HLS: Clarke Inverse

1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Introduction

Function

This IP core, implemented in the form of a C function with Vivado HLS, realizes the `inverse Clarke transform` used in the `field-oriented control (FOC)` method.It transforms the input AXI4-Stream, consisting of values V_α and V_β, to the output AXI4-Streams, consisting of the three phase voltages, V_a, V_b and V_c, by using the following equations:

$$V_a = V_\alpha, \tag{1.1}$$

$$V_b = \frac{-V_\alpha + \sqrt{3}V_\beta}{2}, \tag{1.2}$$

$$V_c = \frac{-V_\alpha - \sqrt{3}V_\beta}{2}. \tag{1.3}$$

Implementation

Applicable Devices

This HLS C function and generated IP core can be used on any Xilinx devices supported by Vivado HLS.

Synthesis Report

The target device used for synthesis: xc7z020clg400-1.

See the chapter Vivado HLS Report for 'Clarke_Inverse' for the synthesis report, including the following:

- Estimates of the used primitives in the section "Utilization Estimates".
- Timing performance estimates in the section "Performance Estimates" for the following:
  - Maximum clock frequency.
  - Latency, both minimum and maximum.
  - Interval, both minimum and maximum.
- RTL interfaces, including AXI4-Stream interfaces and additional RTL ports added by the HLS synthesis, in the section "Interface".

Interface

The interface described in the form of a C function is as follows:

```
void Clarke_Inverse(
    hls::stream<int64_t> &inputStream,
    hls::stream<int64_t> &outputStream);
```

See the description of the function Clarke_Inverse() for the encoding of the input and output streams.

Simulation

A C-based testbench for C/RTL cosimulation is in the file test_clarke_inverse.cpp.

Tools

Vivado HLS is needed for C to RTL synthesis, for C simulation and for IP packaging (export). The function itself can be implemented with Vivado.

Doxygen is used for generating documentation from the comments included in the C source code.

Tool	Version	Notes
Vivado HLS	2017.1	Synthesis, C simulation, RTL export
Vivado	2017.1	Implementation
Doxygen	1.8.11	Documentation extraction
MiKTeX	2.9	PDF generation

Synthesis Report

See the chapter Vivado HLS Report for 'Clarke_Inverse'

Chapter 2

Vivado HLS Report for 'Clarke_Inverse'

Date:	Sat Jun 10 12:29:05 2017
Version:	2017.1 (Build 1846317 on Fri Apr 14 19:19:38 MDT 2017)
Project:	Clarke_Inverse
Solution:	solution1
Product family:	zynq
Target device:	xc7z020clg400-1

Performance Estimates

Timing (ns)

Table 2.2 Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.19	1.25

Latency (clock cycles)

Table 2.3 Summary

Latency		Interval		Pipeline
min	max	min	max	Type
4	4	5	5	none

Detail

Instance: N/A

Loop: N/A

Utilization Estimates

Table 2.4 Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	165
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	99
Register	-	-	354	-
Total	0	1	354	264
Available	280	220	106400	53200
Utilization (%)	0	∼0	∼0	∼0

Detail

Instance: N/A

Table 2.5 DSP48

Instance	Module	Expression
Clarke_Inverse_mubkb_U0	Clarke_Inverse_mubkb	i0 ∗ i1

Memory: N/A

FIFO: N/A

Table 2.6 Expression

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
sum_fu_132_p2	+	0	0	18	18	18
tmp_6_fu_116_p2	-	0	0	25	18	18
tmp_8_fu_138_p2	-	0	0	18	1	18
m_axis_V_1_load_A	and	0	0	2	1	1
m_axis_V_1_load_B	and	0	0	2	1	1
s_axis_V_0_load_A	and	0	0	2	1	1
s_axis_V_0_load_B	and	0	0	2	1	1
icmp3_fu_180_p2	icmp	0	0	1	2	1
icmp_fu_164_p2	icmp	0	0	1	2	1
m_axis_V_1_state_cmp_full	icmp	0	0	1	2	1
s_axis_V_0_state_cmp_full	icmp	0	0	1	2	1
tmp_7_fu_192_p2	icmp	0	0	13	17	16
tmp_s_fu_204_p2	icmp	0	0	13	17	16
Vb_fu_186_p3	select	0	0	17	1	15
Vc_fu_198_p3	select	0	0	17	1	15
tmp_10_fu_214_p3	select	0	0	16	1	16
tmp_12_fu_226_p3	select	0	0	16	1	16
Total		0	0	165	87	156

Table 2.7 Multiplexer

Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	33	6	1	6
m_axis_V_1_data_out	9	2	64	128
m_axis_V_1_state	15	3	2	6
m_axis_V_TDATA_blk↩_n	9	2	1	2
s_axis_V_0_data_out	9	2	64	128
s_axis_V_0_state	15	3	2	6
s_axis_V_TDATA_blk↩_n	9	2	1	2
Total	99	20	135	278

