PynqCopter - An Open-source FPGA Overlay for UAVs

Conference Paper · December 2018

DOI: 10.1109/BigData.2018.8622102

5 authors, including:

Brennan Cain
University of South Carolina
5 PUBLICATIONS **24** CITATIONS
SEE PROFILE

Ryan Kastner
University of California, San Diego
286 PUBLICATIONS **4,993** CITATIONS
SEE PROFILE

PynqCopter - An Open-source FPGA Overlay for UAVs

Brennan Cain, Zain Merchant, Indira Avendano, Dustin Richmond, Ryan Kastner

***Abstract*— FPGAs are a computing platform that excel in performing signal processing, control, networking, and security in a high performance and power efficient manner. This makes FPGAs attractive for unmanned aerial vehicles (UAVs) especially as they require smaller payloads and are processing multiple high data rate input sources (e.g. cameras, lidar, radar, gyroscopes, accelerometers). Unfortunately, FPGAs are notoriously difficult to program and they require significant hardware design expertise. However, there are newly released design tools aimed at making FPGAs easier to use, which drove the initial hypothesis for this paper: could three undergraduates program an FPGA to control a UAV in 10 weeks? The result of the experiment is PynqCopter – an open source control system implemented on an FPGA. We created and tested a UAV overlay which is able to run multiple computations in parallel, allowing for the ability to process high amounts of data at runtime.**

I. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) are powerful and efficient computer chips that customize important functions to make them faster than CPUs and GPUs [1]. A drawback to using FPGAs is their high barrier to entry. They require advanced hardware design skills like programming in Verilog, interfacing with low level input / output (I/O), and fail to provide a good high level programming environment[2].

Several recent efforts aim to simplify FPGA design and implementation. The first is the emergence of capable High Level Synthesis (HLS) tools [3], [4]. Xilinx Vivado HLS tools take a high level language such as C or C++ and generate Verilog or VHDL. These languages may be used with existing design suites, such as the Vivado Design Suite, to generate a bitstream for programming the Programmable Logic (PL) of the FPGA.

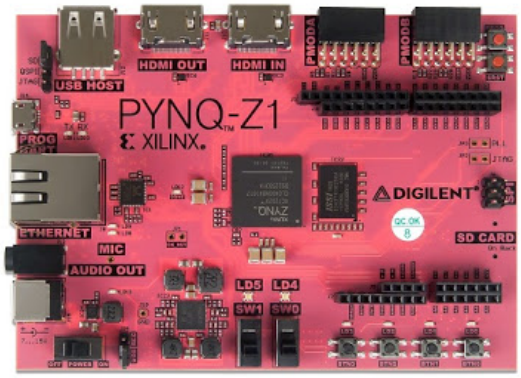

Fig. 1. PYNQ development board by Xilinx with ZYNQ-7020 SoC chipset including an ARMv7 dual-core processor and programmable logic fabric.

A new development in the usability of FPGAs is the Xilinx PYNQ board Figure 1. This board utilizes a Zynq Z7020 chip which contains a dual core ARM processor as well as a PL fabric. PYNQ adds to the default system by adding a set of Python libraries for interacting with the PL. The flow of this system is that a user designs the individual cores in C or C++, links them together in a block design, compiles it into a bitstream and .tcl file, and lastly, uses a Python function to flash these files (collectively called the Overlay) to the PL. When flashed, Python drivers are instantiated for each core to either the default or a custom driver created by the user. The PYNQ libraries can then be used to interact with those IP cores. [5]

In this paper, we demonstrate an open source overlay for the PYNQ-Z1 board. This overlay includes source files for each IP core, Python drivers to interact with and run the cores, and a block design to connect the IP cores. A video of the PynqCopter in flight can be found at https://youtu.be/pmFRnbAjpZQ

The main contributions of this paper are:

- A novel hardened open-source control system.
- Documentation on the process of developing HLS applications.

