

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

## Verteidigung der Bachelorarbeit

Dominik Weinrich  
[dominik.weinrich@tu-dresden.de](mailto:dominik.weinrich@tu-dresden.de)

Dresden, 23.08.2018

# Gliederung

- Aufgabenstellung
- Motivation
- Implementierung
- Auslagerung einzelner Komponenten in Hardware
- Evaluation
- Fazit

# Aufgabenstellung

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

# Motivation

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



**Abb. 1:** Go-Spielbrett [1]



**Abb. 2:** Euromünzen [2]

# Zielform - Zynq Ultrascale+ MPSoC

- 4 GB DDR4 RAM
- 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

# Implementierung

- Sprache: **C**
- Benutzte Bibliotheken/APIs:
  - **SDL2**
    - Kann verschiedene Bildformate laden
    - Leichter Zugriff auf die Roh-Pixel Daten
  - **OpenMP**
    - Einfache Möglichkeit zur Parallelisierung des Codes
    - Es kann leicht eine parallele und eine serielle Version des Programms erstellt werden
- Modularer Aufbau für eine leichtere Anpassung für die HLS

# Implementierung - Grayscale



**Abb. 3:** Eingabebild [2]



**Abb. 4:** Ausgabebild



# Implementierung - Gauss-Filter



**Abb. 5:** Eingabebild



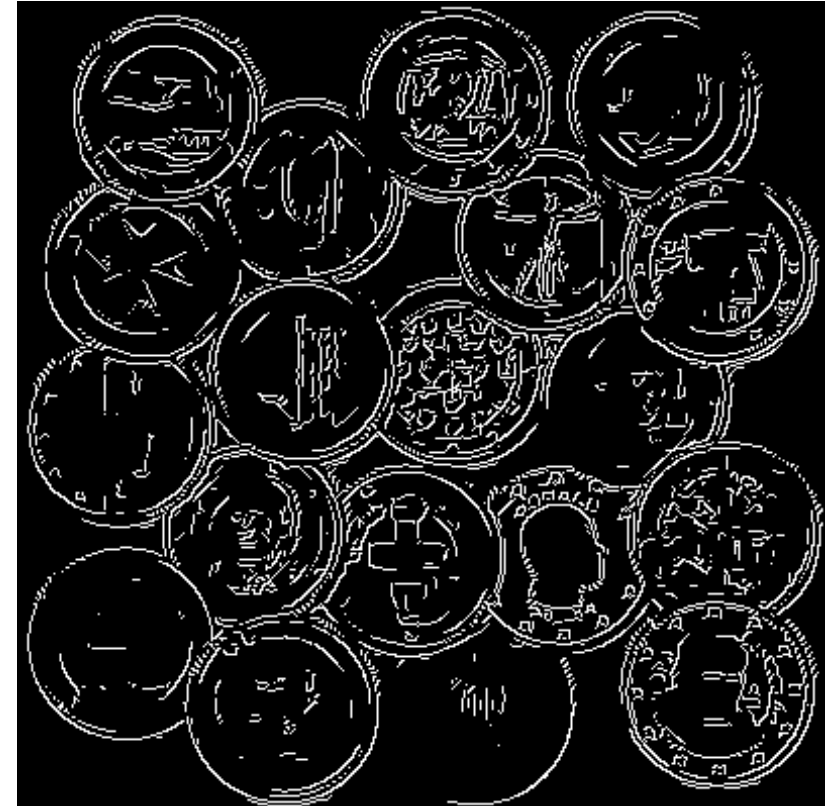
**Abb. 6:** Ausgabebild



# Implementierung - Canny Edge Detection

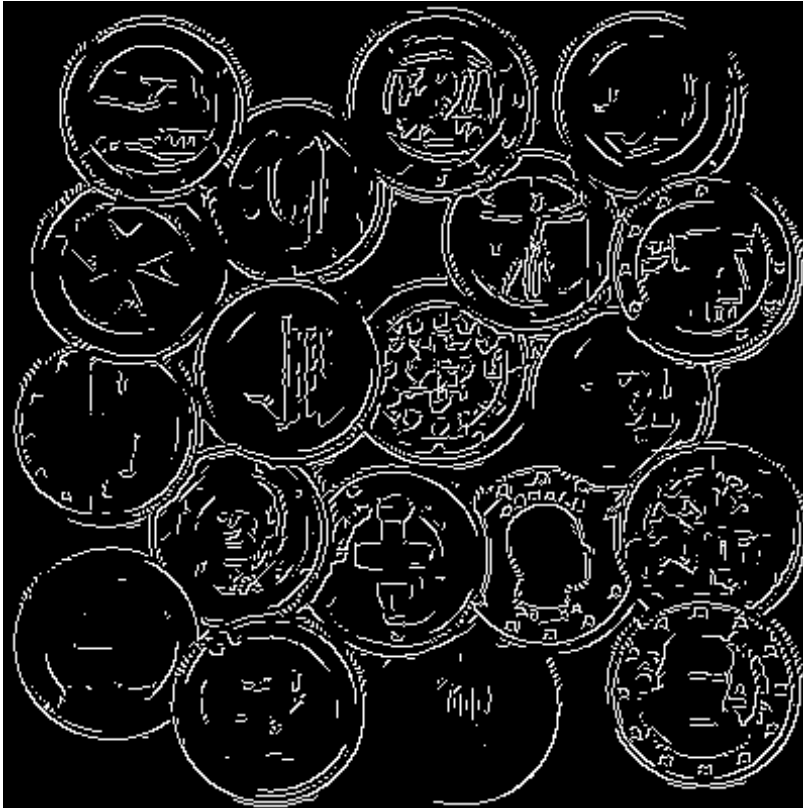


**Abb. 7:** Eingabebild

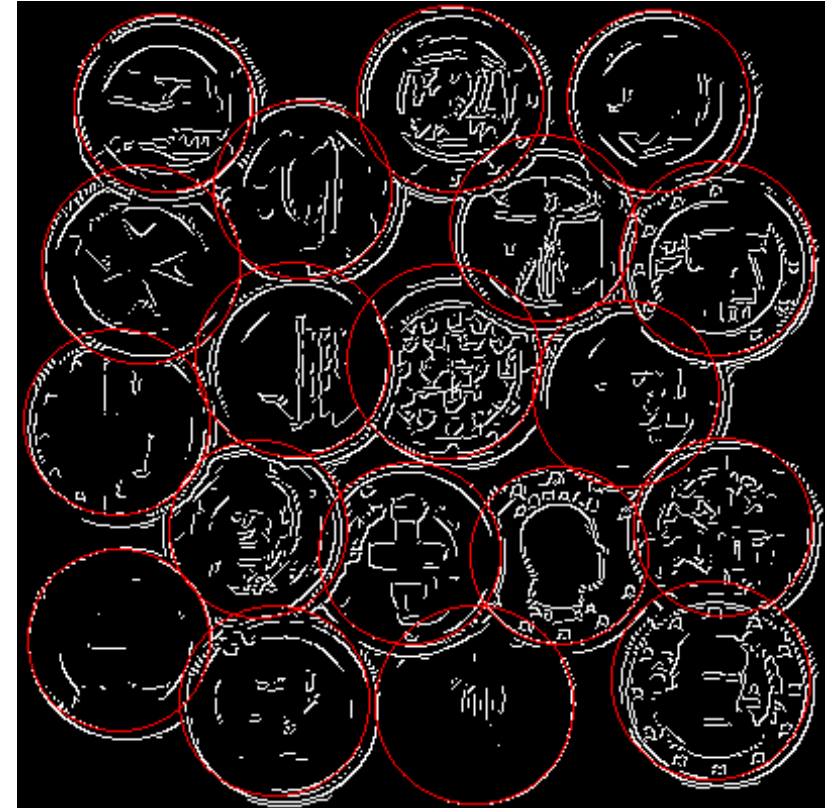