Table 2.8 Register

Name	FF	LUT	Bits	Const Bits
Theta_reg_258	16	0	16	0
Valpha_reg_252	16	0	16	0
ap_CS_fsm	5	0	5	0
icmp3_reg_283	1	0	1	0
icmp_reg_278	1	0	1	0
m_axis_V_1_payload↩_A	64	0	64	0
m_axis_V_1_payload↩_B	64	0	64	0
m_axis_V_1_sel_rd	1	0	1	0
m_axis_V_1_sel_wr	1	0	1	0
m_axis_V_1_state	2	0	2	0
s_axis_V_0_payload_A	64	0	64	0
s_axis_V_0_payload_B	64	0	64	0
s_axis_V_0_sel_rd	1	0	1	0
s_axis_V_0_sel_wr	1	0	1	0
s_axis_V_0_state	2	0	2	0
tmp_1_reg_268	17	0	17	0
tmp_4_reg_273	17	0	17	0
tmp_5_reg_263	17	0	17	0
Total	354	0	354	0

Interface

Table 2.9 Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	Clarke_Inverse	return value
ap_rst_n	in	1	ap_ctrl_hs	Clarke_Inverse	return value
ap_start	in	1	ap_ctrl_hs	Clarke_Inverse	return value
ap_done	out	1	ap_ctrl_hs	Clarke_Inverse	return value
ap_idle	out	1	ap_ctrl_hs	Clarke_Inverse	return value
ap_ready	out	1	ap_ctrl_hs	Clarke_Inverse	return value
s_axis_V_TDATA	in	64	axis	s_axis_V	pointer

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axis_V_TVALID	in	1	axis	s_axis_V	pointer
s_axis_V_TREADY	out	1	axis	s_axis_V	pointer
m_axis_V_TDATA	out	64	axis	m_axis_V	pointer
m_axis_V_TVALID	out	1	axis	m_axis_V	pointer
m_axis_V_TREADY	in	1	axis	m_axis_V	pointer

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

File Documentation

4.1 clarke_inverse.cpp File Reference

Implementation of the inverse Clarke transform.

```
#include "clarke_inverse.h"
```

Functions

- void Clarke_Inverse (hls::stream< int64_t > &s_axis, hls::stream< int64_t > &m_axis)
  Inverse Clarke transform as AXI4-Stream IP core.

4.1.1 Detailed Description

Implementation of the inverse Clarke transform.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

4.1.2 Function Documentation

4.1.2.1 Clarke_Inverse()

```
void Clarke_Inverse (
            hls::stream< int64_t > & s_axis,
            hls::stream< int64_t > & m_axis )
```

Inverse Clarke transform as AXI4-Stream IP core.

It calculates the values V_a, V_b and V_c in the ouput AXI4-Stream `m_axis` by using the following equations:

$$V_a = V_\alpha, \tag{4.1}$$

$$V_b = \frac{-V_\alpha + \sqrt{3}V_\beta}{2}, \tag{4.2}$$

$$V_c = \frac{-V_\alpha - \sqrt{3}V_\beta}{2}. \tag{4.3}$$

where V_α and V_β are from the input AXI4-Stream `s_axis`.

Parameters

s_axis	Input AXI4-Stream with the following layout: • Bits 0..16: V_α • Bits 17..31: V_β • Bits 32..47: Angle, in encoder steps. • Bits 48..63: Unused. All values are 16-bit signed integers.
m_axis	Output AXI4-Stream with the following layout: • Bits 0..15: V_a. • Bits 16..31: V_b. • Bits 32..47: V_c. • Bits 48..63: Angle, in encoder steps. All values are 16-bit signed integers.

Returns

void - functions implementing an IP core do not return a value.

