Advanced Astir2 Core Control

October 2016

Version 1.3001

Astrohn Technology Ltd.
Naugarduko str. 102, LT-03160 Vilnius, Lithuania
Phone +370 5 2034336
Fax +370 5 2034683
E-Mail info@astrohn.lt

Table of Content

Chapter 1. Core Connector

FST and SPI interfaces are available on the 70-pin Hirose connector (model: DF17(4.0)-70DP-0.5V(57)) on the bare core. The pinout of the connector is given in Table 1 with SPI interface pins in **green** and FST interface pins in **blue**. Pin SPI/FST_READY is used by both interfaces.

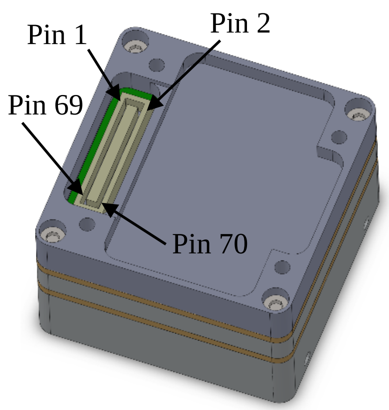


Table 1. Pinout of the bare core's 70-pin connector

Pin	Description	Dir.	Pin	Description	Dir.	Pin	Description	Dir.
1	PAL - GND		**25**	**FST data bit 5**	**Bi**	49	GND	
2	PAL		26	V. sync		50	GND	
3	Reserved		27	H. sync		51	Reserved	
4	Reserved		**28**	**FST data bit 4**	**Bi**	52	Reserved	
5	Reserved		29	GND		53	Reserved	
6	Reserved		30	GND		**54**	**FST EN**	**In**
7	Reserved		31	BT656 data bit 0		55	Reserved	
8	Reserved		32	BT656 data bit 4		**56**	**SPI_SCK**	**In**
9	GND		33	BT656 data bit 1		57	Reserved	
10	GND		34	BT656 data bit 5		**58**	**SPI_SDO**	**Out**
11	Reserved		35	BT656 data bit 2		59	GND	
12	Reserved		36	BT656 data bit 6		60	GND	
13	Reserved		37	BT656 data bit 3		61	Reserved	
14	Reserved		38	BT656 data bit 7		**62**	**SPI/FST_READY**	**Out**
15	Reserved		39	GND		**63**	**FST RD/nWR**	**In**
16	Reserved		40	GND		**64**	**SPI_SDI**	**In**
17	**FST data bit 7**	**Bi**	41	BT656 clock		65	GND	
18	UART (RS232) TX		**42**	**FST data bit 3**	**Bi**	66	Reserved	
19	GND		43	Reserved		67	Power supply - GND	
20	GND		**44**	**FST data bit 2**	**Bi**	68	Power supply	
21	Reserved		45	Reserved		69	Power supply - GND	
22	UART (RS232) RX		**46**	**FST data bit 1**	**Bi**	70	Power supply	
23	**FST data bit 6**	**Bi**	47	Reserved				
24	Reserved		**48**	**FST data bit 0**	**Bi**			

Chapter 2. Hardware layer

Data packets can be transferred to the FPGA using one of two hardware interfaces:

- SPI
- FST

Additionally to the two hardware interfaces listed above core supports communication with the master device over UART interface. However, UART interface only gives access to basic core's parameters adjustment (refer to the Iridium Infrared Core User Guide for more information about UART interface).

The following two sections describe SPI and FST interfaces in more details.

SPI

FPGA acts as SPI slave device. Four signals are used for communication: SPI_SCK, SPI_SDI, SPI_SDO and SPI/FST_READY. All these signals have voltage level of 3.3V.

SPI_SDI and SPI_SCK are inputs to the core. SPI_SCK acts as SPI clock while SPI_SDI – as data input to the core. Recommended SPI_SCK frequency is 20 MHz. Idle clock state is low, data must change on high-to-low clock transition. Data is transferred in 32-bit words, most significant bit first.

After full packet of data has been transferred the core will signal that the response packet is ready by bringing SPI/FST_READY signal high. After SPI/FST_READY transition from low to high master has to provide enough clock pulses for the data to be clocked out from the core. Size of the response packet depends on the data transferred from the master to the core.

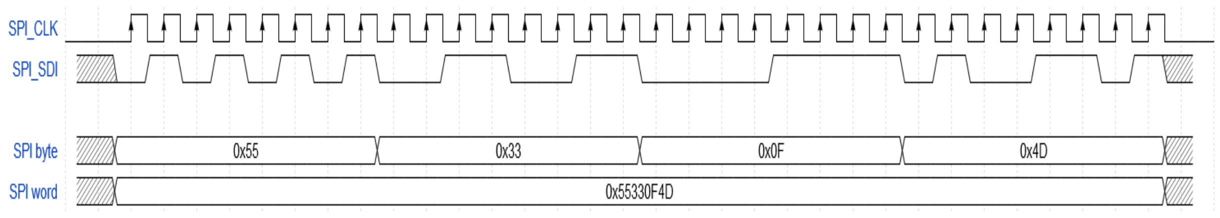


Figure 1. Timing diagram shows the transfer of a single 32-bit word 0x55330F4D from master to the core

FST

FST interface is not implemented yet.

