

Evaluation einer modernen Zynq-Plattform am Beispiel der Implementierung einer Hough Transformation

Verteidigung der Bachelorarbeit

Dominik Weinrich

dominik.weinrich@tu-dresden.de

Dresden, 23.08.2018

Gliederung

1. Aufgabenstellung
2. Motivation
3. Zielplattform
4. Implementierung
5. Auslagerung einzelner Komponenten in Hardware
6. Evaluation
7. Fazit
8. Ausblick

1. Aufgabenstellung

- Softwareimplementierung einer Hough Transformation
 - Grayscaler
 - Gauß-Filter
 - Canny Edge Detection
 - Circle Hough Transformation
- Auslagerung einzelner Komponenten auf den FPGA
- Evaluation

2. Motivation

- Anwendungsbereiche für Hough Transformation vielseitig
 - Erkennung von Passanten in selbst fahrenden Automobilen
 - Zählen von Objekten (Geld, GO-Spielsteine, ...) auf einem Bild
- Beschleunigung von rechenintensiven Aufgaben notwendig, um diese echtzeitfähig zu machen



Abb. 1: Go-Spielbrett [1]



Abb. 2: Euromünzen [2]

3. Zielplattform - Zynq Ultrascale+ MPSoC ZCU102

- 4 GB DDR4 RAM (512 MB geteilter Speicher)
- ARM Cortex-A53 64 Bit Vierkernprozessor
 - 32 KB L1 Cache
 - 1 MB L2 Cache
- Zynq Ultrascale XCZU9EG-2FFVB1156
 - 1.824 BRAM Blöcke je 18 Kb (32,1 Mb)
 - 2.520 DSPs
 - 548.160 FFs
 - 274.080 LUTs
 - 512 MB DDR4 RAM bei 2.666 Mbps



Abb. 3: ZCU102 Evaluation Board [3]

4. Implementierung

- Sprache: **C**
- Benutzte Bibliotheken/APIs:
 - **SDL2**
 - Kann verschiedene Bildformate laden
 - Leichter Zugriff auf die Roh-Pixeldaten
 - **OpenMP**
 - Einfache Möglichkeit zur Parallelisierung des Codes
 - Leichte Erstellung einer parallelen und seriellen Programmversion
- Modularer Aufbau für leichtere Anpassung während HLS

4. Implementierung - Grayscale



Abb. 4: Eingabebild [2]



