SPADIC 1.0 Data Sheet

Version 0.30
Mai 22, 2014

Contents

1 Physics requirements

The table below shows the required characteristics of the different subdetector types that are at least somehow related to certain SPADIC 1.0 parameters. The actual SPADIC 1.0 characteristics can be found further below in the subsequent sections.

	TRD	RICH
channels/chip	32	opt. 64 (32 ok)
channels/subdectector	1 M	$860 \times 64 = 55\,\mathrm{k}$
chips/subdetector	30 k	
power limit/channel	50 mW	threshold dispersion
noise limit		
maximum radiation dose/chip		
maximum input capacity	40 pF	compare Hamamatsu H8500
average input capacity	20 pF	
average hit rate/channel	100 kHz	
maximum hit rate/channel	300 kHz	200 kHz
required shaping time	80 ns	
maximum leakage current/channel		none
minimum charge/hit		
average charge/hit	$120\,\mathrm{ke}^- = 19.2\,\mathrm{fC}$	$10^6\,\mathrm{e}^- = 160.2\,\mathrm{fC}$
maximum charge/hit		$5\text{–}10 \times 10^6\,\mathrm{e}^- = 800\text{–}1600\,\mathrm{fC}$
required dynamic range	60–80 fC	800–2000 fC
charge polarity	positive	negative
type of energy distribution	exponential	gauss
measured quality	spatial res.	binary readout
input signal shape		pulse (< 10 ns)
required energy resolution	8 bit	8 bit
required time resolution		target 2 ns (5 ns max.) sigma
special tasks	baseline rec., ion tail can.	feature extraction (time, energy)

2 SPADIC 1.0 documentation

2.1 General

Parameter	Value
Technology	UMC 180 nm
Number of metal layers	6
Number of channels	32
Channel pitch	120 µm
Pad pitch	95 µm
Pad size	65 µm × 65 µm
Chip size	4.96 mm × 4.96 mm (3 × 3 Mini@sics)
Total power consumption	

	Analog Part	Digital Part	Total
Number of transistors	226 k	2.09 M	2.32 M
Number of nets	58.1 k	1.02 M	1.08 M
Number of pads			191
Number of DACs	48		48
Number of sram blocks		44	44
Total SRAM size		4.4 kB	4.4 kB
Total config register size	584 bit	1270 bit	1854 bit
Total wire length		14.39 m	
Number of FF		23152	
Number of FA		12252	
Number of MUX		13146	
Number of buffers		7380	
Number of gates		81337	
Area	1.12 mm × 4.65 mm	3.46 mm × 4.54 mm	4.96 mm × 4.96 mm

2.2 Analog part

2.2.1 Positive front end

Parameter	Design-/Sim-Value	Meas-Value	Comment
Peaking time (0–100%)	86.7 ns @ 100 ke$^-$		
Input type	N-MOS		
Polarity	positive		
Dynamic Range	75 fC = 468.1 ke$^-$		
Power consumption	3.8 mW		
Size of layout	440 µm × 60 µm		
Number of amplifier cells	12		
Noise (@ 100 ke$^-$)	387 e$^-$ + 11 e$^-$/pF		
Shaping time	80 ns		
Order of shaper	1		
Order of complete CSA	2		
Pulse shape	$t/\tau \exp(-t/\tau)$		

2.2.2 Negative front end

Parameter	Design-/Sim-Value	Meas-Value	Comment
Peaking time (0-100%)	97 ns @ 100 ke$^-$		
Rise time (10-90%)			
Input type	P-MOS		
Polarity	negative		
Dynamic Range	75 fC = 486.1 ke$^-$		
Power consumption	10 mW		not optimized yet
Size of layout	440 µm × 60 µm		
Number of amplifier cells	13		
Noise (@ 100 ke$^-$)	439 e$^-$ + 11 el/pF		
Shaping time	80 ns		
Order of shaper	1		
Order of complete CSA	2		
Pulse shape	$t/\tau \exp(-t/\tau)$		

