HLS: Filters

1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Introduction

1.1 Function

This IP core, implemented in the form of a C function with Vivado HLS, realizes the input values filtering 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, speed (in RPM) and rotor angle to the output AXI4-Stream, consisting of filtered values of corresponding inputs.

For the speed, a simple boxcar filter of length 32 is used, which implements the following equation:

$$speed'(t) = \frac{1}{32} \sum_{i=t-31}^{t} speed(t). \tag{1.1}$$

For the phase currents, a cascade of two IIR filtes are used, both stages of which implement the following equation:

$$I'_{a,b}(t) = BI_{a,b}(t) + (1 - B)I_{a,b}(t - 1), \tag{1.2}$$

where B is the feed forward coefficient. In addition, there is a DC component filtering block, which averages current values when motor is off and subtracts the averaged values from the phase currents in all other modes.

The rotor angle value is passed unprocessed to the output stream.

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 'Filters' 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 Filters(
    hls::stream<int64_t> &s_axis,
    hls::stream<int64_t> &m_axis,
    int16_t *RPM_out,
    const int32_t control);
```

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

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 'Filters'

Date:	Fri Jun 16 13:13:48 2017
Version:	2017.1 (Build 1846317 on Fri Apr 14 19:19:38 MDT 2017)
Project:	Filters
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	9.82	1.25

Latency (clock cycles)

Table 2.3 Summary

Latency		Interval		Pipeline
min	max	min	max	Type
69	69	70	70	none

Detail

Instance: N/A

Table 2.4 Loop

	Latency		Iteration	Initiation Interval		Trip	
Loop Name	min	max	Latency	achieved	target	Count	Pipelined
- Loop 1	62	62	2	-	-	31	no

Utilization Estimates

Table 2.5 Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	8	-	-
Expression	-	-	0	285
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	32	8
Multiplexer	-	-	-	200
Register	-	-	988	-
Total	0	8	1020	493
Available	280	220	106400	53200
Utilization (%)	0	3	∼0	∼0

Detail

Instance: N/A

Table 2.6 DSP48

Instance	Module	Expression
Filters_mac_muladcud_U2	Filters_mac_muladcud	i0 + i1 ∗ i2
Filters_mac_muladcud_U3	Filters_mac_muladcud	i0 + i1 ∗ i2
Filters_mac_muladeOg_U6	Filters_mac_muladeOg	i0 + i1 ∗ i2
Filters_mac_muladeOg_U7	Filters_mac_muladeOg	i0 + i1 ∗ i2
Filters_mul_mul_1bkb_U0	Filters_mul_mul_1bkb	i0 ∗ i1
Filters_mul_mul_1bkb_U1	Filters_mul_mul_1bkb	i0 ∗ i1
Filters_mul_mul_1dEe_U4	Filters_mul_mul_1dEe	i0 ∗ i1
Filters_mul_mul_1dEe_U5	Filters_mul_mul_1dEe	i0 ∗ i1

Table 2.7 Memory

Memory	Module	BRAM_18K	FF	LUT	Words	Bits	Banks	W∗Bits∗Banks
filt_mem↩_U	Filters_filt_mem	0	32	8	32	16	1	512
Total		0	32	8	32	16	1	512

FIFO: N/A

Table 2.8 Expression

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
i_1_fu_255_p2	+	0	0	15	5	2
tmp_20_fu_414_p2	+	0	0	39	32	32
tmp_21_fu_420_p2	+	0	0	39	32	32
tmp_22_fu_480_p2	+	0	0	39	32	1
tmp_6_fu_233_p2	+	0	0	39	32	32
Ia_f_fu_534_p2	-	0	0	23	16	16
Ib_f_fu_547_p2	-	0	0	23	16	16
tmp_5_fu_223_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
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_17_fu_400_p2	icmp	0	0	16	32	15
tmp_23_fu_492_p2	icmp	0	0	16	32	1
tmp_7_fu_249_p2	icmp	0	0	2	5	1
Total		0	0	285	259	171

Table 2.9 Multiplexer

Name	LUT	Input Size	Bits	Total Bits
Ia_DC_acc	9	2	32	64
Ib_DC_acc	9	2	32	64
ap_NS_fsm	47	10	1	10
filt_mem_address0	27	5	5	25
filt_mem_d0	15	3	16	48
i_reg_155	9	2	5	10
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
storemerge_phi_fu_170_p4	9	2	32	64
storemerge_reg_166	9	2	32	64
Total	200	42	289	621

Table 2.10 Register

Name	FF	LUT	Bits	Const Bits
Ia2_filtered_cast_reg_699	16	0	16	0
Ia_DC_acc	32	0	32	0
Ia_DC_val	17	0	17	0
Ia_corr	17	0	17	0
Ia_reg_626	16	0	16	0
Ib2_filtered_cast_reg_704	16	0	16	0
Ib_DC_acc	32	0	32	0

Name	FF	LUT	Bits	Const Bits
Ib_DC_val	17	0	17	0
Ib_corr	17	0	17	0
Ib_reg_631	16	0	16	0
RPM_reg_636	16	0	16	0
Theta_reg_641	16	0	16	0
Y1a_prev	17	0	17	0
Y1b_prev	17	0	17	0
Y2a_prev	17	0	17	0
Y2b_prev	17	0	17	0
ap_CS_fsm	9	0	9	0
ap_reg_ioackin_RPM_out_dummy_ack	1	0	1	0
dc_cnt	32	0	32	0
dc_cnt_load_reg_709	32	0	32	0
filt_acc	32	0	32	0
i_1_reg_659	5	0	5	0
i_cast3_reg_651	5	0	32	27
i_reg_155	5	0	5	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
storemerge_reg_166	32	0	32	0
tmp_10_reg_684	17	0	17	0
tmp_12_reg_689	32	0	32	0
tmp_14_reg_694	32	0	32	0
tmp_17_reg_714	1	0	1	0
tmp_20_reg_718	32	0	32	0
tmp_21_reg_723	32	0	32	0
tmp_24_reg_731	16	0	16	0
tmp_2_reg_674	32	0	32	0
tmp_4_reg_679	17	0	17	0
tmp_6_reg_646	32	0	32	0
tmp_s_reg_669	32	0	32	0
Total	988	0	1015	27

Interface

Table 2.11 Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	Filters	return value
ap_rst_n	in	1	ap_ctrl_hs	Filters	return value

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_start	in	1	ap_ctrl_hs	Filters	return value
ap_done	out	1	ap_ctrl_hs	Filters	return value
ap_idle	out	1	ap_ctrl_hs	Filters	return value
ap_ready	out	1	ap_ctrl_hs	Filters	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
RPM_out	out	16	ap_vld	RPM_out	pointer
RPM_out_ap_vld	out	1	ap_vld	RPM_out	pointer
control	in	32	ap_none	control	scalar

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/Filters_csynth.dox File Reference

4.2 doxygen/src/main_page.dox File Reference

4.3 filters.cpp File Reference

Implementation of the function Filters().