Abb. 5: Ausgabebild

4. Implementierung - Gauss-Filter



Abb. 6: Eingabebild



Abb. 7: Ausgabebild

4. Implementierung - Canny Edge Detection



Abb. 8: Eingabebild

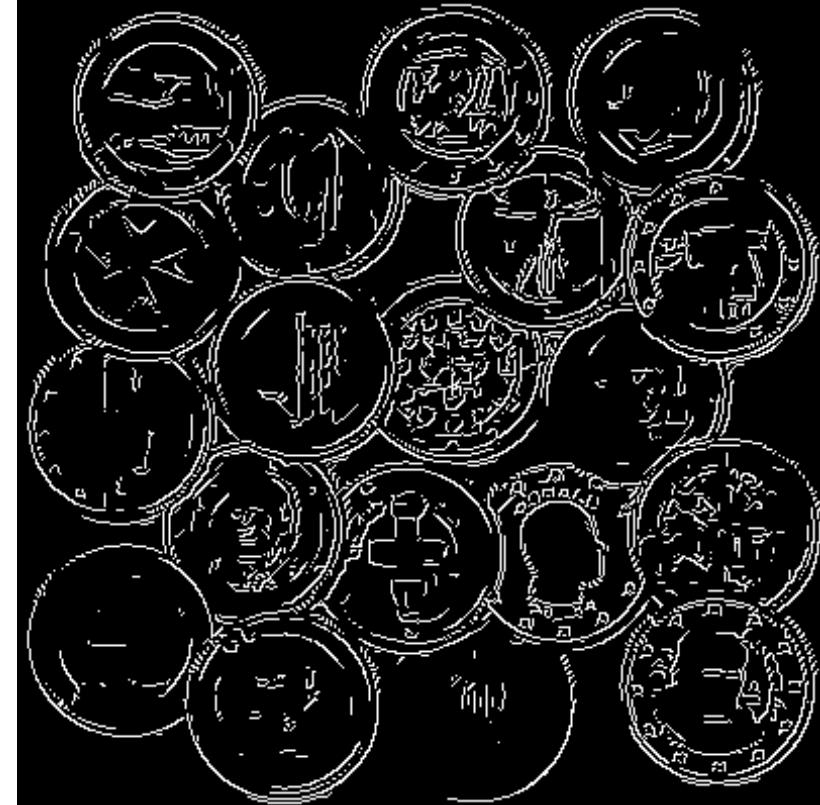


Abb. 9: Ausgabebild

4. Implementierung – Circle Hough Transformation

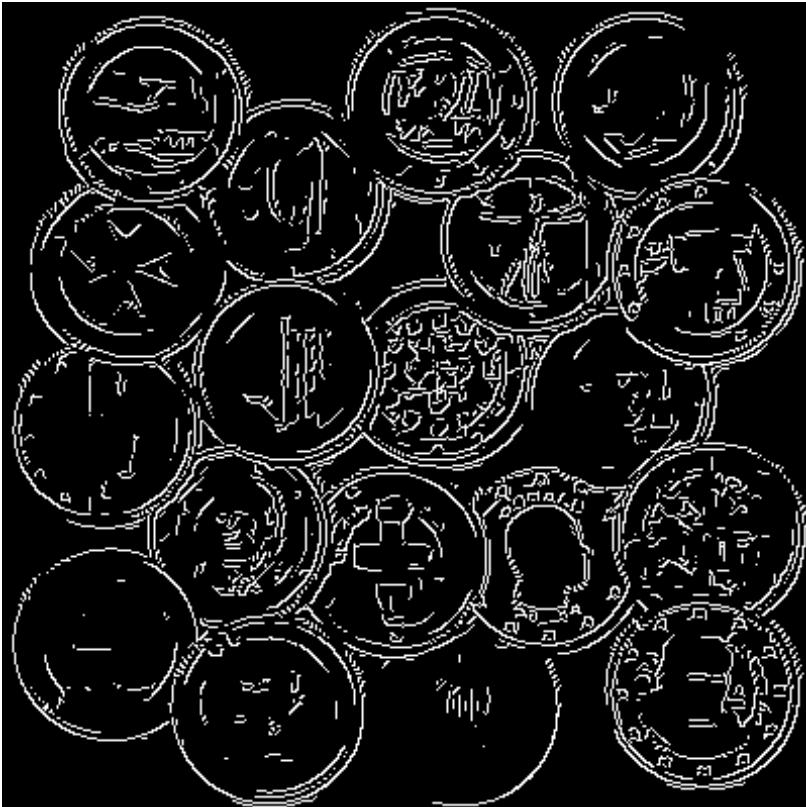


Abb. 10: Eingabebild



Abb. 11: Ausgabebild

4. Implementierung - Laufzeitvergleich

Tabelle 1: Laufzeitvergleich der Software auf dem ARM Cortex A53

	200 x 200	400 x 400	800 x 800	1.200 x 1.200
Grayscaler	1 ms	4 ms	14 ms	33 ms
Gauss-Filter	4 ms	23 ms	96 ms	216 ms
Canny	21 ms	81 ms	329 ms	733 ms
CHT	94 ms	1.044 ms	24.759 ms	274.957 ms
Gesamt	120 ms	1.152 ms	25.198 ms	275.939 ms

- Votingverfahren der CHT bestimmt maßgeblich die Gesamlaufzeit
- Laufzeit bricht für größere Bilder ein (L2-Cache des ARM ist nicht groß genug)