2.2.3 ADC

Parameter	Design-Value	Meas-Value	Comment
Nominal sampling frequency	25 MHz		
Maximum sampling frequency			
Power consumption		≈ 5.5 mW	first guess
Resolution, nominal	9 bit		
Resolution, effective		≤ 8 bit	
Size of layout (core)	140 µm × 120 µm		
Size of layout (complete)	410 µm × 120 µm		input cell and decoupling included
Radiation hardness	yes		Rad. hard layout

2.3 Digital part

2.3.1 General

Parameter	Value	Comment
Input clock	250 MHz	LVDS input
Internal clock domains	250, 125, and 25 MHz	CLK1, CLK2, CLK10
Power Consumption	682 mW	From Velocity 250MHz
Power Consumption	594 mW	Measured at 200MHz

2.3.2 IIR filter

Parameter	Value	Comment
Order	4 zeros, 3 poles	
Internal resolution	16 bit	
Coefficient width	6 bit	
Coefficient range	$-1 \ldots 0.96875$	$\frac{-32}{32} \ldots \frac{31}{32}$
Scaling width	9 bit	
Scaling range	$-8 \ldots 7.96875$	$\frac{-256}{32} \ldots \frac{255}{32}$
Offset width	9 bit	
Offset range	$-256 \ldots 255$	
Number of multiply-accumulate units	8/channel	$1 + 3 \times 2$ (4 stages) + 1 (offset/scaling)

2.3.3 Digital hit detector

Parameter	Value	Comment

2.3.4 Timestamping

Parameter	Value	Comment

2.4 Output interface

Parameter	Value	Comment

3 Configuration

3.1 Analog part

The analog shift register has a length of 584 bit and goes through several DACs as well as many configuration registers. The shift in/shift out mechanism is controlled by a module of the digital part.

Signal	Default Voltage	Default DAC Value	Comment
nCascN	1.0 V	51	
pCascN	700 mV	52	
nSourceBiasN	1.27 V	49	
pSourceBiasN	982 mV	50	
pFBN	264 mV	60	
nCascP	1.1 V	51	
pCascP	800 mV	52	
nSourceBiasP	607 mV	51	
pSourceBiasP	618 mV	55	
nFBP	1.24 V	50	

3.2 Digital part

Note: This subsection describes the configuration registers of the digital part except for the CBMnet module. For information on the CBMnet configuration as well as the exact addresses of the registers given below see register file documentation.

Parameter	Bits	Description	Comment
REG_aCoeffFilter	17:0	a coefficients of IIR filter	see 2.3.2
REG_bCoeffFilter	23:0	b coefficients of IIR filter	see 2.3.2
REG_bypassFilterStage	4:0	bypass IIR filter stages	see 2.3.2
REG_cbmNetAddr	15:0	target address for all packages	
REG_compDiffMode	0	enable diff mode of hit detector	default is true
REG_disableChannelA	15:0	disable channels 0-15 (group A)	channel 0 = bit 0
REG_disableChannelB	15:0	disable channels 16-31 (group B)	channel 16 = bit 0
REG_disableEpochChannelA	0	disable epoch channel (group A)	en/disable periodic epoch messages
REG_disableEpochChannelB	0	disable epoch channel (group B)	en/disable periodic epoch messages
REG_enableAdcDec	20:0	set analog decoupling/select analog signals	signals go directly to analog switches of analog part
REG_enableAnalogTrigger	0	enable signal for analog trigger signal	goes directly to analog switch of analog part
REG_enableTestInput	0	connect digital test input to channel 0	inject 2's complement to channel 0
REG_enableTestOutput	0	connect output of currently selected group to test output	miscOut/In pins used
REG_enableTriggerOutput	0	enable multi purpose trigger output	multi purpose trigger is set via dlm??
REG_epochCounter	15:0	next epoch value to be set, only 12:0 used	
REG_groupIdA	7:0	group id of group A	used in most messages
REG_groupIdB	7:0	group id of group B	used in most messages
REG_hitWindowLength	5:0	define length of single pulse	further hits within this window are threated as multi hits
REG_neighborSelectMatrixA	483:0	configure neighbor trigger scheme (group A)	see TODO
REG_neighborSelectMatrixB	483:0	configure neighbor trigger scheme (group B)	see TODO
REG_offsetFilter	8:0	offset unit of IIR filter	see 2.3.2
REG_scalingFilter	8:0	scaling unit of IIR filter	see 2.3.2
REG_selectMask	31:0	mask to select values of a pulse	only selected values are stored in the hit message
REG_testOutputSelGroup	0	select which group shall be connected to test output	0: group A, 1: group B
REG_threshold1	8:0	first threshold of hit detector	2's complement
REG_threshold2	8:0	second threshold of hit detector	2's complement
REG_triggerMaskA	15:0	channel select mask for external trigger (group A)	external trigger arrives via dlm??
REG_triggerMaskB	15:0	channel select mask for external trigger (group B)	external trigger arrives via dlm??

