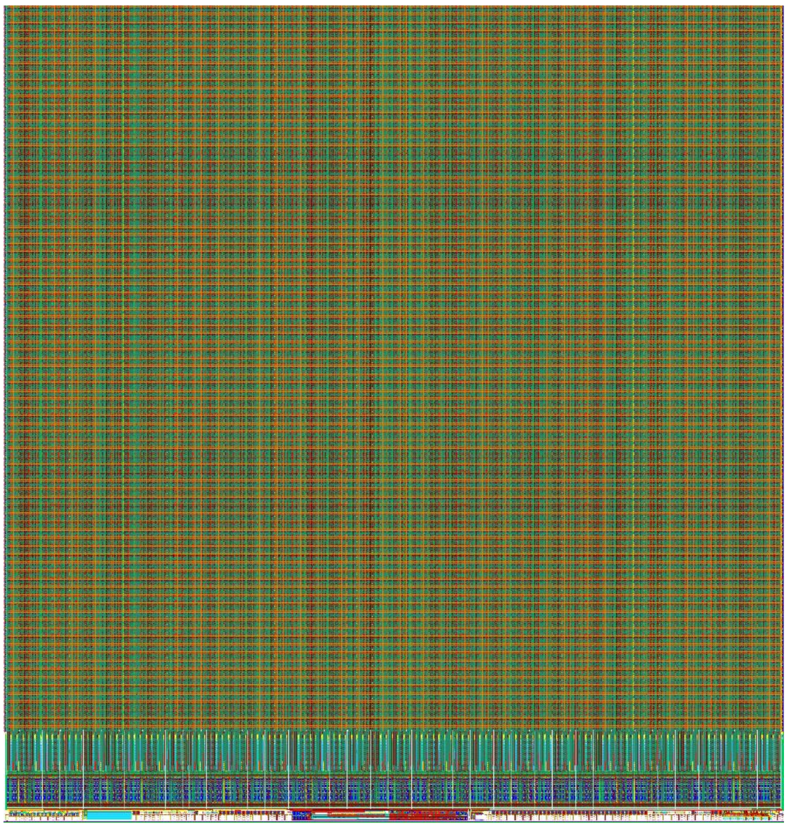
ATLASPIX3 user manual

Version 3

20. 05. 2022



ATLASPIX3 has a pixel matrix with 132 columns and 372 rows. Pixel size is x = 150µm, y = 50µm. The chip size is x = 20.2mm, y = 21mm. The chip has been implemented in 180nm HVCMOS technology of TSI. 200Ωcm wafers have been used.

Chip layout is in **Figure 1.**

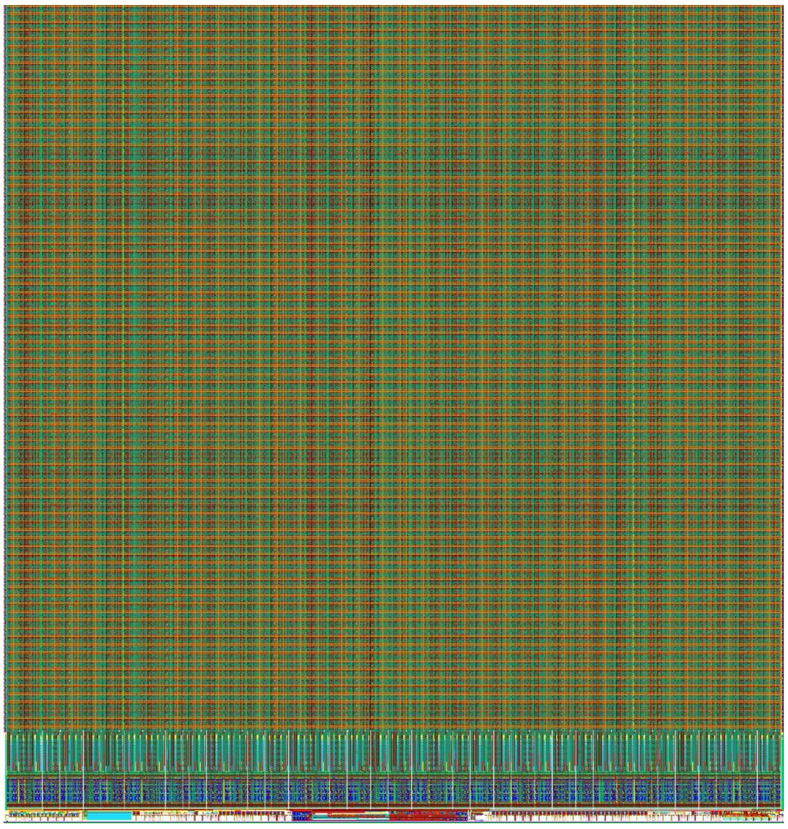


Figure 1: Chip layout

One column contains 372 pixels, a configuration register block, 372 hit buffers, 80 trigger buffers and two end of column (EoC) blocks. EoC1 is attached to hit buffers and EoC2 to trigger buffers.

The block scheme of the pixel matrix and column circuits is shown in **Figure 1B.**

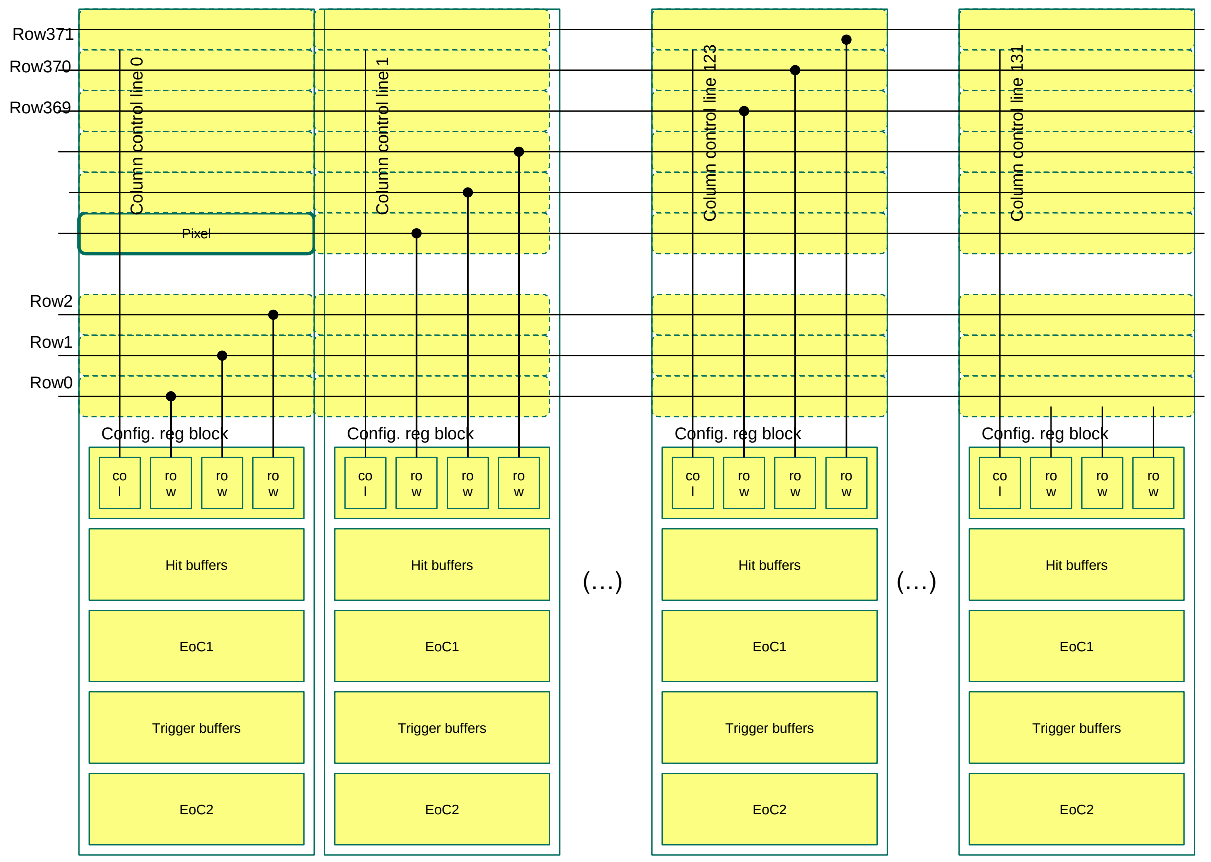


Figure 1B: Block scheme of the pixel matrix and column circuits.

