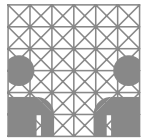
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN Faculty
Department of Informatics



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

Nicolas Frick

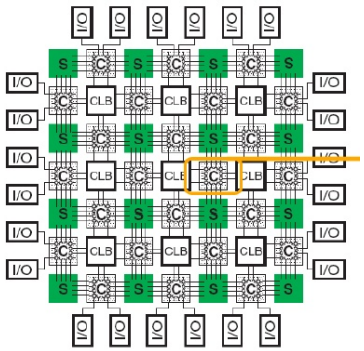
University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
Technical Aspects of Multimodal Systems

January 13, 2020

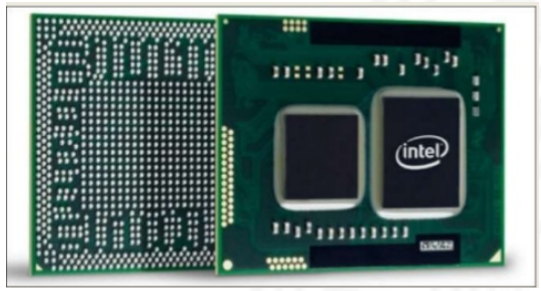
Outline

Motivation

- 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

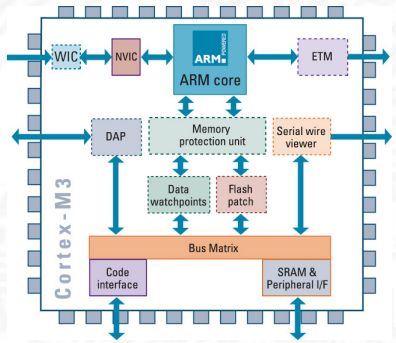


(a) FPGA architecture [10]



(b)

x86 processor [6]



(c)

ARM Cortex MCU [13]

[1] Central Processing Unit
[2] Microcontroller Unit

Motivation (cont.)

- 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)

FPGA

- 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

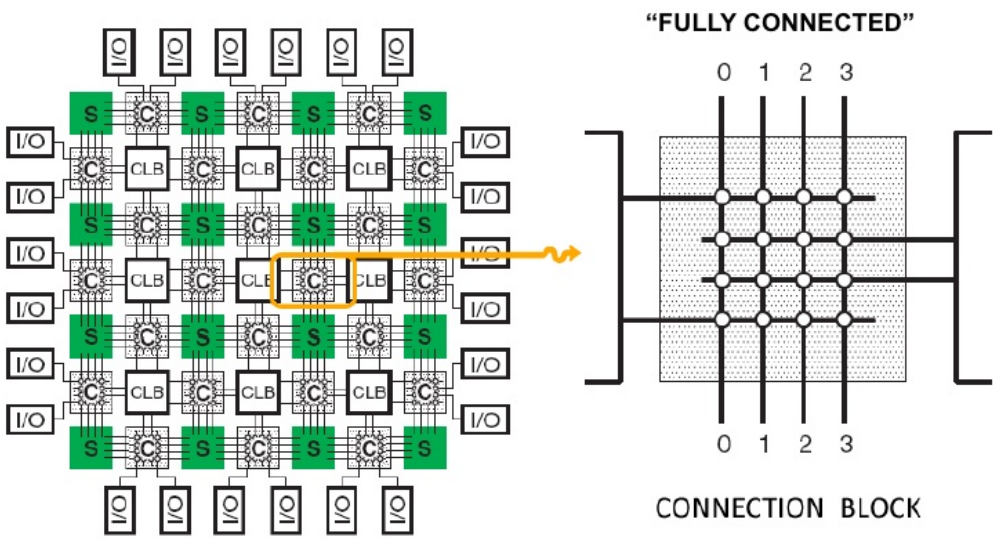


Figure: FPGA architecture [10]

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)

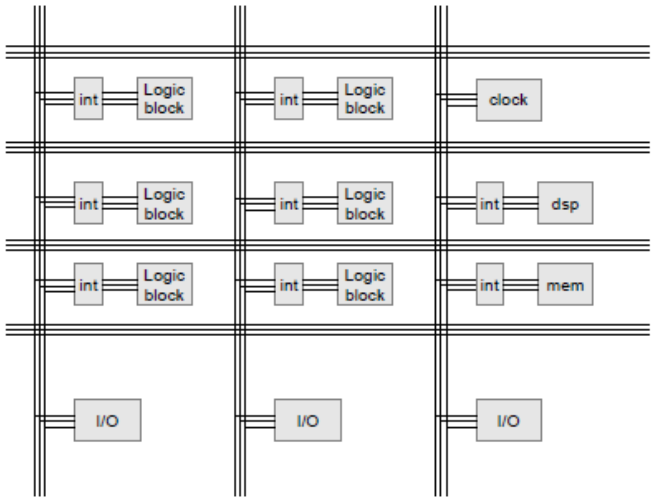


Figure: FPGA simplified architecture [2]

[3]Integrated Development Environment

FPGA (cont.)

- 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)

FPGA (cont.)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity and_logic is
5   port
6       (
7            in_1 : in std_logic;
8            in_2 : in std_logic;
9            out_and : out std_logic
10       );
11   end and_logic;
```