```
#include "filters.h"
```

Functions

- void Filters (hls::stream< int64_t > &s_axis, hls::stream< int64_t > &m_axis, int16_t *RPM_out, const int32_t control)

  Filters Core.

4.3.1 Detailed Description

Implementation of the function Filters().

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

4.3.2 Function Documentation

4.3.2.1 Filters()

```
void Filters (
            hls::stream< int64_t > & s_axis,
            hls::stream< int64_t > & m_axis,
            int16_t * RPM_out,
            const int32_t control )
```

Filters Core.

Parameters

s_axis	Input AXI4-Stream with the following format: • 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. • Bits 48..63: Angle, in encoder steps, just passed through. All values are 16-bit signed integers.
m_axis	Output AXI4-Stream, contains filtered values of the input stream in the same format.
RPM_out	Returns speed, in RPM.
control	Value of the control register. Zero value indicates that the motor is off.

Returns

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

Definition at line 18 of file filters.cpp.

```
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 Ia_f, Ib_f, RPM_f;
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     // Simple average filter for RPM
34     static int filt_acc;
35     static int filt_mem[FILTER_ORDER];
36     filt_acc -= filt_mem[FILTER_ORDER-1];
37     filt_acc += RPM;
38     for(int i = (FILTER_ORDER-1); i > 0; i--){
39         filt_mem[i] = filt_mem[i - 1];
40     }
41     filt_mem[0] = RPM;
42     RPM_f = filt_acc >> FILTER_SHIFT;
```

```

    // IIR lowpass filters for Ia and Ib (2 stages)
    // First filter stage
    int Ia1_filtered, Ib1_filtered;
    int Y1a, Y1b;
    static int Y1a_prev;
    static int Y1b_prev;

