen zwei steigenden Taktflanken verfügbaren Zeitraumes belegen kann.

e) Der Inhalt des besagten Registers wird durch eine Betragskorrektureinheit geführt und an-

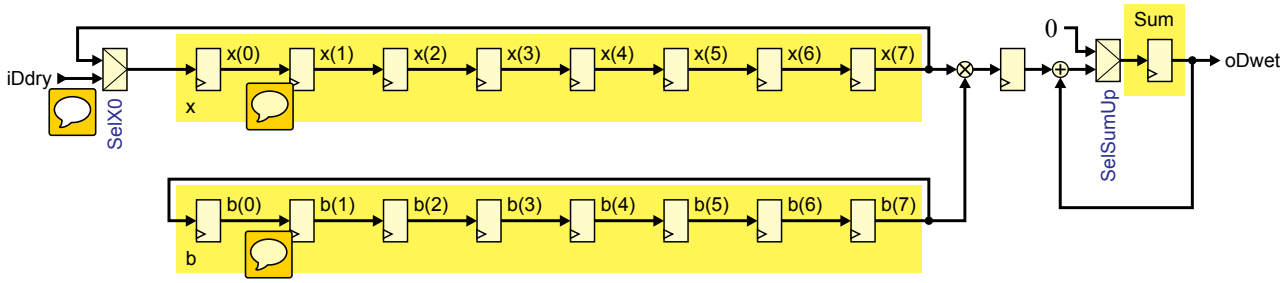

Abbildung 5: Datenpfad für ein FIR-Filter in Direktform. Die Bezeichnungen der Werte in den Registern stellen eine Momentaufnahme dar. Im Laufe der Verarbeitung ändern sich diese Inhalte.

schließend zu der bisher aufgelaufenen Summe aus dem Register `Sum` addiert und in eben diesem Register wiederum abgespeichert. Diese Summe wird zu Beginn der Berechung für jedes Sample mit dem Wert 0 initialisiert.

f) Am Ende der Berechnung steht im Register `Sum` der Ausgangswert zur Verfügung. Dessen Gültigkeit wird vom Steuerwerk durch ein `oValWet` angezeigt.

Für das Design des Steuerwerks bietet sich eine FSMD an. So kann der Vorgang des Aufsummierens über alle gewichteten und verzögerten Eingangswerte `x(i)` mittels eines Zählers erledigt werden, während der Zustand des Datenpfades in Form einer FSM verwaltet wird. Insgesamt ergibt sich ein Entwurf, der aus einem expliziten Datenpfad besteht, welcher von einem Steuerwerk mit implizitem Datenpfad (Zähler) verwaltet wird.

Der Entwurf der FSMD vereinfacht sich erheblich, wenn er durch eine entsprechende Zeichnung (z.B. Waveform) vorbereitet wird.

a☐$_1$ Überprüfen Sie den Datenfluss durch den Datenpfad im Detail und bringen Sie, falls notwendig bzw. sinnvoll, Verbesserungen am Datenpfad an. Beschreiben Sie Ihre Änderungen kurz.

b☐ Überlegen Sie, ob der FPGA-Baustein **genügend Flipflops** für den Gesamtentwurf bereithält.

Weit effizienter ist eine FPGA-Implementierung, bei der an Stelle der Schieberegister für die Koeffizienten und die verzögerten Samples jeweils ein Block-RAM tritt. In Abb. 6 ist ein Blockschaltbild dieser Struktur zu finden. Die Koeffizienten stammen aus einem als ROM verwendeten Block-RAM. Die Samples werden dagegen in einem Dual-Ported-RAM verwaltet. Die Auswahl der richtigen RAM-Zellen erfolgt durch Adressierung dieser RAMs mittels mehrerer Zähler. Die Ausgänge beider RAMs sollten über Register geführt werden, damit der Pfad, in welchem der nachfolgende Multiplizierer liegt so kurz wie möglich ausfällt. Dieses ausgangsseitige Flipflop ist in die Block-RAM-Zellen der meisten FPGA-Familien bereits integriert.

