machine vision lab.

T-COR-30 VIDEO_TRACKING_IPCORE

FPGA IP CORE FOR AUTOMATIC TRACKING OF OBJECTS IN VIDEO
(programmer manual)

IP Core version: 3.0
IP Core release date: 17.03.2017
Document version: 1.0
Document release date: 28.03.2017

RIFTEK-Russia
www.riftekvision.com

CONTENTS

DOCUMENT VERSIONS 3
IP core versions 3
DESCRIPTION 3
BASIC FEATURES AND FUNCTIONS 3
principle of operation of the ip core tracking algorithm 5
OPERATION MODES 6
Description of the tracking algorithm operation modes 6
Criteria of automatic changing of the algorithm operation modes 7
ADDRESS SPACE OF THE VIDEO_TRACKING_CORE IP CORE 8
Register CORE_CTRL_REG 9
Register TRGT_ATTR_REG 10
Register SRCH_AREA_ATTR_REG 10
IP CORE USAGE PROCEDURE 10
Importing the IP core into existing or created project 10
Project infrastructure and connection of the IP core 12
BASE PROJECT FOR TESTING THE VIDEO_TRACKING_CORE IP CORE 12
Base project hardware 12
Base project software 13

DOCUMENT VERSIONS

Table 1 – Document versions.

Version	Description
1.0	Description of FPGA IP core for automatic tracking of objects, version 3.0.

IP CORE VERSIONS

Table 2 – Versions of the software library.

Version	Description
3.0	Automatic tracking algorithm has been improved. The IP core is adapted for use in Altera FGPA.

DESCRIPTION

The FPGA VIDEO_TRACKING_CORE INTELLECTUAL PROPERTY CORE (hereinafter referred to as IP core) implements the algorithm of automatic tracking of objects in video and calculation of their motion parameters. The core ensures a stable tracking of small-sized and low-contrast objects of any type against a complex background. In case of object loss (tracking collapse), the IP core performs prediction of the tracked object trajectory up to its automatic re-capture or tracking reset if the corresponding criteria are fulfilled. The IP core is a stand-alone module easily integrable into projects based on field-programmable gate arrays (FGPA) and application-specific integrated circuits (ASIC). The core interfaces are universalized for connection to IP cores of other manufacturers.

IMPORTANT: All coordinate and translation values given in this document should be considered in the coordinate system of an image (frame) with the origin in the upper left corner.

BASIC FEATURES AND FUNCTIONS

Table 3 presents basic characteristics of the T-COR-30 IP core.

Table 3 – Basic characteristics of the IP core and tracking algorithm.

Parameter	Value and note
IP core dev. language	Verilog-2001, VHDL.
Platform supported	Xilinx Vivado 2014 и выше; Altera Quartus.
Utilization of FGPA resources after the core synthesis	Results of synthesis in Xilinx Vivado 2016.2 for 7 series ICs: LUT: 25511; FF: 36577; BRAM: 72; DSP: 116.
Number of objects tracked simultaneously	The IP core ensures automatic tracking of one object in video frames. If tracking of several objects is necessary, several IP core devices need to be used.
Core clock frequency	150 MHz. The clock speed indicated may be higher or lower depending the synthesis and implementation parameters, load of FGPA chip, its type and speed grades. The core has been tested on FPGA Zynq XC7Z020-1CLG484C at 166 MHz clock speed and chip load of 70% for default settings of synthesis and implementation.
Maximum size of tracked object	Maximum dimensions of the tracked object are limited by maximum dimensions of the tracking strobe. For large objects, tracking can be performed over a fragment (part) of the object. In this case, the position of the tracking strobe relative to the object can be changed by a command.
Minimum size of tracked object	8x8 pixels for minimum dimensions of the tracking strobe of 16x16 pixels. The algorithm is adapted for tracking small-sized low-contrast objects in the presence of occludes. It is recommended to set the tracking strobe dimensions larger than the dimensions of the tracked object image.
Maximum size of tracking strobe	128x128 pixels. Any aspect ratio of the tracking rectangle is possible within maximum and minimum allowed values and increment sizes.

Parameter	Value and note
Minimum size of tracking strobe	16x16 pixels.
Increment size of tracking strobe	16 pixels both horizontally and vertically. For example, 16x16 pixels, 32x16 pixels, 32x32 pixels, 32x48 pixels, etc.
Maximum allowed translation of tracked object per frame	The allowable object translation per frame without tracking collapse is 52 pixels in any direction (horizontal and(or) vertical).
Discreteness of object coordinates computation	No less than 1/256 pixel. The algorithm calculates the tracked object position for each frame on the video. The computed object coordinates are presented in two forms: integer (1 pixel accuracy) and fractional (fractional value in pixels multiplied by 256).
Automatic computing of tracked object translation speed	The IP core automatically calculates the object motion in video frames with a discreteness of no less than 1/256 pixel/frame. Speed calculation is performed in the course of tracking with speed estimate available within a maximum of 256 video frames from the beginning of the object's motion or change of direction. In the interval before 256 video frames have elapsed, possible calculation error exceeds 1/256 pixel/frame.
Changing of tracking algorithm parameters	The IP core makes it possible to change the algorithm parameters, including in times of automatic tracking.
Format of input video data	The IP core receives video stream in mono_8 format (1 byte per pixel in grayscale) via AXI-Stream Video slave interface, 32 bit data bus (at 4 pixels per clock).
Control of IP core	The IP core is controlled via AXI-Lite slave interface. Control of the core and importation of data from the core are performed through an address space.
Frame size of input video	The allowable frame size of the input video ranges from 240x240 pixels (width and height, respectively) to 2048x2048.
Computation speed	Computation begins upon the strobe of a new frame (arriving through the AXI-Stream interface) and is performed for each frame of the video independently of the previous frame. For the IP core clock frequency of 150MHz, the full computation cycle is: for 128x128 pixels strobe: no more than 25 ms; for 128x64 pixels strobe: no more than 13 ms; for 64x64 pixels strobe: no more than 6.5 ms; for 64x32 pixels strobe: no more than 3.5 ms.
Automatic detection of tracking failure	The IP core automatically evaluates the quality of tracking and decides about its collapse (loss of object). If the object is lost, tracking goes on based on the object motion parameters calculated before the loss (its speed and direction). When the object is detected again, it is automatically recaptured for tracking. If the detection does not occur within a preset number of video frames, tracking is automatically reset.
Reduced probability of automatically capture of similar objects near the tracked object	**This function will be implemented in the core version 3.1.** The algorithm evaluates the trajectory of the object motion and minimizes the probability of capture of similar objects near the tracked object (tracking swap). The algorithm selects several objects similar to the tracked object and analyzes their positions relative to the extrapolated position of the object. If any of the selected objects comes closer to the extrapolated position and has a larger degree of similarity than the others, this object is taken as the tracked one provided that some additional criteria are met.
Shape and configuration of tracked objects	The IP core allows tracking of any types and shapes of objects. The tracking algorithm implemented in the IP core does not perform object recognition and identification, but rather it performs tracking of any specified objects including their parts.

Parameter	Value and note
Allowed rate of change of shape and size of tracked object	The allowed change of the object area is by no more than 50% over at least 50 video frames. The allowed change of the object shape (maximum width-to-height ratio) is by no more than 50% over at least 50 video frames.
Allowed rate of change of the mean brightness of the tracked object	The allowed change of the mean brightness of the tracked object or the area near it bounded by the tracking strobe is by 50% over no less than 50 video frames.
Maximum partial overlap of the object by an occluder (screen)	The allowed partial overlap of the tracked object by an occluder (screen) is by no more than 50% of its area over no more than 40 video frames. In this case, there must be no tracking collapse.
Type of implemented tracking algorithm	Modified correlation tracking algorithm with filtering of background components of the frame.

