HLS: SVPWM

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 SVPWM transform used in the field-oriented control (FOC) method.In this core used simplified SVPWM algorithm which is equivalent to the conventional modulation. At first stage calculated minimum and maximum of all three phase voltages to provide voltage offset for second stage. Then voltage offset is substracted from the instantaneous three phase voltages. It transforms the input AXI4-Stream, consisting of the sinus voltages of the three phases V_a, V_b and V_c, to the output AXI4-Stream, consisting of values V_a', V_b' and V_c' by using the following equations:

$$V_a' = V_a - V_{off}, \tag{1.1}$$

$$V_b' = V_b - V_{off}, \tag{1.2}$$

$$V_c' = V_c - V_{off}, \tag{1.3}$$

where

$$V_{off} = \frac{\min(V_a, V_b, V_c) + \max(V_a, V_b, V_c)}{2} \tag{1.4}$$

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 'SVPWM' 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 SVPWM(
    hls::stream<int64_t> &inputStream,
    hls::stream<int64_t> &outputStream);
```

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

Simulation

A C-based testbench for C/RTL cosimulation is in the file test_svpwm.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 'SVPWM'

Date:	Fri Jun 16 11:19:31 2017
Version:	2017.1 (Build 1846317 on Fri Apr 14 19:19:38 MDT 2017)
Project:	SVPWM
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	8.13	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	-	-	-	-
Expression	-	-	0	343
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	99
Register	-	-	421	-
Total	0	0	421	442
Available	280	220	106400	53200
Utilization (%)	0	0	∼0	∼0

Detail

Instance: N/A

DSP48: N/A

Memory: N/A

FIFO: N/A

Table 2.5 Expression

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
tmp_7_fu_161_p2	+	0	0	24	17	17
Van_fu_184_p2	-	0	0	24	17	17
Vbn_fu_193_p2	-	0	0	24	17	17
Vcn_fu_202_p2	-	0	0	24	17	17
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_234_p2	icmp	0	0	1	2	1
icmp6_fu_250_p2	icmp	0	0	1	2	1
icmp_fu_218_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_10_fu_274_p2	icmp	0	0	13	17	16
tmp_12_fu_286_p2	icmp	0	0	13	17	16
tmp_2_fu_262_p2	icmp	0	0	13	17	16
tmp_4_fu_103_p2	icmp	0	0	8	16	16
tmp_6_fu_117_p2	icmp	0	0	8	16	16
tmp_8_fu_123_p2	icmp	0	0	8	16	16
tmp_s_fu_137_p2	icmp	0	0	8	16	16
Van_1_fu_256_p3	select	0	0	17	1	15
Vbn_1_fu_268_p3	select	0	0	17	1	15
Vc_1_fu_143_p3	select	0	0	16	1	16
Vc_2_fu_152_p3	select	0	0	16	1	16
Vcn_1_fu_280_p3	select	0	0	17	1	15

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
tmp_14_fu_296_p3	select	0	0	16	1	16
tmp_17_fu_320_p3	select	0	0	16	1	16
tmp_27_cast_fu_308_p3	select	0	0	16	1	16
tmp_5_fu_109_p3	select	0	0	16	1	16
tmp_9_fu_129_p3	select	0	0	16	1	16
Total		0	0	343	207	346

Table 2.6 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 2.7 Register

Name	FF	LUT	Bits	Const Bits
Theta_reg_357	16	0	16	0
Va_reg_340	16	0	16	0
Van_reg_382	17	0	17	0
Vb_reg_345	16	0	16	0
Vbn_reg_387	17	0	17	0
Vc_reg_350	16	0	16	0
Vcn_reg_392	17	0	17	0
ap_CS_fsm	5	0	5	0
icmp3_reg_402	1	0	1	0
icmp6_reg_407	1	0	1	0
icmp_reg_397	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_5_reg_362	16	0	16	0
tmp_6_reg_367	1	0	1	0
tmp_9_reg_372	16	0	16	0

Name	FF	LUT	Bits	Const Bits
tmp_s_reg_377	1	0	1	0
Total	421	0	421	0

Interface

Table 2.8 Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	SVPWM	return value
ap_rst_n	in	1	ap_ctrl_hs	SVPWM	return value
ap_start	in	1	ap_ctrl_hs	SVPWM	return value
ap_done	out	1	ap_ctrl_hs	SVPWM	return value
ap_idle	out	1	ap_ctrl_hs	SVPWM	return value
ap_ready	out	1	ap_ctrl_hs	SVPWM	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 doxygen/src/main_page.dox File Reference

4.2 doxygen/src/SVPWM_csynth.dox File Reference

4.3 svpwm.cpp File Reference

SVPWM.

```
#include "svpwm.h"
```

Functions

- void SVPWM (hls::stream< int64_t > &s_axis, hls::stream< int64_t > &m_axis)
  SVPWM transformation as AXI4-Stream IP Core.

4.3.1 Detailed Description

SVPWM.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

4.3.2 Function Documentation

4.3.2.1 SVPWM()

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

SVPWM transformation as AXI4-Stream IP Core.

Parameters

s_axis	Input AXI4-Stream with the following layout: • Bits 0..15: V_a • Bits 16.31: V_b • Bits 32..47: V_a • Bits 48..63: Carried over to the output stream unchanged. 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'_a • Bits 48..63: Carried over from the input stream unchanged.

All values are 16-bit signed integers.

Returns

void - functions implementinga an IP cores do not return a value.

Definition at line 24 of file svpwm.cpp.