    Y1a = Ia*FILT_B + Y1a_prev*FILT_A;
    Y1b = Ib*FILT_B + Y1b_prev*FILT_A;
    Ia1_filtered    = Y1a >> 15;
    Ib1_filtered    = Y1b >> 15;
    Y1a_prev = Ia1_filtered;
    Y1b_prev = Ib1_filtered;

    // Second filter stage
    int Ia2_filtered, Ib2_filtered;
    int Y2a, Y2b;
    static int Y2a_prev;
    static int Y2b_prev;

    Y2a = Ia1_filtered*FILT_B + Y2a_prev*FILT_A;
    Y2b = Ib1_filtered*FILT_B + Y2b_prev*FILT_A;
    Ia2_filtered    = Y2a >> 15;
    Ib2_filtered    = Y2b >> 15;
    Y2a_prev = Ia2_filtered;
    Y2b_prev = Ib2_filtered;

    // Calculate DC in Idle mode
    static int Ia_DC_acc = 0;
    static int Ib_DC_acc = 0;
    static int Ia_DC_val = 0;
    static int Ib_DC_val = 0;
    static int dc_cnt = 0;
    static int Ia_corr, Ib_corr;

    if(dc_cnt >= (DC_ACC_SAMPLES-1)){ // End of accomulation
        Ia_DC_val   = Ia_DC_acc >> DC_ACC_BITS;
        Ib_DC_val   = Ib_DC_acc >> DC_ACC_BITS;
        Ia_DC_acc   = Ia2_filtered;
        Ib_DC_acc   = Ib2_filtered;
        dc_cnt      = 0;
    }
    else{                               // Accomulation
        Ia_DC_acc   = Ia_DC_acc + Ia2_filtered;
        Ib_DC_acc   = Ib_DC_acc + Ib2_filtered;
        dc_cnt++;
    }
    if(control == 0){           // Save DC in Idle mode
        Ia_corr = Ia_DC_val;
        Ib_corr = Ib_DC_val;
    }

    // Apply DC correction
    Ia_f = Ia2_filtered - Ia_corr;
    Ib_f = Ib2_filtered - Ib_corr;

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

4.4 filters.h File Reference

Header file for Filters.

```
#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 FILTER_SHIFT 5

    Boxcar filter order for the speed filter.
- #define FILTER_ORDER (1 << FILTER_SHIFT)

    This is automatically determined from FILTER_SHIFT.
- #define DC_ACC_BITS 15

    Order of the DC averaging filter.
- #define DC_ACC_SAMPLES (1 << DC_ACC_BITS)

    This is automatically determined from the DC_ACC_BITS.
- #define FILT_B 18120

    Feedforward filter coefficient for the current filter.
- #define FILT_A (MAX_LIM - FILT_B)

    Feedback filter coefficient for the current filter.

Functions

- void Filters (hls::stream< int64_t > &s_axis, hls::stream< int64_t > &m_axis, int16_t *RPM_out, const int32_t control)

    Filters Core.

4.4.1 Detailed Description

Header file for Filters.

Author

Oleksandr Kiyenko

Version

1.0

Date

2017

Copyright

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

4.4.2 Macro Definition Documentation

4.4.2.1 DC_ACC_BITS

```
#define DC_ACC_BITS 15
```

Order of the DC averaging filter.

Order of 15 corresponds to 32768 samples averaged.

Definition at line 34 of file filters.h.

4.4.2.2 DC_ACC_SAMPLES

```
#define DC_ACC_SAMPLES (1 << DC_ACC_BITS)
```

This is automatically determined from the DC_ACC_BITS.

Definition at line 37 of file filters.h.

4.4.2.3 FILT_A

```
#define FILT_A (MAX_LIM - FILT_B)
```

Feedback filter coefficient for the current filter.

This is derived from FILT_B such that the filter is not amplifying.

Definition at line 48 of file filters.h.

4.4.2.4 FILT_B

```
#define FILT_B 18120
```

Feedforward filter coefficient for the current filter.

Choose a number between 1 and 32766. Higher values correspond to less filtering and lower values correspond to more filtering.

Note: the number 0.553 converted to Q16.16 is 18120.

Definition at line 44 of file filters.h.

4.4.2.5 FILTER_ORDER

