CERN-RD53-PUB-xxxx
Version 0.21, October 23, 2019

The RD53B Pixel Readout Chip Manual

ABSTRACT: Manual for the RD53B design common to the ATLAS and CMS pixel readout chips.

Contents

1. Overview

The readout chips for the ATLAS and CMS HL-LHC pixel detectors are two separate instances of a common design framework called RD53B. The main difference between ATLAS and CMS is the size of the pixel matrix. RD53B is based on the RD53A integrated circuit [1] together with production requirements defined by the experiments [**?**].

RD53B is a pixel readout chip framework that can be instantiated into different size physical chips. The design work and much of the verification are largely independent of the final instantiated size. RD53B consists of a *pixel matrix* and a *chip bottom*. The pixel matrix is built up of identical 8 by 8 pixel *cores* stepped and repeated in columns and rows. A core is physically 400 μm by 400 μm. The selected numbers of core columns and rows determine the chip size. The chip bottom contains all the system functionality and should be viewed as a fixed element that does not depend on matrix size. A physical chip, therefore, cannot be *narrower* than 20 mm (50 cores), because that is the width of the unique wire bonding pad frame in the chip bottom, but it can be wider. The *height* (number of core rows) is not constrained by the chip bottom, but is limited to a maximum of 50 by power and bias distribution as well as readout timing. This high level organization concept is shown in Fig. 1. The instantiated dimensions are detailed in Sec. 2.

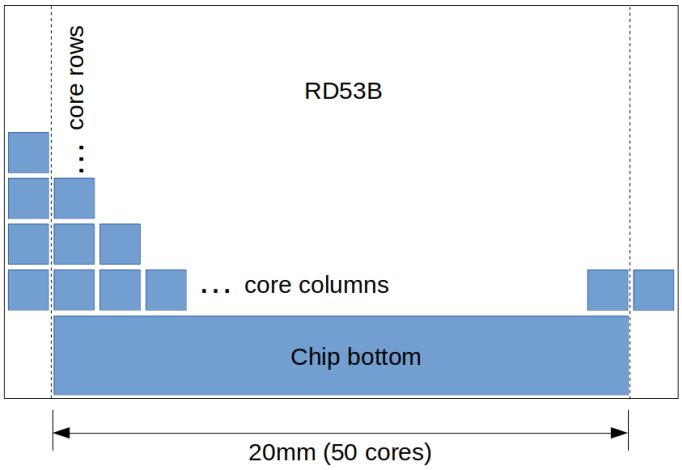

Figure 1: Conceptual depiction of RD53B framework, with a matrix composed of 50 or more columns by up to 50 rows of identical cores, and a fixed chip bottom. The dashed lines indicate the minimum width of 50 cores.

The core contains 64 pixel front ends organized in 16 identical so-called *analog islands* with 4 fronts ends each, which are embedded in a flat digital synthesized "sea" as shown in Fig. 2. Although the circuitry around each island is not identical but depends on the placement of gates by the synthesis tool, RD53A and prior small prototypes have shown that this approach (with suitable synthesis constraints) does not introduce any visible systematic differences between islands. The core size has not been made larger so that it can be verified with a transistor level analog simulation. The analog front end and island design are described in Sec. 5. The digital core design is described in Sec. 7. Further details about the pixel matrix produced by stepping and repeating cores, including the distribution of analog biases, are given in Sec **??**.

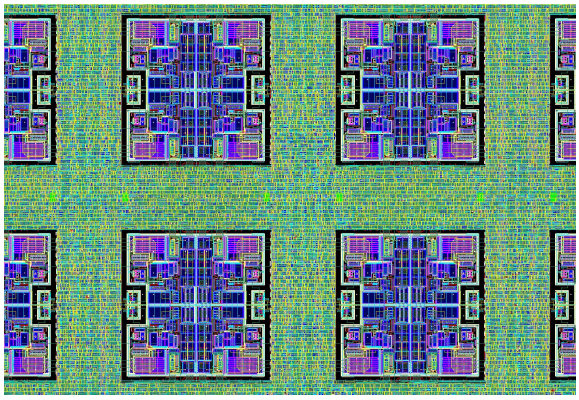

Figure 2: Layout view of analog islands within synthesized logic. Four complete islands can be seen in the center of the figure. One core contains four by four analog islands.

The chip bottom contains all system functionality and the wire bond pads. RD53B is a system-on-chip including power management, sophisticated digital communication, sensing and monitoring. An overview of the system operation, including a description of the reset scheme, is given in Sec 3 This also serves as a quick-start guide, with details about each function covered in independent sections. All tabular information, including pinout and configuration register values, is collected in the Reference section (Sec 17) for efficient lookup, rather than spreading the tables throughout the document.

Power management, including design of the Shunt-LDO regulators are covered in Sec 4. The command and control interface (how one talks to the chip) and the configuration are covered in Sec 8. The data output (what comes out of the chip), including special (non-hit data) and the aggregation of data from multiple chips, are described in Sec. 10. The sensing and monitoring functions are described in Sec 12. Test features and miscellaneous functions are covered in Sec 13. Technical details about the data flow and digital design of the chip bottom are given in Sec. 15. The design of the bump bond and wire bond pads is described in Sec 16. RD53B only has wire bond pads along the bottom edge.

Parameter	ATLAS	CMS
Pixel bump pitch	50 μm $\times$ 50 μm	
pixel rows (H)	384	336
pixel columns (W)	400	432
core rows	48	42
core columns	50	54
Chip width (including seal ring)	20.054 mm	21.654 mm
Chip height (including seal ring)	21.022 mm	18.622 mm

Table 1: Chip sizes for ATLAS and CMS chips.

2. Dimensions and Floorplan

RD53B uses a 9 metal layer stack, consisting of 7 thin, 1 thick and 1 ultra-thick metal layers. In addition, the 28K AP layer is also used for power lines distribution. In Fig. 3 the layout and functional view of RD53B floorplan are shown. The sensitive area of the chip is placed at the top of the chip and is arranged as a matrix of pixels of 50 μm $\times$ 50 μm according to Table 1 The peripheral circuitry is placed at the bottom of the chip and contains all global analog and digital circuitry needed to bias, configure, monitor and readout the chip. The wire bonding pads are organized as a single row at the bottom chip edge and are separated from the first row of bumps by 1.7 mm in order to allow for wire bonding after sensor flip-chip (Sec. 16). However, the wire bond pads are also designed compatible with Thru-Silicon Via post processing.

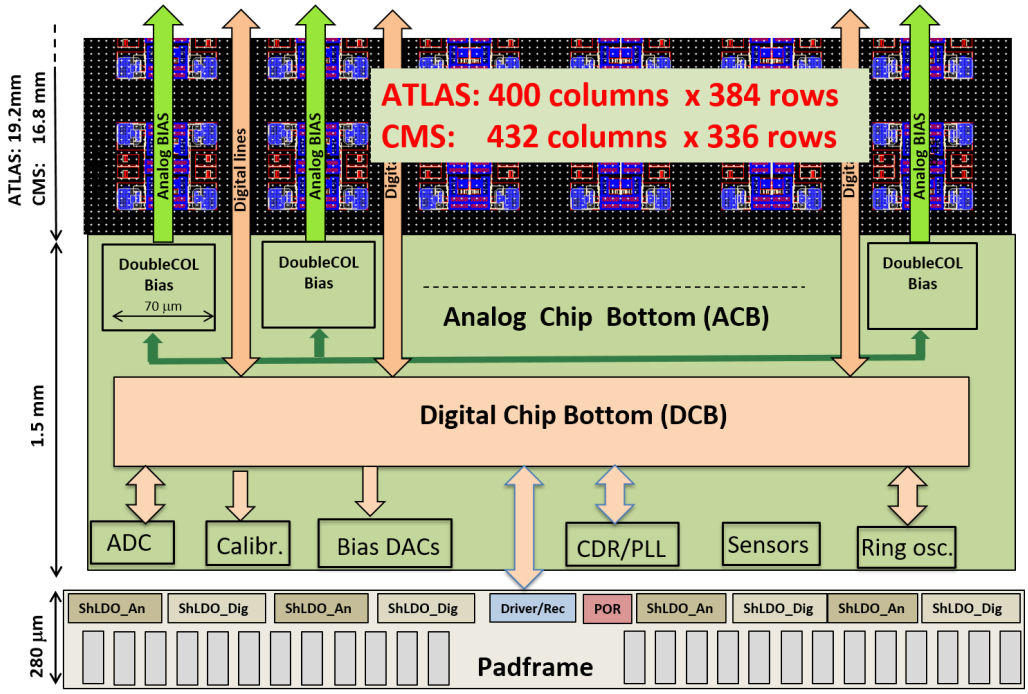

Figure 3: RD53B floorplan, functional view.

In the chip periphery, all the analog building blocks are grouped in a macroblock called Ana-

log Chip Bottom (ACB), which is fully assembled and characterized in an analog environment. The ACB block is surrounded by a synthesized block, called Digital Chip Bottom (DCB), which implements the Input, Output and Configuration digital logic, as described in (Sec. 15).

3. System Operation and Reset (useful Quick Start Guide)

This section walks through the steps for basic operation of a single chip on the bench and describes the reset strategy, which is different from RD53A. In general every step is just one possible option from several possible choices of connections and configuration. In addition to a practical quick-start guide, this section can serve as a launching point to the rest of the document, as sections with options and details are referenced throughout. Whenever registers or pins are mentioned they can be found in the reference section (17).

3.1 Chip Startup

The startup sequence is power, clock, communication, configuration, operation.

Power Typical bench testing will use LDO powering in which the internal regulators are used as classic voltage regulators fed from the constant voltage power supply, rather than serial power regulators fed from a constant current supply. A single chip card will contain jumpers to select LDO mode. A single power supply will be connected to all the chip's V_IN pins, the shunt current control resistors R_{shunt} will be open, and the offset control voltage R_{ofs} will be open (disconnected). For details see Sec. 4. External components should be set their nominal values (Table 14).

LDO mode allows to view the current consumption. The supply voltage should be a minimum of 1.4 V and the chip and never more than 2.0 V. 1.7 V is a typical setting. When power is turned on, the current consumption will be low (under 100 mA), because the default configuration in RD53B is for minimal power. The VDDA and VDDD regulator outputs, which connect to external decoupling capacitors, should be checked to be at approximately 1.2 V, which is the default setting.

Clock All the above happens upon application of power. The reference and power regulation circuits do not have any reset capability. The most critical function after power is clock. The PLL Clock Data Recovery circuit will become active as soon as it has power and will produce clock edges on all the internal clocks even in the absence of any external bit pattern supplied to the command input. The CML data output 0 should be active and producing edges. The frequency of the internal clocks and CML output might be wrong in the absence of an external control signal, but there should be edges. To proceed further, initialization and reset procedures are needed.

The reset organization is described in Sec. 3.2. Regardless of the state of the command input during power up, after power is stable, communication must be initialized by setting the command line to a DC level for at least 10 μs and then supplying a clock pattern for at least 1 ms. The clock pattern is an uninterrupted stream of PLL_LOCK (noop) symbols (Sec. 8.2). (Which DC level-low or high- is irrelevant, especially since the command input will be AC-coupled and the internal logic value will be set by the hysteresis circuit.) This idling of the command line is the main reset for RD53B. It can be done at any time to recover the chip from a bad state. After the command idle reset the PLL will enter lock mode and supplying a clock pattern is critical for it to lock to the correct frequency. There will be no indication that the PLL has locked and so it is necessary to hold the clock pattern long enough to leave no doubt that there has been enough time to lock. Further details are given in Sec 14.

Communication Now all clocks will be at the correct frequency, but the chip is not yet ready to understand commands, because the channel synchronizer has not yet locked to a particular frame alignment. Therefore, after the clock pattern, the sending of sync symbols should be enabled. One can send a constant string of sync symbols or simply enable automatic insertion of one sync every so many frames (eg. every 32). So either (sync, sync, sync,...) or (noop, noop, noop, sync, noop, noop,...)- it makes no difference. The important thing is to send a large number of syncs (say, 100) before sending any commands. When the channel synchronizer locks, the lock signal can be seen on the general purpose CMOS output (for diagnostic purposes). Commands will not be accepted. (Perhaps a pattern noop, noop, noop, sync, noop, noop,... can also work for PLL lock instead of a pure clock pattern- this can be decided by testing.)

Configuration The default configuration is low power and allows limited tests. To use a programmed configuration the configuration selection register must be programmed with the key that causes the switching (Sec. 3.2.1). A soon as clock edges present in the above command idle reset state, the programmed configuration will also be reset to equal the default, so when the configuration is switched from default to programmed nothing will actually change. The default configuration enables a single lane AURORA output (lane 0) at 1.28 Gbps with pixel matrix is entirely disabled. It is intended for communication tests. Writing and reading configuration registers and temperature and radiation monitoring should all be possible with the default configuration. In general a new configuration will have to be loaded for most single chip testing. Test setups will include a baseline configuration suitable for most tests (which may also be called default in test setup documentation, should not be confused with the internal chip default configuration).

3.2 Reset

The driving requirements of the RD53B reset scheme are:

- Avoiding introduction of Single Event Effect vulnerability. This led to having reset capability only on circuits that absolutely need it, and to use only synchronous reset for them. A synchronous reset signal is a logic level that is sampled locally every clock edge. Spurious glitches on this reset signal have no effect (in contrast to an asynchronpus reset, for which a transition produces a reset regardless of clock).

- Need for a low power default configuration present immediately upon power up. This is done without the use of a power-on reset, as this would require an asynchronous reset on the global configuration registers. The default configuration is not stored in registers, but hard-wired and selected by a 2-to-1 multiplexer (Sec. 3.2.1)

- The ability to reset a chip (or a subcircuit within) without cycling the power, which would require tuning off and on an entire serial chain. This is accomplished with a new function that detects activity on the command input (Sec. 3.2.2)

The overall reset organization is shown in Fig. 4. All signal use negative logic: low means reset. There is a power-on reset generation circuit in the chip, inherited from RD53A, but the output of this circuit is not used to reset anything in RD53B and is only sent to the general purpose output multiplexer so that is available for external routing. The only asynchronous reset signal that

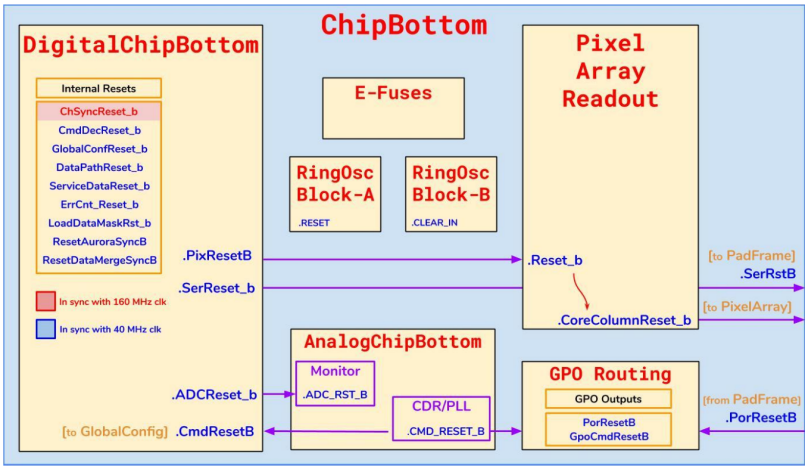

Figure 4: Block diagram of reset signals in the RD53B chip as described in the text.

is used is the command activity detector (Sec. 3.2.2), labeled .CMD_RESET_B in the figure. This signal resets the PLL circuit that recovers the clock, selects the default configuration when low (Sec. 3.2.1), and is sent to the digital chip bottom where it is synchronized, and after synchronization used to actuate all the synchronous resets.

All digital blocks have synchronous resets. These can be individually actuated at any time using the Global Pulse command, in addition to the actuation by the synchronized .CMD_RESET_B signal. The organization of the digital block resets is shown in Fig. 5. The global configuration registers are explained in Sec. 3.2.1. The logic to write and read global configuration has its own synchronous reset, labeled .GlobalConfReset_B in the figure.

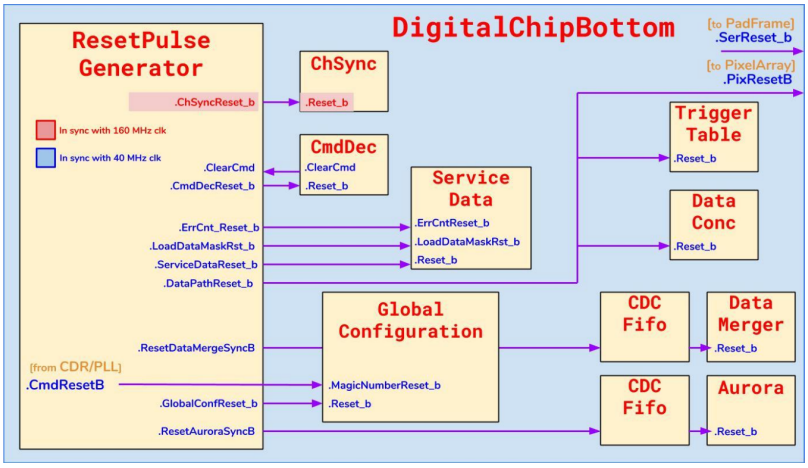

Figure 5: Block diagram of reset signals in the RD53B digital bottom as described in the text.

3.2.1 Chip Default Configuration

The global configuration registers have a synchronous reset in RD53B, whereas they had an asynchronous reset in RD53A. The reset operation sets the registers to the default values as it did in RD53A (with the difference that the default is now low power). However, because the reset is

now synchronous, it cannot be guaranteed that the registers will be at the default value immediately upon power-up. Furthermore, the generation of the clock needed for the synchronous reset depends on biases that are controlled by configuration. For these reasons, in order to ensure that the default configuration is present immediately upon power-up or upon CMD idle reset, there is a hard-wired configuration parallel to the registers that is selected by a multiplexer without any clock requirement. Thus there are in two configurations at all times, a default hard-wired one and a programmable one stored in the global registers, as indicated in Fig. 6 by the MUX symbols following each register. Only one configuration is active at any given time. The control of this multiplexing is a logic level. This mechanism was already implemented for the pixel configuration of RD53A (where it has been preserved for RD53B). But unlike the pixel matrix, where the programmed configuration has no reset at all, the global configuration has both the MUX and a synchronous reset, so that when the MUX selects the hard-wired configuration the programmed values will soon also reset to equal the hard-wired values (as long as there is a clock).

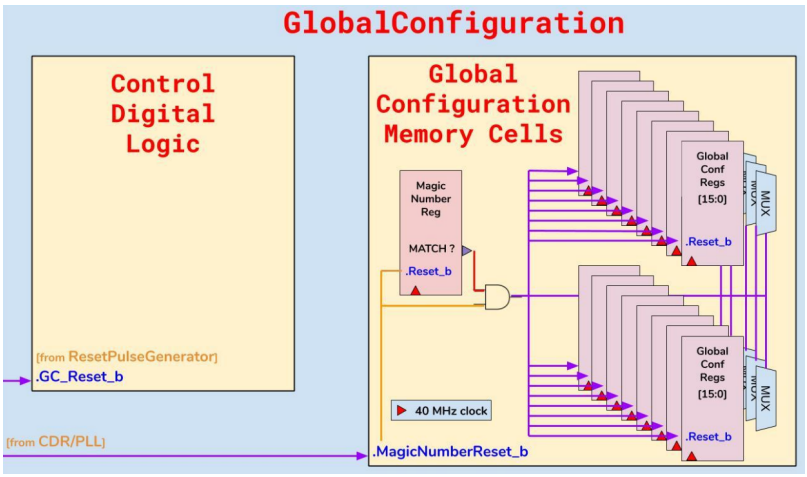

Figure 6: Configuration selection and reset.

The global configuration select level is controlled by both the CMD idle reset signal and a logic comparator that compares the value stored in a special configuration register (with 32 bits, or two 16-bit registers) to a hard-wired key code or "magic number" (labeled MagicNumberReg in the figure). When the stored value does not match the key the output is low and the default (hard-wired) global configuration is selected. At power up, the register will contain something arbitrary, and soon after it will be reset to zero as soon as clock edges are present. Neither case will match the key code. The user must write the key into the register in order to switch to the programmed configuration.

An SEE can put a glitch in the MUX control level and that will cause the active configuration bits to switch between hard-wired and programmed for the duration of the glitch. This is relatively benign and not persistent: after the glitch everything will be in the original state and there is no corruption of the stored configuration. Most configuration bits control DACs, which have a slow response time and will therefore not propagate a glitch in their control bits to their output analog level. Furthermore, many DAC values will be close to their default, which means glitches in the selection do not actually do anything. For further SEU protection, the matching of register contents

to the key value allows for one bit to be wrong (any single bit flip).