The interface will work with the same packet based protocol as SPI, but because of parallel nature will allow faster data transfers.

Chapter 3. Packet based communication

Core communicates with the external master device using SPI or FST hardware interface by exchanging data packets. Communication is always initiated by external master device by sending a request packet. Upon successful reception of the request packet, the core responds with response packet.

Packet structure

Both request and response packets consist of two parts – packet header and packet body. Currently core supports fixed size request packets consisting of 64 byte long header and 1024 byte long body. The size of the response packet may vary – response packet header is always 64 bytes long while response packet body may be 0 or 1024 bytes long depending on the request packet header content. Figure 2 depicts request and response packet exchange between master and core.

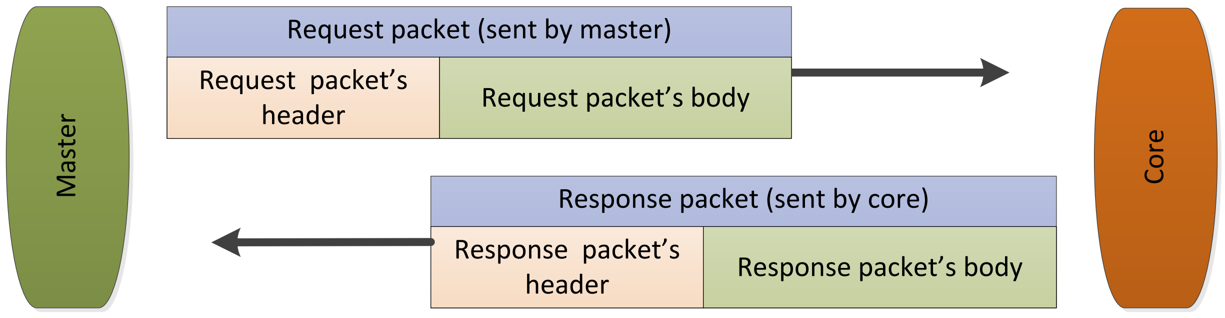


Figure 2. Request and response packet exchange between master and core

Request packet header

All request packets start with the header. Currently header size is fixed to 64 bytes. Header structure is given in Table 2. Some fields of the header are reserved for future use. All bytes belonging to the reserved data fields must be set to 0x00.

Table 2. Request packet's header structure (field offsets and field sizes are given in bytes)

Field offset	Field size	Field name	Field description
0x00	0x04	Title	Always: 0x54, 055, 0x53, 0x42 (four ASCII characters: 'TUSB')
0x04	0x04	Checksum	Contains the sum of packet's body words (data in packet body is interpreted as 16-bit words)
0x08	0x02	Command	Contains command code. Command code can have one of four valid values (given in Table 3). It determines the action the core

Field offset	Field size	Field name	Field description
			will take upon reception of data packet and type of response packet it will respond with.
0x0A	0x02	Data size	Indicates size of valid data in packet's body. Packet's body size is fixed to 1024 bytes, but the size of valid data may vary.
0x0C	0x02	Memory select	For memory writing or memory reading commands indicates which core's memory will be accessed. List of available memories is given in Table 6.
0x0E	0x04	Memory offset	Memory offset in bytes for memory writing and memory reading commands.
0x12	0x01	Reserved	Must be set to 0x00.
0x13	0x01	Reserved	Must be set to 0x00.
0x14	0x02	Window X	Window's upper left corner X coordinate (multiple of 8). The field is only used for windowed memory writing operations.
0x16	0x02	Window Y	Window's upper left corner Y coordinate (multiple of 1). The field is only used for windowed memory writing operations.
0x18	0x02	Window width	Window width in bytes (multiple of 8). The field is only used for windowed memory writing operations.
0x1A	0x02	Frame width	Frame width in bytes (multiple of 8). The field is only used for windowed memory writing operations.
0x1C	0x01	Window mode	Windowed memory write mode.
0x1D	0x03	Reserved	All bytes in this field must be set to 0x00.
0x20	0x20	Reserved	All bytes in this field must be set to 0x00.

Table 3. List of valid command codes and their meaning in request packet's header

Command code	Transaction purpose
0x00	Read core's register(s)
0x01	Read core's memory
0x02	Write core's register(s)
0x03	Write core's memory

Request packet body

Content of the request packet's body depends on the *command* field in request packet's header.

For register read commands, bytes at even addresses in request packet's body contain addresses of core's registers to be read from, while bytes at odd addresses in request packet's body are ignored. For instance, to read the content of three core's registers at addresses 0x05, 0x20 and 0xAB, request packet's body will look like this: [0x05, 0x00, 0x20, 0x00, 0xAB, 0x00, 0x00, 0x00, …]. With request packet's body size of 1024 bytes, maximum number of read register can be 512.