The basic functions performed by the IP core are as follows:

1. Capture of an object for tracking by using the tracking strobe dimensions and position in the image coordinate system that are specified in the corresponding registers;
2. Computation of the position of the tracked object in the image coordinate system on subsequent video frames (computation is executed by the core automatically and starts with arrival of the tracking strobe of a new frame);
3. Computation of the object motion speed in the image coordinate system;
4. Changing of the tracking algorithm parameters, including in times of tracking;
5. Stopping (resetting) of tracking when the corresponding mode sets in (through writing in the control register) or automatically when automatic reset criteria are fulfilled.

Quality of the automatic tracking depends on the conditions of observation and parameters of the tracked object (its shape and contrast relative to background, and others). To assess the quality of tracking in a variety of situations, a demonstration program is provided.

PRINCIPLE OF OPERATION OF THE IP CORE TRACKING ALGORITHM

The working principle of the algorithm is based on the correlation search method (comparing fragments of a video frame with the reference image of an object formed at the time of capture for tracking and updated in the process of tracking). At the time of capture of the object for tracking, the rectangular area of the video frame specified in the capture settings (position and dimensions) is taken as a reference image of the object. The dimensions of this area correspond to the dimensions of the tracking strobe. In subsequent frames, the object is searched in the area bounded by the algorithm parameters (rectangular area) whose center coincides with the center of the tracking strobe calculated in the previous video frame (or with the center of the capture rectangle if the first frame after the capture is processed). The most likely computed location of the object (calculated center of the tracking strobe) is taken as the coordinates of the tracked object in the current video frame. Figure 1 shows the principle of searching the object in video frames.

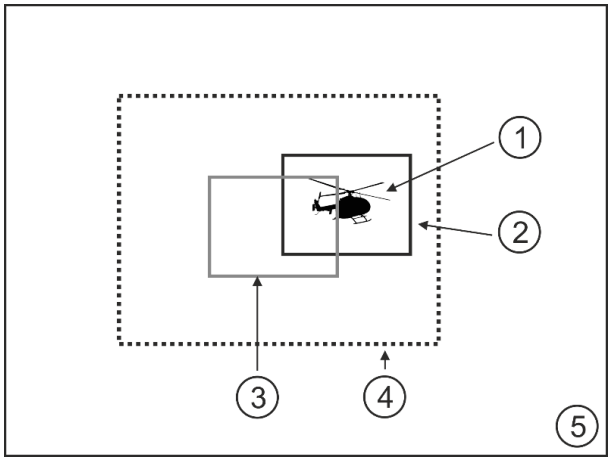

Figure 1 – Principle of searching an object in video frame
(1 – image of the object in a frame, 2 – tracking strobe computed for the current frame, 3 – position of the tracking strobe in the previous frame, 4 – area of search of the object in the current frame, 5 – frame dimensions)

Once the object is captured, the tracking algorithm does not make distinctions between pixels belonging to background or object within the tracking strobe. Over time (as a certain number of frames are processed), the algorithm evaluates whether a pixel inside the rectangle belongs to the object or background. Based on this information, the tracking quality is further increased and the algorithm evaluates the size and position of the object (its image) within the tracking rectangle which enables subsequent adjustment of the algorithm parameters.

Motion parameters of the object (horizontal and vertical velocity components) are computed for each processed video. For each processed frame, the IP core returns the position of the tracking strobe center, the position and size of the target strobe (the rectangle which specifies the object dimensions) in the tracking rectangle, as well as the velocity components of the tracked object. Search of the object in each video frame passed to the processor is carried out in all possible object positions inside the search area. The computation process results in the formation of the surface of spatial distribution of the likelihood of the tracked object's location in the search area on the current frame.

When said surface is formed it is analyzed to determine the most probable position of the tracked object in the processed frame. The analysis takes into account not only the computed probabilities but also the motion trajectory of the tracked object.

OPERATION MODES

Description of the tracking algorithm operation modes

The automatic tracking algorithm implemented in the VIDEO_TRACKING_IPCORE IP core can operate in several different modes (states). Switching from one mode to another can be done by writing 1 or 0 into the respective bits of the control register TRGT_ATTR_REG or automatically. Full description of all registers of the IP core is given in the section «ADDRESS SPACE OF THE VIDEO_TRACKING_CORE IP CORE». Table 4 lists the operating modes of the tracking algorithm.

Table 4 - Operating modes of the automatic tracking algorithm.

Mode name	Mode description and switching conditions
FREE	Free mode. No computation is executed by the IP core in this mode. The core is in this mode after reset. Switching to this mode can be effected from any other mode by setting bit 31 of the register TRGT_ATTR_REG to 0 or automatically when the tracking reset criteria are met.
TRACKING	Automatic tracking mode. In this mode, automatic tracking is computed and all computed parameters of the object are updated. Switching to this mode from the **FREE** mode is effected by setting bit 31 of the register TRGT_ATTR_REG to 1 and can also be performed automatically from the **LOST** mode if the criteria for automatic re-capture for tracking are met. When the reset criteria are fulfilled, automatic switching to the **FREE** mode occurs.
LOST	Tracking collapse mode with prolongation of the object motion trajectory. In this mode, the tracked object coordinates are updated based on the object motion parameters computed before switching to this mode (object's horizontal and vertical velocity components). Switching to this mode occurs automatically from the **TRACKING** mode when the object loss criteria are met. In this mode, the search of the object is effected on the video frames and when the criteria of re-capture for tracking are met the tracker automatically jumps to the **TRACKING** mode. When the reset criteria are fulfilled, the tracker automatically switches to the **FREE** mode.
INERTIAL	Inertial tracking mode. In this mode, the tracked object coordinates are updated based on the object motion parameters calculated before changing to this mode (object's horizontal and vertical velocity components on the video frames). No search for the object is performed in this mode. Switching to this mode is only possible by setting bit 29 of the register TRGT_ATTR_REG to 1 from the **TRACKING** and **LOST** modes. When the reset criteria are fulfilled, automatic changeover to the **FREE** mode occurs.
STATIC	Static mode. In this mode, no operations are executed by the IP core while all parameters of the tracked object computed prior to switching to this mode are retained (coordinates, parameters of the tracking strobe, etc.). The difference from the

Mode name	Mode description and switching conditions
	INERTIAL mode is that the tracking rectangle coordinates are not changed. Switching to this mode is only possible by setting bit 28 of the register TRGT_ATTR_REG to 1.

Figure 2 shows an operating mode switching diagram for the IP core and bits of the register TRGT_ATTR_REG meant for control of the operating modes.