5. Auslagerung einzelner Komponenten in Hardware

- Auslagerung der Komponenten mittels Vivado HLS 2017.4
- Auslagerung des Grayscales
 - Einfaches Beispiel für eine High Level Synthese
 - Besitzt ein hohes Potential, da hochgradig parallelisierbar
- Auslagerung der Circle Hough Transformation
 - Modul mit höchster Komplexität und daher auch größter Laufzeit
 - Laufzeit fast ausschließlich von diesem Modul abhängig

5. Auslagerung - Grayscaler

- Auslagerung für ein Bild der Größe von 400 x 400 Pixel
- Schritte der HLS
 - 1) Änderung der Berechnungsmethode des Intensitätswertes auf integerbasierende Methode
 - $\text{output}[\text{index}] = (30 * \text{r} + 59 * \text{g} + 11 * \text{b}) / 100$
 - 2) Pipelining der inneren Schleife
 - 3) Pipelining der äußeren Schleife & Ausrollen der inneren Schleife

5. Auslagerung - Grayscaler

Tabelle 2: Auswertung der Synthese des Grayscalers für ein 400 x 400 Pixel großes Bild

	Keine Optimierungen	Integerbasierte Berechnung	Pipelining d. inneren Schleife	Pipelining d. äußeren Schleife
Laufzeit Software [ms]	4,000 ms	4,000 ms	4,000 ms	4,000 ms
Laufzeit [ms]	26,404 ms	3,204 ms	0,800 ms	0,002 ms
Speedup	0,15	1,25	5	2.000
BRAM 18K	0 (0%)	640 (35%)	640 (35%)	1.600 (88%)
DSP48E	25 (1%)	3 (0%)	4 (0%)	1.200 (48%)
FF	2.837 (1%)	332 (0%)	445 (0%)	80.190 (15%)
LUT	3.017 (1%)	374 (0%)	466 (0%)	64.599 (24%)
FPGA-Ressourcen	1%	9%	9%	44%
Speedup ÷ FPGA-Ressourcen	15	13,89	55,56	4.545,45

5. Auslagerung - Grayscale

- Begrenzender Faktor ist der BRAM
- Sehr hoher Speedup mit sehr hohem Ressourcenaufwand
- Herausforderungen für größere Bilder (ab 800 x 800 Pixel)
 - Es müssen Puffer für das Eingabefeld verwendet werden, um den BRAM-Verbrauch zu verringern
 - Die innere Schleife kann nicht mehr vollständig ausgerollt werden
 - Der Speedup gegenüber der Softwarelösung wird kleiner

5. Auslagerung - CHT

- Auslagerung für ein Bild der Größe von 400 x 400 Pixel
- Akkumulatorfeld in Software als *unsigned int* (32 Bit) implementiert, in Hardware als *uint9*
→ 10,08 Mb BRAM benötigt
- Schritte der HLS
 - 1) Pipelining der Bresenham Schleife (Teil des Votingverfahrens)
 - Zyklische Partitionierung des Akkumulatorfeldes mit Faktor 8
 - Setzen der DEPENDENCE Direktive, um falsche Datenabhängigkeiten zu kennzeichnen
 - 2) Pipelining der Clearing Schleife
 - 3) Pipelining der Initialisierung des Akkumulatorfeldes
 - Ausrollen der Initialisierungsschleife des Akkumulatorfeldes mit einem Faktor von 16

5. Auslagerung - CHT

Tabelle 3: Auswertung der Synthese der CHT für ein 400 x 400 Pixel großes Bild

	Keine Optimierungen	Pipelining d. Bresenham	Pipelining d. Clearing	Pipelining d. Initialisierung
Laufzeit Software [ms]	1.044,00 ms	1.044,00 ms	1.044,00 ms	1.044,00 ms
Laufzeit [ms]	527.762,90 ms	251.732,43 ms	134,70 ms	123,85 ms
Speedup	0,002	0,004	7,75	8,43
BRAM 18K	681 (37%)	686 (37%)	686 (37%)	686 (37%)
DSP48E	6 (0%)	14 (1%)	19 (1%)	19 (1%)
FF	1.674 (0%)	3.035 (1%)	3.932 (1%)	3.911 (1%)
LUT	2.699 (1%)	6.588 (2%)	8.651 (3%)	8.764 (3%)
FPGA-Ressourcen	10%	10%	11%	11%
Speedup ÷ FPGA-Ressourcen	0,02	0,04	70,45	76,74

