Advantages of FPGA Based Robot Control Compared to CPU and MCU Based Control Methods

Nicolas Frick

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics

Technical Aspects of Multimodal Systems

January 13, 2020
1. Motivation

2. Basics

3. Paper

   Fast Real-Time LIDAR Processing on FPGAs
   Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA

4. Conclusion

5. Appendix
Robot control is dominated by CPUs\(^1\) and MCUs\(^2\)

A CPU offers high abstraction levels but lose performance

Field Programmable Gate Array (FPGA) technology improves

In special applications they outperform CPUs

High performance computing by concurrent hardware

\(^1\) Central Processing Unit

\(^2\) Microcontroller Unit
Goal: speeding up processing time

Idea: intelligent behaviour can be determined by reactivity

... fast reaction results in more intelligent behaviour

Example: time constraints in collision avoidance

Figure: [1]
Video

DLR Crash Report [4]
Motivation (cont.)

- **Paper**: 'Fast Real-Time LIDAR Processing on FPGAs’ [12] by Shih et al.
- Speed up airborne LIDAR processing by multi-level parallelism
- **Published**: ERSA 2008 May 2014 (2008)
- **Paper**: 'Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA’ [9] by Lyu, Bai and Huang
- Convolutional Neural Networks (CNNs) on FPGAs
- **Published**: IEEE International Symposium on Circuits and Systems 2018-May (2018)
Integrated Circuits (ICs) with reconfigurable components

Basic elements: memory cells, logical gates and flip flops

Peripheral components: dedicated memory blocks, clock generators, Digital Signal Processing (DSP) blocks ...

Core functionality: Configurable Logic Blocks (CLBs) and connection blocks

Figure: FPGA architecture [10]
FPGA (cont.)

- Programming i.e. configuration by special software (IDEs$^3$)
- Mapping of electronic circuit descriptions to CLBs
- Setting of a CLB by Look Up Tables (LUT)
- Efficient routing between components necessary (setting of connection blocks)

Figure: FPGA simplified architecture [2]

$^3$Integrated Development Environment
Programming for a computer: writing instructions for a CPU
Sequential execution of the program
Programming for an FPGA: writing a hardware description
Use of Hardware Description Language (HDL)
Hardware description is translated into configuration data
Effectively creating circuits in hardware (concurrent)
library ieee;
use ieee.std_logic_1164.all;

entity and_logic is
port
  (  
in_1 : in std_logic;
in_2 : in std_logic;
out_and : out std_logic
)
end and_logic;

architecture impl of and_logic is
begin
  out_and <= in_1 and in_2;
end impl;

(a) VHDL ‘and’ logic [2]

Figure: Logical gate (and) [2]

FPGA section is based on [2]
Introduction

LIDAR coordinates calculation

Hardware implementation

Results
Terrain mapping by Airborne Laser Scanning (ALS)
Provide high resolution position information from a remote distance
Multi-modal system: LIDAR, GPS\textsuperscript{4}, IMU\textsuperscript{5}
Fast onboard processing in time constraint scenario
Difficult to achieve by traditional embedded CPU solutions
Micro-laser altimeter developed by NASA: pulse rate 10kHz, 10×10 detector generates 1×10\textsuperscript{6} return events / second

\textsuperscript{4}Global Positioning System
\textsuperscript{5}Inertial Measurement Unit
Fast Real-Time LIDAR Processing on FPGAs (cont.)

- Multi-level parallelism of FPGA is exploited
- Nearly 14x speedup obtained over software solution
- Different setups are investigated and compared
- Extension of the system is possible (pattern recognition, feature extraction)
- Possible application: autonomous driving where resources are rare and real-time computing is necessary
▶ Major components: pulsed laser, scanner and optics, receiver and receiver electronics, position and navigation systems
▶ Receiver registers laser photons reflected from the terrain
▶ GPS provides better absolute position solution
▶ IMU updates aircraft attitude i.e. the roll, pitch and yaw angles
▶ Data fusion of GPS and IMU improves estimation of trajectory

Figure: LIDAR terrain mapping [12]
Fundamental calculation:

Angles from IMU: roll $\varphi_r$, pitch $\varphi_p$, yaw $\varphi_y$

Position from GPS: $X_{ac}$, $Y_{ac}$, $Z_{ac}$

LIDAR: range $\rho$, angle $\Theta$

Return’s coordinates are obtained by:

1. Determine unit vector for each laser pulse using scan angle $\Theta$

2. Align aircraft fixed vectors to earth fixed GPS coordinates

3. Apply generated rotation matrices to unit vector

4. Scale rotated unit vector by range value $\rho$

5. Translate the obtained range vector to GPS coordinate frame
Resulting formula:

\[
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} = \begin{bmatrix}
\rho \left( C_{\varphi_y} C_{\varphi_r} S_{\Theta} - C_{\varphi_y} S_{\varphi_r} C_{\varphi_p} C_{\Theta} - S_{\varphi_y} S_{\varphi_p} C_{\Theta} \right) + X_{ac} \\
\rho \left( S_{\varphi_y} C_{\varphi_r} S_{\Theta} - S_{\varphi_y} S_{\varphi_r} C_{\varphi_p} C_{\Theta} - C_{\varphi_y} S_{\varphi_p} C_{\Theta} \right) + Y_{ac} \\
\rho \left( -S_{\varphi_r} S_{\Theta} - C_{\varphi_r} C_{\varphi_p} C_{\Theta} \right) + Z_{ac}
\end{bmatrix}
\]

where \(C\) and \(S\) abbreviate cosine and sine operations.
Fast Real-Time LIDAR Processing on FPGAs (cont.)

- Different update rate of parameters: multi rate
- Laser returns are independent of one another
- Parallel processing by buffering in FPGA
- One buffer captures 33,000 laser returns and angles plus IMU angles and GPS position

Figure: Buffering of LIDAR input [12]
Host-PC captures data from LIDAR

Data transferred to FPGA from Host using Direct Memory Access (DMA)

Pipelining applies to data input

LIDAR processing core on FPGA computes coordinates

State machine governs data flow

Parallel computation of (X,Y,Z) and angular values $\varphi$
Fast Real-Time LIDAR Processing on FPGAs (cont.)

Figure: Dataflow of onboard LIDAR processing [12]
Fast Real-Time LIDAR Processing on FPGAs (cont.)

- Xilinx Virtex2 Pro 50 FPGA with clock frequency 125MHz
- Processes 1s of data in below 1ms
- Cray XD1 (super-) computer with 6x two 2.4GHz AMD Opteron processors, only one node used for LIDAR
- Software baseline computed from a C application executed on a 2.4 GHz AMD Opteron processor

![Table III](image)

**Table III**

<table>
<thead>
<tr>
<th>Description</th>
<th>Design 1</th>
<th>Design 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HLL (%)</td>
<td>HDL (%)</td>
</tr>
<tr>
<td>Slices</td>
<td>38</td>
<td>31</td>
</tr>
<tr>
<td>MULT18x18s</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Actual Speedup</td>
<td>9.9</td>
<td>10.2</td>
</tr>
</tbody>
</table>

**Figure:** [12]

5 The previous section is based on [12]
Introduction

Convolutional neural network design

Hardware implementation

Results
Convolutional Neural Network based road segmentation algorithm (semantic segmentation)

- Provide drivable region area
- Real time LiDAR processing on FPGA in 16.9 ms each scan
- Obtain 3D geometry information of vehicle surroundings with very high accuracy
- Quality of road markings and light conditions less important

**Figure:** Camera view and LiDAR points [9]
Network: cascading blocks that contain a convolutional layer and non-linear layer

Multiplexing is applied on the processing blocks on the chip

Goal: label the drivable region (free space)

Input: LIDAR, GPS, IMU

Pre-processing, neural network processing and post-processing
Preprocessing: arrange data points and project into a 3D blob with MxN tensors and C channels

Input blob: 64 scan rows x 256 columns (polar angles) x 16 feature channels

Figure: Input map to NN [9]
Neural network processing by new network architecture

Minimize memory by multiplexing blob memory

Hidden layers use same structure

All internal results can be stored in same memory space directory

No allocation or reshaping of the blob

**Figure:** CNN architecture [9]
Post processing: NN output is projected back to targeted views (camera and top view)

Challenge: non-uniformly distributed points in targeted view after projection

Determine contour by projecting furthest points in each angle \( \Theta \) (each column of output) onto target view

Draw a polyline along those points on all angles of target view

Add a straight line to the bottom and the polyline becomes a polygon

Polygon is treated as contour of drivable area (segmentation result)
Figure: Drivable area on camera view and top view [9]
Memory usage: 64 memories x 256k bits for intermediate feature maps

3D convolution is broken into 64 parallel 2D convolutions

each with two filters, followed by adder tree to generate feature map

**Figure:** Hardware architecture of convolutional layer [8]
Loop based control because of large RAM consumption of feature maps

Finite state machine (FMS) is used to generate 64 feature maps in 32 loops reusing block RAM

Another FSM controls the first one for a full completion of 11 layers

**Figure:** Block diagram dataflow [8]
Xilinx UltraScale XCKU115 FPGA at 350MHz
Each 2D convolution takes about 18,000 clock cycles
Results in 16.9 ms processing time for each scan
LIDAR normally scans at 10Hz
Real time processing requirement fulfilled and factor 30 speedup
Intel Xeon CPU E5-2687Wv3 processing time takes 500ms for same task
Another own evaluation on K20 GPU results in 120ms run time
Training on KITTI road/lane detection dataset [3]