Brennan Cain is with the University of South Carolina, Columbia, SC, USA. bscain@email.sc.edu

Zain Merchant is with the University of Texas at Dallas, Richardson, TX, USA. ztm140130@utdallas.edu

Indira Avendano is with the University of Central Florida, Orlando, FL, USA. indirajhenny@knights.ucf.edu

Dustin Richmond is with the University of Washington, Seattle, WA, USA. dustinar@uw.edu

Ryan Kastner is with the University of California, San Diego, CA, USA. kastner@ucsd.edu

A. Paper Overview

In this paper, we present our FPGA-based hexacopter design process in a way that educators or students may be able to recreate it without expertise in the subject of FPGA design. Section 2 will describe related research in the field of using FPGAs for flight hardware, the adoption of FPGAs by the hobbyist and general purpose computing communities, as well as recent educational initiatives and projects to teach hardware design to more people. Section 3 will explain each IP core we designed, how they communicate between each other, and how a user may interact with the system. Section 4 will focus on the physical design of our system as well as why we chose to use certain components and how they interconnect. In Section 5, we explain how an educator may best use this system to teach computer architecture and controls using our free and open source system. Section 6 gives potential uses of our system as well as future works we would like to see done.

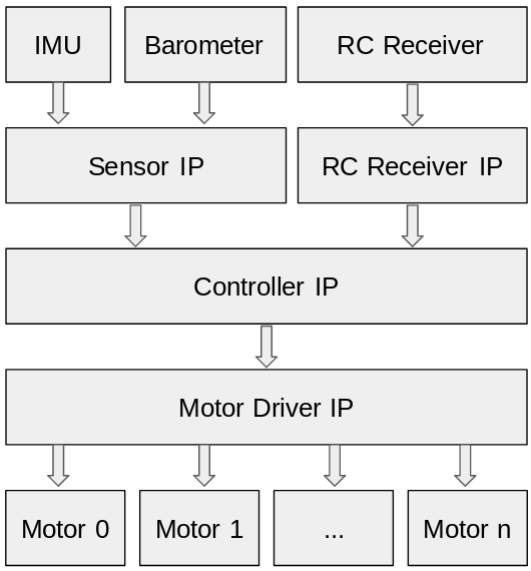

Fig. 2. General flow of data through the PynqCopter system. The top layer shows external devices which connect to the second layer. The data flows through to the fourth layer which generates signals to drive the external motors in the last layer.

II. Related Work

In-flight computation has come to the forefront of research in recent years[6]. Projects like the Small Adaptive Flight Control System at Georgia Tech [7], and the FPGA Based Stability System from The University of Queensland [8] seek to improve in flight performance and controls using re-programmable hardware. Schlender et al. at the Carl von Ossietsky University of Oldenburg [9] taught a course on developing hardware for UAVs using a ZYNQ 7020 SoC. In the system used by his course, the students developed C programs to be compiled to run on soft processing cores. Soft processing cores are essentially microprocessors, such as those found in an Arduino, that are implemented within the PL fabric. This method is good for teaching students about utilizing C for pipelineing in a microprocessor, but does not allow the student to think about how their logic functions at the lower, hardware level.

Hobbyists have attempted to lower the barrier of entry to FPGA development by developing tutorials for new users to create their own cores. A list of tutorials is found here: http://www.fpgadeveloper.com/. Although the guides and examples are good for designing the simple flow of instructions and data from CPU→FPGA→CPU, they do not explain how to allow the FPGA to run self-sufficiently. We used these guides to learn about individual cores, but these tutorials did not provide sufficient information on allowing cores to intercommunicate and talk to external devices. One book that we used in the development of our system provides users the ability to learn more deeply about the use and programming of FPGAs by providing implementations of complex algorithms [4]. Other books and articles are good when attempting to create lower latency algorithms [10] or learn about how the IPs interconnect at a higher level [11].

Other projects like the Open Vision Computer[12] and XFOpenCV[13] seek to do more with the computational resources on the FPGA by implementing commonly used algorithms in the PL. We use them to justify the need of better computing hardware for both flight control and a better performing system for in-flight computations.