4 Output message format

4.1 General description

In this context the term message is used to describe the merged hit information which consists of the ADC raw data and/or the hit metadata (such as timestamp, hit type, etc.). The term package in contrast is used to describe a data bundle generated by the CBMnet output interface which contains a cutout of the message data stream framed by different kind of transport information (e.g. CRC, routing, ...).

Each message consists of a variable number of 16 bit data words. The type of each data word can clearly be identified by checking the preamble bits—see coding below. This way it is assured that one can resynchronize a stream of messages even if some bits are corrupted or parts of the data stream got lost.

When running the ASIC, a continuous stream of messages is generated in each channel. An inter-channel arbitration logic is used to merge the data message streams of multiple channels. Due to the arbitration method the messages in the resulting stream are guaranteed to be sorted in the order in which they were generated. For hit messages (see 4.2.3) this means also that they are sorted by their timestamps.

4.2 Message specification

4.2.1 Word types

Every 16-bit word in the data stream has a *type*, which is encoded in the first few bits of the word (the *preamble*). Depending on the type of a word, the remaining bits after the preamble have a different meaning. The different word types are summarized in the following table (dots (`.`) indicate the bits that are taken by the preamble, dashes (`-`) indicate bits that have no meaning in a message of a given type):

Preamble	Type	Description	Format	Content
`1000`	`SOM`	start of message	`.... gggg gggg cccc`	`g`: group ID, `c`: channel ID
`1001`	`TSW`	timestamp word	`.... tttt tttt tttt`	`t`: timestamp
`1010`	`RDA`	begin of raw data	`.... dddd dddd dddd`	`d`: raw data
`0`	`CON`	continued raw data	`.ddd dddd dddd dddd`	`d`: raw data
`1011`	`EOM`	end of message	`.... nnnn nnhh -sss`	`n`: number of samples contained in raw data block, `h`: hit type, `s`: stop type
`1100`	`BOM`	buffer overflow	`.... ---- bbbb bbbb`	`b`: number of hits lost due to buffer overflow
`1101`	`EPM`	epoch marker	`.... eeee eeee eeee`	`e`: epoch counter
`1110`	`EXD`	extracted data[1]	`.... xxxx xxxx xxxx`	`x`: extracted data
`1111`	`INF`	info word	`.... iiii aaaa aaaa`	`i`: info type, `a`: additional data

4.2.2 Message structure

Several words in succession form a *message*. Four kinds of messages can be distinguished:

- hit messages (described in 4.2.3)
- buffer overflow messages (described in 4.2.4)
- epoch markers (described in 4.2.5)
- info messages (described in 4.2.6)

[1] not implemented in the current version of the chip

The number of words that a message comprises depends on the kind of message and on the content of the message. In order to be able to split a sequence of words into individual messages, the beginning and end of a message is marked by certain word types:

Hit messages, buffer overflow messages and epoch markers each begin with an `SOM` word and end with an `EOM`, `BOM`, or `EPM` word, respectively. In exceptional cases, they can end with an `INF` word.

