mle
missing link electronics

08. Oktober 2014


Konferenz für ARM-Systementwicklung

Beschleunigen von Algorithmen mit High-Level Synthese auf Xilinx Zynq

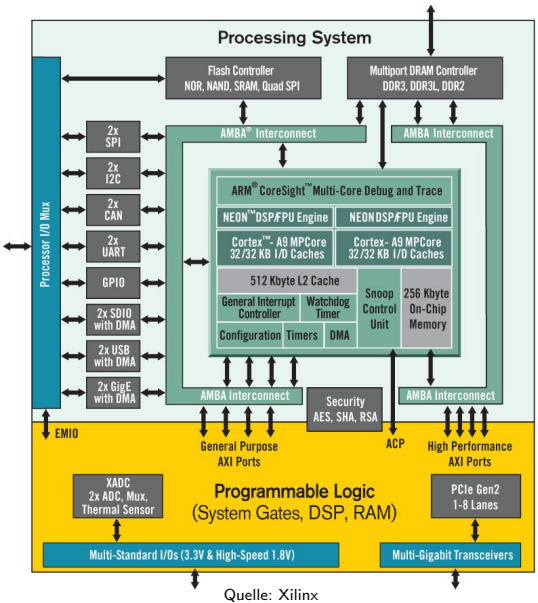
Florian Hagel, Missing Link Electronics GmbH

We are
a Silicon Valley based technology company with offices in Germany. We are partner of leading electronic device and solution providers and have been enabling key innovators in the automotive, industrial, test & measurement markets to build better Embedded Systems, faster.

Our Mission is
to develop and market technology solutions for Embedded Systems Realization via pre-validated IP and expert application support, and to combine off-the-shelf FPGA devices with Open-Source Software for dependable, configurable Embedded System platforms.

Our Expertise is
I/O connectivity and acceleration of data communication protocols, additionally opening up FPGA technology for analog applications, and the integration and optimization of Open Source Linux and Android software stacks on modern extensible processing architectures.

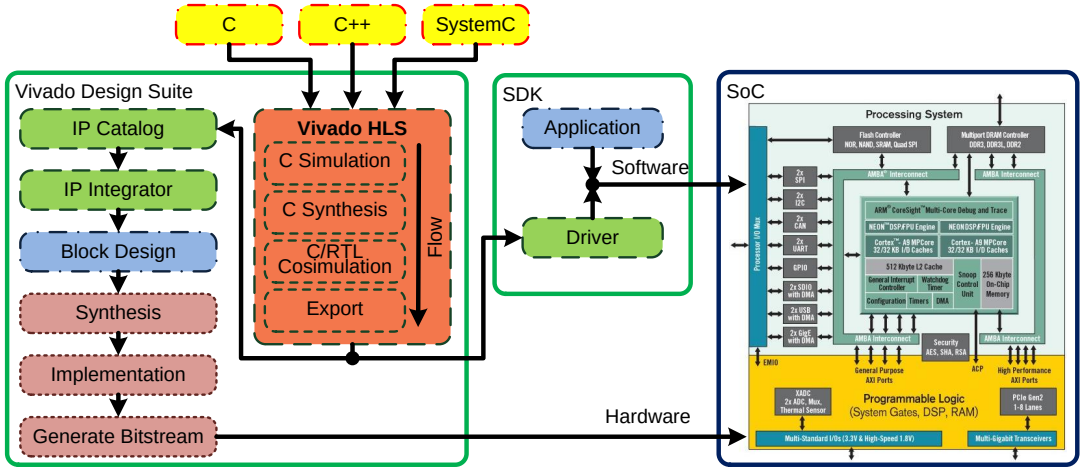
Beispiel eines „All Programmable SoC“: Zynq^{TM}-7000 von Xilinx



Quelle: Xilinx

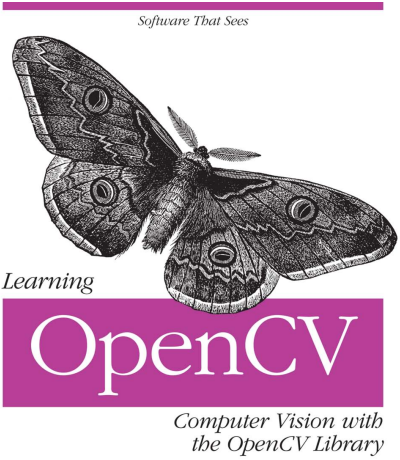
- Processing System:
  ARM Cortex-A9 DualCore mit
  $\sim$ 5 GFLOPS
- Programmable Logic:
  FPGA-Teil mit
  $\sim$ 50 GFLOPS (Z 7045)

