

FE4D128x32

The Readout DATA SHEET and USER GUIDE

Alessandro Gabrielli
Filippo Maria Giorgi
Mauro Villa

INFN Bologna & Phys. Dept. University of Bologna

Last update Nov 3rd 2009

INDEX

INDEX	2
DATA SHEET	3
Introduction	3
The Test Chip Matrix	3
The Matrix Connection to Readout	5
Readout I/O	7
The Readout Blocks	8
The Slow Control Interface	10
The Output Stage	18
USER GUIDE	19
The Power-up	19
0-effort Start-up	19
Calibration Run	20
BIBLIOGRAPHY	21
APPENDIX A – the test cycle output sequence	22
APPENDIX B – evaluation of the hit coordinates	24

DATA SHEET

Introduction

The APSEL6D readout logic is an extension, with several innovations, of that introduced and implemented on the chip APSEL4D [1]. At the base of the readout technique remains a *pixel data bus* that is switched over the matrix columns in order to perform a sweep over the sensors grid. The whole bus is read in a parallel way in one clock cycle by the sparsifiers. Their role is to identify all the hits read out of the *pixel data bus* with a proper *xy (+time)* label and store them into an asymmetric FIFO: the *barrel*. The concepts of sparsifier and barrel are reintroduced with only few modifications, mainly due to the different organization of the matrix. The concept of *MacroPixel* (MP) is also reintroduced as a group of pixel with shared interconnections towards the readout logic. This technique allows a greater pixel density since fewer connections have to be routed over the sensor area.

The developed readout logic has been tailored to cope with wide sensor matrices (in particular to a 320x256 pixel matrix) but in this *Test Chip*, the sensor is only 128x32 pixels wide (same as the 4D version). The way the readout is adapted to this lower size sensor will be explained later.

In case of such large matrices, using a column scan strategy, the key point to reduce the average dead time is strictly bound to the mean sweeping time of the matrix; in this direction we decided to involve more than one readout instance, dividing the matrix into sub-matrices. A final output stage retrieves data from all the readouts involved and compresses them into a single data stream.

Several studies have been conducted to point out an optimal readout strategy, in order to reduce the mean dead time of pixels. The results of these studies showed that the best way to exploit the parallelism of the structure is to have a vertical subdivision of the whole matrix and a vertical shape of the MPs. This explains some planning choices illustrated later.

The Test Chip Matrix

The *Test Chip* matrix sensor is 128x32 pixels wide. The sensor is made up of binary pixels, giving a 0/1 kind of information. When a sufficient amount of charge is released within a pixel, the hit information is stored in a digital MOS latch.

The pixels are grouped into *MacroPixels* (MPs), small sets of sensors with a common interface towards and from the readout. The MPs have a vertical shape, 2x8 pixels wide; each sub-matrix is thus made up of 128 MPs (32 columns of 4 MPs).

The time granularity is provided by a dedicated clock, called *BCO*, which increments the *time counter* register. This clock determines the time window in which the hits are collected. The readout is responsible for the association of a hit MP to a determined time window.

The MP is enabled when the acquisition starts, which means all its latches can be triggered. When a hit (particle or noise) activates at least one pixel of a MP, the MP is considered fired and the *fast OR* signal, which is the logic OR of all the pixel latches, goes high. When the current time window ends, with the arrival of a *BCO* rising edge, all the currently active and fired MPs get frozen. Each MP has a *Latch enable* line that for this purpose which, if not active, stops the collection of hits (the not-fired pixel can latch no more, even if the threshold is crossed). In this way, the hit pattern of a MP is preserved till the readout phase; all the hits of the pattern refer to single and precise time stamp. The freezing logic and the time counter are implemented in the readout block and not into the MP itself, but they have been introduced here for a clearer explanation of the matrix features.

The readout of the matrix takes place by columns; a column-wide *pixel data bus*[31:0] is driven in turn by the columns of those MPs which contain at least one hit. The readout logic will perform a sweep activating only the columns of the MPs previously frozen.

The three-state outputs of the 16 latches of a MP are controlled by three enable signals: the *output enable* and 2 bits of *column enable*. The *output enable* is shared among all the MPs of a row while the *column enable* is shared among a pixel column, the cross combination of *column enable* and *output enable* individuates the pixels that are meant to drive the *pixel data bus*. To avoid conflicts on the bus, it is allowed to enable only 1 column of pixels at a time, while the full bus width can be exploited activating all the *out enable* lines.

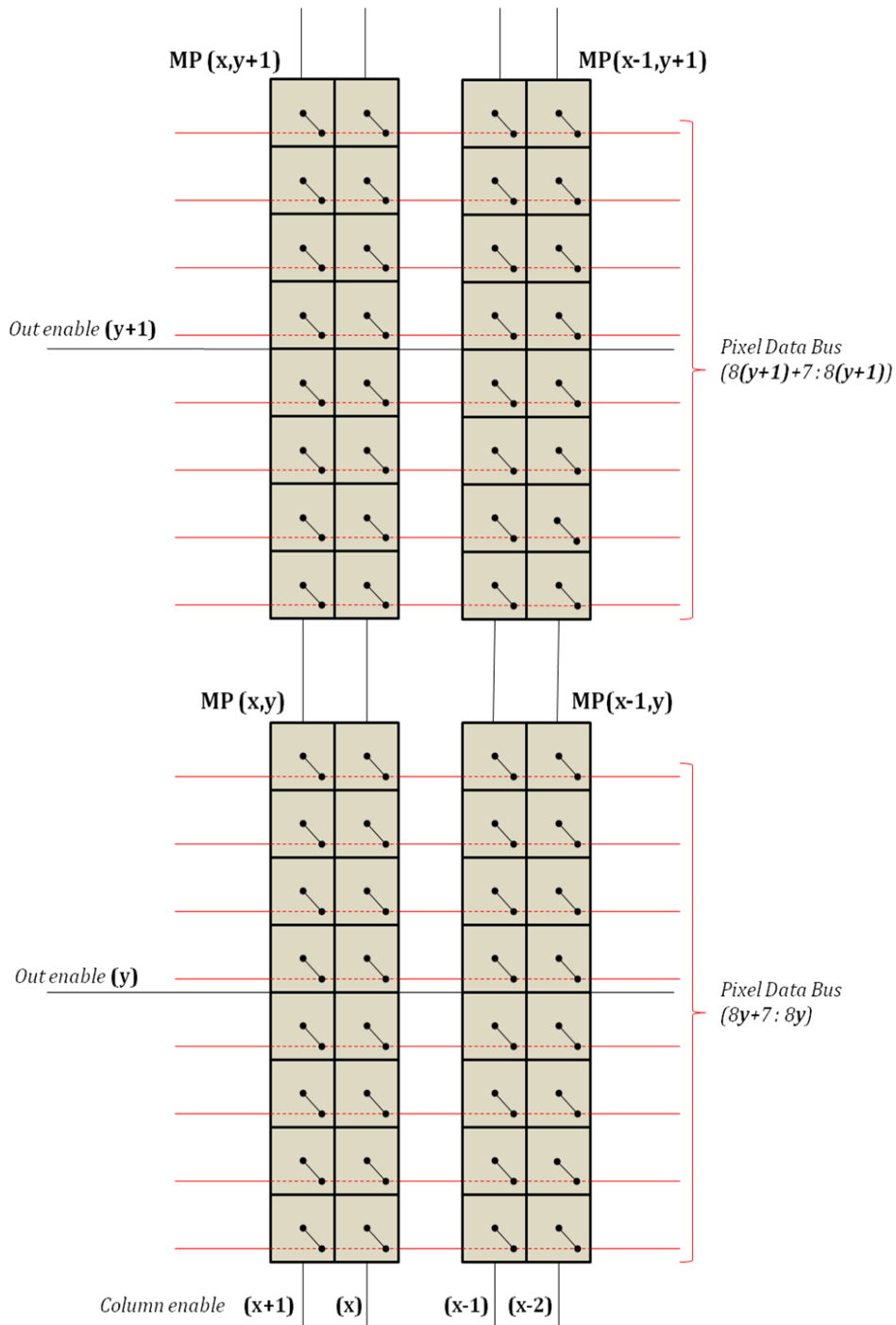


Fig 1. MP enable policy. The pixel data bus is driven by those pixels with active *column enable* and *out enable*.