Info messages begin with an `INF` word and end with the *same* word, i. e. they consist of exactly this one word. *(But not all `INF` words form info messages on their own!)*

Algorithm In the following, a simple algorithm is described which outputs a sequence of messages for a given sequence of words. It uses a buffer, which can hold any number of words, as temporary storage for a message being built up. At the beginning of the algorithm, the buffer is assumed empty.

1. Read the next word, if there is one left, else finish.
2. If it is an `INF` word with the `NOP` info type,[1] repeat from step 1.
3. If it is an `SOM` word, or an `INF` word with any of the `NGT`, `NRT`, or `NBE` info types, clear the buffer.
4. Append the word to the buffer.
5. If it is an `EOM`, `BOM`, `EPM`, or `INF` word, output the contents of the buffer, then clear the buffer.
6. Repeat from step 1.

4.2.3 Hit messages

Hit messages begin with an `SOM` word, followed by a `TSW` word, an `RDA` word, any number (including zero) of `CON` words, and finally an `EOM` word:

Word type	Format	Content
`SOM`	`1000 gggg gggg cccc`	`g`: group ID, `c`: channel ID
`TSW`	`1001 tttt tttt tttt`	`t`: timestamp
`RDA`	`1010 dddd dddd dddd`	`d`: raw data
`CON`	`0ddd dddd dddd dddd`	`d`: raw data
⋮	⋮	⋮
`CON`	`0ddd dddd dddd dddd`	`d`: raw data
`EOM`	`1011 nnnn nnhh -sss`	`n`: number of samples contained in raw data block, `h`: hit type, `s`: stop type

In case the channel is disabled while building a hit message, it is aborted with an `INF` word of type `DIS`. In case of data corruption inside the message builder, the message is aborted with an `INF` word of type `MSB`.

The data that is carried in a hit message consists primarily of a number of 9-bit ADC samples (coded in 2's complement), and the following metadata:

- the 8-bit group ID (`g`)
- the 4-bit channel ID (`c`)
- the 12-bit timestamp (`t`)
- the 6-bit number of contained ADC samples (`n`)
- the 2-bit hit type (`h`)

[1] see 4.2.6

- the 3-bit stop type (`s`)

ADC samples In order to retrieve the ADC samples, the `RDA` and all `CON` words must be joined together, excluding the preamble bits. The resulting raw data block (all the `d` bits) is then simply the concatenation of all the 9-bit values, plus a number of zero bits that fill up the leftover space of the last `CON` word.

In most cases, the information from the `EOM` word (the "`n`" bits) must be used to determine the number of ADC samples contained in the message, because the last `CON` word was filled with 9 or more zero bits:

number of `CON` words	0	1	2	3	4	5	6	7	8	9
possible number of ADC samples	1	2, 3	4	5, 6	7, 8	9	10, 11	12, 13	14	15, 16

For example, in a message with 7 `CON` words, there are $12 + 7 \times 15 = 117 = 13 \times 9$ bits in the data block, which means there are either 12 or 13 ADC samples contained. If all of the 13 9-bit sections were interpreted as an ADC sample when there are only 12 samples in the message, one would get an additional false zero value.

Hit types The *hit type*, that is stored in the "`h`" bits of the `EOM` word, tells about the reason why the hit message was generated. Any channel has three sources of input that causes it to generate a hit message. They are coded as follows:

Hit type	Description
`00`	triggered by global signal (DLM)
`01`	self-triggered
`10`	triggered by neighbor channel
`11`	both of the above simultaneously

Stop types The *stop type*, that is stored in the "`s`" bits of the `EOM` word, tells about the circumstances under which the message building process was finished, i. e. why there are no more `CON` words. Normally, the hit message simply ends because all ADC samples that belong to the hit have been included. If this was not possible, the stop type explains why:

Stop type	Description
`000`	normal end of the hit message
`001`	aborted because the message output buffer was full
`010`	aborted because the ordering FIFO was full
`011`	multi hit: the next hit message was triggered before the current one was finished
`100`	output buffer was full, multi hit detected simultaneously
`101`	ordering FIFO was full, multi hit detected simultaneously

Example hit message To illustrate the description of the hit message format, an example follows:

Data stream	Word type	Content
`1000000001110010`	`SOM`	group ID: 7, channel ID: 2
`1001001100101010`	`TSW`	timestamp: 810
`1010111111101000`	`RDA`	
`0000010111111100`	`CON`	
`0000000010000011`	`CON`	
`0111000000000000`	`CON`	
`1011000101010001`	`EOM`	number of ADC samples: 5, hit type: self-triggered, stop type: output buffer was full

The ADC samples are contained in the following block of the data stream:

.....	SOM
.....	TSW
....111111101000	RDA
.000010111111100	CON
.000000010000011	CON
.111000000000000	CON
.....	EOM

→ rearranged as 9-bit sections →

Binary	2's complement value
111111101	−3
000000010	2
111111100	−4
000000010	2
000011111	31
000000000	no ADC sample
000	(only 5 contained)

4.2.4 Buffer overflow messages

When the message output buffer of a channel or the ordering FIFO of its group is full and therefore no messages can be sent out, the channel still reacts to incoming hit signals (self-triggered, neighbor-triggered, or globally triggered by DLM) by incrementing a buffer overflow counter. As soon as the channel can send messages again, a *buffer overflow message* is generated which contains the value of the buffer overflow counter, i. e. the number of hits that could not be sent as hit messages.

The buffer overflow messages also contain a timestamp that corresponds to the most recent hit that was missed. Since the buffer overflow messages are generated and sent out delayed with respect to this timestamp, they may appear later in the data stream than messages of *different* channels with a more advanced timestamp.

Buffer overflow messages consist of an `SOM`, a `TSW`, and a `BOM` word:

Word type	Format	Content
SOM	1000 gggg gggg cccc	g: group ID, c: channel ID
TSW	1001 tttt tttt tttt	t: timestamp
BOM	1100 ---- bbbb bbbb	b: number of hits lost due to buffer overflow

4.2.5 Epoch markers

Apart from the regular channels that send out hit messages and buffer overflow messages, each group has an additional *epoch channel* that sends out *epoch markers* whenever the globally distributed timestamp wraps around. [2]

Epoch markers are inserted into the message stream in the same way as the messages from all the other channels and normally consist of two words:

Word type	Format	Content
SOM	1000 gggg gggg 0000	g: group ID, channel ID fixed to 0
EPM	1101 eeee eeee eeee	e: epoch number

Since the timestamps are 12 bits wide, at 25 MHz clock frequency (40 ns period), an epoch marker is sent every 163.84 µs. The epoch counter is also 12 bits wide and wraps around every 671.1 ms.

But if DLM1 occurs asynchronously to the time-stamp wrap-around, instead an epoch marker out of sync of the message is generated instead:

Word type	Format	Content
SOM	1000 gggg gggg 0000	g: group ID, channel ID fixed to 0
INF	1111 0110 eeee eeee	e: last 8 bit of epoch number

[2] Also, DLM1 needs to be sent to the chip at the right time.

4.2.6 Info messages

At various positions, in individual channels as well as the channel groups, special *info words* can be inserted, which notify about unusual conditions. They contain the *info type* and, for some info types, additional data:

Word type	Format	Content
`INF`	`1111 iiii aaaa aaaa`	`i`: info type, `a`: additional data, depends on info type
Info type	**Format**	**Description**
`DIS`	`.... 0000 cccc ----`	channel disabled during readout, `c`: channel ID
`NGT`	`.... 0001 cccc ----`	corruption, next grant timeout in switch, `c`: channel ID
`NRT`	`.... 0010 ---- ----`	corruption, next request timeout in switch
`NBE`	`.... 0011 cccc ----`	corruption, new grant but channel empty in switch, `c`: channel ID
`MSB`	`.... 0100 cccc ----`	corruption, unknown metadata type in message builder, `c`: channel ID
`NOP`	`.... 0101 ---- ----`	empty word (may appear within other messages)
`SYN`	`.... 0110 eeee eeee`	out of sync warning, `e`: least significant bits of epoch counter

All info words mark the end of a message. The `NGT`, `NRT`, `NBE` type info words form single-word *info messages* on their own and thus mark the start and end of a message at the same time.

