Acceleration of image processing algorithms based on a Single Board Computer and FPGA co-design

Petros Kokotis
Dpt. of Computer, Informatics and Telecommunications Engineering
International Hellenic University
Serres, Greece
petroskokotis@fpga.gr

John Vourvoulakis
Dpt. of Computer, Informatics and Telecommunications Engineering
International Hellenic University
Serres, Greece
jvourv@ihu.gr

Abstract*— During recent years, various hardware platforms were developed, each one suitable for use in different kind of applications. Platforms based on FPGAs, DSPs, GPUs, Single Board Computers, microcontrollers extend processing capabilities and functionality in comparison with traditional personal computers based on a single CPU. Furthermore, co-design combines advantages from different types of processing units, rendering such architectures more attractive to researchers. In this paper, we achieve acceleration of image processing algorithms using a hardware platform based on a Raspberry Pi Single Board Computer and a custom designed FPGA HAT (Hardware Attached on Top) for RPi. The FPGA HAT consists of a Cyclone 10LP device. The FPGA undertakes a computationally demanding load, such as robotic vision algorithms exploiting parallelism, while the RPi can apply higher level operations such as running ROS (Robot Operating System). In order to overcome bottleneck in exchanging data between RPi and FPGA, a 16-bit parallel customized protocol was developed from scratch. The achieved transfer rate was about 50 Mbytes/sec when multi threaded software was implemented for the RPi. An image edge detector was implemented in order to verify the system performance. When only the RPi was used, the processing rate was 48fps for images with resolution 512x512 pixels. RPi and FPGA co-design achieved processing rate 170fps for the same resolution images, which means an acceleration of about 350%. The proposed system was also evaluated in terms of power consumption.

Keywords—fpga, accelerator, raspberry pi

I. INTRODUCTION

In many embedded systems, the need for high computing power is imperative. In some UAVs, AUVs and ROVs, the requirements demand the use of a personal computer running an operating system. In the late 90's, a common way to include a personal computer in a device was to place a motherboard with its processor and plug all the necessary cards in the motherboard [1] in order to achieve the desired functionality. In recent years, SBCs (Single Board Computers) are mainly used whenever embedded systems require such computing capabilities. The potential of using SBCs became more attractive with the advent of the Raspberry Pi when a group of students at University of Cambridge started prototyping a small single board computer in 2006 and released it as a product in 2012. Raspberry Pi is now widely used in robotic platforms for commercial [2], educational [3], even for space [4] applications. On the other hand, there is immense literature in which researchers use specialized hardware to increase execution speed when a high data volume must be processed. Applications based on GPUs [5], FPGAs [5], co-designs [6] take advantage of parallelism in hardware and accelerate computer vision applications.

In this paper, we accelerate image processing algorithms on a custom hardware platform which is based on Raspberry Pi and FPGA. The FPGA board constitutes a Hardware Attached on Top (HAT) of the Raspberry Pi board. It consists of a Cyclone 10LP, configuration memory and the required support circuits. The FPGA HAT is connected to the expansion connector of the Raspberry Pi. It is used as a co-processor for the overall system. It takes over heavy processing load, relieving the RPi's ARM processor for other operations. The Raspberry Pi captures images and sends frames to the FPGA HAT. The FPGA processes the incoming frames according to the required task and sends back the results. By implementing a Prewitt edge detector in the FPGA, we achieved to execute the algorithm more than three and a half times faster in comparison with RPi's standalone operation. Furthermore, a 16-bit fast parallel communications interface has been implemented from scratch for data exchange between Raspberry Pi and FPGA through their GPIOs, achieving high data rates. The interface follows an asynchronous handshaking logic. The communication interface contributed to minimize bottleneck in communication between the two processing units.

The proposed work targets robotic vision applications in which the Raspberry Pi runs ROS (Robot Operating System) and performs higher level operations as well as image acquisition. Generally, image processing algorithms are computationally demanding and may require FFTs, convolutions, filtering, etc. Single Board Computers usually fail to operate satisfactorily due to the low computing power of its main processor and to the non real-time nature of their operating system. When real time operation is required, which is quite common in robotic vision applications, special hardware is adopted to act as a co-processor.

Following from the above considerations, the contribution of this paper is summarized as follows.

- We propose a hardware architecture based on a Raspberry Pi and an FPGA HAT. The architecture targets robotic vision applications. Schematic and PCB designs were implemented from scratch.
- We accelerate execution of image processing algorithms achieving about 350% speed up for a Prewitt Edge Detector. FPGA takes over the computational demanding tasks, while Raspberry Pi

can execute higher level operations such as running running ROS over embedded Linux.

