www.ietdl.org

Published in IET Computers & Digital Techniques
Received on 16th November 2011
Revised on 16th May 2012
doi: 10.1049/iet-cdt.2011.0156

ISSN 1751-8601

Image and video processing platform for field programmable gate arrays using a high-level synthesis

C. Desmouliers[1] *E. Oruklu*[1] *S. Aslan*[1] *J. Saniie*[2] *F.M. Vallina*[3]

[1]*Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, Illinois 60616, USA*
[2]*Ingram School of Engineering, Texas State University, San Marcos, Texas 78666, USA*
[3]*Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, USA*
E-mail: erdal@ece.iit.edu

Abstract: In this study, an image and video processing platform (IVPP) based on field programmable gate array (FPGAs) is presented. This hardware/software co-design platform has been implemented on a Xilinx Virtex-5 FPGA using a high-level synthesis and can be used to realise and test complex algorithms for real-time image and video processing applications. The video interface blocks are done in Register Transfer Languages and can be configured using the MicroBlaze processor allowing the support of multiple video resolutions. The IVPP provides the required logic to easily plug-in the generated processing blocks without modifying the front-end (capturing video data) and the back-end (displaying processed output data). The IVPP can be a complete hardware solution for a broad range of real-time image/video processing applications including video encoding/decoding, surveillance, detection and recognition.

1 Introduction

Today, a significant number of embedded systems focus on multimedia applications with almost insatiable demand for low-cost, high-performance and low-power hardware. Designing complex systems such as image and video processing, compression, face recognition, object tracking, 3G or 4G modems, multi-standard CODECs and high definition (HD) decoding schemes requires integration of many complex blocks and a long verification process [1]. These complex designs are based on the I/O peripherals, one or more processors, bus interfaces, A/D, D/A, embedded software, memories and sensors. A complete system used to be designed with multiple chips and was connected together on PCBs, but with today's technology, all the functions can be incorporated in a single chip. These complete systems are known as system-on-chip (SoC) [2].

Image and video processing applications require a large amount of data transfers between the input and output of a system. For example, a 1024 × 768 colour image has a size of 2 359 296 bytes. This large amount of data needs to be stored in the memory, transferred to the processing blocks and sent to the display unit. Designing an image and video processing unit can be complex and time consuming, and the verification process can take months depending on the system's complexity.

The design difficulty and longer verification processes create a bottleneck for the image and video processing applications. One of the methods used to ease this bottleneck is to generate a 'virtual platform', [3–6] which is a software level design using high-level languages to test an algorithm, to create a software application even before the hardware is available and most importantly to create a model also known as a 'golden reference model' that can be used for the verification process of Register Transfer Languages (RTL) design [3].

In many image and video processing applications, most of the I/O, memory interface and communication channels are common across designs, and the only block that needs to be altered would be the processing block. This reuse of platform components allows for accelerated generation of the golden reference for the new processing block and faster HW/SW co-design at the system level. Also, the RTL generation and verification process would be shorter. Therefore in this paper, an embedded HW/SW co-design platform based on reconfigurable field programmable gate array (FPGA) architecture is proposed for image and video processing applications. The proposed platform uses FPGA development tools, provides an adaptable, modular architecture for future-proof designs and shortens the development time of multiple applications with a single, common framework.

In the past, several platforms have been developed for multimedia processing, including DSP chips based on very-large instruction word (VLIW) architectures. DSPs usually run at higher clock frequencies compared with the FPGAs, however, the hardware parallelism (i.e. number of accelerators, multipliers etc.) is inferior. More importantly, they are not as flexible and may not meet the demands of firmware updates or revisions of multimedia standards. The shortcomings of the DSP and general purpose processors led to a more rapid adoption of reprogrammable hardware such as FPGAs in multimedia applications [7]. Schumacher *et al.* [8] proposed a prototyping framework for multiple hardware IP blocks on an FPGA. Their MPEG4 solution creates an abstraction of the FPGA platform by having a

© The Institution of Engineering and Technology 2012

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

virtual socket layer that is located between the design and test elements, which reside on desktop computers. A different approach [9, 10] uses a layered reconfigurable architecture based on a partial and dynamic reconfigurable FPGA in order to meet the needs for adaptability and scalability in multimedia applications. In [11], an instruction set extension is used for a motion estimation algorithm required in the H.263 video encoder, and the authors incorporate custom logic instructions into a softcore NiOS II CPU within an Altera FPGA. In [12], a high-level synthesis (HLS)-based face detection algorithm is presented for implementing the convolutional face finder algorithm on Virtex-4. ultraSONIC [13, 14] is a reconfigurable architecture offering parallel processing capability for video applications based on plug-in processing elements (PIPEs) – each of which can be an individual FPGA device. It also provides an application programming interface and a software driver that abstracts the task of writing software from the low-level interactions within the physical layer. Although this is a very capable system with a scalable architecture, it is still difficult to develop for and create applications because of the low-level design of PIPEs and/or accelerator functions. A new driver has to be written for each PIPE design and the corresponding video applications.

More closely related works to the image and video processing platform (IVPP) presented in this study are the IP core generators from Altera and Xilinx. The Altera video and image processing (VIP) suite [15] is a collection of IP core (MegaCore) functions that can be used to facilitate the development of custom VIP designs. The functions include frame reader, colour space converter, deinterlacer, filtering and so on. Xilinx offers a similar set of IP cores such as LogiCORE IP video timing controller [16], which supports the generation of output timing signals, automatic detection and generation of horizontal and vertical video timing signals, support for multiple combinations of blanking or synchronisation signals and so on.

Although these IP cores can also be used and integrated in the IVPP, the main feature of the IVPP is the support for the HLS tools by generating the necessary interfacing signals that can be used in high level C programs. This feature requires the custom IVPP blocks described in this paper.

We present a new design framework targeting FPGA-based video processing applications with the purpose of accelerating the development time by utilising pre-built hardware blocks. In our approach, the designers can put custom logic into the existing framework by adding additional accelerator units (user peripherals). These accelerator units are used for any single or multiple frame video processing. The development phase would be limited to these custom logic components. A unique design flow that incorporates the HW/SW components is used (Fig. 1). This platform provides a rapid development of image and video processing algorithms because of a software-based approach. A development tool called Synphony C HLS tool from Synopsys [17] is used to convert C-based algorithms to hardware blocks that can easily be incorporated into the IVPP.

2 IVPP design

The board used for the IVPP is the Virtex-5 OpenSPARC evaluation platform developed by Xilinx. This board has a Xilinx Virtex-5 XC5VLX110T FPGA with 69 120 logic cells, 64 DSP48Es and 5328 Kb of block ram. It also has a 64-bit wide 256-MB DDR2 small outline DIMM. The board has an analogue-to-digital converter (ADC) AD9980 for video input. The ADC is an 8-bit, 95 MSPS, monolithic analogue interface optimised for capturing YPbPr video and RGB graphics signals. Its 95 MSPS encode rate capability and full power analogue bandwidth of 200 MHz support all the HDTV video modes and graphics resolutions up to XGA (1024 × 768 at 85 Hz). Moreover, the board has a digital-to-analogue converter (DAC) CH7301C for video output. It is a display controller device that accepts a digital graphics input signal, and encodes and transmits data through a digital visual interface (DVI). The device accepts data over one 12-bit wide variable voltage data port, which supports different data formats including RGB and YCrCb. It can support UXGA (1600 × 1200 at 60 Hz). This board is ideal as a video processing platform since it has all the hardware necessary to capture and display the data on a monitor. Nevertheless, the proposed design can be implemented on any FPGA as long as it is large enough. Video data are captured from a camera using the VGA input port at a resolution of 1024 × 768 at 60 Hz. Then, these video data are buffered in the DDR2 memory and displayed on the monitor through the DVI output port. With this design, we have built a flexible architecture that enables the user to perform real-time processing on a single frame or multiple frames. The overview of the design is given in Fig. 2. Multiple processing options are then possible, giving flexibility to the user:

- The user can choose to display the RGB video data without any processing.
- The user can perform real-time processing on a single frame of the RGB video data and display the output.
- The user can perform multi-frame processing and display the output.