**Abb. 8:** Ausgabebild

# Implementierung - Canny Edge Detection



**Abb. 9:** Eingabebild



**Abb. 10:** Ausgabebild

# Implementierung - Laufzeitvergleich

**Tabelle 1:** Laufzeitvergleich der verschiedenen Module

	<b>200 x 200</b>	<b>400 x 400</b>	<b>800 x 800</b>	<b>1.200 x 1.200</b>
<b>Grayscale</b>	<b>1 ms</b>	<b>4 ms</b>	<b>14 ms</b>	<b>33 ms</b>
Gauss-Filter	4 ms	23 ms	96 ms	216 ms
Canny	21 ms	81 ms	329 ms	733 ms
<b>Hough</b>	<b>94 ms</b>	<b>1.044 ms</b>	<b>24.759 ms</b>	<b>274.957 ms</b>
<b>Gesamt</b>	<b>120 ms</b>	<b>1.152 ms</b>	<b>25.198 ms</b>	<b>275.939 ms</b>

- Das Votingverfahren des Houghmoduls bestimmt maßgeblich die Gesamtlaufzeit.
- Die Softwarelösung bricht für größere Bilder ein

# Auslagerung einzelner Komponenten in Hardware

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

# Auslagerung - Grayscale

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

# Auslagerung - Grayscale

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

	Keine	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
<b>Speedup</b>	0,15	1,25	<b>5</b>	<b>2.000</b>
<b>BRAM 18K</b>	0 (0%)	640 (35%)	640 (35%)	<b>1.600 (88%)</b>
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%)
<b>FPGA-Ressourcen</b>	1%	9%	<b>9%</b>	<b>44%</b>
<b>Speedup ÷ FPGA- Ressourcen</b>	15	13,89	<b>55,56</b>	<b>4.545,45</b>



# Auslagerung - Grayscale

- Begrenzender Faktor ist der BRAM
- Sehr hoher Speedup mit sehr hohem Ressourcenaufwand
- Herausforderungen für größere Bilder (ab 800x800 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