The pixel schematics is shown in **Figure 2.**

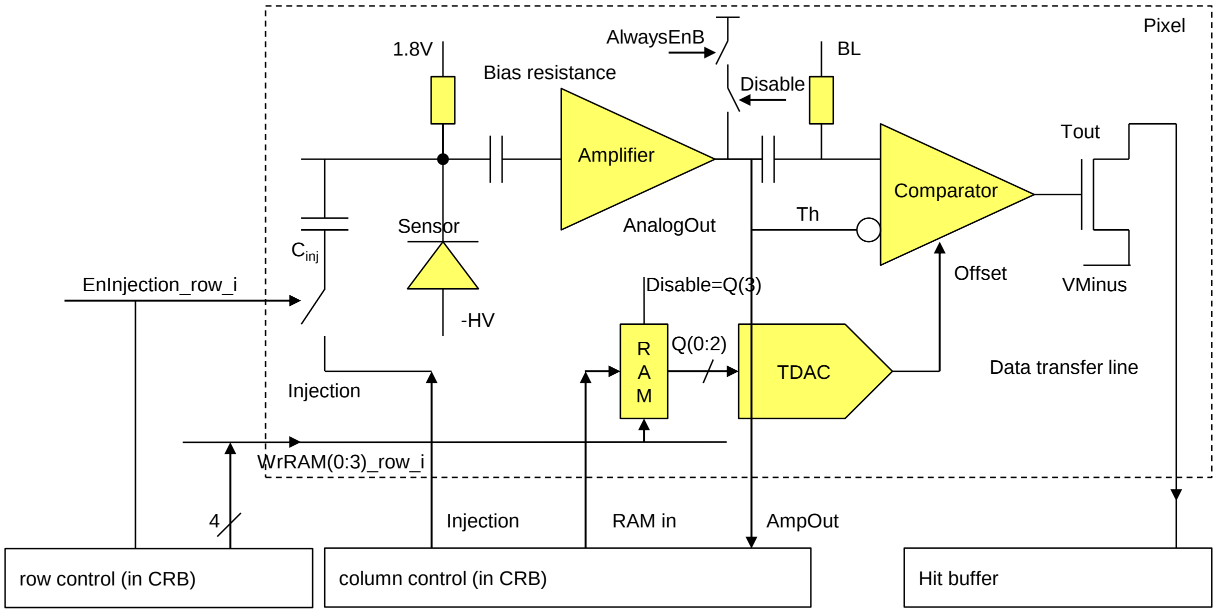


Figure 2: Pixel schematics

Schematics of configuration register block is shown in **Figure 3.**

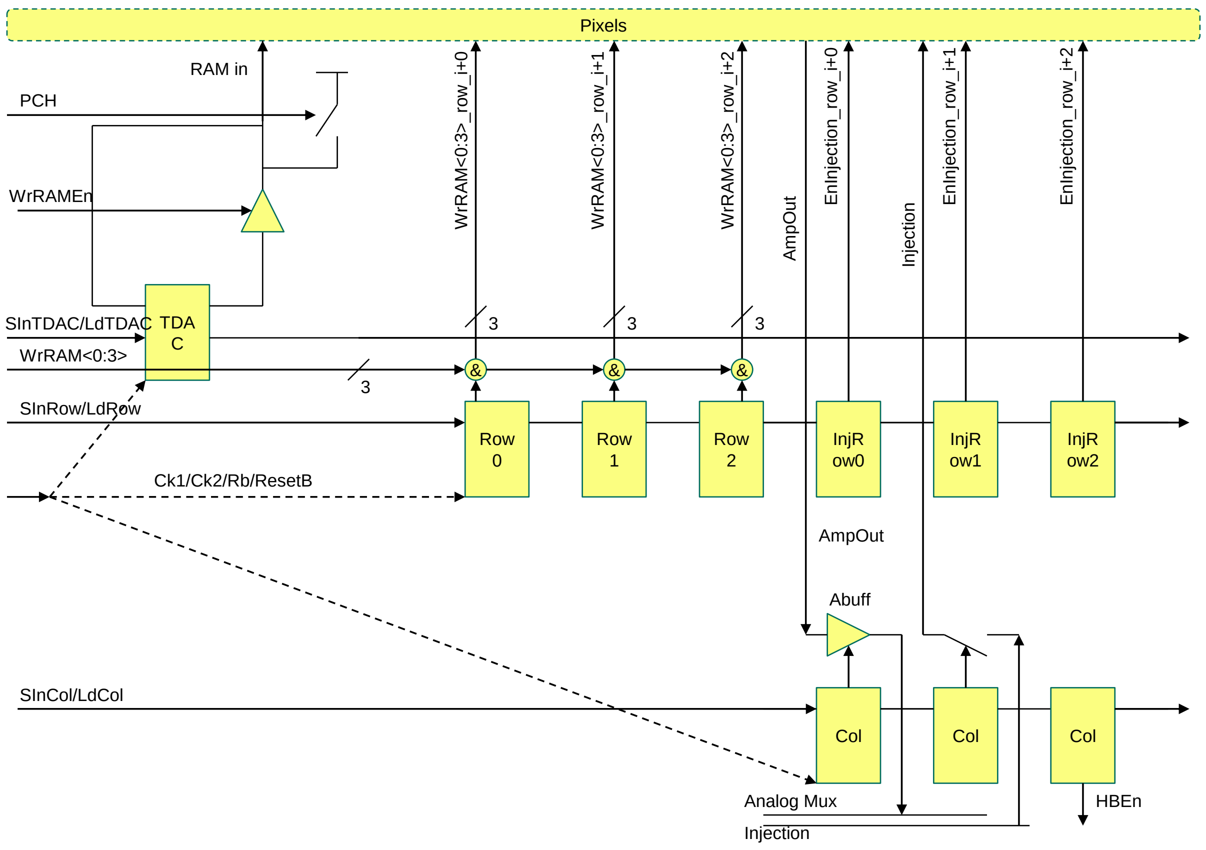


Figure 3: Schematics of configuration register

Configuration register block

One configuration register block (CRB) is used for the pixel column where it is placed and for three pixel rows. The CRB in column i will be connected to column i and to rows 3i, 3i + 1 and 3i + 2. This controlling scheme is illustrated in Figure 1B. In this way we avoid a row control register placed along the vertical matrix edge, as was used in ATLASPIX1. CRB (Figure 3) has a row-register segment (6 bits), a column-register segment (3 bits) and TDAC-register segment (one bit). The register segments of different columns are connected in form of shift registers. Hence, there are three separated shift registers for the matrix control. The registers use three different global load signals LdRow, LdCol, LdTDAC (generated by the command processor CP and control unit CU). All registers have same serial input, same clocks (Ck1 and Ck2) and same read back and reset signals (global signals generated by CP). The structure of the register cell will be explained in the next session.

Row register

The purpose of the row register is to produce WrRAM(0:3)_rowj signals for three rows. The register bits select the rowj. Bit index is derived from the global signals WrRAM(0:3) generated by CP.

Row register also produces also EnInjection_rowj signals.

Column register

The purpose of column register is to enable analog buffer used to amplify the pixel output of the bottom most pixel, to enable injection into the column and to enable the hit bus output of this column.

TDAC register

The purpose of TDAC register is to store and generate RAM in signal (bit line of the pixel RAM) and to provide read back possibility for the pixel RAM. For this purpose, two additional global signals (generated by CP) are used: WrRAMEn (enables writing into RAM) and pre charge enable (PCH).

The following table shows the bits of the registers

Row register table

bit // purpose

SIN ->

bit0 // enrow (0 + 3*colindex)

bit1 // enrow (1 + 3*colindex)

bit2 // enrow (2 + 3*colindex)

bit3 // eninjrow (0 + 3*colindex)

bit4 // eninjrow (1 + 3*colindex)

bit5 // eninjrow (2 + 3*colindex)

- ➔ SOUT

Write signal for RAM is calculated in the following way:

WrRAM(i)_row j = enrow(j) & WrRAM(i)

WrRAM(i) are global signals generated by CP.

For writing into RAM signal WrRAMEn = 1, PCH = 0

For reading from RAM signals Rb should be 1, and WrRAMEn should be 0. Precharge sequence should be generated. The read and write RAM sequence will be explained in detail later.

Column register table

Bit // purpose

SIN ->

bit0 // enable analog buffer

bit1 // enable injection

bit2 // enable hitbus

- ➔ SOUT

TDAC register table

Bit // purpose

Bit0 // used to store TDAC bit and for read back

Register cell

The bits of the registers are stored in triple redundant register cells (output latches). Schematic of this register cell is shown in **Figure 4**.