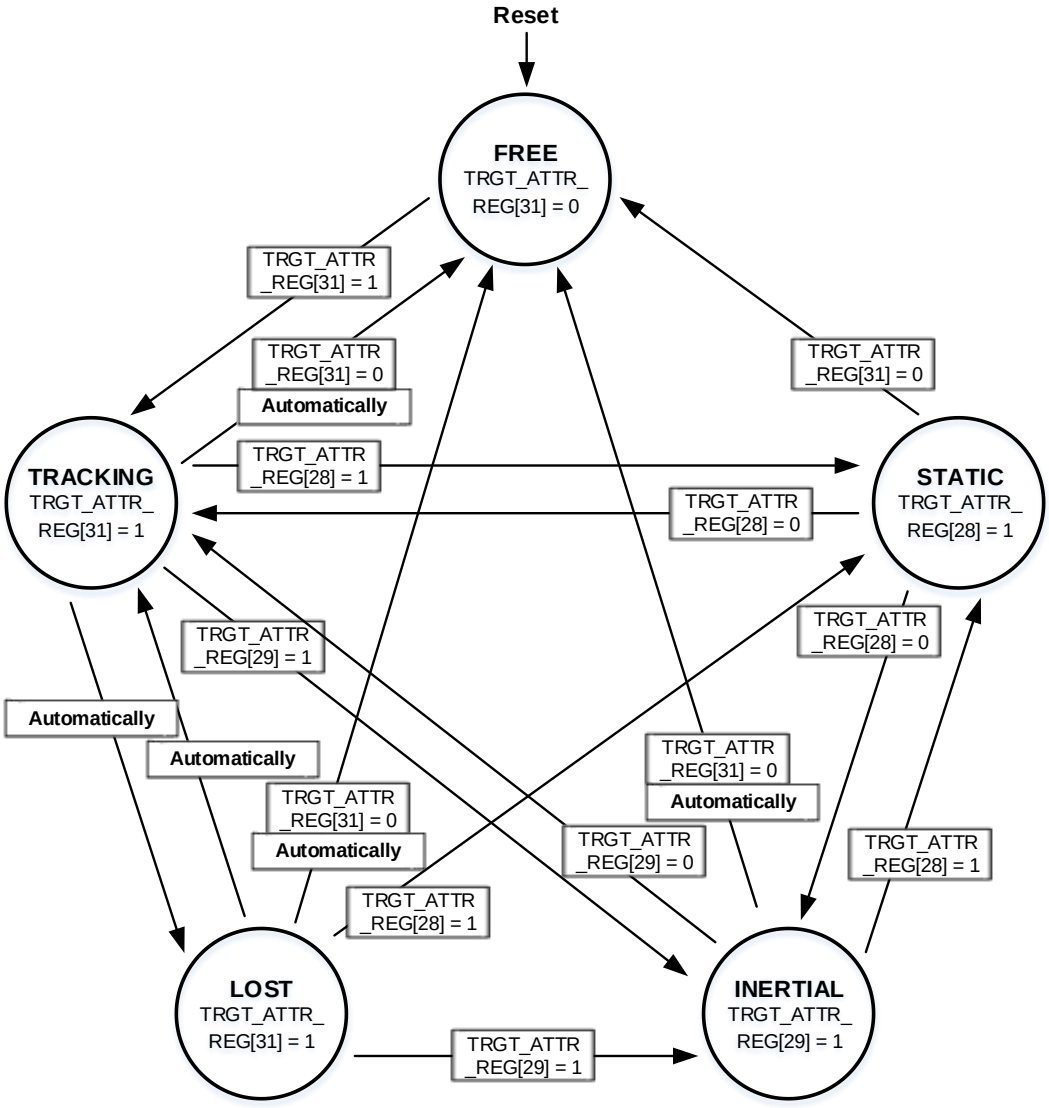

Figure 2 – Diagram of switching between operating modes.

Criteria of automatic changing of the algorithm operation modes

Switching between the modes is effected both by a command (by setting/resetting of the respective bits of the register TRGT_ATTR_REG) and automatically. Automatic switching of the modes can occur if the following criteria are met: tracking collapse criteria, criteria of automatic object re-capture for tracking, and tracking stop criteria.

Automatic changeover from the TRACKING mode to the LOST mode takes place in cases where the greatest computed probability of the tracked object's location in the search area of the current processed frame is less than the algorithm adaptive threshold. In this case, the algorithm decides that the object is lost (tracking collapse occurred) since the highest likelihood of the object's image location in any of the positions in the search area is too low (below the threshold).

Automatic changeover from the LOST mode to the TRACKING mode occurs in the cases where the greatest computed probability of the tracked object's location in the search area on the current processed frame is higher than the algorithm adaptive threshold.

Automatic switching to the FREE mode from the LOST mode can take place in two cases: when the number of video frames in the LOST mode is larger than that set by the register PROLONG_CNTR_REG or when one of the sides of the tracking strobe comes close to the image bounds to a distance of 2 pixels.

Automatic switching to the FREE mode from the other modes occurs if only one of the sides of the tracking strobe comes close to the image bounds to a distance of 2 pixels.

ADDRESS SPACE OF THE VIDEO_TRACKING_CORE IP CORE

Control of the IP core and importation of data about the results of the core functioning are carried out by reading/writing in the corresponding registers. The set of registers of the IP core represents an address space projected to the CPU memory domain (hardware or synthesized). Its structure is given in Table 5.

Table 5 – Address space of the IP core VIDEO_TRACKING_IPCORE.

Shift	Access	Data format	Default value	Description
00h	R/W	binary	00000000h	CORE_CTRL_REG – control register for IP core (parameters not directly pertaining to tracking process)
04h	R/W	binary	00000000h	TRGT_ATTR_REG – control register for operation modes
08h	R/W	unsigned int	200	STRB_RAW_X_REG – register for setting and reading of the tracking strobe position along X axis (horizontal) during IP core work
0Ch	R/W	unsigned int	200	STRB_RAW_Y_REG – register for setting and reading of the tracking strobe position along Y axis (vertical) during IP core work
10h	R/W	unsigned int	51200	STRB_FILT_X_REG – register for setting and reading of the tracking strobe position along X axis (horizontal) at a subpixel accuracy during IP core work. It is related to the register STRB_RAW_X_REG as STRB_FILT_X_REG = 256 * STRB_RAW_X_REG when there is no fractional part.
14h	R/W	unsigned int	51200	STRB_FILT_Y_REG – register for setting and reading of the tracking strobe position along Y axis (vertical) at a subpixel accuracy during IP core work. It is related to the register STRB_RAW_Y_REG as STRB_FILT_Y_REG = 256 * STRB_RAW_Y_REG when there is no fractional part.
18h	R/W	unsigned int	64	STRB_W_REG – register for setting and reading of the tracking strobe width along X axis during IP core work.
1Ch	R/W	unsigned int	64	STRB_H_REG – register for setting and reading of the tracking strobe height along Y axis during IP core work.
20h	R	unsigned int	16	TRGT_X_REG – register for reading of the target tracking strobe position along X axis during IP core work relative to the upper left corner of the tracking strobe.
24h	R	unsigned int	16	TRGT_Y_REG – register for reading of the target tracking strobe position along Y axis during IP core work relative to the upper left corner of the tracking strobe.
28h	R	unsigned int	32	TRGT_W_REG – register for reading of the target tracking strobe width along X axis during IP core work.
2Ch	R	unsigned int	32	TRGT_H_REG – register for reading of the target tracking strobe height along Y axis during IP core work.
30h	R	signed int	0	VEL_X_REG – register for reading of the tracked object velocity along X axis during IP core work. Readability (least significant bit) is 1/256 px/frame. I.e. for the object motion speed of 1 px/frame the value of this register will be 256.
34h	R	signed int	0	VEL_Y_REG – register for reading of the tracked object velocity along Y axis during IP core work. Readability (least significant bit) is 1/256 px/frame. I.e. for the object motion speed of 1 px/frame the value of this register will be 256.
38h	R	signed int	0	ACC_X_REG – register for reading of the tracked object acceleration along X axis during IP core work.
3Ch	R	signed int	0	ACC_Y_REG – register for reading of the tracked object acceleration along Y axis during IP core work.
40h	R	4 x unsigned char	FFFFFFFFh	CORRELATION_REG – value of the correlation factor of the viewed object and the reference over the last 4 frames.
44h	R/W	unsigned int	255	PROLONG_CNTR_REG – starting value of the prolongation countdown counter. In case of tracking collapse the IP core switches to the «LOST» mode and this counter is decremented by each frame. When the counter reaches 0, the IP core switches to the «FREE» mode.