5. Auslagerung - CHT

- Anzahl der benötigten Schleifendurchläufe oft unbekannt oder bei inneren Schleifen variabel
→ Kein Ausrollen der inneren Schleifen und damit kein Pipelining der äußeren Schleifen möglich
- Verwendung von Puffern für Akkumulator zur Verringerung des BRAM-Verbrauchst bei größeren Bildern (ab 800 x 800 Pixel)

6. Evaluation

- Zur Evaluierung der Plattform wurden Messdaten für vier verschiedene Bildgrößen aufgenommen
 - 200 x 200 Pixel
 - 400 x 400 Pixel
 - 800 x 800 Pixel
 - 1.200 x 1.200 Pixel
- Das Akkumulatorfeld wurde mit einem zweidimensionalen *Memory Window Buffer* implementiert, um den Speicherplatzbedarf zu reduzieren
 - BRAM ist damit für Bildgrößen bis 5.793 x 5.793 Pixel (33,56 Megapixel) ausreichend
 - Ladezeiten können für gegebene Bildgrößen durch eine ausreichende Puffergröße kaschiert werden

6. Evaluation

Tabelle 4: Laufzeit und Speedup der von der HLS erzeugten Hardware

Bildgröße [Pixel]	200 x 200	400 x 400	800 x 800	1.200 x 1.200
Laufzeit [ms]	32,00	123,85	1.080,42	4.172,64
Benötigte Ladezeit [ms]	0,6	4,26	33,13	106,98
Laufzeit + Ladezeit [ms]	32,6	128,11	1.113,55	4.279,62
Laufzeit Software [ms]	94,00	1.044,00	24.759,00	274.957,00
Speedup	2,88	7,81	22,23	64,25