```
12  architecture impl of and_logic is
13  begin
14     out_and <= in_1 and in_2;
15  end impl;
```

(a) VHDL 'and' logic [2]

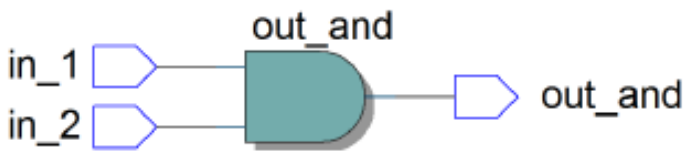


Figure: Logical gate (and) [2]

[3] FPGA section is based on [2]

Fast Real-Time LIDAR Processing on FPGAs

Introduction

LIDAR coordinates calculation

Hardware implementation

Results

Fast Real-Time LIDAR Processing on FPGAs

- Terrain mapping by Airborne Laser Scanning (ALS)
- Provide high resolution position information from a remote distance
- Multi-modal system: LIDAR, GPS[4], IMU[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, 10x10 detector generates $1 * 10^6$ return events / second

[4] Global Positioning System
[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

Fast Real-Time LIDAR Processing on FPGAs (cont.)

- 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

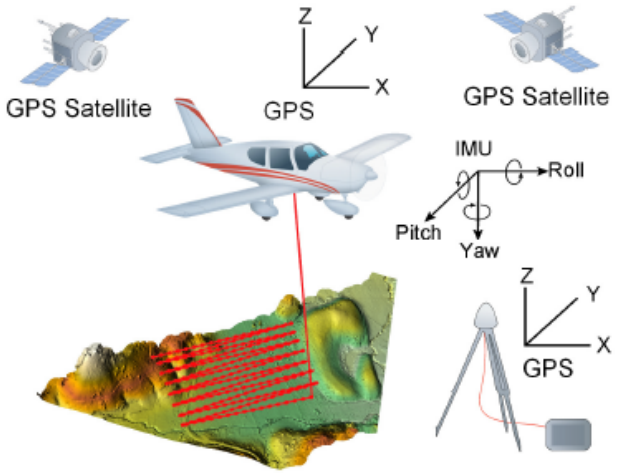


Figure: LIDAR terrain mapping [12]

Fast Real-Time LIDAR Processing on FPGAs (cont.)

- Fundamental calculation:

Angles from IMU: roll φ_r, pitch φ_p, yaw φ_y
Position from GPS: X_{ac}, Y_{ac}, Z_{ac}
LIDAR: range ρ, angle Θ

Return's coordinates are obtained by:

1. Determine unit vector for each laser pulse using scan angle Θ
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 ρ
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

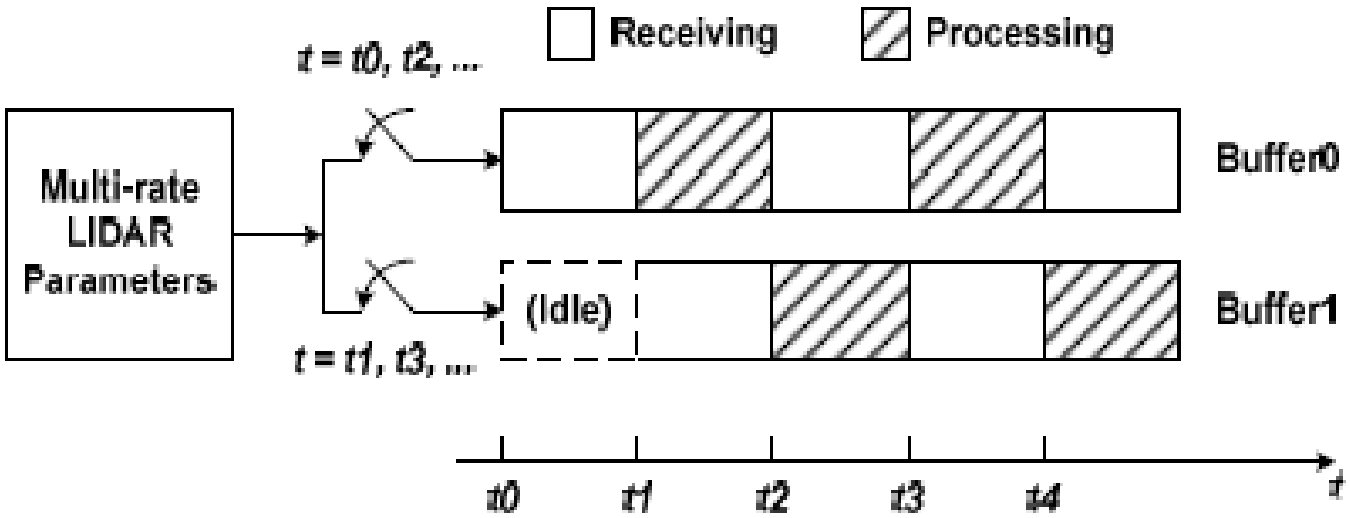


Figure: Buffering of LIDAR input [12]

Fast Real-Time LIDAR Processing on FPGAs (cont.)

- Host-PC captures data from LIDAR
- Data transferred to FPGA from Host using Direct Memory Accesss (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 φ

Fast Real-Time LIDAR Processing on FPGAs (cont.)

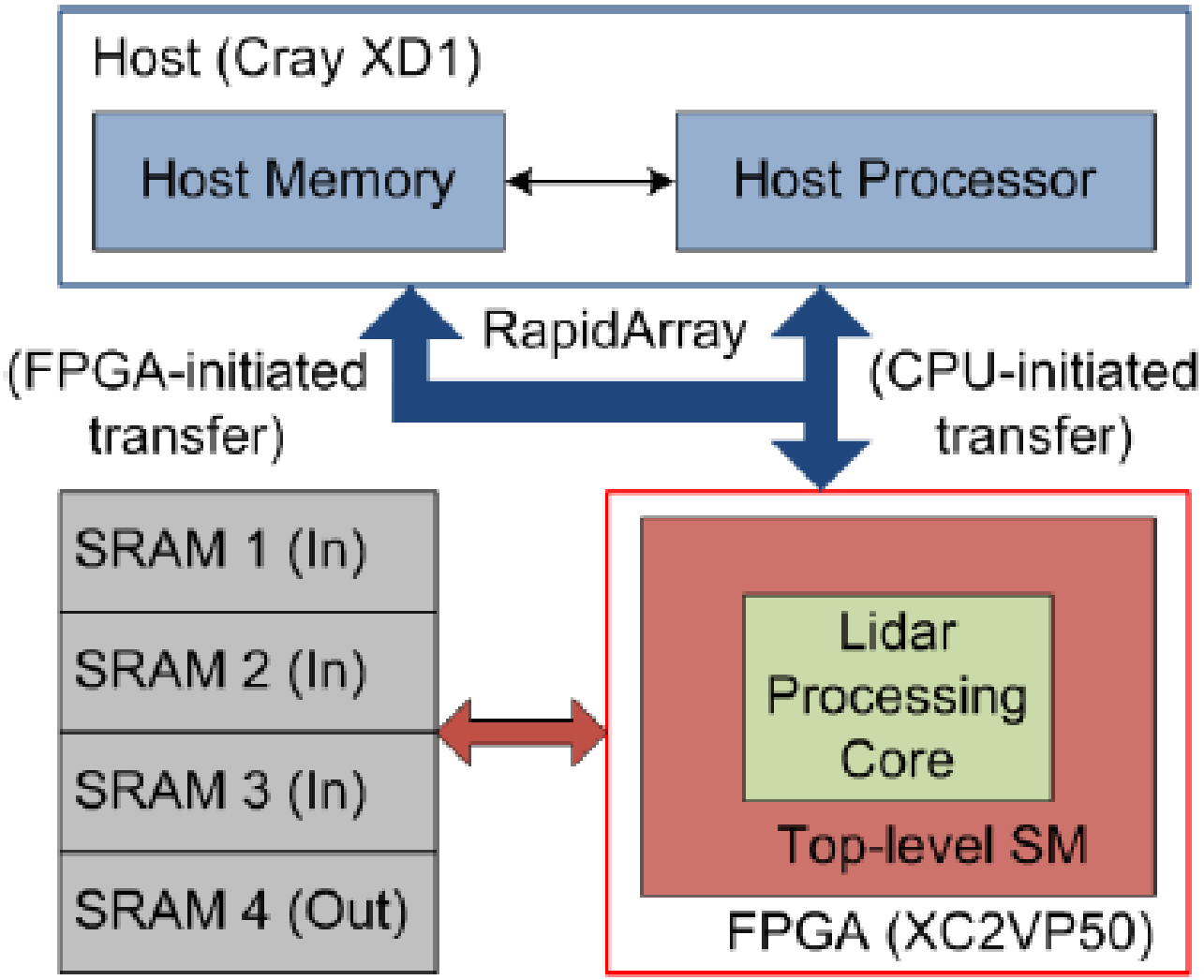


Figure: Dataflow of onboard LIDAR processing [12]

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
SPEEDUP & DEVICE UTILIZATION OF DESIGNS 1 AND 2 ON CRAY XD1

Description		Design 1		Design 2	
		HLL	HDL	HLL	HDL
Slices	(%)	38	31	45	42
MULT18x18s	(%)	5	5	5	5