Definition at line 17 of file clarke_inverse.cpp.

```
17                                                                  {
18
```

```
19 #pragma HLS interface axis port=m_axis
20 #pragma HLS interface axis port=s_axis
21     int64_t in_data, res;
22     int16_t Valpha, Vbeta, Theta;
23     int32_t s3vb;                                   // Clarke Inverse
24     int32_t Va, Vb, Vc;                          // Clarke Inverse -> SVPWM
25
26     // Decode Input stream
27     in_data = s_axis.read();                     // Read one value from AXI4-Stream
28     Valpha = int16_t(in_data & 0xFFFF);             // Extract Valpha - bits[15..0] from input stream
29     Vbeta = int16_t((in_data >> 16) & 0xFFFF);      // Extract Vbeta - bits[31..16] from input stream
30     Theta = int16_t((in_data >> 32) & 0xFFFF);      // Extract Theta - bits[47..32] from input stream
31
32     // Process data
33     Va = Valpha;
34     s3vb = Vbeta * SQRT3C;                         // (sqrt(3)*(2^15))*Vbeta
35     Vb = ((s3vb >> 15) - Valpha) >> 1;           // (-Valpha + sqrt(3)*Vbeta)/2
36     Vc = (0 - Valpha - (s3vb >> 15)) >> 1;       // (-Valpha - sqrt(3)*Vbeta)/2
37     Vb = (Vb > MAX_LIM) ? MAX_LIM : Vb;            // Clip max
38     Vb = (Vb < MIN_LIM) ? MIN_LIM : Vb;            // Clip min
39     Vc = (Vc > MAX_LIM) ? MAX_LIM : Vc;            // Clip max
40     Vc = (Vc < MIN_LIM) ? MIN_LIM : Vc;            // Clip min
41
42     // Write output stream
43     res =   (((int64_t)Theta << 48) & 0xFFFF000000000000) | // Put Theta bits[63:48]
44             (((int64_t)Vc << 32)    & 0x0000FFFF00000000) | // Put Vc bits[47:32]
45             (((int64_t)Vb << 16)    & 0x00000000FFFF0000) | // Put Vb bits[31:16]
46             ( (int64_t)Va           & 0x000000000000FFFF);  // Put Va bits[15:0]
47     m_axis.write(res);                             // Write result to the output stream
48 }
```

4.2 clarke_inverse.h File Reference

Header file for the inverse Clarke transform.

```
#include <hls_stream.h>
#include <ap_axi_sdata.h>
#include <ap_int.h>
#include <ap_cint.h>
#include <stdint.h>
```

Macros

- #define MAX_LIM 32767

  Maximum positive value for saturated arithmetic.

- #define MIN_LIM -32767

  Minimum negative value for saturated arithmetic.

- #define SQRT3C 0x0000DDB4

  The number $\frac{1}{\sqrt{3}}$ *in the Q16.16 format.*

Functions

- void Clarke_Inverse (hls::stream< int64_t > &s_axis, hls::stream< int64_t > &m_axis)

  Inverse Clarke transform as AXI4-Stream IP core.

4.2.1 Detailed Description

Header file for the inverse Clarke transform.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

4.2.2 Macro Definition Documentation

4.2.2.1 MAX_LIM

```
#define MAX_LIM 32767
```

Maximum positive value for saturated arithmetic.

Definition at line 20 of file clarke_inverse.h.

4.2.2.2 MIN_LIM

```
#define MIN_LIM -32767
```

Minimum negative value for saturated arithmetic.

Definition at line 23 of file clarke_inverse.h.

4.2.2.3 SQRT3C

```
#define SQRT3C 0x0000DDB4
```

The number $\frac{1}{\sqrt{3}}$ in the Q16.16 format.

Definition at line 26 of file clarke_inverse.h.

4.2.3 Function Documentation

4.2.3.1 Clarke_Inverse()

```
void Clarke_Inverse (
            hls::stream< int64_t > & s_axis,
            hls::stream< int64_t > & m_axis )
```

Inverse Clarke transform as AXI4-Stream IP core.

It calculates the values V_a, V_b and V_c in the ouput AXI4-Stream `m_axis` by using the following equations:

$$V_a = V_\alpha, \tag{4.4}$$

$$V_b = \frac{-V_\alpha + \sqrt{3}V_\beta}{2}, \tag{4.5}$$

$$V_c = \frac{-V_\alpha - \sqrt{3}V_\beta}{2}. \tag{4.6}$$

where V_α and V_β are from the input AXI4-Stream `s_axis`.

Parameters

s_axis	Input AXI4-Stream with the following layout: • Bits 0..16: V_α • Bits 17..31: V_β • Bits 32..47: Angle, in encoder steps. • Bits 48..63: Unused. All values are 16-bit signed integers.
m_axis	Output AXI4-Stream with the following layout: • Bits 0..15: V_a. • Bits 16..31: V_b. • Bits 32..47: V_c. • Bits 48..63: Angle, in encoder steps. All values are 16-bit signed integers.

Returns

void - functions implementing an IP core do not return a value.