Shift	Access	Data format	Default value	Description
48h – 5Ch	-	-		Reserved.
60h	R/W	signed int	0	STRB_SHIFT_X_REG – forced shift of the tracking strobe relative to the search area along X axis. It is used for correction of the strobe position when the object is tracked by its fragment.
64h	R/W	signed int	0	STRB_SHIFT_Y_REG – forced shift of the tracking strobe relative to the search area along Y axis. It is used for correction of the strobe position when the object is tracked by its fragment.
68h	R/W	binary	00000000h	SRCH_AREA_ATTR_REG – register for controlling of the search area shifting mode.
6Ch	R/W	signed int	0	SRCH_AREA_SHIFT_X_REG – forced shift of the target search area along X axis during IP core work. .
70h	R/W	signed int	0	SRCH_AREA_SHIFT_Y_REG – forced shift of the target search area along Y axis during IP core work.
74h – 9Ch	-	-		Reserved.
A0h	R/W	unsigned int	640	FRAME_WIDTH_REG – width of video frame passed through AXI Stream interface.
A4h	R/W	unsigned int	480	FRAME_HEIGHT_REG – height of video frame passed through AXI Stream interface.
A8h	R/W	unsigned int	128	TRACK_STRB_MAX_W_REG – maximum width of the tracking strobe.
ACh	R/W	unsigned int	128	TRACK_STRB_MAX_H_REG – maximum height of the tracking strobe.
B0h	R/W	unsigned int	56	SEARCH_BORDER_X_REG – target search area along X axis.
B4h	R/W	unsigned int	56	SEARCH_BORDER_Y_REG – target search area along Y axis.
B8h	R/W	unsigned int	8	EXTEND_BORDER_X_REG – additional target search along X axis. It is used for compensation of the frame processing time.
BCh	R/W	unsigned int	8	EXTEND_BORDER_Y_REG – additional target search along Y axis. It is used for compensation of the frame processing time.
C0h	R/W	unsigned char	170	TRACK_DETECTED_LEVEL_REG – object detection threshold. When the values of the correlation factor in the current frame rendering are lower than this threshold the IP core switches to the «LOST» mode.
C4h	R/W	unsigned char	200	TRACK_MEAN_MIN_LEVEL_REG – threshold level for object re-capture for tracking. When the correlation factor exceeds this threshold over the last 4 frames the IP core performs re-capture of the tracked object and switches to the «TRACKING» mode.
C8h	R/W	unsigned char	204	TRACK_COORDS_FILTER_K1_REG – coefficient of the contribution of the current shift of the object during the filtering of its coordinates. It is calculated as K * 256, where K can take on a value from 0 to 1.
CCh	R/W	unsigned char	154	TRACK_COORDS_FILTER_K2_REG – coefficient of the contribution of the measured velocity of the object during the filtering of its coordinates. It is calculated as K * 256, where K can take on a value from 0 to 1.
D0h	R/W	unsigned char	125	TRACK_COORDS_FILTER_K3_REG – coefficient of the contribution of the measured acceleration of the object during the filtering of its coordinates. It is calculated as K * 256, where K can take on a value from 0 to 1.

Register CORE_CTRL_REG

The register is intended for control of general IP functions that are not engaged in object tracking process.

Table 6 – Bit assignments for the register CORE_CTRL_REG.

Bit number	Description
0	Interrupt control bit. Automatically set to 1 after termination of IP core computations (in synchrony with signal DataRdyIntr), which means that the tracked object data is ready. Writing of 0 to this bit removes interrupt signal DataRdyIntr.
1 – 7	Reserved.
8 - 9	Frame format control bits for frames arriving through the AXI Stream interface. They are used to specify type of line scan in the frame processed by the IP core (progressive, even half-frame first interlacing, odd half-frame first interlacing). Values: 00 – progressive scanning, lines are passed sequentially; 01 – interlaced scanning, first half frame is even;

Bit number	Description
	10 – interlaced scanning, first half frame is odd; 11 – reserved;
10 - 31	Reserved.

Register TRGT_ATTR_REG

The register is intended for control of the IP core operation mode and retrieval of the IP core status during object tracking.

Table 7 – Bit assignments for the register TRGT_ATTR_REG.

Bit number	Description
0 - 28	Reserved.
28	Bit switching IP core to the «STATIC» mode. 0 – normal operation mode; 1 – static mode.
29	Bit switching IP core to the «INERTIAL» mode (forced prolongation of the object trajectory). 0 – normal operation mode; 1 – forced prolongation mode.
30	Flag of detection of object in the processed frame. Writing to this bit does not change its value. If reading is: 0 – object not detected; 1 – object detected.
31	Bit switching IP core to the «TRACKING» mode: 0 – «FREE» mode; 1 – «TRACKING» mode.

Register SRCH_AREA_ATTR_REG

The register is intended for control of the search area shifting mode. Allows to shift the search area and put core to the "STATIC" mode for specified number of frames. This function can be used in the presence of a priori information about the potential tracking collapse (mechanical or other impact on the source of video information, etc.).

Table 8 – Bit assignments for the register SRCH_AREA_ATTR_REG.

Bit number	Description
0 - 15	The number of frames to put core to the "STATIC" mode. If the value is greater than 0, then after the addition of the shifts (if bit 31 is set), the kernel will be in the "STATIC" mode for the specified number of frames.
16 - 30	Reserved.
31	Activation of the search area shifting function: 0 – function is not active; 1 – activate the function, while the values of the SRCH_AREA_SHIFT_X_REG and SRCH_AREA_SHIFT_Y_REG registers will be added to the current position of the tracking strobe, search for the object on next frame will be in a new area. A bit with auto reset, i.e. the operation of adding one-time.

IP CORE USAGE PROCEDURE

In this document version, the usage procedure of the VIDEO_TRACKING_CORE IP core is considered using the example of development environment Xilinx Vivado 2016.2. For other programming environments the procedure is similar but their specific features have to be taken into account.

Importing the IP core into existing or created project

The VIDEO_TRACKING_CORE IP core is supplied as a folder containing pre-configured IP core project according to the customer's order:

1. source codes in the hardware languages Verilog and VHDL;
2. the results of synthesis for selected IC series in the format of EDIF Implementation Netlist File «*.edn»;
3. the results of synthesis for selected IC series in the format of Xilinx Generated Netlist File «*.ngc»;

4. the results of synthesis for selected IC series in the format of the IP core Altera Qsys (on request).

Importing the IP core into a project is carried out as follows:

1. In the «Project Manager» menu, select the item «IP Catalog» (Figure3). The tab will open that controls the IP core repositories (Figure 4).

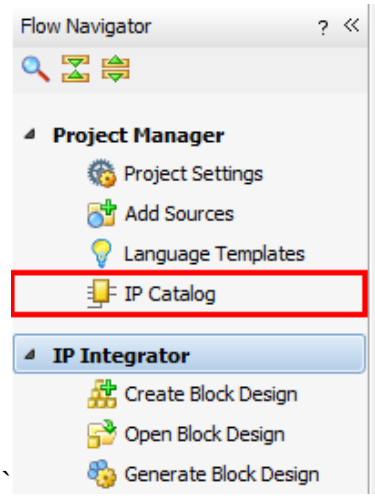

Figure 3 –«IP Catalog» menu item.