6. Evaluation

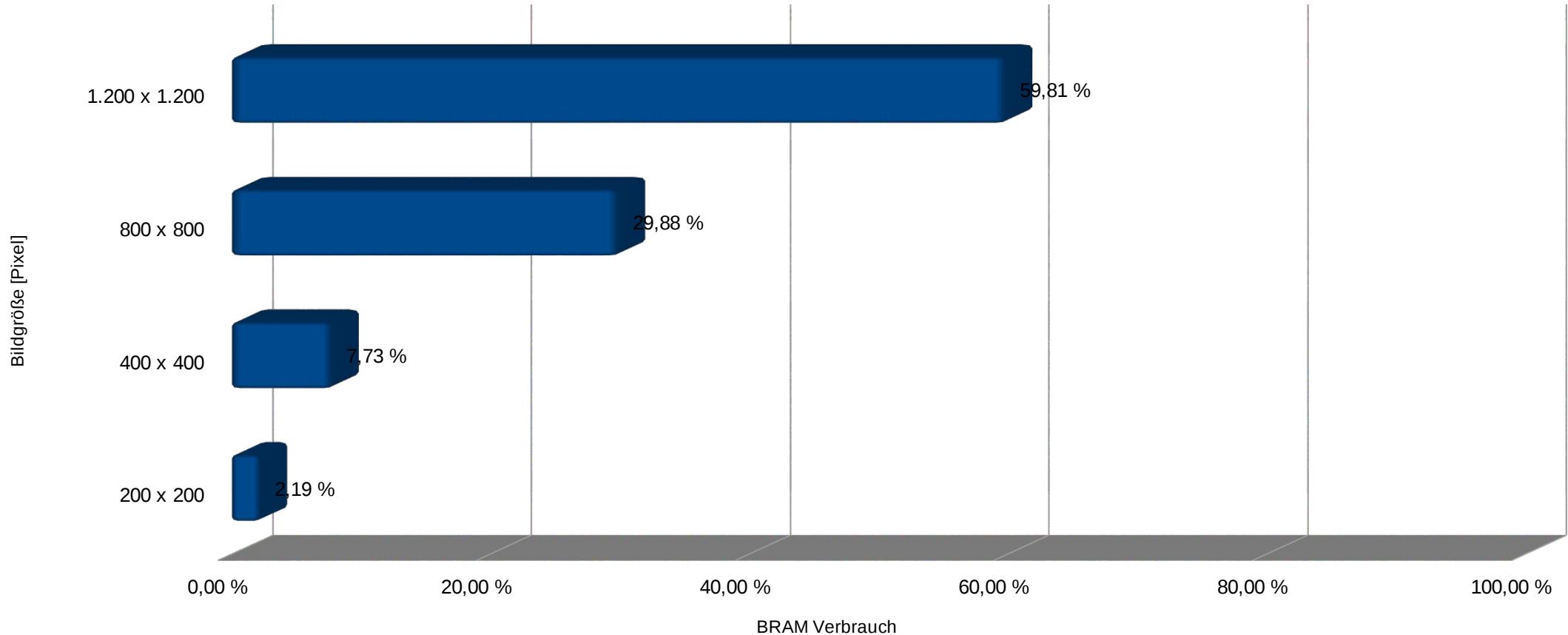


Abb. 12: BRAM Verbrauch der erzeugten Hardware

6. Evaluation

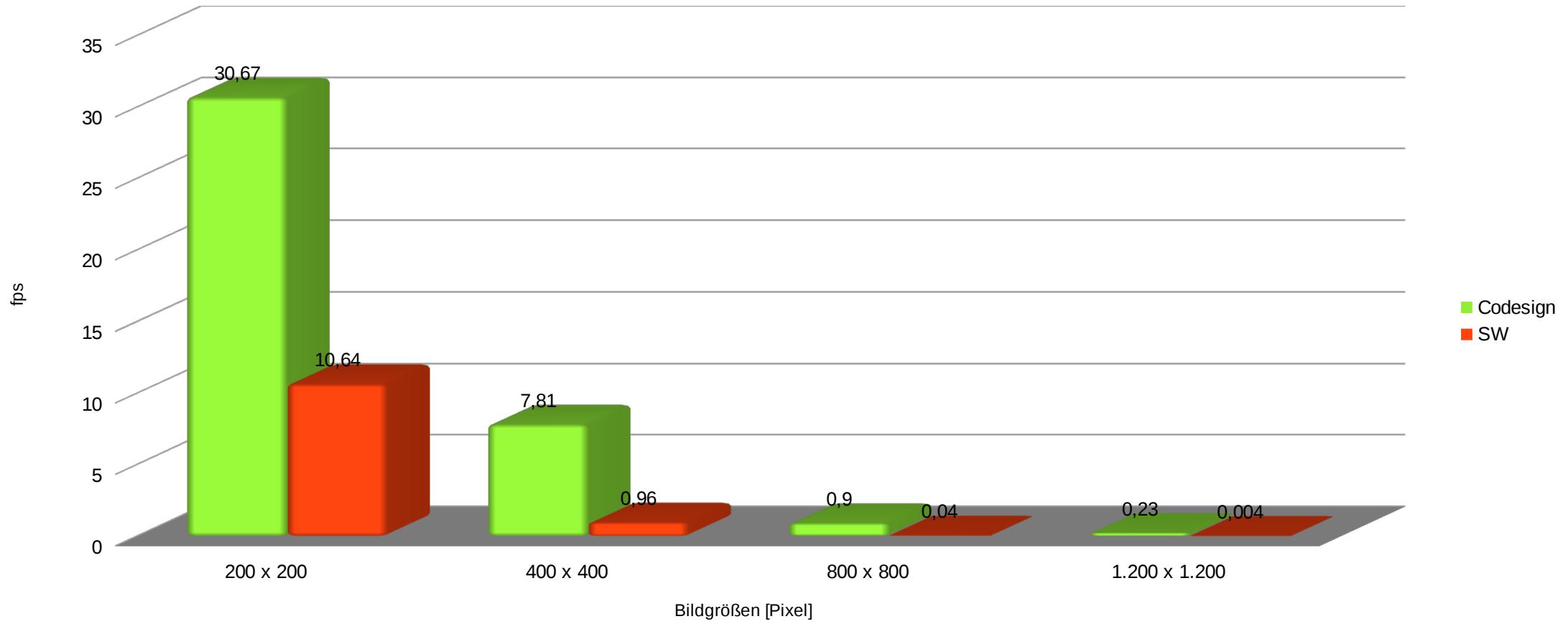


Abb. 13: fps-Vergleich zwischen Codesign und Software

7. Fazit

- Das Zynq Ultrascale+ MPSoC eignet sich gut für die Umsetzung eines Hardware/Software Codesigns
 - Das Board bietet durch CPU und FPGA mit gemeinsamen Speicher und AXI4 als Schnittstelle die Grundvoraussetzungen
 - Die von Xilinx bereitgestellte Software eignet sich gut für die HLS
 - Das resultierende Design kann über die Software einfach auf den FPGA und die CPU aufgespielt werden
 - Probleme ergaben sich bei dem Versuch das Codesign über ein Betriebssystem zu starten. Treiber müssen hierfür manuell angepasst werden
- Durch Implementierung der HT in Hardware geht Flexibilität verloren (Parameter müssen fest implementiert werden)
- Die Hough Transformation kann mit den implementierten Optimierungen für größere Bilder nicht mehr in Echtzeit realisiert werden
- BRAM ist begrenzender Faktor für die Optimierung des Designs

8. Ausblick

- Weitere Optimierungsmöglichkeiten
 - Umschreiben der Softwarelösung, um feste Iterationsgrenzen zu erhalten
→ Pipelining und Ausrollen von weiteren Schleifen möglich
 - Realisierung des Eingabefeldes für die CHT als *uint1*
→ Einsparung von BRAM
- Offene Fragen:
 - Welche Möglichkeiten gibt es, um die Softwarelösung weiter zu Beschleunigen?
 - Kann Hough Transformation auch für hochauflösende Bilder auf Echtzeit beschleunigt werden?

Quellen

- [1] Go-Spielbrett: <https://www.japanwelt.de/media/image/go-spiel.jpg>, 30.05.2018
- [2] Euro-Münzen: <http://www.historia-hamburg.de/media/product/1ec/19-x-1-euro-satz-aus-19-euro-staaten-511.jpg>, 30.05.2018
- [3] ZCU102 Evaluation Board: <https://www.xilinx.com/content/dam/xilinx/imgs/kits/whats-inside/ZCU102-evaluation-board.jpg>, 22.08.2018
- [4] Xilinx (Hrsg.): *Vivado Design Suite User Guide*. San Jose: Xilinx, 2014
- [5] Xilinx (Hrsg.): *ZCU102 Evaluation Board User Guide*. San Jose: Xilinx, 2017

Weiterführende Literatur

- Burger, Wilhelm ; Burge, Mark J.: *Principles of Digital Image Processing: Core Algorithms*. London : Springer, 2009. - ISBN 978-1-84800-194-7
- Burger, Wilhelm ; Burge, Mark J.: *Principles of Digital Image Processing: Fundamental Techniques*. London : Springer, 2009. - ISBN 978-1-84800-190-9
- Hauck, Scott (Hrsg.) ; André, DeHon (Hrsg.): *Reconfigurable Computing*. Burlington : Morgan Kaufmann Publishers, 2008. - ISBN 978-0-12-370522-8
- Ha, Soonoi ; Teich, Jürgen: *Handbook of Hardware/Software Codesign*. Dordrecht : Springer, 2017. - ISBN 978-94-017-7267-9
- Märtin, Christian: *Einführung in die Rechnerarchitektur*. Leipzig : Fachbuchverlag Leipzig, 2003. - ISBN 978-34-462-2242-7