HLS: Clarke Direct

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 `Clarke transform` used in the `field-oriented control (FOC)` method.It transforms the input AXI4-Stream, consisting of the currents of the two phases, I_a and I_b, to the output AXI4-Stream, consisting of values I_α and I_β by using the following equations:

$$I_\alpha = I_a, \tag{1.1}$$

$$I_\beta = \frac{I_a + 2I_b}{\sqrt{3}}. \tag{1.2}$$

The value of the third phase current, I_c, is not included in the calculations because it has been optimized out by using the invariant that the sum of the three phase currents is zero.

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 is xc7z020clg400-1.

See the chapter Vivado HLS Report for 'Clarke_Direct' 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_Direct(
    hls::stream<int64_t> &inputStream,
    hls::stream<int64_t> &outputStream);
```

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

Simulation

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

Chapter 2

Vivado HLS Report for 'Clarke_Direct'

Date:	Thu Jun 8 16:14:43 2017
Version:	2017.1 (Build 1846317 on Fri Apr 14 19:19:38 MDT 2017)
Project:	Clarke_Direct
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	6.38	1.25

Latency (clock cycles)

Table 2.3 Summary

Latency		Interval		Pipeline
min	max	min	max	Type
3	3	4	4	none

Detail

Instance: N/A

Loop: N/A

Utilization Estimates

Table 2.4 Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	39
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	93
Register	-	-	332	-
Total	0	1	332	132
Available	280	220	106400	53200
Utilization (%)	0	$\sim$0	$\sim$0	$\sim$0

Detail

Instance: N/A

Table 2.5 DSP48

Instance	Module	Expression
Clarke_Direct_am_bkb_U0	Clarke_Direct_am_bkb	$i0 * (i1 + i2)$

Memory: N/A

FIFO: N/A

Table 2.6 Expression

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
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
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_8_fu_114_p2	icmp	0	0	13	16	17
tmp_3_fu_119_p3	select	0	0	16	1	16
Total		0	0	39	25	39

Table 2.7 Multiplexer

Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	27	5	1	5
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

Name	LUT	Input Size	Bits	Total Bits
Total	93	19	135	277

Table 2.8 Register

Name	FF	LUT	Bits	Const Bits
la_reg_144	16	0	16	0
ap_CS_fsm	4	0	4	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_4_reg_155	32	0	32	0
tmp_s_reg_149	16	0	16	0
Total	332	0	332	0

Interface

Table 2.9 Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	Clarke_Direct	return value
ap_rst_n	in	1	ap_ctrl_hs	Clarke_Direct	return value
ap_start	in	1	ap_ctrl_hs	Clarke_Direct	return value
ap_done	out	1	ap_ctrl_hs	Clarke_Direct	return value
ap_idle	out	1	ap_ctrl_hs	Clarke_Direct	return value
ap_ready	out	1	ap_ctrl_hs	Clarke_Direct	return value
s_axis_V_TDATA	in	64	axis	s_axis_V	pointer
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_direct.cpp File Reference

Implementation of the Clarke transform.