Definition at line 17 of file clarke_inverse.cpp.

```
17                                                                  {
18
```

```
19 #pragma HLS interface axis port=m_axis
20 #pragma HLS interface axis port=s_axis
21     int64_t in_data, res;
22     int16_t Valpha, Vbeta, Theta;
23     int32_t s3vb;                                   // Clarke Inverse
24     int32_t Va, Vb, Vc;                          // Clarke Inverse -> SVPWM
25
26     // Decode Input stream
27     in_data = s_axis.read();                     // Read one value from AXI4-Stream
28     Valpha = int16_t(in_data & 0xFFFF);             // Extract Valpha - bits[15..0] from input stream
29     Vbeta = int16_t((in_data >> 16) & 0xFFFF);      // Extract Vbeta - bits[31..16] from input stream
30     Theta = int16_t((in_data >> 32) & 0xFFFF);      // Extract Theta - bits[47..32] from input stream
31
32     // Process data
33     Va = Valpha;
34     s3vb = Vbeta * SQRT3C;                         // (sqrt(3)*(2^15))*Vbeta
35     Vb = ((s3vb >> 15) - Valpha) >> 1;           // (-Valpha + sqrt(3)*Vbeta)/2
36     Vc = (0 - Valpha - (s3vb >> 15)) >> 1;       // (-Valpha - sqrt(3)*Vbeta)/2
37     Vb = (Vb > MAX_LIM) ? MAX_LIM : Vb;            // Clip max
38     Vb = (Vb < MIN_LIM) ? MIN_LIM : Vb;            // Clip min
39     Vc = (Vc > MAX_LIM) ? MAX_LIM : Vc;            // Clip max
40     Vc = (Vc < MIN_LIM) ? MIN_LIM : Vc;            // Clip min
41
42     // Write output stream
43     res =   (((int64_t)Theta << 48) & 0xFFFF000000000000) | // Put Theta bits[63:48]
44             (((int64_t)Vc << 32)    & 0x0000FFFF00000000) | // Put Vc bits[47:32]
45             (((int64_t)Vb << 16)    & 0x00000000FFFF0000) | // Put Vb bits[31:16]
46             ( (int64_t)Va           & 0x000000000000FFFF);  // Put Va bits[15:0]
47     m_axis.write(res);                             // Write result to the output stream
48 }
```

4.3 doxygen/src/Clarke_Inverse_csynth.dox File Reference

4.4 doxygen/src/main_page.dox File Reference

4.5 test_clarke_inverse.cpp File Reference

Testbench for the inverse Clarke transform.

```
#include "clarke_inverse.h"
```

Macros

- #define TEST_SIZE 10

  Number of values to test with.

Functions

- int main ()

  Main function of the C testbench.

Variables

- int Valpha [TEST_SIZE] = {-600, 2000, 100, 555, -255, 3333, -765, 333, 200, -543}

  Values of V_α to test Clarke_Inverse() with.
- int Vbeta [TEST_SIZE] = {-888, 3000, -500, 7000, 1000, -123, -800, 9000, 789, -444}

  Values of V_β to test Clarke_inverse() with.

4.5.1 Detailed Description

Testbench for the inverse Clarke transform.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

4.5.2 Macro Definition Documentation

4.5.2.1 TEST_SIZE

```
#define TEST_SIZE 10
```

Number of values to test with.

Definition at line 14 of file test_clarke_inverse.cpp.

4.5.3 Function Documentation

4.5.3.1 main()

```
int main ( )
```

Main function of the C testbench.

The function Clarke_Inverse() will be called with the values of V_α and V_β in Valpha and Vbeta and the results will be printed along with separately calculated values.

Definition at line 28 of file test_clarke_inverse.cpp.

```
28          {
29     int i;
30     hls::stream<int32_t> inputStream;
31     hls::stream<int64_t> outputStream;
32     int32_t tx_data;
33     int64_t rx_data;
34     int16_t ia, ib, ic;
35     float fa, fb, fc;
36 
37     for(i=0; i<TEST_SIZE; i++){
38         tx_data = (int32_t(Vbeta[i]) << 16) | (int32_t(Valpha[i]) & 0x0000FFFF);
39         inputStream << tx_data;
40 
41         Clarke_Inverse(inputStream, outputStream);
42 
43         outputStream.read(rx_data);
44         ia = int16_t(rx_data & 0xFFFF);
45         ib = int16_t((rx_data & 0xFFFF0000) >> 16);
46         ic = int16_t((rx_data & 0xFFFF00000000) >> 32);
47 
48         fa = float(Valpha[i]);
49         fb = (- float(Valpha[i]) + sqrt(3.0)*Vbeta[i])/2.0;
50         fc = (- float(Valpha[i]) - sqrt(3.0)*Vbeta[i])/2.0;
51         printf("Values is Ia=%d Ib=%d Ic=%d (%f %f %f)\n",ia, ib, ic, fa, fb, fc);
52     }
53 }
```

4.5.4 Variable Documentation

4.5.4.1 Valpha

```
int Valpha[TEST_SIZE] = {-600, 2000, 100, 555, -255, 3333, -765, 333, 200, -543}
```

Values of V_α to test Clarke_Inverse() with.

Definition at line 17 of file test_clarke_inverse.cpp.

4.5.4.2 Vbeta

```
int Vbeta[TEST_SIZE] = {-888, 3000, -500, 7000, 1000, -123, -800, 9000, 789, -444}
```

Values of V_β to test Clarke_inverse() with.

Definition at line 20 of file test_clarke_inverse.cpp.

Index