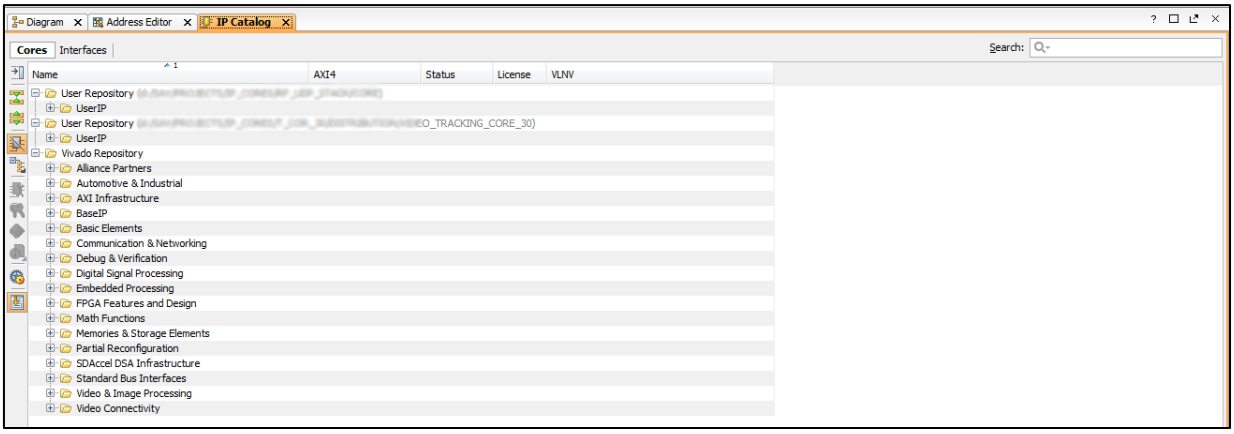

Figure 4 – IP core repository control tab.

2. By right-clicking on the «Vivado Repository» item, select item «Add Repository» and designate the folder containing the VIDEO_TRACKING_CORE IP core;
3. By opening or creating a new block design (Block Design, Figure 2) add the IP core symbol to the project working space (Figure 5).

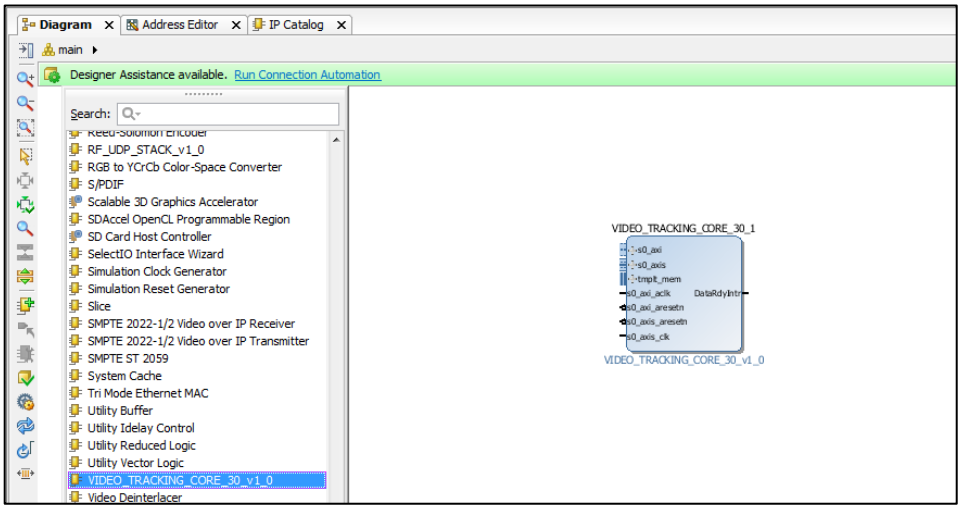

Figure 5 –Symbol of the VIDEO_TRACKING_CORE IP core in a new project.

Project infrastructure and connection of the IP core

The VIDEO_TRACKING_CORE IP core has the following interfaces:

1. AXI-Lite «s0_axi» (including signals s0_axi_aclk and s0_axi_aresetn) gives access to the control and core status registers on the CPU side;
2. AXI-Stream Video «s0_axis» (including signals s0_axis_aclk и s0_axis_aresetn) serves to receive streaming video by the core;
3. BRAM «tmplt_mem» gives CPU access to the tracked object image; this interface is optional and is used if it is necessary to show tracked object;
4. DataRdyIntr is the interrupt signal for the CPU and is set to 1 when the incoming frame has been rendered; this signal is reset to 0 by the CPU through writing in the control register (see section where the IP core registers are described).

In general, the project should contain the following elements:

1. Module or IP-core for receiving video via standard interfaces (for example, Video in to AXI4-Stream or other);
2. Synthesized or hardware CPU for control of the IP core;
3. VIDEO_TRACKING_CORE IP core.

BASE PROJECT FOR TESTING THE VIDEO_TRACKING_CORE IP CORE

The base project is designed to verify the correctness of integration and gain experience with the IP core. The project includes only ready-made blocks and requires minimal developer training, does not require the use of a video source (video patterns are generated programmatically), or other external interfaces other than UART for outputting debugging information. The structure of the base project for the core testing is shown in Figure 6.

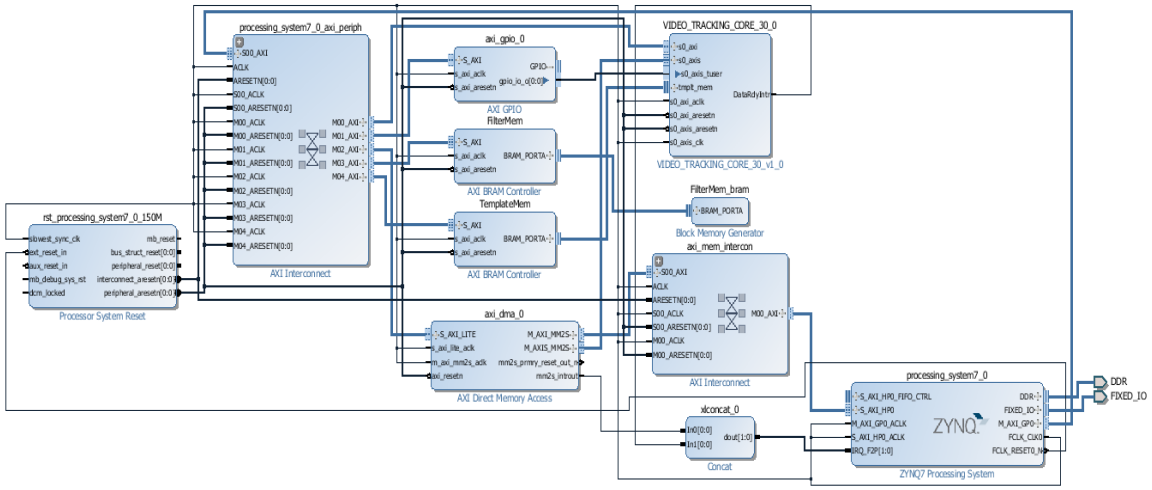

Figure 5 – Structure of the base project for testing the VIDEO_TRACKING_CORE IP core.

Logically, the basic project is divided into 2 global parts: hardware and software.

1. The hardware part implements the logic of the IP-cores used;
2. The software part is executed by the CPU and performs the following tasks: forming an image frame, setting up and launching the DMA for transferring the frame to the VIDEO_TRACKING_CORE IP core, generating a new frame strobe through AXI GPIO, controlling the operation modes of the VIDEO_TRACKING_CORE IP core, outputting the work results and debugging information through the UART.