Smoothing and edge detection filters are examples of frame processing. Motion detection and video compression are examples of multi-frame processing. The next section

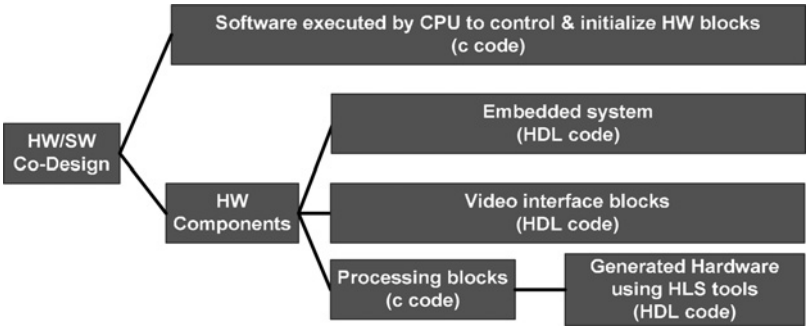


Fig. 1 *HW/SW IVPP design flow*

© The Institution of Engineering and Technology 2012

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

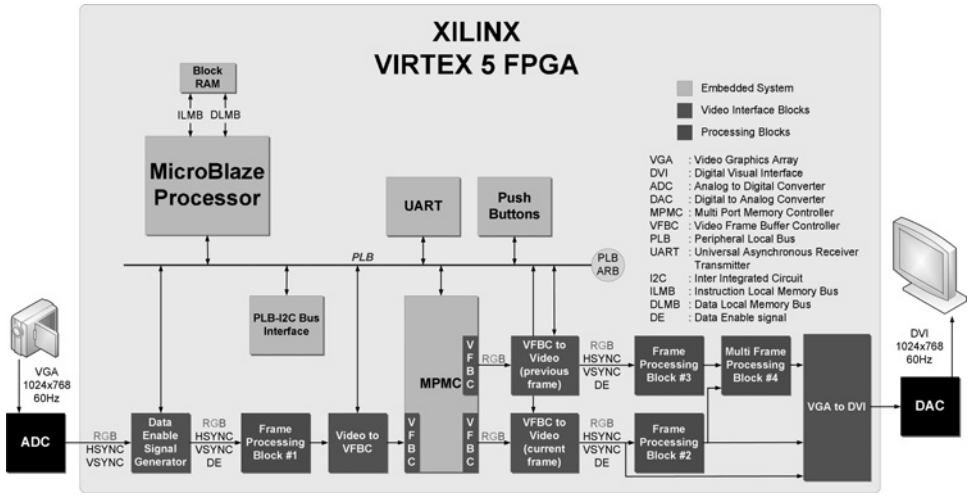


Fig. 2 *Platform design overview*

describes the constraints that need to be met by the platform and the video processing applications.

2.1 IVPP features

The IVPP must be adaptable so that any video input resolution can be used. Hardware blocks can be configured to support any resolution and they are all connected to the microprocessor. When the board is powered up and configured, a microprocessor initiates the configuration of the hardware blocks to support the resolution chosen by the user.

The IVPP must be easy to use. The users can easily plug-in custom processing blocks without knowing details regarding the platform architecture. Moreover, by using HLS tools such as Synphony C, a user does not need to know anything about the hardware language; its application can be designed in C and translated automatically into hardware blocks.

The applications must be designed so that real-time processing is achieved. A Synphony C compiler tool has been used to develop our applications. The software analyses the application C code and gives advices in order to improve it so that the frequency and area constraints are met. In our case, the pixel clock frequency is 65 MHz, hence each application must be able to run at that frequency in order to do real-time processing. Moreover, after a known latency, the algorithm must output a pixel for every clock cycle. The Synphony C compiler will try to achieve this frequency by optimising the datapath operations in the algorithms. If this is not feasible because of algorithm complexity, frame skipping can be used in order to relax the timing constraints.

The next section describes the different options available to communicate with the IVPP.

2.2 Communication with the IVPP

Multiple options are available to the user in order to control the platform:

(i) A universal asynchronous receiver transmitter can be used for direct communication between the computer and the FPGA through a serial link. A menu is displayed on the screen and the user can specify the type of video processing needed.

(ii) The IVPP can be controlled over the internet via an HTTP server. An HTTP server can be optionally implemented on the FPGAs for enabling remote access and control to the video processing platform. We have implemented a server function using a lightweight Internet protocol library [18] which is suitable for embedded devices such as a MicroBlaze processor on Xilinx FPGAs and implements the basic networking protocols such as IP, TCP, UDP, DHCP and ARP. This allows any user connected to the Internet to control the platform, choose the type of video processing to perform and see the results on its computer.

(iii) The IVPP can be controlled by MATLAB through a USB interface.

(iv) Push buttons can be used to select which processing to display on the screen.

The next section describes the video interface blocks.

2.3 Video interface and synthesis results

Video interface blocks are necessary for processing the video data coming from the ADC. We implemented several hardware modules in order to provide the necessary video interface. These hardware components form the basis of the proposed platform and they are shared by all the user applications. Pre-existing blocks include a data enabled (DE) signal generator, a video-to-video frame buffer controller (VFBC), VFBC, VFBC-to-video and VGA-to-DVI blocks, which are explained next.

All the video fields and line timing are embedded in the video stream (see Fig. 3). The purpose of the DE signal generator is to create a signal that will be high only during the active portion of the frame and low otherwise. The timing information is given to the block during the initialisation phase. The timing information for the resolution 1024 × 768 at 60 Hz is given in Table 1.

The visible RGB pixels are then extracted using the DE signal and written to the DDR2 memory using the VFBC [19]. The VFBC allows a user IP to read and write data in two-dimensional sets regardless of the size or the organisation of external memory transactions. It is a connection layer between the video clients and the multiple port memory controller (MPMC). It includes separate asynchronous FIFO interfaces for command input, write data input and read data output.