For register write commands, bytes at even addresses in request packet's body contain addresses of core's registers to be written to, while bytes at odd addresses in request packet's body contain data to be written. For instance, to write value 0x12 to register at address 0x34 and value 0x56 to register at address 0x78, request packet's body will look like this: [0x34, 0x12, 0x78, 0x56, 0x00, 0x00 …]. With request packet's body size of 1024 bytes, maximum number of written register can be 512.

For memory write command, request packet's body contains data to be written to one of core's memories.

For memory read command, request packet's body data is ignored.

Response packet header

Response packet body contains status code, indicating whether request packet has been successfully received and command executed. Response code is located at the first byte of response packet's header. Meaning of the status code is given in the table below.

Table 4. Meaning of different status code values.

Status code	Status code meaning
0x00	Command execution successful
0x01	Wrong request packet's header title (bytes 0x00 – 0x03 in request packet's header)
0x02	Wrong checksum in request packet's header
0x03	Unrecognized command (*command* code field in request packet's header).

Response packet body

Response packet's body has non-zero size only for read register(s) and read memory commands. Otherwise response packet contain only header (body size is zero bytes).

For read register(s) command, bytes at even addresses in the response packet body contain value of the registers which addresses where transferred to the core with request packet containing read register(s) command.

For read memory command, request packet's body contains data from the core's memory.

Chapter 4. Data packet types

Request data packets can be sent for one of the following four purposes: read core's register(s), read core's memory, write core's register(s), write core's memory which is determined by the *command* field in request packet's header. Below is a detailed description of each of these four actions.

Reading core's register(s)

Following fields are used in request packet's header when core's register reading is intended:

- *Title*
- *Checksum*
- *Command*
- *Data size*

Values in other fields are ignored, except for reserved fields which must have all bytes set to 0x00.

Command field has to be set to 0x00, while *data size* field contains number of registers to be read.

Writing core's register(s)

Following fields are used in request packet's header when core's register writing is intended:

- *Title*
- *Checksum*
- *Command*
- *Data size*

Values in other fields are ignored, except for reserved fields which must have all bytes set to 0x00.

Command field has to be set to 0x02, while *data size* field contains number of registers to be written.

Reading core's memories

Following fields are used in request packet's header when core's memory reading is intended:

- *Title*
- *Checksum*
- *Command*
- *Data size*
- *Memory select*
- *Memory offset*
- *Window mode*

Values in other fields are ignored, except for reserved fields which must have all bytes set to 0x00.

Command field has to be set to 0x01. *Data size* field indicates number of bytes to be read from the memory. Possible values for *memory select* field are given in Table 7. *Memory offset* field contains address we want memory reading to start at. Memory reading in windowed mode is not supported and *window mode* field must be set to 0x00.

Writing core's memories

Following fields are used in request packet's header when core's register reading is intended:

- *Title*
- *Checksum*
- *Command*
- *Data size*
- *Memory select*
- *Memory offset* (for some *memory select* field values, memory offset is chosen by the core automatically and value in the *memory offset* field is ignored)
- *Window mode*

Values in other fields are ignored, except for reserved fields which must have all bytes set to 0x00.

Command field has to be set to 0x03. *Data size* field indicates number of bytes to written to the memory (has to be a multiple of 8). Possible values for *memory select* field are given in Table 7. *Memory offset* field contains address we want memory writing to start at.

If data is written to flash memory (indicated as *nonvolatile flash memory*) in Table 7 at least 30ms have to pass between 1024 byte write operations.

Windowed writing to core's memories

If windowed memory writing is to be used (*window mode* field value greater than 0) the following additional fields are used:

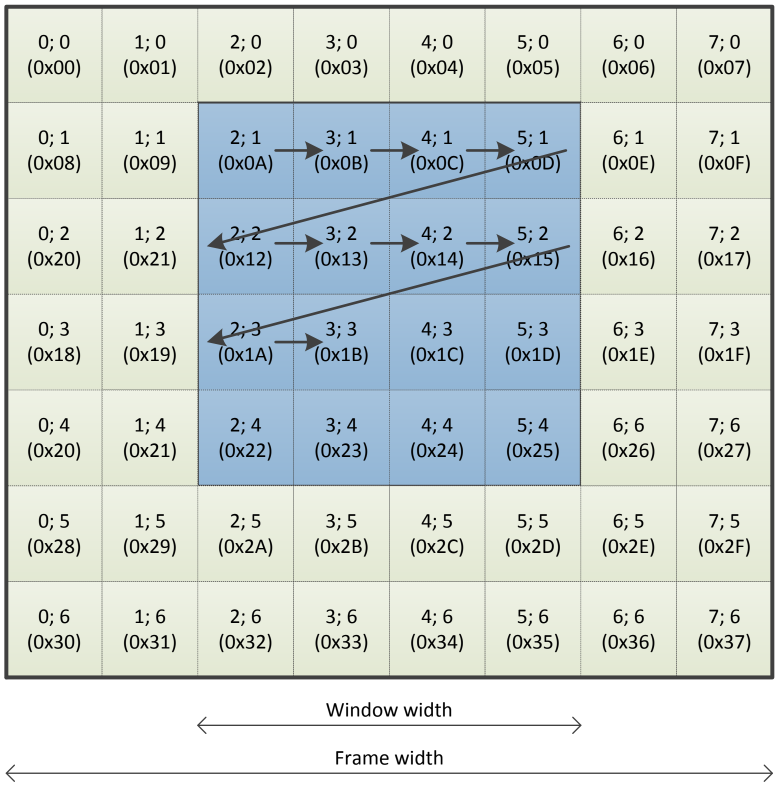
- *Window X*
- *Window Y*
- *Window width*
- *Frame width* (for some *memory select* values frame width is chosen by the core automatically and value in the *frame width* field is ignored)