```
24                                                                  {
25
26 #pragma HLS interface axis port=m_axis
27 #pragma HLS interface axis port=s_axis
28     int64_t in_data, res;
29     int16_t Va, Vb, Vc, Theta;
30     int32_t Vmin, Vmax, Voff;                    // SVPWM internals
31     int32_t Van, Vbn, Vcn;                       // Normalized SVPWM data
32
33     // Decode Input stream
34     in_data = s_axis.read();                  // Read one value from AXI4-Stream
35     Va = int16_t(in_data & 0xFFFF);           // Extract Va - bits[15:0] from input stream
36     Vb = int16_t((in_data >> 16) & 0xFFFF);   // Extract Vb - bits[32:16] from input stream
37     Vc = int16_t((in_data >> 32) & 0xFFFF);   // Extract Vc - bits[47:32] from input stream
38     Theta = int16_t((in_data >> 48) & 0xFFFF);     // Extract Theta - bits[63:48] from input stream
39
40     // Process data
41     Vmin = (Va < Vb) ? Va : Vb;
```

```
42     Vmin = (Vc < Vmin) ? Vc : Vmin;
43     Vmax = (Va > Vb) ? Va : Vb;
44     Vmax = (Vc > Vmax) ? Vc : Vmax;
45     Voff = (Vmin + Vmax) >> 1;
46     Van = Va - Voff;
47     Vbn = Vb - Voff;
48     Vcn = Vc - Voff;
49     Van = (Van > MAX_LIM) ? MAX_LIM : Van;        // Clip max
50     Van = (Van < MIN_LIM) ? MIN_LIM : Van;        // Clip min
51     Vbn = (Vbn > MAX_LIM) ? MAX_LIM : Vbn;        // Clip max
52     Vbn = (Vbn < MIN_LIM) ? MIN_LIM : Vbn;        // Clip min
53     Vcn = (Vcn > MAX_LIM) ? MAX_LIM : Vcn;        // Clip max
54     Vcn = (Vcn < MIN_LIM) ? MIN_LIM : Vcn;        // Clip min
55
56     // Write output stream
57     res =   (((int64_t)Theta << 48) & 0xFFFF000000000000) | // Put Vcn bits[63:48]
58             (((int64_t)Vcn << 32)   & 0x0000FFFF00000000) | // Put Vcn bits[47:32]
59             (((int64_t)Vbn << 16)   & 0x00000000FFFF0000) | // Put Vbn bits[31:16]
60             ( (int64_t)Van          & 0x000000000000FFFF);  // Put Van bits[15:0]
61     m_axis.write(res);                            // Write result to the output stream
62 }
```

4.4 svpwm.h File Reference

Header file for SVPWM.

```
#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.

Functions

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

  SVPWM transformation as AXI4-Stream IP Core.

4.4.1 Detailed Description

Header file for SVPWM.

Test bench for the SVPWM.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

4.4.2 Macro Definition Documentation

4.4.2.1 MAX_LIM

```
#define MAX_LIM 32767
```

Maximum positive value for saturated arithmetic.

Definition at line 20 of file svpwm.h.

4.4.2.2 MIN_LIM

```
#define MIN_LIM -32767
```

Minimum negative value for saturated arithmetic.

Definition at line 23 of file svpwm.h.

4.4.3 Function Documentation

4.4.3.1 SVPWM()

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

SVPWM transformation as AXI4-Stream IP Core.

Parameters

s_axis	Input AXI4-Stream with the following layout: • Bits 0..15: V_a • Bits 16.31: V_b • Bits 32..47: V_a • Bits 48..63: Carried over to the output stream unchanged. 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'_a • Bits 48..63: Carried over from the input stream unchanged.

All values are 16-bit signed integers.

Returns

void - functions implementinga an IP cores do not return a value.