A dedicated hardware reset signal for the latches of a MP is not foreseen; each MP is provided with an auto-reset logic that activates after the *column enable* sequence "01" "10" when *out enable* is high.

The matrix 128x32 is divided vertically into 2 independent sub-matrices 64x32. Each one has its own and independent readout instance. This means that two identical sets of signals connect the readout and the two parts of the matrix.

Sub-matrix 0	Sub-matrix 1
Latch enable_0[127:0]	Latch enable_1[127:0]
Fast OR_0[127:0]	Fast OR_1[127:0]
Column enable_0[63:0]	Column enable_1[63:0]
Out enable_0[3:0]	Out enable_1[3:0]
Pixel data bus_0[31:0]	Pixel data bus_1[31:0]

Tab1. Matrix interconnection signals. These are the digital signals which interconnects the matrix and the readout logic.

The Matrix Connection to Readout

As mentioned in the introduction, the readout project was meant to cope with wide matrices, in particular the target was 320x256 pixels. This area is 20 times greater than the Test Chip (128x32). Once it was established the target dimension, we elaborated a readout strategy which is supposed to afford a 1 MHz/mm² rate keeping the readout efficiency > 98,0%*. (This efficiency refers to readout only, no internal pixel dead time is taken into account. It has been evaluated with systematic simulation of the device models, using a 1us BC and a 60MHz of readout clock. The high data throughput generated must be granted by a wide-band external bus)

This led us to some choice, for example the implementation of several instances of readout working in parallel. Since the efficiency drops with the *mean sweeping time*, it should be kept as low as possible, one solution is to keep narrow the regions to sweep. That's is why we divided vertically the whole matrix into 4 sub-matrices 80x256 pixel wide.

All the parallel structures foreseen for the final application are excessive here but, in order to test their simultaneous functionality, the readout logic is left a bit oversized respect to the *Test Chip* needs.

The matrix available is a 128x32 array of pixels, which is much "narrower" than the final matrix version (32 pxl vs 256 pxl), but it is only about one half in width. By means of this it has been subdivided into two independent sub-matrices 64x32 (instead of four 80x256). In this way the parallel working of multiple readout cores can be tested.

For what concern the number of rows, that is about a order of magnitude lower, some redundancy in the readout components had to be left for less than a complete rework of the project (that would make useless this production step). The sparsifiers are the elements that read a portion of the pixel data bus in order to label the hits with a (x,y,t) coordinate set. The point is that each sparsifier developed can in principle read as many as twice the rows of the whole *Test Chip* matrix. Thus, always in order to test the parallel functionality of the components, it couldn't be realized a test chip with only one, half-used sparsifier. That's why 4 reduced sparsifiers are present in the readout, and each one is connected to a single MP following the connection scheme reported below:

<i>Matrix side</i>		<i>Readout side</i>
	0	↔ Spars_in_3[31:8]
Pix_data[31:24]	↔	Spars_in_3[7:0]
	0	↔ Spars_in_2[31:8]
Pix_data[23:16]	↔	Spars_in_2[7:0]
	0	↔ Spars_in_1[31:8]
Pix_data[15:8]	↔	Spars_in_1[7:0]
	0	↔ Spars_in_0[31:8]
Pix_data[7:0]	↔	Spars_in_0[7:0]

Each spars has a 32-bit wide input bus, then it has 3 unconnected MPs (see the Fig below).

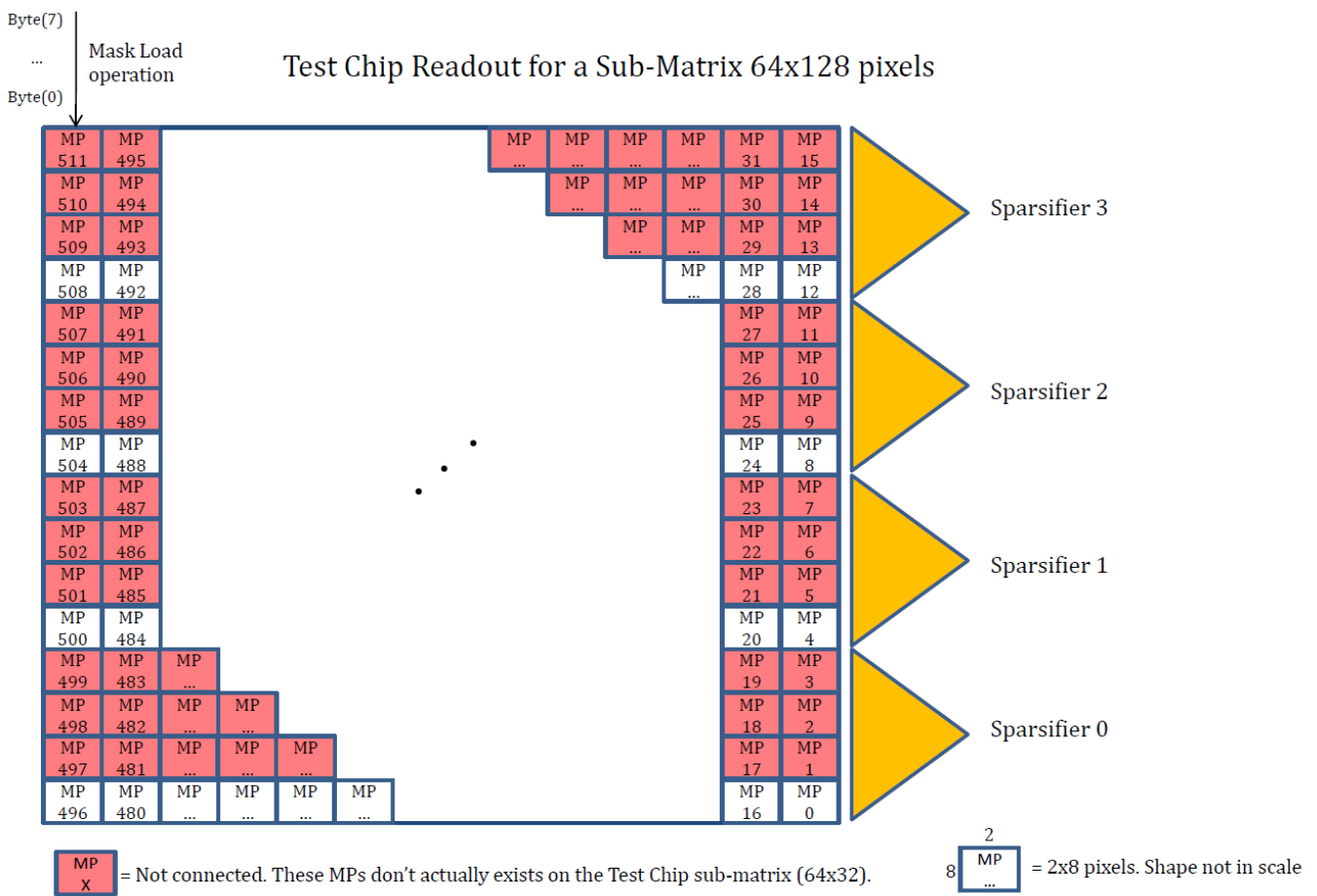


Fig 2. Submatrix scheme. Only 1 MP row is connected to each sparsifier. The other inputs can still be stimulated with a test cycle. (see Slow control-registers-command register).