- We implemented a custom fast 16-bit parallel interface between Raspberry Pi and FPGA, which provides communication speed approximately 44.5MBytes/sec.

The rest of the paper is organized as follows. Section II presents a brief literature survey. Section III describes the details of the implemented hardware/software co-design. In Section IV, the evaluation of the architecture is provided. Section V concludes the paper.

II. Literature Survey

The use of an FPGA as a co-processor/accelerator for the Raspberry Pi is not a new concept. It has been tried multiple times since the first Raspberry Pi came out in 2012. However, most of those implementations have been rendered outdated or lack the speed needed to justify their use as a co-processor. The proposed co-design is fast, expandable and cost efficient. Below, we present briefly related works found in the literature.

In [7], the DE0-Nano Cyclone IV development board was used to capture data from the onboard accelerometer and send the data to a Raspberry Pi 3 via an 8 bit parallel interface. A python script receives and passes accelerometer data to a ROS instance to simulate a humanoid robot fall.

In [8], an image processor on a Basys-2 FPGA development board was used to apply various image filters e.g. convert to B/W or grayscale, flip or rotate a predefined image icon and view the result on a computer monitor through the VGA output of the Basys board. A python script running on the Raspberry Pi transmits a 4 bit code to the FPGA and the corresponding filter is applied.

Similarly, in [9], a web server is running on the Raspberry Pi, where the user can select the filter to be applied using a web interface. Selection is then transmitted from the Raspberry Pi to the image processor on the FPGA. Afterwards, filtering is applied and the results are displayed to a monitor. The input image is preloaded to the FPGA memory as a ".coe" file.

In [10], a modular architecture using FPGA and/or microcontroller boards for Raspberry Pi is presented. Each module can be connected to the platform in a cascading manner. Communication between Raspberry Pi and FPGA/microcontroller is performed using the I^2C bus which, however, induces a bottleneck in case of large data volumes.

In the proposed architecture, Raspberry Pi is the main controller and the FGPA accelerator card acts as an image co-processor. Filters to be applied are hardcoded on the Cyclone 10LP. All the processed frames are returned back to the Raspberry Pi for further processing or to be viewed using the HDMI output port.

III. Hardware/Software co-design

A. Hardware Implementation

The FPGA accelerator HAT (Fig. 1) was built around Intel's Cyclone 10 LP (10CL025YE144) which consists of 25k logic elements (LEs) and 88 GPIOs. The Cyclone 10 LP requires 3 different power supply sources to operate properly. Core requires 1.2V, analog blocks (e.g. PLL) need 2.5V, and supply for I/O pins can reach up to 3.3V. A pair of dual output LTC3419 step down regulators was used to provide the required voltage levels to the FPGA chip. Voltage level on one of the outputs of the buck converter can be selected by the user using jumpers, setting the I/O voltage to banks 7 and 8 of the Cyclone FPGA to either 1.8V, 2.5V or 3.3V.

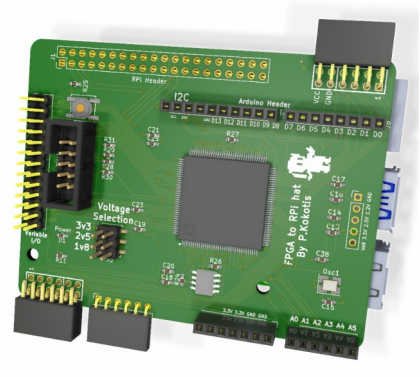

Fig. 1. The FPGA HAT.

All but one of 76 available I/O pins of the Cyclone 10LP are connected to an Arduino UNO R3 expansion header, 2 PMOD expansion ports and a 2x13 pin header. There is also a momentary switch connected to the last remaining I/O of the FPGA, primarily used as a software reset button. The Cyclone 10LP and Raspberry Pi share access to the Arduino Uno header pins but only one device can access it at a time. If no arduino shield is used then those shared lines can act as a bus between the FPGA and the Raspberry Pi.

Moreover, a 50 MHz crystal oscillator was used as clock source for the Cyclone 10LP to function.

In order to configure the FPGA, Intel EPCQ4 serial configuration memory is included in the board. Cyclone 10LP can be programmed either by using the physical onboard JTAG port or via the Raspberry Pi without the need of a JTAG programmer.