3.2.2 Command Activity Detector

The command activity detector circuit essentially rectifies the incoming command signal. When CMD is an AC signal (with frequent edges) the rectified output will be high, and when it is a DC signal it will be low. The circuit bandwidth is low enough so that it takes of order 2 μs after the command goes DC for the activity detector to switch. I will take of order 0.5 μs to release the reset once command line starts switching again.

The activity detector is part of the PLL/CDR block. It directly resets the PLL, which means it puts it back into frequency lock mode.

This is the main reset mechanism of the RD53B chip- conceptually equivalent to cycling the power in a typical system. This is because in a serial chain, cycling the power is truly an action of last resort that should never be needed.

4. Power

4.1 Shunt LDO Regulator

The Shunt LDO regulator (ShuLDO) regulator is a combination of a linear Low Drop-Out voltage regulator (LDO) and a shunt element. The goal is to provide constant current operation with multiple devices connected in parallel (which is not possible for conventional shunt regulators). The circuit was invented as part of the FE-I4 chip development [3], but the design has evolved significantly towards the final implementation in RD53B. There are significant additions and improvements relative to RD53A, including redesigned startup and references (Sec. 4.2), low power mode (Sec. 4.3), "under-shunt" transient protection (Sec. 4.4), and overvoltage protection (Sec. 4.5).

The basic principle of operation of the circuit can be explained using Fig. 7. A conventional LDO voltage regulator is used to power the external load as usual, plus an internal load in parallel. This internal load (referred to as the shunt element) is actively controlled to achieve the desired behavior at the input, no matter what the external load does. To first order, the desired behavior is $I_{in} = I_{ext} + I_{int} = constant$.

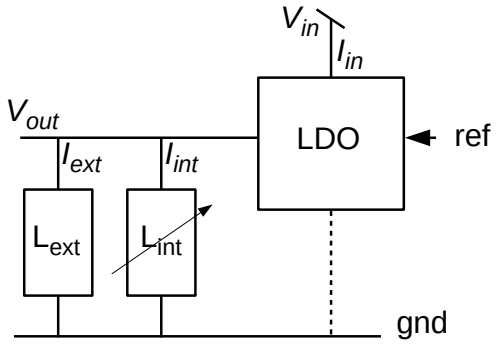

Figure 7: Concept of ShuLDO operation as a linear regulator (LDO) powering an external load and a variable internal load.

The real needed behavior for serial power operation is more complex in order to achieve efficient current sharing among parallel chips and is given by Eq. 4.1.

$$I_{in} = I_{ext} + I_{int} = (V_{in} - V_0)R_{\text{eff}} \qquad [V_{in} > V_0] \tag{4.1}$$

where V_0 is a constant but programmable offset needed for high efficiency and R_{eff}, also user programmable, gives an ohmic behavior necessary to share current evenly among parallel devices. A diagram of the desired behavior is shown in Fig. 8.

The simplified circuit schematic of the RD53B ShuLDO is shown in Fig. 9. The red part of the circuit is a classic LDO regulator with pass device M1. The rest of the circuit can be disabled in order to operate in pure LDO mode, which is useful for testing individual chips and for observing the current consumption. The load external to the ShuLDO is not shown (it is the chip itself). Device M4 is the internal load of Fig. 7 and the rest of the black circuitry is the active control. This

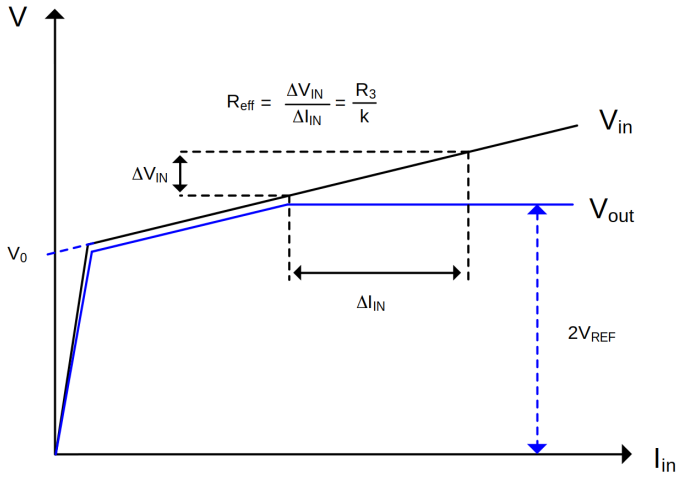

Figure 8: Desired current vs. voltage characteristics for ShuLDO. The unregulated input voltage and regulated output voltage are shown. Indicated values are discussed in the text. R3 is also known as R_{shunt}.

control ensures that the current in the pass device M1 is equal to the the current through R3 times the scale factor K, which has a design value of 1000. R3 is an external resistor (also known as R_{shunt}) to allow the user precise control R_{eff} (internal resistors are also provided for testing convenience, with dedicated wire bond pads, but not expected to be used in operation- see Table 13). It can be seen that $R_{eff} = R3/1000$. Finally, the blue circuit provides the offset V_0. This is controlled by a reference voltage labeled V_{ofs}. User control of V_{ofs} is described in Sec. 4.3.

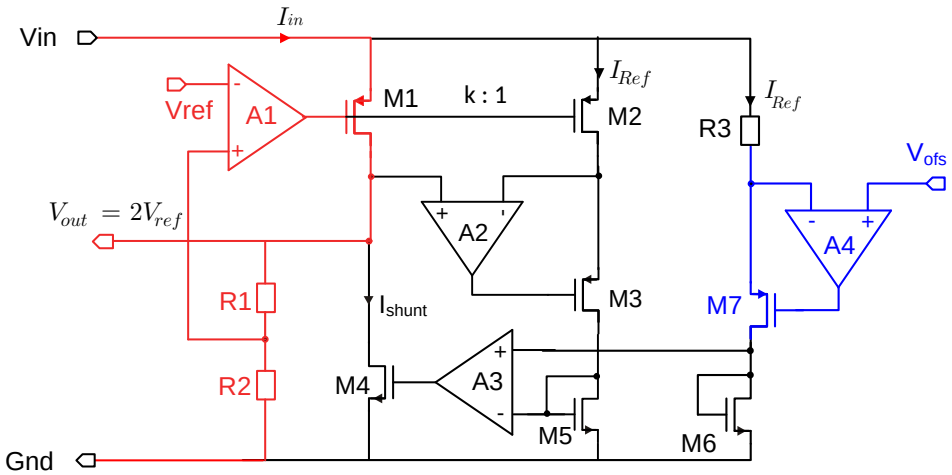

Figure 9: Simplified schematic RD53B ShuLDO regulator. The colors differentiate the LDO (red), shunt (black), and offset (blue) functions as discussed in the text. R3 is also known as R_{shunt}.

The ShuLDO circuit is designed to be compatible with 2 V input voltage, Even during transients, no gate in the circuit will have more than the allowed maximum of 1.32 V across it. All

transistors are cascoded in order to always more than two transistors between voltage supply and ground, with supply voltage distributes across several transistors. Device voltage limits checks in static and dynamic simulations show that transistor sees more than 1.32 V across any two terminals. The one exception is the pass device M1, where cascoding would cause higher drop-out voltage and therefore higher power consumption. The ShuLDO circuit also uses a Low-ESR output capacitor compensation scheme, such that careful control of the external component equivalent series resistance (ESR) is not necessary.

4.2 References and Startup

For serial chain operation the ShuLDO must become active immediately upon current flow, before communication is possible. Once operational it must work with high efficiency and uniformity among chips in the chain, and it must support a low power mode for detector integration. These requirements often conflict and complicate startup. Furthermore, startup must work reliably over a wide temperature range, from room temperature for bench testing and wafer probing, to the evaporative cooling base temperature (taken to be -40°C, that may be reached before power is applied. The generation of current and voltage references is intimately connected to the startup behavior.

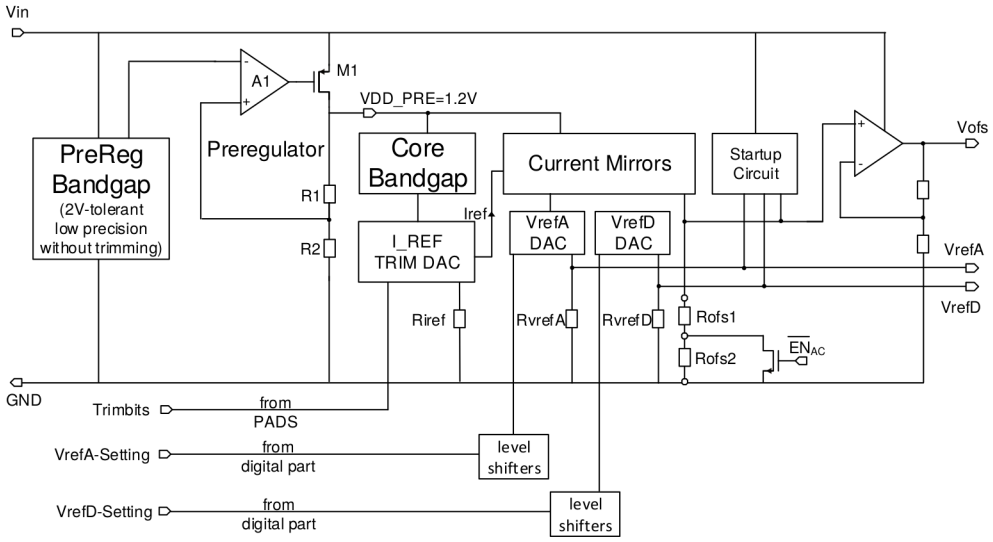

Figure 10: Generation of references and startup circuits. See text for description.

The RD53B reference scheme is fundamentally different form RD53A. An effort was made to remove circular connections in order for more robust startup. The RD53B scheme is shown in Fig. 10. RD53B does not use the ShuLDO output to power any reference circuit. A new low current linear regulator (the *preregulator*) is used to power the master reference current generator. All other references are then derived from this unique master reference current. The preregulator is outside of and in parallel to the chip power concept of Fig. 7, but because it is low current it does not noticeably alter the behavior of Eq. 4.1 and Fig. 8. The preregulator includes its own dedicated bandgap voltage reference, which does not need to be very precise, as the preregulator output does not need to be exactly 1.2 V, but merely between 1.1 V and 1.32 V.

The Core Bandgap generates the master reference current, which can be adjusted with a 4-bit trim set by wire bonds as in RD53A. This allows to compensate for process variations. The generated master reference current also depends on a resistor value, which was internal in RD53A but has been made external in RD53B, in order to avoid the temperature variation of internal devices. The two LDO reference voltages, V_{ref} analog and V_{ref} digital, are each generated by a known current (derived from the master reference) across a dedicated external resistor. Each V_{ref} is independently adjustable by configuration to allow some fine adjustment of the chip internal operating voltage. V_{ref} adjustment does not change the V_{in} vs. I_{in} behavior of the chip, making sudden jumps due to configuration upset or operator error harmless for serial chain operation. The offset reference $V_{\rm ofs}$ is common to both ShuLDOs and is not adjustable by configuration, as sudden jumps in $V_{\rm ofs}$ would be problematic for serial chain operation.

4.2.1 Offset Voltage Start-up

While circular dependencies requiring start-up circuits have mostly been removed from RD53B, the offset voltage does require a startup, because a very low offset voltage will cause the shunt device (M4 of Fig. 9) to be fully on, and this will clamp the input voltage, V_{in}, to a low value even if a large current is supplied. This is especially critical for low power mode (Sec. refsec:vofs). The built in start-up circuit shown in Fig. 11 boosts the offset voltage to follow V_{in} until V_{in} is high enough for the pre-regulator to work and all references to be at their correct values. The circuit injects a current into the offset voltage setting resistor until the pre-regulator reference voltage rises. The rise of the pre-regulator reference shuts off this startup circuit. It is also possible to override this circuit from a dedicated wire bond pad.

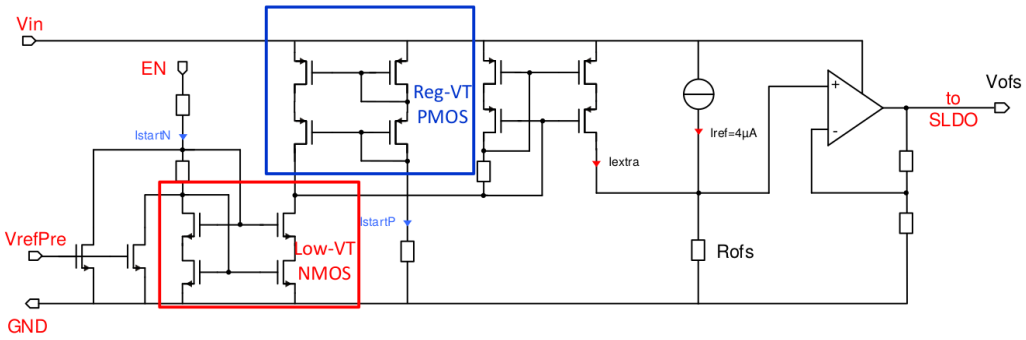

Figure 11: Offset voltage startup circuit.

4.3 Low Power Mode

The ShuLDO offset voltage plays critical roles. It is the most important voltage for regulators operating in parallel, because the total current in a given ShuLDO, I_{in}, is very sensitive to V_0 (and therefore to $V_{\rm ofs}$), as can be seen by the dI_{in}/dV_0 derivative to Eg. 4.1, which is -1/$R_{\rm eff}$. This is also true for dI_{in}/dV_{in}, but V_{in} is by construction equal for all ShuLDOs wired in parallel. On the other hand, $V_{\rm ofs}$ is not necessarily equal for different chips in parallel. While a small value of $R_{\rm eff}$ will make a single ShuLDO more efficient (lower voltage drop between V_{in} and V_{out}), it can make a multi-chip module less efficient by amplifying a small $V_{\rm ofs}$ chip-chip mismatch into a large current

imbalance. Two solutions to this problem are possible in RD53B. First, it is possible to trim the master current reference to produce a target $V_{\rm ofs}$ value, rather than to produce a target current value. This will result in a larger chip-chip variation of reference current, but since all internal biases are adjusted with dedicated DACs this is not a problem. In return for the larger variation of reference current there will be a smaller variation in offset voltage. The second solution is to tie together the $V_{\rm ofs}$ outputs of all chips in the same module via resistors (Fig. 12). For this purpose, in RD53B the $V_{\rm ofs}$ output of Fig 10 and the $V_{\rm ofs}$ input of Fig 9 are on separate wire bond pads.

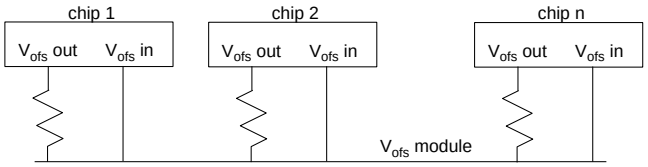

Figure 12: Common offset voltage wiring option for a multi-chip module.

The common $V_{\rm ofs}$ wiring of Fig. 12 is robust against chip failure. Should one of the $V_{\rm ofs}$ outputs be grounded, the common $V_{\rm ofs}$ will be reduced. This will cause the working chips to draw more current for a given V_{in}, which is actually beneficial in the case the failing chip draws low current, as the working chips must now carry the extra current from the failing one.

Rather than a single external resistor to set $V_{\rm ofs}$, Fig 10 shows two resistors with a center tap switch- effectively a 1-bit variable resistor. The switch is internal in RD53B, while the resistors are external. This allows implementation of a low power serial chain mode. For normal serial chain operation the switch is conducting and the resistor value is just $R_{\rm ofs1}$. When the switch is off, the resistor becomes $R_{\rm ofs1}+R_{\rm ofs2}$, leading to a higher $V_{\rm ofs}$. A higher $V_{\rm ofs}$ means that a small current will develop a high enough value of V_{in} needed for the ShuLDO output to reach 1.2 V. This also requires the default configuration to be very low current, as is the case in RD53B. The switch is controlled by a dedicated A/C input as this mode is only intended for use during detector construction, when additional contacts can be made. An A/C signal on this special input will turn the switch off, and the absence of a signal (as will be the case for normal operation) will leave the switch conducting. Note that if this high $V_{\rm ofs}$ low power mode is never needed one can simply leave out $R_{\rm ofs2}$ and connect $R_{\rm ofs1}$ to ground, in which case it no longer even matters what the state of the switch is.

4.4 Under-shunt Current Protection

The variable internal load of Fig. 7 can act to keep the total current constant as long as the external load current draw I_{ext} is less than the programmed total current I_{in}. If due to an error or fault condition $I_{ext} > I_{in}$ there is nothing the variable load can do to prevent the total current from exceeding I_{in}. The under-shunt circuit acts to prevent the $I_{ext} > I_{in}$ condition. It is different from a classic current limiting circuit, because the programmed value of I_{in} is not fixed in advance, but set by an external resistor. Thus it is not possible to have a hard-wired absolute current limit.

Turning around Eq. 4.1, $I_{int} = I_{in} - I_{ext}$, where I_{int} is the internal shunt current in M4 of Fig. 9. Preventing $I_{ext} > I_{in}$ can be done by maintaining a non-zero shunt current,$I_{int} > 0$. The under-shunt protection compares a scaled replica of the M4 current to a threshold (which does not have

to be precise), and it goes below threshold (known as the under-shunt condition) it reduces V_{ref} (by reducing the current it is derived from). Lowering the voltage powering the load will reduce the load current I_{ext}. The circuit is shown in Fig. However, the V_{ref} is not allowed to drop below 0.35 V, to avoid the possibility of a voltage greater than 1.32 V across M1 of Fig. 9, which could cause permanent damage to the device.

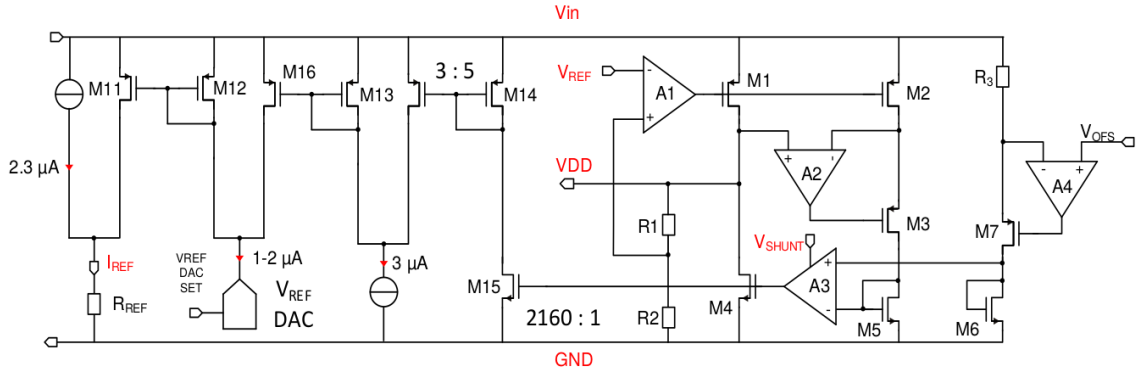

Figure 13: Under-shunt protection circuit.

The under-shunt protection is disabled by default and must be enabled in the global configuration. It can prevent internal shorts from being visible outside the chip, as long as their effective resistance is greater than 0.7 V/I_{in}. It can also prevent transient "shorts" (for example due to a simultaneous firing all comparators or a wrong configuration setting) from drawing more than the programmed I_{in}, but it has a relatively slow response time so as not to interfere with normal voltage and current regulation. So fast internal transients can still be visible in I_{in}. Simulations of selected test cases show that the under-shunt protection generally mitigates both DC shorts and transients, but it can also lead to internal oscillation when the reduction of the load voltage removes the under-shunt condition, but then the condition returns when the load voltage recovers. These internal oscillations are not expected to be a problem for the system outside the chip. Ultimately, the use or not of under-shunt protection will have to be informed by system tests.

4.5 Over-voltage Protection

In the ShuLDO design the shunt element M4 of Fig. 9 is placed after the pass device M1. The total current draw is limited by the pass device and additional current cannot be shunted by M4. Therefore, classic over-voltage protection (OVP) is implemented with a current clamp in parallel to the ShuLDO, rather than within it. Since the voltage being clamped is V_{in}, which is common to both ShuLDOs, there is only one single clamp for the whole chip. The circuit is shown in Fig. 14.

The OVP must only become active if the input voltage is close to 2 V. This threshold must be set by a dedicated internal reference that must be absolutely safe. Otherwise OVP may become enabled prematurely and interfere with chip start-up. Also if the threshold is too low (due to lack of precision, temperature, radiation damage, etc.) the clamp could prevent operation at a valid input voltage such as 1.8 V. For these reasons, in RD53B it is possible to override OVP with a wire-bond pad. System tests will determine if there are any issues with use of OVP.