Optimal performance ($F_{\text{max}}$) and average precision (AP)

Result: less processing time at comparable performance including pre-processing, neural network, post-processing, and visualization

<table>
<thead>
<tr>
<th>Name</th>
<th>$F_{\text{max}}$</th>
<th>AP</th>
<th>run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work on FPGA</td>
<td>91.79%</td>
<td>84.76%</td>
<td>16.9ms</td>
</tr>
<tr>
<td>HybridCRF [25]</td>
<td>90.81%</td>
<td>84.79%</td>
<td>1500ms</td>
</tr>
<tr>
<td>LidarHisto [35]</td>
<td>90.67%</td>
<td>84.79%</td>
<td>100ms</td>
</tr>
<tr>
<td>MixedCRF</td>
<td>90.59%</td>
<td>84.24%</td>
<td>6000ms</td>
</tr>
<tr>
<td>FusedCRF [24]</td>
<td>88.25%</td>
<td>79.24%</td>
<td>2000ms</td>
</tr>
<tr>
<td>RES3D-Velo [36]</td>
<td>86.58%</td>
<td>78.34%</td>
<td>360ms</td>
</tr>
</tbody>
</table>

Figure: Comparison with KITTI road/lane detection dataset [9]

5The previous section is based on [9]
FPGA indeed (can) increase processing speed

Its high adaptivity, parallelism and efficiency brings advantages over CPUs/MCUs especially on autonomous robot applications

Other applications:
- Multi-axis motion controller for robotic applications
- Decentralized inverse optimal neural control
- ...

FPGAs can be very expensive per piece

Low abstraction level and not easy to program

High Level Synthesis (use of High Level Languages) produces overhead and cost performance


Thanks for paying attention!

Questions?
Figure: Ford dataset [8]
**Figure**: Kitti dataset [8]
CNN repetitive structure:

- 12x convolutional layer and activation layer
- Conv. layer: 64 filters with each 5x5 kernel stride size 1 and padding size 2
- Stride and padding make output size equal to input size
- Relu activation function for fast training
- Two drop out layers after 6th and 10th block in training to accelerate convergence
- No pooling layers
- 11 conv. layers and 5x5 kernel because of resource and performance tradeoff
Zero padding is applied to control size of feature maps and reserve boundary information of input images.

Dual RAM port is designed for next stage convolution.

Padded zeros are stored in advance.

Control logic store each pixel in proper memory location.

Scanning circuit reads pixel by pixel.

HDL-64E LiDAR is used in KITTI road benchmark.

64 scan channels and emits 1.3 million points per second.

2D convolution implemented in conjunction with a line buffer.
Output is a 5x5 pixel window for multiplication with weight matrix using 25 multipliers

Highly pipelined adder tree computes the sum

RELU activation implemented by comparator and multiplexer

**Figure:** System diagram [8]
Figure: Line buffer, 4 lines 5 register [9]
**Figure:** Zero padding in RAM [9]
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity binary_adder_tree is
  port
  (
    a : in unsigned (7 downto 0);
    b : in unsigned (7 downto 0);
    c : in unsigned (7 downto 0);
    d : in unsigned (7 downto 0);
    e : in unsigned (7 downto 0);
    clk : in std_logic;
    result : out unsigned (7 downto 0)
  );
end entity;
architecture rtl of binary_adder_tree is
-- Declare registers to hold intermediate sums
signal sum1, sum2, sum3 : unsigned (7 downto 0);

begin
process (clk)
begin
if (rising_edge(clk)) then
-- Generate and store intermediate values in the pipeline
sum1 <= a + b;
sum2 <= c + d;
sum3 <= sum1 + sum2;
-- Generate and store the last value, the result
result <= sum3 + e;
end if;
end process;
end rtl;

Figure: Binary adder tree VHDL [7]
library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
  use IEEE.NUMERIC_STD.ALL;

entity relu is
  Generic (
    WIDTH: natural
  );

  Port (
    din: in std_logic_vector(WIDTH - 1 downto 0);
    dout: out std_logic_vector(WIDTH - 1 downto 0)
  );
end relu;

architecture rtl of relu is
begin
  dout <= (others => '0') when din(WIDTH - 1) = '1' else din;
end rtl;

Figure: Rectified linear unit VHDL [5]
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity relu_wrapper is
  Port (
    din: in std_logic_vector(9 downto 0);
    dout: out std_logic_vector(9 downto 0)
  );
end relu_wrapper;

architecture Structural of relu_wrapper is
begin
  relu_inst : entity WORK.relu
    generic map (  
      WIDTH => 10
    )
    port map (  
      din => din,
      dout => dout
    );
end Structural;
Figure: Relu wrapper VHDL [5]
Figure: Difference of MCU and MPU [14]