Actual Speedup		**9.9**	**10.2**	**13.1**	**13.8**

Figure: [12]

[5] The previous section is based on [12]

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

Introduction

Convolutional neural network design

Hardware implementation

Results

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

- 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

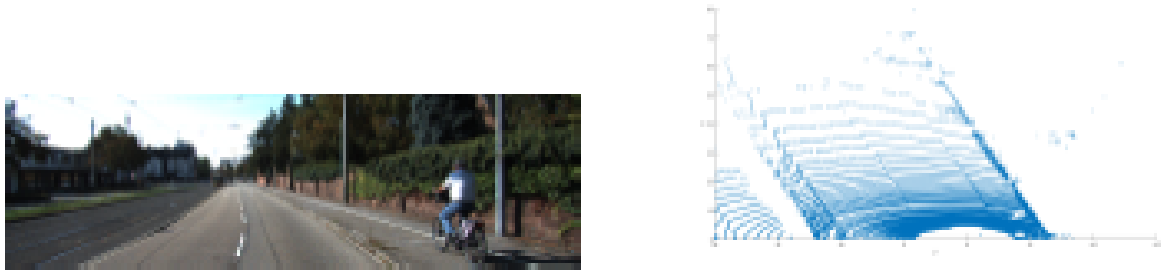


Figure: Camera view and LIDAR points [9]

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

- 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

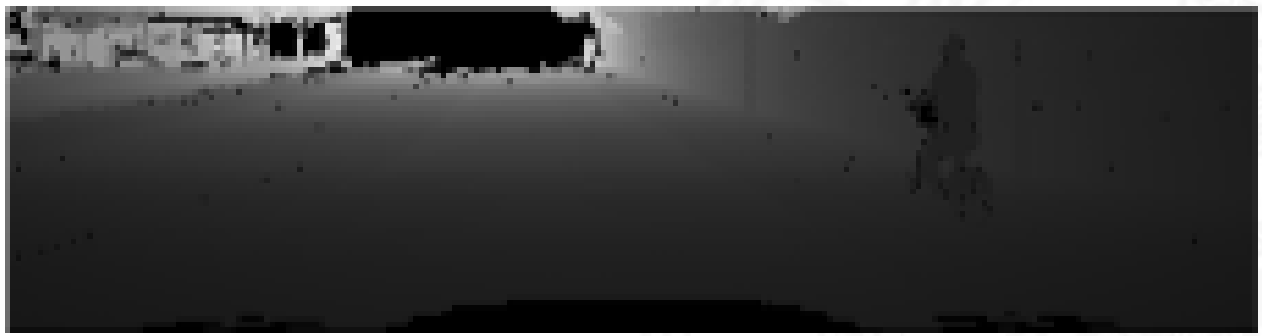


Figure: Input channel to NN [9]

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

- 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

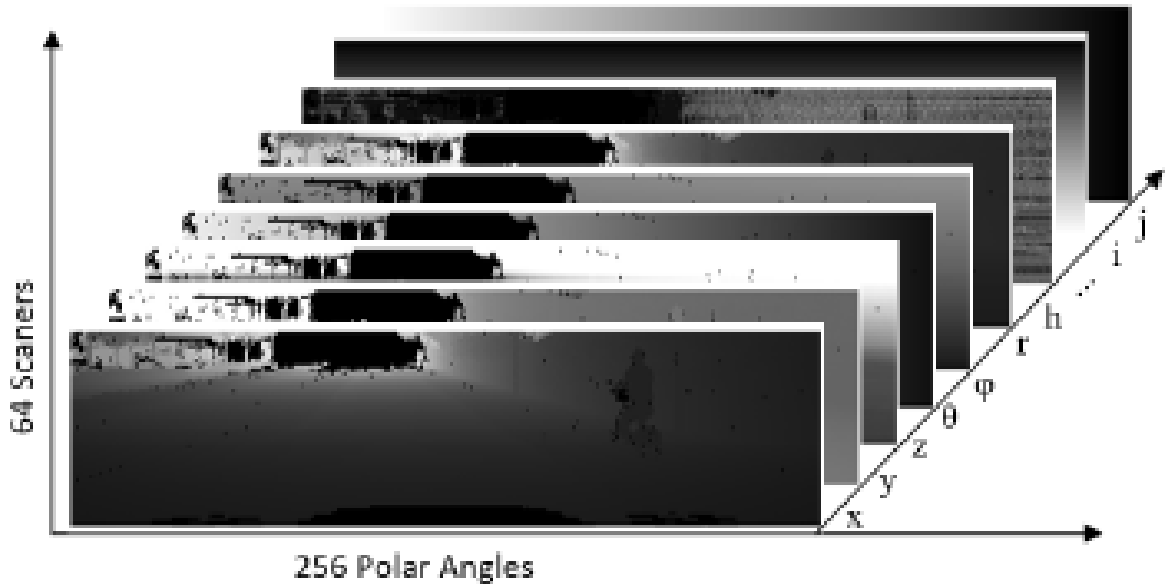


Figure: Input map to NN [9]

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

- 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

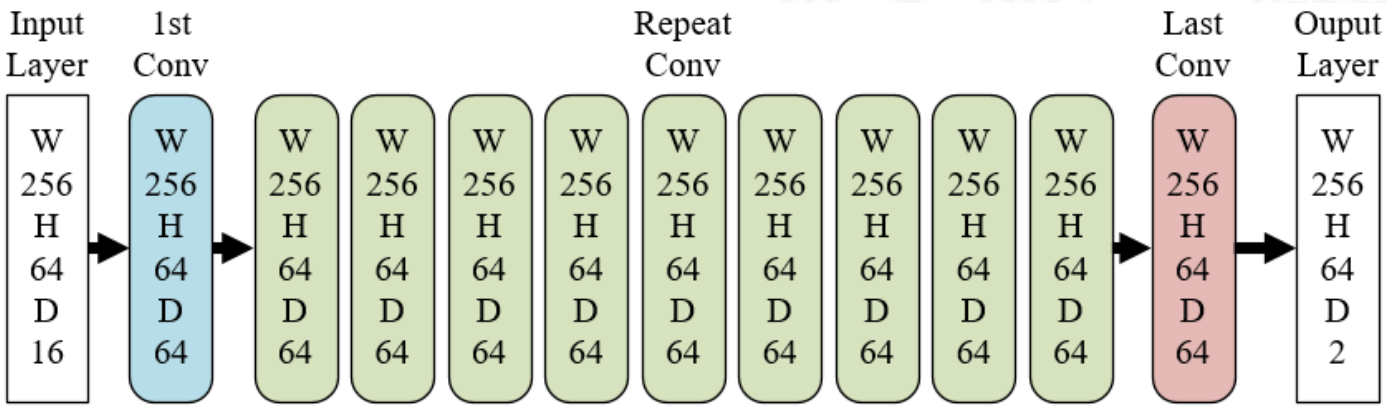


Figure: CNN architecture [9]

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

- 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 Θ (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)

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

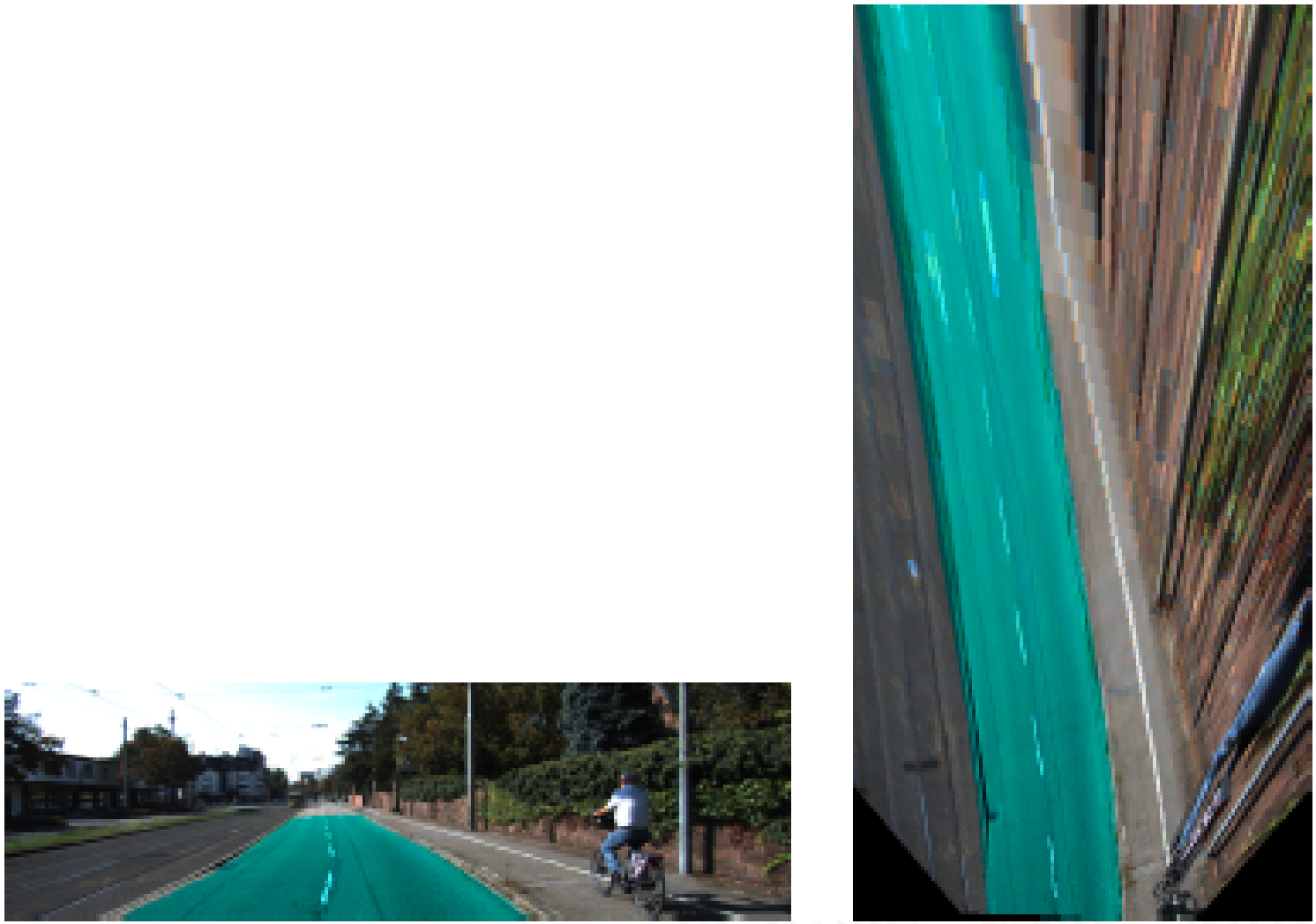


Figure: Drivable area on camera view and top view [9]

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

- 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

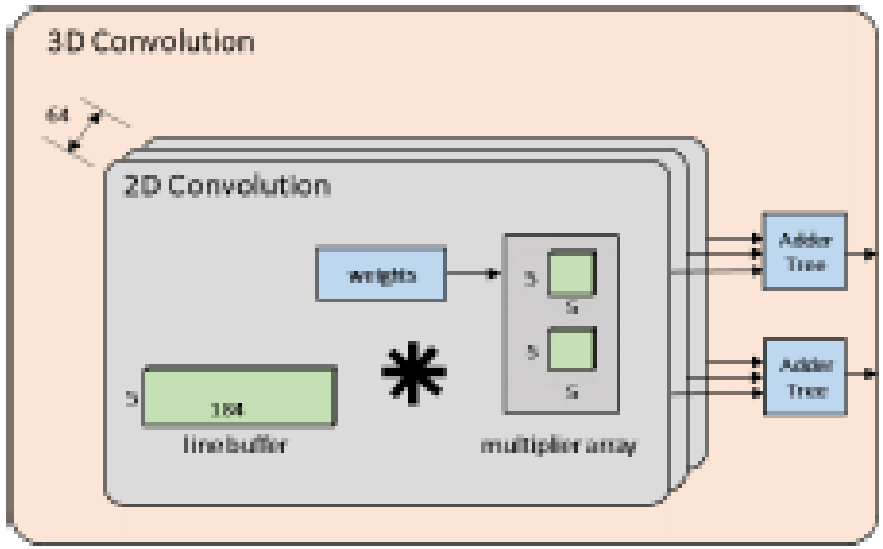


Figure: Hardware architecture of convolutional layer [8]

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

- 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

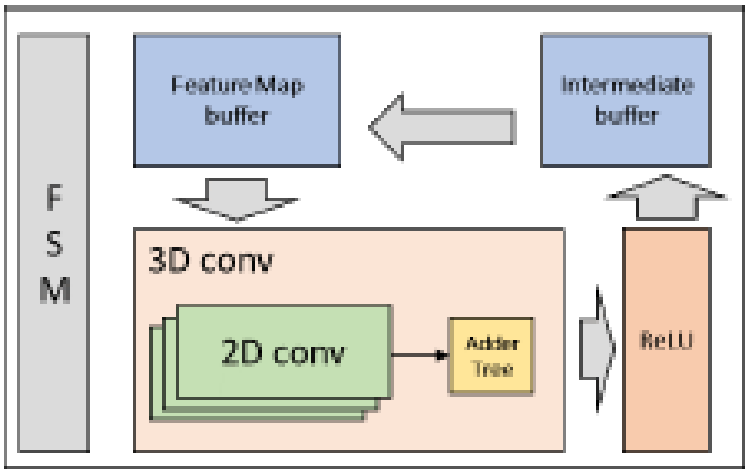


Figure: Block diagram dataflow [8]

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

- 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

Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA (cont.)

- Training on KITTI road/lane detection dataset [3]
- Optimal performance (F_{max}) and average precision (AP)