Readout I/O

- Reset* – Asynchronous reset, active high. When released (and clocks are running) an automatic matrix cycle starts to reset all the MPs. This operation takes 64 RDclk cycles.
- RdClk* – This is the readout clock that feeds the matrix sweep and the first stages of readout (sparsifiers, barrel2, concentrator). It serves also most of the service logic like the I2C interface, the registers and all the slow control features (mask loading, test cycles...).
- FastClk* – It is used to transmit on a broad band external bus. It drives the final stages of readout (barrel1, final concentrator).
- BCclk* – The timing clock, it increments the time counter register and determines the time granularity of the acquisition.
- Master_Latch_enable* – This is a global enable signal that allows the MPs to work properly. If set to 0 all the MPs get frozen until it is re-activated. When set to 1, the MP can be hit and be frozen by the internal logic following the acquisition strategy.
- Global_fast_OR* – This signal export outside the chip the global OR of the Fast OR of all the MPs. It intended for debug purpose, not directly useful to a standard acquisition process.
- SDA* – The I2C-like data line, it is a bidirectional pin which must be connected to a pulled-up line where multiple chip can be connected together to the master.
- SCL* – The “clock” of the I2C-like transaction. Must be at least 4 times slower than the RDclk.
- Chip_addr[2:0]* – these are the pins that need to be hard-wired to assign a hardware address to the chip. Address “111” is reserved as a broadcast address.
- Data_out[13:0]* – This is the data bus for the fast bus, running synchronous on the FastClk. The hits on this bus are coded in the following way:

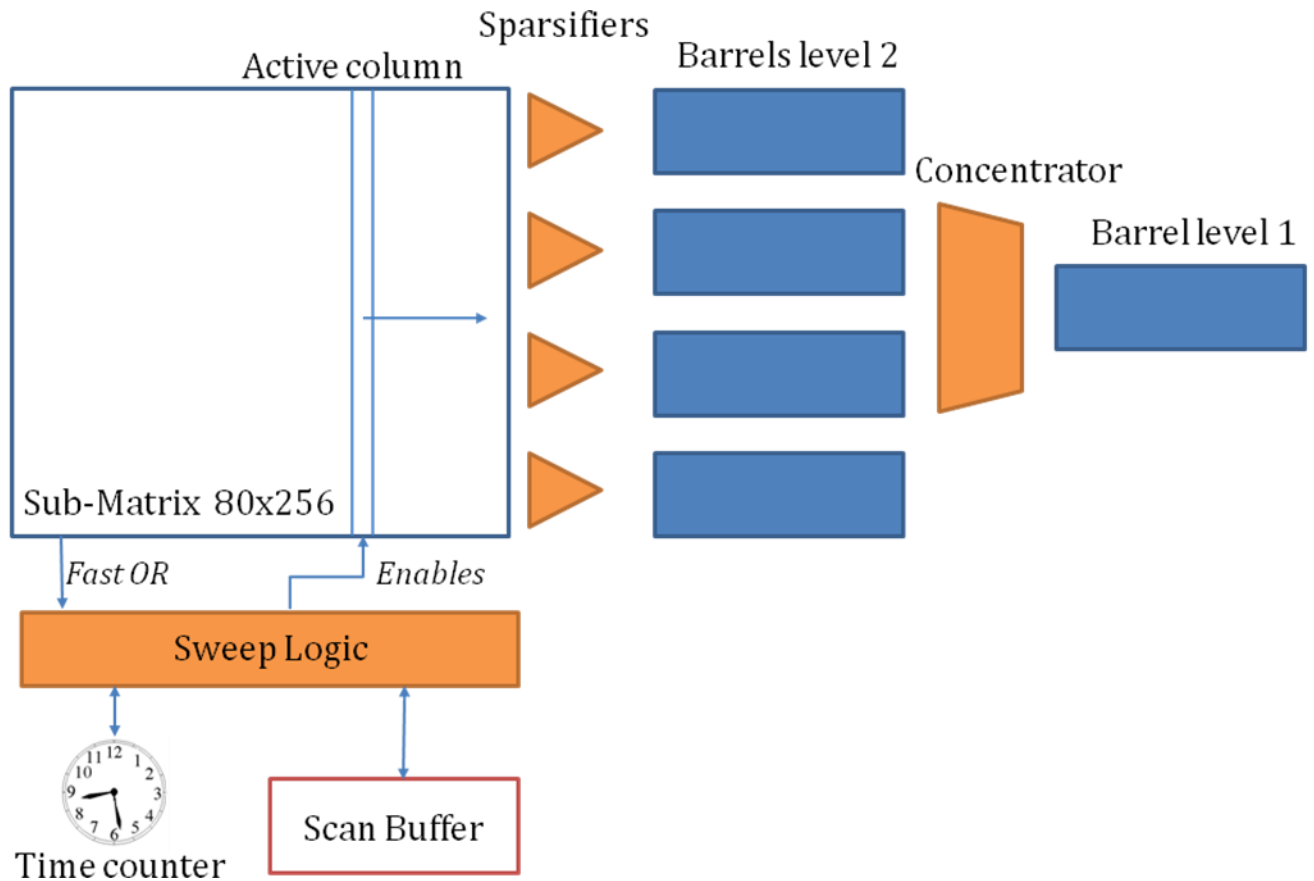
<i>If data_out[13]</i>	= 0;
<i>data_out[12:11]</i>	= sparsifier address;
<i>data_out[10]</i>	= zone Y address;
<i>data_out[9:4]</i>	= zone X address (which correspond to the sub-matrix relative x address of the fired pixel);
<i>data_out[3:0]</i>	= zone pattern;
<i>If data_out[13]</i>	= 1;
<i>data_out[12:10]</i>	= 000;
<i>data_out[9:8]</i>	= sub-matrix address (since only 2 sub-m. are present bit[9]=0);
<i>data_out[7:0]</i>	= time stamp field;

All values are coded as *unsigned*.

Data_valid – this bit indicates if the *data_out* bus contains a valid data.

The Readout Blocks

In the following figure it is presented the block diagram of readout. This is the instance replicated twice in the Test Chip, one for each sub-matrix.



The functional logic blocks that characterize one instance of *Readout* are the following:

The Sweeper

This block is responsible for three main tasks:

1. Freezing logic.
2. Time tagging of hits.
3. Sweeping logic.

The *freezing* logic states if a MP must be frozen or not. Only at the end of each time window the *freeze* signals array is re-evaluated. There is 1 bit for each MP, it is an enabling signal thus it freezes if active low. All the active MPs which were hit during the last time window get frozen until the relative readout sweep re-enables their latches.