If *windowed mode* field is set to 1, new virtual window will be created within the selected memory and data will be written to the memory starting at the beginning of the window. If *windowed mode* field is set to 2, data will be written further to the previously created window.

Windowed memory writing is visualized in Figure 3. Continuous region of memory starting at address 0x00 is visualized as two dimensional 8 bytes wide array (array width corresponds to *frame width* field in request packet's header). Within the two dimensional memory array virtual window is created at location with two dimensional coordinates (2;1) (in request packet's header *window X* field set to 2 and *window Y* field set to 1). Window width is set to 4 bytes (*window width* field in request packet's header). In the figure, each memory cell contains its absolute address within memory (in parenthesis) as well as its two dimensional coordinates within arbitrary chosen two dimensional memory array. Data written to the memory using windowed mode with parameters described above will be written to the memory bytes colored in blue.

Windowed memory writing is particularly useful when using graphics overlay functionality.

Important: care must be taken during memory write operations not to go over the allowed memory region limits as this will corrupt other parts of core's memory. However, unless writing to flash memory, core's operation after memory corruption can be fully restored by re-powering it.



0; 0 (0x00)	1; 0 (0x01)	2; 0 (0x02)	3; 0 (0x03)	4; 0 (0x04)	5; 0 (0x05)	6; 0 (0x06)	7; 0 (0x07)
0; 1 (0x08)	1; 1 (0x09)	2; 1 (0x0A)	3; 1 (0x0B)	4; 1 (0x0C)	5; 1 (0x0D)	6; 1 (0x0E)	7; 1 (0x0F)
0; 2 (0x20)	1; 2 (0x21)	2; 2 (0x12)	3; 2 (0x13)	4; 2 (0x14)	5; 2 (0x15)	6; 2 (0x16)	7; 2 (0x17)
0; 3 (0x18)	1; 3 (0x19)	2; 3 (0x1A)	3; 3 (0x1B)	4; 3 (0x1C)	5; 3 (0x1D)	6; 3 (0x1E)	7; 3 (0x1F)
0; 4 (0x20)	1; 4 (0x21)	2; 4 (0x22)	3; 4 (0x23)	4; 4 (0x24)	5; 4 (0x25)	6; 6 (0x26)	7; 6 (0x27)
0; 5 (0x28)	1; 5 (0x29)	2; 5 (0x2A)	3; 5 (0x2B)	4; 5 (0x2C)	5; 5 (0x2D)	6; 5 (0x2E)	7; 5 (0x2F)
0; 6 (0x30)	1; 6 (0x31)	2; 6 (0x32)	3; 6 (0x33)	4; 6 (0x34)	5; 6 (0x35)	6; 6 (0x36)	7; 6 (0x37)

Figure 3. Windowed memory write operation (see text for explanation)

Nonvolatile flash memory sector erase

Before core's nonvolatile flash memory can be written with new content, sector erase operation has to be performed (however it does not have to be performed if write operation only involves changing flash memory bits from '1' to '0' – this might be the case when new bad pixel is to be marked in bad pixel map for instance). Flash sector size is 65536 bytes.

To perform flash sector erase, the following core's registers (in the exact given order) have to be written to:

Table 5. Core's nonvolatile flash sector erase procedure

Register address	Register data	Explanation
0x25	0x03	Switch to manual flash memory control
0x27	0x00	Deselect flash chip
0x27	0x01	Select flash chip
0x26	0x06	Set write enable
0x27	0x00	Deselect flash chip
0x27	0x01	Select flash chip
0x26	0xD8	Sector erase command

Register address	Register data	Explanation
0x26	Address2	Third (most significant) byte of the flash sector address
0x26	Address1	Second byte of the flash sector address
0x26	Address0	First (least significant) byte of the flash sector address
0x27	0x00	Deselect flash chip
0x25	0x00	Switch to default flash memory control

After flash sector erase command has been issued sector is guaranteed to be erased in 3 seconds – no other operation requiring flash access can take place during this time. At least one millisecond delay has to be made between each register write operation for the core to perform serial transaction to flash memory. It means that registers have to be written to the core using separate request packets (one register per packet).

Chapter 5. Data exchange mechanisms

Data exchange between external master device and core can be done by reading/writing core's registers or by reading/writing core's memories.

Core's registers

Registers are 8 bit memory locations within the core. There can be a total of 256 different registers which can be addressed by 8 bit addresses. List of available registers is given in Table 6 along with their description and purpose.