- Result: less processing time at comparable performance including pre-processing, neural network, post-processing, and visualization

Name	F_{max}	AP	run time
This work on FPGA	91.79%	84.76%	16.9ms
HybridCRF [25]	90.81%	84.79%	1500ms
LidarHisto [35]	90.67%	84.79%	100ms
MixedCRF	90.59%	84.24%	6000ms
FusedCRF [24]	88.25%	79.24%	2000ms
RES3D-Velo [36]	86.58%	78.34%	360ms

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

[5] The previous section is based on [9]

Conclusion

- 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

References

[1] Electronics Tutorials. "D-type Flip Flop Counter or Delay Flip-flop". In: (2018), pp. 1–15. URL: https://www.electronics-tutorials.ws/de/sequentielle/d-flipflop.html.

[2] Cord Elias. *FPGAs für Maker*. Heidelberg: dpunkt.verlag, 2017. ISBN: 978-3-96088-030-1.

[3] Andreas Geiger et al. "The KITTI Vision Benchmark Suite". In: *The KITTI Vision Benchmark Suite* (2013), pp. 1–13. URL: http://www.cvlibs.net/datasets/kitti/eval_road.php%20http://www.cvlibs.net/datasets/kitti/eval_object.php%0Ahttp://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo.

References (cont.)

[4] Sami Haddadin et al. *Human-Robot Collision Study - YouTube*. URL: https://www.youtube.com/watch?v=R5Gx8jpwyQ0.

[5] Jakub Hrabovsky. *jhrabovsky/cnn-fpga-rtl: The CNN architecture elements implemented with RTL approach in VHDL*. URL: https://github.com/jhrabovsky/cnn-fpga-rtl.

[6] IGZ Software house. *X86 Architecture*. 2016. URL: https://www.slideshare.net/tousifirshad/x86-architecture%20https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture.

[7] Intel. *VHDL: Binary Adder Tree*. URL: https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/vhdl/vhd-binary-adder-tree.html.

References (cont.)

[8] Yecheng Lyu, Lin Bai, and Xinming Huang. "ChipNet: Real-Time LiDAR Processing for Drivable Region Segmentation on an FPGA". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.5 (2019), pp. 1769–1779. ISSN: 15498328. DOI: 10.1109/TCSI.2018.2881162.