The sweeping logic enables the proper MPs to talk on the *pixel data bus* following a precise scheme activating one pixel column at a time and activating only the rows of the desired MPs. The sweep of the active column over the sensor matrix is time dependent, which means that one sweep is dedicated to each time stamp. If no MPs are frozen during a time window, then no dedicated sweep is necessary. If two MPs, laying on the

same columns, are hit in different time stamp, then two sweeps are required over those columns, each time enabling only the row of the MP interested for that scan.

The map of the MPs hit during the last *time window* is stored in a dedicated FIFO together with the relative *time stamp*. This FIFO is called *Scan Buffer*. Until the *Scan Buffer* is not empty, a sequence of matrix scans succeed one after the other, one for each time stamp. When an active column scan ends, if the *not empty* flag of the Scan Buffer is active, the Finite State Machine of the sweeping logic pulls out of the buffer the new pattern of MPs that need to be scanned. The sweeper also informs the sparsifiers that a new scan, referring to time stamp X, is about to start.

The *Scan Buffer*

When no space is available in the Scan Buffer, if there are MPs that need to be frozen they are left active for the next BC periods until an empty memory location is available.

If the readout “loose” one BC, the event tagged with the successive time stamp will include also the hits belonging to the previous time window. In this case no hit information is lost, the only inconvenient is the worse time resolution.

It is important to explain what happens in not ordinary condition when this buffer go full. The FIFO has a depth of 8 maps, which grant a reasonable buffer in case of hit rate bursts. The nominal condition foresee a *time window* of about 500 ns / 1 us. If the *Rate On Area* (ROA) / *RDclk* ratio leads to a *Mean Sweeping Time* (MST) greater than the time window (*BCO* period), this FIFO fills up. The expected environmental conditions (ROA: 1 MHz/mm²) and the expected performance of the device (*RDclk*>60 MHz) should ensure this condition is far enough.

By the way, if the *BCO* period is shrunk too much, we can encounter a Scan Buffer Overflow. In this case no hit is lost, since the Scan Buffer simply stores the timing information. If no place is available to store the map of the fired MPs, they will not be frozen, waiting that some space will be freed during the incoming time window. As soon as the buffer starts to empty out, the new cumulative MP map is stored (always at the end of a *BCO* period) and the MPs involved are frozen. In this case the MP map is complete, and the hits will be all read out. The loss is in time resolution since those hits refer to a larger time window.

The user is informed with a dedicated register of how many times the readout instances undergo a Scan Buffer Overflow. No information is stored about which is the event with larger time uncertainty. Anyway the corrupted event can be pointed out since this situation take place at high hit-rates when we typically observe 1 scan for EACH time stamp. An overflow of the Scan Buffer would produce a sporadic couple of events separated by two or more time windows, making it easily detectable.

The *Time Counter*

The Time Counter is a counter synchronous on *RDclk* which counts the BC positive edges. The value of this counter is read by the Scan Buffer

The *Sparsifier*

The active column drives the Pixel Data Bus which is analyzed by the sparsifiers. Their task is to encode the space coordinates of the fired pixels into hit-words. Sparsified data is then stored in a formatted asymmetric FIFO called *Barrel*.

The sparsifiers encode also the information about the beginning of a matrix scan. When a new scan starts, each sparsifier stores a special word containing the associated time stamp in its adjacent Barrel. These words are called *SOS* (Start Of Scan) and they divide into bunches the hit-words cropped during different scans.

In the considered sub-matrix, we have 256 rows of pixels and thus a 256-bit wide Pixel Data bus. The developed sparsifier has a 64-bit wide input bus, and it is able to process the whole of it in one clock cycle. In the proposed architecture 4 sparsifiers working in parallel are implemented to cover the full Pixel Data Bus. Refer to “Matrix Connection to Readout” for details about the connections to the actual matrix implemented in this test chip.

To profit from possible clustering of hits, the sparsification is not done at the pixel level. The 32-bit sparsifier input bus is divided into 4-bit segments called zones. A fired pixel in a certain zone generates a hit-word containing information of the entire zone. A hit-word consists of the XY zone addresses plus the zone hit pattern.

In principle, all the 8 zones connected to a sparsifier can present fired pixels, thus it was made possible to encode all of them in the same clock cycle. This technique has been implemented foreseeing the presence of clustered patterns, allowing to reduce the total number of transmitted hits.

The Barrel B2

The Concentrator

The Barrel B1

The barrels directly connected to the sparsifiers are called Level 2 Barrels (B2s) while those collecting data from a whole sub-matrix are called Level 1 Barrels (B1s).

The Barrel is basically an asymmetric FIFO buffer that can store up to 8 hit-words per clock cycle. Each hit-word refers to a 4-bit zone, then each B2 can store up to 32 fired pixels per clock cycle. Since the complexity of synthesized logic increases fast with the number of hits that can be stored simultaneously, the introduction of the zone technique extends the range of inspected rows of the sparsifiers and barrels with a consequent reduction of the total required components at a fixed fifo depth.

A tree of barrels has been realized, it is composed of 4 B2, driven by their respective sparsifiers, and 1 B1 collecting data from the whole sub-matrix. In between, a smart data concentrator controls the flux of data preserving the time sorting of the hits. In B1 the set of scanned hit is stored after a single leading SOS word containing the common Time Stamp. In addition 2 bits are added to every hit in order to encode the respective B2 source address. The B2s have a depth of 8 hit-words, while the B1 can buffer up to 128 hit-words. The asymmetry is not only due to the 4 to 1 correspondence but also for the different emptying methods. B2s are data-through FIFOs, no hold condition on the output is foreseen. B1 outputs instead, for the adopted Round Robin algorithm, are kept in hold for one average emptying time, requiring more space for buffering. These depth values have been investigated in several simulations in order to find the optimal parameters.

The Slow Control Interface

The I2C interface

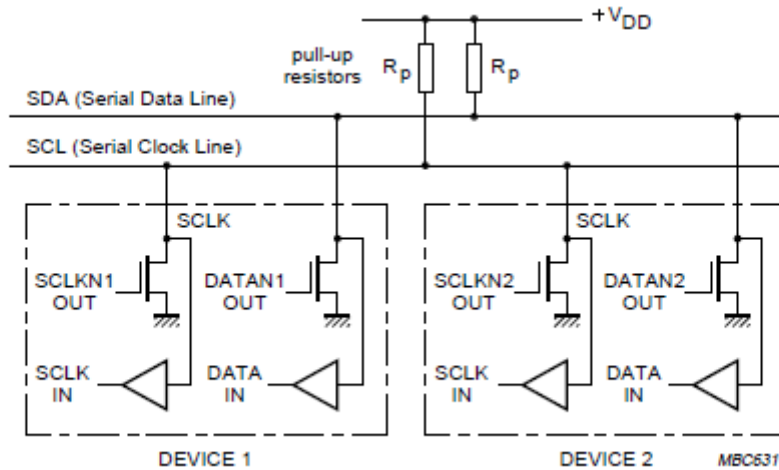
The Register File

The Slow Control Interface is based on an I2C-like environment with a fixed and predefined master-slave hierarchy. The I2C bus is used for read/write operations over a set of registers. All the instructions and the settings are passed to the chip by writing on these registers. Some of these registers are connected to inner flags of the chip and therefore are accessible in read mode only.