The FPGA accelerator board was designed as a 4 layer layout with 2 signal planes, one power and one ground plane. The exterior dimensions of the PCB are 85x64 mm.

B. Serial Connectivity

There is a direct JTAG connection between Raspberry Pi and the FPGA accelerator card. Using software like OpenOCD [11] the configuration can be sent from the Raspberry Pi to the FPGA or saved to onboard configuration memory without the need of cables and external programmer.

The Raspberry Pi is connected to the FPGA using various interfaces. The Raspberry Pi's UART (serial pins) are connected to the Arduino Uno header RX-TX pins. Those pins are also connected to the FPGA. When the UART is not used for an externally connected device, it can be used for communication between FPGA and RPi. The same also applies to the I^2C lines. Two FPGA pins are connected to the SCL, SDA lines of the Rpi, and it can act as a secondary I^2C device.

A third serial interface is supported. SPI communication interface between Raspberry Pi and Cyclone 10LP can be achieved but only in software mode from the Raspberry Pi's side. The RPi's hardware SPI lines are reserved for JTAG communication.

C. 16-bit parallel connectivity using GPIO

A 16-bit parallel protocol was implemented as high rate of data exchange between the Raspberry Pi and the FPGA

accelerator HAT was needed. To implement this protocol, all of the shared Arduino Uno header lines were utilized: 16 for data, 1 as write enable line, 1 as read enable line and a read-write strobe line, 19 data lines in total.

The Raspberry PI treats the FPGA HAT as an EPROM memory chip. There is no clock source for synchronization but rather a strobe signal along with the WE/RE lines to signal a read or write as fast as the RPi can provide or read data. When the RPi is about to send data to the FPGA, the WE signals asserts. Data are launched at each falling edge of the strobe signal and considered valid at each rising edge. The FPGA samples data lines at each rising edge of the strobe signal as long as WE signal asserts. When the RPi receives data, the RE signal asserts. The data from the FPGA side are launched at each falling edge of the strobe signal and are considered valid at each rising edge. The timing diagram of the communication interface is shown in Fig. 2.

RPi determines how fast strobe asserts and de-asserts according to its processing speed. It is assumed that the FPGA, as a faster device than RPi, will always have data available to launch when strobe de-asserts or will be able to read data when the strobe line asserts. By implementing sophisticated software from the Raspberry Pi's side, the data exchange rate between the two processing elements can reach up to 50MBytes/sec.

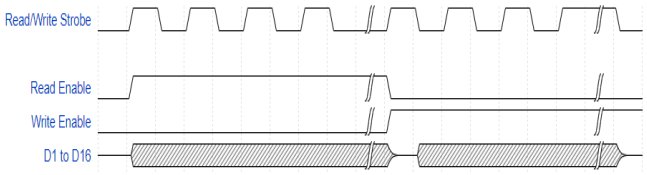

Fig. 2. Timing diagram.

D. *Software/Firmware implementation*

In order to verify the processing acceleration of the proposed co-design, various tests have been performed. Firstly, from the Raspberry Pi's side, a single threaded C++ program was created that pushed a predefined array of characters, containing a sine wave and a ramp, to the FPGA using the aforementioned parallel protocol, sending 2 bytes (16-bit) per single transfer. To verify that the data is received correctly, a configuration that included an instance of signalTap was developed on Cyclone 10LP. This one way of data transmission achieved around 50 Mbytes/sec.

When it was verified that Cyclone 10LP received the data correctly, a 4096 bytes FIFO buffer was added to our configuration to push the received data back to the Raspberry Pi. The FPGA accelerator HAT does not use any external memory. As a result, the need for intermediate storage or buffering is satisfied by the on-chip memory of the Cyclone 10LP. The actual data exchange at this point dropped down to 24 Mbytes per second due to the fact that delays are inserted when the program switches mode of the GPIO from read to write and vice-versa.

In the next step, a second identical FIFO buffer was added to our FPGA configuration. Between these two FIFO buffers we added a Prewitt edge detection filter [12], as it is shown in Fig. 3. Edge detection configuration on the FPGA HAT processes incoming data from the first FIFO buffer and stores them in the second buffer. FIFO buffers have different data lengths on their inputs and their outputs. Since the edge detection algorithm processes bytes instead of 16-bit short integer values, half of the data are processed during write cycles and the other half during read cycles.