III. Architecture

This project differentiates itself from other FPGA-based flight controllers by implementing a hardened controller, making way for possible latency improvements and overall performance. In this context, hardened refers to the fact that no soft-core CPU (ex. MicroBlaze, Nios) was used. Beyond the hardened control, the system also never passes data through the CPU. This leads to a system fully on the PL fabric of the FPGA with no need to interact with the CPU other than for configuration.

A. Intellectual Processor Cores and General Data Flow

Our general data flow can be visualized from Figure 2. We take a modular approach in the design of the platform, not only to simplify development, but also to clearly define scopes of the different tasks. Overall, we process raw data values from our sensors and remote controller, use them in calculations that can stabilize and control the UAV, and push an output signal to our electronic speed controllers (ESC) in a format they can recognize to control the 6 brushless DC motors.

The IP cores, as well as their interfaces, used in our block design are explained below:

1) Sensor IP Cores: These are separate IP cores, but are utilized at the same stage of the data path. The IMU sensor IP core is used to configure and collect data from our IMU. The core utilizes the AXI IIC Bus Interface to communicate with the sensors over I2C, configures them to receive the correct data in a format we can use, and reads values from them after proper configuration. The data is then converted into a 16 bit fixed-width integer value representing the current yaw, pitch, roll, and altitude, which can then be sent to other IP cores for normalization and processing. The data can also be configured to be used in attitude or rate modes (providing angular velocity rather than position).

2) RC Receiver IP Core: The receiver IP core samples the Pulse Width Modulation (PWM) signal received from our RC receiver to determine the duty cycle it is receiving, and converts this duty cycle value to a 32 bit unsigned integer. This 32 bit unsigned integer can then be used by other IP cores as the input target value we want for our hexacopter.

3) Normalizer IP Core: To properly use data we are receiving from these multiple sources, we need them to be in a common format and range. Data from both our sensor and receiver IP cores are pipelined into the Normalizer IP, which can normalize everything into the same units and within expected output ranges. This is an important intermediary stage for proper data formatting and processing. Afterwards, the data is now ready to be used for calculations to enable stable flight. Example outputs from our Normalizer IP core is shown with the inputs in Figure 3.

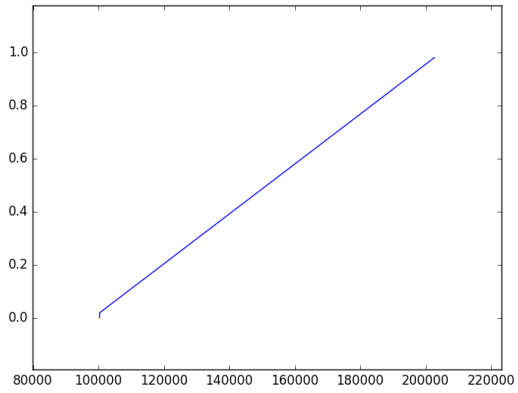

Fig. 3. Plot of the inputs on the x axis from a minimum value of 100000 ticks to a maximum 200000 and the outputs from zero to one on the y axis.

4) Controller IP Core: This core is designed to serve as a Proportional-Integral-Derivative (PID) controller, which is a type of general purpose closed-loop control algorithm. Here we take our target values obtained from the RC receiver, our measured values from the sensors, and our PID constants (configurable via Jupyter Notebooks during tuning stage) to calculate a estimated roll and pitch values that we can send to our motors to execute. Using this IP core, we try to stabilize our roll and pitch during flight; the PID helps take into account proportional differences, accumulated errors, and rate of change to make sure we dont overshoot our target values. For yaw and thrust, the controller is fully proportional which is taken into account during the mixing. The mixing is done in this core and our output data is prepared into six fixed point duty cycle outputs. The output of this mixer is shown in Figure 4.