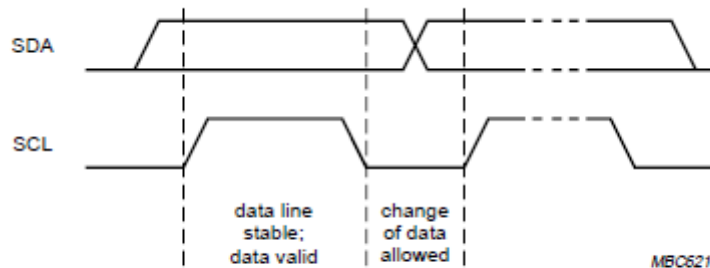
The I2C-like interface

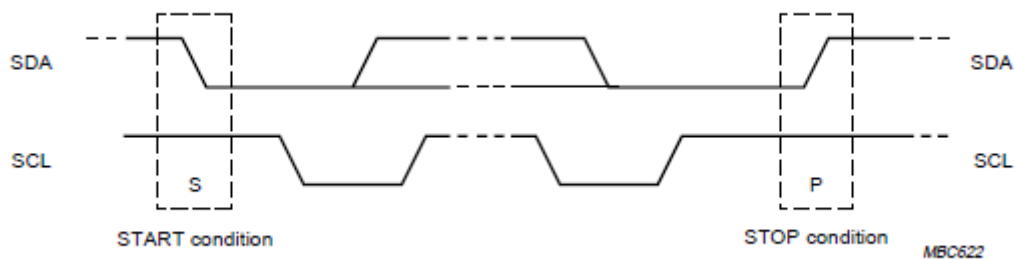
The I2C bus is made up of a single-bit bidirectional data line (SDA) and by a “clock” line (SCL). Since it is an asynchronous protocol, SCL can’t be defined as a real clock; it should be intended instead, as a control signal operated by the master unit. Since in our system there is no need to have a multi master compliant unit, we omitted all the logic for the initial negotiation; by means of this, the front-end chips are always operated in a slave configuration.

Like in the real I2C, each node on the bus must be individuated by unique hardware-defined address. In our case the *chip_addr[2:0]* pins are intended to be bonded properly for this purpose. One address must be reserved as a broadcast address. In our case it is fundamental that all the chips of a module (installed on the same I2C bus) perform some actions altogether. The communication is serial and asynchronous, only the master can start a communication and the slaves can only reply. The SDA line is bidirectional, it is externally pulled-up by a proper resistor, and all the drivers on this line are open drain/collector three-state buffers that can be tied to GND or kept in high-impedance. In the first case we obtain a logic 0 on the line (positive logic), otherwise the line carries a logic 1 due to the pull-up.



In normal condition (no transmission) both the lines are kept high (logic ‘1’s, no one is driving) while, during a transmission, changes of data on *SDA* are allowed only with *SCL* low. Communication is initialized with a special condition on the *SDA* line, while *SCL* is still high: the master put *SDA* to 0. A 1 to 0 transition on *SDA*, while *SCL* is high, is interpreted by the slaves as a *Start Condition*. After a start condition the *SCL* line starts to oscillate with the frequency of the desired data-rate on *SDA*.





At the end of communication the *slave* counterpart waits for a stop condition which must be generated by the *master* unit. Like the *start condition*, the *stop condition* is encoded with an *SDA* transition while *SCL* is high, this time it is a low to high transition.

The communication in between is divided into 3 parts; each part must be acknowledged by the receiving counterpart pulling down the *SDA* line to 0 for at least one *SCL* cycle. The transmitting unit must leave *SDA* in high impedance waiting to sense a logical 0 on it.

1. The first part is the *directive*, it is 9-bit long and it is always directed from master to slave (it follows that it is always a slave that must acknowledge the *directive*). It contains the information about the requested operation (read/write) and the addressed chip and register (chip number X, register number Y). Data is transferred *msb* (most significant bit) first.

Directive Structure:

Directive	Field name	Bit length	Comment
Directive[8:6]	Chip address	3 bit (msb first)	Address of chip "111" : broadcast
Directive[5]	RW_RO	1 bit	'1': Read/Write Reg. set '0': Read Only Reg. set
Directive[4:1]	Reg. Address	4 bit (msb first)	Address of the Register
Directive[0]	RW	1 bit	'1': Read operation '0': Write operation

2. The 2 other parts are byte exchanges, 1-byte long each. They can be directed master to slave (in case of a write operation), or slave to master (in case of a read operation). In the first case the slave must acknowledge the bytes and in the last case the master is intended to send the ACKs. Data is transferred *msb* and MSB (most significant Byte) first.

The sequence for a correct *Slow Control* transaction on the *I2C bus* would be:

(write operation):

MASTER: *StartCondition* + Directive + Z + byte1 + byte0 + Z + *StopCondition*

SLAVE: Z + Z + ACK + Z + Z + ACK + Z

(read operation)

MASTER: *StartCondition* + Directive + Z + Z + Z + ACK + *StopCondition*

SLAVE: Z + Z + ACK + byte1 + byte0 + Z + Z

The ACK low pulse of master can be sent with an arbitrary delay (multiple of the SCL period) since the slave chip will wait forever if no ack is sent.

WARNING: The *SCL* frequency must be at least 4 times slower than the *RDclk*.

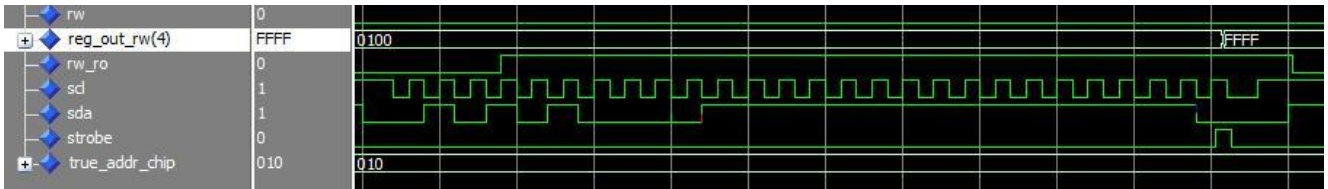


Fig N. A Slow control write operation in RW register(4).

The Registers

The chip is provided with 2 sets of 16-bit registers, one set is read/write and the other is read-only. Each set contains 16 registers for a total amount of $16 \times 2 \times 16 = 512$ bits of memory. Each register has a name and a specific purpose, when a register is composed by different fields of different meaning, it has a generic name, and the delimitations of the fields are illustrated here.

The RW registers

Here is presented the list of all the implemented RW registers. The hexadecimal values between the parentheses are the default values loaded after a hardware reset.

RW0 - address 0x0 - Command register* (0x0000)

The first of the RW registers is the command register. This register has a particular meaning and behaviour(*). For each word written in this location a FSM starts decoding the instruction and executes it. All the other registers simply store the data sent over the I2C bus. In the Test Chip there are 3 commands implemented:

×"0002" - *Load Mask*: This command push the mask shift register with the data contained in the operand register. The mask pattern is not stored into an accessible set of register, but it is loaded into a private shift register. Each sub-matrix has a dedicated shift register, 512-bit long (one for each MP - remember that the readout is tailored for 2 sub-matrix 64×128 , see *The Matrix Connection to Readout* section and FIG 2). For each Load Mask operation, *Operand[15:8]* is pushed into the shift register of sub-matrix 1 and *Operand[7:0]* is pushed into the shift register of sub-matrix 0. The shift operation is performed lsb first from bit 511 to bit 0 of the shift register. Each Load Mask operation masks a semi-column of MPs. This means that to mask MP0 it is necessary to perform $32MPcol \times 2 = 64$ load mask operations. The bits loaded that do not correspond to a MP actually connected are meaningless. The only bits to take care of, are those whose index *i* satisfies: $i \bmod 4 = 0$ ($i=0,4,8,12\dots$). The shift operation is performed all in one RDclk cycle.