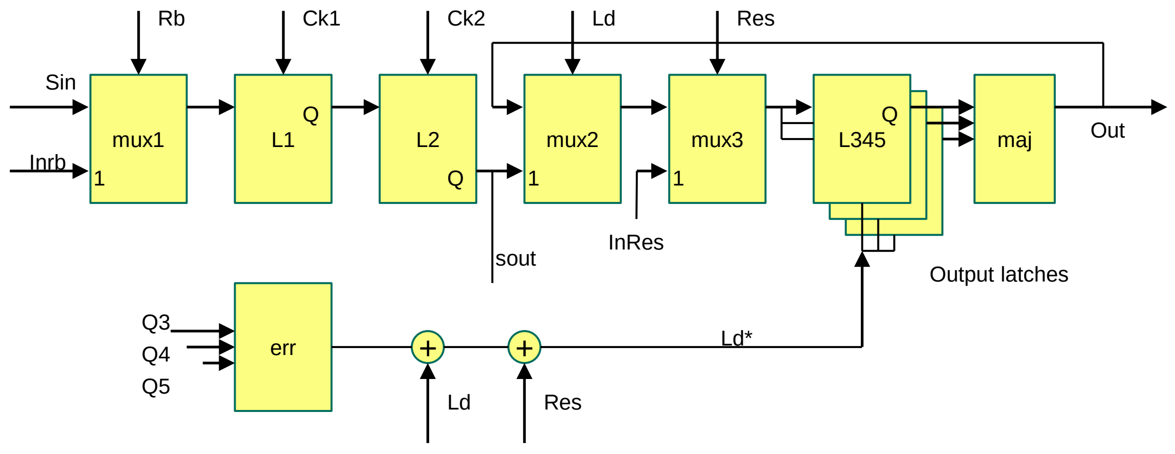


Figure 4: Schematic of the register cell

The scan - flip flop part of the cell is used for shifting of the bits. (The output Sout is connected to the Sin of next cell.) The two latches (L1 and L2) are forming the flip flop and they use two separated clock input signals Ck1 and Ck2. With such a non overlapping clock structure (Ck1 and Ck2 should never be 1 at the same time), we avoid hold time violations. The multiplexer mux1 selects either the serial input Sin (for Rb = 0) or the read back input Inrb (for Rb = 1) for the L1.

Loading of state into output register: When input Ld = 1, the Q output of L2 (serout signal), will be passed (through mux2 and mux3) to inputs of three output latches (L3/L4/L5). Their load signal Ld* will be high (because of OR-gate "+") and the serout value will be stored in all three latches (triple redundancy). The main output "Out" is calculated by majority logic (maj) from the states of L3, L4 and L5.

Reset: When input Res = 1, input signal InRes will be passed to the inputs of three latches. Signal Ld* will be high (because of OR-gate "+") and the InRes value will be stored in three latches. Reset is used to put the register into a desired state after power on reset.

Repairing of bit flips: When register is neither loaded nor reset, the input of the three latches will be the output of the majority logic. The error logic (err) checks whether all three signals Q3, Q4 and Q5 are equal. Imagine, we have a bit flip in one of the latches (L3, 4 or 5) due to SEU. Main output Out will still give a correct value. Error logic output will go high. This will set Ld* = 1 and the value Out will be loaded into three latches. This will restore the correct value in the latch that had bit flip.

For all cells, the read back input Inrb is connected to Out, except for the cells in TDAC register where Inrb is RAMin. Initial states (InRes) are written in tables if they are not 0.

Chip periphery

The chip periphery has the following circuits:

Main control unit (for triggered readout)

Control unit for untriggered readout

Clock and data block (customized digital part)

Configuration shift register

Bias block with the DAC shift register

Voltage DAC block with the VDAC shift register

Voltage and power regulators

Pads

Clock and data block (customized digital part)

Figure 5 shows the clock and data block (CDB) (customized digital part).

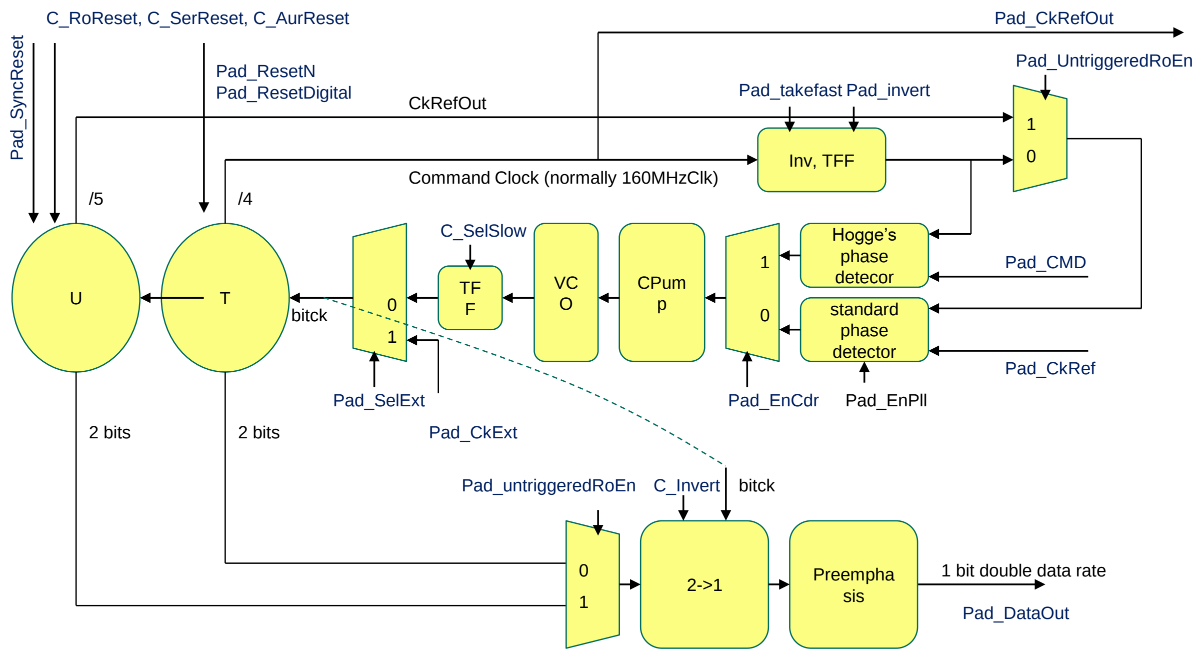


Figure 5: Clock and data block (CDB)

Signals in blue with prefix "Pad_" are coming from pads. Signals in blue with prefix "C_" are stored in the configuration shift register. The purpose of the customized digital block is to provide clock ("bitck") for the control units. In the case of triggered control unit, bitck is 640MHz, in the case of untriggered CU, bitck is 800MHz. Bitck can be generated by three ways: 1) directly from Pad_CkExt. For this, Pad_SelExt must be logic 1 (1.8V). 2) Using oscillator, PLL and the standard phase detector from Pad_CkRef. For this, Pad_EnPll must be 1 and Pad_EnCdr = 0. 3). 3) Using oscillator, PLL and the Hogge's phase detector from Pad_CMD. For this, Pad_EnPll must be 1 and Pad_EnCdr = 1. More details will be given later.

Main control unit

Figure 6 shows block scheme of the main control unit (CU). The figure also shows the shift registers of ATLASPIX3 (TDAC-, Row-, Col-, DAC-, VDAC- and Config.-SR) and the Clock and Data Block.

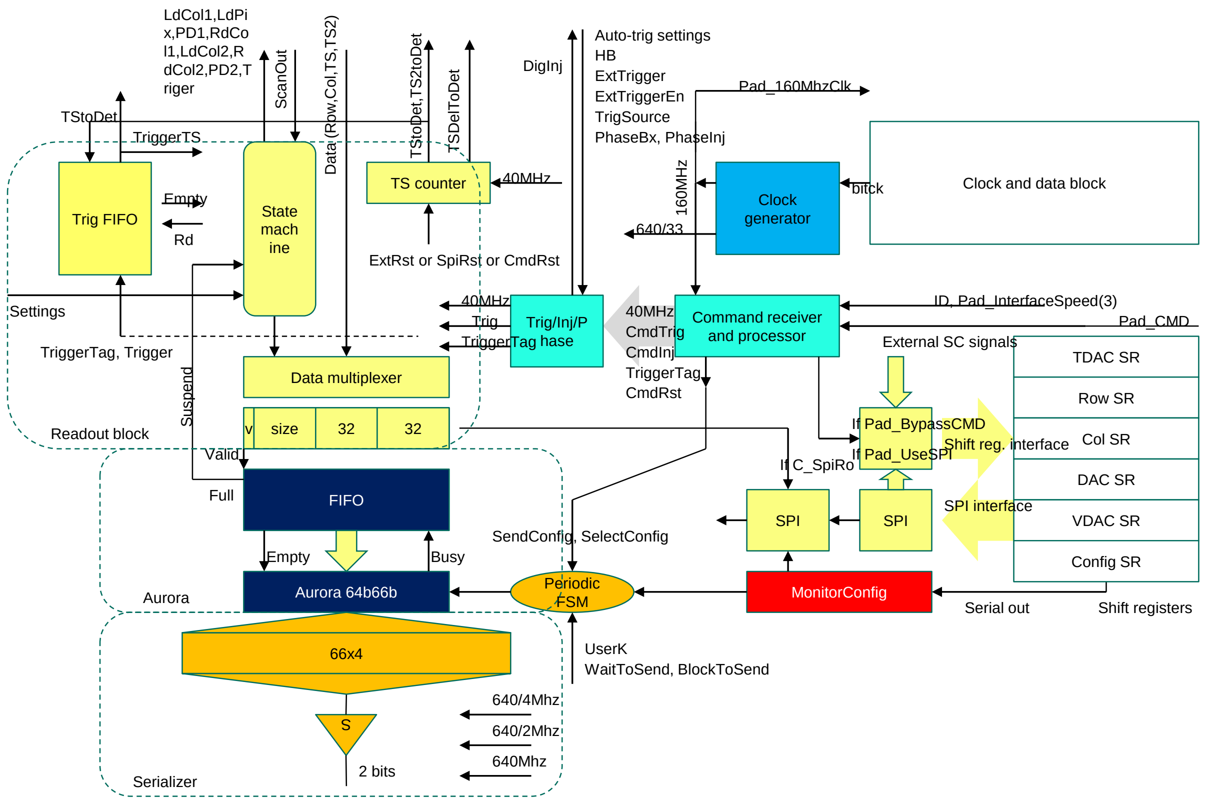


Figure 6: block scheme of the main control unit (CU)

The CU contains:

Clock generator. This block receives 640MHz bit clock and generates 160MHz command clock and a clock with the frequency 640/33 used by Aurora transmitter.