```
#define FILTER_ORDER (1 << FILTER_SHIFT)
```

This is automatically determined from FILTER_SHIFT.

Definition at line 30 of file filters.h.

4.4.2.6 FILTER_SHIFT

```
#define FILTER_SHIFT 5
```

Boxcar filter order for the speed filter.

Order of 5 corresponds to 32 samples buffered in the boxcar filter.

Definition at line 27 of file filters.h.

4.4.2.7 MAX_LIM

```
#define MAX_LIM 32767
```

Maximum positive value for saturated arithmetic.

Definition at line 20 of file filters.h.

4.4.2.8 MIN_LIM

```
#define MIN_LIM -32767
```

Minimum negative value for saturated arithmetic.

Definition at line 23 of file filters.h.

4.4.3 Function Documentation

4.4.3.1 Filters()

```
void Filters (
            hls::stream< int64_t > & s_axis,
            hls::stream< int64_t > & m_axis,
            int16_t * RPM_out,
            const int32_t control )
```

Filters Core.

Parameters

s_axis	Input AXI4-Stream with the following format: • 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. • Bits 48..63: Angle, in encoder steps, just passed through. All values are 16-bit signed integers.
m_axis	Output AXI4-Stream, contains filtered values of the input stream in the same format.
RPM_out	Returns speed, in RPM.
control	Value of the control register. Zero value indicates that the motor is off.

Returns

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

Definition at line 18 of file filters.cpp.

```
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 Ia_f, Ib_f, RPM_f;
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     // Simple average filter for RPM
34     static int filt_acc;
35     static int filt_mem[FILTER_ORDER];
36     filt_acc -= filt_mem[FILTER_ORDER-1];
37     filt_acc += RPM;
38     for(int i = (FILTER_ORDER-1); i > 0; i--){
39         filt_mem[i] = filt_mem[i - 1];
40     }
41     filt_mem[0] = RPM;
42     RPM_f = filt_acc >> FILTER_SHIFT;
43
44     // IIR lowpass filters for Ia and Ib (2 stages)
45     // First filter stage
46     int Ia1_filtered, Ib1_filtered;
47     int Y1a, Y1b;
48     static int Y1a_prev;
49     static int Y1b_prev;
50
51     Y1a = Ia*FILT_B + Y1a_prev*FILT_A;
52     Y1b = Ib*FILT_B + Y1b_prev*FILT_A;
53     Ia1_filtered    = Y1a >> 15;
54     Ib1_filtered    = Y1b >> 15;
55     Y1a_prev = Ia1_filtered;
56     Y1b_prev = Ib1_filtered;
57
58     // Second filter stage
59     int Ia2_filtered, Ib2_filtered;
60     int Y2a, Y2b;
61     static int Y2a_prev;
62     static int Y2b_prev;
63
64     Y2a = Ia1_filtered*FILT_B + Y2a_prev*FILT_A;
65     Y2b = Ib1_filtered*FILT_B + Y2b_prev*FILT_A;
66     Ia2_filtered    = Y2a >> 15;
67     Ib2_filtered    = Y2b >> 15;
```

```
   Y2a_prev = Ia2_filtered;
   Y2b_prev = Ib2_filtered;

   // Calculate DC in Idle mode
   static int Ia_DC_acc = 0;
   static int Ib_DC_acc = 0;
   static int Ia_DC_val = 0;
   static int Ib_DC_val = 0;
   static int dc_cnt = 0;
   static int Ia_corr, Ib_corr;

   if(dc_cnt >= (DC_ACC_SAMPLES-1)){ // End of accomulation
       Ia_DC_val   = Ia_DC_acc >> DC_ACC_BITS;
       Ib_DC_val   = Ib_DC_acc >> DC_ACC_BITS;
       Ia_DC_acc   = Ia2_filtered;
       Ib_DC_acc   = Ib2_filtered;
       dc_cnt      = 0;
   }
   else{                               // Accomulation
       Ia_DC_acc   = Ia_DC_acc + Ia2_filtered;
       Ib_DC_acc   = Ib_DC_acc + Ib2_filtered;
       dc_cnt++;
   }
   if(control == 0){           // Save DC in Idle mode
       Ia_corr = Ia_DC_val;
       Ib_corr = Ib_DC_val;
   }

   // Apply DC correction
   Ia_f = Ia2_filtered - Ia_corr;
   Ib_f = Ib2_filtered - Ib_corr;

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

Index