```
#include "clarke_direct.h"
```

Functions

- void Clarke_Direct (hls::stream$<$ int64_t $>$ &s_axis, hls::stream$<$ int64_t $>$ &m_axis)
  Clark transform as AXI4-Stream IP core.

4.1.1 Detailed Description

Implementation of the Clarke transform.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

4.1.2 Function Documentation

4.1.2.1 Clarke_Direct()

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

Clark transform as AXI4-Stream IP core.

It calculates the values I_α and I_β in the ouput AXI4-Stream `m_axis` by using the following equations:

$$I_\alpha = I_a, \tag{4.1}$$

$$I_\beta = \frac{I_a + 2I_b}{\sqrt{3}}, \tag{4.2}$$

where I_a and I_b are from the input AXI4-Stream `s_axis`.

Parameters

s_axis	Input AXI4-Stream with the following layout: • Bits 0..15: First phase current I_a, from the ADC. • Bits 16..31: Second phase current I_b, from the ADC. • Bits 32..47: Speed, in RPM, just passed through. • Bits 48..63: Angle, in encoder steps, just passed through. All values are 16-bit signed integers.
m_axis	Output AXI4-Stream with the following layout: • Bits 0..15: I_α • Bits 16..31: I_β • Bits 32..47: Speed, in RPM. • 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_direct.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 Ia, Ib, Theta, RPM;
23     int32_t Ialpha, Ibeta, Ibd;
24
```

```
    // Decode Input stream
    in_data = s_axis.read();                    // Read one value from AXI4-Stream
    Ia = int16_t(in_data & 0xFFFF);             // Extract Ia - bits[15..0] from input stream
    Ib = int16_t((in_data >> 16) & 0xFFFF);     // Extract Ib - bits[32..16] from input stream
    RPM = int16_t((in_data >> 32) & 0xFFFF);    // Extract RPM - bits[47..32] from input stream
    Theta = int16_t((in_data >> 48) & 0xFFFF);  // Extract Angle - bits[63..48] from input stream

    // Process data
    Ialpha = (int32_t)Ia;
    Ibd = Ialpha + ((int32_t)Ib << 1);              // calculate Ia+2*Ib
    Ibeta = (Ibd * SQRT3A) >> 16;                 // * 1/SQRT(3)
    Ibeta = (Ibeta > MAX_LIM) ? MAX_LIM : Ibeta;  // Clip max
    Ibeta = (Ibeta < MIN_LIM) ? MIN_LIM : Ibeta;  // Clip min

    // Write output stream
    res =    (((int64_t)Theta << 48) & 0xFFFF000000000000) | // Put Angle bits[63:48]
             (((int64_t)RPM << 32)   & 0x0000FFFF00000000) | // Put RPM bits[47:32]
             (((int64_t)Ibeta << 16) & 0x00000000FFFF0000) | // Put Ibeta bits[31:16]
             ( (int64_t)Ialpha       & 0x000000000000FFFF);  // Put Ialpha bits[15:0]
    m_axis.write(res);                              // Write result to the output stream
}
```

4.2 clarke_direct.h File Reference

Header file for the 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 SQRT3A 0x000093CD

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

Functions

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

  Clark transform as AXI4-Stream IP core.

4.2.1 Detailed Description

Header file for the Clarke transform.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

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_direct.h.

4.2.2.2 MIN_LIM

```
#define MIN_LIM -32767
```

Minimum negative value for saturated arithmetic.

Definition at line 23 of file clarke_direct.h.

4.2.2.3 SQRT3A

```
#define SQRT3A 0x000093CD
```

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

Definition at line 26 of file clarke_direct.h.

4.2.3 Function Documentation

4.2.3.1 Clarke_Direct()

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

Clark transform as AXI4-Stream IP core.

It calculates the values I_α and I_β in the ouput AXI4-Stream `m_axis` by using the following equations:

$$I_\alpha = I_a, \tag{4.3}$$

$$I_\beta = \frac{I_a + 2I_b}{\sqrt{3}}, \tag{4.4}$$

where I_a and I_b are from the input AXI4-Stream `s_axis`.

Parameters

s_axis	Input AXI4-Stream with the following layout: • Bits 0..15: First phase current I_a, from the ADC. • Bits 16..31: Second phase current I_b, from the ADC. • Bits 32..47: Speed, in RPM, just passed through. • Bits 48..63: Angle, in encoder steps, just passed through. All values are 16-bit signed integers.
m_axis	Output AXI4-Stream with the following layout: • Bits 0..15: I_α • Bits 16..31: I_β • Bits 32..47: Speed, in RPM. • 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_direct.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 Ia, Ib, Theta, RPM;
23     int32_t Ialpha, Ibeta, Ibd;
24
25     // Decode Input stream
26     in_data = s_axis.read();                    // Read one value from AXI4-Stream
27     Ia = int16_t(in_data & 0xFFFF);             // Extract Ia - bits[15..0] from input stream
28     Ib = int16_t((in_data >> 16) & 0xFFFF);     // Extract Ib - bits[32..16] from input stream
29     RPM = int16_t((in_data >> 32) & 0xFFFF);    // Extract RPM - bits[47..32] from input stream
30     Theta = int16_t((in_data >> 48) & 0xFFFF);  // Extract Angle - bits[63..48] from input stream
31
32     // Process data
33     Ialpha = (int32_t)Ia;
34     Ibd = Ialpha + ((int32_t)Ib << 1);              // calculate Ia+2*Ib
35     Ibeta = (Ibd * SQRT3A) >> 16;                 // * 1/SQRT(3)
36     Ibeta = (Ibeta > MAX_LIM) ? MAX_LIM : Ibeta;  // Clip max
37     Ibeta = (Ibeta < MIN_LIM) ? MIN_LIM : Ibeta;  // Clip min
38
39     // Write output stream
40     res =    (((int64_t)Theta << 48)  & 0xFFFF000000000000) | // Put Angle bits[63:48]
41              (((int64_t)RPM << 32)    & 0x0000FFFF00000000) | // Put RPM bits[47:32]
42              (((int64_t)Ibeta << 16)  & 0x00000000FFFF0000) | // Put Ibeta bits[31:16]
43              ( (int64_t)Ialpha        & 0x000000000000FFFF);  // Put Ialpha bits[15:0]
44     m_axis.write(res);                              // Write result to the output stream
45 }
```

4.3 doxygen/src/Clarke_Direct_csynth.dox File Reference

4.4 doxygen/src/main_page.dox File Reference

4.5 test_clarke_direct.cpp File Reference

Testbench for the Clarke transform.