Base project hardware

The hardware part of the basic project includes:

1. reset generation module "rst_processing_system7_0_150M";

2. module for connection and arbitration of the periphery to the interface Master AXI CPU "processing_system7_0_axi_periph";
3. module AXI GPIO "axi_gpio_0" which generates a strobe for the start of a new frame (because the video frames in this project are formed by the CPU);
4. modules AXI BRAM Controller "FilterMem" and Block Memory Generator "FilterMem_bram", not used in the current version of the project;
5. module AXI BRAM Controller "TemplateMem" which secures CPU access to the image of tracked object;
6. module AXI Direct Memory Access module "axi_dma_0" for outputting program-generated frame image via the AXI-Stream interface to the VIDEO_ TRACKING_CORE IP core;
7. VIDEO_TRACKING_CORE IP-core "VIDEO_TRACKING_CORE_30_0";
8. AXI Interconnect module "axi_mem_intercon" for accessing through the CPU to external memory;
9. Concat module "xlconcat_0" for receiving and combining interrupts from "axi_dma_0" and "VIDEO_TRACKING_CORE_30_0";
10. CPU ZYNQ7 Processing System hardware module

Base project software

The software part of the base project is a CPU executable program module designed to generate regular video frames wherein a white rectangle moves at a specified speed, output each line of the generated frame to the VIDEO_TRACKING_CORE IP core via DMA, switch the core to the "TRACKING" mode, handle interrupt of the end of frame rendering by the core and receive the current results represented as the core status, position of the tracked target, its speed and other parameters.

The interaction between the software part and the VIDEO_TRACKING_CORE IP core is carried out by reading/writing of the address space (memory space, see “ADDRESS SPACE OF THE VIDEO_TRACKING_CORE IP CORE"). Below is the source code for the software part of the base project.

Listing:

```
#include <stdlib.h>
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xscugic.h"

//Interrupt controller – used to prepare and trigger transfer of the next line
XScuGic                 InterruptController;

//Frame buffers – one is being filled while the other is sent line by line to the tracking core
int                     FrameW          = 640;
int                     FrameH          = 480;
unsigned   char*        FrmBuff1;
unsigned   char*        FrmBuff2;
unsigned   char*        FrmBuffToDraw;
unsigned   char*        FrmBuffToSend;
char                    TrnsmLineDone   = 0;// Flag of end of transmission of a line – next transmission can be started

//Strobe parameters
unsigned char           StrbW           = 64;
unsigned char           StrbH           = 32;
//Target drawing parameters
float                   TargetX         = 200;
float                   TargetY         = 200;
float                   TargetW         = 6;
float                   TargetH         = 6;
float                   TargetVelX      = 1;
float                   TargetVelY      = 0;
float                   TargetAccX      = 0;
float                   TargetAccY      = 0;
unsigned   char         TargetBr        = 128;
unsigned   char         BackBr          = 32;
unsigned   char         BackNoiseAmp    = 0;