5) PWM IP Core: This is the final IP core of our system. Here is where all of our computed values can be turned into PWM signals which we feed to the 6 ESCs in our hexacopter. We take in data from the motor mixer IP, use each of the computed values we have assigned to the individual motors and turn them into resulting PWM signals, and output them to pins that will be fed to the ESCs, thus controlling each of the motors in our UAV. Example outputs from this

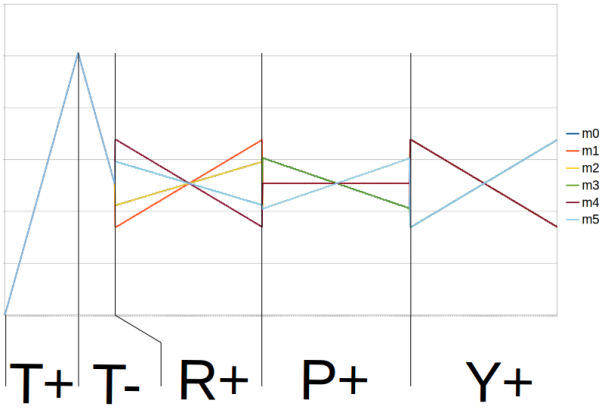

Fig. 4. Output of the 6-motor mixer given changing RPTY inputs. The changes being applied are on the x axis with the signals given to individual motors on the y axis. T, R, P, and Y refer to thrust, roll, pitch, and yaw respectively with the + or - being whether they are increasing or decreasing in the area.

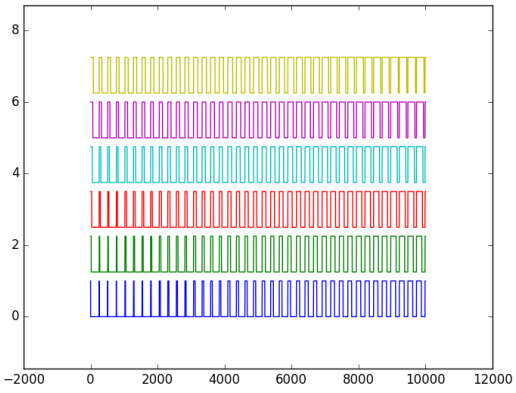

Fig. 5. Output of the 6-motor PWM generator with increasing duty cycles and constant frequencies. The PWM cycles were constantly increased over the duration of the test. The duty cycle at the start was highest at the top and lowest at the bottom.

core over time are shown in Figure 5.

Through these steps we are able to utilize sensors and designate input to properly control and fly our PYNQcopter. All of the IP cores mentioned above are written in HLS to be fully hardened on the FPGA. The only communication with the CPU is to set constants for the controller and to start/stop IP cores.

B. Inter-IP Communication

The primary communication methods and protocols in this project are the Advanced Extensible Interface (AXI), Inter-Integrated Circuit (I^2C, I2C, or IIC), Pulse-Width Modulation (PWM), and direct.

1) AXI: Most communication within the programmable logic fabric of the FPGA uses the AXI interface. This interface allows two IP cores to agree on when data is ready to be used and keeps IP cores safe from race conditions. From the perspective of the individual IP cores, we used 3 interfaces within this family: M_AXI, S_AXILITE, and AXI_FIFO. The M_AXI interface allows an IP core to request or to send data to another IP core which it may subscribe to using a slave interface. The M in M_AXI stands for master and this interface represents a full AXI master. S_AXILITE subscribes to a master and allows the master to read or write a single word of information at a time. AXI_FIFO acts as a master AXI interface which can be written to multiple times within the same clock cycle. This is a First-In-First-Out shift register which allows for some data to accumulate and be used over several clock ticks.

2) I^2C: The I^2C interface is used for communication with the sensors. Vivado Design Suite supplies an AXI IIC IP core that can be used for data communication via an AXI_FIFO. This FIFO passes data into the AXI IIC core which communicates with the external GPIO pins and returns values given by the sensors. Using I^2C, we are able to communicate with both of our sensors using the same two data/clock lines. The abstraction given by the AXI IIC core allows us to avoid re-implementing the protocol ourselves [14].

3) Pulse-Width Modulation: Pulse-Width Modulation (PWM) is used for communication from the on-board radio antenna, as well as to the ESCs. The core theory behind this communication technique is that by keeping a consistent signal frequency, or placing every leading edge at the same interval, and changing the time duration of the high signal, values can be transmitted robustly without the need for verification. We received PWM signals in 6 channels from the FrSky X8R receiver and generated 6 channels of PWM signals, which were sent to the ESCs.