# Auslagerung - Hough

- Auslagerung für ein Bild der Größe von 400 x 400 Pixel
- Anpassung an Software notwendig
  - Realisierung des Akkumulatorfeld als *uint9* im BRAM (benötigt 10,08 Mb)
  - Realisierung des Eingabefeldes als *uint1* (nicht berücksichtigt)
- 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 des Akkumulatorfeldes mit einem Faktor von 16

# Auslagerung - Hough

**Tabelle 3:** Auswertung der Synthese des Houghmoduls für ein 400 x 400 Pixel großes Bild

	Keine	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
<b>Speedup</b>	0,002	0,004	7,75	<b>8,43</b>
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%)
<b>FPGA-Ressourcen</b>	10%	10%	11%	<b>11%</b>
<b>Speedup ÷ FPGA-Ressourcen</b>	0,02	0,04	70,45	<b>76,74</b>

# Auslagerung - Hough

- Anzahl der benötigten Schleifendurchläufe oft unbekannt oder bei inneren Schleifen variabel  
→ Kein Ausrollen der inneren Schleifen und damit kein Pipelining der äußeren Schleifen möglich
- Für größere Bilder (ab 800 x 800 Pixel) müssen Puffer für das Akkumulatorfeld verwendet werden, um den BRAM-Verbrauch zu verringern

# Evaluation

- Zur Evaluierung wurden Messdaten für vier verschiedene Bildgrößen aufgenommen
  - 200 x 200
  - 400 x 400
  - 800 x 800
  - 1.200 x 1.200
- Zur Implementierung des Akkumulatorfeldes wurde für alle Größen ein zweidimensionaler *Memory Window Buffer* verwendet, 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
  - Zeit um Puffer einmalig zu füllen kann vernachlässigt werden