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2011.0156

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

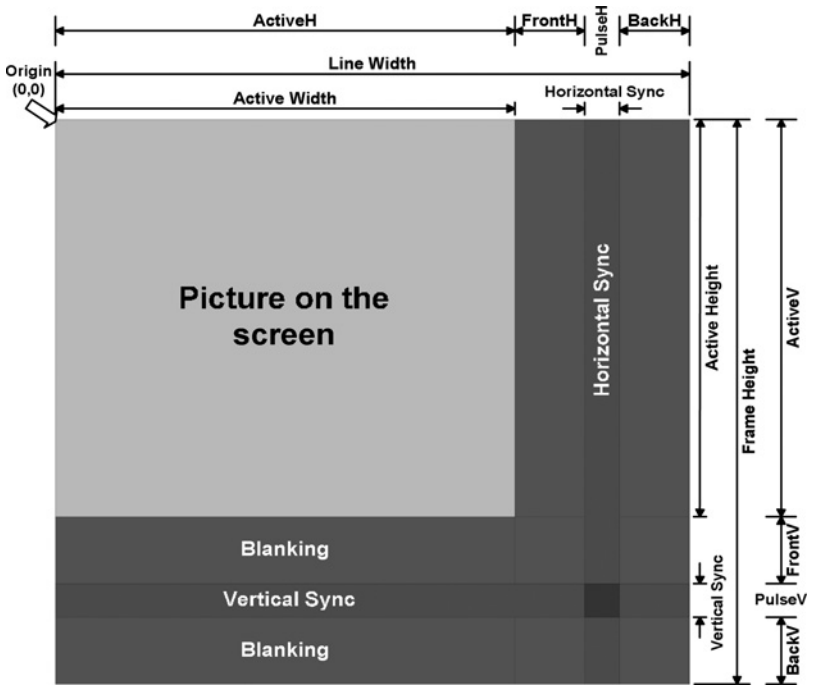


Fig. 3 *Video format*

Table 1 Timing information

Horizontal timing		Vertical timing	
Scanline part	Pixels	Frame part	Lines
visible area	1024	visible area	768
front porch	24	front porch	3
sync pulse	136	sync pulse	6
back porch	160	back porch	29
whole line	1344	whole frame	806

Table 2 Synthesis results of the data enabled generator

Resource type	Percentage of FPGA, %
slices registers	<1
slices look-up tables (LUTs)	<1

Table 3 Synthesis results of video to VFBC

Resource type	Percentage of FPGA, %
slices registers	1
slices LUTs	<1

The visible RGB pixels are then retrieved from the DDR2 memory using the VBFC to video block. Two frames are retrieved at the same time, so that multi-frame processing can be carried out. When 'Frame #i + 2' is being buffered, 'Frame #i' and 'Frame #i + 1' are retrieved. Finally, the data are sent to the DVI output port in the format supported by the DAC. The synthesis results of the individual modules and the overall system are given in Tables 2–7. The IVPP uses a few resources of the FPGA; hence space is available for additional logic such as image and video processing applications.

Table 4 Synthesis results of VFBC to video

Resource type	Percentage of FPGA, %
slices registers	1
slices LUTs	1

Table 5 Synthesis results of VGA to DVI

Resource type	Percentage of FPGA, %
slices registers	<1
slices LUTs	<1

Table 6 Synthesis results of MPMC

Resource type	Percentage of FPGA, %
slices registers	10
slices LUTs	9
block RAM/FIFO	11

Table 7 Synthesis results of the proposed platform supporting multi-frame processing

Resource type	Percentage of FPGA, %
slices registers	18
slices LUTs	16
block RAM/FIFO	11

3 HLS tools

SoC design is mainly accomplished by using RTL such as Verilog and VHDL. An algorithm can be converted to the RTL level by using the behavioural model description method or by using pre-defined IP core blocks. After completing this RTL code, a formal verification needs to be done, followed by a timing verification for proper operation.

© The Institution of Engineering and Technology 2012

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

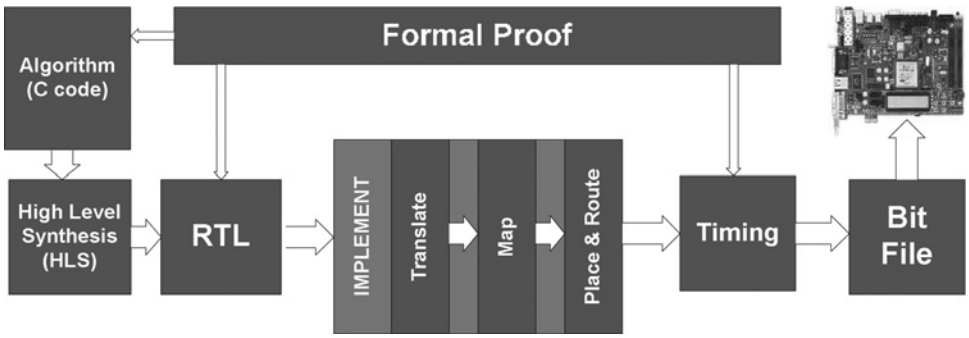


Fig. 4 *FPGA HLS flow*

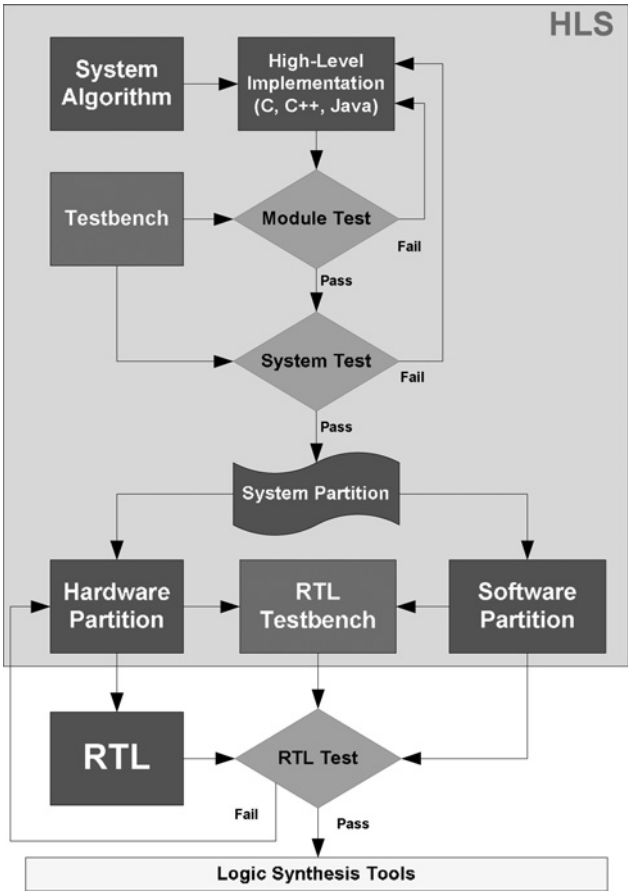


Fig. 5 *HLS design flow*

The RTL design abstracts logic structures, timing and registers [20]. Therefore every clock change causes a state change in the design. This timing dependency causes every event to be simulated. This results in a slower simulation time and longer verification period of the design. The design and verification of an algorithm in RTL can take 50–60% of the time-to-market (TTM). The RTL design becomes impractical for larger systems that have a high data flow between the blocks and require millions of gates. Even though using behavioural modelling and IP cores may improve design time, the difficulty in synthesis, poor performance results and rapid changes in the design make IP cores difficult to adapt and change. Therefore the systems rapidly become obsolete.

The limitations of RTL and longer TTM forced designers to think of the design as a whole system rather than blocks. In addition, software integration in SoC was always done after hardware was designed. When the system becomes more complex, integration of the software is desired during hardware implementation. Owing to improvements in SoC and shorter TTM over the last two decades, designers can use alternative methods to replace RTL. Extensive work has been done in electronics system level design. Hence, HW/SW co-design and HLS [21–24] are now integrated into FPGA and ASIC design flows. The integration of HLS into FPGA design flow is shown in Fig. 4.

