



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.

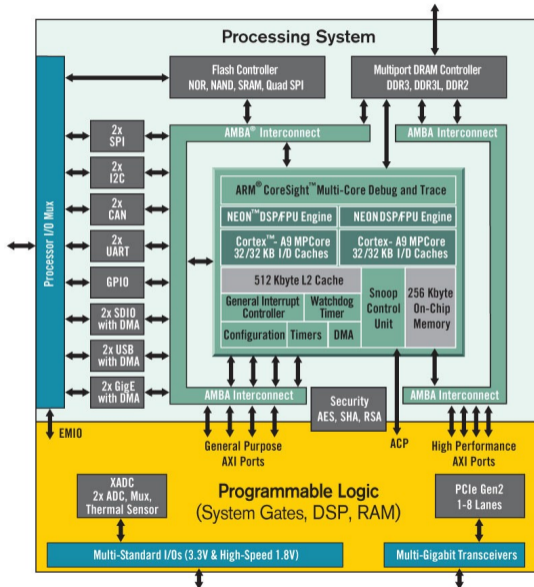
Our Mission is

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.

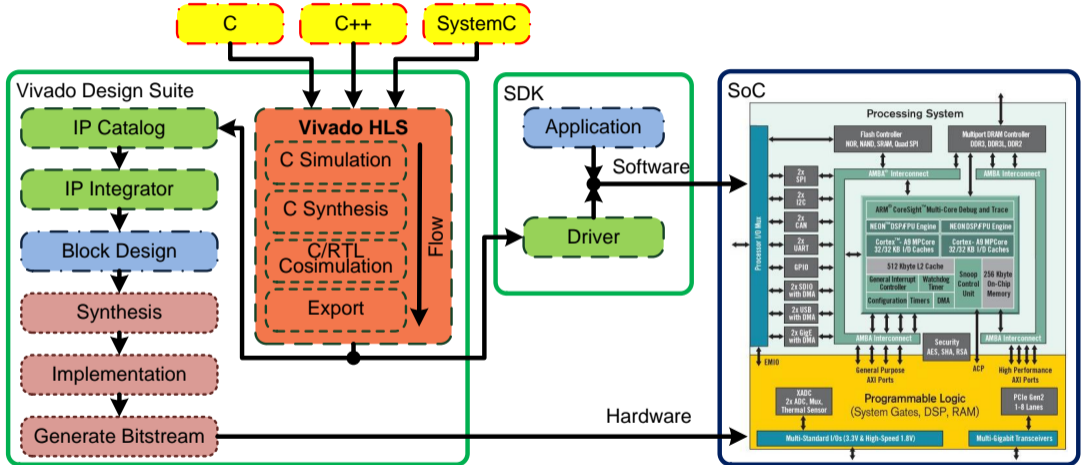
Beispiel eines „All Programmable SoC“: Zynq™-7000 von Xilinx



Quelle: Xilinx

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

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



Quelle (SoC Grafik): Xilinx

Über OpenCV

Software That Sees



Learning

OpenCV

*Computer Vision with
the OpenCV Library*

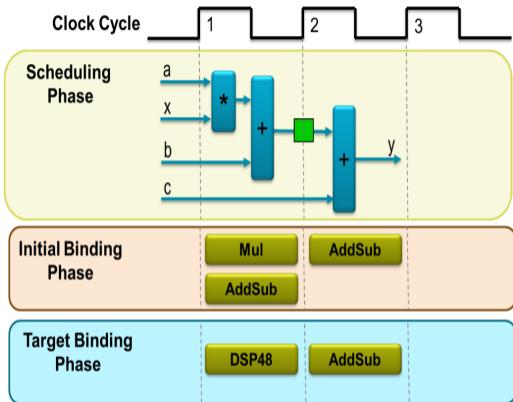
- öffentliche, kostenlose, plattformunabhängige Library
- Computer Vision und maschinelles Lernen
- 2500 optimierte Algorithmen
- immer mehr Funktionen in Vivado HLS Library von Xilinx vorimplementiert