```
#include "clarke_direct.h"
#include <math.h>
```

Macros

- #define TEST_SIZE 1000
  Set the loop count for the testbench.
- #define MAX_VAL 32257
  Maximum value for 16 bit signed integer.
- #define M_PI 3.14159265358979323846
  Constant Pi as float number.

Functions

- int main ()
  Main function of the C testbench.

4.5.1 Detailed Description

Testbench for the 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 M_PI

```
#define M_PI 3.14159265358979323846
```

Constant Pi as float number.

Definition at line 31 of file test_clarke_direct.cpp.

4.5.2.2 MAX_VAL

```
#define MAX_VAL 32257
```

Maximum value for 16 bit signed integer.

Definition at line 25 of file test_clarke_direct.cpp.

4.5.2.3 TEST_SIZE

```
#define TEST_SIZE 1000
```

Set the loop count for the testbench.

Definition at line 19 of file test_clarke_direct.cpp.

4.5.3 Function Documentation

4.5.3.1 main()

```
int main ( )
```

Main function of the C testbench.

The function Clarke_Direct() will be called 1000 times with different values for i_a and i_b and the results will be printed along with separately calculated values.

Definition at line 42 of file test_clarke_direct.cpp.

```
        {

    hls::stream<int32_t> inputStream;

    hls::stream<int32_t> outputStream;

    int32_t tx_data, rx_data;
    int16_t Ia, Ib, Ialpha, Ibeta;

    int i;
    float Iaf, Ibf, Theta_a, Theta_b;
    float Ialphaf, Ibetaf;

    for(i=0;i<TEST_SIZE;i++){
        Theta_a = ((2.0*M_PI)/float(TEST_SIZE))*i;
        Theta_b = ((2.0*M_PI)/float(TEST_SIZE))*i;
        Iaf = MAX_VAL*cos(Theta_a);
        Ibf = MAX_VAL*cos(Theta_b+M_PI/3);

        tx_data = (int32_t(round(Ibf))<<16) | (int32_t(round(Iaf)) & 0x0000FFFF);
        //----------------------------------------------------------------------------
        // Call the RTL function, prepare input values and read the result
        //
        inputStream << tx_data;                     // send test data to input stream to be read by the function implemented in RTL
        Clarke_Direct(inputStream, outputStream);  // This function is executed as RTL simulation
        outputStream.read(rx_data);
        // End of function that interact with the function in RTL
        //-----------------------------------------------------------------------------

        Ialphaf = Iaf;
        Ibetaf = (Iaf + 2.0*Ibf)/sqrt(3.0);


        Ialpha = int16_t(rx_data & 0xFFFF);
        Ibeta = int16_t(rx_data >> 16);

        printf("\n%d Ialpha=%d(%.1f) Ib=%d(%.1f) Ibeta=%d(%.1f)\n",i, Ialpha,Ialphaf,int32_t(round(Ibf)),Ibf,Ibeta,Ibetaf);
    }
}
```

Index