Designflow für „All Programmable SoC“ mit High-Level Synthese



Quelle (SoC Grafik): Xilinx

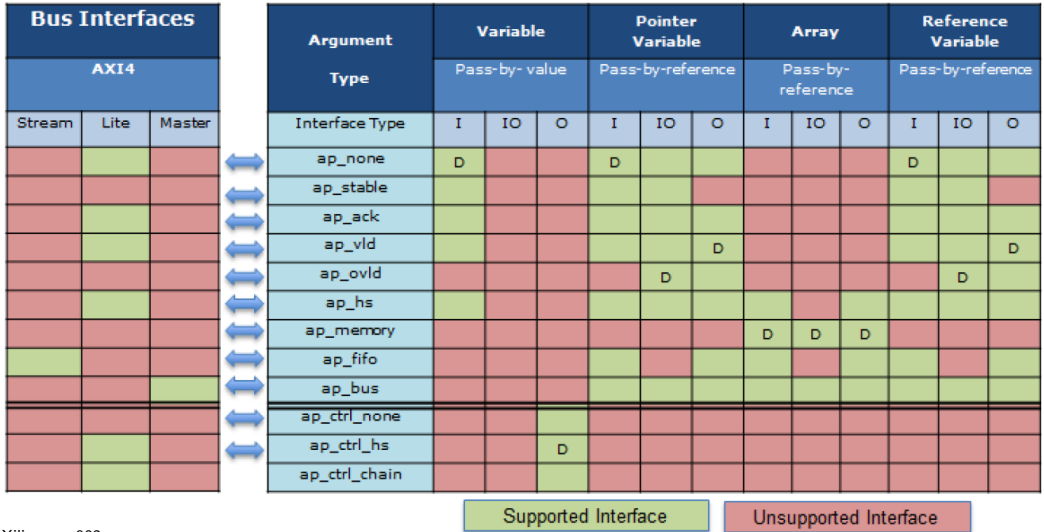
Über OpenCV



Quelle (Grafik): www.docstoc.com

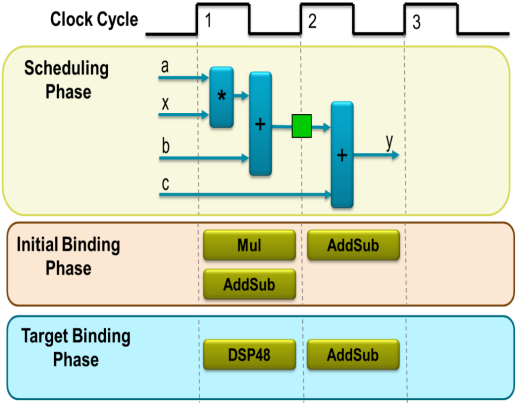
- öffentliche, kostenlose, plattformunabhängige Library
- Computer Vision und maschinelles Lernen
- 2500 optimierte Algorithmen
- immer mehr Funktionen in Vivado HLS Library von Xilinx vorimplementiert

Vorteile der High-Level Synthese - Schnittstellen

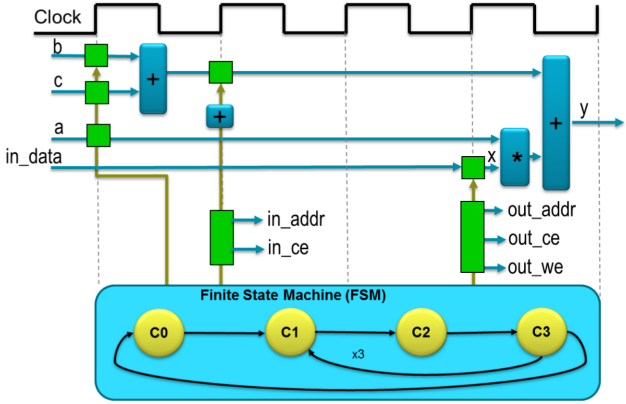


Quelle: Xilinx - ug902

Vorteile der High-Level Synthese - Automatisierung



Quelle: Xilinx - ug902



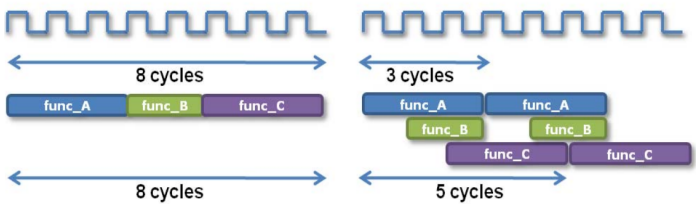
Quelle: Xilinx - ug902

- Zeitplanung
- Anbindung

- Kontrolle von Schreib-/ Lesezyklen
- wählbarer Tradeoff: Performance / Ressourcen