```

```
35.
36. //Addresses of VIDEO_TRACKING_CORE IP core control registers
37. #define       CORE_CTRL_REG_ADDR                0x00
38. #define       TRGT_ATTR_REG_ADDR                0x04
39. #define       STRB_RAW_X_REG_ADDR               0x08
40. #define       STRB_RAW_Y_REG_ADDR               0x0C
41. #define       STRB_FILT_X_REG_ADDR              0x10
42. #define       STRB_FILT_Y_REG_ADDR              0x14
43. #define       STRB_W_REG_ADDR                   0x18
44. #define       STRB_H_REG_ADDR                   0x1C
45. #define       TRGT_X_REG_ADDR                   0x20
46. #define       TRGT_Y_REG_ADDR                   0x24
47. #define       TRGT_W_REG_ADDR                   0x28
48. #define       TRGT_H_REG_ADDR                   0x2C
49. #define       VEL_X_REG_ADDR                    0x30
50. #define       VEL_Y_REG_ADDR                    0x34
51. #define       ACC_X_REG_ADDR                    0x38
52. #define       ACC_Y_REG_ADDR                    0x3C
53. #define       CORRELATION_REG_ADDR              0x40
54. #define       PROLONG_CNTR_REG_ADDR             0x44
55. #define       STRB_SHIFT_X_REG_ADDR             0x60
56. #define       STRB_SHIFT_Y_REG_ADDR             0x64
57. #define       SRCH_AREA_ATTR_REG_ADDR           0x68
58. #define       SRCH_AREA_SHIFT_X_REG_ADDR        0x6C
59. #define       SRCH_AREA_SHIFT_Y_REG_ADDR        0x70
60. #define       FRAME_WIDTH_REG_ADDR              0xA0
61. #define       FRAME_HEIGHT_REG_ADDR             0xA4
62. #define       TRACK_STRB_MAX_W_REG_ADDR         0xA8
63. #define       TRACK_STRB_MAX_H_REG_ADDR         0xAC
64. #define       SEARCH_BORDER_X_REG_ADDR          0xB0
65. #define       SEARCH_BORDER_Y_REG_ADDR          0xB4
66. #define       EXTEND_BORDER_X_REG_ADDR          0xB8
67. #define       EXTEND_BORDER_Y_REG_ADDR          0xBC
68. #define       TRACK_DETECTED_LEVEL_REG_ADDR     0xC0
69. #define       TRACK_MEAN_MIN_LEVEL_REG_ADDR     0xC4
70. #define       TRACK_COORDS_FILTER_K1_REG_ADDR   0xC8
71. #define       TRACK_COORDS_FILTER_K2_REG_ADDR   0xCC
72. #define       TRACK_COORDS_FILTER_K3_REG_ADDR   0xD0
73.
74. //Frame counter for periodic generation of various commands
75. int                         FramesCntr      = 0;
76.
77.
78. //Forward declarations of the functions
79. void                       disable_caches();
80. void                       EnableTracking(int X, int Y, u8 W, u8 H);
81. void                       DrawLine(unsigned char* FrameBuff, int LineIdx);
82. void                       DMATrnsmLineDone(void);
83. void                       TrackingDataReady();
84.
85. #define       BOOL(x) (!(!(x)))
86. #define       BitSet(arg,posn) ((arg) | (1L << (posn)))
87. #define       BitClr(arg,posn) ((arg) & ~(1L << (posn)))
88. #define       BitTst(arg,posn) BOOL((arg) & (1L << (posn)))
89. #define       BitFlp(arg,posn) ((arg) ^ (1L << (posn)))
90.
91.
92. int main()
93. {
94.       u32       RegVal;
95.       int       LineIdx = 0;
96.
97.       init_platform();
98.       disable_caches();
99.       xil_printf("Start T_CORR_30_DEMO.\n\r");
100.
101.      FrmBuff1      = malloc(2048*2048);
```

```
102.    FrmBuff2      = malloc(2048*2048);
103.    FrmBuffToDraw  = FrmBuff1;
104.    FrmBuffToSend  = FrmBuff2;
105.
106.    // Set default parameters
107.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRGT_ATTR_REG_ADDR, 0x00);  // Reset tracking flags
108.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + FRAME_WIDTH_REG_ADDR, FrameW);  // Horizontal frame
        dimension
109.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + FRAME_HEIGHT_REG_ADDR, FrameH);  // Vertical frame
        dimension
110.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRACK_STRB_MAX_W_REG_ADDR, 128);// Maximum strobe
        dimensions along X axis
111.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRACK_STRB_MAX_H_REG_ADDR, 128);// Maximum strobe
        dimensions along Y axis
112.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + SEARCH_BORDER_X_REG_ADDR, 56);    // Size of search area
        along X axis
113.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + SEARCH_BORDER_Y_REG_ADDR, 56);    // Size of search area
        along Y axis
114.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + EXTEND_BORDER_X_REG_ADDR, 8);// Size of extension area
        along X axis
115.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + EXTEND_BORDER_Y_REG_ADDR, 8);// Size of extension area
        along Y axis
116.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRACK_DETECTED_LEVEL_REG_ADDR, 128);// Detection
        threshold
117.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRACK_MEAN_MIN_LEVEL_REG_ADDR, 128);    // Stable
        detection threshold
118.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRACK_COORDS_FILTER_K1_REG_ADDR, 256 * 0.8);
          //Coefficient of K1 filter
119.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRACK_COORDS_FILTER_K2_REG_ADDR, 256 * 0.6);
          //Coefficient of K2 filter
120.    Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRACK_COORDS_FILTER_K3_REG_ADDR, 256 * 0.49);
          //Coefficient of K3 filter
121.
122.    //Configure DMA controller for outputting lines to the tracking core
123.    //Reset AXI DMA core
124.    RegVal    = 0x04;
125.    Xil_Out32(XPAR_AXIDMA_0_BASEADDR + 0x00, RegVal);
126.    //Wait for end of the core reset
127.    do{
128.      RegVal  = Xil_In32(XPAR_AXIDMA_0_BASEADDR + 0x00);
129.    }while(RegVal & (1 << 2));
130.    // Initialize AXI DMA
131.    RegVal    = Xil_In32(XPAR_AXIDMA_0_BASEADDR + 0x00);
132.    RegVal    = RegVal | 0x1001;
133.    Xil_Out32(XPAR_AXIDMA_0_BASEADDR + 0x00, RegVal);
134.    RegVal    = Xil_In32(XPAR_AXIDMA_0_BASEADDR + 0x00);
135.    xil_printf("AXI DMA control register value: %x\n\r", RegVal);
136.    RegVal    = Xil_In32(XPAR_AXIDMA_0_BASEADDR + 0x04);
137.    xil_printf("AXI DMA status register value: %x\n\r", RegVal);
138.
139.    // Configure interrupt controller and turn on interrupts
140.    Xil_ExceptionInit();
141.    XScuGic_DeviceInitialize(XPAR_SCUGIC_SINGLE_DEVICE_ID);
142.    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IRQ_INT, (Xil_ExceptionHandler)XScuGic_DeviceInterruptHandler, (void
        *)XPAR_SCUGIC_SINGLE_DEVICE_ID);
143.    // Set up interrupt handler for end of transmission of the next line
144.    XScuGic_RegisterHandler(XPAR_SCUGIC_CPU_BASEADDR, XPAR_FABRIC_AXI_DMA_0_MM2S_INTROUT_INTR,
        (Xil_ExceptionHandler)DMATrnsmLineDone, NULL);
145.    // Set up interrupt handler for end of operation of the T_CORR_30 core
146.    XScuGic_RegisterHandler(XPAR_SCUGIC_CPU_BASEADDR, XPAR_FABRIC_VIDEO_TRACKING_CORE_30_0_DATARDYINTR_INTR,
        (Xil_ExceptionHandler)TrackingDataReady, NULL);
147.    // Enable interrupts
148.    XScuGic_EnableIntr(XPAR_SCUGIC_DIST_BASEADDR, XPAR_FABRIC_AXI_DMA_0_MM2S_INTROUT_INTR);
149.    XScuGic_EnableIntr(XPAR_SCUGIC_DIST_BASEADDR, XPAR_FABRIC_VIDEO_TRACKING_CORE_30_0_DATARDYINTR_INTR);
150.
151.    Xil_ExceptionEnable();
152.
```

```
153.      // Generate strobe of a new frame
154.      Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + 0x0000, 0x01);
155.      Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + 0x0000, 0x00);
156.      // Prepare the first frame and start its transmission
157.      for (LineIdx = 0; LineIdx < FrameH; ++LineIdx)
158.      {
159.        DrawLine(FrmBuffToDraw, LineIdx);
160.      }
161.      // Swap buffers
162.      FrmBuffToSend   = FrmBuffToDraw;
163.      FrmBuffToDraw   = FrmBuff2;
164.      LineIdx         = 0;
165.
166.      while(1)
167.      {
168.        // Start transmission of the next line
169.        Xil_Out32(XPAR_AXIDMA_0_BASEADDR + 0x18, (unsigned int)FrmBuffToSend);
170.        Xil_Out32(XPAR_AXIDMA_0_BASEADDR + 0x28, FrameW);
171.        // Draw the next line in the buffer
172.        DrawLine(FrmBuffToDraw, LineIdx);
173.        // Execute shift to the next line
174.        LineIdx++;
175.        FrmBuffToSend+= FrameW * sizeof(typeof(*FrmBuffToSend));
176.        // Check flag of end of transmission of the line from DMA to the tracking
177.        while (!TrnsmLineDone)     {};
178.        // Check if the line was the last one for drawing; if yes, then reset lines counter,
179.        // set the buffer ready flag and change the target parameters
180.        if (LineIdx == FrameH)
181.        {
182.            // Generate strobe of a new frame
183.            Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + 0x0000, 0x01);
184.            Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + 0x0000, 0x00);
185.
186.            // Increment the frame counter and check if there is a need to send command to the core
187.            FramesCntr++;
188.            if (FramesCntr == 2)      EnableTracking(TargetX, TargetY, StrbW, StrbH);
189.
190.            // Swap buffers
191.            FrmBuffToSend= FrmBuffToDraw;
192.            FrmBuffToDraw      = (FrmBuffToDraw == FrmBuff1) ? FrmBuff2 : FrmBuff1;
193.            // Zero the number of the line that we work with
194.            LineIdx       = 0;
195.            // Update the target parameters
196.            TargetX      += TargetVelX;
197.            TargetY      += TargetVelY;
198.            if (TargetX >= (FrameW - 0.6*StrbW))
199.            {
200.                TargetX      = (FrameW - 0.6*StrbW);
201.                TargetVelX = -TargetVelX;
202.            }else     if (TargetX <= (0 + 0.6*StrbW))
203.                      {
204.                          TargetX      = (0 + 0.6*StrbW);
205.                          TargetVelX = -TargetVelX;
206.                      }
207.            if (TargetY >= (FrameH - 0.6*TargetH))
208.            {
209.                TargetY      = (FrameH- 0.6*TargetH);
210.                TargetVelY = -TargetVelY;
211.            }else     if (TargetY <= (0 + 0.6*TargetH))
212.                      {
213.                          TargetY      = (0 + 0.6*TargetH);
214.                          TargetVelY = -TargetVelY;
215.                      }
216.            TargetVelX += TargetAccX;
217.            TargetVelY += TargetAccY;
218.        }
219.      };
```

```
220.
221.
222.      cleanup_platform();
223.      return 0;
224.      }
225.
226.      void EnableTracking(int X, int Y, u8 W, u8 H)
227.      {
228.        Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + STRB_RAW_X_REG_ADDR, X); // Strobe position along X
229.        Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + STRB_RAW_Y_REG_ADDR, Y); // Strobe position along Y
230.        Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + STRB_W_REG_ADDR, W);  // Strobe size along X
231.        Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + STRB_H_REG_ADDR, H);    // Strobe size along Y
232.        Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + PROLONG_CNTR_REG_ADDR, 1000);   // Prolongation counter
233.        Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRGT_ATTR_REG_ADDR, 0x80000000);     // Command to
          capture
234.      };
235.
236.
237.      void DrawLine(unsigned char* FrameBuff, int LineIdx)
238.      {
239.        int       X;
240.        int       PixValue;
241.
242.        // Shift to the line start address within the frame buffer
243.        FrameBuff  += LineIdx * FrameW * sizeof(typeof(*FrameBuff));
244.
245.        for (X = 0; X < FrameW; ++X)
246.        {
247.            if ((X >= (TargetX-TargetW/2))&&(X < (TargetX+TargetW/2))&&
248.                (LineIdx >= (TargetY-TargetH/2))&&(LineIdx < (TargetY+TargetH/2)))
249.                    PixValue     = 0xFF;//TargetBr;
250.            else
251.            if ((X >= (TargetX-(64)))&&(X < (TargetX+(64)))&&
252.                (LineIdx >= (TargetY-(64)))&&(LineIdx < (TargetY+(64))))
253.                    PixValue     = 1 + X - (TargetX-(64));
254.            else      PixValue     = 0;//BackBr + rand()%BackNoiseAmp;
255.            // Check the limit for the brightness range
256.            if (PixValue < 0)               *FrameBuff      = 0;
257.            else      if (PixValue > 255)       *FrameBuff      = 255;
258.                      else                      *FrameBuff      = PixValue;
259.            // Shift to the next pixel
260.            FrameBuff++;
261.        }
262.      };
263.
264.      void DMATrnsmLineDone(void)
265.      {
266.        u32      RegValue;
267.        // Reset interrupt flag
268.        RegValue             = Xil_In32(XPAR_AXIDMA_0_BASEADDR + 0x04);
269.        RegValue             = RegValue | 0x1000;
270.        Xil_Out32(XPAR_AXIDMA_0_BASEADDR + 0x04, RegValue);
271.        // Set flag of end of transmission of a line
272.        TrnsmLineDone          = 1;
273.
274.      };
275.
276.      void TrackingDataReady()
277.      {
278.        u32RegValue;
279.        // Reset flag of the core interrupt
280.        RegValue             = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + CORE_CTRL_REG_ADDR);
281.        RegValue             = BitClr(RegValue, 0);
282.        Xil_Out32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + 0x00, RegValue);
283.
284.        // Read current coordinates of the tracking strobe and other parameters
285.        u32      TrgtAttr      = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + TRGT_ATTR_REG_ADDR);
```

```
286.      u32    StrbExtX    = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + STRB_FILT_X_REG_ADDR);
287.      u32    StrbExtY    = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + STRB_FILT_Y_REG_ADDR);
288.      u32    StrbX       = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + STRB_RAW_X_REG_ADDR);
289.      s32    VelX        = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + VEL_X_REG_ADDR);
290.      s32    AccelX      = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + ACC_X_REG_ADDR);
291.      u32    StrbY       = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + STRB_RAW_Y_REG_ADDR);
292.      u32    CorrData    = Xil_In32(XPAR_VIDEO_TRACKING_CORE_30_0_BASEADDR + CORRELATION_REG_ADDR);
293.      xil_printf("Attr: %x;Corr: %x;    ExtX: %d;    ExtY: %d;    X: %d;  Y: %d;\n\r", TrgtAttr, CorrData, StrbExtX, StrbExtY, StrbX,
          StrbY);
294.      xil_printf("                      VelX: %d;    AccelX: %d;\n\r", VelX, AccelX);
295.    };
```