RTL description of a system can be implemented from a behavioural description of the system in C. This will result in a faster verification process and shorter TTM. It is also possible to have a hybrid design where the RTL blocks can be integrated with HLS.

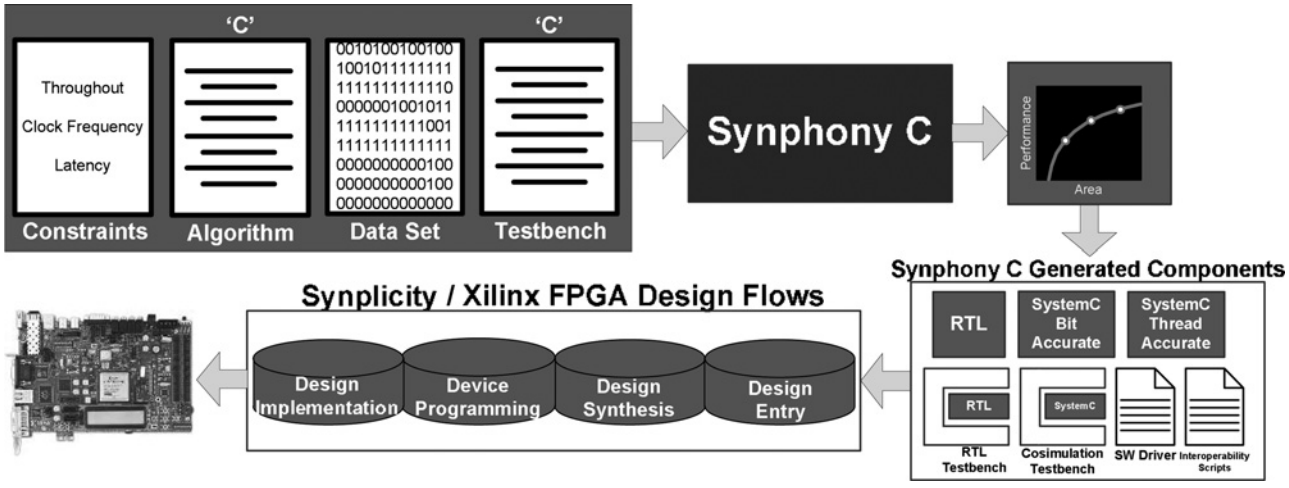


Fig. 6 *Synphony C-based design flow for hardware implementation*

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2011.0156

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

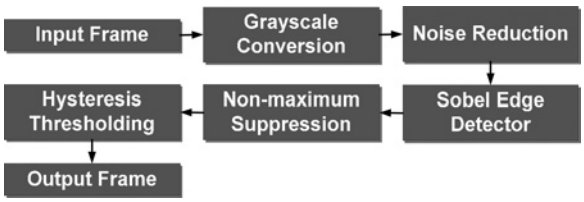


Fig. 7 *Canny edge detector algorithm*

The HLS design flow (see Fig. 5) shows that a group of algorithms (which represent the whole system or parts of a system) can be implemented using one of the high-level languages such as C, C++, Java, MATLAB and so on [20, 25]. Each part in the system can be tested independently before the whole system is tested. During this testing process, the RTL testbenches can also be generated. After testing is complete, the system can be partitioned into HW and SW. This enables the SW designers to join the design process during HW design; in addition, RTL can be tested using HW/SW together. After the verification process, the design can be implemented using FPGA synthesis tools.

Many HLS tools are available such as Xilinx's AutoESL, Synopsys's Synphony C compiler (also known as PICO) [18], Mentor Graphics' Catapult C and so on. An independent evaluation of HLS tools for Xilinx FPGAs has been carried out by Berkeley Design Technology [26]. It shows that using HLS tools with FPGAs can improve the video applications' performance significantly compared with conventional DSP processors. Moreover, this study shows that for a given application, the HSL tools will achieve similar results compared with the hand-written RTL code with a shorter development time.

The proposed IVPP uses a Synphony C Compiler from Synopsys [1, 27] to generate the RTL code for the processing blocks (Fig. 1). Synphony C takes a C-based description of an algorithm and generates a performance-driven device-dependent synthesisable RTL code, testbench files, application drivers, simulation scripts as well as SystemC-based transaction-level modelling (TLM) models. The Synphony C design flow is shown in Fig. 6.

4 Integration of HLS hardware blocks into the IVPP

With integration of the Synphony C compiler into the FPGA flow, the designers can create complex hardware [1] sub-systems from sequential untimed C algorithms. It allows the designers to explore programmability, performance, power, area and clock frequency. This is achieved by providing a

```
if(j > 0 & j < rows + 1 & i > 0 & i < cols + 1){
    input = pico_stream_input_yin();
    temp = (((input >> 2 * 8) & 0xFF) +
    ((input >> 1 * 8)& 0xFF) +
    ((input >> 0 * 8) & 0xFF)) * 85/256;}
else temp = 0;

if (j < rows + 1) {
    if (i == 0) {
        SmoothH[4] = SmoothH[3] = 0;
        SmoothH[2] = SmoothH[1] = 0;
      SmoothH[0] = 0;}
   else {
    for(k = 0; k < 4;k++)
        SmoothH[k] = SmoothH[k + 1];
    SmoothH[4] = i < cols + 1 ? temp : SmoothH[3];}
output1 = (SmoothH[0] + 4 * SmoothH[1] +
6 * SmoothH[2] + 4 * SmoothH[3] +
SmoothH[4]) >> 4;}

 if (j == 0) {
     SmoothV[4] = SmoothV[3] = 0;
    SmoothV[2] = SmoothV[1] = 0;
    SmoothV[0] = 0;}
else {
    data1 = SmoothLB[i];
   SmoothV[0] = data1.p1;
    SmoothV[1] = data1.p2;
    SmoothV[2] = data1.p3;
    SmoothV[3] = data1.p4;
    SmoothV[4] = j < rows + 1 ? output1 : SmoothV[3];}
data1.p1 = SmoothV[1]
data1.p2 = SmoothV[2]
data1.p3 = SmoothV[3]
data1.p4 = SmoothV[4]
SmoothLB[i] = data1;
output2 = (SmoothV[0] + 4 * SmoothV[1] +
6 * SmoothV[2] + 4 * SmoothV[3] +
SmoothV[4]) >> 4;
```

Get RGB and convert to BW

Apply horizontal filter

Apply vertical filter

Fig. 8 *HLS C code for noise reduction*

© The Institution of Engineering and Technology 2012

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

comprehensive and robust verification and validation environment. With these improvements to TTM [28], the production cost can be reduced.