4) Direct: The direct method of communication is with no true protocol in place. Communication over PWM is not a core protocol implemented in Vivado. The protocol-less way to transfer data is using the Arbitrary Precision None Interface (AP_NONE). PWM generation and reading was done by setting AP_NONE bits high or low, which is considered unsafe in other applications. In our application this is fine. However, transferring data using this approach is normally dangerous.

Fig. 6. Fully-assembled PynqCopter UAV including batteries, sensors, controller, propeller guards, and propulsion system.

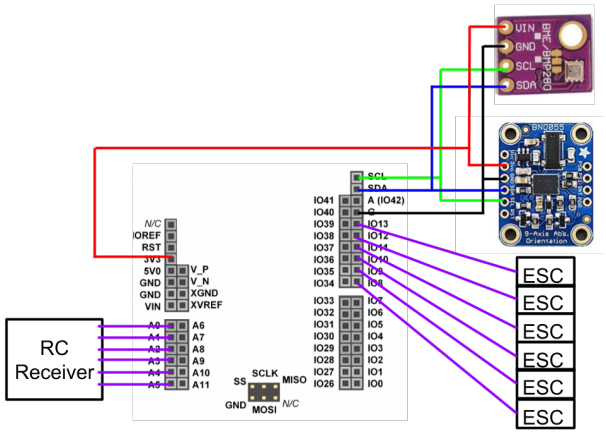

Fig. 7. Hookup diagram for the PynqCopter's sensors, RC receiver, and ESCs. Red denotes 3.3V power, black denotes ground, blue denotes I^2C data, green denotes I^2C clock, and purple denotes PWM.

IV. Physical System

UAVs are generally comprised of several basic parts: a radio receiver, controller, sensors, propulsion systems, and a chassis. This section will go over each of these parts, how they connect together logically and physically, and why we used each part for our system. Figure 6 is a photo of our completed UAV.

A. Radio Receiver

We chose to use an FrSky Taranis 16-channel RC radio with an X8R 8-channel receiver. The reason for this choice was to allow us to have a minimum of six channels of communication so that we could switch operating modes using the switches provided on the controller. Another key feature of this device is that the 8 channels are broken out into 8 different lines. We physically connected the first six channels to Arduino analog pins 0 through 5 on the PYNQ board. Within the programmable logic, we passed the PWM signal through a 100 register synchronizer to stabilize the rising and falling edges, and prevent errors that occur in the transition from high to low and low to high states. The signals are then passed into the RC_Receiver IP.

B. Controller

The core of this project's hexacopter system design focuses on the controller. We utilized a Xilinx PYNQ-Z1 board which consisted of a Xilinx Z7020 SoC. The ZYNQ-7000 line of SoCs consist of a dual-core ARMv7 microprocessor as well as a programmable logic fabric. Using the PYNQ libraries, we were able to use Python to interact with the registers of the IP cores implemented within the PL fabric. This made it easy for us to tune the various parts of the system in a scripting language. Figure 7 shows a hookup diagram for the Sensors, RC Receiver, and ESCs to the PYNQ board.

C. Sensors

To obtain the hexacopters state, we chose the Adafruit BNO055 IMU sensor and BME280 Barometer sensor. These sensors receive information regarding the hexacopters attitude in space, angular velocity, magnetic heading, and altitude. The current physical hexacopters design has been implemented and tested with only the IMU sensor, since the latest system design bypasses altitude and uses the raw thrust given by the pilot. The sensors are connected to the PYNQ-Z1 through the SCL and SDA pins in the Arduino headers on the PYNQ board and to the common 3.3V and GND pins. Inside the PL, they are connected to the AXI_IIC IP via tri-state pins.

