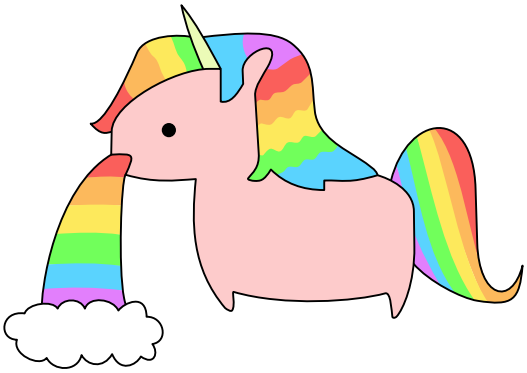
5. Übung: z^{-1} ... Delay (24 P)



Soooooo much noise!

1. Aufgabe *DspDly: Pseudostereo*

Die Verzögerung eines Audiosignals um genau eine (bzw. ein ganzzahliges Vielfaches der) Abtastperiode (z^{-1}) ist ein wichtiges Element beim Entwurf digitaler Filter.

Eine Verzögerung des Signals um einen Abtastwert bedeutet, dass jeder Wert nicht etwa sofort nach seiner Verfügbarkeit weitergegeben wird, sondern erst beim folgenden Abtastintervall. Hierzu muss dieser Wert zwischengespeichert werden.

Die Tiefe des erforderlichen Speichers für ein Abtastintervall ist 1. Der Initialwert dieses Speichers sollte „Stille“ sein. Letzteres wäre besonders beim Hintereinanderschalten beispielsweise 100.000 solcher Verzögerungsglieder wichtig: Enthielten diese zufällige Werte, so würde über mehr als zwei Sekunden ein Rauschsignal (der uninitialisierte Inhalt des Verzögerungsspeichers) ausgegeben. Erst danach würde das verzögerte Signal einsetzen.

In den Verzögerungsspeicher (der Tiefe 1) wird ein neuer Wert gespeichert, sobald dieser verfügbar ist, also gesteuert von `iVal`. Zugleich wird der zuletzt dort abgelegte Wert weitergegeben, was nichts anderes bedeutet, als dass auch `oVal` aktiviert wird, um der nachfolgenden Unit die Verfügbarkeit des Abtastwertes zu signalisieren. Damit sind die beiden Gültigkeitssignale `iVal` und `oVal` also identisch. Zwar könnte man das Signal `iVal` durch die Unit `DspDly` durchschleifen und als `oVal` wieder herausführen, aber die Unit würde damit einen Ausgang aufweisen, der nicht über ein Flipflop geführt ist. Dies sollte in Hinsicht auf die Verwendung in großen Designs vermieden werden, da das *Time Budgeting* des Synthesewerkzeugs erheblich beeinträchtigt würde.

Eine einfache Lösungsmöglichkeit wäre es, das Signal ebenfalls um ein Abtastintervall zu verzögern. Damit würde auch der oben erwähnte Zwang zur Initialisierung entfallen. Allerdings bedeutet diese Lösung auch einen Mehraufwand von einem Flipflop.

Eine weitere Lösungsmöglichkeit besteht darin, das Signal nur in die Unit `DspDly` *hinein*zuführen aber nicht *hindurch*zuführen. Sollen mehrere dieser Units verwendet werden, so werden diese sämtlich von einem zentralen Punkt aus mit dem Signal `iVal` versorgt.

a☐$_4$ Beschreiben Sie die Unit `DspDly` mit der Architecture `RtlReg` so, dass vom Synthesewerkzeug Flipflops zur Speicherung verwendet werden. Diese sollen, wie üblich, mittels Inference erzeugt werden. Beachten Sie, dass sich mit Hilfe des `for...loop` Statements (oder auch des `for...generate` Statements) eine Möglichkeit bietet, die gewünschte Funktion „Verzögerung um ein Abtastintervall“ mehrfach hintereinanderzusetzen. Wie oft dies geschehen soll, wird mittels des Generic `gDelay` gesteuert. Die einzelnen Stufen kommunizieren miteinander über die Elemente eines Arrays, welches Sie in der Architecture deklarieren müssen.