Table 6. List of core's registers

Address	Name	R/W	Bits	Default value	Description
0x00	firmwareVersion	R	7:0	-	Contains FPGA firmware version
0x01	testReg	R/W	7:0	0x00	Register can be written and read for test purposes and has no effect on core's operation
0x45	groControl	W		0x00	Graphics overlay control register.
			0		Graphics overlay enable bit. Writing 1 will enable graphics overlay, while writing 0 will disable it
			4		Writing 1 to this bit will copy the content of hidden graphics overlay buffer to the visible one.
0x46	groClearColor	W	7:0	0xFF	Color code to be used for clearing graphics overlay hidden buffer after its' content has been transferred to the visible graphics overlay buffer
0x47	groTriggerAreaSize	W	7:0	0x20	Defines half-width of the square area located at the center of the image whose content will be used to decide whether graphics overlay pixels with color code 0xFE have to be displayed as black or white.
0x47	*gOverlayTriggerHysterisis*	W	7:0	0x80	Defines minimum difference between the numbers of bright and dark pixels within the color inversion trigger window required for the color inversion of pixels with 0xE0 color code to take place
0xA0	systemControl	W			System control register
			0		Writing 1 to this bit will force core to reinitialize (reload all settings, NUC correction tables, bad pixel map, etc. from non-volatile flash memory)
0xD6	zoomFactor	W	7:0	0x00	Defines zoom factor to be used on the infrared image. Values of 0, 2 and 4 correspond to fixed zoom factors of x1, x2 and x4. Any other value will switch to

					custom zoom mode. In this mode image panning and zoom is defined by values written to registers *rowsStartL, rowsStartH*, *rowsCountL, rowsCountH*, *columnStartL*, *columnStartH*, *columnCountL*, *columnCountH*. For the new data written to any of these four registers to become valid write operation to *zoomFactor* register must occur.
0xD8	*rowsStartL*	W	1:0	-	Lower two bits of the X coordinate of the top left corner of the cropping window.
0xD9	*rowsStartH*	W	7:0	-	Upper eight bits of the X coordinate of the top left corner of the cropping window.
0xDA	*rowsCountL*	W	1:0	-	Lower two bits of the width of the cropping window.
0xDB	*rowsCountH*	W	7:0	-	Upper eight bits of the width of the cropping window.
0xDC	*columnStartL*	W	1:0	-	Lower two bits of the Y coordinate of the top left corner of the cropping window.
0xDD	*columnStartH*	W	7:0	-	Upper eight bits of the Y coordinate of the top left corner of the cropping window.
0xDE	*columnCountL*	W	1:0	-	Lower two bits of the height of the cropping window.
0xDF	*columnCountH*	W	7:0	-	Upper eight bits of the height of the cropping window.

Core's memories

Table 7. List of core's memories

Memory select	Memory description
0x01	Reserved
0x02	Reserved
0x03	Reserved
0x04	Reserved
0x05	Graphics overlay hidden buffer
0x06	Reserved
0x07	Factory bad pixel map (nonvolatile flash memory)
0x08	User bad pixel map (nonvolatile flash memory)
0x09	Gain NUC table (nonvolatile flash memory)
0x0A	Reserved
0x0B	Frame accumulator buffer for odd frames
0x0C	Frame accumulator buffer for even frames

Chapter 6. Image scaling

Prior to video output, processed infrared sensor image is scaled to match the required video output resolution. Core works with sensors having resolution of 640x480 or 384x288 (referred in this document as *sensor resolution*). Video output resolution (referred in this document as *video resolution*) depends on selected video output mode. For PAL video output resolution is 640x576; for BT.656 video output it is 720x576. Image scaling is carried out using bilinear interpolation method.

If zoom function is used, processed sensor image is cropped to a required size before scaling operation. For example zooming 640x480 sensor image by a factor of two, is equivalent to cropping the center part (with a size of 320x240 pixels) and scaling it to video resolution.

Core supports three fixed zoom levels of x1, x2 and x4 as well as custom zoom/panning. Switching between these modes is carried out using *zoomFactor* register. In custom mode eight other registers (*rowsStartL, rowsStartH*, *rowsCountL, rowsCountH*, *columnStartL, columnStartH*, *columnCountL, columnCountH*) are used to define exact area to be cropped from the infrared image at *sensor resolution*. Cropped area will then be scaled to *video resolution*.

If graphics overlay is used it is always superimposed on the scaled image at video resolution.

Chapter 7. Gain calibration and bad pixel correction

Gain calibration

Gain calibration procedure consist of acquiring two averaged frames – one using hot black body in front of the lens and the other using cold black body.

To prepare the core for averaged reference frame acquisition core registers have to be written as indicated in Table 8.