D. Chassis, Battery, and Propulsion System

We chose to use an off-the-shelf propulsion system by DJI, the Flamewheel F550 hex-rotor with 15A Opto ESCs. This frame and propulsion system was chosen for two main reasons: moderate lift and a large user base from ArduPilot and other projects. An application which we will discuss in the future work section requires several cameras or a lidar to be mounted to the chassis. This propulsion system will allow the mounting of up to 2kg of additional cameras and possibly a lidar. Since the ESCs on this frame are popular within the hobbyist space, their widespread use has created

detailed documentation on them, which allowed us to easily implement and interface. The PYNQ board was mounted using zip ties and double sided tape to the chassis on the top center to allow the IMU to be near the center of rotation along each axis. The ESCs were connected to Arduino pins 8 through 13. The grounds were common and connected to the ground pin. The ESCs' power cables were soldered per the instructions in the F550 kit to the chassis' PCB. A battery cable with a male connector was then soldered to the PCB, allowing a battery to be plugged and unplugged. We placed a 5v converter in line of the battery to power our PYNQ board and RC Receiver.

E. Additional Considerations

In the development of this system, we found several improvements to the base system which were helpful regarding both the safety of the system as well as its stability in flight. On the safety side, we added propeller guards and legs. These prevented the UAV from hitting objects with the blades and lowered the possibility of damage to person or property. The legs allowed the UAV to land with some momentum without damaging the batteries and mounted equipment. The batteries were mounted on the bottom of the UAV to create a pendulum effect. This placed the system into a stable equilibrium in the roll and pitch axes. A drawback to this design, however, is the pendulum-like oscillatory motion which occurred with an improperly tuned controller. A way to fix this would be to use thinner batteries and mount them between the two parallel plates in the center of the UAV.

V. Educational Use

FPGAs are an important tool that can be used by nearly anyone in the electronics and computer science industry, and open up more possibilities to what can be done with computing technology. Xilinx, Intel, and other FPGA manufacturers hope that with the incorporation of new tools seeking to simplify hardware development, such as HLS, there can be a higher rate of adoption for FPGAs. We as undergraduate computer science and engineering students with little background in hardware development, found tools like HLS and the Vivado Design Suite relatively straight forward when compared to Verilog or VHDL.

Traditionally computation power has been a significant bottleneck in real time computing applications such as machine learning, computer vision, digital signal processing, and flight control to name a few. By creating a useful, well documented, open-source project, we can help enlarge the community of those working with hardware by bringing in more people from different fields that can benefit from application specific hardware design. We not only sought to teach ourselves how to use FPGAs and HLS, but also demonstrate some of the possibilities and applications that others can apply to their own project. We have done this by making education a primary motivating factor behind our project this summer.

We have documented our entire learning process using the Xilinx Vivado Design Suite to develop the necessary IP cores for a hardened hexacopter controller system. We detail every step that went into the project's development so that even novice users can learn from and utilize our designs in their own hardware design projects.

Our source code along with sample overlays can be found in the UCSD PynqCopter repository on github here: https://github.com/UCSD-E4E/pynq-copter. The repository has two primary sub-directories:

- The '/notebooks' folder directory contains the Jupyter Notebooks files with the instructions and source code to activate and interact with the IP cores on the Pynq-Z1 board.
- The '/pynqcopter' folder directory contains the bitstreams of completed block designs and their accompanying .tcl files, which can be flashed onto the PYNQ-Z1 board as overlays via the PYNQ libraries. It also contains an IP folder directory containing synthesized HLS IP Cores for easy access and integration into a Vivado block design.

Further documentation for educators, hobbyists, and those wishing to learn how to develop our system in greater detail can be found on our team's Google Drive here: http://goo.gl/GLRXiC.

VI. Future Work

Primarily, our goal is for this project to be used in classes and by hobbyists, allowing them to quickly learn how to effectively design hardware and controls. On top of this, we hope that others will contribute to our system and documentation by adding additional multicopter configurations and sensor drivers.

