Vorlesung Architekturen und Entwurf von Rechnersystemen

Prof. Andreas Koch, Yannick Lavan, Johannes Wirth, Mihaela Damian

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 2022/2023
Übungsblatt 4

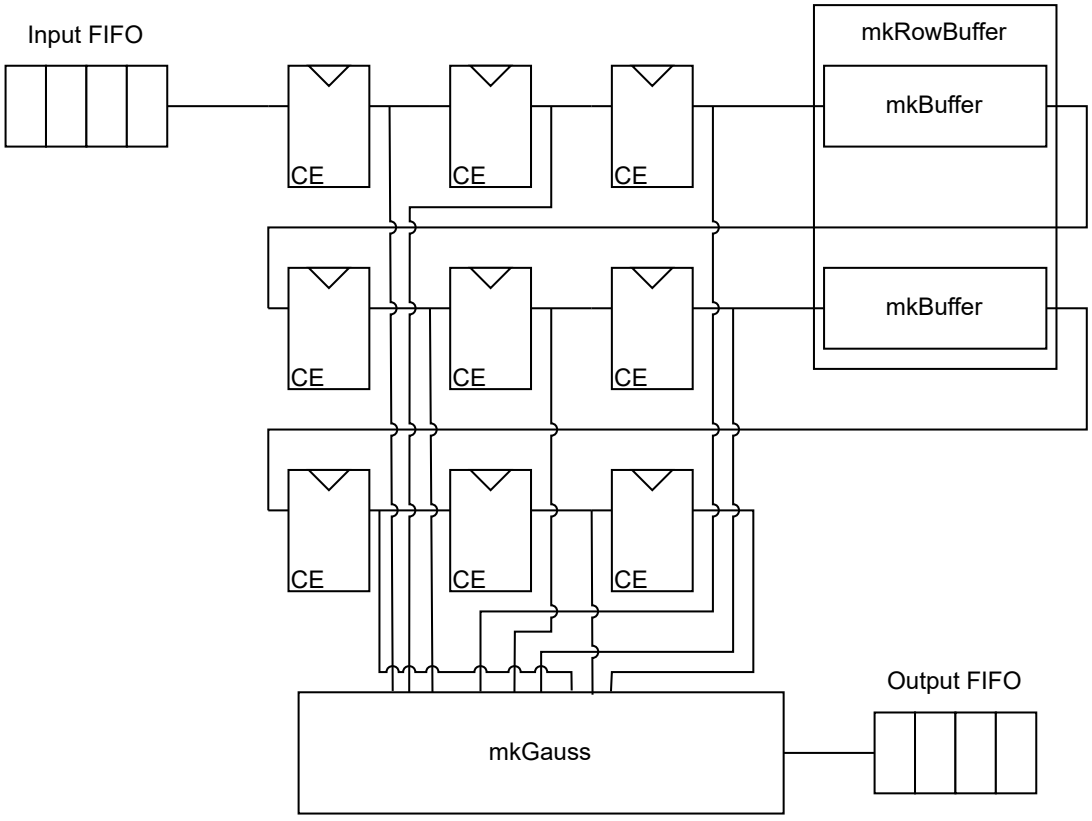

Abbildung 1: Architektur des Gauss-Filter basierten Designs.

In dieser Übung implementieren Sie mit Hilfe der in der vorherigen Übung entworfenen Untermodule den gesamten Beschleuniger. Die entsprechende Architektur ist noch einmal in Abbildung 1 zu sehen. Der Gauss Kernel, den wir in Übung 3 implementiert haben, wird wie ein Fenster über das gesamte Bild geschoben (*sliding window*).

Da der Filter ein Arbeitsfeld aus neun Pixeln benutzt, müssen wir im *sliding window* neun Pixel speichern. Diese werden anschließend an das `mkGauss`-Modul weitergereicht. Das ganze Bild wird **ohne Padding** gefiltert. Das bedeutet, dass alle Pixel bis auf die Randpixel des originalen Bildes einmal als mittlere Pixel in einem sliding window betrachtet werden und das Ergebnisbild zwei Pixel schmaler und zwei Pixel niedriger wird.

Die Implementierung des Beschleunigers findet im Modul `mkGaussAcceleratorstatt`. Sie finden den Rahmen des Moduls in der Datei `Top.bsv`.

Das Interface des Moduls ist wie folgt definiert.

```
typedef Server#(GrayScale, GrayScale) AcceleratorServer;

interface Accelerator;
    interface AcceleratorServer server;
    method Action setRes(UInt#(32) n_pixels);
    (* always_ready *)
    method Bool irq();
    method Action ack();
endinterface
```

Bevor der Beschleuniger mit der Verarbeitung beginnen kann, muss die Anzahl der Pixel im Bild über die Methode `setRes` übergeben werden. Pixel des Bildes werden reihenweise an das `server` Subinterface übergeben. Wenn der Beschleuniger das gesamte Bild verarbeitet hat soll ein Interrupt-Signal über die Methode `irq` ausgegeben werden. Das Interrupt-Signal darf erst wieder `False` werden, wenn der Interrupt über die Action-Methode `ack` zur Kenntnis genommen wurde. Erst wenn ein Interrupt generiert und über `ack` bestätigt worden ist, ist der Beschleuniger wieder bereit für ein neues Bild.