Die Schnittstelle der Unit ist:

```
entity DspDly is
  generic (
    gAudioBitWidth : natural := cDefaultAudioBitWidth;
    gDelay         : natural := 1);
  port (
    -- Sequential logic inside this unit
    iClk         : in std_ulogic;
    inResetAsync : in std_ulogic;
    -- Input audio channels
    iDdry   : in aAudioData(0 downto -(gAudioBitWidth-1));
    iValDry : in std_ulogic;
    -- Output audio channel
    oDwet : out aAudioData(0 downto -(gAudioBitWidth-1)));
    -- Output oValWet is identical to iValDry. Do not route a simple cable
    -- through a unit, because this will make time budgeting more difficult.
    --oValWet : out std_ulogic);
end DspDly;
```

b☐ Sichern Sie die Funktionsfähigkeit Ihrer Beschreibung mit der zur Verfügung stehenden Testbench zur Systemsimulation ab.

c☐1 Erstellen Sie ein Testbed für die Realisierung auf dem Rapid Prototyping Board. Um die Auswirkungen der Verzögerung hörbar zu machen, bietet sich ein Pseudostereoeffekt an, der auf der Verzögerung eines Kanals gegenüber dem anderen beruht. Dieser Effekt ist besonders ausgeprägt bei einer Verzögerung zwischen 4 und 10 ms. Als Signalquelle ist ein Musikstück (in Mono) gut geeignet, z.B. `Calexico16TrackScratchMono.wav`.

d☐2 Nutzen Sie so viele Flipflops wie irgend möglich (Datenblatt des FPGA, „Primary Logic Registers“ und „Secondary Logic Registers“). Der Nutzungsgrad, welcher während des *Placement and Routing* für die Flipflops angezeigt wird, soll mindestens 97 % betragen. Wie viele Stufen lassen sich auf dem Rapid Prototyping Board unterbringen? Welche Flächennutzung ergibt sich dadurch (ALM, LAB, Logic Registers)? Rechnen Sie bitte damit, dass jeder vollständige Durchlauf von Synthese und Place & Route für ein Testbed mit dieser umfangreichen Flächennutzung ca. 2 bis 4 Stunden in Anspruch nehmen wird. Überlegen Sie daher gut welche Werte Sie wirklich in der Synthese testen möchten. Ermitteln Sie die maximal möglichen Verzögerungsstufen (unter der Anahme dass keine zusätzliche Logik notwendig ist) zunächst theoretisch.

Wichtig: Aktuelle Quartus-Versionen optimieren Flipflop-basierte FIFOs *automatisch* zu RAM-basierten FIFOs. Um dieses Verhalten von Quartus Prime[1] zu deaktivieren, verwenden Sie den Menüpunkt *Assignments → Settings → Compiler Settings → Advanced Settings (Synthesis)* und stellen den Parameter *Auto Shift Register Replacement* auf ***Off***. Diese Einstellung lässt sich auch mittels eines Global Assignments konfigurieren:

```
set_global_assignment -name AUTO_SHIFT_REGISTER_RECOGNITION OFF
```

In Ihrer fhlow-Konfiguration können Sie dieses Global Assignment einfach mit Hilfe der folgenden Zeilen ergänzen (in den Konfigurationen der Units `DspDly` und `TbdDspDly` sind diese Zeilen bereits vorhanden):

```
append ChipGlobalAssignments {
    {AUTO_SHIFT_REGISTER_RECOGNITION OFF}
}
```

[1] In früheren Quartus-Versionen ist diese Einstellung unter dem Menüpunkt *Assignments → Settings → Analysis & Synthesis Settings → More Settings* zu finden.

2. Aufgabe *DspDly: Effizienter mittles Block-RAM*

Praktisch alle modernen FPGA-Bausteine bieten RAM-Blöcke, die als *Single-* oder *Dual-Ported*-RAM genutzt werden können. Diese RAM-Blöcke können einerseits mittels *Instantiation* und andererseits durch *Inference* in den Entwurf eingebunden werden. In dieser Aufgabe geht es darum die zweite genannte Methode am Beispiel des Verzögerungsgliedes `DspDly` anzuwenden.

In der ersten Aufgabe dieser Übung haben Sie zwar bereits festgestellt, dass moderne Synthesewerkzeuge auch Strukturen wie Schieberegister automatisch erkennen und zu RAM-basierten Schieberegisterblöcken optimieren können. Leider ist dieser Ansatz stark von den verwendeten Werkzeugen und FPGA-Technologien abhängig. Ziel dieser Aufgabe ist es daher, solche RAM-basierten Schieberegister aus Strukturen aufzubauen, die (plattformunabhängig) von allen gängigen Synthesewerkzeugen *per Inference* erkannt werden können.

Hinweis: Während dadurch eine plattformunabhängige VHDL-Beschreibung entsteht, kann diese Plattformunabhängigkeit auch dazu führen, dass optimierte Strukturen (wie dedizierte Schieberegisterblöcke) einzener FPGA-Architekturen nicht genutzt werden. In manchen FPGA-Projekten kann daher eine gezielte Optimierung für die Zieltechnologie oder die explizite Nutzung bestimmter Strukturen (durch *Instatiation*) durchaus sinnvoll sein.

a☐$_1$ Welche Speicherblöcke (abgesehen von den Primary/Secondary Logic Registers) stehen auf dem von Ihnen verwendeten FPGA zur Verfügung? Wie viele Speicherbits ergeben sich daraus insgesamt?