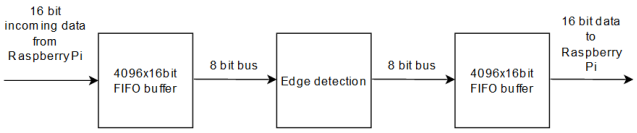

Fig. 3. Data Flow diagram on the FPGA configuration

On the Raspberry Pi side, a CSI V2 camera was used to capture a greyscale video feed at resolution of 512x512 pixels. Each frame was pushed to the FPGA HAT in order to apply the edge detection filter. The Raspberry Pi V2 camera can capture up to 90fps at this resolution.

Finally, the processed frame was sent back to the Raspberry Pi and the frame was viewed in a window. Image results are depicted in Fig. 4. and in Fig. 5. At this point, the actual data transfer rate has dropped significantly down to 8 Mbytes/sec or close to 30 frames per second on a Raspberry Pi 4 with 8Gbytes of RAM. This is because the Raspberry Pi uses an embedded Linux (Raspbian) which is a non Real-time operating system. It adds some delay to the transmission of the data through its GPIO, but most importantly because of how Raspberry Pi captures the video feed. The Raspberry Pi has to pause the execution of the main data transmission program, to capture a frame from the camera and store it on a buffer.

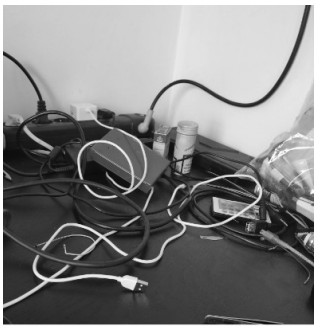

Fig. 4. Input frame.

Fig. 5. Frame after edge detection filter.

For each captured frame the delay is approximately 13.5ms per frame and up to 25ms as shown in Fig. 6. At 30 frames per second this translates to no data being transmitted by the FPGA HAT up to 75% of the time.

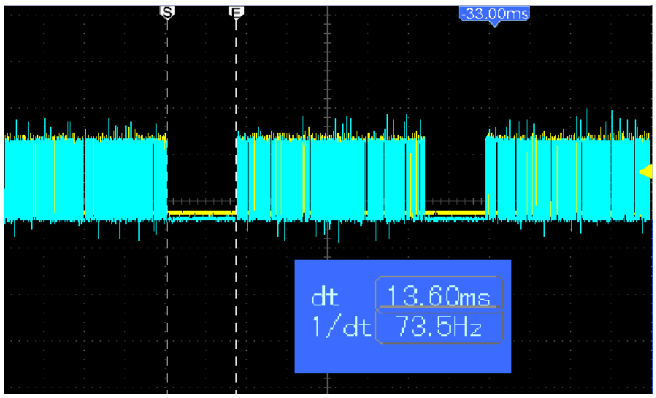

Fig. 6. Transmission gap between frames.

To overcome this delay, a multithreaded program was developed. In this program, one execution thread constantly captures frames and stores them into a buffer. Another thread transmits the frame data to the FPGA HAT and receives data back after the edge detection filter is applied. At the end, a third thread shows the processed frame on screen, as it is depicted in Fig. 7.

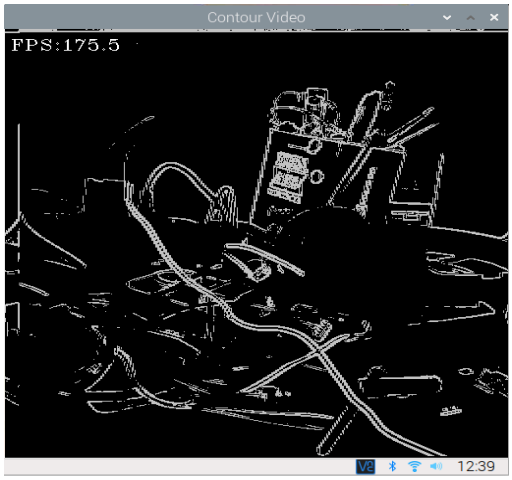

Fig. 7. Output view of the multithreaded process.

By using this technique, the actual frame rate has been increased to 170 fps which is a significant speed-up. This change is also visible on the oscilloscope view, where there is no delay between frames transmission, as it is shown in Fig. 8. As mentioned before, the Raspberry Pi camera can capture up to 90 frames per second, meaning excess frames are processed twice.