Command receiver and processor (CP). The purpose of the block is to receive and process commands. The commands are received via Pad_CMD and the block uses 160MHz clock provided by the clock generator. The commands use special encoding, like Rd53, and the bit rate is 160 Mbit/s. The commands are sent in 8 bit blocks. Every block encodes a command type, or 5-bit word that can be chip ID, register address or data. Command processor generates 40MHz clock (bunch crossing clock) from the data stream. There are three command types: trigger command, RW register command and SET bit command.

The trigger command generates trigger signal (CMDTrigger) and trigger tag (triggerTag).

RW register generates sequence on shift register interface signals: SIn, Ck1 and Ck2 that shifts in 10 bits of data. The data that should be shifted are in the data part of the command. The speed of the shifting can be reduced by setting Pad_InterfaceSpeed(3) to 1. Normally, the state machine frequency is 10MHz; with Pad_InterfaceSpeed(3) = 1 it is 8 times slower. Shifting in of one bit requires 4 clock cycles.

The SET bit command writes ten bits in a triple redundant 10 bit command-register. These bits are used for shift register interface (for instance for load signals) and as internal configuration bits (e.g. to select sending of configuration bits that were read back (they are stored into a MonitorConfig regsiter) - select_config_int, to initiate injection CMDInjection, or to reset time stamp counter ResetTS_int). The commands and bits will be explained in detail later.

The block Trigger/Phase generates auto-trigger and adds additional delay (changes phase) of the injection, 40MHz clock, trigger and trigger tag. There are many input and configurations signals for this block. Auto trigger settings (stored in configuration register) define the trigger delay and trigger width. BX- and Injection phase (also stored in configuration register) change the phase of bunch crossing clock/trigger/trigger tag and injection. Trigger source (stored in configuration register) defines how the trigger is done: 0 and 3: from trigger command, 1: using auto trigger generator that is initiated by hit bus and 2: using auto trigger generator that is initiated by injection. ExtTriggerEn selects external trigger signal.

The readout block contains the readout state machine and the data multiplexer. This block receives the signal ScanOut, pixel addresses (row and column), leading edge time stamp (TS) and time over threshold time stamp called TS2[1] from the matrix and generates matrix control signals (LdCol1, LdPix, PD1, RdCol1, LdCol2, RdCol2, PD2, Triger). The block also receives empty signal from trigger FIFO. It has various settings stored in configuration register. The block generates 64-bit data word (consists of two 32-bit half-words), 8-bit size-word that indicate data size. Additionally a data-valid bit is generated as notice for transmitter that data are present. The data format will be explained later.

Aurora transmitter receives input-words from the readout block (64-bit data word + 8-bit size word). It scrambles the data and adds 2 extra bits. In this way 64 to 66 conversion is done. There is the possibility to include from time to time a user-defined 64-bit word into the data stream. This user-defined word can be the user-K-word, defined in the configuration register, or the state of the MonitorConfig register. The period of sending and the number or repetitions can be programed. It is also possible to force sending of the user defined word by a SendConfig command. The output of the Aurora block is 2 bit, and the output bit rate is 640Mbits/s. The final 2->1 seriallizer is placed in the customized digital block. The bit rate at the final output is 1.28Gbits/s. Mode details about Aurora will be described later.

MonitorConfig register is used to store the values of the shift registers (TDAC-, Row-, Col-, DAC-, VDAC- and Config.-SR) after read back. The bits will be explained later.

SPI register is a debug feature. It allows us to configure the chip without command decoder and to readout both the MonitorConfig register and the data words without Aurora transmitter. The bit positions will be explained later. SPI register is written and read back using SPI interface: signals MISO, MOSI, SCK and CSB.

Generally, the important signals such as: interface signals to shift registers, injection, reset of time stamp counter can be generated from three sources: from SPI register if Pad_UseSPI = 1, from pads (external signals) if Pad_BypassCMD = 1 and by command processor. Internal signals SelectConfig and SendConfig are 1 and 0 when Pad_UseSPI = 1 and Pad_BypassCMD = 1.

Starting the chip

Ideal sequence (triggered readout, clock recovery, Hogge's phase detector)

1 The chip has a power on reset pad. The pad can be connected on PCB to the Pad_ResetDigital. Power on reset puts the configuration and bias DAC/VDAC registers into correct state, so that oscillator and PLL can work.

For clock recovery mode we need following settings: PadEnCdr = 1, PadEnPll = 1, PadUntriggeredRo = 0, PadTakeFast = 0, PadInvert = 0, PadSelExt = 0.

[1] Time over threshold should be calculated as the difference between TS2 and TS. Notice that TS and TS2 can have different periods.

In this sequence PadCkRef and PadCkExt are not used. The Hogge's phase detector locks to the training pattern sent at PadCMD. After phase lock, command receiver can understand commands sent at PadCMD. Phase lock can be observed by measuring PadCkRef out. This is the 160MHz command clock, its rising edge should be in the middle between two data transmissions at PadCMD. The edge of the command clock can be exchanged by setting PadInvert = 1. Training patters are still to be defined. It may possible to lock at SYNC_WORD as well.

Sequence with external reference clock and PLL (triggered readout, standard phase detector)

Step 1 from ideal sequence is performed (power on reset)

Pad CkRef is used to provide externally the 160MHz command clock (the rising edge of the clock should be in the middle between two data transmissions at PadCMD). The settings are: PadEnCdr = 0, PadEnPll = 1, PadUntriggeredRo = 0, PadTakeFast = 0, PadInvert = 0, PadSelExt = 0.

Standard phase detector compares the internal 160MHz command clock (used to sample the commands) and the external reference clock. PLL locks so that two clocks have equal phases. After phase lock, command receiver can understand commands sent at PadCMD.

Sequence with external clock without PLL (triggered readout)

Also here, power on reset should be performed but it is not so critical since oscillator and phase detector are not used.

The settings for this sequence are PadEnCdr = 0, PadEnPll = 0, PadUntriggeredRo = 0, PadTakeFast = 0, PadInvert = 0, PadSelExt = 1. PadCkRef is not used. PadCkExt is used to provide external 640MHz clock. Internally generated command clock can be received via PadCkRefOut and used in FPGA to synchronize the commands.

Use of untriggered readout

The digital part for untriggered readout does not contain its command processor. Configuring of the chip can be done in three ways: 1: with SPI interface (PadUseSPI=1) 2: with external interface (PadBypassCMD=1) 3: using the command processor of the triggered part.

One way to use the command processor is the following:

Standard phase detector is used to compare PadCkRef (should be 160MHz) with the CkRefOut clock generated by untriggered digital part. In this case, the bitck will be 800MHz and CkRefOut 160MHz. On chip generated command clock will be 200MHz and it can be used in FPGA to synchronize the commands.

Commands

At the beginning we will describe the shifting directions:

The command words (CMDword(15:0)) have 16 bits and should be sent send LSB first.

Commands are received LSB first: sin -> command_register(15:0)

Shift registers are filled by command processor MSB first: SR(n:0) <- sin

Shift register bits are received during read back MSB first.

Trigger patterns are sent to trigger LSB first.

Command headers are defined here:

SYNC_WORD 16'b1000_0001_0111_1110

RW_REG 16'b0110_0110_0110_0110

SET_BIT 16'b0110_0101_0110_0101

NO_OP 16'b0110_1001_0110_1001

TRIGGER_01 8'b0010_1011

TRIGGER_02 8'b0010_1101

TRIGGER_03 8'b0010_1110

TRIGGER_04 8'b0011_0011

TRIGGER_05 8'b0011_0101

TRIGGER_06 8'b0011_0110

TRIGGER_07 8'b0011_1001

TRIGGER_08 8'b0011_1010

TRIGGER_09 8'b0011_1100

TRIGGER_10 8'b0100_1011

TRIGGER_11 8'b0100_1101

TRIGGER_12 8'b0100_1110

TRIGGER_13 8'b0101_0011

TRIGGER_14 8'b0101_0101

TRIGGER_15 8'b0101_0110

Data codes are defined here:

8'b0110_1010 -> 5'b00000

8'b0110_1100 -> 5'b00001

8'b0111_0001 -> 5'b00010

8'b0111_0010 -> 5'b00011

8'b0111_0100 -> 5'b00100

8'b1000_1011 -> 5'b00101

8'b1000_1101 -> 5'b00110

8'b1000_1110 -> 5'b00111

8'b1001_0011 -> 5'b01000

8'b1001_0101 -> 5'b01001

8'b1001_0110 -> 5'b01010

8'b1001_1001 -> 5'b01011

8'b1001_1010 -> 5'b01100

8'b1001_1100 -> 5'b01101

8'b1010_0011 -> 5'b01110

8'b1010_0101 -> 5'b01111

8'b1010_0110 -> 5'b10000

8'b1010_1001 -> 5'b10001

8'b1010_1010 -> 5'b10010

8'b1010_1100 -> 5'b10011

8'b1011_0001 -> 5'b10100

8'b1011_0010 -> 5'b10101

8'b1011_0100 -> 5'b10110

8'b1100_0011 -> 5'b10111

8'b1100_0101 -> 5'b11000

8'b1100_0110 -> 5'b11001

8'b1100_1001 -> 5'b11010

8'b1100_1010 -> 5'b11011

8'b1100_1100 -> 5'b11100

8'b1101_0001 -> 5'b11101

8'b1101_0010 -> 5'b11110

8'b1101_0100 -> 5'b11111

SYNC_WORD pattern

SYNC_WORD pattern (16'b1000_0001_0111_1110) should be sent at the beginning of communication, it is used by the receiver to find the phase of the data stream. SYNC_WORD pattern should be also sent between the commands.