5 DLMs (Deterministic Latency Messages)

From each node to each node of the CBMnet so called deterministic latency messages (DLM) can be sent. Hereby the idea is that the CBMnet protocol always guarantees for a constant transmission delay between two nodes which moreover can be measured exactly with special DLMs (DLM0). This way one can exactly (in terms of clock cycles) calculate when a certain DLM arrives at a certain node. The DLM mechanism is the basic synchronization principle of the whole DAQ. A detailed description of the DLMs the SPADIC 1.0 chip uses is listed below.

TODO

6 Pin list

The 196 pins are counted from 0 to 195, starting from the uppermost pin of the left side, counterclockwise. Pads 65 and 179 are unused (DNC) due to analog/digital separation.

6.1 Analog part

49 pins are on the left (0-49), 15 on the bottom (49-64) and 15 pins on the top (180-195) side.

Pins	Net	Dec vs.	Type	Comment
0,45	vddp1	—	protection	to vddc
1,46	gndp1	—	protection	to gndc
2, 13, 24, 35, 47	vddc	gndc	power	CSA 1.8 V
3, 14, 25, 36, 48	gndc	vddc	power	CSA 0.0 V
4-11, 15-22, 26-33, 37-44	ampIn[0:31]	—	input	CSA, to connector
12, 23, 34, 62, 182	gndb	—	bulk	also in digital part
49	DecAdcVdda3	vdda	bias	ADC + analog CSA trigger
50	DecAdcVdda2	vdda	bias	ADC, internally switchable
51	DecAdcVdda1	vdda	bias	ADC, internally switchable
52	DecAdcAmpLow2	AmpLow	bias	ADC, internally switchable
53	DecAdcAmpLow1	AmpLow	bias	ADC, internally switchable
54	DecAdcGnda2	gnda	bias	ADC, internally switchable
55	DecAdcGnda1	gnda	bias	ADC, internally switchable
56, 188	vdda	gnda	power	ADC analog 1.8 V
57, 187	gnda	vdda	power	ADC analog 0.0 V
58, 186	AmpLow	vdda	power	ADC analog 0.3 V ?
59, 185	RefIn	?	power	ADC analog 0.96 V ?
60, 184	vddd1	gndd1	power	ADC digital 1.8 V
61, 183	gndd1	vddd1	power	ADC digital 0.0 V
63	vddp3		protection	lower ADC pins, to vdda
64	gndp3		protection	lower ADC pins, to gnda
180	gndp2		protection	upper ADC pins, to gnda
181	vddp2		protection	upper ADC pins, to vdda
189	Monitor	—	monitor	ADC + CSA, switchable, mmcx
190	BiasP	vddc	bias	DAC reference
191	nSourceBias	gndc	bias	CSA
192	pSourceBias	vddc	bias	CSA
193	nCasc	gndc	bias	CSA
194	pCasc	vddc	bias	CSA
195	xFB	p: gndc, n: vddc	bias	CSA

6.2 Digital part

33 pins are on the bottom (66-97), 49 on the right (98-146) and 33 pins on the top (147-178) side.