Aufgabe 4.1: Sliding Window und Kontrolllogik

Da die wesentlichen Untermodule schon implementiert wurden fehlen noch zwei wichtige Komponenten:

1. Einen Registervektor (Typ `Maybe#(GrayScale)`), der unser *sliding window* speichert,
2. Kontrolllogik, die verhindert, dass Randpixel in der Mitte des *sliding windows* an das Filtermodul weitergegeben werden[1].

Implementieren Sie die Hauptfunktionalität des Moduls `mkGaussAccelerator`. Dafür müssen Sie zunächst die Methode `setRes` implementieren und das `server` Subinterface definieren. Speichern Sie die Anzahl der Pixel in einem Register und verwenden Sie wie gewohnt FIFOs für die Eingabe und Entnahme von Pixeln in das und aus dem Modul. Sie können die Methoden `irq` und `ack` für diese Aufgabe ignorieren. Wir geben Ihnen für diese beiden Methoden eine Standardimplementierung vor, die zulässt, dass der Code kompiliert werden kann.

Instanziieren Sie in diesem Modul außerdem die Module `mkGauss` und `mkRowBuffer` und verbinden Sie diese sinnvoll mit Ihrem Registerfeld.

Sie können Ihren Code visuell debuggen, indem Sie ihn mit `make bluesim_AcceleratorTb` kompilieren und mit `./AcceleratorTb_bluesim` ausführen. Falls Sie eigene Bilder verwenden möchten, können Sie den Dateinamen in `AcceleratorTb.bsv` ändern und die Bildauflösung in `Settings.bsv` anpassen.

Hinweis: `DRegs` (Package `DReg`) sind praktische Register, die ihren Wert für genau einen Takt halten und dann wieder zu ihrem Standardwert zurückschalten.

Aufgabe 4.2: Wiederverwendbarkeit

Der implementierte Beschleuniger ist in der Lage ein einziges Bild zu verarbeiten. Wenn wir aber versuchen mehrere Bilder hintereinander zu verarbeiten, treten Probleme auf. Wir benötigen einen Weg die Buffer zu leeren, um die nicht mehr verwendeten letzten Reihen des vorherigen Bildes zu verwerfen. Erweitern Sie dafür die `Buffer` und `RowBuffer` Interfaces um eine zusätzliche `clear` Methode, die die Inhalte der internen FIFOs löscht[2]. Setzen Sie weiterhin bei einem Aufruf von `clear` die restlichen internen Zustände der Buffer-Module zurück.

[1] Sie können mal testen wie das Bild aussieht, wenn diese Kontrolllogik noch nicht vorhanden ist.
[2] Das `FIFO` Interface stellt hierfür eine Methode `clear` zur Verfügung.

Aufgabe 4.3: Berechnungsende

Der Beschleuniger muss dem restlichen Rechensystem in irgendeiner Form mitteilen, dass er mit der Bearbeitung eines Bildes fertig ist. In dieser Übung verwenden wir Interrupts. Typischerweise löst ein Rechenelement einen Interrupt aus, der von einem anderen Rechenelement in irgendeiner Form verarbeitet wird. Nach der Verarbeitung wird der Interrupt bestätigt (*acknowledged*). Implementieren Sie hierfür die Methoden `irq` und `ack`.

Zur Erinnerung:

- `irq` soll jeden Takt aufrufbar sein und nur `True` zurückliefern, sofern der Beschleuniger ein vollständiges Bild verarbeitet hat.
- `ack` soll nur einen Effekt haben, wenn der Interrupt über `irq` gesetzt ist und den Beschleuniger in seinen Ausgangszustand versetzen.

Aufgabe 4.4: Selbstprüfende Testbench

Implementieren Sie eine selbstprüfende Testbench. Wir stellen Ihnen wieder ein Orakel zur Verfügung.

Die Funktion `oracle_create(UInt#(64) input_addr, UInt#(32) width, UInt#(32) height)` akzeptiert als Parameter die Adresse des Eingangsbildes (Rückgabewert von `readImage_create(filename)`) und die Auflösung des Eingabebildes.

Die Funktion liefert einen Pointer zu einem Orakel-Objekt zurück, der an die Funktionen `oracle_delete` und `oracle_get_next_pixel` übergeben werden muss.

Die Funktion `oracle_get_next_pixel` liefert den nächsten gefilterten Pixel zurück, wenn man das Bild reihenweise verarbeitet.

Implementieren Sie Ihre Testbench als `StmtFSM` im Modul `mkAcceleratorChecker` (Datei `AcceleratorChecker.bsv`).

Aufgabe 4.5: *(Zusatzaufgabe)* Beliebige Bildbreiten

Aus gängigen Bildanwendungen kennen Sie die Möglichkeit beliebige Fotos laden und bearbeiten zu können. Der bisher implementierte Beschleuniger kann aber lediglich für eine feste Bildbreite verwendet werden.

Erweitern Sie den Beschleuniger, sodass die Breite des Bildes übergeben werden kann. Legen Sie dafür zunächst eine maximale Bildbreite fest, die der Beschleuniger unterstützt (z.B. 1920 bei 1080p Auflösung). Überlegen Sie anschließend welche Module Sie ändern müssen, um die neue Funktionalität umzusetzen. Wo benötigen Sie neue Register? Wie müssen die Interfaces erweitert werden?

Zum Testen können Sie eine visuelle Testbench schreiben, die ein Eingabebild filtert und anschließend das Ausgabebild bei immer kleiner werdenden Auflösungen wiederholt filtert. Das Bild sollte mit jedem Filtervorgang verschwommener wirken.