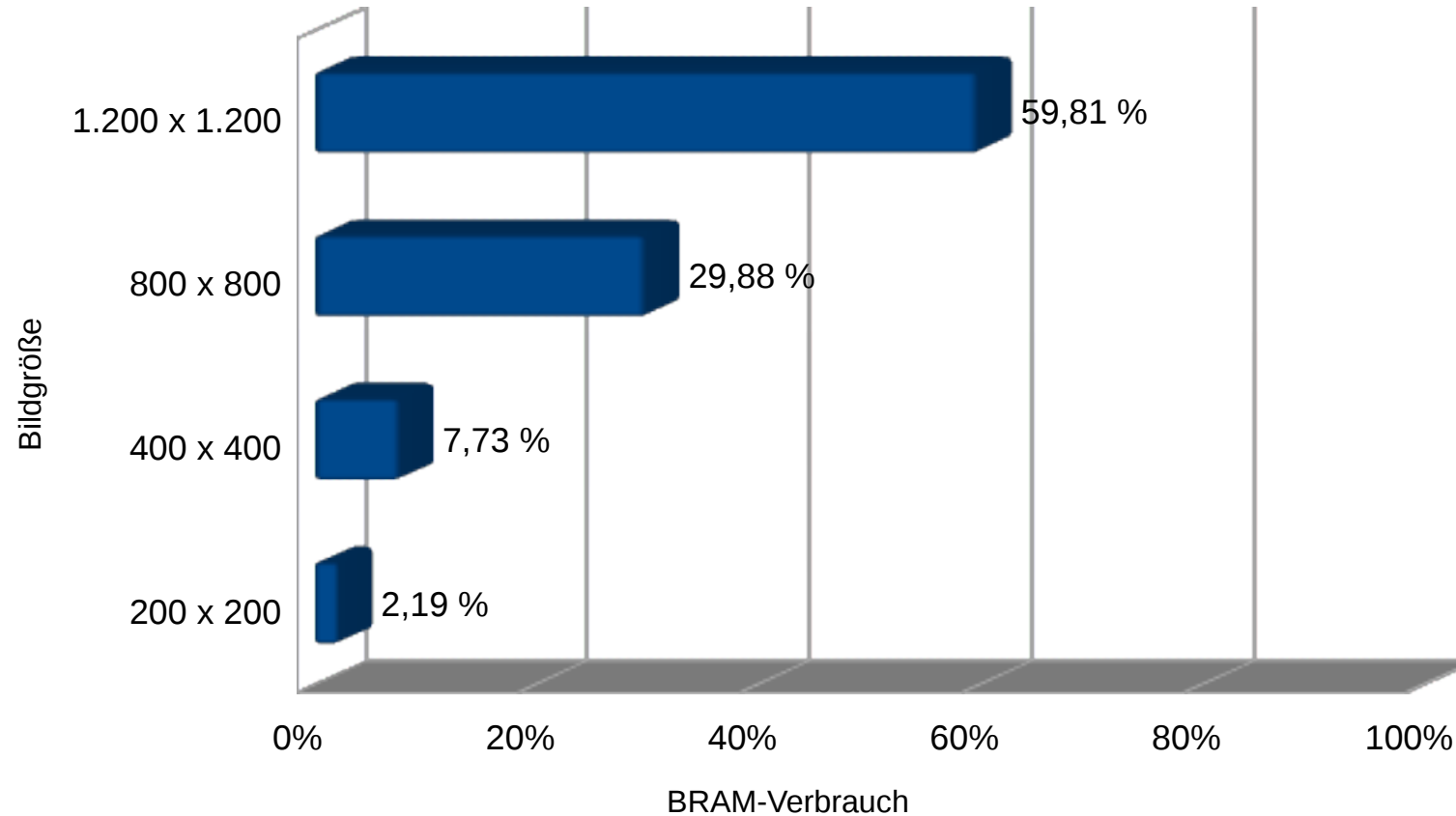
# Evaluation

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

	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
<b>Speedup</b>	<b>2,88</b>	<b>7,81</b>	<b>22,23</b>	<b>64,25</b>

