

camera_rotate

(Laborprojekt)

Vorlesung: Labor Rechnersysteme

Erik Alexander Hennig

tae@erik-hennig.me

Technische Akademie Esslingen



Inhaltsverzeichnis

1 Aufgabenstellung	1
2 Implementierung	1
2.1 Parameter	2
2.1.1 Pragmas	2
2.1.2 INPUT_PTR_WIDTH, OUTPUT_PTR_WIDTH und TYPE	2
2.1.3 COLS und ROWS	2
2.1.4 NPC und TILE_SIZE	3
2.2 Probleme	3
2.2.1 Rotation um 0 Grad	3
2.2.2 Dokumentation	3
3 Performance	4
4 Fazit	4
5 Bibliographie	5

1 Aufgabenstellung

Ziel des Laborprojekts ist die eigenständige Implementierung einer Bildverarbeitung in Hardware. Als Rahmenbedingung gegeben ist die Verwendung des Kria KV260 Vision AI Starter Kits sowie die Nutzung der Xilinx Toolchain für High-Level-Synthese. Darüber hinaus ist die Wahl des Projekts offen. In diesem Fall wurde als selbstgestellte Aufgabe gewählt, ein aufgenommenes Kamera-Bild unter Zuhilfenahme eines Orientierungssensors so zu rotieren, dass es immer aufrecht angezeigt wird. Dabei soll die Rotation performant genug sein, um ein flüssiges Live-Bild anzuzeigen.

2 Implementierung

Um das Performance-Ziel zu erreichen, soll die Rotation des aufgenommenen Bilds in Hardware implementiert werden. Die übrige Logik wird nur in Software implementiert. Abbildung 1 zeigt die architekturelle Umsetzung des Projekts. Der Mikrocontroller-Baustein in der Mitte stellt den nicht-programmierbaren Teil des Kria-Boards dar. Dieser kommuniziert mit dem FPGA-Teil via AXI und AXI-Lite. Die Aufnahme des (verdrehten) Live-Bilds erfolgt mit einer per USB angeschlossenen Webcam. Zur Bestimmung der Kameraorientierung wird ein modernes Smartphone genutzt. Solche Geräte verfügen in der Regel alle über einen Orientierungssensor und sind leicht zu integrieren, da sie über zahlreiche Wege kommunizieren können. Das Smartphone wird mittels einiger Gummibänder mechanisch an die Webcam gekoppelt, sodass beide die gleiche räumliche Orientierung aufweisen. Zum Auslesen der aktuellen Orientierung implementiert das Programm auf dem Kria-Board in einem extra Thread einen minimalistischen Webserver. Der in der Website enthaltene Javascript-Code liest den Orientierungssensor des Smartphones aus und sendet ihn per HTTP zurück zum Webserver. Anhand der erhaltenen Orientierung kann dann das Kamerabild auf dem FPGA um die passende Gradzahl rotiert werden. Die Ausgabe des Bildes erfolgt per HDMI auf einem externen Display oder per SSH mittels X11 Display Forwarding auf einem weiteren PC.

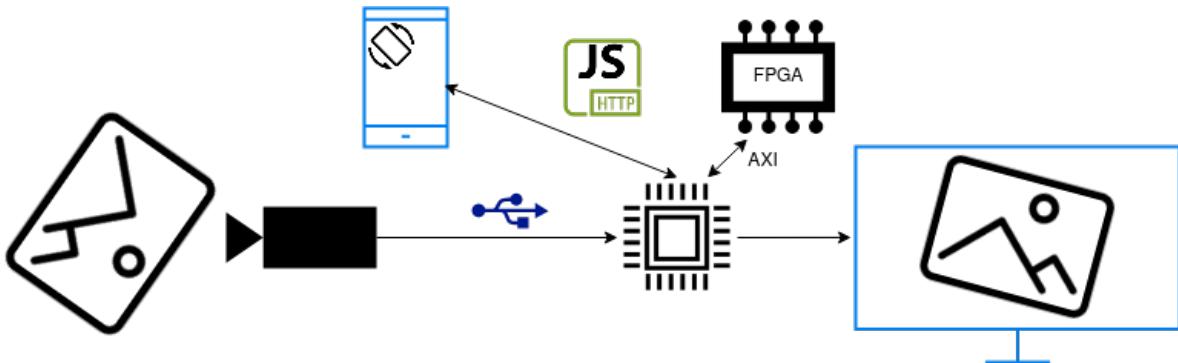


Abbildung 1: Kommunikationskanäle

Zur Umsetzung der Rotation in Hardware kommen folgende Funktionen der Vitis Vision Library auf den ersten Blick in Betracht:

- `remap` [1]: vertauscht Pixel anhand einer Relokationsmatrix
- `warpTransform` [2]: wendet eine affine Transformation auf das Eingangsbild an
- `rotate` [3]: rotiert das Eingangsbild um 90, 180 oder 270 Grad

Die ersten beiden Funktionen erlauben eine beliebige Rotationen. Allerdings sind sie nicht ganz so trivial zu verwenden wie `rotate`, das explizit nur für die Rotation implementiert wurde. Aus diesem Grund wurde `rotate` für die erste Implementierung genutzt und dann aus Zeitgründen auch nicht mehr ersetzt.

2.1 Parameter

2.1.1 Pragmas

Die Argumente für `rotate` wurden größtenteils einfach durchgereicht. Dabei wurde die Übergabe kleiner Argumente per AXI-Lite umgesetzt. Dies betrifft die tatsächliche Breite und Höhe des Bilds, den Rotationswinkel und die Steuerung des IP-Blocks. Die Übertagung der Eingabe- und Ausgabe-Bilddaten erfolgt über einen AXI-Bus. Dabei wurden folgende Parameter gewählt:

- `offset=slave`: Die Vitis-Dokumentation schreibt diesen Wert vor [4]
- `depth=__XF_DEPTH`: Repräsentiert die Größe des Adressbereichs [4], also die maximale Größe des Bilds in Bytes (= Höhe * Breite * Kanäle * Kanalbreite = $512 * 512 * 1 * 1 = 262144$)
- `bundle=gmem0/bundle=gmem1`: Eingabe- und Ausgabebild werden über getrennte Bundles übertragen, um Performance-Einbußen zu vermeiden [5].

Die Auflistung der INTERFACE-Pragmas sieht somit folgendermaßen aus:

```
#pragma HLS INTERFACE mode=s_axilite port=rows
#pragma HLS INTERFACE mode=s_axilite port=cols
#pragma HLS INTERFACE mode=s_axilite port=direction
#pragma HLS INTERFACE mode=s_axilite port=return

#pragma HLS INTERFACE mode=m_axi depth=__XF_DEPTH bundle=gmem0 port=src_ptr
offset=slave
#pragma HLS INTERFACE mode=m_axi depth=__XF_DEPTH bundle=gmem1 port=dst_ptr
offset=slave
```

2.1.2 INPUT_PTR_WIDTH, OUTPUT_PTR_WIDTH und TYPE

Der Template-Parameter `TYPE` gibt die Anzahl an (Farb-)Kanälen und die Bitbreite eines Kanals an. `INPUT_PTR_WIDTH` und `OUTPUT_PTR_WIDTH` geben ebenfalls die Breite eines einzelnen Eingabe- bzw. Ausgabepixels in Bit an. Unglücklicherweise klärt die Dokumentation die Doppelung im Pixelformat und daraus die resultierende Einschränkungen nicht auf. Mit den Werten `INPUT_PTR_WIDTH=OUTPUT_PTR_WIDTH=8` und `TYPE=XF_8UC1` wird die Funktion korrekt ausgeführt, aber andere Kombinationen führen bereits in der C-Simulation zu Segmentierungsverletzungen und Assertion-Fehlern, z.B. `INPUT_PTR_WIDTH=8, OUTPUT_PTR_WIDTH=16, TYPE=XF_8UC1`. Die Nutzung von 3-Kanal-Farbbildern mit `TYPE=XF_8UC3` scheint ebenfalls nicht möglich. Bei einer Eingabe-/Ausgabebreite von 8 Bit ergibt sich ein Assertionfehler und 24 Bit sind bereits laut Dokumentation untersagt, da nur 2er-Potenzen erlaubt sind.