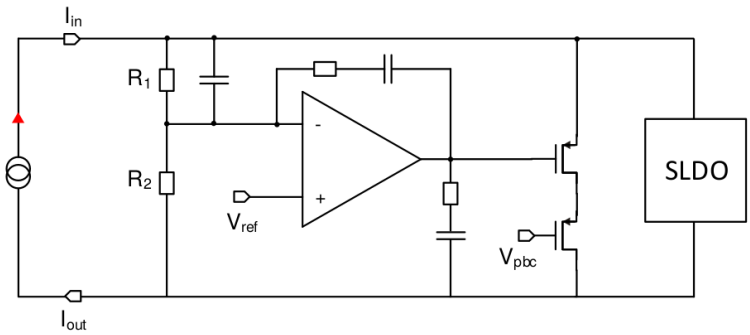

Figure 14: Over-voltage protection clamp.

5. Analog Front End (ATLAS)

The front end (FE) for ATLAS is based on the Differential FE of RD53A, with modifications in the precomparator and comparator as noted. The FE is a pure analog circuit: it contains no memory latches, flip-flops or counters. Static configuration values are provided by the digital core, which receives only the comparator out signal from the analog part. The design is a small-area, low-power, free-running amplifier and discriminator for negative input charge. It contains a single ended charge integrator feeding a differential second stage and comparator. The ADC function is implemented entirely in the digital core, by digitizing the time-over-threshold (ToT) of the comparator output pulse. Fig. 15 shows the FE block diagram.

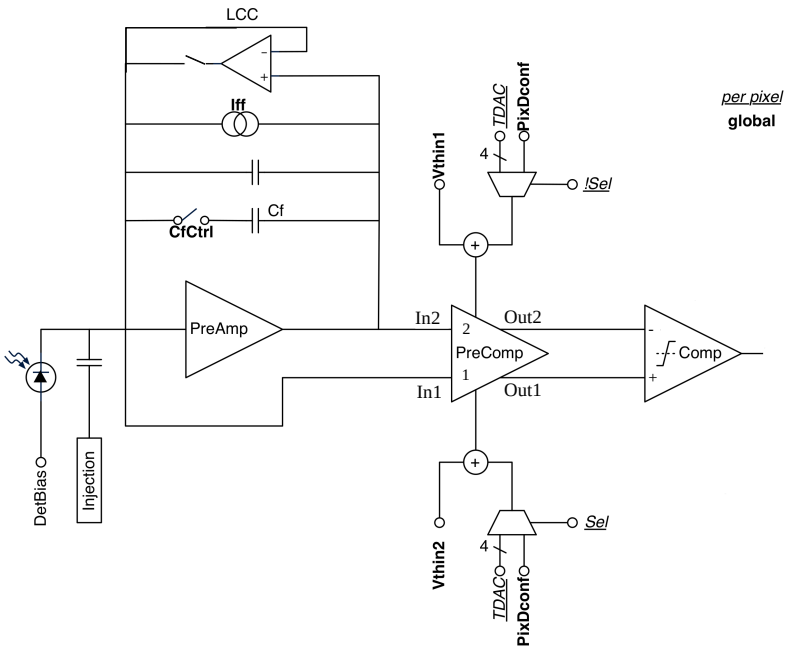

Figure 15: Schematic of analog front end with differential second stage and comparator.

The pre-amplifier (preamp) or first stage is exactly the same circuit as in RD53A (Fig. 17). It has a straight regulated cascode architecture with NMOS input transistor in weak inversion. A high-VT type input device is used. It has a continuous reset and adjustable gain by adding or not a feedback capacitor, cf0 (this choice is made globally, not per pixel). The preamp can operate at very low currents and has two biases: the main bias (input transistor current), and the continuous reset feedback current. The feedback current is set globally and cannot be trimmed in each individual pixel. The preamp is single ended, but the feedback ensures that, in the absence of signal, input and output are at the same potential. Input and output are thus taken as a differential input to the next stage. The left and right two columns of pixels, the top two rows and the top left and top right corners, each have their own bias setting for the input transistor current, separate from the setting of the internal part of the matrix. This is illustrated in Fig. 16. All other biases are common to the whole matrix.

A leakage current compensation circuit (LCC) provides additional optional feedback in the

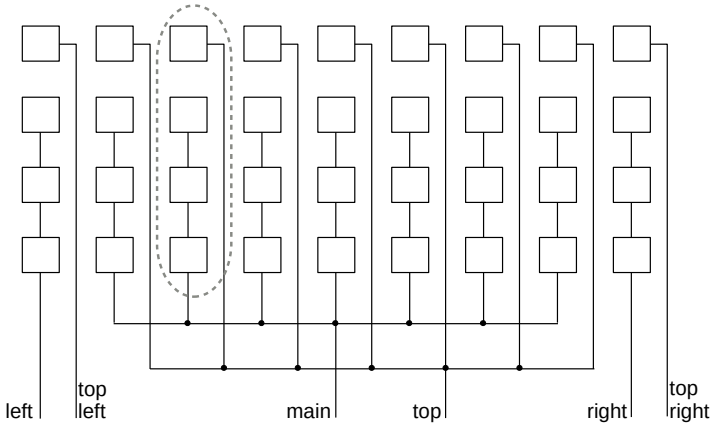

Figure 16: Distribution of preamp bias (input device current). Each square represents a 2 by 2 pixel analog island. Six different DACs at the chip bottom control different lines as indicated. An arbitrary column of analog islands is circled, to show that all columns are identical, with two bias lines. The distinction between center, sides, top, and corners is made by which DACs they connect to in the chip bottom. DAC's can be programmed all the same or different, as needed.

preamp. It is normally off and should only be used in case of large sensor leakage current (3 nA/pixel or higher). It is an active low pass filter, shown in Fig. 19. Because it is a low pass filter, it can sculpt the response to periodic burst injections as done in calibrations, so calibration scans to be run with LCC enabled should keep this in mind.

The DC-coupled pre-comparator or second stage maintains the same schematic as RD53A (Fig. 18), but two transistors (those connecting to Out1 and Out2 in the figure) have been resized. Their length has been reduced to increase operating margin at low temperature at the expense of reduced gain (from 3 V/V to 2 V/V). This has a negligible effect on overall performance, as the main function of the pre-comparator is to introduce the threshold, not to add gain. The global threshold is adjustable through two distributed voltages (VTH1 and VTH2) which introduce an offset between the two branches of the pre-comparator. The left and right analog island columns (including the corners) have dedicated VTH2 threshold settings. This, VTH2 is set in common for the whole chip, while VTH1 can be set separately for left, middle, and right (no top row distinction). The threshold is trimmed in each pixel using one 4-bit resistor ladder in each pre-comparator branch. The branch current is turned into additional voltage offset by these resistor ladders. An effective 5-bit adjustment is obtained by adjusting one branch resistance or the other using a single 4-bit value. The 5^{th} bit is thus a select bit, which determines which branch is adjusted. The branch that is not adjusted is set to all 1 or all 0, depending on a global configuration value.

The FE design is optimized for low-threshold operation. The pseudo-differential design reduces variation due to mismatch and provides improved power supply rejection. The pre-comparator stage is followed by a classic continuous time comparator stage (Fig. 20 with output connected to the digital pixel region through a buffer placed in close proximity to the comparator output. The polarity of the comparator has been flipped relative to RD53A, by taking the single output from the

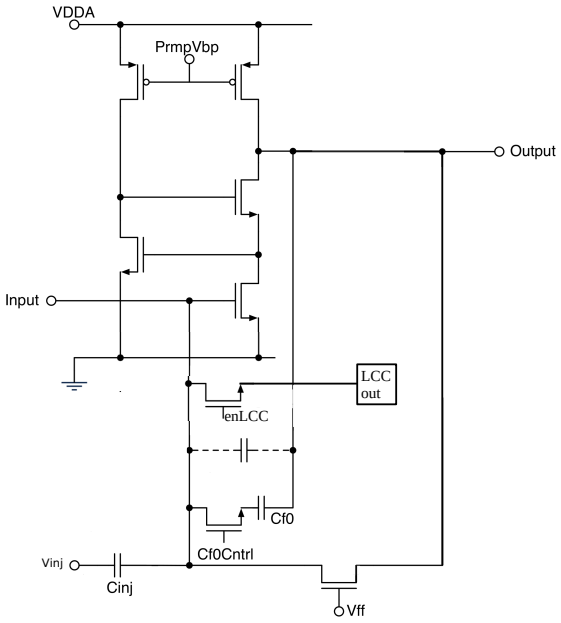

Figure 17: First stage (preamp) schematic. LCC is a Leakage Current Compensation circuit. The cf0 capacitor to enable low gain mode has a value of 3.28 fF. In high gain mode cf0 is off and the only feedback capacitance is parasitic, with a value of 3.73 fF.

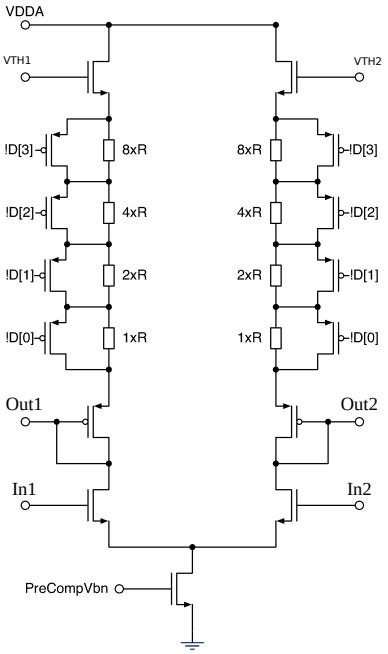

Figure 18: Second stage (pre-comparator) schematic. The threshold is introduced in this stage by offsetting the two branches using resistor ladders. The preamp output is connected to In2 and the preamp input to In1.

other differential branch than RD53A. Thus, when a negative input charge hit fires the RD53B front end, the comparator output switches from low to high. This has two advantages relative to RD53A (which switched from high to low). (1) The slew rate is higher, because it is limited only by the gain of PMOS transistor in the output stage, whereas the falling edge slew rate is limited by the bias current in the NMOS transistor, which is purposely small to save power. (2) The quiescent state of the output is low. Therefore, if the analog supply rail happens to be much lower than the digital supply rail (for example during an under-shunt current transient triggering the internal protection) this will not cause a digital transient, as would be the case if the quiescent state was high.

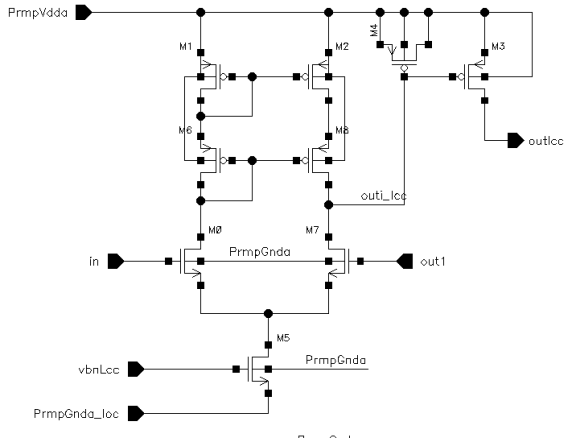

Figure 19: Leakage current compensation circuit (LCC). The high impedance ports "in" and "out1" are connected to the preamp input and output, respectively. The LCC output is connected via a switch to the preamp input as shown in Fig. 17

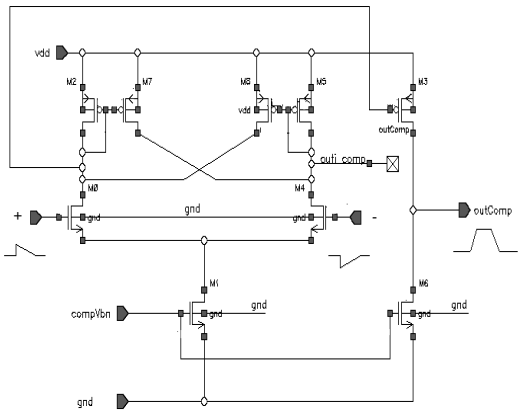

Figure 20: Comparator schematic. Note the a single bias is used for the tail current in the comparator proper as well as the output stage current.

6. Calibration Injection

This pixel circuit and the control signals have been kept the same as in RD53A. However, the generation and distribution of the DC calibration voltages has been improved (Sec. 6.3), and the Cal command has been modified (Sec. 6.2). The calibration injection circuit uses two distributed DC voltages plus in-pixel switches to chop them and generate steps fed to an injection capacitor. Having two voltages allows a precision differential voltage that will be independent of local ground drops in the chip, as well as two consecutive injections into the same pixel. The circuit topology is shown in Fig. 21. Every pixel contains this circuit. The control signals, S0 and S1, are generated in the digital domain as explained in Sec. 6.1 and can be phase shifted relative to the bunch crossing clock with a fine delay, which is global for the whole chip. The enable bit is programmable for each pixel. The cascaded "inverter" configuration makes for simple control without the need to synchronize non-overlapping edges. Injection takes place only for cal enabled pixels when either S0 or S1 switch from low to high. Analog injection must therefore be primed by setting at least one control signal low, prior to being able to inject. This priming is not automatic, so that the user is able to control the amount of settling time allowed prior to injection. The CAL command is used for both functions: prime and inject (see Sec. 6.2).

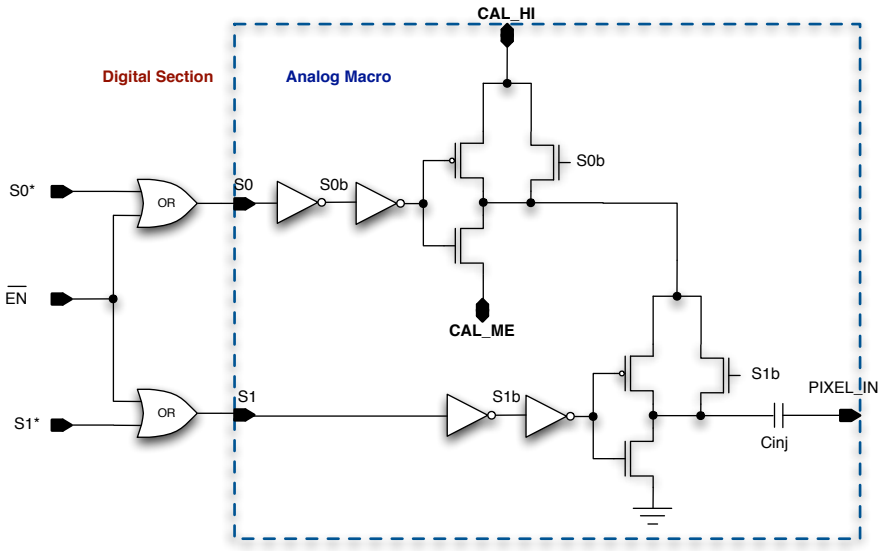

Figure 21: Calibration injection circuit in each pixel. The injection capactior nominal value is 8.5 fF.

Just as in a common inverter, there will be a switching transient when a control signal switches from low to high. Simulations show these transients to have a negligible impact on the distributed CAL_HI and CAL_MED voltages. Note that at the top of each inverter there is an NMOS transistor in parallel with the PMOS, which switches first (before the PMOS) when injecting. This allows the switches to operate for any choice of voltages CAL_HI>CAL_MED>GND, but since the top NMOS switches first, it does not contribute transients during injection. During priming, on the other hand, the top NMOS switches first, while the bottom NMOS is still conducting, resulting in a

short circuit lasting one inverter delay. This will cause a transient on the CAL_HI and CAL_MED voltages, and the user must therefore allow some settling time between priming and injection. In addition to this transient, priming injects a positive polarity pulse into each enabled front end, so one must allow for the front end to settle in any case. The two-voltage injection circuit allows injection of two successive pulses without priming in between, and with arbitrary delay between these pulses. This was a new feature introduced in RD53A, not found in previous chips. The two voltages also mean that the charge injected by S0 is given by a differential voltage and not affected by local ground potential differences. Keeping S1=0 and only toggling S0 will result in single pulse differential injection. The two voltage system also makes it possible to inject different amount of charge simultaneously in neighboring pixels by changing the meaning of S0 and S1 in different pixels (see Sec. 6.1.

6.1 Generation of S0 and S1 signals

The signals S0 and S1 of Sec.6 exist locally in each pixel but are derived from different internal signals produced by the command decoder and distributed to the array. This two-step scheme is necessary in order to have either sequential or simultaneous injection, and more importantly to avoid having to distribute two switching signals with precise timing. Since the calibration input is used to study and calibrate timing, it must occur simultaneously in all enabled pixels, just as is the case for the bunch crossing clock (here, simultaneously means within a 2 ns window). Two control signals are distributed: CAL_edge and CAL_aux. As the name implies, CAL_edge needs to be simultaneous in all pixels, while CAL_aux does not. CAL_edge has a fine phase adjustment relative to the beam crossing clock, which is called CAL_delay. In fact CAL_edge looks like a traditional injection pulse, with user controlled leading edge time and duration. In *uniform* injection mode (which allows injecting two pulses close in time into all selected pixels), S0 and S1 are derived from CAL_edge and CAL_aux identically for all pixels:

$$\mathrm{S0} = \mathrm{CAL\_edge}\ \ OR\ \ \mathrm{CAL\_aux} \tag{6.1}$$

$$\mathrm{S1} = \overline{\mathrm{CAL\_edge}}\ \ AND\ \ \mathrm{CAL\_aux} \tag{6.2}$$

The rising edge of CAL_edge throws the S0 switch, while the falling edge throws the S1 switch. The CAL_aux starts low and then goes high after CAL_edge, but not with precise timing. In uniform mode the injection switches can only be thrown in that order. Either only use S0 for single pulse, differential voltage injection, of use S0, then S1, for double pulse injection.

In order to allow injection of different size pulses simultaneously into adjacent pixels, there is an *alternating* analog injection mode that can be selected instead of the default uniform mode. In this mode the S0 and S1 signals are derived as above only for *even* pixels, but swapped for *odd* pixels:

$$\mathrm{S1}_{odd} = \mathrm{CAL\_edge}\ \ OR\ \ \mathrm{CAL\_aux} \tag{6.3}$$

$$\mathrm{S0}_{odd} = \overline{\mathrm{CAL\_edge}}\ \ AND\ \ \mathrm{CAL\_aux} \tag{6.4}$$

where an even (odd) pixel is one for which the sum of row + column is an even (odd) number. Thus, for example, in single injection mode the CAL_edge rising edge throws S0 for even pixels, but S1 for odd pixels. The S0 and S1 assignment options are independent of the cal enable bit in each

pixel. The Analog Mode bit of the injection configuration controls whether injection is uniform (mode=0) or alternating (mode=1).

6.2 Cal Command

The Cal command controls the generation of the two internal signals CAL_edge and CAL_aux. For digital injection only the CAL_edge signal is relevant. The CAL_edge signal to be generated is specified by the first 14 data bits of the Cal command, while the CAL_aux signal is specified by last 6 data bits. The detailed bit assignment of the command payload (four 5-bit fields) is shown in Fig. 22.

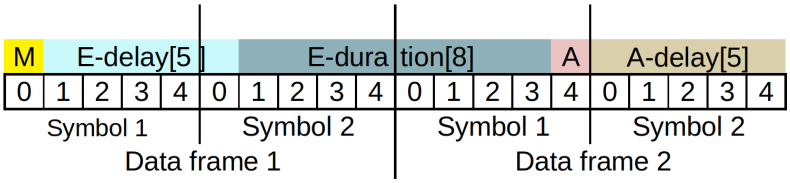

Figure 22: Bit assignment of the Cal command payload. Two data frames totaling 20 bits. M= mode bit, E= CAL_edge parameters, A= CAL_aux parameters (value and delay). All delays and duration are in units of 160 MHz clock cycles.

The CAL mode bit (M) selects between two behaviors for the CAL_edge signal: a singe step (mode=0) or a pulse (mode=1). The step is always from low to high, so an edge is only produced if the prior state of CAL_edge was low; if it was high it remains high. Thus, for typical injection it is necessary to first arm the system to ensure CAL_edge is low, as shown in Fig. 23. The standard injection sequence is then shown in Fig. 24. In this case voltage at the injection capacitor of the selected pixels is switched form CAL_HI and CAL_MED, effectively providing a differential injection voltage that will be insensitive to power and ground local voltage variations across the matrix. The delay value controls the “coarse” delay from the Cal command to the injection (in cycles of the 160 MHz clock), and a global fine delay is added on top of that. This fine delay is pre-programmed in a global register and is in units of 640 MHz clock cycles. This allows precision scanning of the pixel timing. The duration serves no purpose in edge mode and the CAL_aux signal is unchanged. But note that CAL_aux must be low the whole time in order for the rising CAL_edge to switch from CAL_HI and CAL_MED. If the CAL_aux signal were instead high, the switching would be from ground to CAL_edge. This would inject positive polarity charge, which the Front End is not designed for, but may be of interest for special tests.