[9] Yecheng Lyu, Lin Bai, and Xinming Huang. "Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA". In: *Proceedings - IEEE International Symposium on Circuits and Systems* 2018-May (2018). ISSN: 02714310. DOI: 10.1109/ISCAS.2018.8351244.

[10] omutukuda. *Plugin_Architecture_Presentation*. URL: https://www.slideshare.net/omutukuda/presentation-1993175.

References (cont.)

[11] rei/dpa. *Bosch startet Laserradar-Entwicklung für autonomes Fahren - manager magazin*. 2020. URL: https://www.manager-magazin.de/unternehmen/autoindustrie/bosch-startet-laserradar-entwicklung-fuer-autonomes-fahren-a-1303361.html.

[12] K. Shih et al. "Fast real-time LIDAR processing on FPGAs". In: *Proceedings of the 2008 International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA 2008* May 2014 (2008), pp. 231–237.

[13] The Kairos Initiative. *KairosFocus: Capacity Focus, 51: The Raspberry Pi in action, with side notes on Python and the ARM microprocessor architecture*. URL: http://kairosfocus.blogspot.com/2012/06/capacity-focus-51-raspberry-pi-in.html.

References (cont.)

[14] Scott Thornton. *Microcontrollers vs. Microprocessors: What's the difference?* 2017. URL: https://www.microcontrollertips.com/microcontrollers-vs-microprocessors-whats-difference/%20http://blmrgnn.blogspot.com/2018/04/microcontrollers-vs.html.

Thanks for paying attention!

Questions?

Appendix

Figure: Ford dataset [8]

Appendix (cont.)

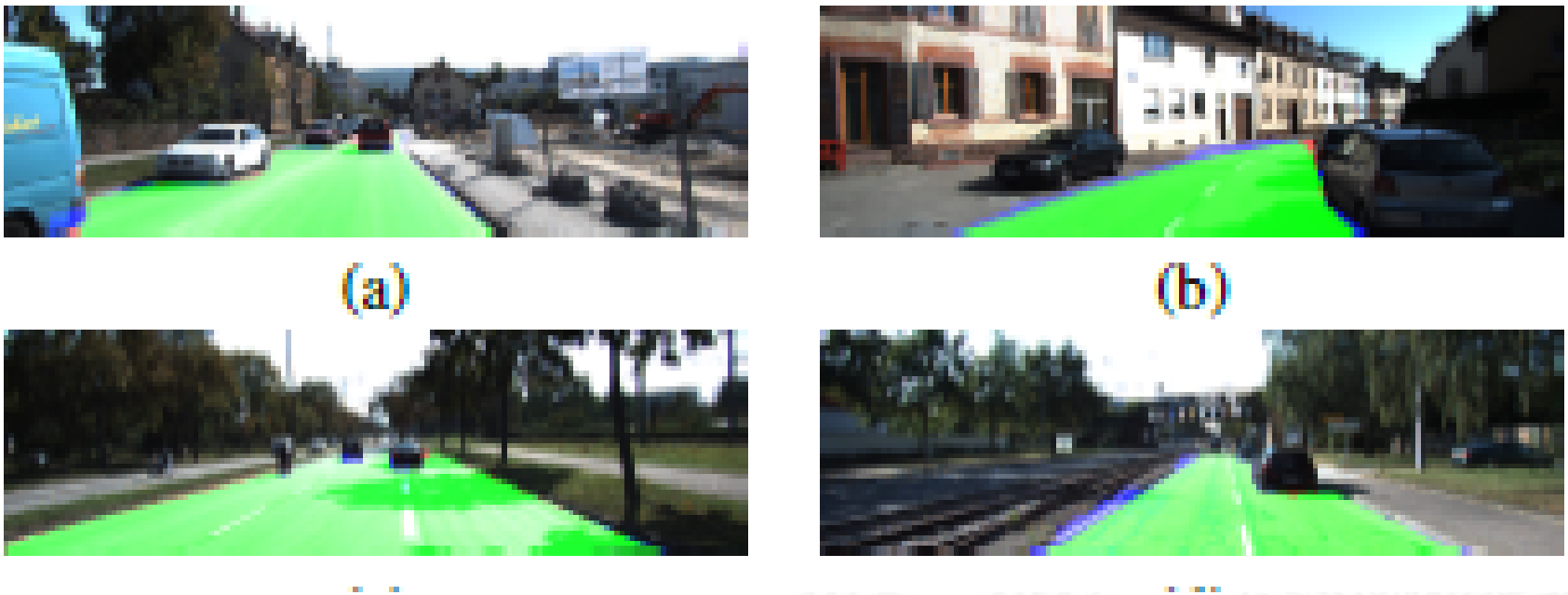


Figure: Kitti dataset [8]

Appendix (cont.)

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

Appendix (cont.)

- 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

Appendix (cont.)

- 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

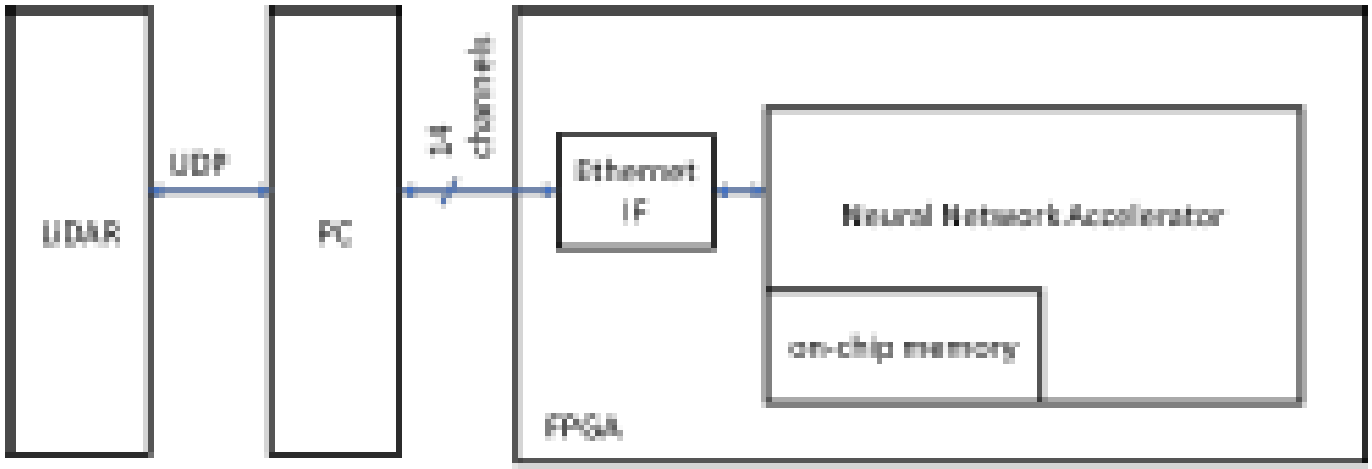


Figure: System diagram [8]

Appendix (cont.)

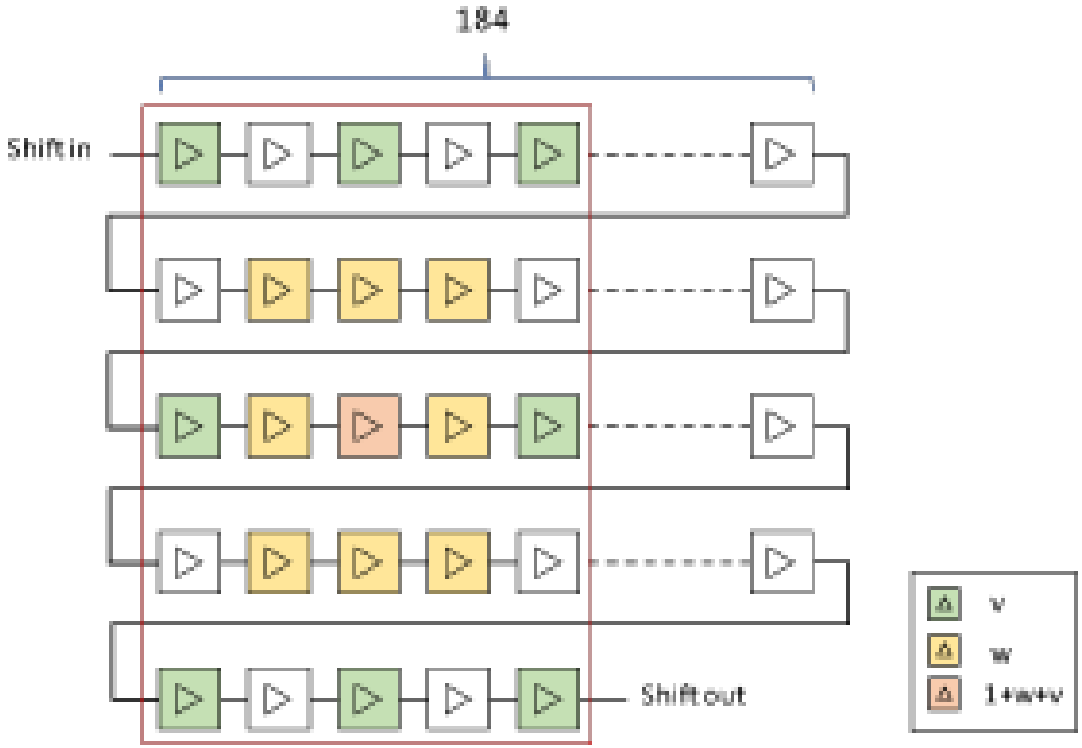


Figure: Line buffer, 4 lines 5 register [9]

Appendix (cont.)

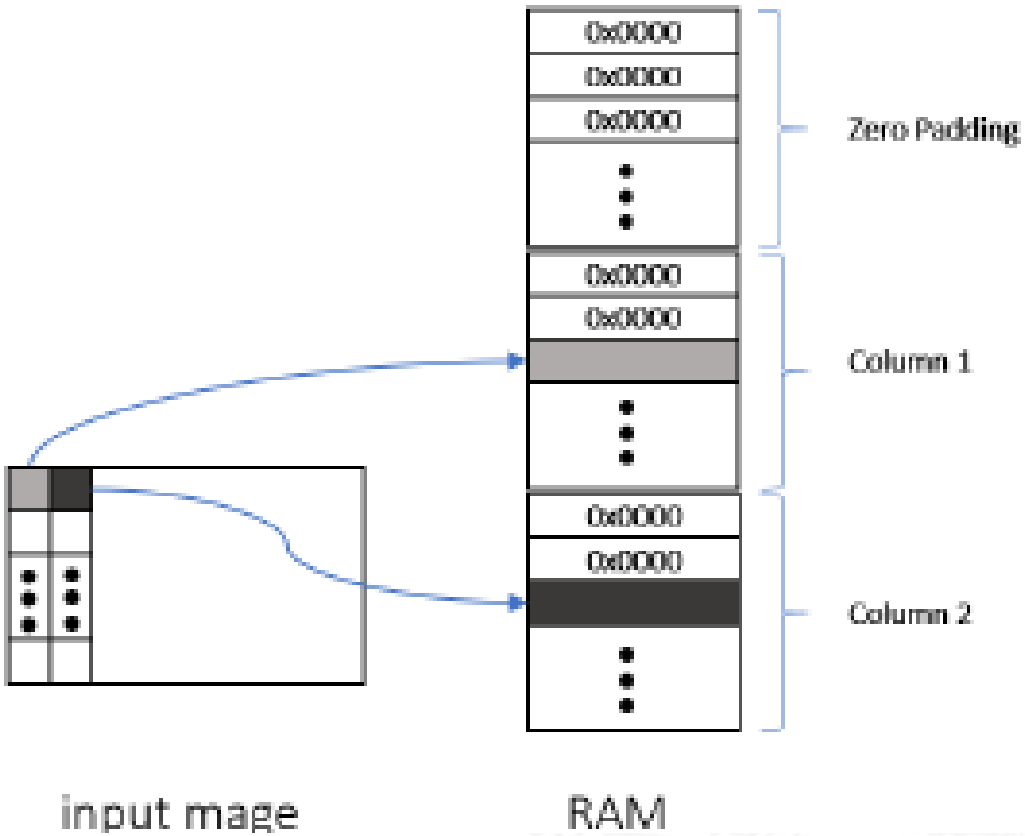


Figure: Zero padding in RAM [9]

Appendix (cont.)