Other projects, such as the E4E's efforts in mangrove classification, can benefit from this system as well. Mangrove classification currently relies on UAVs being flown over Mexico's mangrove forests to collect aerial imagery data[15]. This data is later used in the lab for species classification and biomass density estimation,

but sometimes experiences problems due to incomplete or incorrect data being collected. Rather than plan a return trip, this classification could be done while the UAV is in flight, adjusting itself and repeating sections with bad data. This project utilizes a convolutional neural network (CNN) and can take advantage of improvements in visual processing on FPGAs[16] as well as in placing CNNs on FPGAs[17], [18].

UAVs are also now widely used for LIDAR measurements, such as to map through the South American canopy when searching for Maya temples in archaeological expeditions[19]. These applications depend on precise timing of measurements to generate 3 dimensional LIDAR maps, but has currently been limited by the operational capabilities of the on board microcontroller and CPU. Designing a PL fabric specific to this application might be a way to overcome this issue and help with the project.

With research going into the use of FPGAs for digital signal processing [20], [21], we also hope to use this system to extend and improve a radio collar tracking project[22] by taking advantage of hardware acceleration in signal processing.

VII. CONCLUSIONS

Our goals for this paper were to show that undergraduates are able to use current tools and technologies to develop hardened programs for control systems. Through our time in the University of California, San Diego Engineers for Exploration Research Experience for Undergraduates program, and with the guidance received from our supervisors, we have been able to design a working, hardened FPGA hexacopter.

After being exposed to new and forthcoming developments in making hardware design an easier experience, we have come to the conclusion that it can be done with relatively little prerequisite knowledge and see future adoption throughout industry and academia. The tools have been abstracted to a level at which someone with hobby electronics and programming experience could pick up and develop a working product suited to their needs. In addition to creating a hexacopter that can be used for future projects, we've created a pool of resources that can be used as education materials for those seeking to work with FPGAs. We believe these resources can help users ease into working at a lower technical level than they were previously exposed to.

ACKNOWLEDGMENT

The authors would like to thank the National Science Foundation for its support through the Research Experience for Undergraduates. The authors would also like to thank the Engineers for Exploration group in the Jacob's School of Engineering at UCSD for hosting the REU program, as well as for their guidance, and support.

REFERENCES

[1] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145704

[2] D. Bacon, R. Rabbah, and S. Shukla, "Fpga programming for the masses," *Queue*, vol. 11, no. 2, pp. 40:40–40:52, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2436696.2443836

[3] J. Matai, D. Richmond, D. Lee, and R. Kastner, "Enabling fpgas for the masses," *CoRR*, vol. abs/1408.5870, 2014. [Online]. Available: http://arxiv.org/abs/1408.5870

[4] R. Kastner, J. Matai, and S. Neuendorffer, "Parallel Programming for FPGAs," *ArXiv e-prints*, May 2018. [Online]. Available: https://arxiv.org/abs/1805.03648

[5] Xilinx, "Pynq: Python productivity for zynq," http://www.pynq.io/, accessed: 2018-10-07.

[6] M. Bouhali, F. Shamani, Z. E. Dahmane, A. Belaidi, and J. Nurmi, "Fpga applications in unmanned aerial vehicles - a review," in *Applied Reconfigurable Computing*, S. Wong, A. C. Beck, K. Bertels, and L. Carro, Eds. Cham: Springer International Publishing, 2017, pp. 217–228. [Online]. Available: https://doi.org/10.1007/978-3-319-56258-2_19

[7] H. Christopherson, W. Pickell, A. Koller, S. Kannan, and E. Johnson, ser. Infotech@Aerospace Conferences. American Institute of Aeronautics and Astronautics, Sep 2004, ch. Small Adaptive Flight Control Systems for UAVs Using FPGA/DSP Technology, 0. [Online]. Available: https://doi.org/10.2514/6.2004-6556

[8] B. Eizad, A. Doshi, and A. Postula, "Fpga based stability system for a small-scale quadrotor unmanned aerial vehicle," in *Proceedings of the 8th FPGAWorld Conference*, ser. FPGAWorld '11. New York, NY, USA: ACM, 2011, pp. 3:1–3:6. [Online]. Available: http://doi.acm.org/10.1145/2157871.2157874