In pulse mode (M=1) the CAL_edge signal will be set to high after the coarse plus fine delay, just as it happened for edge mode, but then it will be set to low after the duration value elapses. Note that if duration is set to zero then CAL_edge will simply go low after the delay (a duration zero pulse with final state low). Thus, pulse mode with duration zero is the complement of step mode: the former brings CAL_edge low while the latter brings CAL_edge high. Step mode is used to produce two consecutive injections. Starting form the armed state of Fig. 23, the rising CAL_edge will inject from CAL_HI and CAL_MED as usual, but then, before the falling CAL_edge at the end of injection the CAL_aux signal is set high, which then causes the falling CAL_edge to switch form CAL_MED to ground. This sequence is shown in Fig. 25. The CAL_aux signal will be set

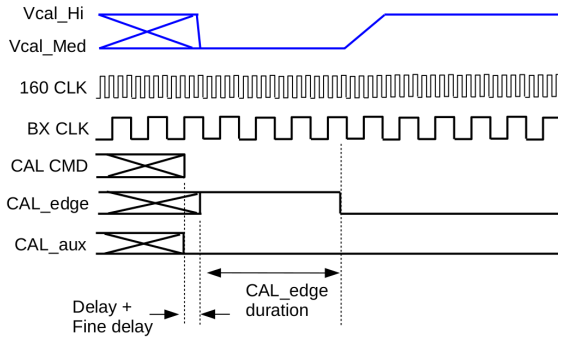

Figure 23: Timing diagram illustrating the arming of calibration injection, to set both CAL_edge and CAL_aux to the correct levels without knowing their prior state. The Cal command parameters are as follows: M=1, delay=1, duration=15, A=0, Adelay=0 (the exact delay and duration values are not important).

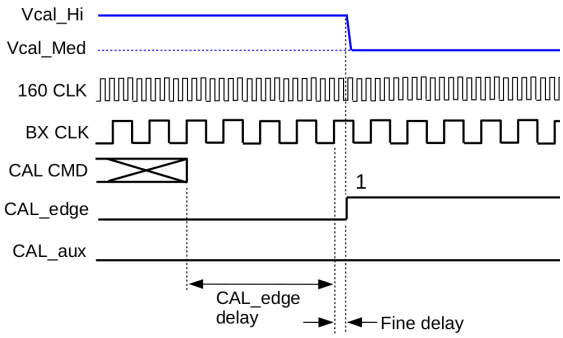

Figure 24: Timing diagram illustrating standard calibration injection. Only the CAL_edge signal is active. The Cal command parameters are as follows: M=0, delay=16, duration=1, A=0, Adelay=0 (the exact delay and duration values are not important).

to the level indicated in the command (A) after the given delay value. The fine scale for the delay allows changing the CAL_aux value in the middle of a bunch crossing cycle, as is needed to inject charge in two consecutive crossings. Since at the end of this sequence CAL_edge is low, it's no longer necessary to repeat the Fig. 23 arming sequence. However, the CAL_aux signal has to be returned low, which can be done with the sequence in Fig. 26.

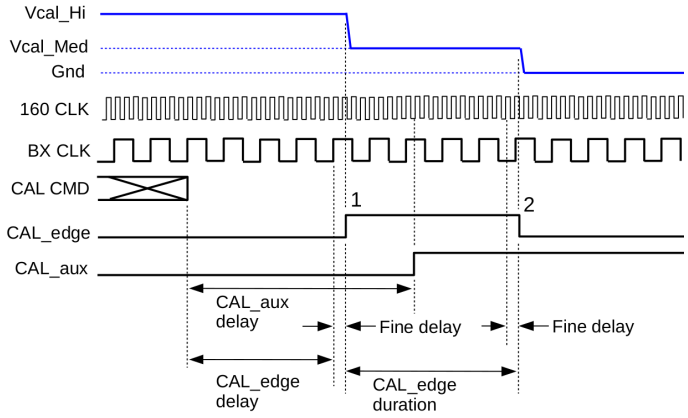

Figure 25: Timing diagram illustrating double calibration injection using the pulse mode of the Cal command. The Cal command parameters are as follows: M=1, delay=16, duration=19, A=1, Adelay=25 (the exact values are representative. The Cal_AUX transition must be between the CAL_edge rising and falling edges).

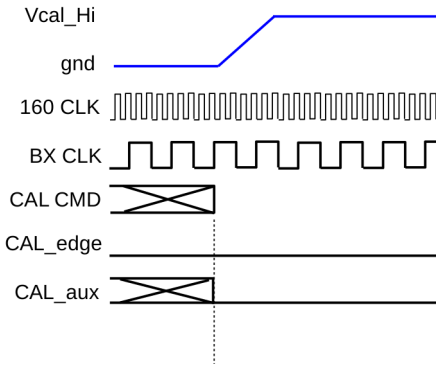

Figure 26: Timing diagram illustrating the re-arming of the CAL_aux signal after double injection. The Cal command parameters are as follows: M=1, delay=0, duration=0, A=0, Adelay=0.

Executing a double injection with a single Cal command in pulse mode is limited the injections being closer together in time than the maximum CAL_edge duration. To perform a double

injection separated by longer times, two separate Cal commands can be used. The first would be a standard injection (Fig. 24), while the second command would accomplish the CAL_MED to ground injection as shown in Fig. 27. Note that this is not the same as two consecutive standard injection sequences, because in between standard injections one must re-arm, which produces a positive polarity injection at the time of re-arming. In contrast, in the double injection there is no activity between the two injections. This can be important when studying threshold stability vs. time, for example. One can achieve the same thing using standard injection, but it requires more commands and, therefore, more time between injections.

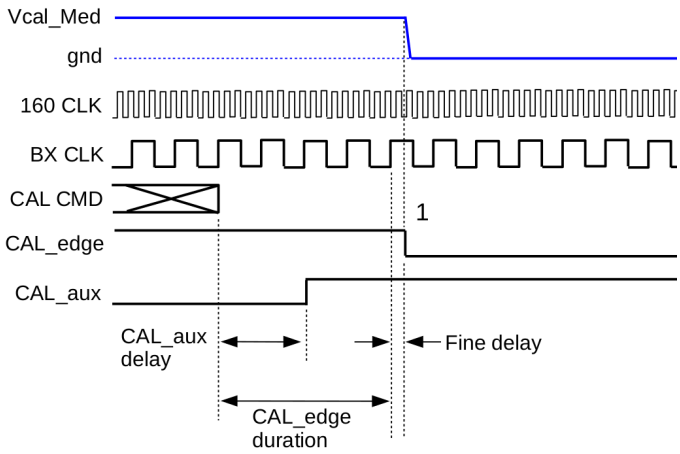

Figure 27: Timing diagram for an arbitrarily delayed second injection in a double injection sequence. The Cal command parameters are as follows: M=1, delay=0, duration=16, A=1, Adelay=8 (the exact values are representative. The Cal_AUX transition must be before the CAL_edge falling edge).

6.3 Injection Voltages

The two injection voltages CAL_HI and CAL_MED are generated by one DACs each in the chip bottom and are common to the whole chip as shown in Fig. 28. The circuit has two ranges to allow either fine step sizes or high dynamic range. Each voltage is generated by a 10-bit DAC. The voltage reference that the injection voltages are derived from is the same as for the ADC (Sec.). As these voltages are relatively high impedance, injection into many pixels at once will introduce a bias, as the current pulse from the combined effect of all injected pixels will cause a voltage shift of the injection voltages. This effect is small and can be ignored for many applications, but should be considered for precision studies.

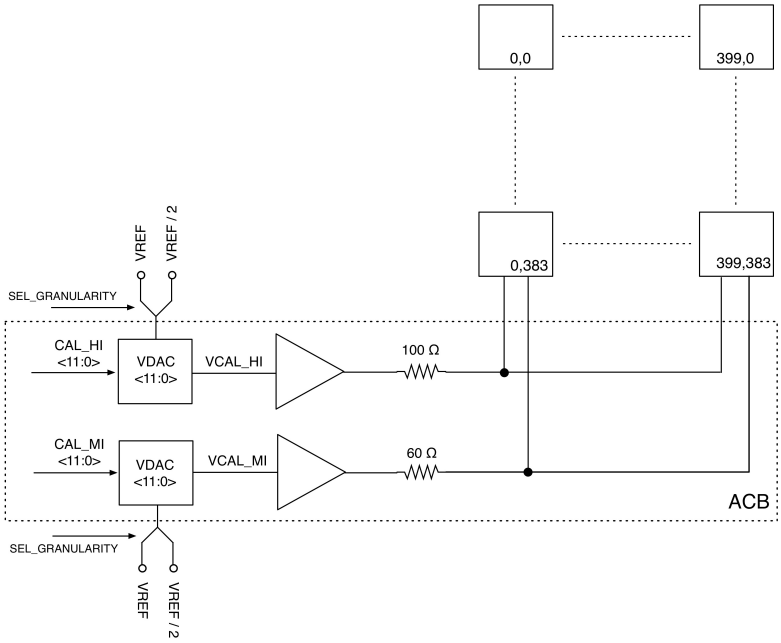

Figure 28: Generation of the injection voltages CAL_HI and CAL_MED.

7. Digital Core

The digital core implements all the functionality of digitization, time stamping, storage, trigger retrieval, configuration memory, and injection control for the 64 pixels in the core. The bunch crossing clock and cal edge distribution do not use a clock tree, but are synchronized though delays that depend on the core address. All cores are identical and the layout is stepped and repeated to form the matrix, and the address of each core is generated by a combinatorial adder that adds 1 to the address of the previous core, starting form a hard-wired seed address of 111111 at the top of the matrix (thus the top-most core has address 111111+1=0).

Within the core, the pixel hits are processed in 4-pixel regions (Sec. 7.1). Even though each analog island also has 4 pixels, the region pixels are organized in 4×1 rows, so two pixels from one island and two from another. The 4 pixels in a region share timing information, while each individual pixel has its own hit discrimination (Sec. 7.2), ToT counter and memory (Sec. 7.3). Additionally the region manages clock gating, which reduces digital power consumption.

7.1 4-Pixel Region

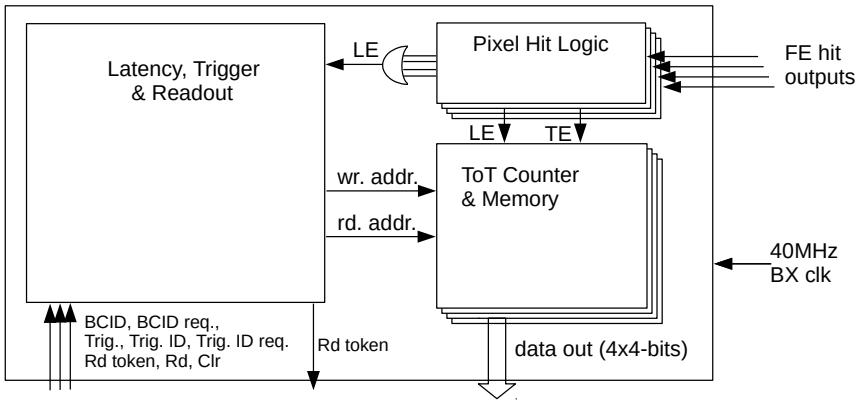

Figure 29: 4-Pixel region block diagram. LE is leading edge, TE is trailing edge, BCID is bunch crossing counter value. See text for details.

The pixel region logic contains three main blocks as indicated in Fig. 29: the Hit Logic (four instances), the ToT counter and storage (four instances), and one Latency, Trigger and Readout (LTR) block. Sharing the LTR block among four pixels leads to a more compact layout than if each pixel was independent.

The Hit Logic (Sec. 7.2) determines if an hit is present in a given bunch crossing and drives the ToT counter (Sec. 7.3). The LTR block (Sec. 7.4) is the brains of the region. It keeps track of the timing of all hits, decides which ToT memories to use, and manages triggering, reading and clearing of ToT data.

7.2 Pixel Hit Logic

Fig. 30 is a schematic of the logic to process the asynchronous hit output of the analog front end,

while Fig. 31 shows the corresponding waveforms. This has changed from RD53A where the analog FE hit output was first synchronized with the clock before processing. It provides greater accuracy for ToT measurement at higher effective clock speed than 40 MHz. The block does *not* contain a method to separate "small hits" from "large hits" as in the FE-I4 chip, because the RD53B front end time-walk is very small, resulting in very few late hits.

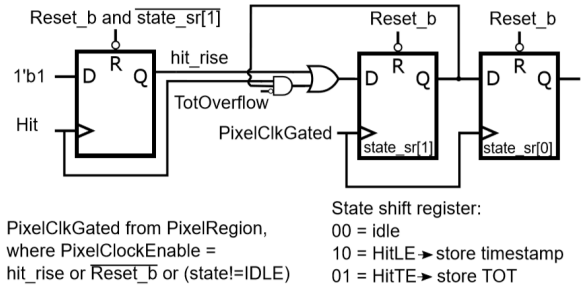

Figure 30: Schematic of the logic to process the asynchronous hit output of the analog FE. The reset_b signal globally provided during initialization.

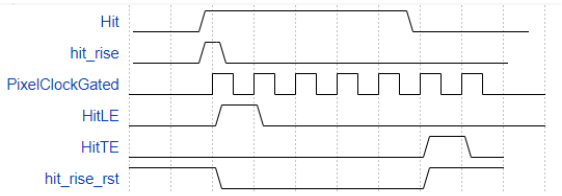

Figure 31: Waveforms showing the processing of a hit output from the analog FE.

7.3 ToT counter and storage

ToT is counted independently for each pixel using the 40 MHz clock, but providing 80 MHz effective resolution by capturing the clock phase with the asynchronous hit. In addition to 80 MHz resolution, the RD53B counter can run dual slope mode where the 4-bits are used to compress the value of a 6-bit counter. In any mode, if the ToT counter reaches maximum while the pixel comparator output is still high, the counting concludes and the maximum ToT is recorded. The region clock is gated off whenever there is no ToT counting taking place. The ToT counter schematic is shown in Fig. 32.

Figure 32: ToT counter schematic.

The ToT storage has 8 locations, 4 bits each. The value of the ToT counter is stored once the conversion is finished, given by a trailing edge pulse or by the counter reaching max count. Which of the memories the ToT is stored in is fixed at the start of the ToT conversion by a select bit from the LTR, shown as wr. addr. in Fig. 29 (there are 8 bits and only one can be high at a given time).

The ToT memory bank has a 4-bit output port. Which memory is presented on this port is given by a select from the LTR block (rd. addr. in Fig. 29).

7.4 Latency, Trigger and Readout (LTR) block

A hit in any of the region pixel starts a latency timer for the whole pixel region. If multiple pixels fire in the same region in the same crossing, still only one latency timer starts.

Each 4-pixel region has 8 latency timers just as each pixel has 8 ToT storage registers. The latency timers are common to the whole region, but the ToT storage is at the single pixel level. Each of the 8 ToT registers of a given pixel is associated with one region latency timer (hard-wired). This way, when one pixel is counting ToT it does not prevent the other pixels in the region from being hit: no region dead time.

The latency timers are not implemented as counters, but as 9-bit memories and comparators, as shown in Fig. 33, which is a lower power solution. When a timer "starts", the 9-bit memory stores the value of a global bunch crossing counter (BCID) distributed form the chip bottom. Every subsequent bunch crossing, this value is compared to a delayed bunch crossing counter (BCID request), also global, that is delayed by the trigger latency relative to the BCID. Both are Gray counters so that only one of the 9 bits changes every bunch crossing.

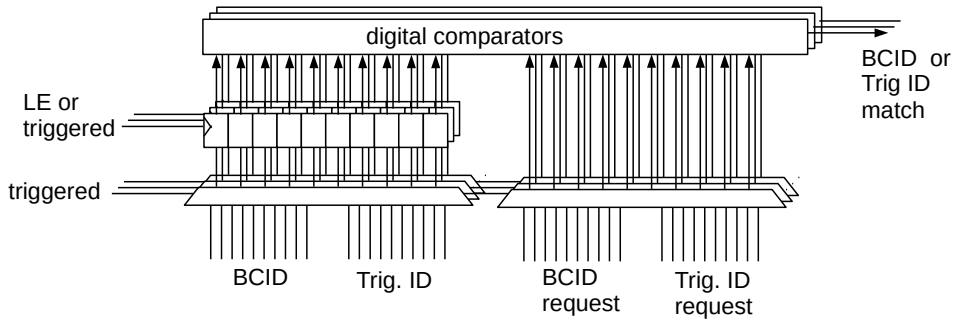

Figure 33: Latency timer diagram. The same memory and comparators are used to store BCID values or Trigger ID values, as there is never a case when both need to be stored at the same time. Only 3 instances of each circuit are shown in the figure, but in reality there are 8 each 9-bit memories and comparators in the region.

The latency timers are 9 bits wide, allowing for a programmed latency of up to 511 bunch crossings. The timers are idle until a LE signal arrives and with each new LE a new timer is started. Only one timer can start in any given bunch crossing regardless of how many pixels were hit. When a timer reaches the programmed latency (BCID match) the presence of a trigger is checked. If no trigger is present, the timer and associated ToT memory are released (marked available). If a trigger is present, then the ToT memories are marked triggered (not available) and a trigger ID is stored to label the hits for later readout.

When a region hit is selected by a trigger, the latency timer mechanism is reused for queuing the hit for readout. The stored BCID value is overwritten with a Trigger ID value, which is compared to a trigger ID request value. When the trigger ID request matches the stored value, the LTR will hold the read token and select ToT data to be placed on the output. The read token travels from one region to the next to scan a core column for data. The generation of the trigger ID is explained in Sec. 9.3.

The Precision ToT modules in the chip bottom reuse the region latency counter mechanism and reading Sec. 13.6 may give further insight about the region operation.

7.5 Pixel Addressing

The pixel addressing has been preserved from RD53A. It is hierarchical, first in cores (like postal codes) and then regions within a core (like the street address). This structure is shown graphically in Fig. 34. In the RD53B data output (Sec. 10.4) the basic unit is a quarter core, which contains two rows of 8 pixels. Thus Fig. 34 is vertically divided into four quarter cores, with the numbering of the quarter cores as shown in Fig. 35. Also shown is an example of the compressed bit codes for all 16 cases of a single hit pixel in the quarter core.

For configuration, the Core_Col and Core_Row values are preserved in registers 1 and 2, respectively, but the four Region_in_Core are divided between the two registers as follows. Additionally, there is a Pair_in_Region bit need (17 total bits instead of 16), because configuration is written in pixel pairs rather than quads.

```
Register 1= [7:2]=Core_Col, [1]=Region_in_Core[0], [0]=Pair_in_Region
Register 2= [8:3]=Core_Row, [2:0]=Region_in_Core[3:1]
```

In this way, Register 1 has the traditional column pair meaning, while Register 2 has the row meaning.

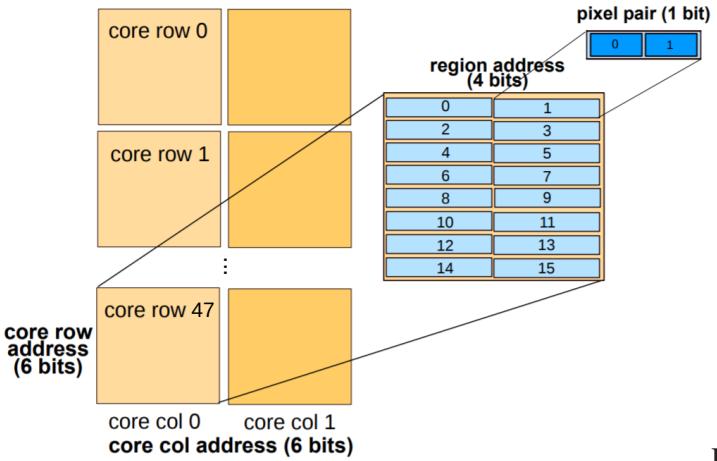

Figure 34: Pixel core addressing scheme

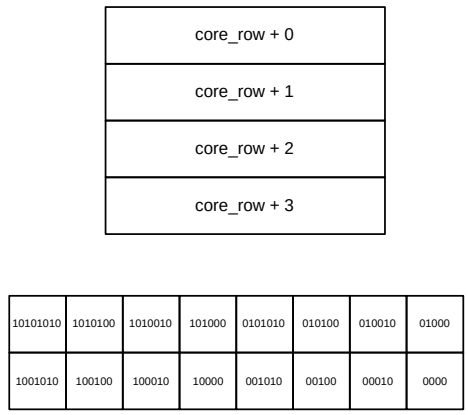

Figure 35: Numbering of quarter-cores for readout (top) and compressed binary code for each single hit pixel (bottom).

7.5.1 25 μm $\times$ 100 μm pixels

While for 50 μm $\times$ 50 μm sensors the chip pixel numbering will carry over unchanged to the sensor, for 25 μm $\times$ 100 μm a mapping is needed to know which sensor pixel is connected to which chip channel. This mapping is determined by the sensor metalization and there are two possible mappings as shown in Fig. 36.