O'REILLY®

Gary Bradski & Adrian Kaehler

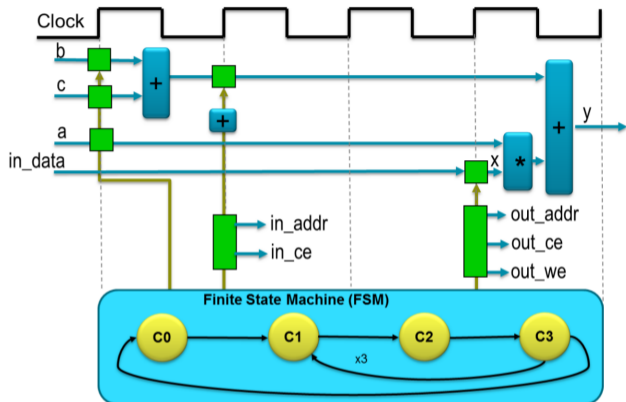
Quelle (Grafik):
www.docstoc.com

Vorteile der High-Level Synthese - Automatisierung



Quelle: Xilinx - ug902

- Zeitplanung
- Anbindung



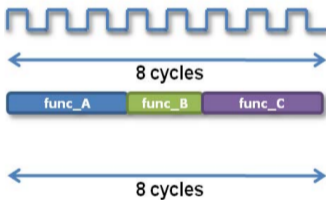
Quelle: Xilinx - ug902

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

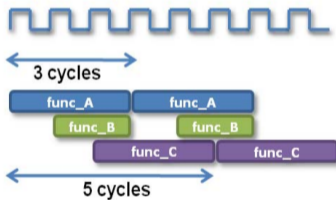
Vorteile der High-Level Synthese - Optimierung auf Datenfluss

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

func_A
func_B
func_C



(A) Without Dataflow Pipelining



(B) With Dataflow Pipelining

Analyse-Ansicht in Vivado HLS

Vivado HLS - ownErode (/home/local/work/florianh/test90_HLS_example/ownErode/ownErode) <@tasse>

File Edit Project Solution Window Help

Debug Synthesis Analysis

Module Hierarchy

	BRAM	DSP	FF	LUT	Latency	Interval	Pip
ownErode	9	0	1913	3294		undef	dat
init	0	0	50	50	0	0	nor
init_1	0	0	26	26	0	0	nor
AXIvideo2Mat_32_1080_1920_16_s	0	0	180	220		undef	nor
Erode_16_16_1080_1920_s	9	0	1526	2617		undef	nor
filter_opr_erode_kernel_16_16_unsigned_char	9	0	980	1859	63~208225	63 ~ 208225	nor
getStructuringElement_unsigned_char_int_in	0	0	469	756		undef	nor
Mat2AXIvideo_32_1080_1920_16_s	0	0	57	111	1~2076841	1 ~ 2076841	nor

Performance Profile

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth
ownErode	9	0	1913	3294				
I/O Ports(16)					152			
Instances(5)	9	0	1839	3024				
Memories(0)	0	0	0	0	0		0	
Expressions(3)	0	0	0	6	3	3	0	
Registers(14)			14		14			
FIFO(12)	0	60	264	120				18
Multiplexers(0)	0	0	0	0				0

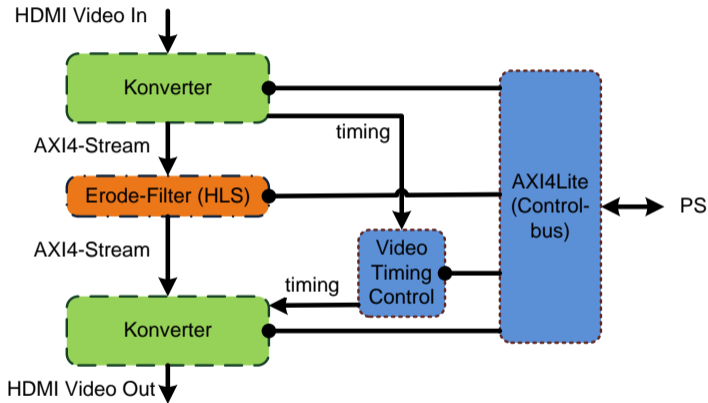
Performance - ownErode

Current Module : ownErode

Operation\Control Step	C0	C1	C2	C3	C4	C5
cols_read(wire_read)						
rows_read(wire_read)						
init(function)						
init_1(function)						
AXIvideo2Mat_32_10...						
Erode_16_16_1080_1...						
Mat2AXIvideo_32_10...						

Performance Resource

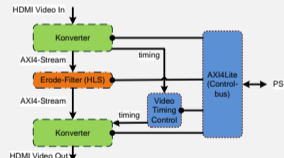
Anbindung für Videoanwendungen



- 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