The Synphony C compiler can explore different types of parallelism and will choose the optimal one. The results in terms of speed and area are given along with detailed reports that will help the user to optimise its code. When the achieved performance is satisfactory, the RTL code is generated and implemented in the targeted FPGA. Since the testing is done at the C level, RTL and testbench files are generated based on these inputs and testing and verification time can be drastically reduced [1]. When an application is created, a wrapper is generated by the Synphony C compiler to help instantiate a user block into the design. Each block is synchronised using the synchronisation signals VSYNC and DE. The Synphony C block needs to be enabled initially at the beginning of a frame. This is done easily by detecting the rising edge of VSYNC which indicates the beginning of a frame. Then, the 'data ready' input of the Synphony C block is connected to the DE signal. During blanking time (DE signal is not asserted), additional processing time is available for user video applications which is equivalent to vertical (3 + 29 lines) and horizontal (24 + 160 pixels) blanking (see timing information in Table 1). All the required signals such as HSYNC, VSYNC and DE are generated by the IVPP hardware base components; hence, the users can connect their custom HLS blocks to these interface signals.

The user needs to analyse the latency of the Synphony C block in order to synchronise HSYNC, VSYNC and DE with the output. For example, for a latency of ten clock cycles, those signals need to be delayed by ten. This is accomplished using sliding registers. Nevertheless, if the latency is greater than 128 clock cycles, FIFOs are used instead of registers in order to reduce the hardware usage. Canny edge detector and motion detector applications have an important latency (multiple lines), hence FIFOs have been used; the object tracking application has a latency of three clock cycles.

5 Case study using the Synphony C compiler for the IVPP

As a demonstration of the possibilities of the proposed platform, three video processing applications have been designed and developed using the Synphony C compiler. If other HLS tools are used, the design flow would be very similar. Canny edge detector, motion detector and object tracking blocks have been tested with the IVPP. We have three different applications with different integrations into the IVPP. From Fig. 2, we can see four possible positions for processing. Processing block #1 can be used for stream processing before the RGB pixels are written to the memory. Processing block #2 is used for reading the current frame from memory and processing it. Processing block #3 is used for reading the previous frame from memory and processing it. Finally, Processing block #4 is used for multi-frame processing where current and previous frames are handled simultaneously.

The canny edge detector will be at position #2 in the IVPP, the motion detector will be at processing blocks #2, #3 and #4 and the object tracking at processing blocks #1 and #2. The output images are real-time video results of the different hardware components generated by the Synphony C compiler.

The C algorithms have been modified in order to achieve optimal results compared with the hand-written RTL. The structure of the algorithms is very similar to the RTL design. Each algorithm has a stream of pixels as input instead of a matrix.

5.1 Canny edge detector

The algorithm is shown in Fig. 7 [29].

First, the RGB data are converted to grey scale. Then noise reduction is performed by applying a 5 × 5 separable filter (see Fig. 8)

$$S = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{1}{16}\begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

The Sobel operator [30] is a discrete differentiation operator that computes an approximation of the gradient of the image intensity function. It is based on convolving the image with a small, separable and integer valued filter in horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. Basically, 3 × 3 filters (F_1 and F_2) are applied to the video data (see Fig. 9). We obtain horizontal (O_1) and vertical (O_2) gradients.

The magnitude and direction for each gradient are obtained as follows:

$$G_{x,y} = |O_1(x,\ y)| + |O_2(x,\ y)|$$

(see equations at the bottom of the page)
where $D_{x,y} = 0$ corresponds to a direction of 0°; $D_{x,y} = 1$

$$D_{x,y} = 0 \quad \text{if} \begin{cases} |O_1| > 2 \times |O_2| \wedge (O_1 > 0 \wedge O_2 \geq 0 \vee O_1 < 0 \wedge O_2 \leq 0) \\ |O_2| > 2 \times |O_1| \wedge (O_1 \geq 0 \wedge O_2 < 0 \vee O_1 \leq 0 \wedge O_2 > 0) \end{cases}$$

$$D_{x,y} = 1 \quad \text{if} \begin{cases} 2 \times |O_2| \geq |O_1| > |O_2| \wedge (O_1 > 0 \wedge O_2 \geq 0 \vee O_1 < 0 \wedge O_2 \leq 0) \\ 2 \times |O_1| \geq |O_2| \geq |O_1| \wedge (O_1 > 0 \wedge O_2 > 0 \vee O_1 < 0 \wedge O_2 < 0) \end{cases}$$

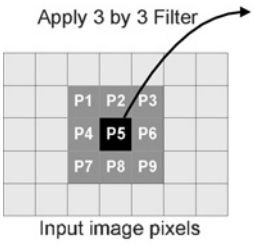
$$D_{x,y} = 2 \quad \text{if} \begin{cases} |O_2| > 2 \times |O_1| \wedge (O_1 > 0 \wedge O_2 > 0 \vee O_1 < 0 \wedge O_2 < 0) \\ |O_2| > 2 \times |O_1| \wedge (O_1 \geq 0 \wedge O_2 < 0 \vee O_1 \leq 0 \wedge O_2 > 0) \end{cases}$$

$$D_{x,y} = 3 \quad \text{if} \begin{cases} 2 \times |O_1| \geq |O_2| > |O_1| \wedge (O_1 \geq 0 \wedge O_2 < 0 \vee O_1 \leq 0 \wedge O_2 > 0) \\ 2 \times |O_1| \geq |O_2| \wedge (O_1 > 0 \wedge O_2 < 0 \vee O_1 < 0 \wedge O_2 > 0) \end{cases}$$

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2011.0156

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.



$$F_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -2 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$F_2 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

$$O_1 = p_3 + 2 \times p_6 + p_9 - p_1 - 2 \times p_4 - p_7$$

$$O_2 = p_3 + 2 \times p_2 + p_4 - p_7 - 2 \times p_8 - p_9$$

Fig. 9 *Sobel edge filters*

corresponds to a direction of 45°; $D_{x,y} = 2$ corresponds to a direction of 90°; and $D_{x,y} = 3$ corresponds to a direction of 135°.

The HLS C code for the Sobel edge filter and gradient is shown in Fig. 10.

Non-maximum suppression is then carried out by comparing $G_{x,y}$ with the magnitude of its neighbours along the direction of the gradient $D_{x,y}$. This is done by applying a 3×3 moving window. For example, if $D_{x,y} = 0$, the pixel is considered as an edge if $G_{x,y} > G_a$ and $G_{x,y} > G_b$

```
if (i == 0) {
    SobelH[2] = output2;
   SobelH[1] = 0;
   SobelH[0] = 0;}
else {
    for(k = 0; k < 2; k++)
        SobelH[k] = SobelH[k + 1];
                  SobelH[2] = output2;}
output3 = SobelH[2] - SobelH[0];
output4 = SobelH[0] + 2 * SobelH[1] +
SobelH[2];
if (i > 1) {
    if (j == 0) {
        SobelV1[2] = output3;
        SobelV1[1] = 0;
        SobelV1[0] = 0;
        SobelV2[2] = output4;
        SobelV2[1] = 0;
        SobelV2[3] = 0;}
    else {
        data01 = SobelLB1[i - 2];
        SobelV1[0] = data01.p1;
        SobelV1[1] = data01.p2;
        SobelV1[2] = output3;
        data02 = SobelLB2[i - 2];
        SobelV2[0] = data02.p1;
        SobelV2[1] = data02.p2;
        SobelV2[2] = output4; }
    data01.p1 = SobelV1[1];
    data01.p2 = SobelV1[2]
    SobelLB1[i - 2] = data01;
    data02.p1 = SobelV2[1];
    data02.p2 = SobelV2[2]
    SobelLB2[i - 2] = data02;
O1 = SobelV1[0] + 2 * SobelV1[1] +
 SobelV1[2];
O2 = SobelV2[0] - SobelV2[2];
aO1 = ABS(O1);
aO2 = ABS(O2);
G = aO1 + aO2;
if(aO1 > aO2 &
((O1 > 0 & O2 >= 0) | (O1 < 0 & O2 <= 0))){
    if(aO1 > 2 * aO2) D = 0;
    else D = 1;}
else if(aO1 ≤ aO2 &
((O1 > 0 & O2 > 0) | (O1 < 0 & O2 < 0))){
    if(aO2 > 2 * aO1) D = 2;
    else D = 1;}
else if(aO1 < aO2 &
((O1 >= 0 & O2 < 0) | (O1 <= 0 & O2 > 0))){
    if(aO2 > 2 * aO1) D = 2;
    else D = 3;}
else{
    if(aO2 > 2 * aO1) D = 0;
    else D = 3;}
GD = (G ≪ 2) | D;}
```