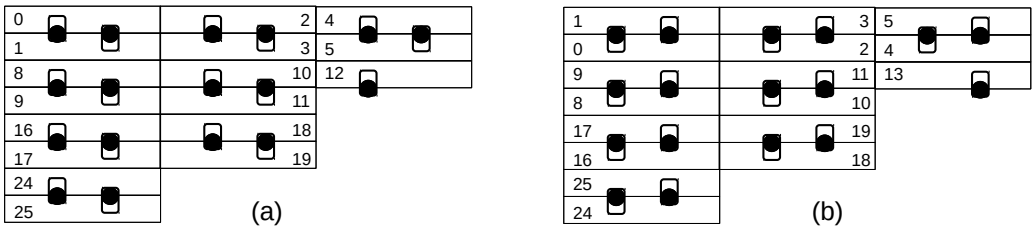

Figure 36: Two options (a and b) for mapping of 25 μm $\times$ 100 μm pixel sensors to the core pixel address. Which option is correct is determined by the sensor metalization. The top left corner of an 8 by 8 pixel core is shown.

8. Commands and Configuration

RD53B is fully controlled with a 160 Mbps differential serial input stream with a custom, DC-balanced encoding described in Sec. 8.2. The differential receiver circuit is described in Sec. 8.1. The received signal, without any processing, can be optionally repeated on the general purpose differential outputs (Sec. 13.1). The input also contains an activity detector, which will cause a reset when no transitions are detected (Sec. 3 and 14). A Clock and Data Recovery circuit (CDR) recovers the input bitstream and also produces the internal clocks for the chip, based on the transitions on input stream, as described in Sec. 14. A dedicated command equivalent to a clock pattern is provided to properly lock the internal phase locked loop.

8.1 Receiver Circuit

8.2 Command Protocol

RD53B uses a slightly modified version of the command protocol implemented in RD53A, such as the addition of the PLL lock command mentioned above. Thus, it is not backwards compatible with RD53A control systems. The command protocol is DC-balanced with short run length for A/C coupling, reliable transmission, and clock recovery. It has built in framing and error detection. These are all the properties that could have been obtained with 8b/10b encoding, but the 10-bit frame length of 8b/10b would have required 200 Mbps link speed in order to maintain an integer number of bunch crossings per frame, as needed for synchronous trigger operation. In contrast, the custom protocol developed runs at 160 Mbps, which makes for better transmission on low mass cables and can be directly driven from GBT e-links.

The protocol consists of a continuous stream of 16-bit frames. Each frame is exactly DC balanced. There is a unique sync frame (for frame alignment), plus 3 kinds of TTC frames: trigger, command, or data. All these frames contain two 8-bit symbols which are themselves DC-balanced. Furthermore, symbols that begin or end with three or more 1's or 0's are not used, resulting in a maximum run length of 4, except for the sync frame which has a run length of 6. The valid symbols and commands are given in Tables 2, 3, and 17. There is one sync frame, 7 non-trigger commands, 15 trigger symbols allowing the encoding of 15 trigger patterns (Tables 2, 3), and 32 data symbols allowing the encoding of 10 bits of content per data frame or 5 bits of chip ID per command frame (Table 17). All valid symbols are allowed to be used as trigger tags in the trigger frame; thus there are 54 possible tags (see Sec. 9). A single bit flip always results in an invalid symbol (formally, all symbols are separated by a Hamming distance of 2).

Because one 16-bit frame spans 4 LHC bunch crossings, the trigger commands must specify a 4-bit map indicating which of the 4 bunch crossings are actually triggered; hence 15 trigger patterns. The triggering is synchronous, and therefore trigger frames must be sent at specific times. All other frames "fill the spaces" between trigger frames. It is recommended that one sync frame be inserted every 32 or so frames to maintain lock. The the second symbol in a trigger frame is interpreted as a 6-bit tag base to identify the trigger(s) in later readout. The trigger tag will be returned with the data corresponding to that trigger (See Sec. 10).

RD53B interprets the protocol in three phases (which will be transparent to the user): Initialization 8.3, Data Transmission 8.4 and Decoding 8.5. The decoding timing and exception handling are covered in Sec. 8.6

Command	Encoding		(T)ag, (A)ddress or (D)ata 5-bit content					
Sync	1000_0001	0111_1110						
PLLlock (noop)	1010_1010	1010_1010						
Trigger	tttt_tttt	Tag[0..53]						
Read_trigger	0110_1001	ID<4:0>	00,T<7:5>	T<4:0>				
Clear	0101_1010	ID<4:0>						
Global Pulse	0101_1100	ID<4:0>						
Cal	0110_0011	ID<4:0>	D<19:15>	D<14:10>	D<9:5>	D<4:0>		
WrReg(0)	0110_0110	ID<4:0>	0,A<8:5>	A<4:0>	D<15:11>	D<10:6>	D<5:1>	D<0>,0000
WrReg(1)	0110_0110	ID<4:0>	1,A<8:5>	A<4:0>	N×(D<9:5>	D<4:0>)		
RdReg	0110_0101	ID<4:0>	0,A<8:5>	A<4:0>				

Table 2: List of protocol commands/frames and address or data fields associated with each. Unused padding bits are indicated by "0". Double vertical lines denote frame boundaries. tttt_tttt is one of 15 trigger commands (Table 3). The before-encoded bit content of chip ID, Address or Data is shown. These are all encoded as 8-bit data symbols (Table 17). Not backwards compatible with RD53A.

Symbol Name	Encoding	Trigger Pattern	Symbol Name	Encoding	Trigger Pattern
			Trigger_08	0011_1010	T000
Trigger_01	0010_1011	000T	Trigger_09	0011_1100	T00T
Trigger_02	0010_1101	00T0	Trigger_10	0100_1011	T0T0
Trigger_03	0010_1110	00TT	Trigger_11	0100_1101	T0TT
Trigger_04	0011_0011	0T00	Trigger_12	0100_1110	TT00
Trigger_05	0011_0101	0T0T	Trigger_13	0101_0011	TT0T
Trigger_06	0011_0110	0TT0	Trigger_14	0101_0101	TTT0
Trigger_07	0011_1001	0TTT	Trigger_15	0101_0110	TTTT

Table 3: List of trigger command and symbols used to encode the 15 possible trigger patterns spanning four bunch crossings. Note there is no 0000 pattern as that is the absence of an trigger. All trigger symbols are the same as in RD53A.

The executable non-trigger commands consist of a command symbol identifying which of the 7 commands it is, and a data symbol specifying the chip ID that the command is addressed to, since one single command line might drive multiple chips. The PLLlock (no operation) command does not have a chip ID and instead repeats the same symbol twice to produce a clock pattern. There are 16 possible chip ID values (set by 4 wire bonds to ground overriding internal pull-up resistors). Any chip ID value greater than 15 means broadcast mode, and all chips that receive the command will execute it. A chip that receives a command not broadcast or addressed to it will still process it (so as not to produce "unexpected data frame" errors), but will not execute it. The trigger command is always broadcast. The text descriptions below provide further information to supplement the tables. The trigger and read trigger commands are explained in Sec. **??**.

PLLlock (PLL lock or no-operation):
This commands allows a clock pattern to be sent to the chip without any action being ex-

ecuted by the command decoder. The clock pattern is needed to efficiently lock the Phase Locked Loop (PLL) to the correct frequency at start of operation. Once locked, the PLL no longer needs a perfect clock pattern and regular commands and sync frames can be sent. This command can also be used as an idle (when there is nothing to be sent) during normal operation. Hence this is also a no-operation (noop) command.

Clear:
Clears the entire data path. All pending triggers and stored hits will be erased.

Global Pulse:
The global pulse command sends a single pulse with a duration of 1, 2, 4,... 512 bunch crossings depending on the value of the duration register. The global pulse can be routed to different places of the chip and has many uses. It can provide reset signals, control the ring oscillators, the ADC, etc. The global pulse routing table is **??**.

Cal (Calibration Injection):
The same command is used for both analog and digital injection. Whether injection will be analog or digital is decided by a global configuration setting, but the Cal command produces the same output regardless. To understand the Cal command it is necessary to understand how the calibration injection circuit works. Therefore, the description of the command was given in Sec. 6.

WrReg(0) (Write Register, single):
The WrReg command has two modes: single write and multiple writes to register 0. The command frame is the same and the distinction between single and multiple is made by the first bit of the payload (0=single, 1=multiple). The WrReg(0) or single has 9 bits of Address and 16 bits of Data. Up to 512 16-bit wide registers can be addressed, but not all 512 possible register addresses are used. If an attempt is made to write to an unused address, no register will be written and a warning code will be generated. The register memory map is given in Tables **??**, **??**. This command does not produce any output from the chip.

WrReg(1) (Write Register, multiple):
The WrReg command has two modes: single write and multiple writes to register 0. The command frame is the same and the distinction between single and multiple is made by the first bit of the payload (0=single, 1=multiple). The WrReg(1) or multiple has no address and no data. It can only be used to initiate multiple writing to register 0. The payload consists of the mode bit (1) and 9 mask or mode bits reserved for future use. Register 0 is a virtual register called pixel portal, used to write and read pixel configuration (Sec. 8.8).

Following a WrReg(1) command, one may send data frames to the chip (as many as desired) without any preceding command. All these data frames will be written to the pixel portal (register 0). This must be done in conjunction with auto-increment (see Sec. 8.8). This permits very efficient transfer of data to the pixel portal. The write multiple mode remains in effect until a new WrReg or RdReg command is received. Note that the chips not addressed by the WrReg(1) command will still recognize the multiple write mode and will therefore not issue “unexpected data frame” warnings, but they will not write the data to their register 0.

The placement of the 10 bits from each data frame into the 16 bits of register 0 is described in Sec. 8.8

RdReg (Read Register):
This command has 9 bits of address and no data. It initiates the readout of the addressed register. Address 0 is special: it is the the pixel register as described in Sec. 8.8. The 16-bit register value is returned in the data stream as described in Sec. 10. Not all 512 possible register addresses are used. If readback of an unused address is requested, the value returned will be 0 and warning code will be generated. The register assignment list is given in Tables **??**, **??**.

8.3 Command Protocol Initialization

Until the PLL is locked and produces a stable chip clock, the command decoder will be inactive. During this period, PLLlock frames should be sent to the chip. The transitions in the string of PLLlock frames will allow the clock recovery circuit to lock to the correct 160 MHz frequency. The user does not know when the PLL has locked, but simply sends PLLlock frames for a long enough time that the lock cycle is surely completed. For debugging, the PLL lock condition can be observed in the recovered CMD output of the general purpose LVDS, which is a default output (see Sec. 13.1). At this point the protocol initialization begins. Before any command decoding, the input bitstream is processed by the Channel Synchronizer circuit (Fig.37), and the initialization correctly sets up this circuit.

The sync pattern (Table 2) can not be produced through any combination of TTC frames and therefore can be searched for to lock the correct frame boundaries. Sync frames must be sent at the start of operation so that the framing can be locked (this different from PLL lock!). It is then recommended to send one sync frame in every 32 frames or so in order to maintain lock or allow the command decoder to re-lock if lock was lost. If no sync frames are received in a long time frame lock will be declared lost and the command decoder will stop interpreting commands until a new lock is acquired. Typically at the start of operation (power up) there are no commands or triggers to immediately send, and so sending a large number of sync frames to ensure initial lock is not a problem. The the channel synchronizer lock is available in the chip status CMOS output (Sec. **??**).

Using the 160 MHz recovered clock, the channel synchronizer will search for sync symbols and count each valid appearance of this pattern in 16 separate channels (one channel for each possible frame alignment). When the count for one of the channels, i, reaches a threshold N_{lock}, *sync lock* is declared as acquired, channel i is adopted as the correct channel, and the count of the remaining 15 channels is reset. The number of sync symbols needed at the start of transmission to guarantee a lock is $2N_{lock}$. The 40 MHz bunch crossing clock is generated as the bit pattern 1100110011001100 aligned to channel i. Thus there are 4 bunch crossings with a fixed phase relationship to the sync frame, which can be labeled $BXa..BXd$. The counting of sync sequences continues in all the channels, but every new sync sequence detected on the lock channel i resets the count for all the other channels. If the count for a channel that is not the lock channel ever reaches a second threshold N_{unlock}, lock is declared lost, and a new sync lock is acquired on the first channel that reaches the locking threshold N_{lock}. This allows for continuous channel monitoring

and automatic sync lock as long as enough sync symbols are transmitted. Additionally, if zero sync frames are received in the lock channel within N_{lost} frames (regardless of other channels), lock will be declared lost and no further commands will be decoded until a new lock is acquired. This is useful to prevent prolonged, random input due to an upstream exception from corrupting the chip operation. (N_{lost} is set in multiples of 256.) The values of N_{lock}, N_{unlock}, and N_{lost} are contained in configuration register CH_SYNC_CONF. They are user configurable with non-zero defaults. Unless there is a problem the default values will likely be used.

8.4 Command Protocol Transmission

During transmission the necessary sequence of frames is sent to control the chip. Trigger frames are sent at specific times, and the “space between trigger frames" is filled with commands, data and sync frames. The data frames following a command frame are interpreted as belonging to that command regardless of intervening trigger, idle, or sync frames. The PLLlock command (noop) should be used as the idle frame, as it has the most transitions and will therefore best maintain PLL operation.

8.5 Command Protocol Decoding

The data bits recovered from the locked channel are fed to the Command Decoder as shown in Fig. 37. In the absence of a sync lock, nothing is fed to the command decoder, so until a lock happens no commands will be interpreted. The locked condition guarantees that the bits fed to the command decoder are correctly aligned with the clock. Protocol consistency is checked by requiring that there is always a valid symbol pair or a sync in each 16-bit frame. The 16 bits are fed to the command decoder with a parallel bus. In case of correct detection, the indicated action is performed according to the command type or Chip ID. All symbols are always checked and decoded, even if they follow a Chip ID that does not match the wire bonded ID. However, the Command Decoder will act on the rest of the chip only if the command is a trigger, if decoded Chip ID matches the wire bonded ID, or if the decoded broadcast bit is 1 (the PLLlock command is not addressed, but has no internal action- no operation). The detection of an invalid symbol is handled differently depending on the frame and expectation (current state). The handling of exceptions is shown in Table 4.

8.6 Command Protocol Timing

The decoded commands are executed 25 ns after the end of the last frame of the command data. “Executed” means that the outputs of the Command Decoder block in Fig. 37 change state, which happens of a rising edge of the beam clock. In many cases the execution is instantaneous (outputs change state and that’s it), but the Trigger, Cal and Global Pulse commands have a delay and duration. The trigger command sends 1 to 4 pulses in 4 consecutive beam clock cycles, and thus is completely finished before a new command can be completely received (since 1 frame is 4 beam clock cycles). The Cal and Global pulse commands can occupy their respective output lines (CAL_edge, CAL_aux, and Global_pulse) for many clock cycles. A new trigger or global pulse command should not be sent before the prior such command is complete (up to the DAQ to ensure this), but any other command can be sent and will be executed normally.

Frame received	Frame Expected	Error/Action
invalid, data	data	Aborted command
data, invalid	data	Aborted command
invalid, invalid	data	Aborted command
invalid, data	not data	Lost trigger
invalid, invalid	not data	Corrupted frame
invalid, sync	any	Corrupted sync
sync, invalid	any	Corrupted sync
invalid, command	any	Execute with warning
command, invalid	any	Execute with warning
trigger, invalid	any	Lost Tag (*)
command, command	data	Ignored command

Table 4: Command Decoder response to invalid or unexpected symbols. (*) Tag value 0 is substituted. User is free to never send tag value 0, in which case there will be no confusion.

8.7 Global Configuration

8.8 Pixel Configuration

Each pixel has 8 bits of local configuration as detailed in Table **??**. These are divided into 5 TDAC bits (threshold tuning bits) and 3 enable bits (also known as mask bits). These two types of bits can be written independently. Thus one can choose to write only the 5 TDAC bits, only the 3 enable bits, or all 8 bits at once. The choice is controlled by two global configuration bits.

From the point of view of the WrReg(0) and RdReg commands, each pixel is seen as one half of a 16-bit register. All pixels are paired as shown in Sec. 7.5. For the WrReg(1) command, each pixel is still half of a 16-bit register, but only 10 or 6 bits are written by each write cycle: the TDAC bits (5+5) or the mask bits (3+3), never both at once. Which one is written (TDAC or mask) is determined by a global configuration bit. The mapping from 10 bit data frame to two pixel TDAC or mask is also shown in Table **??**.

Internally, the writing and reading of configuration values from the pixels uses an addressed bus to every 4x1 pixel region. All reading and writing is done two pixels at a time in a given column of 4-pixel regions. (See Sec. 7.5 for address encoding). However, multiple core columns can write in parallel, while readback can only take place from one quad column at a time. There are thus two write modes, single region and broadcast, while read is always single region.

The write and read operations are controlled by three global registers, the core column, the core row, and the pixel mode configuration registers. The core column and core row registers have a special feature called auto increment, which reduces the number of commands needed to fully configure the chip. The pixel data is written into or retrieved form global register 0 with write and read register commands (see Sec. 8.2). This is a virtual register acting as a portal to whatever region(s) is(are) pointed to be column and row config registers.

The typical pixel matrix configuration write sequence, using the write single register command, is given in Table 5. Note that this takes 78400 commands to accomplish; at 40 bits per Write Register command (before encoding) and effective non-trigger command bandwidth of 64 Mbps

during 4 MHz L0 + 1 MHz L1 trigger operation, 78400 write single register commands will take 80 ms. With some overheads, should assume 100 ms. So for a 4-chip module configuration will take 400 ms and for four 4-chip modules 1.6 s. Writing a uniform (all pixels the same) configuration is 200 times faster because each Write Pixel command can write a whole row Table 6. The readback of the pixel configuration for the whole matrix can proceed exactly as shown in Table 5, substituting the Read Register command instead of Write Register. This can be carried out in broadcast mode to any number of chips in parallel, so will always take 50 ms (half as much as writing a single chip because the read register command is shorter). Writing or reading an individual, arbitrary pixel pair follows steps 1-3 of Table 5.

Using the multiple write version of the write register command significantly speeds up writing configuration. In this case, writing TDACs (which are the bits that are different for every pixel), will be 5 times faster, because each pixel pair is written with one data frame instead of the 5-frame write single register command. So four 4-chip modules on a single command line will take 320 ms to configure. Furthermore, writing mask bits can be even faster because this can be done in broadcast mode (same mask on all chips even though the TDACs are different), and this will take 20 ms regardless of the number for chips sharing the command line.

Step	Command	Address	Explanation
1	Write_Register	column and mode config	set columns 0-1 and auto row mode
2	Write_Register	row config	set row 0
3	Write_Register	0	config first 2 pixels
4	Write_Register	0	config for next row 2 pixels
392	Write_Register	0	config for last row 2 pixels
393	Write_Register	column and mode config	set columns 2-3 and auto row mode
394	Write_Register	row config	set row 0
395	Write_Register	0	config for next row 2 pixels
78400	Write_Register	0	config last 2 pixels in chip

Table 5: Sequence to write an arbitrary pixel configuration using write register single commands. For readback replace Write_Register 0 with Read_Register 0 commands.

Step	Command	Address	Explanation
1	Write_Register	column and mode config	set broadcast and auto row mode
2	Write_Register	row config	set row 0
3	Write_Register	0	config all pixels, first row
4	Write_Register	0	config all pixels, second row
196	Write_Register	0	config all pixels, last row

Table 6: Sequence to write a default (all pixels the same) configuration

9. Trigger Processing and Tags

A very simplified description of the chip triggered readout is as follows.

1. A trigger command is received, including an identifier called tag,
2. The tag is stored along with a "read or clear" bit, and the hits from the appropriate bunch crossing are assigned to this tag (but left in the pixel matrix),
3. An "egg timer" is started dedicated to this tag. The (programmable) start value is the same for all triggers (this value is T_R in Fig. 38),
4. When the egg timer expires, the hits from this tag are either cleared or read out, depending on the value of the read or clear bit.
5. The hits are read or cleared from all core columns in parallel, and then assembled into whole events at the chip bottom, along with the tag. Whole events (tagged) are placed in streams and sent to the AURORA encoder for output.
6. The processed tag is erased from storage and is now available to be used again from another trigger.

There are two trigger modes, known as single level and two-level, selectable by configuration. They are more correctly called auto-read and manual read modes, respectively. They differ only in how the read or clear bit is set and the egg timer value, but otherwise follow the above processing identically. In auto-read mode, the value of the read or clear bit is set to read for every incoming trigger. There is no way it can be set to clear. Every trigger will be read out. The egg timer delay is small (one bunch crossing)

In manual read mode (two level trigger), the read or clear bit is initialized to clear for every incoming trigger and the egg timer delay is long (level 2 latency). At any time before the egg timer expires, a read_trigger command can be received for the tag in question, in which case the read or clear bit will be set to read. The read_trigger has no action other than setting the read or clear bit. Readout will happen when the egg timer expires, as usual. Any events for which the bit has not been set to read with a read_trigger will be cleared.

The following subsections give a more detailed description.