Muster für VHDL-Beschreibungen, welche vom Synthesewerkzeug in RAM-Blöcke umgesetzt wird, finden Sie in den entsprechenden Kapiteln der Dokumentation des Synthesewerkzeugs[2]. Für unsere Zwecke ist ein *Single-Clock Simple Dual-Port RAM* (Seite 12-13, Beispiel 12-13) gut geeignet:

```
ENTITY single_clock_ram IS
  PORT (
    clock         : IN STD_LOGIC;
    data          : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    write_address : IN INTEGER RANGE 0 to 31;
    read_address  : IN INTEGER RANGE 0 to 31;
    we            : IN STD_LOGIC;
    q             : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END single_clock_ram;

ARCHITECTURE rtl OF single_clock_ram IS
  TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL ram_block: MEM;
BEGIN
  PROCESS (clock)
  BEGIN
    IF (rising_edge(clock)) THEN
      IF (we = '1') THEN
        ram_block(write_address) <= data;
      END IF;
      q <= ram_block(read_address);  -- q gets old value during write
    END IF;
  END PROCESS;
END rtl;
```

b☐$_2$ Ist es auch möglich, ROM mittels Inference zu realisieren? Welche Möglichkeiten gibt es um ROM zu realisieren? (Sie sollten zumindest zwei Varianten nennen!)

c☐$_2$ Berechnen Sie, wie viele Verzögerungsstufen sich für ein 16 Bit breites Audiowort (ein Kanal) bei

[2] Quartus Prime Standard Edition Handbook Volume 1: Design and Synthesis, Kapitel 12: Recommended HDL Coding Styles → Inferring Memory Functions from HDL Code → Inferring RAM functions from HDL Code

optimaler Ausnutzung der im FPGA auf dem Rapid Prototyping Board befindlichen RAM-Blöcke verwirklichen lassen. Beachten Sie dabei, dass sich die Speicherblöcke nur in bestimmten Konfigurationen verwenden lassen! Welche Konfiguration werden Sie verwenden?

d☐5 Implementieren Sie nun eine Architecture `RtlRam` der Unit `DspDly`. Diese soll selbstverständlich Block-RAM als Speicher einsetzen. Auf Basis dieser RAM-Blöcke soll ein „Schieberegister“ erstellt werden, mit dem die Eingangsdaten vom Datentyp `aAudioData` um eine variable Anzahl an Verzögerungsstufen verzögert ausgegeben werden.

Auf der Eingabeseite des Schieberegisters wird in das RAM geschrieben und auf der Ausgabeseite daraus gelesen. Möglicherweise passiert beides gleichzeitig (dies ist sogar sehr wahrscheinlich). Werte, welche innerhalb des Schieberegisters liegen, interessieren uns im Rahmen der gegebenen Anforderungen nicht.

e☐2 Welche Rolle spielt die Initialisierung der RAM-Blöcke bei der Verzögerung von Audiosignalen? Wie können Sie die Initialwerte bestimmen (mind. zwei Möglichkeiten)? Welche Initialwerte werden verwendet, wenn Sie nichts unternehmen?

Die Adressierung auf der Eingabe- und der Ausgabeseite wird so durchgeführt, dass sich die beabsichtigte Länge des Schieberegisters (`gDelay`) ergibt. Zur Adressierung lässt sich beispielsweise ein Zähler verwenden.

Für Schieberegister der Länge $2^n - 1$ („maximum length sequence“) würde sich alternativ auch ein LFSR (*Linear Feedback Shift Register*) für die Adressierung eigenen. Hierbei werden gegenüber Zählern wesentlich kleinere Flächen (Anzahl von LUT und FF) bei gleichzeitig höherer Grenzfrequenz erreicht.

f☐2 Warum kommt es bei der Adressierung mittels LFSR zu einer wesentlich geringeren Flächennutzung bei gleichzeitig höherer Grenzfrequenz? Warum wächst dieser Vorteil bei sehr langen Schieberegistern sogar überproportional an?

Hinweis: Sie brauchen das Schieberegister dennoch ***nicht*** mit LFSR-Addressierungseinheit(en) zu implementieren.

g☐2 Erweitern Sie die Testbench der Unit `DspDly` so, dass Sie die korrekte Funktion Ihrer Beschreibung bei unterschiedlichen Werten für die Verzögerungstiefe automatisiert nachweisen.

h☐1 Synthetisieren Sie Ihre Beschreibung und prüfen Sie, ob tatsächlich Block-RAM-Zellen in der von Ihnen erwarteten Anzahl im Syntheseergebnis vorhanden sind.

i☐ Implementieren Sie Ihre Beschreibung auch auf dem Rapid Prototyping Board.

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-05` mit dem Endstand Ihrer Übungsausarbeitung.