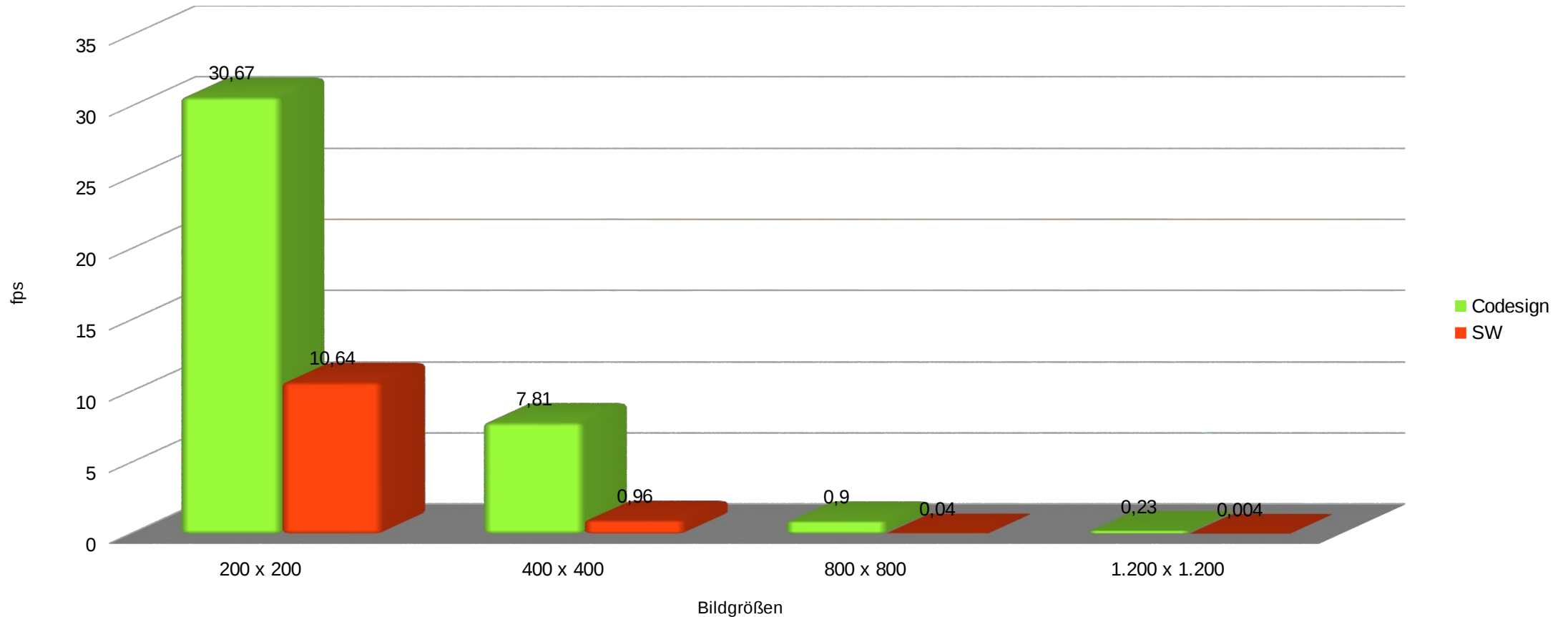


# Evaluation



**Abb. 11:** BRAM Verbrauch der erzeugten Hardware

# Evaluation



**Abb. 12:** fps-Vergleich zwischen Codesign und Software

# Evaluation

- Echtzeitfähigkeit nur für die Bildgröße von 200 x 200 Pixel gegeben
- Für die betrachteten Bildgrößen sind noch ausreichend FPGA-Ressourcen frei  
→ Es gibt noch Potential zur weiteren Optimierung, dafür muss der Code allerdings weitgehend umgeschrieben werden
- Großer Speedup bei größeren Bildern. Die Laufzeit der Softwarelösung ist ab einer Bildgröße von 800 x 800 Pixel sehr hoch