9.1 Bunch crossing ID selection and latency

The pixel matrix operates in steps of the 40 MHz beam crossing clock (BX). Within the matrix each BX is triggered or not based on the state of a trigger signal (high is triggered, low is not). Thus a trigger is a one BX long pulse on this trigger signal. Trigger pulses are normally issued by the command decoder in response to commands received- they are not provided externally. The trigger command path, from chip input to internal trigger pulses, is shown Fig. 37. Note that the internal BX clock is generated by the channel synchronizer based on the frame alignment of the input control stream (see Sec. 8.3). An individual chip phase adjustment, in 1/(1.28 MHz) steps, is introduced by the clock and data recovery circuit (CDR). Thus, each chip can be individually "timed in" to the bunch crossings.

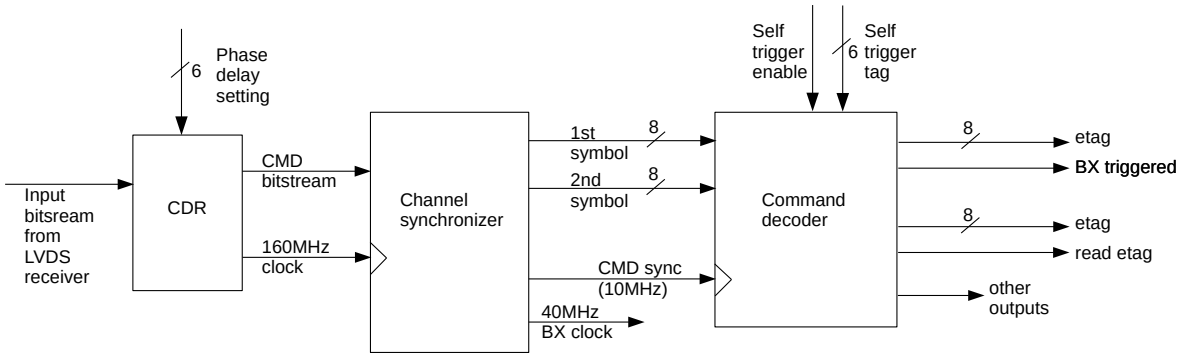

Figure 37: Clock can command recovery and decoding path from chip input to internal signals, showing trigger pluses and tags in particular. Other outputs of the command decoder, such as global register address and write signal, not shown.

Each of the 15 trigger commands of Table 3 generates a different pattern of pulses spanning four BX's. Trigger pulses can also be generated by the internal self trigger source (Sec. 9.4). The command decoder arbitrates the trigger sources, with trigger commands always having priority.

In RD53B each trigger pulse marks data in the matrix as triggered and associates it with a trigger identifier (ID), but *does not* initiate readout. The readout of data marked by a trigger ID is initiated later, as described in Sec. 9.3. Readout can be automatic (every trigger received is read out), which is the single level trigger operation mode, or deliberate with an explicit read trigger command for every previously supplied trigger, which is the two-level trigger mode.

Fig. 38 shows the timing form a trigger to the completion of data readout. The time to start of read (T_R) is a fixed value (programmable) between 1 and 1024 bunch crossings. For single trigger level mode it will be 1 BX, while for two level mode it will be the level-1 latency (after level-0). In single level mode the start of read happens automatically, while in two level mode a read_trigger command is needed for each trigger to be read out. This command can be received at any point within T_R.

The data bits for a given trigger do not instantly come out of the chip upon start of read, but are subject to queuing wait time and processing delay. The total wait time between start of read and bits at the chip output is T_W and varies depending on activity. It contains fixed delays including the 3 BX token transit to retrieve data from be pixel matrix, another 3 BX for column hit data encoding, the AURORA encoding, etc. The sum of all these fixed delays and, therefore, the minimum T_W is 31 BX, while the distribution is given by the added queuing time, which can be truncated with a programmable time-out. Finally, the readout time ($T_{R/O}$) is given by the number of bits being sent out times the output multi-lane bit rate, which can be up to 5.12 Gbps (4 lanes at 1.28 Gbps each). At a given trigger rate, the average $T_{R/O}$ must be less than the mean trigger period (λ), and significantly less to avoid long queuing wait time. Since the whole chip can be analyzed as a single server queue, the wait time plus readout time $T_W + T_{R/O}$ will have a distribution like Eq. 9.1,

$$P(W > t) = \frac{T_{R/O}}{\lambda} e^{-(\lambda - T_{R/O})t} \tag{9.1}$$

It is clear from Eq. 9.1 that as $T_{R/O}$ approaches λ the total wait time diverges. The condition $T_{R/O} = \lambda$ roughly corresponds to 100% data link occupancy.

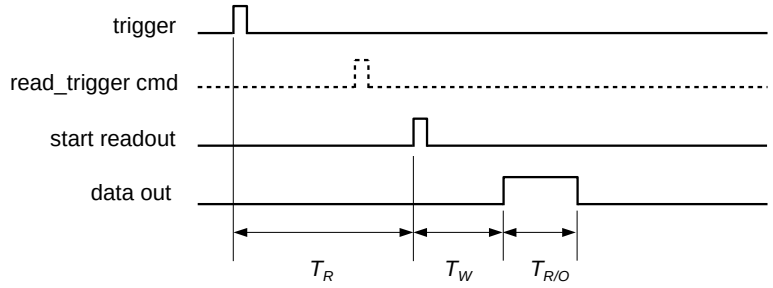

Figure 38: Timing of trigger to data readout.

9.2 Tags

The term tag is overloaded with two meanings. In the command protocol *tag* refers to the 6-bit code received with each trigger command (more correctly called tag base). The tag base can only take on 54 values, which is the total number of DC-balanced symbols. Inside the chip and in the output data *tag* refers to an 8-bit extended tag. The two additional bits indicate which of the four BX's spanned by a trigger command the data correspond to. For example, command Trigger_04 in Table 3 with tag base value abcdef will result in one extended tag: abcdef01, while Trigger_05 will result in two extended tags: abcdef01 and abcdef11 (along with two trigger pulses).

While there are only 54 tag bases, which can lead to at most 4×54 = 216 extended tags, the extended tag space in the chip spans all 256 8-bit codes. The extra codes that cannot be generated from one of the 54 tag bases are used for a number of conditions as indicated in Table 7. In RD53B the main use of special tags is to label self-triggered events. Since self-triggers are generated internally in the chip, they are not constrained to the 54 tag bases, and so they are labeled with extended tags that could never result from a trigger command.

Tag values (decimal)	**Meaning**
0-215	extended tags from trigger command
216-219	Self triggers with tag base 55
220-223	Self triggers with tag base 56
224-227	Self triggers with tag base 57
228-231	Self triggers with tag base 58
232-235	Corrupted tag base in trigger command
236	Special tag assigned when received tag is still in use
237-243	additional still-in-use tags
244-255	spares

Table 7: Possible extended tag values and their meaning.

The number of tag bases has been sized for 4 MHz trigger rate with all triggers being held for a read timeout of 20 μs after the trigger. The average number of triggers in a 20 μs interval is 80, but the average number of trigger commands is 69 (each command spans 4 BCIDs). Furthermore, 50 of these trigger commands (on average), contain only one triggered BCID's. Assuming a random distribution of the triggered BCID within the four spanned by the command, these 50 trigger commands require just 12.5 tag bases (each tag base can be used 4 times). A similar analysis shows the remaining 19 trigger commands require 10 tag bases. The average number of tag bases in 20 μs is thus 22.5. The Poisson probability for this fluctuating to more than 54 is 5×10^{-9}. Even if all multiple trigger commands are assigned a unique tag base, to simplify tag base book-keeping by the DAQ, this results in an average of 31 tag bases each 20 μs, which has a probability of fluctuating above 54 of 6×10^{-5}.

9.3 Trigger Book Keeping

RD53B keeps track of pending and in progress triggers using a master table holding all triggers plus one replica table in each core column with those triggers staged for readout or clear. This is shown conceptually in Fig. 39. Within the chip the identifier of each trigger is the table row number, which is Gray coded. If a trigger is placed in row 5 of the master table, when propagated to the column tables it will be written in to row 5 of every table, and so on. While the trigger tag could have been used to identify triggers inside the chip as well as outside, use of the Gray coded row number minimizes the number of signal transitions propagated to the pixel matrix every time a trigger is received, cleared, or read out. The master table serves as the translator between tags (external trigger ID) and row number (internal trigger ID).

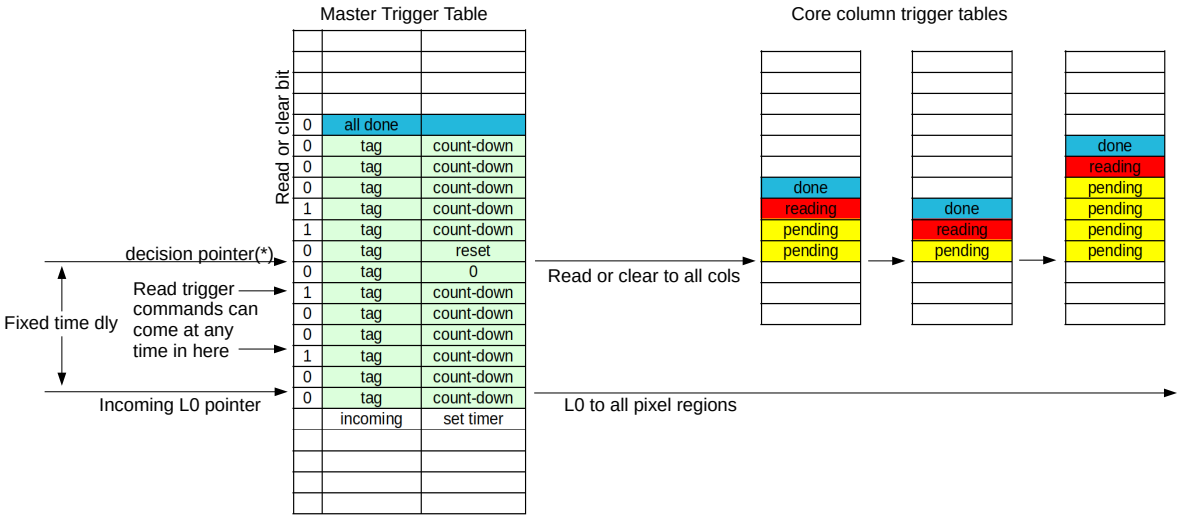

Figure 39: Conceptual diagram of trigger tables in RD53B. All tables have the same row number, and the row number is used as the trigger ID in the pixel matrix. The tables are circular buffers filled from the bottom and emptied from the top.

When a L0 trigger is received, it is placed in the master table and the row number is forwarded to the full matrix, to flag any data for the triggered BCID in the pixel regions. The incoming trigger pointer is advanced at this time. The hit data are now protected and will not be erased until read out or explicitly cleared. A fixed time later, which could be 1 BX or programmable up to 1024 BX's (T_R in Fig. 38), a read or clear signal is forwarded from the master table to all the core column tables (not yet to the pixel regions). Recall that the choice between the read or clear actions was made before T_R, either automatically (single trigger mode) or by a read trigger command (two-level trigger mode). Which L0 is being read or cleared at the present time is marked by a decision pointer as showing in Fig. 39. This pointer is advanced whenever a stored L0 reaches the programmed T_R. Each row of the master trigger table has its own count-down timer. This ensures two things. (1) during any given BCID at most one read or clear signal needs to be forwarded and (2) the column trigger tables are in sync with the master trigger table (the row number is the trigger ID everywhere). Both the master and column tables are circular buffers with one trigger per row.

All triggers stay in the master table until fully read out or cleared from the array, or until a time-out is reached, and are then erased in the order received. Normally, a trigger is not erased from the master table until all columns have finished processing it. But, since and all triggers received after it must wait for it, there is also a time-out to clear triggers that are taking too long. This prevents a single column that gets stuck from blocking the entire readout. The row timer in master table is recycled to count this readout time-out after decision pointer is reached. A column could get stuck, for example, if it has many noisy pixels increasing its occupancy, which will result in a long queuing wait time. While a column truncation limit can be programmed to abort the readout of a column with too many hits, this does not protect against all columns having above average number of hits, as would be the case from random noise.

Each column operates as a single server queue, with a 40 MHz output rate. Each trigger requires 3 bunch crossings to process (regardless of hit content) and each hit quarter core another 3 bunch crossings. Thus, for a 1 MHz (4 MHz) trigger rate, the input rate to the queue is 3 MHz (12 MHz) even without any hits, leaving room for 5.7 (2.7) hit quarter cores per trigger on average before the column readout bandwidth reaches 50% saturation, beyond which point the wait times grow rapidly (Eq. 9.1). This corresponds to quarter core occupancies of 3% (1.5%), or, assuming an average of 3 hits per hit quarter core, 0.6% (0.3%) pixel occupancy.

9.4 Self Trigger Source

The self trigger functionality is implemented in two parts. The command decoder includes a self trigger action as discussed in Sec. **??**. This relies on a self trigger source, described here.

The self trigger source takes the HitOr signals at the bottom of chip and generates synchronous pulses and tags to feed to the command decoder. It incorporates combinatorial logic among the four HitOr buses, programmable latency from HitOr edge to trigger pulse, the possibility to make a ToT cut so that only HitOr pulses greater than a programmable duration cause a trigger, and the programming of a number of consecutive triggers (from 1 to 4) for each accepted HitOr event.

The combinatorial logic consists of a global OR operation on 16 possible inputs derided from the four HitOr signals as summarized in Table 8. Each bit of a 16-bit configuration register enables one of the input ports to the global OR. For example, configuration setting 0000 0000 0000 1111 will result in the simple OR of the four HitOr buses (which is the straight OR of all selected pixels

in the chip). The global OR output is synchronized with the 40 MHz BX clock and fed into a synchronous delay line. The length of the delay line is programmable (in BX clock units) with a minimum value of 16 and a maximum equal to the maximum trigger latency. A minimum of 16 is used, rather than zero, to allow for the maximum ToT counting to elapse in the pixels. This conveniently permits to also perform "for free" a ToT cut for the trigger generation, as shown in Fig. 40. The output of the ToT cut delay line is fed to a one-shot circuit to produce a pulse from one to four BX clocks long, to feed to the command decoder. Each leading edge of a HitOr pulse satisfying the ToT cut will produce a fixed duration pulse (as programmed in the one-shot). If two consecutive edges are closer together than the programmed one-shot width, the second edge will be ignored. The one-shot output leading edge is also used to advance a wrap-around counter generating the tag base to be used but the command decoder. The low and high values of the tag base counter are programmable within the range of Table 7.

Config. bit	Inputs	Comment
0	none	A 0 setting disables trigger
1-4	HitOr_0 to HitOr_3	direct connection of each HitOr
5-10	HitOr_i AND HitOr_j	ij = 01, 02, 03, 12, 13, 23
11-14	HitOr_i AND HitOr_j AND HitOr_k	ijk = 012, 013, 023, 123
15	AND of all four	

Table 8: Selection of inputs to global OR operation feeding the self trigger generation.

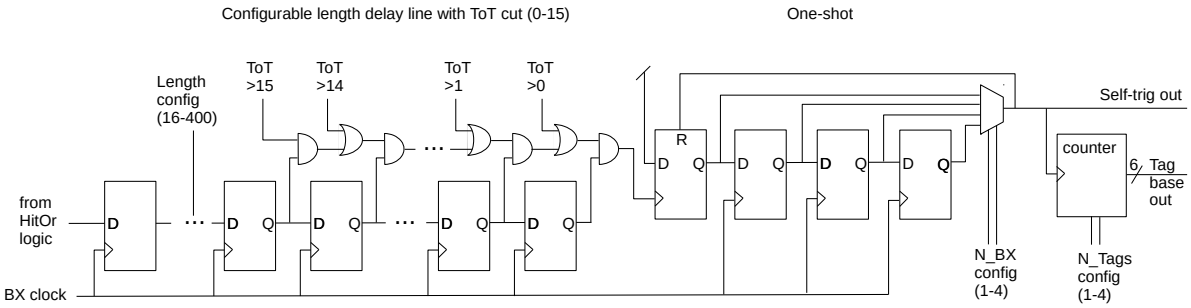

Figure 40: Synchronous trigger pulse generation circuit. It consists of a delay line with built in ToT cut, a one-shot circuit with programmable pulse width, and a counter to step through the selected tag base values.

10. Data Output

The RD53B data output improves upon RD53A on several fronts. It consists of tagged events, such that the readout will automatically recover from transmission errors without any action from the DAQ. While tagged data would permit event building to be performed off chip if desired, RD53B builds events on-chip, such that a full event is output before sending any data for the next event. The RD53B output is more efficient than RD53A (25% fewer bits per hit or better) for the full range of event sizes across the detector layers. The transmission protocol used is multilane AURORA 64b/66b as in the RD53A chip, but the bit packing (encoding) prior to the AURORA processing is different. The electrical characteristics of the outputs are described in Sec. 10.1 and the AURORA formatting is described in Sec. 15.2.

Within the AURORA blocks, the hit data are packaged in *streams*, not fixed frames. AURORA framing is not used (in other words, the output is one infinite length AURORA frame). A stream is a self-contained, variable length data container beginning with a tag (8 bits) and followed by a mix of hit data and possibly other tags (called internal tags, which are 11 bits). Streams and their contents are described in Sec. 10.3 to 10.9.

There are two encoding modes: single chip and multi-chip. Multi-chip encoding must be used when performing data aggregation. The encoding description in Sec. 10.4 is given for single chip mode, and the effect of multi-chip mode is described in Sec. 10.7. The use of multi-chip mode for data aggregation is described in Sec. 11.

The output is highly configurable and must be correctly set up to perform as required. The basic configuration for single chip operation was described in Sec. 3. Control of event size and data filtering options are covered in Sec. 10.8. Use of pre-emphasis for operation with lossy cables is included in Sec. 10.1. Use of test modes, for example for bit error rate studies, is covered in Sec.13. Technical details of clock and data recovery and serialization are given in Sec.14.

10.1 Output drivers

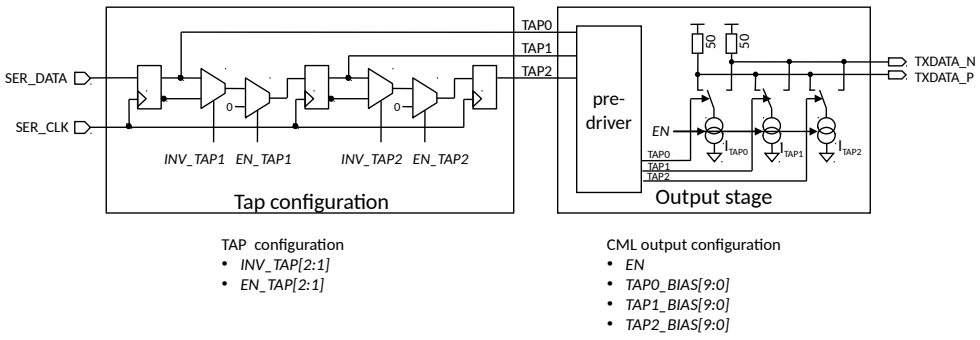

Figure 41: Detailed CML driver functional block diagram including TAP circuit

10.2 AURORA and register blocks

At the highest level, the RD53B output is encoded with AURORA [**?**] 64b/66b on 1 to 4 parallel lanes (programmable). Each lane runs nominally 1.28 Gbps, but can be divided down by 2, 4, or 8.

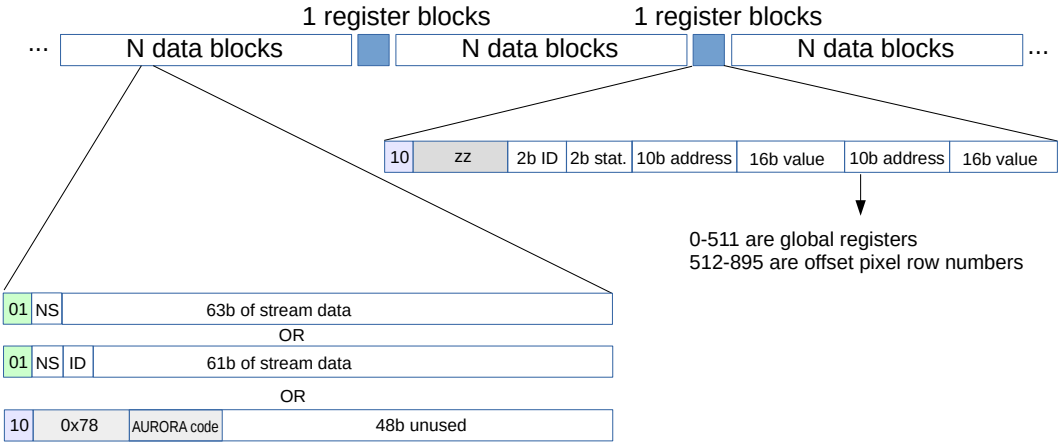

Figure 42: Schematic diagram of output data highest level format, consisting of N data or idle blocks followed by one register block. Each block consists of an AURORA 2-bit header that can be 01 or 10, plus 64 scrambled bits. The diagram shows the content of the 64 bits before scrambling. The gray shaded 8-bit fields with values given in hex have a meaning defined in the AURORA protocol. The possible zz values are given in the text. NS stands for New Stream bit and ID for the two least significant bits of the chip ID.