- MCU: Single chip computer, single threaded, bare metal interface
CPU: Peripheral chips not integrated, multi threaded, operating system, in this context: main processor of a PC

**Conversion errors:**
Fractional precision of fixed point configurations (angular values) & (position values) and root-mean-squared error (RMSE) and maximum error (Max E) induced by conversion

<table>
<thead>
<tr>
<th>Precision</th>
<th>RMSE (m)</th>
<th>Max E. (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(16, 14) &amp; (16, 5)</td>
<td>0.254</td>
<td>0.962</td>
</tr>
<tr>
<td>(31, 28) &amp; (16, 5)</td>
<td>0.183</td>
<td>0.718</td>
</tr>
</tbody>
</table>

**Figure:** Errors in fixed point configurations [12]
Design 2 of ALS:

▶ Balance communication and computation by migrating interpolation for parameters
▶ Interpolate data and send increments to FPGA
▶ Increase of fixed point bits in total (31,28)
▶ $(\rho, \Theta)$ 64bits/ element and 18*64 bits for GPS/IMU data
▶ $(\varphi_r \varphi_p \varphi_y) \Rightarrow (\Delta \varphi_r \Delta \varphi_p \Delta \varphi_y)$
▶ $(X_{ac}, Y_{ac}, Z_{ac}) \Rightarrow (\Delta X_{ac}, \Delta Y_{ac}, \Delta Z_{ac})$
Figure: State machine with host signals (dashed lines = design 2) [12]
Appendix (cont.)

Figure: Pseudocode design 2 [12]
### Table II

<table>
<thead>
<tr>
<th>Table Title</th>
<th>Design Progression through Rat Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Set Parameters</strong></td>
<td><strong>Design 1</strong> (Nallatech)</td>
</tr>
<tr>
<td>$N_{\text{elem}_{\text{in}}}$ input (elem.)</td>
<td>66000</td>
</tr>
<tr>
<td>$N_{\text{elem}_{\text{out}}}$ output (elem.)</td>
<td>33000</td>
</tr>
<tr>
<td>$N_{\text{byte}_{\text{elem}}}$ (B/elem.)</td>
<td>8</td>
</tr>
<tr>
<td><strong>Communication Parameters</strong></td>
<td><strong>Design 1</strong> (Nallatech)</td>
</tr>
<tr>
<td>Throughput ideal (MB/s)</td>
<td>1000</td>
</tr>
<tr>
<td>$\alpha_{\text{read}}$</td>
<td>0.25</td>
</tr>
<tr>
<td>$\alpha_{\text{write}}$</td>
<td>0.25</td>
</tr>
<tr>
<td><strong>Computation Parameters</strong></td>
<td><strong>Design 1</strong> (Nallatech)</td>
</tr>
<tr>
<td>$N_{\text{elem}_{\text{proc}}}$ (elem.)</td>
<td>33000</td>
</tr>
<tr>
<td>$N_{\text{ops}_{\text{elem}}}$ (ops/elem.)</td>
<td>9</td>
</tr>
<tr>
<td>Throughput ideal (ops/cycle)</td>
<td>9</td>
</tr>
<tr>
<td>$f_{\text{clock}}$ (MHz)</td>
<td>125</td>
</tr>
<tr>
<td><strong>Software Parameters</strong></td>
<td><strong>Design 1</strong> (Nallatech)</td>
</tr>
<tr>
<td>$t_{\text{soft}}$ (sec)</td>
<td>1.09E-02</td>
</tr>
<tr>
<td>$N_{\text{iter}}$ (iter.)</td>
<td>1</td>
</tr>
<tr>
<td><strong>Calculated Metrics</strong></td>
<td><strong>Design 1</strong> (Nallatech)</td>
</tr>
<tr>
<td>$t_{\text{comm}}$ (sec)</td>
<td>3.16E-03</td>
</tr>
<tr>
<td>$t_{\text{comp}}$ (sec)</td>
<td>2.64E-04</td>
</tr>
<tr>
<td>$t_{\text{RC}}$ (sec)</td>
<td>3.43E-03</td>
</tr>
<tr>
<td>Predicted Speedup</td>
<td>3.2</td>
</tr>
</tbody>
</table>

**Figure:** FPGA processing time: below 1ms [12]
Video

DLR Crash Report [4]
Figure: Circuit and symbol of D-FlipFlop [1]
Figure: Shift register by FFs in series [2]
Will LIDAR become cheaper?

"In the future, the automotive supplier Bosch will also rely on so-called laser radar in the development of automated driving, and is entering into the development of such sensors. The goal is to make the technology suitable for mass production and thus significantly cheaper than before, Bosch announced on Thursday."[11]

"Only the parallel use of three sensor principles makes automated driving as safe as possible, the company argues."[11]