RW register command

This command shifts a 10 bit word into **all** shift registers by producing a pattern of signals SIn, Ck1, Ck2.

Notice that the result of this is that a correct pattern will be written in a shift register we want to write but a wrong pattern ends in all other shift registers. This is not a problem, since the circuits that are configured do not use shift register outputs (SOut in Figure 4); they use the main outputs (Out in Figure 4). The main outputs are loaded by signals Ld (Figure 4) and all registers have separated load signals. After shifting of data that are determined for resister X we will issue signal LoadX.

RW command can be also used for read back, since shifting in of new bits, causes shifting out of bits that were stored in latches L1 and L2. The outputs Sout of the last register cells are connected to the MonitorConfig register.

The command RW consist of command header RW_REG (16 bits), double data code for ID and address (address is not used) (2 x 8 bits) and double data code for 10 bits that should be shifted (code: 2 x 8 bits). Chip ID should either match with the hard coded ID or it should be number 16 (broadcast mode). Otherwise the command will be discarded.

To summarize, the following words are sent for one command (shifting LSB first):

CMDword[15:0] = RW_REG

CMDword[15:0] = ID, ADDR

CMDword[15:0] = DATAMSB, DATALSB

Example of the command sequence:

CMDword <= 16'b0110_0110_0110_0110//header RW_REG

CMDword <= 16'b1001_0110_0110_1010//ID 10, address 0

(or CMDword <= 16'b1010_0110_0110_1010//ID 16, address 0)

CMDword <= 16'b1000_1011_1001_1001//code of 00101 01011 or A B

SET BIT

This command is used to write ten bits into one of two command-registers. The bits of this register can be then used for the following purposes: loading of bits into output latches, read back of bits from shift registers, injection, reset of time stamp, RAM-write and pre charge and for the sending of user defined words.

The command consist of command header SET_BIT (16 bits), double data code for ID and register address (2 x 8 bits) and double data code for 10 bits that should be written (code: 2 x 8 bits). Chip ID should either match with the hard coded ID or it should be number 16 (broadcast mode). Otherwise the command will be discarded.

To summarize, the following words are sent for one command (shifting LSB first):

CMDword[15:0] = SET_BIT

CMDword[15:0] = ID, ADDR

CMDword[15:0] = DATAMSB, DATALSB

Example of the command sequence:

CMDword <= 16'b0110_0101_0110_0101//header SET_BIT

CMDword <= 16'b1010_0110_0110_1010//ID 16, address 0

CMDword <= 16'b0110_1100_1010_0110//data example

Bits of the command registers

There are two registers DataOut0 with address 0 and DataOut1 with address 1.

SOut = SOutInt OR DataOut0[0]//used as serial in for the shift registers, notice SOut is OR function of the bit and SOutInt generated automatically from RW_REG command

Ck1 = Ck1Int OR DataOut0[1]//Ck1 for the shift registers (…Int generated automatically from RW_REG command)

Ck2 = Ck2Int OR DataOut0[2]//Ck2 for the shift registers (…Int generated automatically from RW_REG command)

Rb = DataOut0[3]//read back

LdDAC = DataOut0[4]//load for DAC register

LdConfig = DataOut0[5]//load for configuration register

LdVDAC = DataOut0[6]//load for vdac register

LdTDAC = DataOut0[7]//load for tdac register

LdRow = DataOut0[8]//load for row register

LdColumn = DataOut0[9] //load for column register

WriteRAM0 = DataOut1[0]//sets the WriteRAM0 bit of the selected row

WriteRAM1 = DataOut1[1]//sets the WriteRAM1 bit of the selected row

WriteRAM2 = DataOut1[2]//sets the WriteRAM2 bit of the selected row

WriteRAM3 = DataOut1[3]//sets the WriteRAM3 bit of the selected row

Injection = DataOut1[5]//injection

PCHB = DataOut1[6]//PCHB bit (active high, precharge not done when WrRAMEn = 1)

WrRAMEn = DataOut1[4]//enables writing into RAM

ResetTS = DataOut1[7]//reset time stamp

CMDSendConfig = DataOut1[8]//initiates sending of user defined word

select_config = DataOut1[9]//selects MonitorConfig register to be sent, otherwise user K word is sent

Programming of the chip

After introducing of the commands RW register and SET bit, and the bit positions in the command-registers, we described all tools for performing of following tasks:

1) Shifting-in of bits into shift registers and their loading into output latches
2) Read back of the content of output latches
3) Writing into pixel RAM cells
4) Read back from the pixel RAM cells

Shifting-in and loading of configuration bits

For shifting-in of bits we would first send several commands RW register. Number of commands is the next integer greater or equal as SizeOfRegister/10. Notice that some bits will “fall out” from register if SizeOfRegister/10 is not an integer.

After shifting in loading into the target register is performed by setting the load bit = 1 in the command register with command SET bit. After this, the bit should be reset by setting the bit to 0, also using SET bit command.

Read back of the content of shift registers’ output latches

The read back sequence starts with setting of the Rb bit = 1 in the command register (command SET bit). Notice that Rb signal is common to all shift registers on chip.

After this, one RW register command should be performed. The input data for the RW command are here of no importance, the sequence Ck1 and Ck2 will load the main output Out (Figure 4) into shift register laches (latches L1 and L2) of all registers.

The Rb bit should be set to zero with next SET bit command.

One RW register command is performed to shift out the last ten bits of all registers.

These bits will be automatically shifted into the register MonitorConfig. The register size is 64 bits and it has space for last ten bits of all 6 shift registers.

The content of MonitorConfig can be received either as user defined word through Aurora transmitted or using SPI register. The bits in MonitorConfig and SPI will be explained later.

The last three steps are repeated to readout the full shift registers.

Writing into pixel RAM cells

Writing into RAM is done in the following way:

1 Using SET bit command, set WrRAMEn to one. This activates the buffers for RAM writing.

2 Pixel row is selected by writing 1 at the correct place in the row shift register (enrow bit). This is done using RW register command (shifting). After this two SET bit commands are sent to set LdRow to 1 and to 0.

3 Bits for the RAM cell i (i can be 0, 1, 2 and 3) of the current pixel row are shifted into TDAC shift register. Two SET bit commands are performed to activate and deactivate LdTDAC. (LdTDAC can be also set high before shifting of TDAC to save time.)

4 SET bit is used to activate and deactivate WriteRAMi. Since WrRAMEn is stored in the same command register (with the address 1), the bitwise OR function of WriteRAMi and WrRAMEn patterns should be stored.

Go back to 3 until all 4 RAM bits are written.

Go back to 2 until all rows are written.

Read back from the pixel RAM cells

1 Using SET bit command, set WrRAMEn to zero. This puts the RAM write drivers into high impedance state and enables reading of RAM state.

2 Pixel row is selected by writing 1 at the correct place in the row shift register (enrow bit). This is done using RW register command (shifting). After this two SET bit commands are sent to set LdRow to 1 and to 0.

3 Activate and deactivate precharge (bit SC_PCHB_int) by using two SET bit commands. Notice that PCHB is active high.

4. Use SET bit to activate WriteRAMi

5. Perform the entire read back sequence described above

6. Use SET bit to deactivate WriteRAMi

Go to 2 until all rows are read

Trigger command

Trigger command is used to produce trigger and to transmit trigger tag. The command consist of a trigger command header (8 bits), which describes one of 15 trigger command patterns, and a single data code for trigger tag (8 bits). Notice that the trigger command has 16 bits in total and that the transmission of one command takes 100ns or 4 bunch crossings. For this reason, it is necessary to encode one of 15 possible trigger patterns. Five bits of trigger tag can be calculated from data code and additional two bits are calculated from the pattern.

To summarize, the following words are sent for one command (shifting LSB first):

CMDword = TRIGGER_01, TAG

Example of the command sequence:

CMDword <= 16'b0101_0110_0110_1100//Trigger pattern TTTT, tag 1

The trigger patterns are sent LSB first.