Register block Aurora code (hex)	**Meaning**
0xB4	both register fields are of type AutoRead
0x55	first frame is AutoRead, second is from a read register command
0x99	first is from a read register command, second frame is AutoRead
0xD2	both register fields are from read register commands
0xCC	Indicates an error. Fields are meaningless

Table 9: Meaning of Aurora code (zz) in the periodic register block.

Each enabled lane outputs an endless succession of 66 bit AURORA blocks. The multi-lane AURORA protocol uses *strict alignment*, which means that all lanes send the same type of AURORA block at any given time. AURORA start and end of frame markers are not used and only some of the non-data blocks are used. AURORA blocks consist of 66-bits (Fig. 42). Each block has a 2-bit sync header that can be 01 or 10, followed by 64 scrambled bits. Because the header is not scrambled, it permits frame alignment of the received data. Frame alignment identifies where each 66 bit block starts. Header 01 always indicates an AURORA data type, while the 10 header blocks can be of user data type or AURORA command type. RD53B implements two independent output "channels", data and register (or service), which are time-multiplexed onto a single bitstream. The output stream basic unit consists of N_D RD53B data blocks plus one register/service block. This N_D:1 ratio is programmable so the user can decide what fraction of bandwidth to reserve for data. When there is no data to send out, idles are sent. Register blocks will not be sent except in

their allocated turn every N data or idle blocks. The interval N is used on every lane regardless of how many lanes are active. For example, with N_D=48, 2% of the output bandwidth is permanently unavailable for hit data (in addition to the 3% consumed by the 2-bit 64b/66b header). At 4×1.28 Gbps output bandwidth this 2% is sufficient for the maximum possible register readback of 64 Mbps, since 2% of 5 Gbps = 100 Mbps (See Sec. 8.8). In the register blocks, an 8-bit code follows the sync header, as specified by the 64b/66b protocol, leaving 56 bits available for user information, which are allocated as a 2-bit chip ID plus two 26 bit registers (10-bit extended address plus 16-bit value = 26 bits) plus 2 status bits, specified in Table 10. (This is a small change relative to RD53A, which did not include chip ID bits and had 4 status bits).

ID[2 bits] 2x([e-address (10 bits)] [value (16 bits)]) [status (2 bits)]

Because of the chip ID, the register block is always compatible with multi-chip mode. The 10-bit extended addresses (e-address) in the register block are: MSB=0, followed by the 9-bit global register address, or MSB=1, followed by the 9-bit pixel row address in case of reading global register 0 (the pixel configuration portal register). The separation of the output into two time-multiplexed channels guarantees a certain bandwidth for both data and register information without the need for a complex priority arbitration containing safeguards against all possible pathologies. The format is depicted schematically in Fig. 42.

The periodic register block coming out every N_D data frames is filled automatically, even without there having been a read register command. The possible AURORA values denoted zz in Fig. 42 are given in Table 9. The two 16-bit registers are denoted Ai and Bi, where i is the lane number (0 to 3). The automatic filling of the Ai and Bi registers is controlled by eight configuration registers Auto-Ai and Auto-Bi, which have default values, but which the user is free to change. The auto-fill register addresses are specific to each lane. Thus if only lane 0 is used then only Auto-A0 and Auto-B0 are functional. RdReg commands will queue the registers specified by the command for output on lane 0 only, with priority over auto fill. Lanes 1 to 3 are unaffected by the RdReg command and only output their assigned auto-fill registers. If only one RdReg command has been received, then the A0 register will be auto-filled while the B0 register will contain the requested register. If more than one was received then both registers will be requested registers and auto-fill will wait. If read register commands are sent too fast for the reserved output bandwidth, the FIFO holding pending read regiseters may fill up, and any read regiser commands received while the FIFO is full will be ignored. The readout of registers staged in the FIFO may also happen out of order.

Status Code (decimal)	**Meaning**
0	Ready
1	There has been an error since the last register frame
2	There has been a warning since the last register frame
3	Both 1 and 2

Table 10: Meaning of 2-bit status code

10.3 AURORA and streams

The AURORA protocol transmits data in 64-bit blocks. These blocks are scrambled and each preceded by a 2-bit AURORA header that permits frame alignment. In the RD53B encoding, each 64 bit data block, before scrambling, begins with a New Stream bit (NS). If NS=1, this indicates the start of a new stream, which can be decoded without any information about the data that came before. If NS=0, this indicates that the previous stream simply continues, and therefore the data following NS=0 cannot be interpreted without having the prior part of the stream.

Fig. 43 shows a continuous bit stream as would be seen after AURORA decoding. The position of the NS bits in this bitstream are known (red and blue), thanks to AURORA having taken care of frame alignment. The figure also shows two RD53B streams, which are self-contained, variable length data packets. The start of each stream in the continuous bitstream is flagged by NS=1 (red), while NS=0 (blue) are simply ignored.

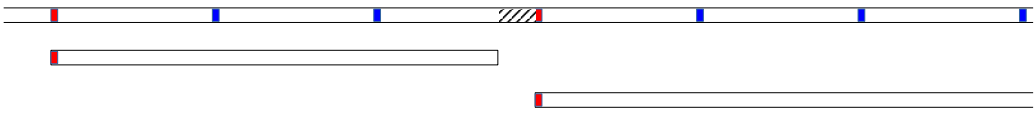

Figure 43: Continuous bitstream after AURORA decoding (top) showing the NS bit positions (which correspond to AURORA block boundaries). Two streams are shown below. NS=1 is shown in red and NS=0 in blue. Orphan bits are shown as hatched.

RD53B streams only contain hit and exception data. Configuration readback and monitoring data are not included streams, but are sent in the periodically inserted AURORA service blocks just as in RD53A.

An RD53B stream contains N_E events, where N_E is programmable form 1 to 64. For short or empty events (as will occur in outer layers), single event streams will be inefficient, because the so-called orphan bits between the end of a stream and the next NS bit (hatched in Fig. 43) are wasted. For long events (as in the inner layer) single event streams only waste a few percent of bandwidth on orphan bits. The default setting is single event streams ($N_E = 1$).

A new stream always begins with a tag (8 bits) and is followed by a mix hit data and if $N_E > 1$, other tags (called internal tags, which are 11 bits). The initial tag is always present. Also a tag is always output for every trigger read out even if the event is empty. (In single level trigger mode every trigger results in an output tag, while in two level trigger mode every read_trigger results in an output tag.) The possible tag values are given in Table 7. The hit data are compressed, and, therefore variable length (number of bits per hit varies).

The end of a stream is normally known by the start of a new stream (NS=1). However, there can be cases where the next NS=1 bit takes a long time because there have been no triggers. Therefore, and end of stream marker is available, consisting of 000000. This is neither a valid ccol address not a valid internal tag (see Sec. 10.4), and is interpreted as an end of stream marker. Its use can be optionally disabled via configuration. Note that in case a stream ends with fewer than

six orphan bits to the end of the 64-bit boundary, an entire 64 bit block of all 0 will be added in order to complete the end of stream marker.

10.4 Hit data encoding

Within a stream, hit data encoding uses a hierarchical address of core-column (ccol), quarter-core row (qrow) within that column, and 2 pixel x 8 pixel quarter-core hit map, compressed as explained later. Following the quarter-core hit map are the ToT values for all hit pixels in the quarter-core (which can suppressed by a configuration option). The qrow address field begins with two flag bits called *islast* and *isneighbor*. The islast bit is set if this is the last qrow address in the ccol and zero otherwise, while the isneighbor bit is set if the previous address was qrow-1 and zero otherwise. When isneighbor is set, the qrow address is omitted, as it is known to be the previous address+1. (This is a form of Huffman coding: since the most frequently occurring qrow address is the previous address+1, due to the clustered nature of hits, a single bit is used to encode this address, wile for all other cases a 0 followed by the full qrow address is used.)

Fig. 44 shows the bit content of various hypothetical short streams, without showing AURORA block boundaries. Each of these streams could span one or more AURORA blocks. The NS bit at the start of each AURORA block is, therefore, only shown for the start of the stream. These examples illustrate the encoding hierarchy, where the different fields appear depending on the data content, and the functioning of the *islast* and *isneighbor* bits. Placing all ToT's in one block after the quarter-core map makes it simpler to drop ToT, should that be needed, but the default encoding contains the 4-bit ToT values. Table 20 provides a stream decoding reference.

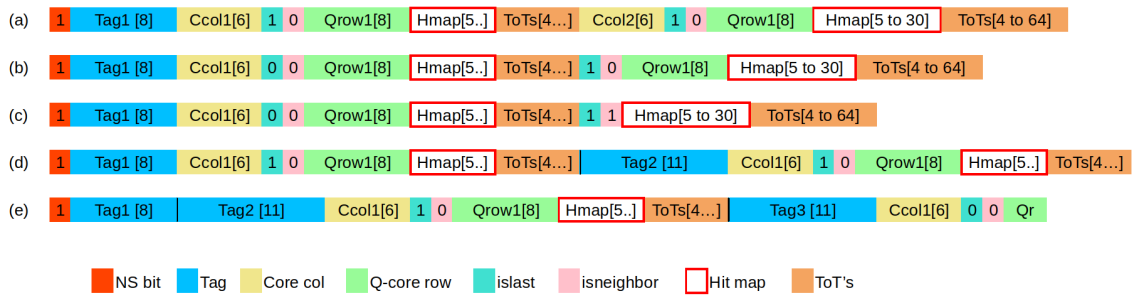

Figure 44: Examples of encoded stream data with no AURORA block boundaries shown: (a) one hit quarter-core each in two ccols (note last hit bit is set for both), (b) two separated quarter-cores hit in same ccol (last hit set only for second), (c) two neighbor quarter-cores hit in same ccol, (d) one hit quarter-core each in two different events, (e) an empty event followed by an event with one hit quarter-core, followed by another event. A color key to the field types is shown at the bottom. The number of bits in each field is shown in square brackets.

The ccol address is not compressed. The allowed range is 1-55. The value 0 is reserved for the end of stream marker mentioned earlier. Since all valid ccol values are < 56 (binary 111000), an address 111xxx is interpreted as the first bits of an internal tag instead of a ccol. The full internal tag is thus 111xxx xxxxx (see Fig.44d,e). The qrow address begins with the two flag bits islast and isneighbor as explained before. There is only compression in the case of isneighbor=1, which is significant, as this condition is common for clustered hits.

10.5 Stream construction and efficiency

The encoder must decide when to end a stream and start a new one. The encoder does not know in advance when a stream will end; the data will determine that. A stream will end when (1) N_E events have been added, or (2) there is no more data to be sent.

For both of the conditions that end a stream, there will typically be a remainder of *orphan* bits between the end of the stream and end of the last 64 bit frame. These bits could in principle be used for something, but in RD53B they are padded with zeros. The DAQ should ignore orphan bits. For easy identification of orphan fragments the core column addresses start at 1 instead of 0. Thus, 000000 effectively marks the end of a stream, whether the end of stream marker is enabled or not. If the end of stream marker is disabled the number of orphan bits can be fewer than 6, even none, while if end of stream is enabled there will always be 6 or more orphan bits. For example if a stream would have 4 orphan bits (0000) with end of stream marker disabled, a new block would be added, increasing the number of orphan + end of stream marker bits to 67. Fig. 45 shows the bit content of a hypothetical stream extending across two AURORA blocks.

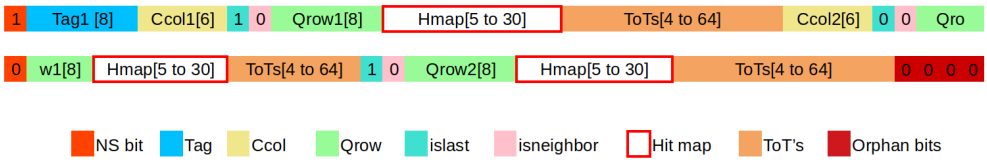

Figure 45: Encoded output for one hit quarter-core in one core column, and two adjacent hit quarter-cores in another core column, spanning two AURORA blocks. The new stream bit (red) is set for the first AURORA block (top) and zero for the second, indicating that the stream continues in the second AURORA block. Orphan bits set to zero (dark red) end the stream in the second AURORA block.

The fraction of bandwidth wasted on orphan bits (inefficiency) can be easily estimated from the stream length. Taking the stream length as an approximately random variable, the average number of orphan (+ end of stream) bits per stream is 31 (37) if end of stream marker is off (on). This, in order to achieve a small fraction of wasted bandwidth, for example <2%, the average stream length must be >1550 (>1850) bits. So one should program $N_E = 1550/\overline{W_E}$, where W_E is the number of bits per event.

10.6 Hit map compression

The quarter-core hit map (16 bits) is (A) encoded using a binary tree and (B) the code is then compressed with a bit code substitution. This section explains the encoding in an algorithmic way that is easy to understand, but does not reflect how it implemented in the chip.

(A) Binary tree construction This can be done recursively in 3 steps (for the 16 pixel quarter-core) as follows

1. divide quarter-core in two halves and label each half with 1 if it contains any hits and 0 if it does not

2. repeat step 1 for each half labeled 1.
3. repeat step 1 for each quarter labeled 1.

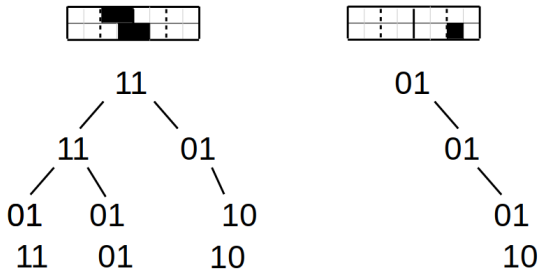

Figure 46: Depiction of binary trees for two example quarter-core maps. The bottom tier of the trees consists of 2-pixel hit maps.

One now has identified all pixel pairs with at least one hit (each quarter has two pixel pairs). The 2-bit map for each hit pixel pair is saved. The results of the encoding are: 2 bits for step 1, from 2 to 4 bits for step 2, from 2 to 8 bits for step 3, plus the 2-bit maps of all the hit pairs. A quarter-core hit map with a single hit will have an 8-bit binary tree representation. A quarter-core with 2 hits will have a binary tree with between 8 and 18 bits, etc. (these numbers will reduced by action B).

A required ingredient for constructing a tree is a definition of the core subdivisions. For example, in (1), we need to decide which pixels are in the first half and which in the second half of the core. The implemented subdivisions are left-right for step (1), top-bottom for step (2), and left-right for step (3). The trees for two example hit maps are depicted in Fig. 46.

(B) Bit code replacement. It should be clear from Fig. 46that the bit code 00 never appears, since only maps with at least one hit are being encoded. As there are only three used 2-bit codes, one of them can be replaced with a 1-bit code. The substitution $01 \rightarrow 0$ is made everywhere. This is a minimal case of Huffman coding. The encoded maps for Fig. 46 thus become:

- $11.1101.010110.110110 \rightarrow 11.110.0010.11010$ (14 bits instead of 18)
- $01.01.01.10 \rightarrow 0.0.0.10$ (5 bits instead of 8)

Note that the choice $01 \rightarrow 0$ instead of $10 \rightarrow 0$ is arbitrary and makes no difference for the given choice of subdivisions, as they are symmetric.

10.7 Multi-chip encoding

Data merging combines data from multiple chips onto a single AURORA output. In this mode each chip still produces streams, but the merged data contains AURORA data blocks from multiple chips. To reconstruct the streams from a given chip, the DAQ must be able to determine which AURORA block belongs to which chip. This is accomplished by adding two chip ID bits immediately after the NS bit at the start of every AURORA 64 bit block (before scrambling). Thus,

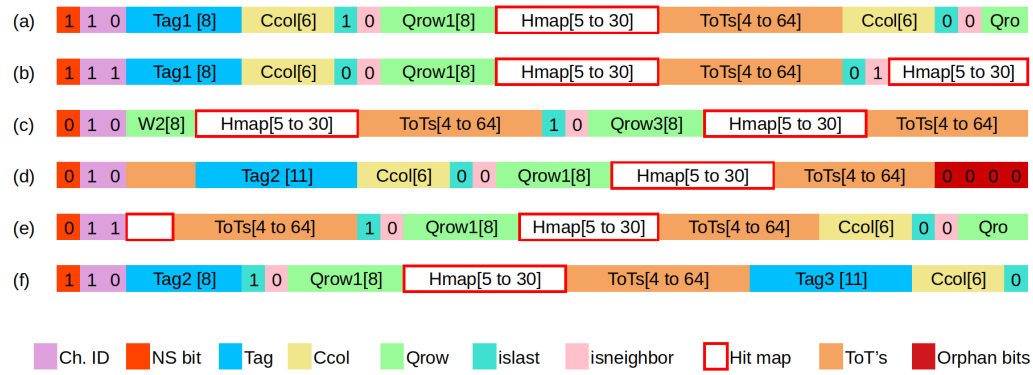

Figure 47: Example of encoded and merged data outputs from two chips with ID LSBs 10 and 11. Six AURORA blocks are shown (a-f), four belonging to chip 01 and two to chip 11 (blocks b and e). Chip 10 has two streams containing 3 events. The second event is split across the stream boundary. The first stream ends with four orphan bits seen at the end of block (d). The first event has two hit Ccols with one hit Qrow in the first and two in the second. The second event starts with an internal tag in block (d) and continues on a new stream in block (f). It has two hit Qrows in the same Ccol. The third event starts in block (f) with an internal tag. Chip 11 has one event with three hit Qrows in the first hit Ccol, the second Qrow being a neighbor of the first. There may also have been AURORA idles or non-data words (such as register readback), which would have been removed or split off by the decoder and are not part of the streams.

instead of 1/64 overhead from the NS bit, one now will have 3/64 overhead (NS bit plus 2 ID bits). These two ID bits can be the least significant bits of the wire bonded chip ID. All other aspects of the stream encoding remain the same. Fig. 47 shows the bit content of a hypothetical merged data output containing two streams, one from chip ID=10 and another from chip ID=11, extending across multiple AURORA blocks.

Because the stream protocol respects AURORA blocks, the decoder just needs to combine all blocks with the same ID in order to reconstruct the streams from that chip. Multi-chip encoding is the default setting, as the presence of ID bits upon power up will be a nice diagnostic tool even when not using data merging. For maximum data transmission efficiency, single-chip encoding would be selected upon configuring the chip.

10.8 Event size limit and data filtering

Unphysically large events due to exception conditions can cause readout problems and it may be desirable to suppress them. Two levels of truncation are available, applied prior to AURORA encoding, such that unwanted data are discarded as early as possible. The first is at the core-column level, where a maximum number of bits (in multiples of 8) allowed for any single core column can be programmed. Additional data will be discarded prior to encoding. When this happens, the fake qrow number ?? will be added with the islast bit set, to signal to the DAQ that column truncation has taken place. This protects against global occupancy extremes, but not against uniform high (but not extreme) occupancy everywhere.

The second truncation mechanism is a readout timeout for each event. If the time elapsed since the readout of an event started reaches a programmed threshold, any further data in that event will be cleared and the event readout will be ended. When this happens a special tag is added to communicate to the DAQ that this has taken place, as indicated in Sec. 9.2.

In addition to event truncation, it can be desirable to filter out hits known to be backgrounds. A filter is implemented to discard single hit clusters with ToT less than a programmed value. A single hit cluster is identified by checking for hits in its neighbors within one core column and not across core column boundaries. The processing cannot tell if pixels on the edges of the core column have hit neighbors or not. Thus, less than 100% of background single hit clusters can be discarded by this filtering. As the purpose of this filtering is to reduce the impact of backgrounds on bandwidth utilization, the benefit from filtering 100% of backgrounds was not worth increased complexity needed. Furthermore, not rejecting a fraction of the background (which can of course still be rejected off-line if desired), has the benefit of allowing to monitor the background level and the effectiveness of the on-chip filtering.

The "neighbor map" used for this filter depends on the bump bonded sensor and possibly on user requirements not known in advance. Thus, the desired neighbor map must be programmed as a bit pattern in the chip configuration. The neighbor bit pattern tells the filter which pixels to consider as a given pixel's neighbors. Two 16-bit mask patterns are programmed, one for even columns and one for odd columns, as illustrated in Fig. 48. To keep the mask to 16 bits, which conveniently fits one global register, the second pixel in the same row and same column pair is always considered a neighbor and, therefore, does not require a mask bit. The 16-bit mask allows selection of diagonal neighbors as well as up/down and left/right. Fig. 48 shows the more complex case of a 25x100 sensor. For a 50x50 sensor, the mask bits selecting all neighbors (including diagonal) would be 1,2,3,7,11,12,13 (2,3,4,8,12,13,14) for even (odd) columns.