Apply horizontal filter

Apply vertical filter

Determine gradient direction

Fig. 10 *HLS C code for sobel edge detector and gradient direction*

© The Institution of Engineering and Technology 2012

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

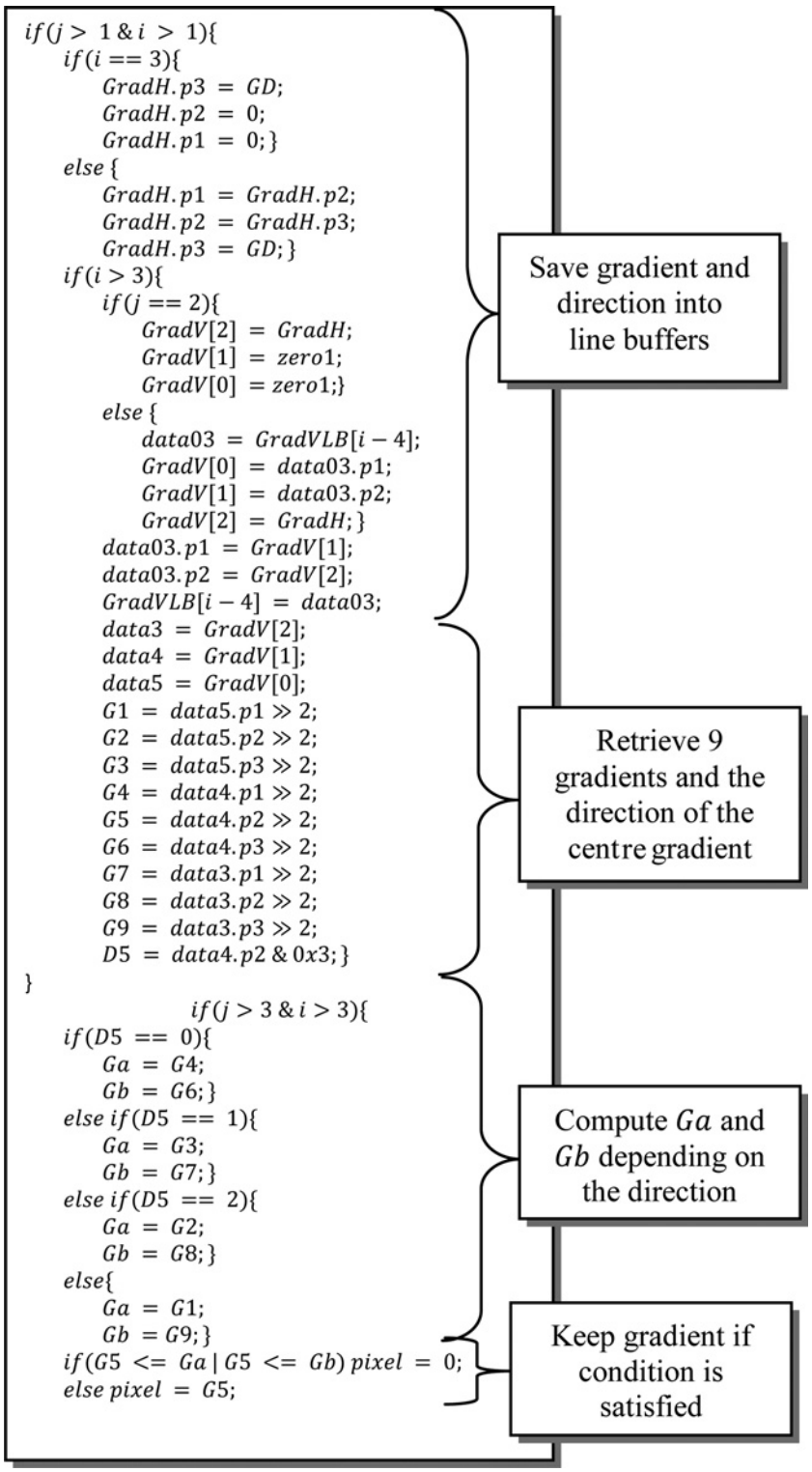


Fig. 11 *HLS C code for non-maximum suppression*

with $G_a = G_{x,y-1}$ and $G_b = G_{x,y+1}$. the HLS C code is shown in Fig. 11.

Then, the result is usually thresholded to decide which edges are significant. Two thresholds T_H and T_L are applied, where $T_H > T_L$. If the gradient magnitude is greater than T_H then that pixel is considered as a definite edge. If the gradient is less than T_L, then, that pixel is set to zero. If the gradient magnitude is between these two, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above T_L. A 3×3 moving window operator is used, the centre pixel is said to be connected if at least one neighbouring pixel value is greater than T_L, and the resultant is an image with sharp edges. The HLS C code for hysteresis thresholding is given in Fig. 12. The Synphony C block has been placed at position #2 in the platform.

Table 8 shows the synthesis results of the canny edge detector.

5.2 Motion detector

A motion detection algorithm (see Fig. 13) has been implemented using the Synphony C compiler: Both the

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2011.0156

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

```
if(i == 4){
    PixH.p3 = pixel;
    PixH.p2 = 0;
    PixH.p1 = 0;}
else {
    PixH.p1 = PixH.p2;
    PixH.p2 = PixH.p3;
    PixH.p3 = pixel;}
if(i > 5){
    if(j == 4){
        PixV[2] = PixH;
        PixV[1] = zero2;
        PixV[0] = zero2;}
    else {
        data07 = PixVLB[i − 6];
        PixV[0] = data07.p1;
        PixV[1] = data07.p2;
        PixV[2] = PixH;}
    data07.p1 = PixV[1];
    data07.p2 = PixV[2];
    PixVLB[i − 6] = data07;
    data7 = PixV[2];
    data8 = PixV[1];
    data9 = PixV[0];
    p1 = data9.p1;
    p2 = data9.p2;
    p3 = data9.p3;
    p4 = data8.p1;
    p5 = data8.p2;
    p6 = data8.p3;
    p7 = data7.p1;
    p8 = data7.p2;
    p9 = data7.p3;
    if(p5 < TL) dataout = 0;
    else if(p5 > TH | p1 > TL |
    p2 > TL | p3 > TL |
    p4 > TL | p6 > TL |
    p7 > TL | p8 > TL | p9 > TL){
        dataout = 255;}
    else dataout = 0;}
}
if(i > 5 & i < cols + 6 &
j > 5 & j < rows + 6)
 pico_stream_output_yout(dataout);}
```

Save gradient into line buffers

Retrieve 9 gradients

Keep gradient if condition is satisfied

Output data as a stream

Fig. 12 *HLS C code for hysteresis thresholding*

current and the preceding frame are converted to black and white. Then, a 5×5 noise reduction filter is applied. Finally, a Sobel edge detector is applied on the difference of the two images and the motion is superimposed on the current frame. The Synphony C block has been placed at processing blocks #2, #3 and #4 in the platform. Table 9 shows the synthesis results of the motion detector.