The trigger patterns correspond to binary representation of the trigger patter index:

Pattern 1 4'b0001;

Pattern 2 4'b0010;

Etc.

Data format

As mentioned, the readout state machine sends the data to the Aurora transmitter. As explained above the input words to Aurora consist of 64-bit data word and 8-bit size word.

The 64-bit data words consist of two 32-bit half-words.

The half-words are produced by the readout state machine and grouped into events which have always even number of 32-bit words. There are 5 different data half-words: Hit word (has two formats), end of event word (EoE-word), beginning of event (actually beginning of data line) word (BoE), spacing word, and no data word. The no data word is used only after asynchronous reset. One empty event contains an EoE word and a spacing word. One event with hits starts with beginning of row word, continues with, one or several hit words, until end of columns are empty. After this, new loading of end of columns are done and either, if there are no hits EoE word is sent, or if there are hits the sequence goes back to first step (BoE-word). If the total number of half-words is odd, a spacing word is attached. First half-word is sent as the LSB part of the 64-bit Aurora data word, the second half-word as MSB part of the 64-bit Aurora data word, and so on.

The half-word formats are given here

Hit = {8'hDA, ColAdd[7:0], TS2[6:0], RowAdd[8:0]}

Hit2 = {4'hC, 1'h0,RowAdd [8:0], ColAdd[7:0], TS[9:0]}

EoE = {8'hEE, 1'h0,TriggerTag[6:0], 5'h0, fifo_full, L1TS_toDet[9:0]}//L1TS_toDet is the time stamp sent from CU to the trigger buffers and it should match with the time stamps of the hits

BoE = {16'hABCD, BinCounter[15:0]}// Binary counter is the counter used to produce current time stamp (last ten bits). The time stamps are Grey coded and calculated from the binary counter.

Let us call Hit, EoE and BoE half-words the data half-word. We have additionally two special half-words:

NoData = {8'hBC, 8'hBC, 8'hBC, 8'hBC}//used only after reset

SpaceWord = {16'hDDDD, 16'hDDDD}

All half- words have unique headers (4 or 8 MS - bits) that can be used to recognize them.

Hit word 1 is used in normal readout (bit in control register tsformat = 0), hit word 2 is used when bit tsformat = 1.

Hit word 1 word contains header, pixel address and TS2 time stamp (used to calculate ToT). The leading edge time stamp (TS) is not sent in hit word since (in normal readout) it is common for all hits within one event and is transmitted within EoE word. Notice that the time stamp in EoE word is the time stamp that is sent from CU to trigger buffers and used in content addressing block to select hits. It must match with the time stamps of the hits that are also readout.

Hit word 2 contains no TS2 but instead the leading edge time stamp TS that is readout with hits. It is useful during unsorted readout (bits forcedread and unsorted of configuration register should be 1). In the case of unsorted readout, triggered hits will be readout randomly, they are not grouped according their time stamps. EoE word contains in normal readout the time stamp of the triggered event (generated by the chip) and the trigger tag received in the trigger command (with extra two bits calculated from the pattern). In the case of autotriggered readout, the trigger tag is 7'h0B. EoE words can be also sent during unsorted readout, but hits between BoE and EoE words can have different time stamps. The intended purpose of unsorted readout is to clean up the trigger buffers of possible old "junk"-hits and not actual readout during operation. During clean-up sequence, triggers should not be issued.

As mentioned, to every 64-bit data word, a byte that indicate size of the word is attached (size word). The size word value is always 8, except for the data word that contains a 32-bit space half-word. In this case the size is 7. The data words plus size word are copied into Aurora FIFO and from there they are taken by Aurora transmitter.

The first part of the Aurora transmitter is a priority multiplexer. This priority multiplexer decides whether Aurora sends the data words or idle words, separation words, user defined words or control words (clock compensation-, channel bonding words etc.). Sending of the control words is not implemented. The priority multiplexer also adds two extra bytes to the 64 bit data word. There are several possibilities:

Sending of Data:

When 64-bit words are sent that have no space word (size word = 8), the multiplexer out word is:

MuxOut = 2'b01, Data2[31:0], Data1[31:0]

Where Data1 is the first data half-word (Hit, EoE, BoE) and Data2 the second data half-word.

When 64-bit words are sent that contain one space word (size word = 7), multiplexer out word is:

MuxOut = 2'b10, SEP7_BLOCK, SpaceWord[23:0], Data[31:0]

Where SEP7_BLOCK is 8'hE1, SpaceWord[23:0] is 24'hDDD and Data[31:0] is a data half-word (Hit, EoE, BoE).

Sending of user defined words

Periodically, or when command SendConfig is generated, the priority multiplexer sends the user defined word. This user word can be either the K-word defined in the configuration register, or it can be the state of the MonitorConfig register. The period of sending and the number or repetitions can be programed.

In the case of user defined word sending the output of the multiplexer is:

MuxOut = 2'b10, UserWord

where:

Possibility 1 (bit select_config = 0):

UserWord = {56'hFFFFFF, monitor_userk_in[7:0]}

monitor_userk_in[7:0] are the bits stored in the configuration register.

Or possibility 2 (bit select_config = 1):

UserWord = MonitorConfig

Sending of idle words

In the time slots, when no data or user-defined words are sent, multiplexer output produces idle word:

MuxOut = {2'b10, IDLE_BLOCK, 8'h10, 48'hADA};

Where IDLE_BLOCK is 8’h78.

After multiplexing, the first two bits (additional bits) of the priority multiplexer separated from the main 64 bits and go different processing.

The main 64 bits are scrambled using combinatorial logic that corresponds a linear shift register according to the following formula:

For (i=0; i<64; i++)

Shift[63:0] <= Input[63-i] exor Shift[39] exor [58]

The two additional bits are then again combined with scrambled main bits Shift[63:0]. All together the 66 bits are serialized with MSB first and send out of the chip.

Data receiver (should be implemented on FPGA)

The idea for the (customized) data receiver can be the following:

Imagine we have done a correct de-serializing.

If we look two most significant bits (66th and 65th bits Bit65 and Bit64) for different 66-bit data words they are either 01 or 10. Remember that these are the extra bits that were not scrambled. The exor function of these bits is always 1. If we look any other two neighbour bits, for different data words, they will be “quasi-random” since at least one passed scrambler. As consequence of this, exor function of these bits will fluctuate.

The most important part of the receiver is the phase detector. We use for this purpose a circular phase counter that counts from 0 to 65 and back to 0. Imagine that the phase of the phase counter matches the data, i.e. when the phase counter = 0, Bit65 is received and when phase counter = 1, Bit64 is received. If we calculate the exor function of the incoming bits for phase counter = 0 and 1, we will get always 1 as result. We can use two additional counters. One counter (number of cycles counter) will count the number of data cycles (increments when phase counter = 1) and the other counter (exor counter) will be incremented only when exor = 1. If the phase is correct, both counters will count the same number. We could wait until some large number of cycles (e.g. 100) and check if the exor counter = 100. This condition confirms that we have correct phase. If the phase is wrong, the incoming bits for phase counter = 0 and 1 are not the stabile bits (not the bits Bit65 and Bit64) and exor output will fluctuate. After 100 cycles, we will certainly have smaller number in exor-counter. This condition indicates wrong phase. We can use the condition to delay the phase counter by one clock cycle and then start the evaluation of the phase again. The synchronization is finished when the phase is achieved when number of cycles and exor counter count equally.

The code of the receiver is given here:

```
always @(posedge clk_400p or posedge reset_top)
   begin
       if (reset_top) begin
                         xorBit <= 1'b1;
                         poly <= 58'h155_5555_5555_5555;
                         shiftcntScr <= 0;
                         lockfound <= 0;
                         parregScr <= 0;
                         shiftregScr <= 30;
                         match <= 0;
                         count <= 0;
                         previous <= 0;
                         hold <= 0;
                         end
                         else begin
                         if (shiftcntScr == 0) previous <= bit_d_out;//SIM
                         if (shiftcntScr == 1) begin
                                 if((((bit_d_out == 0) && (previous == 1)) //// ((bit_d_out == 1) &&
(previous == 0)) )&& (lockfound == 0)) match <= match + 1;//SIM
                                 if(lockfound == 0) count <= count + 1;
                                 if((match == 100)&&(count == 100)) lockfound <= 1;
                                 if((match < 100)&&(count == 100)) begin
                                         match <= 0;
                                         count <= 0;
```

```
                                        hold <= 1;
                              end
                    end
                    if (shiftcntScr >= 2)  begin
                              hold <= 0;
                    end
          if (shiftcntScr >= 2) begin
                              xorBit <= bit_d_out ^ (poly[57] ^ poly[38]);
                              poly <= {poly[56:0],bit_d_out};
                              shiftregScr <= {shiftregScr[62:0],xorBit};
                    end
                    if( shiftcntScr == 8'd3 )//
                    begin
                              parregScr <= shiftregScr[63:0];
                    end
               end  //not reset
end // always
```

MonitorConfig register

The MonitorConfig register looks as follows:

monitor_Config = {4'hF, ShiftRegDAC, ShiftRegConfig, ShiftRegVDAC, ShiftRegTDAC, ShiftRegRow, ShiftRegColumn}

Register parts ShiftReg* will be filled, by shifting, with the 10 bits of the DAC register. The shifting scheme is:

ShiftReg*(9:0) << DACRegister*(n:0) << SIN

SPI register

SPI register SPI(87:0) is accessed through SPI interface. The signals are SCK (clock), MOSI (serial in), MISO (serial out) and CSB (chip select B). The length of the SPI register is 88 bits. After CSB goes from high (inactive) to low (active), at the first SCK rising edge, either the data output provided by readout state machine (64-bit data word), or the monitor_Config register is loaded into 64 MSB bits of the SPI register

SPI(87:24) = Data(63:0) or monitor_Config(63:0)

Data word will be loaded into SPI register only if the mode SPIRo is selected (by setting the bit in the configuration register) and if data word is available. The readout state machine will be then suspended until the SPI register is readout. Otherwise the content of monitor_Config will be loaded. The remaining 88 SCK signals will shift out the content of the SPI register through MISO pad. The shifting out is from MSB. At the same time the bits from pad MOSI will be shifted into the SPI register from LSB.