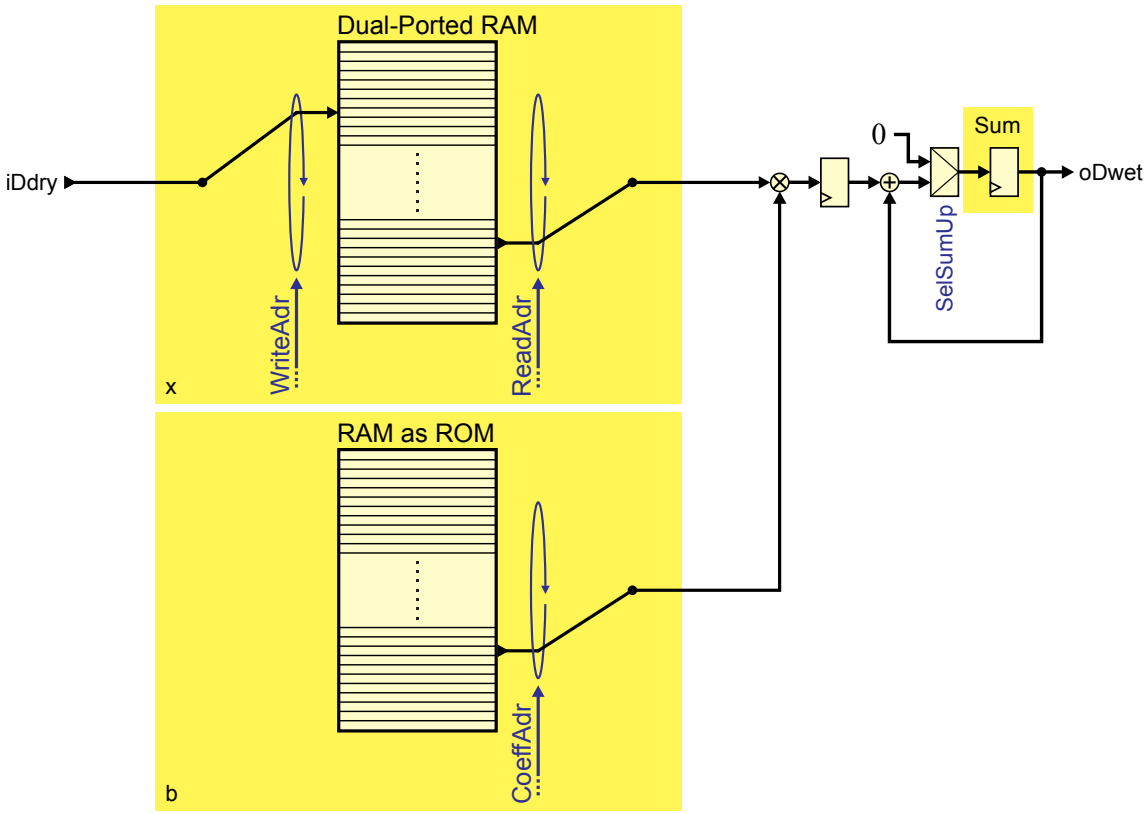

Abbildung 6: Datenpfad für ein FIR-Filter in Direktform unter Verwendung von Block-RAM-Zellen

c☐$_2$ Überprüfen Sie wiederum den Datenfluss durch den Datenpfad im Detail und bringen Sie, falls notwendig bzw. sinnvoll, Verbesserungen am Datenpfad an. Beschreiben Sie Ihre Änderungen kurz.

d☐$_4$ Entwickeln Sie den Datenpfad in einer Unit `DspFir` welche ein FIR-Filter mit Hilfe von Block-RAM realisiert. Die Architecture soll den Namen `RtlRam` erhalten.

e☐$_7$ Stellen Sie die Arbeitsweise eines passenden **Steuerwerks** in einem ***Timing-Diagramm*** dar. Zeichnen Sie dieses Diagramm!

f☐$_{10}$ Implementieren Sie dieses Steuerwerk in der Unit `DspFir(RtlRam)`.

g☐$_4$ Weisen Sie die Funktion mittels Simulation nach. Eine .wav-Datei, in welcher eine sinusförmige Wellenform abgelegt ist, deren Frequenz mit der Zeit ansteigt eignet sich hierzu gut. Damit wird die **Zeitachse im Simulator** quasi zur **Frequenzachse**. Geeignete Dateien sind beispielsweise `Wobble10Hz22050Hz100ms.wav` und `Wobble100Hz20kHz100ms.wav`.

h☐$_4$ Stellen Sie sicher, dass nicht nur Filter realisierbar sind, bei denen die Anzahl der Koeffizienten 2^n beträgt. Funktioniert Ihr Entwurf auch dann noch, wenn beispielsweise **255 Koeffizienten** verwendet werden?

i☐$_2$ Wie viele **Koeffizienten** lassen sich in einem RAM-Block der eingesetzten FPGA-Familie unterbringen? Entspricht die **Anzahl der RAM-Blöcke**, welche vom Synthesewerkzeug vorgesehen werden Ihren **Erwartungen**? Was ergibt sich beispielsweise bei einem Filter mit 703 Koeffizienten?

j☐$_2$ Wie viele **Koeffizienten** darf ein Filter der von Ihnen realisierten Struktur bei der verwendeten Abtastfrequenz **maximal** aufweisen? **Wodurch** wird die Anzahl **begrenzt**?

k☐$_3$ Erstellen Sie ein Testbed, welches einen sinnvollen Satz von Koeffizienten an diese Unit anlegt und die Unit auf dem Rapid Prototyping Board testbar macht.

l☐$_3$ Prüfen Sie die korrekte Funktion mittels des *Sweep Measurement* des Audiotesters. Entspricht die Dämpfung des Filters Ihren Erwartungen?

m☐$_2$ Welchem Filter entspricht der in der Entity der Unit `DspFir` angegebene Koeffizientensatz?

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-07` mit dem Endstand Ihrer Übungsausarbeitung.