Table 8. Core's setup for averaged reference frame acquisition procedure

Register address	Register data	Explanation
0x7f	0x00	Switch off adaptive filter
0x62	0x01	Freeze accumulator when enough frames accumulated
0x2b	0x01	Switch accumulator to 32 bit mode
0x39	numberOfFrames	Number of frames to be accumulated

Once the core is setup for frame acquisition, acquisition process is started by writing value 0x02 to register 0x38. After writing to this register acquired frames will be ready for download from the core after a period of time equal to numberOfFrames/FPS seconds (FPS – sensor frame rate. Currently it is 25Hz for 640x480 sensor and 50Hz for 384x288 sensor). Acquired frame can be read from core's memory using chip selects 0x0B and 0x0C depending on whether acquired number of frames was odd or even (see Table 7). In these reference frames single pixel is represented by 4 bytes, which means for a single frame 4x640x480=1228800 bytes have to be read from the core for the 640x480 sensor. To save master's memory, first (hot) reference frame can be fully read from the core. And second (cold) reference frame can be read in blocks with the size of for example 1024 bytes.

For each reference frame (hot and cold) read pixel values have to be divided by the accumulated number of frames (numberOfFrames).

After two reference frames have been read from the core, the following equation can be used to calculate gain value for each pixel:

$$g_i = 32768 * \frac{\overline{ref^C} - \overline{ref^H}}{{ref^C}_i - {ref^H}_i},$$

where $\overline{ref^C}$ and $\overline{ref^H}$ are average values for all pixels in cold and hot reference frames respectively, ${ref^C}_i$ and ${ref^H}_i$ are values of the ith pixel, and g_i is calculated gain value for the ith pixel to be written to the core's flash memory.

Flash memory part containing gain table can be accessed using chip select 0x09 (see Table 7). Each pixel is represented by a single 16 bit word, thus 640x480x2 bytes have to be written to the gain table for the core with 640x480 sensor. With the sector size of 65536 bytes this corresponds to 10 sectors (10th sector is only partially used, but has to be erased when new gain table is going to be written).

Before new gain values can be written to the core, previous gain table has to be erased in core's flash memory. Flash sector erase procedure is described in Nonvolatile flash memory sector erase subsection. Memory offset for gain table stored in flash is: 0x60000. It is important to use correct addresses when erasing flash sectors, otherwise core with corrupt flash may not power up or function properly.

After changes to core's flash memory are made they do not immediately affect core's operation (core operates using settings and tables loaded during initialization to RAM memory). There are two ways to load new information from flash memory – power cycling the core or performing reinitialization (writing 1 to 0xA0 register).

Normal core's operation is resumed by either reinitializing the core or writing registers given in Table 9.

Table 9. List of registers to be written to the core in order to resume it to normal operation.

Register address	Register data	Explanation
0x62	0x00	Accumulator will never freeze
0x2b	0x00	Switch accumulator to 16 bit mode
0x39	0x01	Number of frames to be accumulated
0x38	0x02	Reset frame accumulator

Bad pixel correction

Two bad pixel (BP) maps are stored in core's nonvolatile flash memory – factory BP map and user BP map. Both tables are read during power up, merged into a single table and used during core's operation to correct bad pixels. Bad pixel correction is performed by averaging up to 8 good neighboring pixels for every bad pixel. If bad pixel is surrounded by 8 other bad pixels it can no longer be corrected by simple neighbor averaging. In such cases different method is used to force it to similar color as surrounding pixels.

For 640x480 resolution sensor 640*480/8=38400 bytes are used to store single bad pixel map. Each pixel in this map is represented by one bit. Sensor pixels are ordered row by row. First (least significant) bit in the first byte in bad pixel map corresponds to the first (left) pixel in the first (top) row. Second bit corresponds to second pixel in first row and so on.

Bits in bad pixel map equal to '1' represent good pixels, while bits equal to '0' – bad pixels. In core's nonvolatile memory bits can be set from '1' to '0' without performing memory sector erase operation, which means that any pixel can be marked as bad without erasing flash memory sector. To change bad pixel into good pixel, full bad pixel map has to be read, flash sector erase operation has to be performed after which previously read (and modified as required) bad pixel map can be written back to core's flash memory.

Flash sector erase procedure is described in Nonvolatile flash memory sector erase subsection. Memory offsets are: 0x4A0000 for factory bad pixel map and 0x4B0000 for user bad pixel map. It is important to use correct addresses when erasing flash sectors, otherwise core with corrupt flash may not power up or function properly.

Two memory chip selects are used to read/write bad pixel map (see Table 7).

After changes to core’s flash memory are made they do not immediately affect core’s operation (core operates using settings and tables loaded during initialization to RAM memory). There are two ways to load new information from flash memory – power cycling the core or performing reinitialization (writing 1 to 0xA0 register).

Chapter 8. Graphics overlay

In order to write to graphics overlay buffer, memory select field value has to be set to 0x05. When graphics overlay buffer is selected, memory offset field in packet header is ignored and frame width is set automatically inside the core to appropriate value (640 for PAL video output and 720 for BT.656 video output). Data in graphics overlay buffer is superimposed on the infrared scene. Single byte in graphics overlay buffer represents single pixel.

Pixels with value 0xFF in graphics overlay buffer are transparent (infrared scene content is visible at those locations).