Pins	Net	Dec vs.	Type	Comment
66, 75, 84, 94, 103, 115, 127, 139, 150, 160, 169, 178	vddd2	gndd2	power	1.8 V
67, 76, 85, 95, 104, 116, 128, 140, 149, 159, 168, 177	gndd2	vddd2	power	0.0 V
68, 69, 110, 120, 121, 175, 176		—	unused	DNC
70, 88, 107, 124, 137, 156, 174	vddo	—	protection	digital pads, to vddd2
71, 89, 106, 123, 136, 155, 173	gndo	—	protection	digital pads. to gndd2
72, 90, 105, 122, 138, 154, 172	gndb	—	bulk	also in analog part
73, 77, 79	miscInBotN[2:0]	—	lvds input	to connector
74, 78, 80	miscInBotP[2:0]	—	lvds input	to connector
81	BiasNBotIn	gndd2	lvds bias	
82, 86, 92, 96	miscOutBotN[3:0]	—	lvds output	to connector
83, 87, 93, 97	miscOutBotP[3:0]	—	lvds output	to connector
91	BiasNBotOut	gndd2	lvds bias	
98	dataOutBN	—	lvds output	to connector, fast link
99	dataOutBP	—	lvds output	to connector, fast link
100	dataOutAN	—	lvds output	to connector, fast link
101	dataOutAP	—	lvds output	to connector, fast link
102	BiasNRightOut2	gndd2	lvds bias	fast links
108	userpin2	—	cmos output	to led + tp
109	userpin1	—	cmos output	to led + tp
111	serdesReady	—	cmos output	to led + tp
112	resN10	—	cmos output	tp
113	resN2	—	cmos output	tp
114	resN1	—	cmos output	tp
117	SCL	—	I2C	pull up (opt.), to 4pin
118	SDA	—	I2C	pull up (opt.), to 4pin
119	linkActive	—	cmos output	to led + tp
125	readoutEnabled	—	cmos output	to led + tp
126	enableReadout	—	cmos input	to connector/button, level?
129	triggerOutN	—	lvds output	to analog injection
130	triggerOutP	—	lvds output	to analog injection
131	dataInN	—	lvds input	to connector
132	dataInP	—	lvds input	to connector
133	BiasNRightOut1	gndd2	lvds bias	slow links
134	resN	—	cmos input	to FPGA + pullup
135	BiasNRightIn	gndd2	lvds bias	slow links
141	clk1N	—	lvds input	to connector
142	clk1P	—	lvds input	to connector
143	clk2N	—	lvds output	to mmcx, termination
144	clk2P	—	lvds output	to mmcx, termination
145	clk10N	—	lvds output	to mmcx, termination
146	clk10P	—	lvds output	to mmcx, termination
157, 151, 147	miscOutTopP[2:0]	—	lvds output	to connector
158, 152, 148	miscOutTopN[2:0]	—	lvds output	to connector
153	BiasNTopOut	gndd2	lvds bias	
170, 166, 164, 161	miscInTopP[3:0]	—	lvds input	to connector
171, 167, 165, 162	miscInTopN[3:0]	—	lvds input	to connector
163	BiasNTopIn	gndd2	lvds bias	

7 Known issues

7.1 I^2C

There is an inverter missing in the I^2C output drivers, which causes the SDA line to be constantly pulled down instead of being released. In order to enable proper write operation, the signal must be forced up by an external transistor, and the master must always *assume* that the slave sends the ACK after 8 bits have been transferred, because it cannot be deduced from the SDA signal anymore.

7.2 Hit detector

There is a bug in the digital comparator logic that leads under certain circumstances to the internal trigger signal not generated when it should be. The behaviour is the same for both thresholds (REG_threshold1, REG_threshold2), independent of the differential mode (REG_compDiffMode—if the differential mode is set, the difference between consecutive data samples is considered as "value") and can be described like follows:

If the threshold is *negative* and the data value is *positive*, no internal trigger is generated, although clearly the data value is above the threshold. In all other cases (i. e. threshold and value are both positive or both negative or threshold is positive and value is negative) the trigger is generated (or not generated) correctly.

The consequence is that it is not (in general) possible to effectively disable one of the thresholds by setting it to a large negative value, like -256.

7.3 Test data output

The test data output does not behave correctly in situations where its data buffer is almost empty; some data words are skipped. This becomes apparent for certain combinations of the hit window length and the selection mask.