×"0003" - *Reset Time Counter*: This command must be imparted in broadcast mode as it deals with the synchronization of the chips residing on the same

slow control bus. It is performed right after the command sequence is completed over the I2Clike bus.

x"0004" – *Cycle Test Scan*: With this command the readout chain is tested by stimulating the sparsifier inputs. A single hit is activated on each sparsifier input for each RDclk cycle. The scan is interleaved with a SOS at the beginning of each new sparsifier. Remember that the readout infrastructure of the Test Chip is tailored over a bigger matrix, and that the *data output bus* has been consequently downscaled. This implies a x4 repetition of the same pattern on the output. The duration of the operation depends on many variables, the RDclk and FastClk frequencies, the Flush Timeout register value and it is not easy to evaluate. Let's say that for a 25 ns RDclk and 12.5ns FastClk, and a timeout value of 256 FastClk periods, the whole operation, including the data transmission over the *output data bus*, takes about 7 us (excluded the command register loading).

The expected sequence on the reduced output bus is reported in APPENDIX A.

RW1 - address 0x1 - Operand (0x0000)

The Operand register contains the 16 bit operand required by some commands. (e.g. the mask pattern for the loadMask command). The operands must always be set before loading the relative instruction in the command register.

RW2 - address 0x2 - Config0 (0x00C0)

This register contains some flags and settings of the chip.

Config0 is divided in many fields:

Config0 word	Field name	Default
Config0[15:14]	--Unused--	"00"
Config0[13:12]	Phase select RO_1	"00"
Config0[11:10]	Phase select RO_0	"00"
Config0[9:6]	Rate Monitor range	"0011"
Config0[5:4]	Funnel operating mode	"00"
Config0[3:2]	Clock Multiplier	"00"
Config0[1]	Calibration Mode	"0"
Config0[0]	Activate MP	"0"

The *phase select* field set the acquisition phase in the boundary region between the RDclk and the FastClk. We can set the phase with which the FastClk logic samples the output of the RDclk logic. Since the FastClk can be at most 4 times faster than the RDclk, the selectable phases are 4:

```
when "00" => 1 fast_clk delay
when "01" => 2 fast_clk delay
when "10" => 3 fast_clk delay
when "11" => 4 fast_clk delay
```

If the 2 clocks have the same frequency, the setting of this register is meaningless.

The phase of the 2 *readout* instances (1 for each sub-matrix) can be set separately, but it is recommended to set them to the same value.

The *Rate Monitor Range* field: The *rate monitor* is a 32 bit counter, this field selects which is the range we want to observe on the dedicated read only register.

```
when "0000" => global_rate(15 downto 0);
```

```

when "0001" => global_rate(17 downto 2);
when "0010" => global_rate(19 downto 4);
when "0011" => global_rate(21 downto 6);
when "0100" => global_rate(23 downto 8);
when "0101" => global_rate(25 downto 10);
when "0110" => global_rate(27 downto 12);
when "0111" => global_rate(29 downto 14);
when "1000" => global_rate(31 downto 16);
when others => global_rate(31 downto 16);

```

The *Funnel Operating Mode* field: The Concentrator that takes data from the B2 was thought to be operated in different modes. This Test Chip can be only operated in mode "00". Other settings on this field may cause a corrupted data stream on the main output.

The *Clock Multiplier* field: The readout logic needs to know which is the ratio between the FastClk and the RDclk frequencies in order to work properly. For this reason we must set this parameter before any operation that implies the use of the data output bus (activating the MPs or performing a test scan).

```

when "00" => ratio=1
when "01" => ratio=2
when "10" => ratio=3
when "11" => ratio=4

```

The *Calibration Mode* bit: this bit is typically set to 0, which indicates a normal working mode for the chip, suitable for any physics run. If turned to 1, it swaps the chip readout into a special working mode, purpose-made for calibration runs. In case of a calibration run there are other mandatory settings. For details refer to User Guide section.

The *Activate MP* field: This bit states if the MPs are enabled to work or not. It is a register based signal with the same purpose of the external *Master Latch Enable* pin. At start-up they (external and reg.) must be kept both to 0, corresponding to all MPs frozen. The MPs get enabled if at least one of them rises to 1. To stop the acquisition, both must be set again to 0. This is a soft stop since it disables the MPs, but the readout can finish the scan of the previously latched hits.

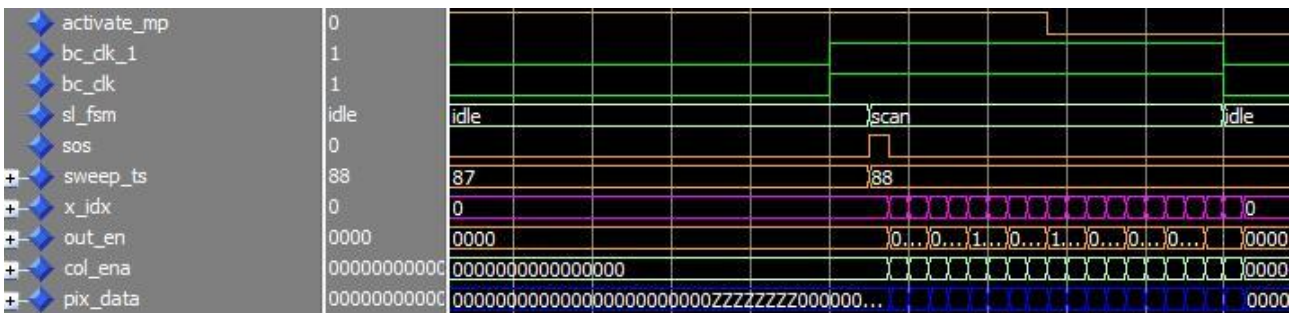


FIG. 3 – The *activate_MP* signal goes low, but the scan keep going until the end of scan. If there is a queue of scans to be performed, all of them are allowed to end.

RW3 - address 0x3 - Config1 (0x0000)

The *Config1* register remains unused in the Test Chip.

RW4 - address 0x4 - Flush Timeout (0x0100)

This register contains a 16-bit unsigned value used by the final stage of the readout, therefore it is represented in units of FastClk period. The default value (256) should be good enough in first approximation for a typical run in the nominal condition of rate on area (1MHz/mm²) and BCclk period (1us). In any case it must be set to a value greater than the BCclk period.

Remember that in even in case of a calibration run, the flush timeout must be longer than a BC period which in turns must be last longer than a calibration sweep. I.e. with a 25 ns RDclk (and at least a 12.5 ns FastClk), BCclk must be at least 50 us and the flush timeout at least 50us/12.5ns + security margin= \sim 5000 FastClk. For more details on the calibration run refer to the User Guide section.

RW5 - address 0x5 - Unused (0x0000)

This register unused in Test Chip.

RW6 - address 0x6 - Unused (0x0000)

This register unused in Test Chip.

RW7 - address 0x7 - Unused (0x0000)

This register unused in Test Chip.

RW8 - address 0x8 - Unused (0x0000)

This register unused in Test Chip.

RW9 - address 0x9 - Unused (0x0000)

This register unused in Test Chip.

RW10 - address 0xA - Unused (0x0000)

This register unused in Test Chip.

RW11 - address 0xB - Unused (0x0000)

This register unused in Test Chip.