The main elements of the software part have the following designations:

- **lines 1-5:** connection of standard and platform-dependent libraries;
- **line 8:** creation of a global reference to the interrupt controller subsequently used to receive interrupts from the DMA controller and VIDEO_TRACKING_CORE IP core;
- **lines 11-17:** setting the size of video frames and creating pointers to frame buffers for drawing frames with a moving rectangle;
- **lines 20-33:** setting the parameters of the tracking strobe, parameters of the target and background to be drawn; the variables have the following sense:
  - StrbW – tracking strobe width
  - StrbH – tracking strobe height
  - TargetX – starting position of the target along X axis;
  - TargetY – starting position of the target along Y axis;
  - TargetW – targert width;
  - TargetH – target height;
  - TargetVelX – target velocity along X axis;
  - TargetVelY – target velocity along Y axis;
  - TargetAccX – target acceleration along X axis;
  - TargetAccY – target acceleration along Y axis;
  - TargetBr – target brightness;
  - BackBr – background brightness;
  - BackNoiseAmp – additive noise component of background brightness.
- **lines 37-72:** setting the addresses of IP core registers as definitions for convenience of further use;
- **lines 79-83:** forward declaration of functions used in the project:
  - disable_caches () – prohibits the use of CPU caches (in this case, frames will be drawn directly in the RAM, whence they will be transferred to the VIDEO_TRACKING_CORE IP core via DMA);
  - EnableTracking (...) – the procedure of capturing a target for automatic tracking;
  - DrawLine (...) – the procedure of drawing the next line of the frame;
  - DMATrnsmLineDone (...) – the procedure of handling the interrupt from the DMA module, it is called after the next line is sent;
  - TrackingDataReady () – the procedure of handling the interrupt from the VIDEO_TRACKING_ CORE IP core, it is called after rendering the next frame.
- **lines 85-89:** auxiliary definitions for controlling individual bits of words;
- **lines 101-104:** allocation of memory for frame buffers and assignment of the first buffer - for drawing, and the second - for sending to the IP core;
- **lines 107-120:** setting the initial parameters of the VIDEO_TRACKING_CORE IP core;
- **lines 124-137:** soft reset and initialization of the DMA;
- **lines 140-151:** configuring the interrupt controller, setting interrupt handlers and enabling interrupts;
- **lines 154-164:** generating the first strobe of a new frame and drawing the first frame of the video;
- **line 166:** entering an endless cycle of drawing and sending video frames, controlling the position of the target;
- **lines 169-170:** starting DMA for transmission of the frame line, the variable FrmBuffToSend specifies the start address in memory (address of the beginning of the line), FrameW – the number of bytes requiring transmission;
- **line 172:** calling the procedure of drawing the next line of the video frame;
- **lines 174-175:** incrementing the index of a line for rendering and shifting the address by the line size;

- **line 177:** waiting for the end-of-line-transmission flag to be set by the DMA module; the flag is set in the interrupt handler DMATrnsmLineDone (...);
- **line 180:** checking the end of frame transmission: the frame is transmitted when the index of the next line is equal to the number of lines in the frame;
- **lines 183-184:** generating the first strobe of the new frame;
- **lines 187-188:** increment of the frame index; check if frame index = 2, i.e. 2 frames are transmitted, then the procedure of capturing the target for tracking is called;
- **lines 191-192:** change of frame buffers – the next frame will be drawn to the buffer from which the data was transmitted, and the buffer with the newly drawn frame will be transmitted to the IP core via DMA;
- **line 194:** resetting the line index because drawing of a new video frame is started;
- **lines 196-217:** control of the position and parameters of the target movement;
- **lines 226-234:** the body of the procedure for capturing the target for automatic tracking; the following actions are successively performed: position of the tracking strobe is set along the X and Y axes, dimensions of the tracking strobe are set, the value of the prolongation counter is set (used in case of tracking collapse), a command is transmitted to switch the IP core to automatic tracking mode;
- **lines 237-262:** the body of the procedure for drawing the line of the video frame;
- **lines 264-274:** the body of the interrupt service routine; it is called by the DMA module on completion of the transmission of the frame line;
- **lines 276-295:** the body of the interrupt service routine for termination of rendering of the next frame by the VIDEO_TRACKING_CORE IP core; the following actions are successively performed: the current value of the core control register CORE_CTRL_REG_ADDR is read, the interrupt bit is reset, a new value is written to the CORE_CTRL_REG_ADDR register, the current parameters of the tracking process are read and output to the standard output port (object attributes - TRGT_ATTR_REG_ADDR, position with subpixel precision - STRB_FILT_X_REG_ADDR, STRB_FILT_Y_REG_ADDR, position in pixels along the X axis - STRB_RAW_X_REG_ADDR, velocity and acceleration along the X axis - VEL_X_REG_ADDR, ACC_X_REG_ADDR, position in pixels along the Y axis - STRB_RAW_Y_REG_ADDR, correlation coefficient value over the last 4 frames - CORRELATION_REG_ADDR).