Vorteile der High-Level Synthese - Optimierung auf Datenfluss

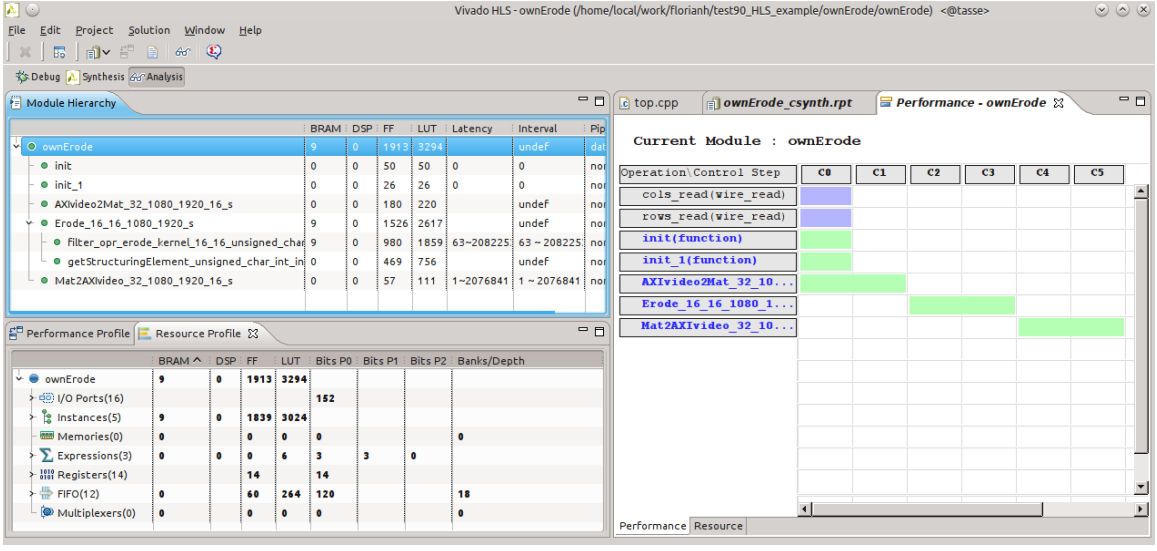
```
void top (a,b,c,d) {
  ...
  func_A(a,b,i1);
  func_B(c,i1,i2);
  func_C(i2,d)

  return d;
}
```



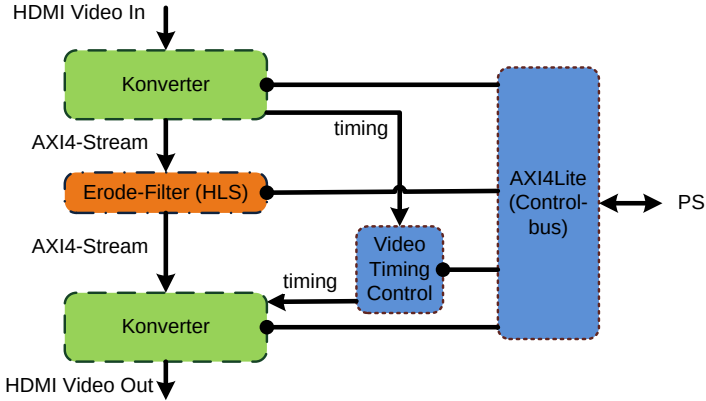
(A) Without Dataflow Pipelining (B) With Dataflow Pipelining

Quelle: Xilinx - ug902

Analyse-Ansicht in Vivado HLS



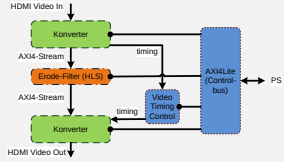
Anbindung für Videoanwendungen



- Konvertierung in AXI4-Stream
- Filter aus OpenCV, implementiert mit High-Level Synthese
- Rück-Konvertierung
- Timing-Überwachung
- AXI4Lite

Beispiel: „Erode“-Filter - Applikation aus OpenCV in Vivado HLS

```
1  #include "top.h"                          // including hls_video.h and type-definitions
2
3  void ownErode(AXI_STREAM& input, AXI_STREAM& output, int rows, int cols) {
4
5  // AXI interface settings
6  #pragma HLS RESOURCE variable=input core=AXIS metadata="-bus_bundle␣INPUT_STREAM"
7  #pragma HLS RESOURCE variable=output core=AXIS metadata="-bus_bundle␣OUTPUT_STREAM"
8  #pragma HLS RESOURCE core=AXI_SLAVE variable=rows metadata="-bus_bundle␣CONTROL_BUS"
9  #pragma HLS RESOURCE core=AXI_SLAVE variable=cols metadata="-bus_bundle␣CONTROL_BUS"
10 #pragma HLS RESOURCE core=AXI_SLAVE variable=return metadata="-bus_bundle␣CONTROL_BUS"
11 #pragma HLS INTERFACE ap_stable port=rows
12 #pragma HLS INTERFACE ap_stable port=cols
13
14 // variables
15     RGB_IMAGE img_in(rows, cols);
16     RGB_IMAGE img_out(rows, cols);
17
18 // dataflow
19 #pragma HLS dataflow
20     hls::AXIvideo2Mat(input, img_in);   // convert in from AXI
21     hls::Erode(img_in, img_out);        // OpenCV function from Xilinx Vivado HLS video-library
22     hls::Mat2AXIvideo(img_out, output); // convert out to AXI
23 }
```



Zusammenfassung

Vorteile der High-Level Synthese:

- Quellcode in C/C++/SystemC
- Generierung von Schnittstellen
- Optimierung auf Architekturebene

Weiteres Beachtenswertes:

- Sensible Reaktion auf Code-Style
- Verifikation

Florian Hagel
Missing Link Electronics GmbH
www.MLEcorp.com
florian.hagel@MLEcorp.com