RW12 - address 0xC - Unused (0x0000)

This register unused in Test Chip.

RW13 - address 0xD - Unused (0x0000)

This register unused in Test Chip.

RW14 - address 0xE - Unused (0x0000)

This register unused in Test Chip.

RW15 - address 0xF - Unused (0x0000)

This register unused in Test Chip.

The RO registers

The RO registers let the internal flags be accessible for monitoring and debug purposes.

RO0 – address 0x0 – Mask Counter 0

Unsigned. The *Mask Counter*[15:8] field increments on each load mask operation performed on sub-matrix instance 1.

Unsigned. The *Mask Counter*[7:0] field increments on each load mask operation performed on sub-matrix instance 0.

These two unsigned values increments together since each load operation acts on both matrix instances.

RO1 – address 0x1 – Mask Counter 1

Unused in Test Chip.

RO2 – address 0x2 – Error Flags

Error Flag[1:0] : *Unused*

Error Flag[2] : B1 readout 0 overflow flag

Error Flag[3] : B1 readout 1 overflow flag

Error Flag[5:4] : *Unused*

Error Flag[6] : Concentrator Error Readout 0 (if any but "00" then wrong time stamp in SOS word at B2).

Error Flag[7] : Concentrator Error Readout 1 (if any but "00" then wrong time stamp in SOS word at B2).

Error Flag[15:8] : *Unused*

RO3 – address 0x3 – Rate Counter

Unsigned. The number of hits found on both *pixel data buses* are accumulated at each RDclk into a dedicated counter. After 51200 RDclk periods the rate counter register is updated with the accumulated value, then the count starts over. data are accumulated from both the sub-matrices, so the rate refers to the whole sensor area.

Actually, the real rate counter register is 32-bit long so, in order to fit it into a standard RO register, only a 16-bit window of it is visible. It is possible to slide this window over the whole register by setting conveniently the *Rate Monitor Range* field of *Config 0* register.

The unsigned value *U* can be expressed in *MHz* with the following conversion formula :

$$\frac{2^n}{51200 \cdot T_{RDclk}} \cdot U$$

(T_{RDclk} expressed in us, and n = window shift expressed in bits)

RO4 – address 0x4 – BC lost Counter 0

Unsigned. Error flag counter that should be 0 at the end of a run.

It concerns the Scan Buffer overflow situations: the counter increments each time a BC edge arrives when the Scan Buffer is full and there are new fired MPs. Refer to BC lost condition in Scan Buffer block description.

Readout instance 0.

RO5 – address 0x5 – BC lost Counter 1

Unsigned. Error flag counter that should be 0 at the end of a run.

It concerns the Scan Buffer overflow situations: the counter increments each time a BC edge arrives when the Scan Buffer is full and there are new fired MPs. Refer to BC lost condition in Scan Buffer block description.

Readout instance 1.

RO6 – address 0x6 – BC lost Counter 2

Unused in Test Chip.

RO7 – address 0x7 – BC lost Counter 3

Unused in Test Chip

RO8 – address 0x8 – B1 Filling Level 0

Unsigned. Filling level of the B1 memory (0-64). Readout instance 0.

RO9 – address 0x9 – B1 Filling Level 1

Unsigned. Filling level of the B1 memory (0-64). Readout instance 1.

RO10 – address 0xA – B1 Filling Level 2

Unused in Test Chip.

RO11 – address 0xB – B1 Filling Level 3

Unused in Test Chip.

RO12 – address 0xC – B1 Mean Filling Level 0

Unsigned. Mean Filling level of the B1 memory (0-64). Readout instance 0.

RO13 – address 0xD – B1 Mean Filling Level 1

Unsigned. Mean Filling level of the B1 memory (0-64). Readout instance 1.

RO14 – address 0xE – B1 Mean Filling Level 2

Unused in Test Chip.

RO15 – address 0xF – B1 Mean Filling Level 3

Unused in Test Chip.

The Output Stage

The Final Concentrator

Each sub-matrix is provided with an independent and parallel readout instance. The Final Concentrator is the element that collects data from the 4 B1 instances in order to drive the output data bus with a proper data protocol. The B1s are emptied with a Round Robin algorithm and a special Header Word is sent before switching to a new B1. In the Header Word are specified the Time Stamp of the following hits and their B1 source address. Following hits preserve the same B1 data formatting. Refer to the *Readout I/O* section for detail about the data format.

USER GUIDE

This appendix is intended to provide the user with some step-by-step instructions in order to easily initialize and start the device; it shows and explains the more common initializing routines and it is intended to describe the device with a practical approach.

The Power-up

When the chip is turned on, the whole logic needs a hardware reset to correctly initialize the values of the registers and the matrix latches. The hardware reset is asynchronous and active high. By the way it is recommended to run the FastClk and the RDclk for at least 1 period while the reset is still high.

After the reset release, the FSM that perform the clean-up sweep over the matrix starts to run. This procedure ensures that all the MP latches are correctly reset after start-up. This operation takes at least 1 RDclk cycle for each column of the matrix, the RDclk is mandatory during this phase. All the eventual hits found during this sweep are neither sparsified nor stored since they have no physical meaning. From power-up, to the end of this procedure, *MasteLatchEnable* pin must be kept at GND to prevent the latching of new noise hits (MP kept frozen).

Before performing another operation (slow control transactions ecc...), it is recommended to wait the latency of this command. (ex. For a 40 MHz Rdclk (period 25 ns), the time to wait will be $64 \times 25 = 1,6$ us. Better to round up to 2 us).

0-effort Start-up

After the reset release, the main configuration registers are preset with default settings. Remember that these values are restored after each reset. The preloaded values, (as reported in the data sheet section) are:

```
FastClk freq/RDclk freq = 0
Flush timeout           = 256
Enable MP               = 0
Rate monitor range     = 3
Calib mode              = 0 (physic mode)
```

These default values have been set foreseeing a run with same RDclk and FastClk ~ 40/60 MHz and a BCclk period of ~ 1 us.

After the reset sweep latency the chip is ready to start an acquisition.

Now the matrix is empty and completely frozen, the user can enable the MPs activating the *MasteLatchEnable* pin on the chip. The MPs can be activated also through the *Slow Control Interface* by setting the appropriate register.

ML ena pin	activate MP (config0 reg.)	
0	0	→ MPs frozen
1	0	→ MPs activated
0	1	→ MPs activated
1	1	→ MPs activated

This start-up requires no I2C transaction.

Calibration Run

To start a calibration run, we are supposed to work a bit with the I2C-like bus in order to load the required settings:

```
Calib mode bit = 1 (config reg)
Activate MP     = 1 (config reg)
Mask 1 sub-matrix
Flush timeout > BCclk period > Max Matrix Sweep Time
```

In the calibration mode the matrix sweep is much slower than in normal mode. This is due to high data throughput that is produced during a calibration step (1 BCclk edge). By means of this, the scan over the matrix is interleaved by fixed delays, in this way all the hits have time to be read out of the data bus when the threshold scan pass through the high occupancy band. For the same reason it is required to calibrate 1 sub-matrix a time masking completely the other.

The formula to evaluate the *Max Matrix Sweep Time* is

$$64 * T_{RDclk} + 1600 * T_{RDclk}$$

i.e. for a 40 MHz RDclk, the Max Matrix Sweep Time is 41.6 us. In this case we adopted for simulation a 50 us BCclk period and a flush timeout register value of 5000 which correspond to 5000*12.5ns of FastClk period. Hence 62.5 us > 50 us > 41.6 us which is OK.