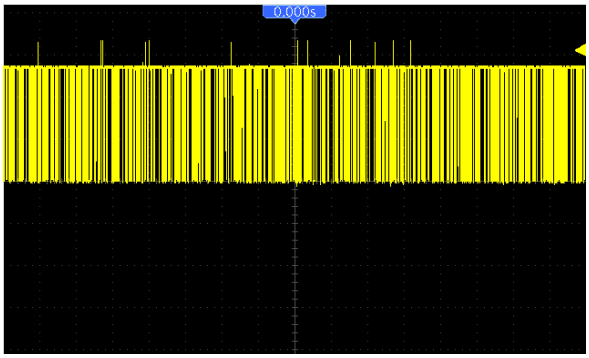

Fig. 8. Data exchange in the multithreaded program.

A similar C++ program that uses OpenCV to apply the same edge detection filter to the video feed can only achieve 48 frames per second, at the same 512x512 video feed resolution. This means that in the proposed architecture, the acceleration is about 350% for this task.

IV. System Evaluation

In this Section, the evaluation of the system in terms of power consumption and speed is presented. A test setup using a Raspberry Pi 4 with 8GBytes RAM and a multimeter connected in series was created. With the aid of the multimeter, we were able to measure min, max and mean current values and to estimate power consumption in various operation states. VNC was used in order to be connected remotely with the Raspberry Pi.

The first measurements concern the RPi's operation with and without the additional processing load which is added when the edge detection task runs. In Table I, as IDLE is represented the state in which the Raspberry Pi has just finished the boot process. The state in which we have opened a VNC connection with RPi is referred as VNC. Finally, the state when the C++ program executes edge detection using the OpenCV library is indicated as OPENCV.

TABLE I. Initial Measurements (FPGA HAT detached).

Current consumption (in A)	**Raspberry Pi without the FPGA HAT attached**		
	MIN	***AVERAGE***	***MAX***
IDLE	0.46	0.51	0.54
VNC	0.48	0.52	0.59
OPENCV	0.8	0.9	1.04

Subsequently, the FPGA HAT was attached and the same measurements were done again. Moreover, two more measurements took place. In these two measurements, FPGA HAT applied edge detection. In the first one, the single threaded program was executed in the Raspberry Pi. The multi threaded program ran in the second one. The corresponding measurements are presented in Table II.

TABLE II. Measurements with the FPGA HAT attached.

Current consumption (in A)	**Raspberry Pi with the FPGA HAT attached**		
	MIN	***AVERAGE***	***MAX***
IDLE	0.51	0.52	0.57
VNC	0.51	0.56	0.6
OPENCV	0.81	0.92	1.1
FPGA (single thread)	0.82	0.82	0.84
FPGA (multi thread)	0.81	1.12	1.2

Table III presents the performance of the system in each case. As it can be seen, when the multi threaded C++ program is executed in conjunction with the FPGA HAT operation, the system performance increases. The acceleration in comparison with single threaded program is more than 500%, while in comparison with the program that uses OpenCV library is more than 350%.

TABLE III. Performance measurements.

Frames per second for each process	**Raspberry Pi 4**
	FPS (Best)
OPENCV	48
FPGA (single thread)	32.4
FPGA (multi thread)	172

The evaluation continued in terms of CPU usage, CPU temperature and overall power consumption. The LXTask system monitor was used to register CPU usage. Furthermore, vcgencmd utility was installed to measure CPU temperature, while the power consumption was calculated using the attached multimeter. Experimental results are quoted in Table IV.

By adopting the multi threaded program, the overall consumption is about 20% more in comparison with the process that uses OpenCV library to apply edge detection. The FPGA HAT consumes the extra power. However, the RPi's CPU temperature appears lower (64ºC vs 71ºC) regardless of the fact that CPU usage is almost twice as high (90% vs 46%). This is attributed to that no intensive functions

are applied to the Raspberry Pi, since edge detection has been implemented in the FPGA HAT. This leads to a lower wear in the long run.

TABLE IV. EVALUATION IN TERMS OF CPU USAGE, TEMPERATURE AND POWER CONSUMPTION.

Power consumption in (A) Amp	**Raspberry Pi with the FPGA HAT attached**		
	CPU usage time (%)	***Temperature (ºC)***	***Power consumption (W)***
IDLE	3	55	2.6
VNC	8	55	2.8
OPENCV	46	71	4.6
FPGA (single thread)	33	62	4.1
FPGA (multi thread)	90	64	5.6

The single threaded process can be used when high speed frame rate is not required and even though it performs 35% slower than OpenCV, the system can run cooler, with lower power requirements and less CPU usage. It performs better in cases where the Raspberry Pi – FPGA combination is powered by batteries and it can offer lower power consumption which translates to longer battery runtime and fewer charging cycles for the same amount of work.