2.1.3 COLS und ROWS

Diese Template-Parameter geben die Maximalbreite und -höhe des zu drehenden Bildes in Pixel an. Diese sind in einem gewissen Rahmen frei-wählbar, allerdings muss bei nicht-quadratischen Bildern darauf geachtet werden, die Maße des Ausgabebildes abhängig von der Rotation anzupassen. Andernfalls werden die Pixel durch fehlerhaft Interpretation der Pixelposition an der falschen Stelle dargestellt wie in Abbildung 2 zu sehen.



Abbildung 2: Rotation eines nicht-quadratischen Bildes

Geringe max size

300x200

All the functions in the library are implemented in streaming model except 4. Crop, EdgeTracing, MeanShiftTracking, Rotate are memory mapped implementations. These functions need to have the flag `_SDA_MEM_MAP_` set for compiling correctly https://xilinx.github.io/Vitis_Libraries/vision/2022.1/api-reference.html#id99

2.1.4 NPC und TILE_SIZE

2.2 Probleme

2.2.1 Rotation um 0 Grad

2.2.2 Dokumentation

Die Dokumentation der Vitis Vision Library hat an einigen Stellen zu Problemen geführt. Beispielsweise beschreibt sie, welcher Header für welche Funktion zu inkludieren ist, separat und für manche Funktionen, wie z.B. die genutzte `xf::cv::rotate`, fehlt ein Eintrag [6]. Somit war nicht direkt ersichtlich, was der korrekte Header ist. Ein weiteres Problem war die fehlerhafte Dokumentation von `rotate`. Als erlaubte Werte für den `direction` Parameter werden explizit 90, 180 und 270 genannt [3]. Bei Aufruf mit diesen Parameter ergibt sich jedoch immer eine Rotation um 90 Grad. Nach einiger Suche im Quellcode der Vision Library findet sich die schuldige Codestelle:

```
// xf_rotate.hpp:69
for (int k = 0; k < NPC; k++) {
    #pragma HLS UNROLL
    if(direction == 0){
        // rotation by 270 degrees
    }
    else if(direction == 1){
        // rotation by 180 degrees
    }
    else {
        // rotation by 90 degrees
    }
}
```

Somit sind die korrekten Parameter-Werte nicht 90, 180 und 270, sondern 2, 1 und 0, was der Dokumentation widerspricht. Mit diesen neuen Werte funktioniert die Rotation in eine beliebige der drei möglichen Richtungen.

3 Performance

4 Fazit

- viel gelernt
- Doku könnte besser sein
- persönlich: hat Spaß gemacht
- performance? measure first? oder HW ist schnell

Die vollständige Implementierung des Projekts findet sich unter https://github.com/ede1998/camera_rotate [7].

5 Bibliographie

- [1] AMD Xilinx, „Vitis Vision Library API Reference: Remap“. [Online]. Verfügbar unter: https://xilinx.github.io/Vitis_Libraries/vision/2022.1/api-reference.html#remap
- [2] AMD Xilinx, „Vitis Vision Library API Reference: Remap“. [Online]. Verfügbar unter: https://xilinx.github.io/Vitis_Libraries/vision/2022.1/api-reference.html#warp-transform
- [3] AMD Xilinx, „Vitis Vision Library API Reference: Remap“. [Online]. Verfügbar unter: https://xilinx.github.io/Vitis_Libraries/vision/2022.1/api-reference.html#rotate
- [4] AMD Xilinx, „Vitis High-Level Synthesis User Guide: Offset and Modes of Operation“. [Online]. Verfügbar unter: <https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Offset-and-Modes-of-Operation>
- [5] AMD Xilinx, „Vitis High-Level Synthesis User Guide: Offset and Modes of Operation“. [Online]. Verfügbar unter: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/M_AXI-Bundles
- [6] AMD Xilinx, „Using the Vitis vision Library“. [Online]. Verfügbar unter: https://xilinx.github.io/Vitis_Libraries/vision/2022.1/using-the-vitis-vision-library.html#id1
- [7] E. Hennig, „camera_rotate“. [Online]. Verfügbar unter: https://github.com/ede1998/camera_rotate