BIBLIOGRAPHY

- [1] APSEL4D User Guide – *Gabrielli Alessandro*
<http://www.bo.infn.it/slim5/Apsel4D/Readout128x32-Users-Guide.pdf>

APPENDIX A - the test cycle output sequence

On the first column there is an incremental index enumerating the hits, the X coordinate is on the third column and the Y coordinate on the last column.

When the idx is -1 it means it is a header word (idx not incremented). In this case the third column (X) points out the instance of *Readout* to which the following data are referred.

Idx	X	Y			
-1	0	0	00	47	0 0 7
0	0	0	0	48	0 0 0
1	0	0	1	49	0 0 1
2	0	0	2	50	0 0 2
3	0	0	3	51	0 0 3
4	0	0	4	52	0 0 4
5	0	0	5	53	0 0 5
6	0	0	6	54	0 0 6
7	0	0	7	55	0 0 7
8	0	0	0	56	0 0 0
9	0	0	1	57	0 0 1
10	0	0	2	58	0 0 2
11	0	0	3	59	0 0 3
12	0	0	4	60	0 0 4
13	0	0	5	61	0 0 5
14	0	0	6	62	0 0 6
15	0	0	7	63	0 0 7
16	0	0	0	-1	0 0 00
17	0	0	1	64	1 0 8
18	0	0	2	65	1 0 9
19	0	0	3	66	1 0 10
20	0	0	4	67	1 0 11
21	0	0	5	68	1 0 12
22	0	0	6	69	1 0 13
23	0	0	7	70	1 0 14
24	0	0	0	71	1 0 15
25	0	0	1	72	1 0 8
26	0	0	2	73	1 0 9
27	0	0	3	74	1 0 10
28	0	0	4	75	1 0 11
29	0	0	5	76	1 0 12
30	0	0	6	77	1 0 13
31	0	0	7	78	1 0 14
-1	0	1	00	79	1 0 15
32	0	0	0	80	1 0 8
33	0	0	1	81	1 0 9
34	0	0	2	82	1 0 10
35	0	0	3	83	1 0 11
36	0	0	4	84	1 0 12
37	0	0	5	85	1 0 13
38	0	0	6	86	1 0 14
39	0	0	7	87	1 0 15
40	0	0	0	88	1 0 8
41	0	0	1	89	1 0 9
42	0	0	2	90	1 0 10
43	0	0	3	91	1 0 11
44	0	0	4	92	1 0 12
45	0	0	5	93	1 0 13
46	0	0	6	94	1 0 14
				95	1 0 15
				-1	0 1 00
				96	1 0 8
				97	1 0 9
				98	1 0 10
				99	1 0 11
				100	1 0 12
				101	1 0 13
				102	1 0 14
				103	1 0 15
				104	1 0 8
				105	1 0 9
				106	1 0 10
				107	1 0 11
				108	1 0 12
				109	1 0 13
				110	1 0 14
				111	1 0 15
				112	1 0 8
				113	1 0 9
				114	1 0 10
				115	1 0 11
				116	1 0 12
				117	1 0 13
				118	1 0 14
				119	1 0 15
				120	1 0 8
				121	1 0 9
				122	1 0 10
				123	1 0 11
				124	1 0 12
				125	1 0 13
				126	1 0 14
				127	1 0 15
				-1	0 0 00
				128	2 0 16
				129	2 0 17
				130	2 0 18
				131	2 0 19
				132	2 0 20
				133	2 0 21
				134	2 0 22
				135	2 0 23
				136	2 0 16
				137	2 0 17
				138	2 0 18
				139	2 0 19
				140	2 0 20
				141	2 0 21
				142	2 0 22
				143	2 0 23

144	2	0	16	201	3	0	25
145	2	0	17	202	3	0	26
146	2	0	18	203	3	0	27
147	2	0	19	204	3	0	28
148	2	0	20	205	3	0	29
149	2	0	21	206	3	0	30
150	2	0	22	207	3	0	31
151	2	0	23	208	3	0	24
152	2	0	16	209	3	0	25
153	2	0	17	210	3	0	26
154	2	0	18	211	3	0	27
155	2	0	19	212	3	0	28
156	2	0	20	213	3	0	29
157	2	0	21	214	3	0	30
158	2	0	22	215	3	0	31
159	2	0	23	216	3	0	24
-1	0	1	00	217	3	0	25
160	2	0	16	218	3	0	26
161	2	0	17	219	3	0	27
162	2	0	18	220	3	0	28
163	2	0	19	221	3	0	29
164	2	0	20	222	3	0	30
165	2	0	21	223	3	0	31
166	2	0	22	-1	0	1	00
167	2	0	23	224	3	0	24
168	2	0	16	225	3	0	25
169	2	0	17	226	3	0	26
170	2	0	18	227	3	0	27
171	2	0	19	228	3	0	28
172	2	0	20	229	3	0	29
173	2	0	21	230	3	0	30
174	2	0	22	231	3	0	31
175	2	0	23	232	3	0	24
176	2	0	16	233	3	0	25
177	2	0	17	234	3	0	26
178	2	0	18	235	3	0	27
179	2	0	19	236	3	0	28
180	2	0	20	237	3	0	29
181	2	0	21	238	3	0	30
182	2	0	22	239	3	0	31
183	2	0	23	240	3	0	24
184	2	0	16	241	3	0	25
185	2	0	17	242	3	0	26
186	2	0	18	243	3	0	27
187	2	0	19	244	3	0	28
188	2	0	20	245	3	0	29
189	2	0	21	246	3	0	30
190	2	0	22	247	3	0	31
191	2	0	23	248	3	0	24
-1	0	0	00	249	3	0	25
192	3	0	24	250	3	0	26
193	3	0	25	251	3	0	27
194	3	0	26	252	3	0	28
195	3	0	27	253	3	0	29
196	3	0	28	254	3	0	30
197	3	0	29	255	3	0	31
198	3	0	30				
199	3	0	31				
200	3	0	24				

APPENDIX B - evaluation of the hit coordinates

The computation of the X coordinate is quite easy, the 5-bit X field of the data output bus, encodes the x coordinate as an unsigned:

```
x := unsigned(data_out[9:4])
```

The Y coordinate is a bit more tricky due to the zone sparsification. In one data word, there could be, in principle, many hits; in that case a different Y coordinate must be evaluated for each hit.

```
y := unsigned(data_out[12:11])*N_SPARS_YZONE*YZONE_SIZE +  
unsigned(data_out[10])*YZONE_SIZE + pattern_relative_y
```

where :

```
N_SPARS_YZONE      := 2;  -- number of zones connected to a sparsifier  
Y_ZONE_SIZE       := 4;  -- dimension of a zone in pixels.  
pattern_relative_y := 0->3 -- depends on the pattern of the zone.
```

i.e. the data word x"1409" or "0 10 1 000000 1001" has a multiple hit in the pattern data_out[3:0] = "1001". The X coordinate is common and it is 0.

Applying the formula above to the hits "1000" and "0001" leads to

X1 = 0

Y1 = 2*N_SPARS_YZONE*YZONE_SIZE + 1*YZONE_SIZE + 3 = 23

X2 = 0

Y2 = 2*N_SPARS_YZONE*YZONE_SIZE + 1*YZONE_SIZE + 0 = 20

The coordinates of the pixels are labeled by convention 0:127 for the X and 0:31 for the Y.