# Fazit

- Das Zynq Ultrascale+ MPSoC eignet sich gut für die Umsetzung eines Hardware/Software Codesigns
  - Das Board bietet mit CPU und FPGA die Grundvoraussetzungen hierfür
  - 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
- Die Circle Hough Transformation kann mit den implementierten Optimierungen für größere Bilder nicht mehr in Echtzeit realisiert werden
- Es gibt noch weitere Möglichkeiten zur Optimierung der HLS und des daraus resultierenden Designs. Der begrenzende Faktor ist hierbei der BRAM des FPGA.

# 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] Ahmad, Ijaz; Moon, Inkyu ; Shin, Seok J.: Color-to-grayscale algorithms effect on edge detection – A comparative study. In: *Electronics, Information, and Communication (ICEIC), 2018 International Conference on IEEE*
- [4] Burger, Wilhelm ; Burge, Mark J.: *Principles of Digital Image Processing: Core Algorithms*. London : Springer, 2009. – ISBN 978-1-84800-194-7
- [5] Burger, Wilhelm; Burge, Mark J.: *Principles of Digital Image Processing: Fundamental Techniques*. London : Springer, 2009. - ISBN 978-1-84800-190-9
- [6] Burger, Wilhelm; Burge, Mark J.: *Digital Image Processing – An Algorithmic Introduction Using Java*. 2. London : Springer, 2016. - ISBN 978-1-4471-6683-2
- [7] Bresenham, Jack: *A Linear Algorithm for Incremental Digital Display of Circular Arcs*. (1977), S. 100–106

# Quellen

- [8] Coussy, Phillipe ; Gajski, Daniel D. ; Meredith, Michael ; Takach, Andres: *An Introduction to High-Level Synthesis*. (2009), S. 8-17
- [9] Hauck, Scott (Hrsg.) ; André, DeHon (Hrsg.): *Reconfigurable Computing*. Burlington : Morgan Kaufmann Publishers, 2008. – ISBN 978-0-12-370522-8
- [10] Ha, Soonoi ; Teich, Jürgen: *Handbook of Hardware/Software Codesign*. Dordrecht : Springer, 2017. – ISBN 978-94-017-7267-9
- [11] Kanan, Christopher ; Cottrell, Garrison W.: *Color-to-Grayscale: Does the Method Matter in Image Recognition*. (2012)
- [12] *Fahrerassistenzsysteme im Visier*. Markt & Technik (Mai 2008)
- [13] Martin, Christian: *Einführung in die Rechnerarchitektur*. Leipzig : Fachbuchverlag Leipzig, 2003. – ISBN 978-34-462-2242-7
- [14] Xilinx (Hrsg.): *Vivado Design Suite User Guide*. San Jose: Xilinx, 2014



# Quellen

- [15] Xilinx (Hrsg.): *Processor System Reset Module v5.0*. San Jose: Xilinx, 2015
- [16] Xilinx (Hrsg.): *AXI Interconnect v2.1*. San Jose: Xilinx, 2017
- [17] Xilinx (Hrsg.): *ZCU102 Evaluation Board User Guide*. San Jose: Xilinx, 2017
- [18] Xilinx (Hrsg.): *UltraScaleArchitecture and Product Data Sheet: Overview*. San Jose: Xilinx, 2018
- [19] Xilinx (Hrsg.): *Zynq UltraScale+ MPSoC Processing System v3.2*. San Jose: Xilinx, 2018