Pixels with value 0xFE in graphics overlay buffer change color depending on the infrared scene content at the center of the image. Size of the square window at the center of the image used to determine what color all 0xFE pixels will be displayed in, is defined by the *groTriggerAreaSize* register (each side of the square window is twice the value written to *groTriggerAreaSize* register). If number of pixels with brightness above medium within the window is greater than the number of pixels with brightness below medium plus *gOverlayTriggerHysterisis* register value, color for all 0xFE pixels is switched to black. If number of pixels with brightness above medium within the window is less than the number of pixels with brightness below medium minus *gOverlayTriggerHysterisis* register value, color for all 0xFE pixels is switched to white. If difference between number of pixels with greater brightness and number of pixels with lower brightness is within the [*–gOverlayTriggerHysterisis; +gOverlayTriggerHysterisis]* interval, color for 0xFE pixels is not changed. This assures color is not inverted too frequently when number of dark and bright pixels is about equal.

Pixel values 0xE0 – 0xFD are reserved.

The rest of graphics overlay pixel values (0x00 – 0xDF) represent graphics overlay content. For PAL video output signal 0x00 represents black overlaid pixel, while 0xDF represent white overlaid pixel. For BT.656 video output signal color palette given in Figure 4 is used.

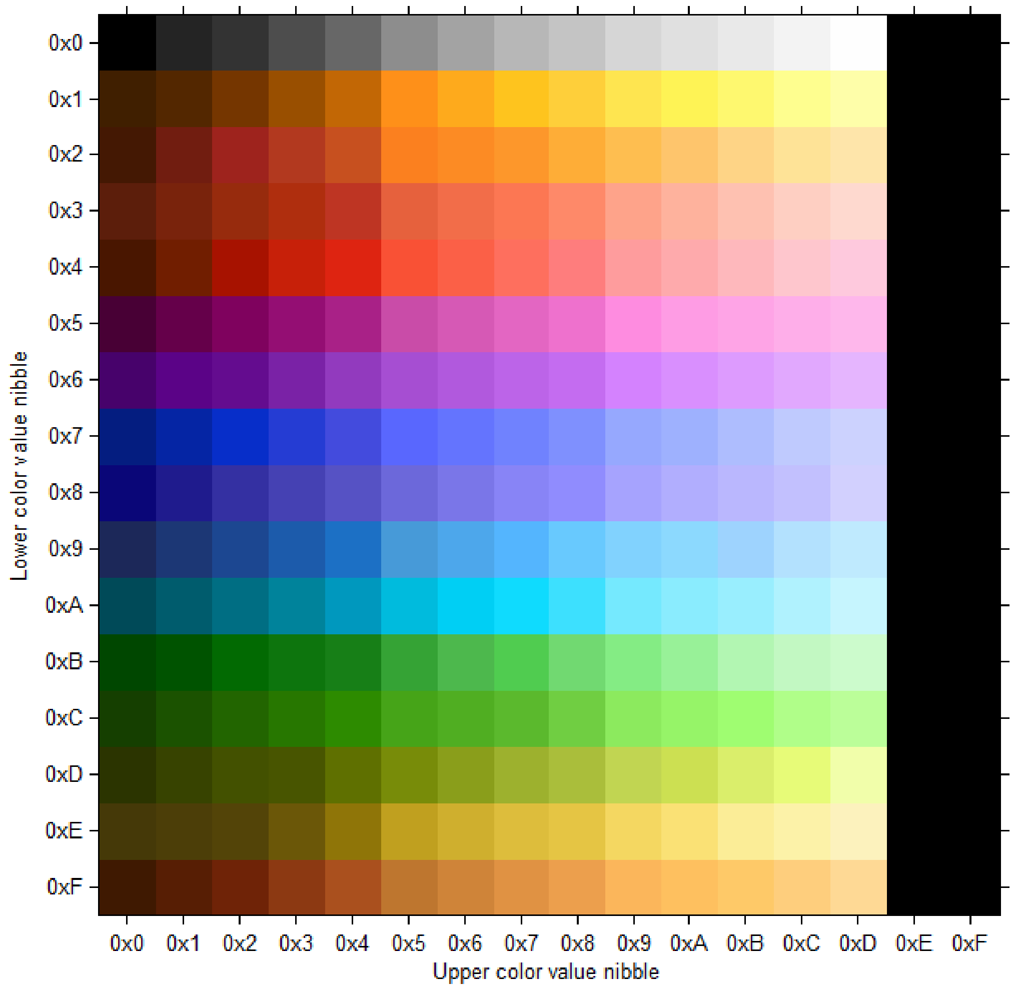


Figure 4. Graphics overlay palette used in BT.656 video output signal.