5.3 Object tracking

An object tracking algorithm has been developed and tested on the platform. It is composed of two main phases:

(i) At processing block #1 in the IVPP, noise reduction is performed on the RGB values, then RGB to hue, saturation and value (HSV) conversion is done and colour segmentation is applied.
(ii) Then, at processing block #2 in the IVPP, a boundary box is created around the pixels from a specific colour selected by

Table 8 Synthesis results of the canny edge detector

Resource type	Usage	Percentage of FPGA, %
slices registers	1028	1
slices LUTs	1388	2
block RAM/FIFO	11	7
DSP48Es	1	1

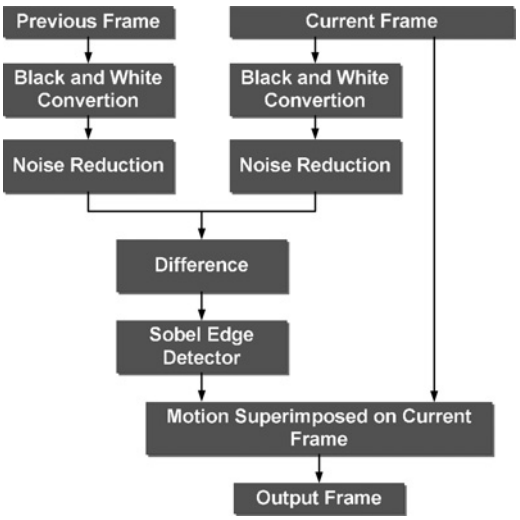


Fig. 13 *Motion detection algorithm*

Table 9 Synthesis results of the motion detector

Resource type	Usage	Percentage of FPGA, %
slices registers	704	1
slices LUTs	1281	1
block RAM/FIFO	9	6
DSP48Es	2	3

```
dRG = Red − Green;
dGB = Green − Blue;
dBR = Blue − Red;
min_t = MIN(Red, Green);
max_t = MAX(Red, Green);
mini = MIN(Blue, min_t);
maxi = MAX(Blue, max_t);
diff = maxi − mini;
div_m = 42 * div_tab[diff];
ddRG = div_m * dRG;
ddGB = div_m * dGB;
ddBR = div_m * dBR;
if (maxi == mini) H = 0;
else if(maxi == Red) H = (ddGB ≫ 16) + 42;
else if(maxi == Green) H = (ddBR ≫ 16) + 127;
else H = (ddRG ≫ 16) + 213;
if (maxi == 0) S = 0;
else S = (255 * diff * div_tab[maxi]) ≫ 16;
V = maxi;
```

Fig. 14 *HLS C code for RGB to HSV conversion*

```
unsigned int xmin_temp = cols − 1;
unsigned int xmax_temp = 0;
unsigned int ymin_temp = rows − 1;
unsigned int ymax_temp = 0;
temp = H − Hin;
if(S > 63 &V > 127){
   if (ABS(temp) < Threshold){
      if (i < xmin_temp) xmin_temp = i;
      if (i > xmax_temp) xmax_temp = i;
      if (j < ymin_temp) ymin_temp = j;
      if (j > ymax_temp) ymax_temp = j;}
}
```

Fig. 15 *HLS C code for boundary information*

© The Institution of Engineering and Technology 2012

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

```
if (j ≥ ymini & j ≤ ymini + 2 & i >= xmini & i <= xmaxi) out = 0xFF0000;
else if (j > ymini + 2 & j < ymaxi − 2 &((i ≥ xmini & i ≤ xmini + 2)|
(i ≥ xmaxi − 2 & i ≤ xmaxi))) out = 0xFF0000;
else if (j ≥ ymaxi − 2 & j ≤ ymaxi & i >= xmini & i <= xmaxi) out = 0xFF0000;
else out = (red ≪ 2 ∗ 8)|(green ≪ 1 ∗ 8)| (blue << 0 ∗ 8) & 0xFFFFFF;
```

Fig. 16 *HLS C code for display of object boundaries*

the user. The boundary box can give information on the orientation of the object and also the distance between the object and the camera. Moreover, the algorithm keeps track of all the positions of the object during 1 s and displays them on the screen.

Noise reduction is performed on the input data by applying the same Gaussian filter as seen above. Then RGB to HSV conversion is done as follows:

$$\text{MAX} = \max(\text{Red},\ \text{Green},\ \text{Blue})$$
$$\text{MIN} = \min(\text{Red},\ \text{Green},\ \text{Blue})$$

$$H = \begin{cases} 0 & \text{if MAX} = \text{MIN} \\ 42\dfrac{\text{Green} - \text{Blue}}{\text{MAX} - \text{MIN}} + 42 & \text{if MAX} = \text{Red} \\ 42\dfrac{\text{Blue} - \text{Red}}{\text{MAX} - \text{MIN}} + 127 & \text{if MAX} = \text{Green} \\ 42\dfrac{\text{Red} - \text{Green}}{\text{MAX} - \text{MIN}} + 213 & \text{if MAX} = \text{Blue} \end{cases}$$

$$S = \begin{cases} 0 & \text{if MAX} = 0 \\ 255\dfrac{\text{MAX} - \text{MIN}}{\text{MAX}} & \text{Otherwise} \end{cases}$$

$$V = \text{MAX}$$

Fig. 14 shows the HLS C code of the RGB to HSV conversion. *div_tab* is an array of 256 values (*div_tab*[0] is set to 0 since it does not affect the results) which stores the division results of $1/i$ for $i = 1, \ldots, 255$ with a precision of 16 bits.

We compare the HSV values of the pixel with an input value (H_{in}) chosen by the user. If H is close enough to that input value and S and V are big enough, then the pixel will be tracked. The boundary information is obtained depending on the colour (H_{in}) selected by the user (see Fig. 15).

At processing block #2, we receive the information from block #1 for the boundary box. A red box is displayed around the object selected by the user (see Fig. 16).The centre of the box is saved for a duration of 1 s before being erased. Hence, the movement of the object can be tracked (see Fig. 17). An example of real-time video output of this

```
if(j == 0 & i == 0){
x_center_mem[write_ct[0]] = (xmini + xmaxi)/2;
y_center_mem[write_ct[0]] = (ymini + ymaxi)/2;
write_ct[0] + +;}
else if(j > 0 & i > 0){
for(k = 0;k < 64;k + +){
if(y_center_mem[k] == j | y_center_mem[k] == j − 1){
if(x_center_mem[k] == i | x_center_mem[k] == i − 1)
out = 0xFFFFFF;}}}
```

Fig. 17 *HLS C code for display of positions of object*

Fig. 18 *Real-time video output shows a green pen framed by a rectangular box*

Trace positions are also displayed which follow the centre of the box for a 1 s duration

Table 10 Synthesis results of object tracking

Resource type	Usage	Percentage of FPGA, %
slices registers	2631	4
slices LUTs	4892	7
block RAM/FIFO	3	2
DSP48Es	9	14

algorithm is given in Fig. 18. Table 10 shows the synthesis results of object tracking.