Definition at line 24 of file svpwm.cpp.

```
24                                                                  {
25
26 #pragma HLS interface axis port=m_axis
27 #pragma HLS interface axis port=s_axis
28     int64_t in_data, res;
29     int16_t Va, Vb, Vc, Theta;
30     int32_t Vmin, Vmax, Voff;                       // SVPWM internals
31     int32_t Van, Vbn, Vcn;                          // Normalized SVPWM data
32
33     // Decode Input stream
34     in_data = s_axis.read();                    // Read one value from AXI4-Stream
35     Va = int16_t(in_data & 0xFFFF);             // Extract Va - bits[15:0] from input stream
36     Vb = int16_t((in_data >> 16) & 0xFFFF);     // Extract Vb - bits[32:16] from input stream
37     Vc = int16_t((in_data >> 32) & 0xFFFF);     // Extract Vc - bits[47:32] from input stream
38     Theta = int16_t((in_data >> 48) & 0xFFFF);      // Extract Theta - bits[63:48] from input stream
39
40     // Process data
41     Vmin = (Va < Vb) ? Va : Vb;
42     Vmin = (Vc < Vmin) ? Vc : Vmin;
43     Vmax = (Va > Vb) ? Va : Vb;
44     Vmax = (Vc > Vmax) ? Vc : Vmax;
45     Voff = (Vmin + Vmax) >> 1;
46     Van = Va - Voff;
47     Vbn = Vb - Voff;
48     Vcn = Vc - Voff;
49     Van = (Van > MAX_LIM) ? MAX_LIM : Van;        // Clip max
50     Van = (Van < MIN_LIM) ? MIN_LIM : Van;        // Clip min
51     Vbn = (Vbn > MAX_LIM) ? MAX_LIM : Vbn;        // Clip max
52     Vbn = (Vbn < MIN_LIM) ? MIN_LIM : Vbn;        // Clip min
53     Vcn = (Vcn > MAX_LIM) ? MAX_LIM : Vcn;        // Clip max
54     Vcn = (Vcn < MIN_LIM) ? MIN_LIM : Vcn;        // Clip min
55
56     // Write output stream
57     res =   (((int64_t)Theta << 48) & 0xFFFF000000000000) | // Put Vcn bits[63:48]
58             (((int64_t)Vcn << 32)   & 0x0000FFFF00000000) | // Put Vcn bits[47:32]
59             (((int64_t)Vbn << 16)   & 0x00000000FFFF0000) | // Put Vbn bits[31:16]
60             ( (int64_t)Van          & 0x000000000000FFFF);  // Put Van bits[15:0]
61     m_axis.write(res);                              // Write result to the output stream
62 }
```

4.5 test_svpwm.cpp File Reference

```
#include "SVPWM.h"
```

Macros

- #define TEST_SIZE 10

  Number of values to test with.

Functions

- int main ()

  Main function of the C testbench.

Variables

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

  Values of V_a to test with.
- const int Vb [TEST_SIZE] = {-888, 3000, -500, 7000, 1000, -123, -800, 9000, 789, -444}

  Values of V_b to test with.
- const int Vc [TEST_SIZE] = { 100, -300, -100, 1000, -100, -500, 800, 1000, 189, -744}

  Values of V_c to test with.

4.5.1 Macro Definition Documentation

4.5.1.1 TEST_SIZE

```
#define TEST_SIZE 10
```

Number of values to test with.

Definition at line 14 of file test_svpwm.cpp.

4.5.2 Function Documentation

4.5.2.1 main()

```
int main ( )
```

Main function of the C testbench.

The function SVPWM() will be called with the values of V_a, V_b and V_c in Va, Vb and Vc and the results will be printed along with separately calculated values.

Definition at line 32 of file test_svpwm.cpp.

```
32          {
33     int i;
34     hls::stream<int64_t> inputStream;
35     hls::stream<int64_t> outputStream;
36     int64_t tx_data, rx_data;
37     int16_t ia, ib, ic, minv, maxv;
38     int32_t voff, ea, eb, ec;
39
40     // tx_data = (3000 << 16) | 2000;
41     // inputStream << tx_data;
42     for(i=0; i<TEST_SIZE; i++){
43         tx_data = ((int64_t(Vc[i]) << 32) & 0xFFFF00000000) | ((int64_t(Vb[i]) << 16) & 0xFFFF0000) | (
       int64_t(Va[i]) & 0x0000FFFF);
44         inputStream << tx_data;
45
46         SVPWM(inputStream, outputStream);
47         outputStream.read(rx_data);
48         ia = int16_t(rx_data & 0xFFFF);
49         ib = int16_t((rx_data >> 16) & 0xFFFF);
50         ic = int16_t((rx_data >> 32) & 0xFFFF);
51         // Voff= [min(Va, Vb, Vc)+max(Va, Vb, Vc)]/2
52         // Vanew = Va - Voff
53         // Vbnew = Vb - Voff
54         // Vcnew = Vc - Voff
55         minv = (Va[i] > Vb[i]) ? Vb[i] : Va[i];
56         minv = (minv > Vc[i]) ? Vc[i] : minv;
57         maxv = (Va[i] > Vb[i]) ? Va[i] : Vb[i];
58         maxv = (Vc[i] > maxv) ? Vc[i] : maxv;
59         voff = (minv + maxv) / 2;
60         ea = Va[i] - voff;
61         eb = Vb[i] - voff;
62         ec = Vc[i] - voff;
63         printf("Values is Ia=%d Ib=%d Ic=%d (%d, %d, %d)\n",ia ,ib ,ic, ea, eb, ec);
64     }
65 }
```

4.5.3 Variable Documentation

4.5.3.1 Va

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

Values of V_a to test with.

Definition at line 17 of file test_svpwm.cpp.

4.5.3.2 Vb

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

Values of V_b to test with.

Definition at line 20 of file test_svpwm.cpp.

4.5.3.3 Vc

```
const int Vc[TEST_SIZE] = { 100, -300, -100, 1000, -100, -500, 800, 1000, 189, -744}
```

Values of V_c to test with.

Definition at line 23 of file test_svpwm.cpp.

Index