```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity binary_adder_tree is
6
7   port
8   (
9   a    : in unsigned (7 downto 0);
10  b    : in unsigned (7 downto 0);
11  c    : in unsigned (7 downto 0);
12  d    : in unsigned (7 downto 0);
13  e    : in unsigned (7 downto 0);
14  clk    : in std_logic;
15  result : out unsigned (7 downto 0)
16  );
17
18  end entity;
```

Appendix (cont.)

```
19  architecture rtl of binary_adder_tree is
20
21  -- Declare registers to hold intermediate sums
22  signal sum1, sum2, sum3 : unsigned (7 downto 0);
23
24  begin
25
26  process (clk)
27  begin
28  if (rising_edge(clk)) then
29
30  -- Generate and store intermediate values in the pipeline
31  sum1 <= a + b;
32  sum2 <= c + d;
33  sum3 <= sum1 + sum2;
34
35  -- Generate and store the last value, the result
36  result <= sum3 + e;
37
38  end if;
39  end process;
40
41  end rtl;
```

Figure: Binary adder tree VHDL [7]

```
19  library IEEE;
20      use IEEE.STD_LOGIC_1164.ALL;
21      use IEEE.NUMERIC_STD.ALL;
22
23  entity relu is
24      Generic (
25          WIDTH: natural
26      );
27
28      Port (
29          din: in std_logic_vector(WIDTH - 1 downto 0);
30          dout: out std_logic_vector(WIDTH - 1 downto 0)
31      );
32  end relu;
33
34  architecture rtl of relu is
35  begin
36
37      dout <= (others => '0') when din(WIDTH - 1) = '1' else din;
38
39  end rtl;
```

Figure: Rectified linear unit VHDL [5]

Appendix (cont.)