Lastly, the following table (Table V) highlights the cost increase of the setup when using an FPGA HAT along with the Raspberry Pi.

TABLE V. COST INCREASE IN REGARD TO THE CYCLONE FPGA IC USED.

Cyclone FPGA on HAT	***Total Cost***	***Total cost with a RPI4 / 4G***	***Cost increase (%)***
10cl006	€ 50.00	€ 100.00	200%
10cl010	€ 65.00	€ 115.00	230%
10cl016	€ 80.00	€ 130.00	260%
10cl025	€ 100.00	€ 150.00	300%

Therefore, the increase in cost (almost 300% when adopting the most expensive pin-to-pin FPGA of the same family) is lower than the performance gain (which is close to 350%) compared to the Raspberry Pi stand-alone operation.

V. CONCLUSION

In this paper, we accelerate image image processing algorithms using a custom built system that includes an FPGA-Raspberry Pi combination. A fast 16-bit parallel protocol was implemented and performance was optimized by using multithread techniques to outperform solutions that use software only implementations like OpenCV. The proposed system achieved 350% acceleration of an edge detector. The system can be used in robotic vision applications where the FPGA undertakes heavy processing load and the Raspberry Pi can run ROS and apply higher-level operations.

Further work includes the design of a TPU (tensor processing unit) configuration for the FPGA HAT that can benefit from the fast 16-bit parallel protocol and accelerate tensorflow calculations on the Raspberry Pi.

REFERENCES

[1] J. N. Lygouras, K. A. Lalakos, and P. G. Tsalides, 'THETIS: an underwater remotely operated vehicle for water pollution measurements', Microprocess. Microsyst., vol. 22, no. 5, pp. 227–237, Sep. 1998, doi: 10.1016/S0141-9331(98)00083-0.

[2] 'TurtleBot'. https://www.robotis.us/turtlebot/ (accessed Feb. 10, 2022).

[3] G. Karalekas, S. Vologiannidis, and J. Kalomiros, 'EUROPA: A Case Study for Teaching Sensors, Data Acquisition and Robotics via a ROS-Based Educational Robot', Sensors, vol. 20, no. 9, 2020, doi: 10.3390/s20092469.

[4] 'Astro Pi'. https://astro-pi.org/ (accessed Feb. 10, 2022).

[5] H. Gao et al., 'cuFSDAF: An Enhanced Flexible Spatiotemporal Data Fusion Algorithm Parallelized Using Graphics Processing Units', IEEE Trans. Geosci. Remote Sens., vol. 60, pp. 1–16, 2022, doi: 10.1109/TGRS.2021.3080384.

[6] J. A. Kalomiros and J. Lygouras, 'Design and evaluation of a hardware/software FPGA-based system for fast image processing', Microprocess. Microsyst., vol. 32, no. 2, pp. 95–106, Mar. 2008, doi: 10.1016/j.micpro.2007.09.001.

[7] T. K. Maiti, 'ROS on ARM Processor Embedded with FPGA for Improvement of Robotic Computing', in 2021 International Symposium on Devices, Circuits and Systems (ISDCS), Mar. 2021, pp. 1–4. doi: 10.1109/ISDCS52006.2021.9397897.

[8] Z. Dave, S. Dhote, P. Charjan, J. Joshi, and G. Gore, 'Article: Reconfigurable Image Processor using an FPGA-Raspberry pi Interface', IJCA Proc. Int. Conf. Comput. Technol., vol. ICCT 2015, no. 5, pp. 11–15, Sep. 2015.

[9] S. Kumar, M. Shah, and A. Singh, 'FPGA – Raspberry pi Interface for low cost IoT based image processing', Invertis J. Sci. Technol., vol. 10, p. 219, Jan. 2017, doi: 10.5958/2454-762X.2017.00034.8.

[10] T. Wang, 'Development of an FPGA and MCU based Stack-able Processing platform incorporated with on-board compute module for Real-time processing applications', Memorial University of Newfoundland, 2017.

[11] 'OpenOCD - Debian Wiki'. https://wiki.debian.org/OpenOCD (accessed Jan. 30, 2022).

[12] J. V. Vourvoulakis, J. Lygouras, and J. A. Kalomiros, 'Acceleration of Image Processing Algorithms Using Minimal Resources of Custom Reconfigurable Hardware', in 2012 16th Panhellenic Conference on Informatics, 2012, pp. 68–73. doi: 10.1109/PCi.2012.11.