[9] H. Schlender, S. Schreiner, M. Metzdorf, K. Grüttner, and W. Nebel, "Teaching mixed-criticality: Multi-rotor flight control and payload processing on a single chip," in *Proceedings of the WESE'15: Workshop on Embedded and Cyber-Physical Systems Education*, ser. WESE'15. New York, NY, USA: ACM, 2015, pp. 9:1–9:8. [Online]. Available: http://doi.acm.org/10.1145/2832920.2832929

[10] G. Wang, W. Gong, and R. Kastner, *Operation Scheduling: Algorithms and Applications*. Dordrecht: Springer Netherlands, 2008, pp. 231–255. [Online]. Available: https://doi.org/10.1007/978-1-4020-8588-8_13

[11] D. Richmond, J. Blackstone, M. Hogains, K. Thai, and R. Kastner, "Tinker: Generating custom memory architectures for altera's opencl compiler," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 21–24. [Online]. Available: https://doi.org/10.1109/FCCM.2016.13

[12] M. Quigley, K. Mohta, S. S. Shivakumar, M. Watterson, Y. Mulgaonkar, M. Arguedas, K. Sun, S. Liu, B. Pfrommer, V. Kumar, and C. J. Taylor, "The open vision computer: An integrated sensing and compute system for mobile robots," 2018. [Online]. Available: https://arxiv.org/abs/1809.07674

[13] S. Neuendorffer, T. Li, and D. Wang, *Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries*. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp1167.pdf

[14] *AXI IIC Bus Interface*, Xilinx, October 2016, v2.0. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_iic/v2_0/pg090-axi-iic.pdf

[15] P. Ezcurra, E. Ezcurra, P. P. Garcillán, M. T. Costa, and O. Aburto-Oropeza, "Coastal landforms and accumulation of mangrove peat increase carbon sequestration and storage," *Proceedings of the National Academy of Sciences*, vol. 113, no. 16, pp. 4404–4409, 2016. [Online]. Available: http://www.pnas.org/content/113/16/4404

[16] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of fpga, gpu and cpu in image processing," in *2009 International Conference on Field Programmable Logic and Applications*, Aug 2009, pp. 126–131. [Online]. Available: https://doi.org/10.1109/FPL.2009.5272532

[17] E. Wang, J. J. Davis, and P. Y. K. Cheung, "A PYNQ-based Framework for Rapid CNN Prototyping," in *IEEE Symposium on Field-programmable Custom Computing Machines (FCCM)*, 2018. [Online]. Available: https://spiral.imperial.ac.uk/handle/10044/1/57937

[18] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 65–74. [Online]. Available: https://arxiv.org/abs/1612.07119

[19] M. A. Canuto, F. Estrada-Belli, T. G. Garrison, S. D. Houston, M. J. Acuña, M. Kováč, D. Marken, P. Nondédéo, L. Auld-Thomas, C. Castanet, D. Chatelain, C. R. Chiriboga, T. Drápela, T. Lieskovský, A. Tokovinine, A. Velasquez, J. C. Fernández-Díaz, and R. Shrestha, "Ancient lowland maya complexity as revealed by airborne laser scanning of northern guatemala," *Science*, vol. 361, no. 6409, 2018. [Online]. Available: http://science.sciencemag.org/content/361/6409/eaau0137

[20] S. Mirzaei, "Design methodologies and architectures for digital signal processing on fpgas," 2010. [Online]. Available: http://cseweb.ucsd.edu/~kastner/papers/phd-thesis-mirzaei.pdf

[21] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 28, no. 1, pp. 7–27, May 2001. [Online]. Available: https://doi.org/10.1023/A:1008155020711

[22] G. A. M. d. Santos, Z. Barnes, E. Lo, B. Ritoper, L. Nishizaki, X. Tejeda, A. Ke, H. Lin, C. Schurgers, A. Lin, and R. Kastner, "Small unmanned aerial vehicle system for wildlife radio collar tracking," in *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, Oct 2014, pp. 761–766. [Online]. Available: https://doi.org/10.1109/MASS.2014.48