```
19  library IEEE;
20  use IEEE.STD_LOGIC_1164.ALL;
21
22  entity relu_wrapper is
23      Port (
24          din: in std_logic_vector(9 downto 0);
25          dout: out std_logic_vector(9 downto 0)
26      );
27  end relu_wrapper;
28
29  architecture Structural of relu_wrapper is
30
31  begin
32
33      relu_inst : entity WORK.relu
34          generic map (
35              WIDTH => 10
36          )
37          port map (
38              din => din,
39              dout => dout
40          );
41
42  end Structural;
```

Appendix (cont.)

Figure: Relu wrapper VHDL [5]

Appendix (cont.)

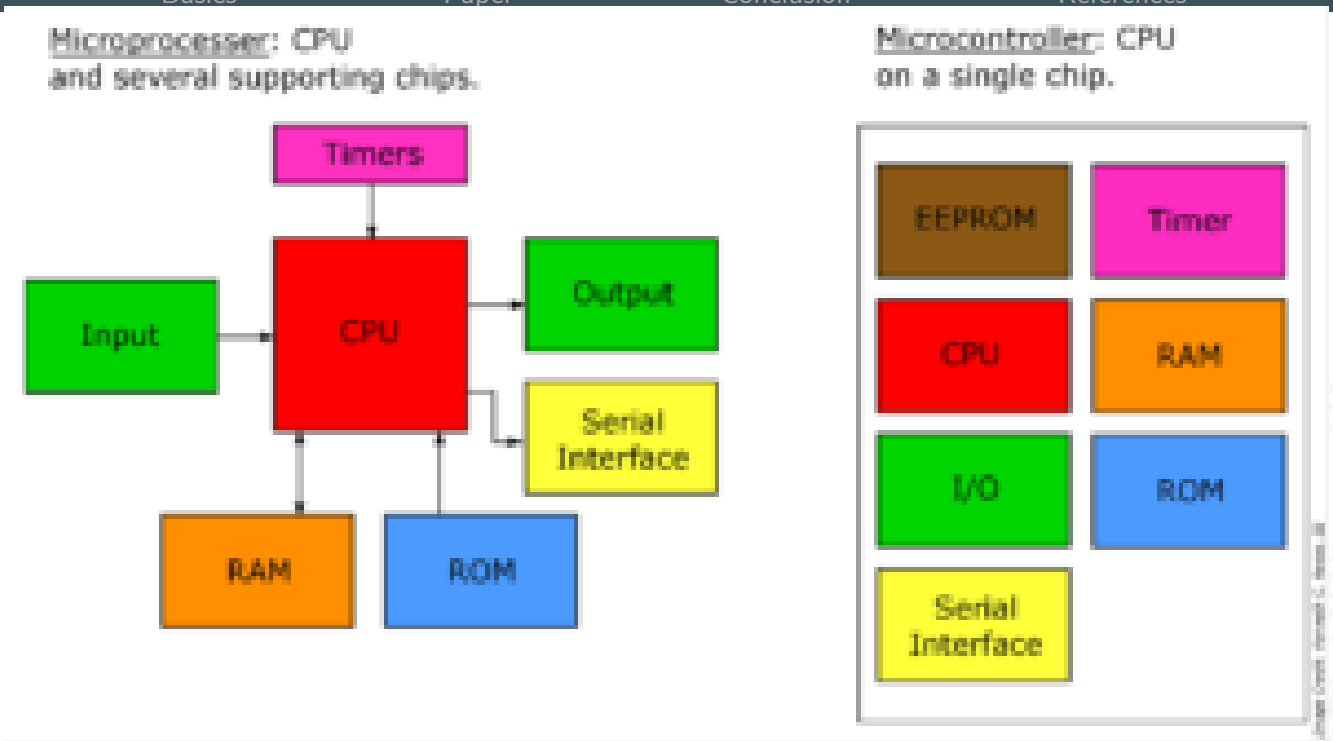


Figure: Difference of MCU and MPU [14]

- MCU: Single chip computer, single threaded, bare metal interface

Appendix (cont.)

- 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 I

ERRORS MEASURED IN PRECISION ANALYSIS

Precision	RMSE (m)	Max E. (m)
(16, 14) & (16, 5)	0.254	0.962
(31, 28) & (16, 5)	0.183	0.718

Figure: Errors in fixed point configurations [12]

Appendix (cont.)

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)
- (ρ, Θ) 64bits/ element and 18*64 bits for GPS/IMU data
- $(\varphi_r\ \varphi_p\ \varphi_y) \Rightarrow (\Delta\varphi_r\ \Delta\varphi_p\ \Delta\varphi_y)$
- $(X_{ac},\ Y_{ac},\ Z_{ac}) \Rightarrow (\Delta X_{ac},\ \Delta Y_{ac},\ \Delta Z_{ac})$

Appendix (cont.)

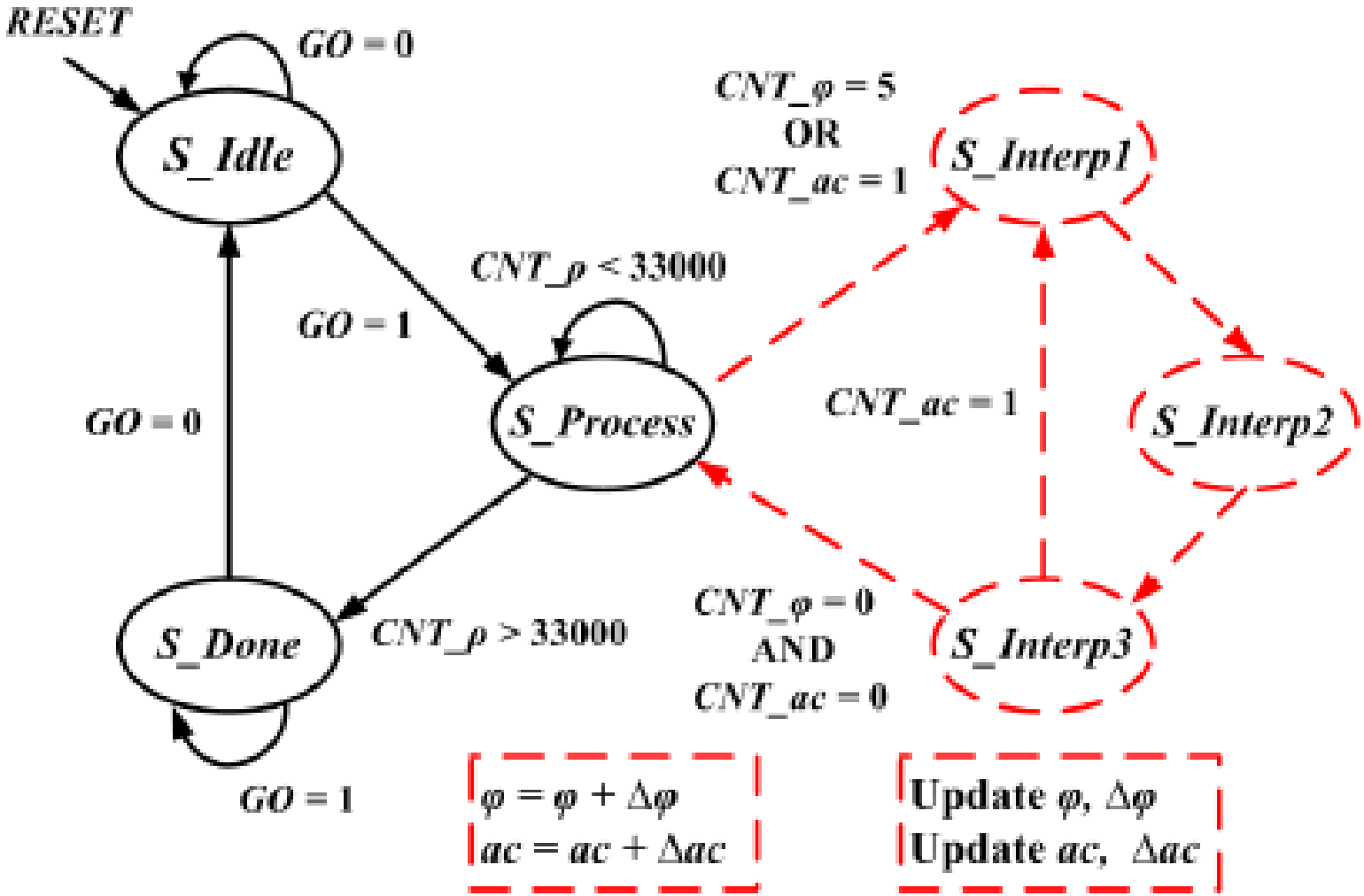


Figure: State machine with host signals (dashed lines = design 2) [12]

Appendix (cont.)