6 Conclusion

In this work, we have developed an IVPP for real-time applications on a Virtex-5 FPGA. A new C-based HLS design flow is presented. The user can design the image and video processing applications in C language, convert them into hardware using the Synphony C compiler tool and then implement and test them easily using the IVPP. The IVPP streamlines the development by providing all the necessary logic blocks for the front-end (capturing video data) and the back-end (displaying processed output data) operations. For a case study, three example applications have been discussed, showing the performance and flexibility of the proposed platform. The IVPP can be a cost-effective, rapid development and prototyping platform for key applications such as video encoding/decoding, surveillance, detection and recognition.

7 Acknowledgment

The authors would like to thank Xilinx, Inc. (www.xilinx.com) and Synfora, Inc. (www.synfora.com) for their valuable support.

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2011.0156

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.

8 References

1 Coussy, P., Morawiec, A.: 'High-level synthesis: from algorithm to digital circuits' (Springer Science + Business Media, Berlin, 2008), Ch. 1, 4
2 Muller, W., Rosenstiel, W., Ruf, J.: 'SystemC: methodologies and applications' (Kluwer Academic Publishing, Dordrecht, 2003), Ch. 2
3 Hong, S., Yoo, S., Lee, S., *et al.*: 'Creation and utilization of a virtual platform for embedded software optimization: an industrial case study'. Proc. Fourth Int. Conf. Hardware/Software Codesign and System Synthesis, 2006, pp. 235–240
4 Ruggiero, M., Bertozzi, D., Benini, L., Milano, M., Andrei, A.: 'Reducing the abstraction and optimality gaps in the allocation and scheduling for variable voltage/frequency MPSoC platforms', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2009, **28**, (3), pp. 378–391
5 Tumeo, A., Branca, M., Camerini, L., *et al.*: 'Prototyping pipelined applications on a heterogeneous FPGA multiprocessor virtual platform'. Design Automation Conf., 2009, pp. 317–322
6 Skey, K., Atwood, J.: 'Virtual radios – hardware/software co-design techniques to reduce schedule in waveform development and porting'. IEEE Military Communications Conf., 2008, pp. 1–5
7 Yi-Li, L., Chung-Ping, Y., Su, A.: 'Versatile PC/FPGA-based verification/fast prototyping platform with multimedia applications', *IEEE Trans. Instrum. Meas.*, 2007, **56**, (6), pp. 2425–2434
8 Schumacher, P., Mattavelli, M., Chirila-Rus, A., Turney, R.: 'A software/hardware platform for rapid prototyping of video and multimedia designs'. Proc. Fifth Int. Workshop on System-on-Chip for Real-Time Applications, 2005, pp. 30–34
9 Zhang, X., Rabah, H., Weber, S.: 'Auto-adaptive reconfigurable architecture for scalable multimedia applications'. Second NASA/ESA Conf. on Adaptive Hardware and Systems, 2007, pp. 139–145
10 Zhang, X., Rabah, H., Weber, S.: 'Cluster-based hybrid reconfigurable architecture for autoadaptive SoC'. 14th IEEE Int. Conf. on Electronics, Circuits and Systems, ICECS, 2007, pp. 979–982
11 Atitallah, A., Kadionik, P., Masmoudi, N., Levi, H.: 'HW/SW FPGA architecture for a flexible motion estimation'. IEEE Int. Conf. on Electronics, Circuits and Systems, 2007, pp. 30–33
12 Farrugia, N., Mamalet, F., Roux, S., Yang, F., Paindavoine, M.: 'Design of a real-time face detection parallel architecture using high-level synthesis', *EURASIP J. Embed. Syst.*, 2008, **938256**, pp. 1–9
13 Haynes, S., Epsom, H., Cooper, R., McAlpine, P.: 'UltraSONIC: a reconfigurable architecture for video image processing' (Springer, Berlin, 2002, *LNCS*), pp. 25–45
14 Sedcole, N.P., Cheung, P.Y.K., Constantinides, G.A., Luk, W.: 'A reconfigurable platform for real-time embedded video image processing' (Springer, Berlin, 2003, *LNCS*, **2778**), pp. 606–615
15 Altera,Video and Image Processing Suite, User Guide, 2012, available at http://www.altera.com/literature/ug/ug_vip.pdf#performance_performance
16 Xilinx LogiCORE IP Video Timing Controller, Product Guide, 2012, available at http://www.xilinx.com/support/documentation/ip_documentation/v_tc/v4_00_a/pg016_v_tc.pdf
17 Synopsys, Inc., Synphony High-Level Synthesis from Language and Model Based Design, available at http://www.synopsys.com/Systems/BlockDesign/HLS/Pages/default.aspx
18 Xilinx Lightweight IP (lwIP) application examples, 2011, available at http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf
19 Xilinx LogiCore Video Frame Buffer Controller v1.0, XMP013, October 2007, available at http://www.xilinx.com/products/devboards/reference_design/vsk_s3/vfbc_xmp013.pdf
20 Ramachandran, S.: 'Digital VLSI system design' (Springer, New York, 2007), Chapter 11
21 Hammami, O., Wang, Z., Fresse, V., Houzet, D.: 'A case study: quantitative evaluation of C-based high-level synthesis systems', *EURASIP J. Embed. Syst.*, 2008, **685128**, pp. 1–13
22 Glasser, M.: 'Open verification methodology cookbook' (Springer, New York, 2009), Chapters 1–3
23 Man, K.L.: 'An overview of SystemCFL', *Research in Microelectron. Electron.*, 2005, **1**, pp. 145–148
24 Hatami, N., Ghofrani, A., Prinetto, P., Navabi, Z.: 'TLM 2.0 simple sockets synthesis to RTL'. Int. Conf. on Design & Technology of Integrated Systems in Nanoscale Era, 2000, vol. 1, pp. 232–235
25 Chen, W.: 'The VLSI Handbook' (CRC Press LCC, Boca Raton, 2007, 2nd edn.), Chapter 86
26 Berkeley Design Technology, 'An independent evaluation of high-level synthesis tools for Xilinx FPGAs', available at http://www.bdti.com/MyBDTI/pubs/Xilinx_hlstcp.pdf
27 Haastregt, S.V., Kienhuis, B.: 'Automated synthesis of streaming C applications to process networks in hardware', *Design Automation & Test in Europe*, 2009, pp. 890–893
28 Avss, P., Prasant, S., Jain, R.: 'Virtual prototyping increases productivity – a case study'. IEEE Int. Symp. on VLSI Design, Automation and Test, 2009, pp. 96–101
29 He, W., Yuan, K.: 'An improved canny edge detector and its realization on FPGA'. Proc. Seventh World Congress on Intelligent Control and Automation, 2008
30 Gonzales, R.C., Woods, R.E.: 'Digital image processing' (Prentice-Hall, New Jersey, 2007, 3rd edn.)

© The Institution of Engineering and Technology 2012

Authorized licensed use limited to: Illinois Institute of Technology. Downloaded on October 02,2020 at 06:13:26 UTC from IEEE Xplore. Restrictions apply.