Table 10. Graphics overlay color codes

Color code(s)	Description
0x00 – 0xDF	For PAL video output these 224 color codes represent graphics overlay content with gradually increasing brightness (0x00 – completely black, 0xDF – completely white). For BT.656 video output graphics overlay content color represented by each of the 224 color codes is given in Figure 4.
0xE0 – 0xFD	Reserved
0xFE	Color code represents black or white graphics overlay content color. Exact color depends on the number of pixels in infrared scene with intensity lower than middle value and number of pixels with intensity higher than middle value within the given color trigger area (controlled by groTriggerAreaSize register).
0xFF	For all video output modes this is transparent color code (infrared scene content is seen at the locations where graphics overlay pixels have value of 0xFF)

Core supports windowed memory writing. It means that data can be written to selected memory by filling virtual window from left to right and from top to bottom. Number of windows is unlimited. Window size has to be equal or smaller than the video frame size. Smallest allowed window width is 8 pixels. Not using windowed writing (*window mode* field is set to 0 in request packet's header) is equivalent to setting window width to frame width.

Video frame size in BT.656 video output mode is 720 x 576 pixels. In PAL video output mode it is 640 x 576.

Graphics overlay data is being written to the invisible buffer. To make prepared buffer visible, 5th bit has to be set to 1 in 0x45 register (see register writing). Additionally 1st bit controls whether graphics overlay is enabled. So, to update visible graphics overlay buffer with newly written content (and to keep it enabled or enable if it hasn't been done before) write 0x11 to 0x45 register. It is not recommended to update visible graphics overlay buffer with new content faster than 50 times per second (otherwise invisible buffer may not be cleared entirely). Figure 5 shows video frame and virtual window arrangement within the core's memory.

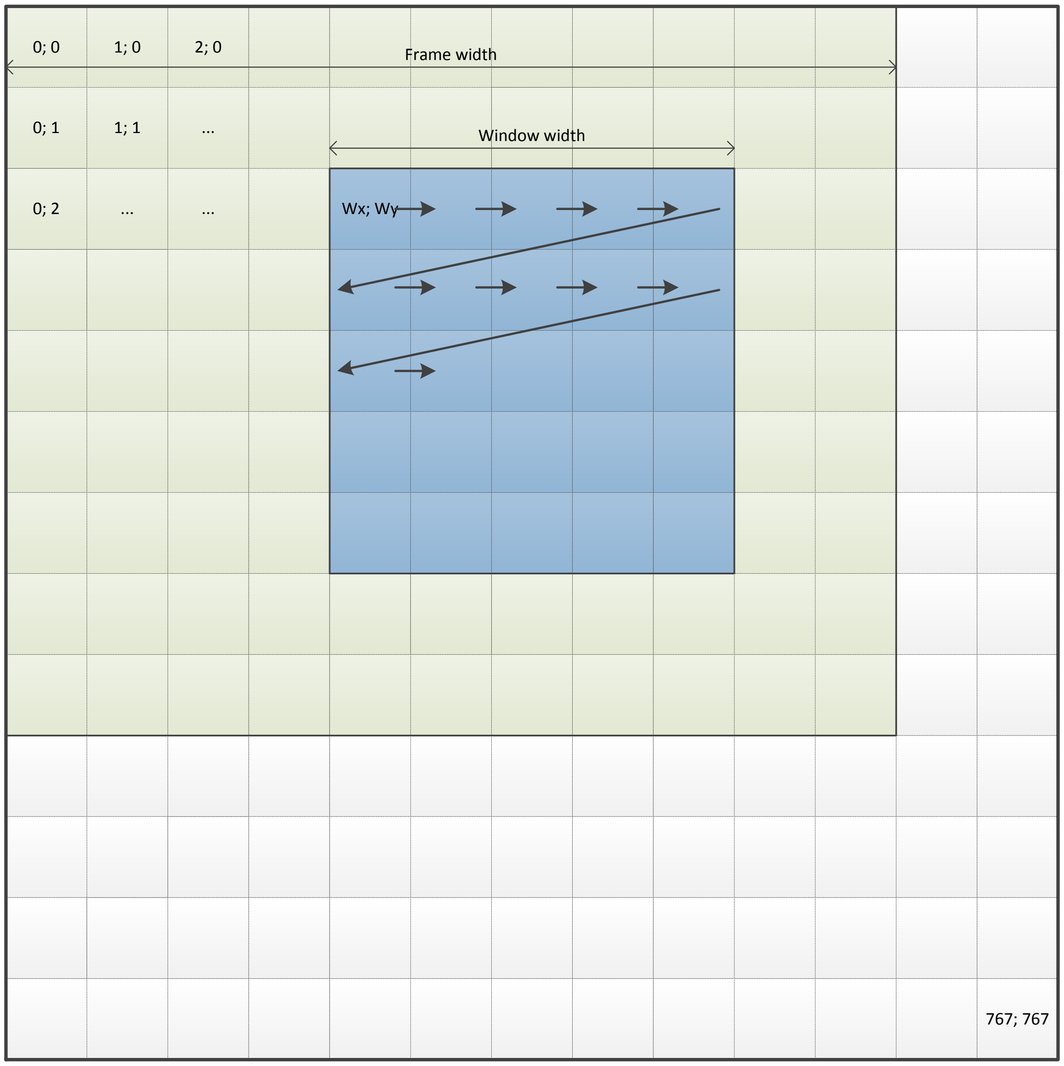


Figure 5. Video frame and virtual window (intended for windowed writing) arrangement within the core's memory.