```
/* The Host side */
for i = 1 to 5
    Δφ_r(i) = [φ_r(i+1) − φ_r(i)]/[33000/5]; /* pre-calculate Δφ_r(i) */
end
```

```
/* The FPGA side */
for i = 1 to 5
    φ̃_r = φ_r(i); /* load new base value */
    for j = 1 to (33000/5)
        φ̃_r = φ̃_r + Δφ_r(i); /* accumulate increment to base value */
        (X, Y, Z) ← f(..., φ̃_r, ...); /* coordinate calculation using φ̃_r */
    end
end
```

Figure: Pseudocode design 2 [12]

Appendix (cont.)

TABLE II
DESIGN PROGRESSION THROUGH RAT ANALYSIS

Data Set Parameters		Design 1 (Nallatech)	Design 1 (Cray)	Design 2 (Cray)
$N_{elements,input}$	(elem.)	66000	66000	33018
$N_{elements,output}$	(elem.)	33000	33000	33000
$N_{bytes/element}$	(B/elem.)	8	8	8
Communication Parameters		**Design 1 (Nallatech)**	**Design 1 (Cray)**	**Design 2 (Cray)**
$Throughput_{ideal}$	(MB/s)	1000	1638.4	1638.4
α_{read}	$0<\alpha<1$	0.25	0.5	0.5
α_{write}	$0<\alpha<1$	0.25	0.5	0.5
Computation Parameters		**Design 1 (Nallatech)**	**Design 1 (Cray)**	**Design 2 (Cray)**
$N_{elements,proc}$	(elem.)	33000	33000	33000
$N_{ops/element}$	(ops/elem.)	9	9	10
$Throughput_{proc}$	(ops/cycle)	9	9	10
f_{clock}	(MHz)	125	125	125
Software Parameters		**Design 1 (Nallatech)**	**Design 1 (Cray)**	**Design 2 (Cray)**
t_{soft}	(sec)	1.09E-02	1.09E-02	1.09E-02
N_{iter}	(iter.)	1	1	1
Calculated Metrics		**Design 1 (Nallatech)**	**Design 1 (Cray)**	**Design 2 (Cray)**
t_{comm}	(sec)	3.16E-03	9.67E-04	6.45E-04
t_{comp}	(sec)	2.64E-04	2.64E-04	2.64E-04
t_{RC}	(sec)	3.43E-03	1.23E-04	9.09E-04
Predicted Speedup		**3.2**	**8.9**	**12.0**

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

Video

DLR Crash Report [4]

Appendix (cont.)

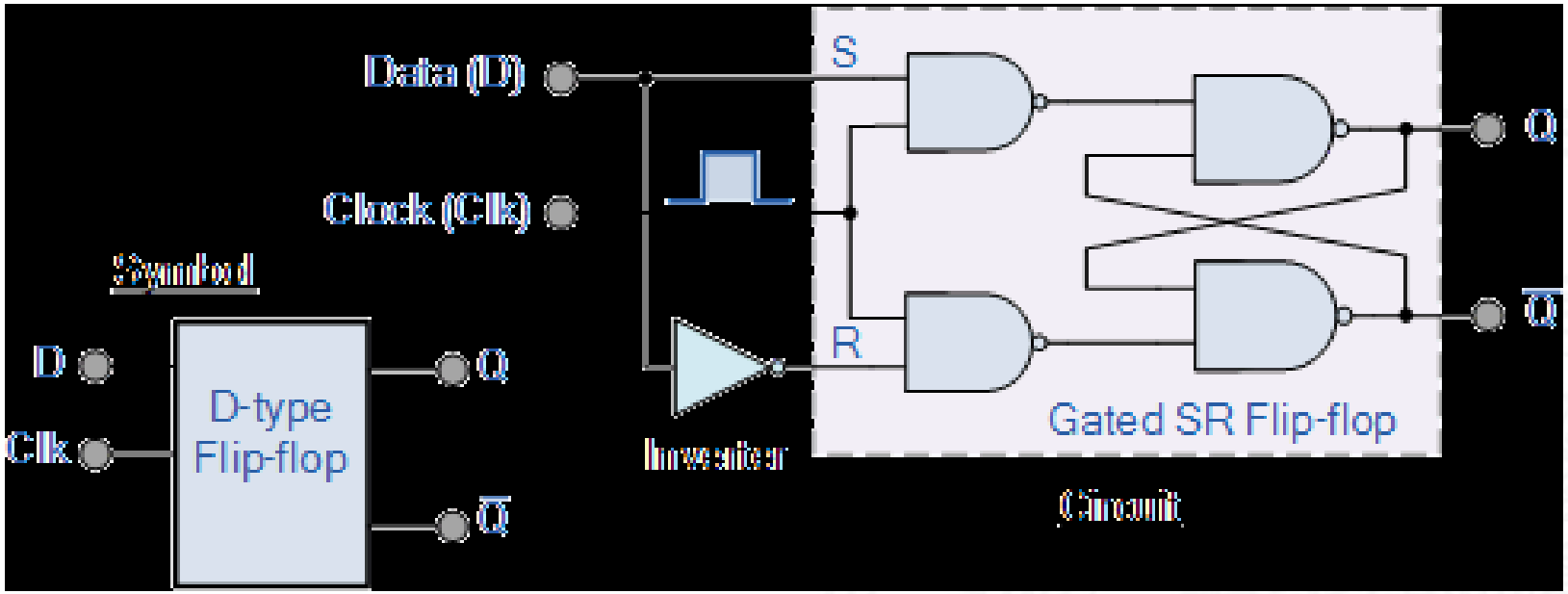


Figure: Circuit and symbol of D-FlipFlop [1]

Appendix (cont.)

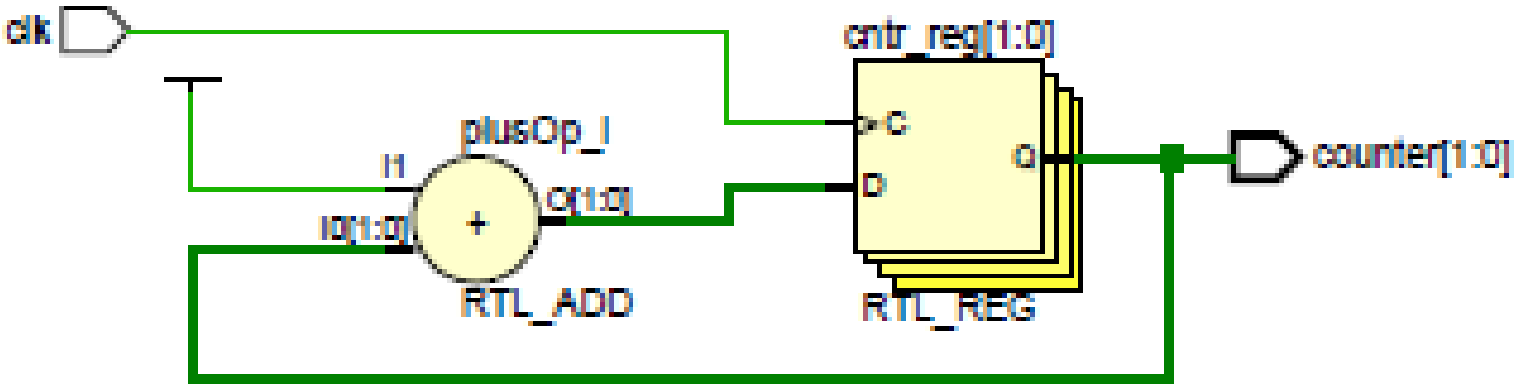


Figure: Shift register by FFs in series [2]

Appendix (cont.)

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]