MISO <- SPI(87:0) <- MOSI

The 24 LSB bits of the SPI register SPI(23:0) are used for chip controlling and configuring via SPI.

The purpose of these bits is the following:

MOSI ->

SPI[0] SIn

SPI[1] Ck1

2 Ck2

3 Rb

4 LdDAC

5 LdConfig

6 LdVDAC

7 LdTDAC

8 LdRow

9 LdColumn

10 WriteRAM0

11 WriteRAM1

12 WriteRAM2

13 WriteRAM3

14 WrRAMEn

15 ResetTS

16 Injection

17 SPCH

Bits 18 to 23 are not used

Bits 24 to 87 are used for data as explained above.

➔ MISO

Configuration bits

Following table describes the bits of the configuration register. The position of serial in is: sin -> bit0 -> … -> bit 97 -> serialout.

Notice important thing: some signals are connected to inverted output of the register cell. These cases are labelled with “b”. In this case, writing of 0 in the register cell will set the signal 1.

The default state can be achieved by shifting all zeros.

Star symbol * means that the signal is used (also) for untriggered CU.

SIN ->

0 unsortedin // used in trigger buffers, every triggered hit is activated for readout

1 resetin // used in trigger buffers, reset

2 triggeredroen // used in EoC1, must be 1 in the case of triggered readout

3 not used

4 resetanalogB // used for power on reset of analog registers

5 extinjen // enable of external injection

6 not used

7 selslow // clock divider after oscillator, used in the case of slow ref ck

8 usespiro // slow readout through SPI register

9 invert // used to invert the clock for the last data mux

10 triggerwidthautot 3 // with of the trigger signal for auto-trigger; default 2; bit 3

1 triggerwidthautot 2 // bit 2

2b triggerwidthautot 1 // bit 1

3 triggerwidthautot 0 // bit 0

4 triggerdelayautot 9 //trigger delay for auto-trigger; default 61; bit 9

5 triggerdelayautot 8 // bit 8

6 triggerdelayautot 7 // bit 7

7 triggerdelayautot 6 // bit 6

8b triggerdelayautot 5 // bit 5

9b triggerdelayautot 4 // bit 4

20b triggerdelayautot 3 // bit 3

1b triggerdelayautot 2 // bit 2

2 triggerdelayautot 1 // bit 1

3b triggerdelayautot 0 // bit 0

4 inttriggersource 1 //trigger source: defines how the trigger is done: 0 and 3: from trigger command, 1: using auto trigger generator that is initiated by hit bus and 2: using auto trigger generator that is initiated by injection; bit 1

5 inttriggersource 0 // bit 0

6 exttrigen // enables external trigger

7 injectphase 1 // injection phase in 160MHz clock periods; bit 1

8 injectphase 0 // bit 0

9 bxphase 1 //bunch crossing clock phase in 160Mhz periods; bit 1

30 bxphase 0 //bit 0

1 monitoruserk 7 //user-K-word; bit 7

2 monitoruserk 6 //bit 6

3 monitoruserk 5 //bit 5

4 monitoruserk 4 *sendcounter //user-K-word bit 4 and send counter for untriggered RO

5 monitoruserk 3 *resetckdivend 3 //user-K-word bit 3 clock division (bit 3) for reset state in untriggered RO

6 monitoruserk 2 *resetckdivend 2

7 monitoruserk 1 *resetckdivend 1

8 monitoruserk 0 *resetckdivend 0

9 waittosend 1 //the 10th bit of the period for user-K-word sending; default period 256

40b waittosend 0 //the 9th bit of the period for user-K-word sending; default period 256

1 blocktosend 1 //number of repetitions in user-K-word sending; default 1; bit 1

2b blocktosend 0 //number of repetitions in user-K-word sending; bit 0

3 nu(was inj speed)

4 nu(was inj speed)

5 nu(was inj speed)

6 nu(was inj speed)

7b maxcycend 7 * //number of hits sent from EoCs per cycle; default 255 (*used also for u.ro); bit 7

8b maxcycend 6 * //bit 6

9b maxcycend 5 * //bit 5

50b maxcycend 4 * //bit 4

1b maxcycend 3 * //bit 3

2b maxcycend 2 * //bit 2

3b maxcycend 1 * //bit 1

4b maxcycend 0 * //bit 0

5 slowdowndpixelend 3 //duration of the load EoC1 signal (hit-buffers); default 3; bit 3

6 slowdowndpixelend 2 //bit 2

7b slowdowndpixelend 1 //bit 1

8b slowdowndpixelend 0 //bit 0

```
9 slowdownend 3 * //duration of the load EoC2 signal (trigger buffers); default 3; bit 3
60 slowdownend 2 * //bit 2
1b slowdownend 1 * //bit 1
2b slowdownend 0 * //bit 0
3 timerendpixel 3 //clock divider for hit-buffer SM; default 0; bit 3
4 timerendpixel 2 //bit 2
5 timerendpixel 1 //bit 1
6 timerendpixel 0 //bit 0
7 timerend 3 * //clock divider for main SM; default 0; bit 3
8 timerend 2 * //bit 2
9 timerend 1 * //bit 1
70 timerend 0 * //bit 0
1 ckdivend2 5 * //clock divider for the ToT time stamp; default 0; bit 5
2 ckdivend2 4 * //bit 4
3 ckdivend2 3 * //bit 3
4 ckdivend2 2 * //bit 2
5 ckdivend2 1 * //bit 1
6 ckdivend2 0 * //bit 0
7 ckdivend 5 * //clock divider for the main time stamp; default 0; bit 5
8 ckdivend 4 * //bit 4
9 ckdivend 3 * //bit 3
80 ckdivend 2 * //bit 2
1 ckdivend 1 * //bit 1
2 ckdivend 0 * //bit 0
3 tsformat //selects hit word format with time stamp
4 forcedread //should be one for unsorted readout
5 trigdelay 9 //trigger delay; default 64; bit 9
6 trigdelay 8 //bit 8
7 trigdelay 7 //bit 7
8b trigdelay 6 //bit 6
9 trigdelay 5 //bit 5
90 trigdelay 4 //bit 4
```

1 trigdelay 3 //bit 3

2 trigdelay 2 //bit 2

3 trigdelay 1 //bit 1

4 trigdelay 0 //bit 0

5 *roreset //readout reset for untriggered CU

6 *serializereset //serializer reset for untriggered CU

7 *aurreset //Aurora reset for untriggered CU

- ➔ SOUT

This register uses PadResetDigital as reset

VDAC

VDAC registers use as reset resetanalogregb = qcon4 AND padreseranalogb. All signals are active low, AND function has the purpose of OR.

There are 4 VDACs. Every of them is configured with an 8 bit shift register. The position of serial in is: sin -> bit0 -> … -> bit 7 -> serialout. The LSB bit is also LSB of the DAC.

The table gives the explanation of the DACs

SIN ->

DAC0 Th Initial value 1.8V

DAC2 BL Initial value 0V

DAC3 puls Initial value 1.125V

DAC4 inject voltage initial value 0

- ➔ SOUT

DAC0 is closest to serial in.

DAC register

The DAC register is used to control the bias block.

It uses resetanalogregb = qcon4 and padreseranalogb as reset. All signals are active low, AND function has the purpose of OR.

Every DAC is controlled by 6 bits. The position of serial in is: sin -> bit0 -> … -> bit 5 -> serialout. However, bit 0 is used as MSB of the DAC and bit 5 as LSB. At the beginning of DAC register block, we have 6 configuration bits.