Additionally, there is one filter enable bit for each column in the chip (as opposed to one per column within a core) to give the user full flexibility for where to use the filter. In addition to disabling the filter of the edge columns of a core, one can, for example, disable the filter for columns where dead pixels would cause real non-isolated hits to appear isolated.

10.9 Precision ToT data

The Precision ToT block is present for every core column and generates four times 16 bits of data, one for each HitOr bus in the core column. These data are stored and triggered the same way as normal hit data. The 16 bits consist of 5 Time of Arrival (ToA) bits and 11 ToT bits, as explained in Sec. 13.6. However, for the purposes of readout, they are considered as a set of four 4-bit fragments. Thus there are sixteen 4-bit fragments. In this way the data can be encoded for readout exactly the same way as a pixel quarter core containing 16 pixels, each with its 4-bit ToT. There will be a compressed hit map, which is not actually mapping hits but simply indicating which of the 4-bit fragments are non-zero, followed by all the non-zero 4-bit fragments. For any given event, some of the four HitOrs may not have fired at all, which is the same as a quarter core without any hits and entirely suppressed from the readout.

10.10 Format Options

The stream format described in so far and exemplified in Fig. 44 is designed for maximum lossless

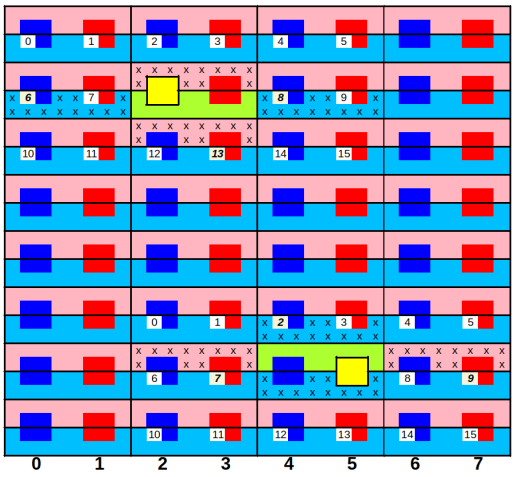

Figure 48: Bump pattern for one core bump bonded to a 25x100 pixel sensor. Column numbers are shown along the bottom. The red (blue) square bump pads are connected to the pink (light blue) pixels. Two pixels/bump pads are highlighted in lime/yellow and the 16 pixels identified by the 16-bit neighbor mask for each are numbered. The up,down,left,right neighbors (hatched) of the upper yellow pixel are selected by setting mask bits 6, 8, and 13; while for the lower yellow pixel by setting 2, 7, and 9. The companion pixel in the same column pair (not numbered) is always set as a neighbor regardless of the mask.

Option	Default	Description	Section
Chip ID	on	Include Chip ID[1:0] after NS bit	10.7
End of stream marker	on	000000 w/enable bit	10.5
BCID and Trig ID	off	Insert BCID[10:0] counter value and internal trigger ID[7:0] after tag	9
Uncompressed map	off	Do not compress hit maps. Map is always 16 bits	10.4
No ToT	off	Leave out ToT values- hit/no hit information only	10.4

Table 11: Optional elements that can be enabled/disabled and affect stream efficiency.

efficiency in bandwidth utilization. More information can be added for debugging or for other functionality when such high efficiency is not needed, or conversely when lower bandwidth utilization must be obtained. Table 11 collects the available options. They can be used in any combination.

10.11 Tags 251 and 252: Exceptions and Reporting

Tags 251 and 252 mean that an exception has taken place, such as a buffer overflow, as shown in Table 7. If autoread of such exceptions is enabled (see Sec. **??**), these tags will be added to the data stream and followed by a 24-bit record of what happened, every time an exception occurs. If autoread is disabled, they can be added to the data stream using the read trigger command, in order to the check the cumulative count of such errors. The meaning of the 24-bit payload for each of these special tags is given in Table 12.

Tag (decimal)	**Payload (24-bits)**
252	description
254	description

Table 12: description of the payload for tags 252 and 253.

11. Multi-Chip Data Aggregation

11.1 Data receivers

11.2 Setup and operation

11.3 Data flow, alignment, and idles

12. Sensing and Monitoring Functions

12.1 Multiplexed analog outputs and ADC inputs

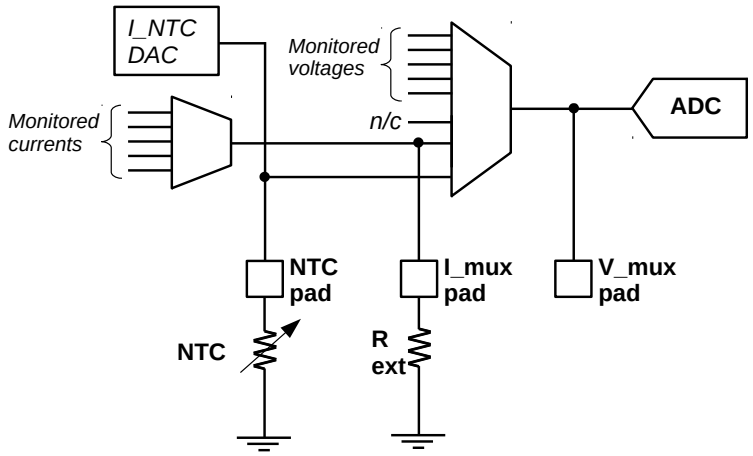

Figure 49: Analog multiplexed outputs and inputs to ADC.

12.2 General Purpose ADC

ADC conversion read-back is always the last value converted. Whatever the result was of the most recent conversion will be read back over and over. In case the auto-read register is configured to be the ADC value in order to automatically monitor some value vs. time, one will need to trigger a new conversion periodically in order to be able to see any time variation.

13. Test and Miscellaneous Functions

13.1 General purpose LVDS and CMOS outputs

RD53B contains four LVDS differential outputs and one CMOS output purely for testing. They will not be used in detector operation.

13.2 Bypass mode

Bypass mode allows to control the chip without the internal PLL Clock and Data recovery function. This mode can only be selected by driving a wire bond pad to high. In bypass a 160 MHz command clock must be supplied in addition to the Command stream, as well as high speed clock to drive the output data serializer. An external reset signal must also be supplied, as the command activity detector is no longer active. The data merging LDVS inputs are taken over for these external clock functions, thus no data merging is possible in bypass mode.

13.3 Scan Chains

13.4 Hit OR

Just as in RD53A, within each RD53B core column there are four independent Hit OR nets, each one fed by one quarter of the pixels. Fig. 50 shows graphically how the 64 pixels in one core are grouped into the 4 OR networks. The figure also indicates two possible sensor formats of 50 μm $\times$ 50 μm (50x50) or 25 μm $\times$ 100 μm (25x100) pixels. It can be seen that in the 50x50 case, a given pixel in network 1 has its two up-down neighbors on network 3, and its left-right neighbors on 2 and 4. Conversely, a given 25x100 pixel on network has has its left-right neighbors in network 3 and its up-down neighbors on 2 and 4.

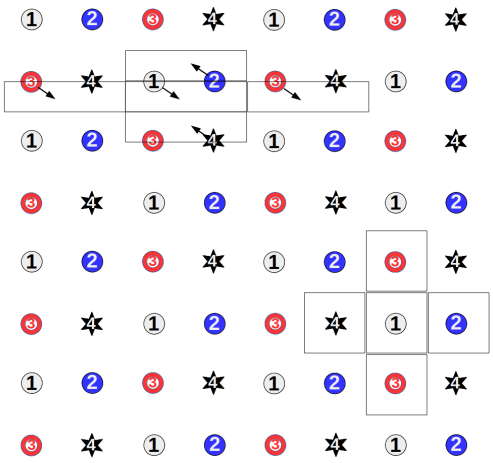

Figure 50: The four Hit Or nets in a 64 pixel core.

In the default mode, each net forms the logical OR of all individual pixel outputs that have been enabled by the HitOr mask bit (one bit per pixel). New in RD53B there may be special modes that allow selecting other pixel digital signals to be ORed into the HitOr net. Options are:

- The hit signal ANDed the negated pixel ToT counter enable signal, such that only a very short leading edge pulse is produced.
- The hit signal ANDed with the pixel ToT counter MSB, such that a (delayed) signal is produced only for high charge hits.

13.5 Heartbeat and test patterns

13.6 Precision ToT module

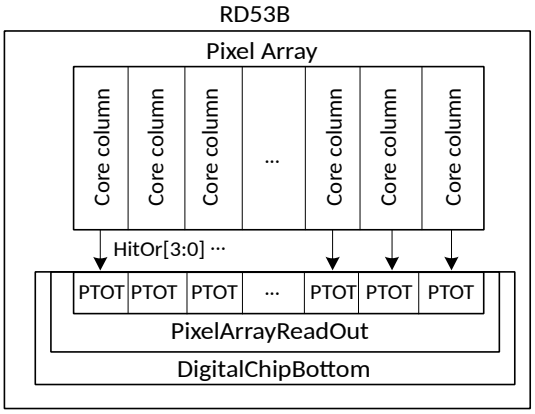

Figure 51: PTOT modules, one per core column.

The Precision ToT (PTOT) module makes measurements on the HitOR signals (13.4) coming out of each core column. There is one PTOT per core column as shown in Fig. 51. Two quantities are measured:

- ToT, just like in the pixels, but with an 11 bit counter counting as 1280 MHz effective rate,
- Time of Arrival (ToA) of the leading edge, as a phase difference from HitOr leading edge to the next BX clock rising edge, with a 5 bit counter counting at 1280 MHz effective rate.

These quantities are stored in memories just like the pixel region memories, associated to latency buffers that keep track of time just as in the pixel regions. A diagram the PTOT module is shown in Fig. 52. The latency buffer depth and logic are the same as in the pixel regions. The readout of the PTOT data is trigger based, exactly as for regular pixels, and the data are included in normal data path as described in Sec. 10.9.

13.7 Capmeasure circuit

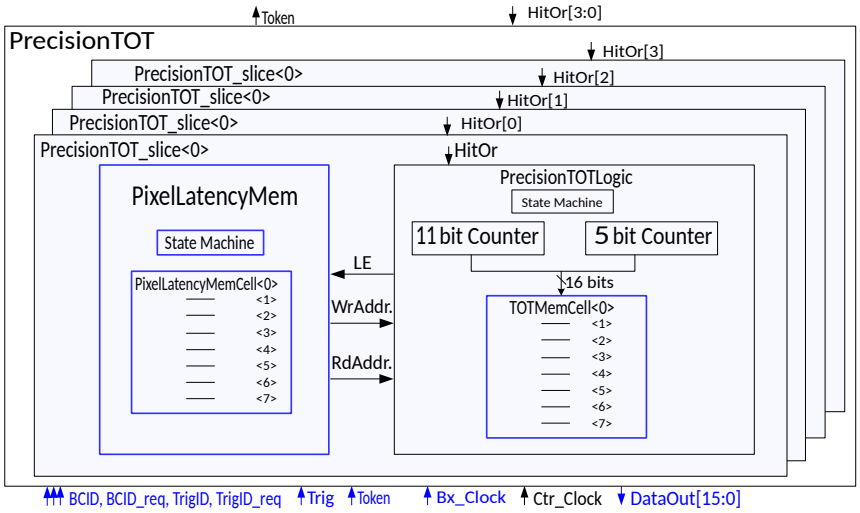

Figure 52: Single PTOT module diagram. The elements outlined in blue are copies (same code) of the pixel region logic.

14. Clock Generation and Data Recovery Technical Details

15. Digital Chip Bottom Technical Details

15.1 Data flow and buffering

15.2 AURORA encoding and decoding

16. Wire Bond and Bump Bond Pad Design (not pinout)

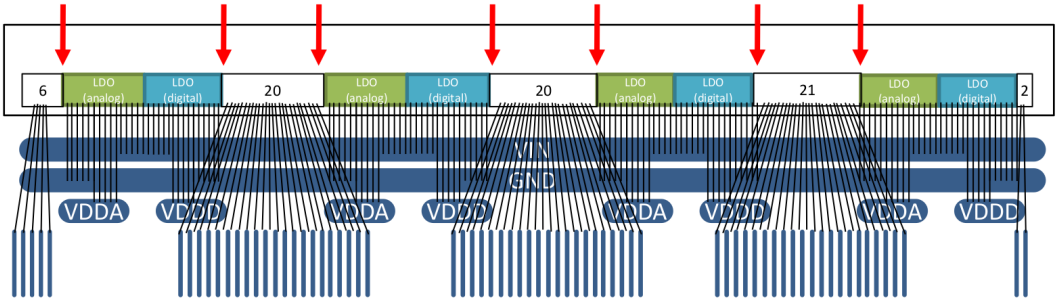

Figure 53: Organization of wire bond pad frame.

17. Reference Tables (pinouts, configuration, etc.)

17.1 Internal Component Values

Component	Type	Value	Tolerance	Reference
Injection capacitor	MOM	8.5 fF	±10%	Fig. 21
Diff FE feedback cap (high gain)	parasitic	3.73 fF	??	Fig. 17
Diff FE feedback cap (low gain)	parasitic	3.28 fF	??	Fig. 17
Lin FE feedback cap	5.5 fF	MOM	±10%	
filter capacitors				
Analog ShuLDO R_int	??	??	??	Sec. 4.1
Digital ShuLDO R_int	??	??	??	Sec. 4.1
Resistive temp sensor	??	??	??	
Internal terminations				
Chip ID pull up resistors	Poly	40 k	10%	to VDDD
IREF trim pull up and down resistors	Poly	40 k	10%	to VDDA or GND

Table 13: Internal passive component types and values.

17.2 External Component Nominal Values

Component	Function	Value	Reference
R_{iref}	Current reference resistor	30 K	Fig. 10
$R_{vref}A$	Sets analog regulator ref. V.	30 K	Fig. 10
$R_{vref}D$	Sets analog regulator ref. V.	30 K	Fig. 10
$R_{shunt}A$	Sets shunt slope	400 Ω	Figs. 8, 9
$R_{shunt}D$	Sets shunt slope	400 Ω	Figs. 8, 9
R_{ofs1}	Sets the shunt offset V.	50 K	Figs. 8, 10
R_{ofs1}	Sets the low power offset V.	10 K	Fig. 10

Table 14: External component nominal values.

17.3 Command and Data Encoding

Command	Encoding		(T)ag, (A)ddress or (D)ata 5-bit content					
Sync	1000_0001	0111_1110						
PLLlock (noop)	1010_1010	1010_1010						
Trigger	tttt_tttt	Tag[0..53]						
Read_trigger	0110_1001	ID<4:0>	00,T<7:5>	T<4:0>				
Clear	0101_1010	ID<4:0>						
Global Pulse	0101_1100	ID<4:0>						
Cal	0110_0011	ID<4:0>	D<19:15>	D<14:10>	D<9:5>	D<4:0>		
WrReg(0)	0110_0110	ID<4:0>	0,A<8:5>	A<4:0>	D<15:11>	D<10:6>	D<5:1>	D<0>,0000
WrReg(1)	0110_0110	ID<4:0>	1,A<8:5>	A<4:0>	N×(D<9:5>	D<4:0>)		
RdReg	0110_0101	ID<4:0>	0,A<8:5>	A<4:0>				

Table 15: This is a duplicate of Table 2. List of protocol commands/frames and address or data fields associated with each. Unused padding bits are indicated by "0". Double vertical lines denote frame boundaries. tttt_tttt is one of 15 trigger commands (Table 3). The before-encoded bit content of chip ID, Address or Data is shown. These are all encoded as 8-bit data symbols (Table 17). Not backwards compatible with RD53A.

Symbol Name	Encoding	Trigger Pattern	Symbol Name	Encoding	Trigger Pattern
			Trigger_08	0011_1010	T000
Trigger_01	0010_1011	000T	Trigger_09	0011_1100	T00T
Trigger_02	0010_1101	00T0	Trigger_10	0100_1011	T0T0
Trigger_03	0010_1110	00TT	Trigger_11	0100_1101	T0TT
Trigger_04	0011_0011	0T00	Trigger_12	0100_1110	TT00
Trigger_05	0011_0101	0T0T	Trigger_13	0101_0011	TT0T
Trigger_06	0011_0110	0TT0	Trigger_14	0101_0101	TTT0
Trigger_07	0011_1001	0TTT	Trigger_15	0101_0110	TTTT

Table 16: This is a duplicate of Table 3. List of trigger command and symbols used to encode the 15 possible trigger patterns spanning four bunch crossings. Note there is no 0000 pattern as that is the absence of an trigger. All trigger symbols are the same as in RD53A.

Symbol Name	Encoding	Data Value	Symbol Name	Encoding	Data Value
Data_00	0110_1010	5'b00000	Data_16	1010_0110	5'b10000
Data_01	0110_1100	5'b00001	Data_17	1010_1001	5'b10001
Data_02	0111_0001	5'b00010	*Data_18*	*0101_1001*	5'b10010
Data_03	0111_0010	5'b00011	Data_19	1010_1100	5'b10011
Data_04	0111_0100	5'b00100	Data_20	1011_0001	5'b10100
Data_05	1000_1011	5'b00101	Data_21	1011_0010	5'b10101
Data_06	1000_1101	5'b00110	Data_22	1011_0100	5'b10110
Data_07	1000_1110	5'b00111	Data_23	1100_0011	5'b10111
Data_08	1001_0011	5'b01000	Data_24	1100_0101	5'b11000
Data_09	1001_0101	5'b01001	Data_25	1100_0110	5'b11001
Data_10	1001_0110	5'b01010	Data_26	1100_1001	5'b11010
Data_11	1001_1001	5'b01011	Data_27	1100_1010	5'b11011
Data_12	1001_1010	5'b01100	Data_28	1100_1100	5'b11100
Data_13	1001_1100	5'b01101	Data_29	1101_0001	5'b11101
Data_14	1010_0011	5'b01110	Data_30	1101_0010	5'b11110
Data_15	1010_0101	5'b01111	Data_31	1101_0100	5'b11111

Table 17: List of command symbols used to encode data values. All symbols are the same as in RD53A except for Data_18, which is shown in italics. The RD53A Data_18 symbol is now the PLLlock command.

Tag values (decimal)	**Meaning**
0-215	extended tags from trigger command
216-219	Self triggers with tag base 55
220-223	Self triggers with tag base 56
224-227	Self triggers with tag base 57
228-231	Self triggers with tag base 58
232-235	Corrupted tag base in trigger command
236	Special tag assigned when received tag is still in use
237-243	additional still-in-use tags
244-255	spares

Table 18: This is a copy of Table 7. Possible extended tag values and their meaning.

Bits	Tag range	Meaning
1	1-215	Tag base autoread on/off
2	216-231	Self trigger autoread on/off
3	232-235	Corrupted tag base autoread on/off
4	236-243	Tag still in use autoread on/off
5	251	Exception record autoread on/off
6	252	Exception record autoread on/off
7	253	column data truncation marker on/off
8	254	event data truncation marker on/off
9	255	End of event on/off

Table 19: This is a copy of Table 7. Configuration bits to control auto-read action for each tag type.

Field	Value	Followed by	Decided by
NS bit	0	anything	data
	1	Tag[8] OR Chip ID[2] then tag[8]	configuration
Tag	0-251	Tag[11] OR ccol[6] OR orphan bits	data
	251, 252	32 bits of metadata	always
	253	Tag[11] OR ccol[6] OR orphan bits	data
	254	Tag[11] OR orphan bits	data
	255	Tag[11] OR orphan bits	config. and data
ccol	1-55	islast[1] then isneighbor[1]	always
isneighbor	0	crow[8]	always
	1	compressed map[var] OR map[16]	configuration
crow	0-193	compressed map[var] OR map[16]	configuration
map	any	ToT[$N_{hits} \times 4$] OR lines below	configuration
ToT (islast)	any (0)	crow[6]	always
	any (1)	Tag[11] OR ccol[6] OR orphan bits	data

Table 20: Stream decoding reference. Each stream begins with an NS bit set to 1. The fields following the NS bit are identified in this table.

Register	Bits	Default	Meaning
AA	0-3	0	number of events per stream (EPS) (0 means no fixed number of events per stream)
AA	4-9	3	E-step starting value. Only used if EPS=0.
AA	10-12	3	E-step increment. Only used if EPS=0.
BB	0	1 (on)	chip ID on/off **CRITICAL**
BB	1	1 (on)	hitmap compression on/off **CRITICAL**
BB	2	1 (on)	Include ToT values on/off **CRITICAL**
BB	3	1 (on)	End of stream marker 000000 on/off

Table 21: Configuration bits to control output data format options including those in Table 20. See also Table 19 for which special tags are allowed.

References

[1] RD53 Collaboration, "RD53A Integrated Circuit Specifications," CERN-RD53-PUB-15-001 (2015).

[2] Xilinx, "Aurora 64B/66B Protocol Specification," SP011 (v1.3) October 1, 2014.

[3] M. Karagouins *et. al,* "An Integrated Shunt-LDO Regulator for Serial Powered Systems," in Proc. of IEEE ESSCIRC '09, (2009).