SIN - >

q00 (=0)

q01 (=0)

qon0 0: resistive reference; 1 band-gap reference (default 0)

qon1 1 on code (default 1)

qon2 0 on code (default 0)

qon3 1 on code (default 1)

dac0 blres (default 8)

dac1 ithres (not used) (default 8)

dac2 vn (default 12)

dac3 vnfb (default 8)

dac4 vnfoll (default 8)

dac5 vnregc (nu) (default 8)

dac6 vndel/vpdel (default 8)

dac7 vpcomp (default phb) (default 8)

dac8 vpdac (nu) (default 8)

dac9 vn (default 8)

dac10 vblresdig (nu) (default 8)

dac11 vnbias (default 8)

dac12 vpload (default 8)

dac13 vnout (nu) (default 8)

digdac0 vpvco (default 16)

digdac1 vnvco (default 16)

digdac2 vpdclmux (default 32)

digdac3 vndclmux (default 32)

digdac4 vpdeldcl (default 32)

digdac5 vndeldcl (default 32)

digdac6 vpdelpreemp (default 32)

digdac7 vndelpreemp (default 32)

digdac8 vpdcl (default 32)

digdac9 vndcl (default 32)

digdac10 vnlvds (default 16)

digdac11 vnlvdsdel (default 0)

digdac12 vppump (default 16)

dac0 vpregcasc (nu) (default 16)

dac1 vprampdig/edge2 (nu) (default 16)

dac2 vncomp (default 12)

dac3 vpfoll (default 16)

dac4 vndac (default 0)

dac5 vpbiasrec (default 8)

dac6 vnbiasrec (default 8)

-> SOUT

Pads

Here we give the pad list. The pad pitch is 100µm.

Number	Name	Protection	Description
1	sub chip	no	
Vdd! SLDO regulator, control part, used for vdd! And gnd!			
2	B0	y (vdd!)	msb2 tune shunt
3	B1	y	1 tune shunt
4	B2	y	0 tune shunt
	B3	y	msb2 tune linear reg
	B4	y	1 tune linear reg
	B5	y	0 tune linear reg
	outref	y	
	gatenmos	y	
	gatepmos	y	
Vdda! SLDO regulator, control part, used for vdda! And gnda!			
	B0	Y (vdda!)	msb2 tune shunt
	B1	y	1 tune shunt
	B2	y	0 tune shunt
	B3	y	msb2 tune linear reg
	B4	y	1 tune linear reg
	B5	y	0 tune linear reg
	outref	y	
	gatenmos	y	
	gatepmos	y	
	regresetoutb	n	Regulator safety reset, use for resetanalogb
Move by 1355µm			
	Th	Y (vdda!)	Threshold (decoupling)
	Bl	y	Base line (decoupling)
	vdda!	n (used to protect)	1.8
	vdda!	n (used to protect)	1.8
Vdda! SLDO regulator – power transistors 1			
	vdda1	n	
	vdda1	n	
	vina1	n	
	vina1	n	

	Gnda!	n (used to protect)	0
	Gnda!	n (used to protect)	0
	Vssa!	n	1.2
	Vssa!	n	1.2
	NU	n	
Voltage regulator for vssa!			
	plusreg1	n	
	plusreg1	n	
Voltage regulator for minus			
	minusreg	n	
	minus	y	Connect to regulator
	por	n	Output of the power on reset generator, active low, connect to resetdigitalb
	gnd!	n	0
	vdd!	n	1.8
Vdd! SLDO regulator – power transistors 1			
	vdd1	n	
	vin1	n	
	minuspd	y	Ground for pull down 0V
	anainjection	y	Analog injection pulse (monitor)
	takefast	y	0: clk160/2 to phase detector, 1: clk160 to phase detector
	resetanalogb	y	resets analog shift regsiters, active low, connect to regulator safety reset
	hbout	y	Hit bus, add 1k pull down
	anaout	y	Analog out, add 1k pull down
	resetdigitalb	y	resets command processor, connect to power on reset generator
	id0	y	Chip id
	id1	y	Chip id
	id2	y	Chip id
	id3	y	Chip id
	untriggeredroen	y	Enables untriggered readout
	bypasscmd	y	Bypass command processor (external signals used)
	usespi	y	Enables use of SPI bits instead of command processor
	resetn	y	Reset for triggered readout and for config register
	invert	y	inverts the command clock before the phase detector
	interface speed 3	y	interface speed 3, slower shifting into regsiter
	gnd!	n (used to protect)	0
	vdd!	n (used to protect)	1.8

Move by 5000µm.

	vdd!	n (used to protect)	1.8
	gnd!	n (used to protect)	0
	nottovco	y	put 1nf in series with 1k
	tovco	y	put 1nf in series with 1k
	vhigh	y	1.8
	vlow	y	0
	dataoutn	y	Data out, needs 100ohm pullup
	dataoutp	y	Data out, needs 100ohm pullup
	ckrefn	y	Reference clock (needs Rterm)
	ckrefp	y	Reference clock (needs Rterm)
	cmdn	y	Command in (needs Rterm)
	cmdp	y	Command in (needs Rterm)
	exttriggern	y	External trigger (needs Rterm)
	exttriggerp	y	External trigger (needs Rterm)
	extinjection	y	External injection
	syncresetn	y	Sync reset for untriggered RO, TSreset external (needs Rterm)
	syncresetp	y	Sync reset for untriggered RO, TSreset external (needs Rterm)
	ckextn	y	External clock (needs Rterm)
	ckextp	y	External clock (needs Rterm)
Bypass command inputs			
	sin	y	External serial in
	ck1	y	External clock 1
	ck2	y	External clock 2
	lddac	y	
	ldconfig	y	
	ldvdac	y	
	ldtdac	y	
	ldrow	y	
	ldcolumn	y	
	wrram0	y	
	wrram1	y	
	wrram2	y	
	wrram3	y	External Wrram3
	encdr	y	enables Hogge's phase detector
SPI interface			
	csb	y	Chip select
	sck	y	Clock
	mosi	y	Serial in
	miso	y	Serial out
	ckrefoutn	y	Monitor for command clock, needs 100ohm pullup
	ckrefoutp	y	Monitor for command clock, needs 100ohm pullup
	enpll	y	enables phase detectors
	selext	y	selects external ck
	gate	y (hv prot)	2.0V, bias

	vdda!	n (used to protect)	
	vdda!	n (used to protect)	
SLDO for vdda!, power transistors 2			
	vdda2	n	
	vdda2	n	
	vina2	n	
	vina2	n	
	gnda!	n (used to protect)	
	gnda!	n (used to protect)	
	vssa!	n	
	vssa!	n	
Voltage regulator for vssa!			
	plusreg2	n	
	minus	y	Connect to voltage regulator
	gnd!	n	
	vdd!	n	
SLDO for vdd!, power transistors 2			
	vdd2	n	
	vin2	n	
	minuspd	y	Ground for pull down, 0V
	alwaysenb	y	Discards RAM disable bit (switch)
	vpbias	y	Add Decoupling to vdda!
Bias voltages (don’t need to be bonded)			
	vnbiasrec	No	
	vpbiasrec	n	
	vndac	n	
	vcasc	n	
	vncomp	n	
	vpload	n	
	vnbias	n	
	vpdel	n	
	vndel	n	
	vnfoll	n	
	vnfb	n	
	vnamp	n	
	vblr	n	
	substrate	n	
	subpixel	n	
	subchip	n	

Serial powering

Atlaspix3 contains two shunt/low dropout regulators (SLDOs). One can be used to produce vdda from vina (analogue regulator) and the other (digital regulator) to produce vddd from vind. Schematic of the regulator is shown in Figure 7. Ground line (gnd) of the analogue regulator is shorted with gnda! and gnd of the digital regulator is shorted with gnd!.

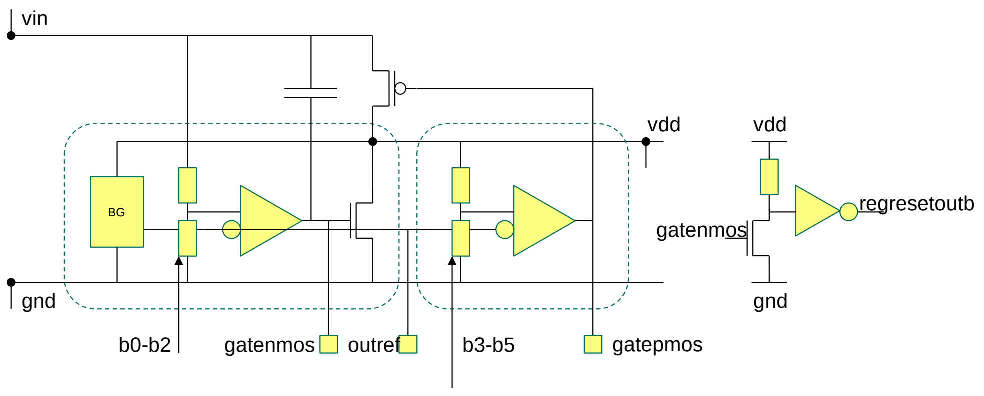


Figure7: Shunt and linear regulator of Atlaspix3

As written in the pad list, for each regulator there is a set of signals B0 – B5. Signals B0 (msb), B1, B2 (lsb) are used to tune the threshold of the shunt regulator (nominal value 2.1V). Signals B3 (msb), B4, B5 (lsb) are used to tune vdd (nominal value 1.8V). Bi signals of a regulator should be connected either to its vdd or to its gnd.

Signals gatenmos, outref, gatepmos are for monitoring.

Signal regresetoutb can be used as power on reset. This signal is generated when no current is flowing through shunt transistor.